Diagnóstico Diferencial de Fiebres Hemorrágicas Utilizando ARTMAP.

William Caicedo Torres
Ingeniero de Sistemas, M.Sc.(c)
Código: T00017565

Universidad Tecnológica de Bolívar
Maestría en Ingeniería
Cartagena
Febrero de 2012

Diagnóstico Diferencial de Fiebres Hemorrágicas Utilizando ARTMAP.

William Caicedo Torres
Ingeniero de Sistemas, M.Sc.(c)
Código: T00017565

Trabajo de tesis para optar al título de
Magister en Ingeniería con énfasis en Ingeniería de Sistemas

Director
Moisés Quintana Álvarez, M.Sc.
Magíster en Informática Aplicada

Universidad Tecnológica de Bolívar
Maestría en Ingeniería
Cartagena
Febrero de 2012

Título en español

Diagnóstico Diferencial de Fiebres Hemorrágicas Utilizando ARTMAP.

Title in English

Differential Diagnosis of Hemorrhagic Fevers Using ARTMAP.

Resumen: El diagnóstico diferencial de las fiebres hemorrágicas endémicas en Colombia no es una tarea fácil para el personal médico. Varias de estas enfermedades se confunden frecuentemente con otras en términos de signos clínicos y síntomas, haciendolo un proceso difícil y propenso al error. Los algoritmos de aprendizaje automatizado (Machine Learning) ofrecen algunas prestaciones que pueden ser útiles a la hora de enfrentar este tipo de problemas de reconocimiento de patrones. En esta tesis se describe una aproximación al diagnóstico diferencial del Dengue, Leptospirosis y Malaria, basada en el uso de la Teoría de la Resonancia Adaptativa (ART - ARTMAP). El objetivo de este trabajo es el desarrollo de un clasificador neuronal ARTMAP capaz de diferenciar entre estas enfermedades, y su entrenamiento por medio de un conjunto de datos compuesto por información extraida de historias médicas de pacientes admitidos en los últimos 10 años en el Hospital Infantil Napoleón Franco Pareja. Los resultados obtenidos sobre el conjunto de datos de prueba son prometedores, y muestran la viabilidad del enfoque propuesto

Abstract: Differential diagnosis of hemorraghic fevers endemic in Colombia is not an easy task for medical practitioners. Several diseases often overlap with others in terms of clinical signs and symptoms, making it a difficult process and prone to error. Machine Learning algorithms offer some qualities that can be useful to tackle this kind of pattern recognition problems. In this thesis a neural network based approach to the differential diagnosis of Dengue Fever, Leptospirosis and Malaria, using the Adaptive Resonance Theory Map (ARTMAP) family is discussed. The final goal of this work is to develop a suitable ARTMAP neural classifier capable of telling apart these diseases from each other, and to train it by means of a dataset comprised of medical charts from pacients admitted in the last 10 years to Napoleón Franco Pareja Children's Hospital. Test set results obtained are promising, and support the viability of this approach

Palabras clave: Redes Neuronales, ARTMAP, fiebre hemorrágica, dengue, leptospirosis, malaria, diagnóstico diferencial

Keywords: Machine Learning, Neural Networks, ARTMAP, Hemorrhagic Fever, dengue, leptospirosis, malaria, differential diagnosis

Nota de aceptación

Trabajo de tesis

"Mención "

Jurado

Jurado

Director
MSc Moisés Quintana

Cartagena, Febrero de 2012

Dedicado a

"I have here completed the work of my calling, with as much intellectual strength as you have granted me."

Johannes Kepler

"Y si tuviese profecía, y entendiese todos los misterios y toda ciencia, y si tuviese toda la fe, de tal manera que trasladase los montes, y no tengo amor, nada soy"

1 Corintios, cap 13 verso 2

Al Padre de las Luces, a Fausto, a Ob-Bel, a Angius, a Doris, a Cecilia, a Katy, a Gabriel y Juan Felipe.

Agradecimientos

Agradezco a Dios por la oportunidad de realizar este trabajo como culminación de un proceso de varios años. Su misericordia me trajo hasta acá, sustentándome en cada momento, muy a pesar de mis humanas debilidades. Toda la gloria sea para El. Infinitas gracias a sus mensajeros, los cuales me enseñaron a creer que El me ama y que junto a El todo es posible.

Agradezco a mi familia, por todo su apoyo. A mi mamás Doris y Cecilia, por su continuo sacrificio y amor, sin el cual no hubiese sido posible llegar hasta acá. A mi esposa por creer en mi todo el tiempo. A mis hijos por ser el aliciente para superarme. A mi padre, por darme la oportunidad de retomar el camino.

Agradezco al Hospital Infantil Napoleón Franco Pareja - casa del Niño, por haber prestado toda la colaboración necesaria para adelantar esta investigación. Agradezco con especial afecto al Doctor Hernando Pinzón, el cual creyó en mi y propuso la idea que a la postre se convirtió en este trabajo, por todo el tiempo que dedicó a la caracterización de las enfermedades involucradas en este trabajo. Muchas gracias a la doctora Ceyla Causil, por todas esas horas dedicadas al arduo trabajo de la construcción del conjunto de datos para el entrenamiento y prueba de los clasificadores.

Agradezco también a mi director de tesis, el profesor Moisés Quintana, por toda la colaboración, y en general a todos los que de una u otra forma colaboraron para que este proyecto saliera adelante, a todos muchísimas gracias.

Índice general

Índice de tablas

Índice de figuras

Introducción

El dengue, leptospirosis y la malaria son enfermedades que pertenecen a un grupo conocido como fiebres hemorrágicas [1] [3]. Estas enfermedades se pueden manifestar inicialmente como síndromes febriles indeterminados, por lo que pueden ser fácilmente confundidas con otros cuadros o incluso entre ellas [2]. A pesar de presentar síntomas parecidos, sus etiologías y tratamiento son muy diferentes, por lo que se hace de especial importancia establecer mecanismos de diagnóstico claramente definidos y con alta especificidad, teniendo en cuenta que la mortalidad asociada a estas enfermedades se reduce dramáticamente al ser diagnosticadas oportunamente. Para el diagnóstico de estas enfermedades, el personal médico dispone de indicaciones semiológicas en la literatura, pero estas enfermedades presentan una alta variabilidad en términos de los síntomas y signos que presentan los pacientes afectados, haciendo difícil establecer con claridad de que enfermedad se trata. Además, se disponen de pruebas de laboratorio para la confirmación del diagnóstico, con confiabilidad altamente variable y resultados que pueden tomar varios días para ser reportados finalmente. Todo lo anterior juega en contra de un diagnóstico acertado y oportuno, llevando muchas veces a falsos diagnósticos. Varios estudios [6] [7] [8] [9] ponen de manifiesto situaciones donde durante un brote epidémico, una enfermedad enmascara una porción significante de casos de otra enfermedad. El dengue, leptospirosis y la malaria son enfermedades de notificación obligatoria a las entidades encargadas de la vigilancia epidemiológica, puesto se consideran riesgos para la salud pública, lo cual las pone en un lugar de importancia manifiesta a la hora de considerar el diseño de estrategias encaminadas a obtener un diagnóstico temprano más acertado. El campo del aprendizaje automatizado (Machine Learning) ha ofrecido históricamente recursos bastante importantes para el diagnóstico médico ayudado por computadora, y a través de estos el problema de diagnóstico se asume como un problema de reconocimiento de patrones. La idea tras este tipo de aplicaciones no es remplazar al personal médico, sino darle herramientas adicionales para la toma de decisiones, y de esta manera contribuir a la mejora de la atención de los pacientes.

Entre los aportes del Machine Learning podemos encontrar las redes neuronales. Una característica importante de las redes neuronales es la capacidad de aproximar cualquier función con precisión arbitraria [10], por lo que se pueden usar para discriminar patrones en un conjunto de datos. Las aplicaciones de las redes neuronales al diagnóstico médico datan de tiempo atrás, con resultados positivos reportados en la literatura [11] [12] [13]. En la investigación que enmarca la presente tesis se propone el uso de una red neuronal basada en la Teoría de la Resonancia Adaptativa (ARTMAP) como clasificador para el diagnóstico diferencial del dengue, leptospirosis y malaria. La organización del

documento es la siguiente: El primer capítulo muestra el estado del arte del uso de redes neuronales en el diagnóstico de fiebres hemorrágicas. El segundo capítulo trata acerca de las generalidades de la Teoría de la Resonancia Adaptativa (ART), los modelos neuronales inspirados en ella y su funcionamiento. Además se exponen las características principales de la implementación de la familia de clasificadores ARTMAP usados en el trabajo.El tercer capítulo trata acerca del Dengue, Leptospirosis y Malaria; sus características etiológicas, semiológicas y epidemiológicas, y se discute lo relacionado a los síntomas y signos necesarios para el diagnóstico diferencial entre estas 3 enfermedades. Por su parte el cuarto capítulo discute la creación de un conjunto de datos (dataset) de pacientes con estas 3 enfermedades extraído de los registros del Hospital Infantil Napoleón Franco Pareja - Casa del Niño, y describe el comportamiento de las variables que lo componen. El quinto capítulo hace referencia a las experiencias realizadas con los clasificadores ARTMAP y el conjunto de datos construido. Se relacionan los resultados de validación y prueba con la respectiva discusión de los hallazgos. Por último se expresan las conclusiones de la investigación y los posibles trabajos futuros.

Objetivos

- **Objetivo General**
  - Desarrollar un clasificador ARTMAP capaz de identificar los patrones asociados a la fiebre dengue, leptospirosis y paludismo, con un grado de acierto suficiente para ser utilizado como ayuda tecnológica en el diagnóstico diferencial de estas enfermedades.
- **Objetivos Específicos**
  - Construir una familia de clasificadores ARTMAP (Fuzzy, Instance-Counting, Default) capaz de ser entrenada para realizar el diagnóstico diferencial de dengue, leptospirosis y malaria.
  - Realizar una caracterización semiológica y epidemiológica de las enfermedades incluidas (dengue, leptospirosis y malaria), con miras a la selección de las entradas del clasificador ARTMAP.
  - Recopilar la información clínica necesaria para la construcción de un conjunto de datos adecuado para el entrenamiento de los clasificadores.
  - Identificar un conjunto de predictores de la fiebre dengue, leptospirosis y paludismo con alta especificidad resaltando los factores diferenciadores, para ser usados como entrada del clasificador neuronal ARTMAP.
  - Entrenar, validar y probar el clasificador neuronal y registrar los resultados, con el fin de encontrar el tipo de clasificador ARTMAP más adecuado y establecer su configuración óptima, y así maximizar su precisión predictiva.

Aspectos Metodológicos

Inicialmente, se procederá a construir los clasificadores ARTMAP, y un optimizador bio-inspirado (sistema inmune artificial) ; utilizando los principios de la programación orientada a objetos (POO). Para lo anterior se utilizará el lenguaje de programación Java. Se toma esta decisión debido a que se pretende utilizar un desarrollo pre-existente[14] y así reutilizar código a la hora de implementar los modelos neuronales propuestos.

Para el entrenamiento de los clasificadores ARTMAP se recabarán las historias médicas correspondientes a los pacientes diagnosticados con dengue, leptospirosis y/o malaria durante los últimos 10 años en el Hospital Napoleón Franco Pareja Casa del Niño (investigación retrospectiva). Se desarrollará un instrumento de recolección de datos diseñado específicamente para filtrar las historias médicas en busca de información relevante. Dicho instrumento será diseñado en conjunto con el personal médico de la Casa del Niño involucrado en la investigación, teniendo en cuenta la literatura disponible. Una vez conformado el conjunto de datos (dataset), se procederá a desarrollar los siguientes pasos:

- Partición del conjunto de datos: El conjunto de datos se dividirá en 3, una parte para el entrenamiento, otra para validación y la última para la prueba del clasificador. En cuanto al tamaño de las particiones, cabe anotar que la literatura no define criterios específicos para ser determinado, siendo elecciones populares los porcentajes que van incluso hasta el 80/20 por ciento para entrenamiento/prueba respectivamente [15]. Los elementos se asignarán a cada conjunto de manera aleatoria para reducir cualquier sesgo del investigador.

- Entrenamiento, validación y selección de características (Feature selection): Una vez establecido el conjunto de datos para entrenamiento y validación, se procederá a realizar un proceso de selección de características para establecer un subconjunto de predictores que maximice el desempeño de los clasificadores. Los resultados de esta etapa se contrastarán con la literatura en busca de hallazgos interesantes que puedan enriquecer la discusión acerca del diagnóstico diferencial y por supuesto la investigación.

- Prueba de los clasificadores: En este paso se procederá a utilizar los predictores identificados en el paso anterior para el entrenamiento de los clasificadores. Una vez finalizado el entrenamiento, se procederá a probar los clasificadores contra el conjunto de prueba por una vez, y se registrarán los resultados.

- Análisis de resultados: En esta última etapa se consolidarán los resultados de la investigación, y se procederá al análisis y desarrollo de las conclusiones de la investigación.

CAPÍTULO 1

Estado del Arte

1.1. Generalidades de las redes neuronales

Las redes neuronales artificiales (en adelante redes neuronales) surgieron como un paradigma de computación, inspirado en la estructura del sistema nervioso de los animales superiores, donde un gran número de unidades altamente interconectadas entre sí conforman una estructura de procesamiento masivamente paralela. Estas conexiones le proveen a la red una propensión al almacenamiento de conocimiento basado en la experiencia, y una gran tolerancia a los fallos. Las redes neuronales representan un modelo de computación con poder equivalente a una máquina de Turing [16], y en los inicios del estudio del problema de la computabilidad de una función, compitió con varios otros modelos de computación, incluyendo el modelo de Von Neumann, que a la postre resultó vencedor. Algunas de las capacidades más espectaculares de este modelo de computación es la capacidad de aprender de la experiencia, almacenando conocimiento en los pesos asociados a las conexiones entre neuronas; el poder utilizar información con alto niver de ruido y/o incompleta para ser procesada con resultados satisfactorios, la capacidad de reconocer y organizar datos que no habían sido vistos con anterioridad, entre otras. Todo lo anterior hace de las redes neuronales una herramienta de gran aplicabilidad en problemas como el reconocimiento de patrones, clasificación y agrupación de datos, predicción, trabajo con información ruidosa, y en general en problemas donde hay la necesidad de modelar comportamientos de sistemas donde la relación entre las entradas y la salidas no es evidente y/o muestra una gran complejidad. Las redes neuronales hoy en día se usan en gran variedad de tareas, que van desde la estabilización y supresión de ruido en telefonía de larga distancia, el reconocimiento de imágenes, filtrado de correo no deseado (Spam), hasta la predicción del comportamiento de acciones en la bolsa, por mencionar unas cuantas [17].

Una red neuronal constituye una abstracción rudimentaria del modelo biológico, donde se consideran unidades fundamentales interconectadas entre sí llamadas neuronas. Una neurona en este contexto es una versión altamente simplificada de su contraparte biológica, que consta de entradas asociadas a un conjunto de pesos, un sumador, una función de transferencia, y una salida [18]. De especial importancia resulta la función de transferencia, puesto que esta es la que define el umbral que debe ser superado por la suma de las entradas para producir una salida. La representación matemática de una

neurona artificial es la siguiente está dada por eq:sum.

$$Y = f(w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n + b) \tag{1.1}$$

Una red neuronal puede estar compuesta de una o más neuronas, organizadas en una o más capas, y pueden existir conexiones cíclicas o no, dependiendo del modelo utilizado, y la arquitectura escogida le provee a la red de ciertas características que pueden ser útiles o no para resolver un problema determinado. Al interactuar las neuronas, la red exhibe comportamientos que pueden llegar a ser muy complejos, a pesar de la relativa simplicidad de la neurona artificial. Debido a que una red neuronal se puede considerar como una agregación de funciones primitivas, se puede llegar a demostrar que una red neuronal (Multilayer Perceptrons y Radial-Basis-Function Networks) con suficientes neuronas y capas es un aproximador universal de funciones, con precisión arbitraria. En esto, las redes neuronales exhiben similitud con las series de Taylor y las series de Fourier; es más, las series de Fourier y Taylor pueden ser representadas como redes neuronales con funciones de transferencia específicas. Donde w_i representa el peso asociado a la entrada x_1, b representa al sesgo o bias, y la salida Y viene dada por la función f o función de transferencia, la cual puede tener varias formas. Los algoritmos de aprendizaje utilizados

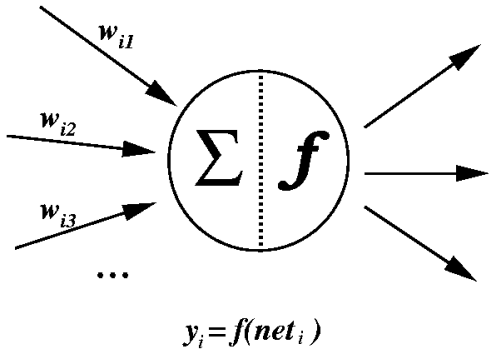

FIGURA 1.1. Modelo de neurona artificial. Tomado de http://www.idsia.ch/NNcourse/ann-overview.html

para el entrenamiento de redes neuronales se pueden clasificar de la siguiente manera: algoritmos de entrenamiento supervisado, algoritmos de entrenamiento sin supervisión y algoritmos de entrenamiento por refuerzo. Los algoritmos de entrenamiento supervisado requieren el uso de ejemplos del que debería ser el comportamiento adecuado de la red en presencia de entradas específicas. Ejemplos de entrenamiento supervisado son la regla de aprendizaje del perceptron, la regla delta (Widrow-Hoff) y el algoritmo de retro propagación (Backpropagation). De otro lado, los algoritmos de aprendizaje sin supervisión no requieren ejemplos de comportamiento adecuado, sino que a través de un proceso ya sea de refuerzo u auto-organización modifican la red de tal manera que aprenden a clasificar un espacio de entradas determinado. Por último, en el caso de los algoritmos de entrenamiento por refuerzo la red no recibe ejemplos de comportamiento correcto, sino una recompensa o calificación por la acción tomada; siendo entonces la meta de la red neuronal, la maximización de la recompensa recibida.

1.2. Aplicación de las redes neuronales ART en el diagnóstico médico

Un problema comúnmente mencionado por el personal médico a la hora de aceptar el uso de herramientas de inteligencia artificial basadas en redes neuronales (específicamente MLPs o Perceptrons Multicapa), es la poca interpretabilidad de una red neuronal entrenada. El personal médico quiere saber muchas veces por qué la red toma la decisión, y ven con reticencia su naturaleza de caja negra. Esta es la principal razón por la cual se escogió la familia ARTMAP para adelantar este trabajo, por encima de algoritmos tales como los Perceptron Multicapa, Las redes RBF, las máquinas de soporte vectorial, entre otros. Estos algoritmos, ofrecen potencialmente un gran desempeño (especialmente las Máquinas de Soporte Vectorial - SVM), pero debido a su "opacidad" y poca interpretabilidad una vez entrenados, no permiten la extracción de conocimiento de forma sencilla, mientras que las memorias de una red ARTMAP entrenada son fácilmente interpretables. Por otro lado, no existe una técnica de Machine Learning que sea estrictamente mejor que las demás en todos los problemas de aprendizaje (esto se conoce como el No-Free-Lunch Theorem [19]), y en el caso del diagnóstico médico muchas técnicas de Machine Learning han sido aplicadas, sin encontrar una que supere a las demás en todas las aplicaciones.

Profundizando en lo anterior, podemos destacar que la naturaleza auto-organizativa de las redes ARTMAP permite la interpretabilidad de sus memorias, puesto que cada memoria representa una categoría o patrón que se relaciona a una de las clases examinadas en un problema de entrenamiento supervisado. Lo anterior permite extraer a partir de las memorias de una red entrenada, una serie de reglas IF-THEN que pueden justificar las decisiones tomadas por la red en un momento dado. Por otro lado, la capacidad de aprendizaje casi perfecto en una sola presentación del conjunto de datos le confiere a las redes ARTMAP una ventaja importante sobre los MLPs, puesto que estos últimos pueden necesitar muchas iteraciones de entrenamiento para lograr los mismos resultados.

De acuerdo a lo anterior, la literatura menciona varios ejemplos de utilización de redes ART/ARTMAP en el diagnóstico médico. Downs et al. [11] reportan la utilización de las redes ARTMAP en la prognosis de pacientes coronarios, diagnóstico de cancer de seno y el diagnóstico de infartos agudos al miocardio. Markuzon et al.[12] reportan el uso de redes ARTMAP para la predicción de eventos adversos en pacientes sometidos a colecistectomía y pacientees diabeticos, con un rendimiento comparable al de la regresión logística. Goodman et al. [13] reporta los resultados del uso de redes ARTMAP para la predicción de la estadía intra-hospitalaria de pacientes admitidos con neumonía, obteniendo resultados satisfactorios.

Carpenter et al [20], reportan el desempeño del modelo ARTMAP IC en el diagnóstico médico, utilizado varios conjuntos de datos clásicos de la literatura relacionada. El rendimiento es satisfactorio, y muchas veces superior a técnicas de machine learning alternativas. Carpenter et al [21] demuestran el uso de ARTMAP y de un procedimiento para la extracción de reglas a partir de las memorias de una red entrenada, sobre un conjunto de datos de pacientes de diabetes, con buenos resultados. Por otro lado Modai et al [22] reportan el uso de ART para sugerir tratamientos a pacientes psiquiátricos, y la comparación del desempeño de la red con personal médico experimentado. Los resultados

fueron bastante prometedores, y la red se comportó de manera comparable al personal médico en términos de desempeño.

1.3. Diagnóstico y prognosis de fiebres hemorrágicas usando redes neuronales

La aplicación de redes neuronales (y en general técnicas de Machine Learning) al diagnóstico y/o prognosis de enfermedades tropicales tales como el dengue, la leptospirosis y malaria, se ha reportado en la literatura especializada. Faisal et al [23] reportan el uso de una red neuronal SOM (Self Organizing Map - Mapa Auto-Organizativo) para la identificación de los signos y síntomas que permiten diferenciar entre pacientes con dengue grave y pacientes sanos. Faisal et al [24] también reportan el uso de perceptrones multicapa para la estimación del riesgo de complicaciones en pacientes con dengue, con una exactitud entre 70,7 % y 75 %; y posteriormente Ibrahim et al [25] presentan una mejora que llevó la exactitud del sistema hasta 96,27 %.

Barral-Netto et al [26] reportan el uso de redes neuronales para el diagnóstico de malaria en la amazonía de Brasil. El desempeño del sistema llegó al 80 % de diagnósticos correctos, en [27] se reporta el uso de redes neuronales y redes bayesianas como soporte para el diagnóstico de malaria, con buenos resultados.

En el caso de la leptospirosis, no se encontró ninguna publicación que hiciera referencia al uso de Machine Learning para el diagnóstico. De la misma forma, no se encontró ninguna publicación que involucrara el uso de machine learning en el diagnóstico diferencial de estas 3 enfermedades .

CAPÍTULO 2

Desarrollo de una familia de clasificadores basada en ART - ARTMAP

2.1. Teoría de la Resonancia Adaptativa (ART) y ARTMAP

La Teoría de la Resonancia Adaptativa (ART) fue propuesta por Grossberg como un intento de resolver el problema de la estabilidad-plasticidad [28]. En las redes neuronales como el Perceptron multicapa, si se desea volver a entrenar con nueva información una red entrenada con anterioridad, es necesario repetir el entrenamiento con tanto los ejemplos utilizados en el entrenamiento original, como con los nuevos ejemplos; de lo contrario se corre el riesgo de que se "olviden" los patrones memorizados inicialmente. La solución propuesta por Grossberg incluye un indicador de similaridad diseñado para evitar este olvido catastrófico de patrones. Las redes ART son un tipo especial de red competitiva, donde la activación de las neuronas depende de la similitud del patrón almacenado en sus pesos con la entrada considerada por la red, de manera similar - en primera instancia - a un Mapa de Kohonen[29]. ART comprende una familia de redes neuronales entre las que podemos citar ART1 [28] (valores binarios), ART2 [30] (valores reales), ART3 [31] (modelo con motivación biológica) y FuzzyART (incorpora lógica difusa para manejar entradas reales entre 0 y 1) [32] (estas últimas operan con valores reales), que funcionan utilizando entrenamiento no supervisado; y FuzzyARTMAP[15], ARTMAP-IC[20] y Default ARTMAP [4] entre otras, estas últimas operando bajo entrenamiento supervisado.

ART está inspirada en el funcionamiento del sistema visual de los mamíferos, y específicamente en la forma como el cerebro interpreta las señales provenientes de la retina. La retina es una capa de tejido sensible a la luz presente en la parte posterior interna del ojo, y está compuesta por células que funcionan como foto-receptores (los conos y los bastones). Una característica interesante de la retina es la disposición de las células foto-sensibles, estas se encuentran detrás de los vasos sanguíneos que las irrigan, dando como resultado que ciertas partes del campo visual se encuentren obstruidas (puntos ciegos). Sin embargo, el cerebro "completa" la información faltante, por lo que los seres humanos por ejemplo, no perciben un disco negro en el lugar en el que se encuentra el punto ciego (disco óptico). Esta capacidad de manejar imágenes parcialmente ocluidas se incorporó en la Teoría de la Resonancia Adaptativa, a través de un indicador de similitud denominado ART Matching Rule.

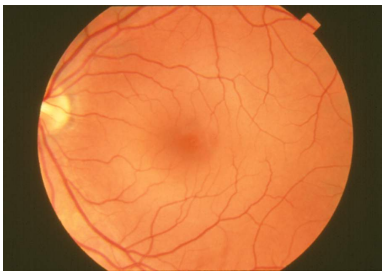

FIGURA 2.1. Fondo del ojo humano. La zona amarilla de la izquierda es el disco óptico. National Eye Institute, National Institutes of HealthDisponible en http://www.nei.nih.gov/photo/eyedis/images/eye15-72.jpg

2.1.1. ART1

Una red ART1 está compuesta por 2 capas de neuronas. La primera capa representa la memoria a corto plazo de la red (Short Term Memory - STM), y la segunda representa la memoria a largo plazo (Long term Memory - LTM). Cuando un patrón es presentado a la red, este se almacena en la memoria STM; posteriormente cada neurona de la memoria a largo plazo recibe una copia del patrón en STM y se realiza una competición entre las neuronas, de manera que al final de esta solo una tendrá una activación diferente a 0. A este tipo de competición se le conoce como Winner-Take-All (WTA) y está inspirado en la activación de grupos de neuronas en el córtex cerebral en respuesta a estímulos determinados. Una parte importante de los modelos ART son las matrices de pesos Bottom-Up y Top-Down, que comunican en su orden, la capa STM con la LTM y viceversa. La función de la matriz Bottom-Up es hacer las veces de filtro adaptativo y permitir la selección de una neurona en la capa LTM como posible categoría de reconocimiento, mientras que la matriz Top-Down codifica los patrones aprendidos por cada neurona (En la práctica se selecciona la neurona con mayor activación, y dicha activación es el producto escalar entre la entrada y los pesos Bottom-up de cada neurona/categoría). Cuando una neurona de la capa LTM es seleccionada como candidata, se compara el patrón codificado por la matriz Top-Down para dicha neurona con el patrón examinado con miras a establecer la similitud de estos. Esta operación de comparación se realiza en el modelo ART1 a través de un AND lógico entre el patrón almacenado en la memoria activada (W) y el patrón de entrada (I) y el cálculo de la norma L1 del vector resultante de dicha operación lógica, como se muestra en (2.1). Si el resultado de la división del vector resultante y la norma del patrón examinado supera un umbral de vigilancia prestablecido (ρ), la red entra en estado de "resonancia" y procede a incorporar el nuevo patrón a la memoria activada. Si no se supera el umbral de vigilancia, la neurona activada es inhibida (reset) para aprender el patrón por lo que resta de la presentación de este, y se repite la competición. Si todas las neuronas de la capa LTM son inhibidas por este mecanismo, se puede agregar un nuevo nodo a la capa LTM para memorizar el patrón actual.

$$\frac{|W \cap I|}{|I|} \geq \rho \tag{2.1}$$

Aprendizaje en ART1 El aprendizaje de un nuevo patrón en ART1 esta dado por (2.2). Vale la pena resaltar que ART1 (y en general toda la familia ART) puede operar bajo 2 modalidades de aprendizaje: Aprendizaje Lento (Slow-Learning) y Aprendizaje Rápido (Fast-Learning). En el modo de aprendizaje lento (el modo original de operación del modelo

de Grossberg), la convergencia de los pesos requiere varias presentaciones de cada patrón, puesto que en cada iteración solo se aprende una fracción del patrón presentado (la tasa de aprendizaje - β - es menor a 1); mientras que en el modo de aprendizaje rápido se asume que el patrón es presentado durante un tiempo suficiente para garantizar la convergencia en una sola iteración (la tasa de aprendizaje - β - es igual a 1). En la mayoría de aplicaciones en ingeniería se prefiere el modo de aprendizaje rápido, puesto que permite la memorización del conjunto de datos en pocas iteraciones, con la posible desventaja de que las memorias obtenidas varían de acuerdo al orden de presentación de los patrones. En cualquiera de los 2 casos, el mecanismo de aprendizaje definido para ART1 garantiza que los pesos de la matriz Top-Down convergerán al cabo de un número finito de presentaciones del conjunto de datos.

$$W_j = \beta(W_j \cap I_j) + (1 - \beta)W_j \tag{2.2}$$

Pseudocódigo 1 Algoritmo ART

```
para todo I en ejemplosEntrenamiento hacer
  mientras Verdadero hacer
    para todo c en categorias hacer
      calcularActivación(c)
    fin para todo
    W ← seleccionarCategoríaConMayorActivación(I)
    similitud ← calcularSimilitud(W, I)
    si similitud ≥ ρ entonces
      break
    sino
      inhibirCategoría()
    fin si
  fin mientras
  aprender(W, I)
fin para todo
devolver W
```

2.1.2. FuzzyART

FuzzyART fue propuesto con la finalidad de superar la principal limitación de ART1, la cual es que su uso está restringido a patrones binarios; para tal fin se remplazó el operador lógico AND utilizado en el cálculo de la similitud entre memorias y patrones por el AND difuso (Fuzzy AND) [33], representado por el símbolo $\wedge$. De esta manera FuzzyART puede trabajar con valores continuos dentro del intervalo [0,1], expandiendo el conjunto de posibles aplicaciones de este modelo neuronal. En el modelo FuzzyART la activación de las neuronas en la capa LTM está dada por (2.3). En esta expresión, α es un parámetro que controla la importancia del segundo término; el cual estimula a la red a escoger categorías que hayan codificado anteriormente pocos patrones.

$$T_j = |I \wedge W_j| + (1 - \alpha)(|I| - |W_j|) \tag{2.3}$$

Complement Coding Las redes ART en general pueden exhibir un comportamiento no deseado llamado proliferación de categorias[34]. En este fenómeno, la red genera más categorías de las necesarias para discriminar efectivamente el conjunto de entrenamiento, llevando a la degradación del desempeño tanto en términos de exactitud de las predicciones como en tiempo empleado para el entrenamiento. Experimentalmente se ha verificado que

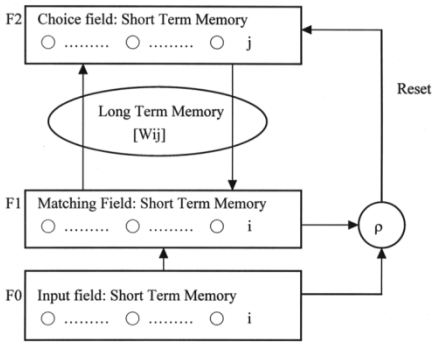

FIGURA 2.2. Estructura de la red ART/FuzzyART. Tomado de http://www.cs.nthu.edu.tw/ jang/courses/cs5611/project/14/

un tipo especial de normalización llamado Código al Complemento (Complement Coding) alivia el problema de proliferación de categorías, llevando a mejores resultados. El código al complemento implica duplicar el tamaño de los vectores de entrada y establecer la norma de acuerdo a (2.4). El código al complemento es parte esencial en el funcionamiento de los modelos de la familia ART y ARTMAP.

$$\begin{aligned} I &= (a, 1-a) \\ |I| &= |(a, 1-a)| \\ &= \textstyle\sum_{i=1}^{M} a + (M - \sum_{i=1}^{M} a) \\ &= M \end{aligned} \tag{2.4}$$

2.1.3. ARTMAP

ARTMAP es un modelo neuronal que consta de una red ART y un mapa asociativo para aplicaciones de entrenamiento supervisado. La red ART intenta hacer una predicción acerca de la clase a la que pertenece el patrón presentado, y si dicha predicción es equivocada la vigilancia se incrementa (la vigilancia se lleva hasta el mínimo valor necesario para causar la inhibición de la neurona seleccionada más un pequeño valor dado por ϵ) de acuerdo a un algoritmo llamado Match-Tracking (MT) y se vuelve a presentar el patrón para forzar a la red a hacer una predicción correcta (2.5). El proceso se repite hasta que la red ART seleccione una memoria que prediga correctamente la categoría del vector de entrada o se agregue una nueva neurona a la capa LTM, creándose así una nueva categoría. Una vez esto sucede, la vigilancia se disminuye hasta su valor inicial. La aplicación del algoritmo Match-Tracking genera memorias de granularidad variada, y permite la memorización de patrones raros que solo se presentan una vez dentro del conjunto de entrenamiento. Para una descripción en detalle del funcionamiento de ARTMAP, el lector interesado puede consultar [15].

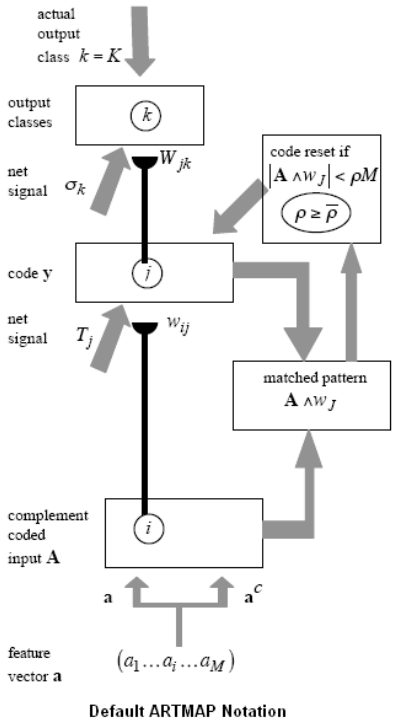

FIGURA 2.3. Estructura de los modelos ARTMAP.[4]

Variaciones de ARTMAP El modelo original ARTMAP incluía un par de redes ART1 (ARTa y ARTb), acopladas a través de un mapa asociativo, ARTa categorizaba los patrones y ARTb hacía lo propio con las etiquetas de entrenamiento. Posteriormente se presentó la versión Simplified ARTMAP [35] que incluía un solo módulo ART, de manera que la complejidad computacional del modelo se viese reducida a expensas de un mínimo de capacidad predictiva. Al cambiar el modulo ARTa por una red FuzzyART se obtiene la red FuzzyARTMAP. Carpenter y Markuzon [20] luego proponen ARTMAP-IC (Instance Counting), que introduce varios cambios al modelo FuzzyARTMAP tradicional: El primero, el cambio del esquema de competición WTA en la capa LTM por una regla de activación distribuida conocida como Q-Rule durante la operación de prueba (El entrenamiento sigue haciéndose bajo el esquema de competición WTA). A través del uso de la regla Q-Rule, la predicción de la red se hace a partir de la suma de las activaciones individuales de las Q neuronas con mayor activación, escogiéndose la clase con mayor suma total. El segundo, la inclusión de un algoritmo Match-Tracking modificado (MT-) (2.6) que permite la codificación de memorias para casos inconsistentes (dentro del entrenamiento supervisado, 2 casos son inconsistentes si exhiben el mismo patrón en el correspondiente vector de entrada pero corresponden a clases distintas). Y el tercero, la adición de una capa extra que sesga las predicciones de la red en función de la cantidad de instancias identificadas por cada neurona LTM durante el entrenamiento. ARTMAP-IC es muy interesante para este trabajo puesto que los casos inconsistentes son particularmente frecuentes en el dominio del diagnóstico médico. Por otro lado, Distributed ARTMAP [36] es una versión de ARTMAP que utiliza activación distribuida tanto en el entrenamiento como en el modo de prueba. Luego de varios años de validación experimental de los modelos ARTMAP, Carpenter propuso la versión Default ARTMAP [4], la cual resumía las mayores ventajas de los modelos anteriores. Default ARTMAP supone el uso de activación distribuida

Pseudocódigo 2 Algoritmo básico ARTMAP

```
para todo I en ejemplosEntrenamiento hacer
  mientras Verdadero hacer
    para todo c en categorias hacer
      calcularActivación(c)
    fin para todo
    W, cat ← seleccionarCategoríaConMayorActivación(I)
    similitud ← calcularSimilitud(W, I)
    si similitud ≥ ρ y claseCorrecta(cat) entonces
      break
    sino
      inhibirCategoría(cat)
      aumentarVigilancia()
    fin si
  fin mientras
  aprender(W, I)
fin para todo
devolver clase(cat)
```

durante el entrenamiento, Match-Tracking MT- y activación WTA durante el modo de prueba. Default ARTMAP utiliza una nueva regla para calcular la predicción en el modo de prueba, llamada Content Addressable Memory (CAM) Rule. Esta regla requiere un parámetro llamado potencia, que especifica que tan "distribuida" será la respuesta. A mayor potencia, menos neuronas serán utilizadas para calcular la respuesta de la red.

$$\rho = \frac{|W \cap I|}{|I|} + \epsilon \textbf{ (MT+)} \tag{2.5}$$

$$\rho = \frac{|W \cap I|}{|I|} - \epsilon \textbf{ (MT-)} \tag{2.6}$$

2.2. Implementación de una familia de clasificadores ARTMAP

La implementación de los modelos neuronales se hizo utilizando el lenguaje de programación JAVA, y el paradigma de la programación orientada a Objetos (OOP). Debido a que los modelos ART/ARTMAP están emparentados entre sí, se construyó una jerarquía de clases (`ART1`,`FuzzyART`,`FuzzyARTMAP`, `DefaultARTMAP` y `ARTMAP_IC`) para reutilizar el código escrito, y así acelerar el desarrollo. La raíz de la jerarquía, para los propósitos de este trabajo, es la clase `ART1`.

2.2.1. ART1

`ART1` representa el modelo propuesto por Carpenter y Grossberg en [28], donde la capa LTM está representada a través de `ArrayList`, y en cada posición de dicha lista encontramos un descendiente de la clase `Neuron`. Los pesos BottomUp y TopDown se almacenan en los descendientes de `Neuron` a través de miembros de tipo `ArrayList<Float>`.

FIGURA 2.4. Diagrama de clase para ART1.

Por otro lado, la capa STM se representa a través de `ArrayList<Float>`, y sobre esta capa se realizan las operaciones lógicas correspondientes al algoritmo ART1. El método `processData(List<Float>)` devuelve el resultado del proceso de clustering, a través de una instancia de `ArrayList<Float>`, donde todas las posiciones son cero excepto la correspondiente a la categoría asignada al vector de entrada.

La implementación del funcionamiento de la red se reduce esencialmente a un ciclo `while`, que se ejecuta mientras no se halle una categoría (neurona) que supere el umbral de vigilancia establecido. Una vez se encuentre la categoría adecuada, la red procede a modificar los pesos de la neurona seleccionada de acuerdo a las ecuaciones de entrenamiento. El método `testForReset()` es un punto de extensión que será modificado para la implementación subsecuente de los modelos ARTMAP. El código fuente del modelo ART1 se encuentra en el anexo 10.

2.2.2. FuzzyART

El modelo FuzzyART representa la extensión de ART1 para lidiar con entradas continuas en el intervalo [0,1]. Para lo anterior, se reemplaza la operación lógica AND presente en ART1 por el AND difuso. La clase correspondiente a este modelo neuronal es la clase `FuzzyART`, y dicha clase sobrecarga los métodos `updateSTMbField(List<Float> pattern)` y `learn(Neuron neuron, float patternNorm)`; las versiones sobrecargadas de estos métodos reemplazan utilizan el AND difuso. Por otro lado, la capa LTM está representada por una lista de objetos de tipo `FuzzyARTNeuron` para acomo-

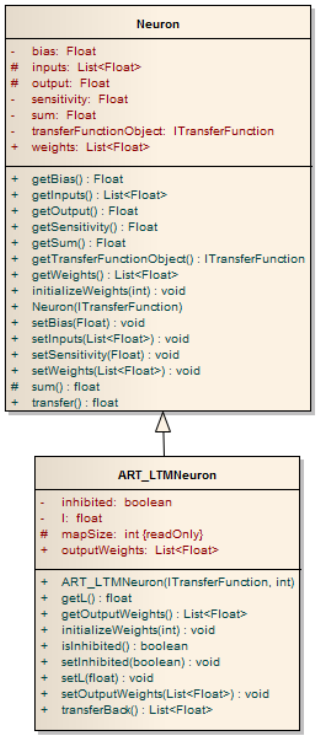

FIGURA 2.5. Diagrama de clase para las neuronas de ART1.

dar las características propias de este modelo, y el método (también sobrecargado) `init(int numberOfClasses, float l)` permite la creación de la capa LTM con las neuronas adecuadas. El código fuente del modelo FuzzyART y de la clase FuzzyARTNeuron se encuentran en los anexos 11 y 4 respectivamente.

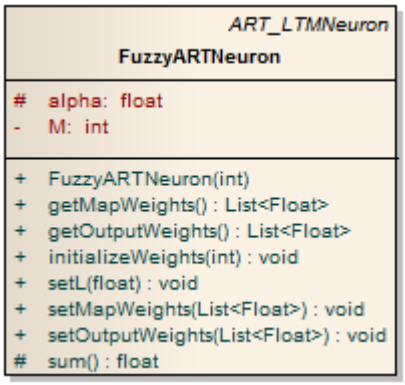

FIGURA 2.6. Diagrama de clase para las neuronas de FuzzyART.

2.2.3. FuzzyARTMAP

FuzzyARTMAP es un modelo neuronal de entrenamiento supervisado (a diferencia de la familia ART), que consta de 2 (solo 1 en el modelo simplificado) módulos FuzzyART, conectados a un mapa auto asociativo. La idea general detrás del funcionamiento de

esta red es incluir un criterio adicional para definir si la neurona seleccionada para aprender el patrón presentado es inhibida. En los modelos ARTMAP hay un conjunto de pesos adicional que va desde las neuronas de la capa LTM hacia el mapa auto asociativo, y estos pesos codifican la categoría que predice cada neurona. Entonces, además del umbral mínimo de vigilancia de ART, si no se predice la categoría correcta la neurona seleccionada también es inhibida por el resto de la presentación del vector de entrada.

La clase que representa al modelo FuzzyARTMAP es `FuzzyARTmap` (anexo 12), y

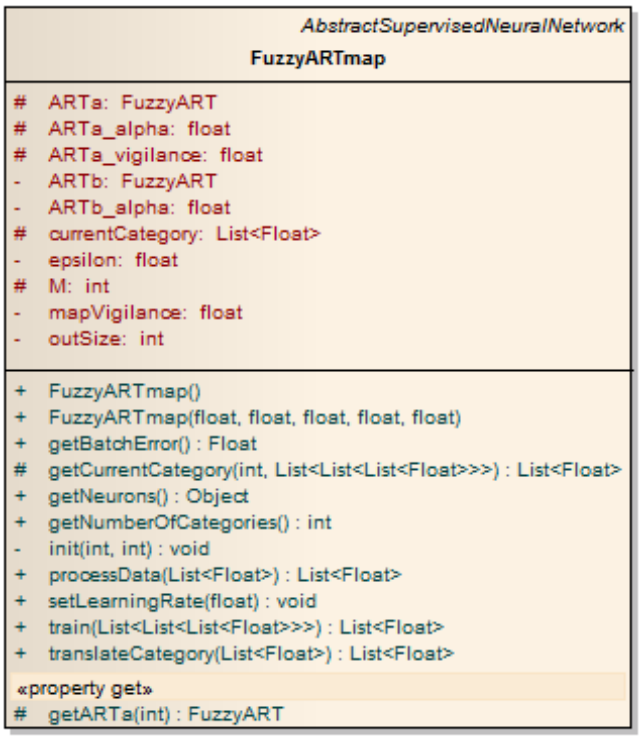

FIGURA 2.7. Diagrama de clase para FuzzyARTMAP.

se compone de una clase interna llamada `ModifiedFuzzyART`, la cual es descendiente de `FuzzyART` y acomoda el nuevo mecanismo de inhibición. La clase `FuzzyARTmap` cuenta con un par de métodos, `List<Float>train(List<List<List<Float>>> trainingSet)` y `List<Float> processData(List<Float> inputs)`, que permiten el entrenamiento y hacer predicciones, respectivamente. Es necesario destacar que el resto de clases que implementan los demás modelos neuronales de la familia ARTMAP considerados en este trabajo, son descendientes de `FuzzyARTmap`.

2.2.4. DefaultARTMAP

La clase que representa a este modelo neuronal es `DefaultARTMAP`. `DefaultARTMAP` (anexo 13) extiende la clase `FuzzyARTmap` para incorporar la activación distribuida en el modo de prueba. A diferencia del modelo FuzzyARTMAP, DefaultARTMAP solo realiza una competición WTA (Winner Takes All) en el modo de entrenamiento; lo cual puede conferirle una mejor capacidad de generalización. La clase `DefaultARTMAP` implementa el algoritmo de activación conocido como IG-CAM [36], que le permite a la red dar una

respuesta normalizada para cada una de las clases consideradas, la cual se puede interpretar como una distribución de probabilidad. La regla IG-CAM determina cuantas neuronas van a ser usadas para generar la predicción (de manera similar al parámetro K en el algoritmo KNN) Para lograr lo anterior, la clase `DefaultARTMAP` incorpora una clase interna que desciende de `ModifiedFuzzyART`, llamada `DefaultART`.

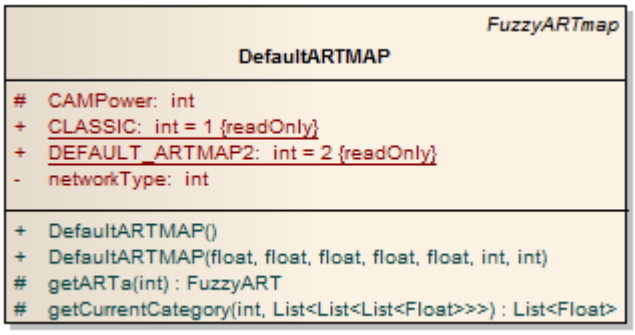

FIGURA 2.8. Diagrama de clase para DefaultARTMAP.

2.2.5. ARTMAP-IC

ARTMAP-IC (Instance Counting ARTMAP) es un modelo de red neuronal que extiende el funcionamiento de DefaultARTMAP agregándole una tercera capa, cuya misión es sesgar las predicciones de la red en función del número de veces que cada neurona ha sido activada. La activación de cada neurona es multiplicada por el número de veces que ha sido seleccionada para el aprendizaje en el modo de entrenamiento, y el resultado es dividido entre la suma total del número de activaciones de las neuronas seleccionadas para hacer la predicción. La idea es, junto con el algoritmo MT-, ayudar a lidiar a la red con casos

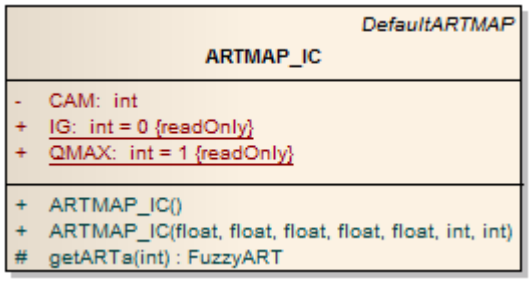

FIGURA 2.9. Diagrama de clase para ARTMAP$_I C$.

inconsistentes. Esta capacidad de manejar casos inconsistentes es de especial importancia en el diagnóstico médico, por lo que se incluye este modelo dentro del trabajo presentado. Para implementar el conteo de instancias (IC), se creó una nueva clase, `ART_ICNeuron`, que registra el número de veces que la neurona ha sido seleccionada para aprender un patrón, y contiene un campo que almacena la activación de la neurona en el modo de prueba (distribuido). El código fuente del modelo FuzzyART y de la clase FuzzyARTNeuron se encuentran en los anexos 14 y 5 respectivamente.

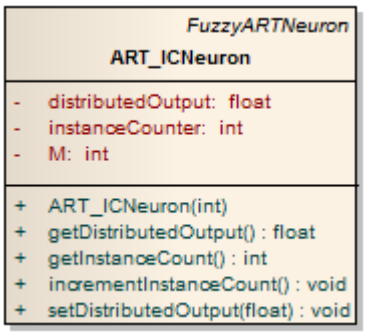

FIGURA 2.10. Diagrama de clase para ART$_I CNeuron$.

TABLA 2.1. Desempeño de los diferentes modelos ARTMAP

Modelo	Circle in a Square	Frey/Slate	Heart Disease
FuzzyARTMAP	94,22 %	91,47 %	81,13 %
ARTMAP-IC	94,66 %	91,40 %	81,13 %
DefaultARTMAP2	95,88 %	89,25 %	81,13 %

2.2.6. Pruebas y validación de los modelos

Para comprobar que los modelos hubiesen sido implementados correctamente, estos fueron puestos a prueba sobre varios conjuntos de datos reseñados en la literatura. Los conjuntos escogidos para tal fin fueron Círculo dentro de un Cuadrado (Circle in a Square), donde la tarea es diferenciar entre puntos que caen dentro de un círculo o fuera de este (Este conjunto de datos es similar al usado en [15]); el conjunto de datos para reconocimiento de dígitos escritos Frey/Slate [37], y por último el conjunto de datos UCI Heart Disease [38], donde se diferencian pacientes con enfermedad coronaria de pacientes sanos. Los resultados muestran que el desempeño de los modelos es comparable con los registrados en los artículos originales, por lo que se asume que la implementación es correcta.

CAPÍTULO 3

Dengue, Leptospirosis y Malaria

El Dengue, Leptospirosis y Malaria son enfermedades febriles endémicas en Colombia y de notificación obligatoria a las entidades estatales, además se consideran peligros potenciales a la salud pública por la dinámica de contagio y las posibles consecuencias mortales de los cuadros más severos. Una característica importante de estas 3 enfermedades (y otras fiebres hemorrágicas) es que en sus formas más leves son prácticamente indistinguibles de un resfriado común, e inclusive entre ellas mismas. En los cuadros serios se pueden presentar manifestaciones hemorrágicas que contribuyen a la muerte.
A pesar de las similitud en síntomas y signos clínicos, estas enfermedades son bastante diferentes en términos de etiología, vector de transmisión y mecanismos de infección; y por supuesto esto implica tratamientos distintos. Cabe resaltar que un tratamiento temprano mejora ostensiblemente el pronóstico de los pacientes. A continuación, se verán más en detalle las características de cada enfermedad.

3.1. Dengue

La fiebre dengue es un síndrome benigno causado por varios virus transmitidos por artrópodos (mosquito aedes aegypti), y se caracteriza por fiebre bifásica, mialgia o artralgia, leucopenia y linfadenopatía. El dengue grave con manifestaciones hemorrágicas es una complicación severa y a menudo fatal de la fiebre dengue, que se caracteriza por la aparición de anormalidades en la hemostasia y la permeabilidad capilar que conducen, en casos severos, a un estado llamado Síndrome de Shock por Dengue (DSS, por sus siglas en inglés)[39]. Existe un consenso general acerca de la existencia de 4 miembros antigénicamente distintos en el subgrupo del dengue dentro del género Flavivirus. Dichos miembros se denominan dengue tipo 1 hasta tipo 4. La fiebre dengue se ha conocido históricamente como la dengue, fiebre dandy, fiebre rompe-huesos y fiebre rompe-corazones. Los sinónimos históricos del dengue grave incluyen púrpura trombocitopénico infeccioso, DSS y fiebre Thai, Filipina o de Singapur.

La inoculación de cepas de dengue de patogenicidad reconocida en humanos no produce infección demostrable en pollos adultos, lagartos, conejillos de indias, conejos, hamsters o ratas algodoneras. Especies pertenecientes a los géneros Macacus, Pongidae, Certhopicus, Cercocebus, Papio, Hylobates y Pan pueden ser infectados por picaduras

TABLA 3.1. Algunos posibles signos del Dengue [1]

Tipo	Signo
Clínicos	Fiebre alta
	Fiebre con duración menor a 10 días
	No hay escalofríos
	Cefalea frontal
	Mialgia
	Artralgia
	Dolor retro-ocular
	Exantema (especialmente petequial)
	Signo de islas blancas en mar rojo
	Eritema facial
	No hay edema de pie
	No suele haber manifestaciones flu-like
	Dolor abdominal
	Hematemesis
	Visceromegalia
	Sangrado en sitios de venopunción
Paraclínicos	Leucopenia
	Linfocitosis
	Trombocitopenia
	Transaminasas (ALT y AST) elevadas
	TP-TPT prolongados

de mosquitos infectados o por inyección del virus. La infección en estos últimos no es particularmente importante, pero la viremia ocurre a niveles suficientes para infectar a mosquitos. Investigaciones han revelado que el macaco filipino (Macaca philipinensis) resiste la infección, mientras que el macaco japonés (Macaca fuscatus) no. Los trabajos de investigación en Malasia han revelado un ciclo complejo de transmisión del dengue, que involucra especies de monos y el mosquito Aedes niveus, el cual se alimenta tanto de monos como humanos [40]. Se han reconocido extensas epizootias de dengue tipo 2 en primates de África oriental [41], y se han logrado aislar cepas de dengue tipo 2 en humanos, luego de ser transmitidas por mosquitos que se alimentaron previamente en primates [42]. Sin embargo, estudios genéticos y epidemiológicos han concluido que el dengue humano urbano y el dengue de monos están relativamente compartimentalizados[43]. En el ciclo urbano, el dengue es transmitido por mosquitos domésticos que se aparean en y alrededor de viviendas humanas, y el virus se desplaza por las rutas humanas de comunicación.

Manifestaciones clínicas El dengue es una enfermedad que cursa con dolor de cabeza, mioartralgias y exantema, cuya severidad y manifestaciones clínicas varían de acuerdo a la edad del paciente. Hasta en el 80 % de los casos en niños, la infección es asintomática, y en el 20 % restante muchas veces no puede ser diferenciada fácilmente de otros padecimientos infantiles [1]. Luego de un periodo de incubación de 4 a 7 días, aparecen los primeros síntomas. La fiebre desaparece al cabo de aproximadamente 7 días, y la reaparición de la fiebre se considera de mal pronóstico, puesto que indica la posible llegada de dengue grave con manifestaciones hemorrágicas.

3.2. Leptospirosis

La leptospirosis es una zoonosis de distribución mundial causada por espiroquetas del género Leptospira. Las leptospiras infectan una variedad de animales salvajes y domésticos que excretan el microorganismo en su orina. Los seres humanos, quienes se contagian por el contacto con animales enfermos o a través de la exposición al agua o suelo contaminado por la orina de animales infectados, desarrollan una enfermedad febril aguda que puede ser seguida de un cuadro más severo (enfermedad de Weil) y algunas veces fatal que puede incluir ictericia y falla renal, meningitis, miocarditis, neumonía hemorrágica o colapso hemodinámico. Solo hasta el descubrimiento del agente causante de la leptospirosis en 1914, esta se diagnosticaba erróneamente como fiebre amarilla o malaria[2]. Incluso hoy persiste la confusión entre leptospirosis y otros cuadros febriles tales como el dengue, hepatitis, malaria e influenza, por mencionar algunos.

Las leptospiras son bacterias gram-negativas delgadas, flexibles y finamente enrolladas, de naturaleza aeróbica y caracterizadas por un crecimiento lento. La infección por leptospira ocurre en todo el mundo, siendo endémica en la mayoría de climas templados y tropicales, con una particular incidencia durante o exactamente después de periodos de alta precipitación. Las estadísticas alrededor de la leptospirosis suelen no ser confiables debido a que la presentación de la enfermedad suele ser no-específica y además porque las capacidades de diagnóstico suelen ser limitadas en los países con mayor cantidad de casos. Grandes epidemias de leptospirosis han sido documentadas luego de investigaciones sobre enfermedad febril asociada a inundaciones, tormentas tropicales y huracanes en el

TABLA 3.2. Algunos posibles signos de la Leptospirosis [2]

Tipo	Signo
Clínicos	Fiebre alta en picos
	Fiebre con duración de 2 a 7 días
	Escalofríos
	Malestar intenso
	Cefalea global
	Mialgia
	Dolor en las pantorrillas
	Vómito
	Exantema polimorfo
	Bradicardia relativa (Signo de Faget)
	Dolor abdominal
	Sufusión conjuntival
	Visceromegalia
	Ictericia (Enfermedad de Weil)
	Meningitis aséptica
	Uveítis
	Miocarditis
Paraclínicos	Leucocitosis
	Neutrofilia
	Trombocitopenia
	Transaminasas, bilirrubina elevadas
	BUN y creatinina elevadas
	Proteína C reactiva elevada
	Elevación de CPK (creatina-fosfocinasa)

caribe y américa central. La lluvia resultante de dichos fenómenos naturales incrementa la exposición humana a aguas subterráneas y suelo contaminados con leptospira, resultando en un número elevado de casos.

Manifestaciones clínicas La severidad de la infección por leptospira varía desde manifestaciones subclínicas solo detectadas por seroconversión entre personas con exposición frecuente a las leptospiras hasta dos síndromes reconocibles clínicamente, la leptospirosis anictérica (la forma más leve) que representa un 90 % de los casos, y la enfermedad de Weil[2]. En ambas formas, la enfermedad cursa con una fase séptica donde se puede aislar la leptospira en sangre, y una fase inmune, donde la leptospira solo se puede encontrar en la orina. El periodo de incubación es usualmente de 5 a 14 días pero puede ir desde pocos días hasta un mes o más. La fase séptica comienza con fiebre alta, repentina y la defervescencia anuncia el comienzo de la fase inmune.

3.3. Malaria

La malaria es una enfermedad causada por cualquiera de los siguientes microorganismos: Plasmodium Falciparum, Plasmodium Vivax, Plasmodium Ovale y Plasmodium Malarie. De los anteriores, solo P. Falciparum y P. Vivax se encuentran en Colombia. La infección por P. Falciparum suele ser la más mortal debido a que este puede invadir a glóbulos rojos de todas las edades y muchas veces es resistente a varios medicamentos. Por otro lado, P. Falciparum no cursa con etapas inactivas y por lo tanto no causa recaidas tardías [3]. En contraste a lo anterior, tanto P. Vivax como P. Ovale pueden causar recaídas tardías desde los 6 hasta 11 meses o incluso más, desde la infección inicial.

Los seres humanos adquieren la malaria a partir de los esporozoitos transmitidos por la picadura del mosquito Anopheles. Los esporozoitos viajan por el torrente sanguíneo hasta el hígado, donde invaden los hepatocitos y maduran hasta convertirse en esquizontes o hipnozoitos. Los esquizontes tienen un papel fundamental en el ciclo de vida de los plasmodios que infectan a los humanos; cada esquizonte puede producir grandes cantidades de merozoitos, los cuales parasitan los glóbulos rojos y producen más merozoitos. De manera alternativa, algunos merozoitos dentro de los glóbulos rojos se convierten en gametocitos (formas sexuales) que eventualmente regresan al mosquito hembra vector y dentro de ellos crean los esporozoitos necesarios para continuar con el ciclo de transmisión a los humanos. Las complicaciones de la infección pueden incluir la malaria cerebral (convulsiones y coma), hipoglicemia, acidosis láctica, anemia severa, edema pulmonar, esplenomegalia crónica, ruptura del bazo, entre otras.

Manifestaciones clínicas La fiebre cíclica es el sello característico de la malaria, y ocurre poco después o al tiempo de la ruptura de glóbulos rojos y liberación de merozoitos en el torrente sanguíneo. En el caso de la infección por P. Vivax o P. Ovale, este ciclo de liberación se da cada 48 horas produciendo las tercianas malignas (fiebre cada tercer día); y para el caso del P. Malarie cada 72 horas, dando lugar a las cuartanas (fiebre cada cuarto día). P. Falciparum tiende a producir fiebre continua con picos intermitentes, sobre los ciclos bien definidos de 48 horas característicos de P. Vivax y P. Ovale. La crisis malárica tiene unas características bastante definidas, tanto así que es la manifestación clínica definitiva de la enfermerdad. Luego de un pródromo de duración variable la crisis tiene 3 etapas: Escalofríos que pueden durar desde 15 minutos hasta varias horas, fiebre alta (hasta 40 grados Celsius) que puede durar varias horas y coincide con la liberación de merozoitos al torrente sanguíneo; y por último una etapa de sudoración, fatiga y resolución de la fiebre.

3.4. Diagnóstico diferencial

Las enfermedades consideradas comparten una serie de síntomas y signos que pueden hacer difícil el diagnóstico. Sin embargo , de acuerdo a la revisión sistemática de la literatura[44] [6] [45], y los conceptos del personal especializado del Hospital Infantil Napoleón Franco Pareja - Casa Del Niño, se construyó una lista preliminar de síntomas y signos para el diagnóstico diferencial del dengue, leptospirosis y malaria. Esta lista de signos y síntomas es el punto de partida para la creación del conjunto de datos utilizado en el entrenamiento de los clasificadores neuronales desarrollados. Dicha lista preliminar

TABLA 3.3. Algunos posibles signos de la Malaria [3]

Tipo	Signo
Clínicos	Hiperpirexia
	Fiebre cada 3 o 4 días
	Escalofríos
	Sudoración
	Fatiga
	Vómitos repetidos
	Diaforesis
	No hay adenopatías
	Esplenomegalia
	Ictericia
	Palidez
	Dificultad respiratoria
	Convulsiones
	Coma (malaria cerebral)
Paraclínicos	Anemia
	Plaquetas con tendencia a la baja
	Hiponatremia
	Hipoglicemia
	Acidosis metabólica
	Proteinuria
	Coagulación intravascular diseminada
	Infecciones concomitantes (salmonella)

se encuentra descrita en la tabla 2.4, donde se relacionan los diferentes síntomas y signos incluidos junto con la enfermedad específica para la cual son relevantes.

A pesar de que las enfermedades sujeto del estudio comparten varios síntomas, cada una tiene algunas características particulares que pueden anunciar su presencia, y el personal médico se entrena para detectar estos signos o síntomas particulares. En el caso del dengue, la presencia de leucopenia puede inclinar la balanza hacia esta enfermedad y ayudar a descartar la presencia de leptospirosis. La fiebre a repetición cada 3 o 4 días y la anemia son características de la malaria, y la elevación del CPK y la presencia de ictericia son signos que elevan la probabilidad de diagnóstico de leptospirosis. Este tipo información es crucial a la hora del diagnóstico diferencial, y deberá ser incluida dentro de las variables a considerar por los clasificadores ARTMAP.

TABLA 3.4. Signos y síntomas para el diagnóstico diferencial y su importancia relativa a cada enfermedad

Sígno o Síntoma	Dengue	Leptospirosis	Malaria
Edad			
Tipo fiebre (continua o cíclica)	x	x	x
Signo del torniquete positivo	x		
Anorexia	x		
Escalofríos		x	x
Tipo de cefalea (frontal o general)	x	x	
Dolor retroocular	x		
Artralgias	x		
Mialgias	x	x	
Lugar de exantema (si lo hay)	x	x	
Islas blancas en mar rojo	x		
Prurito	x		
Eritema facial	x		
Hiponatremia		x	
Presencia de infecciones concomitantes			x
Elevación de CPK (creatinafosfocinasa)		x	
Compromiso renal		x	x
Ictericia		x	
Faringitis		x	
Contacto con perros o roedores		x	
Contacto con agua llovida/estancada?		x	
Dolor en las pantorrillas		x	
Hemoglobina			x
Conteo leucocitos	x	x	
Conteo linfocitos	x	x	
Conteo neutrófilos	x	x	
Mes de consulta	x	x	x
Dias de fiebre	x	x	x
Hematocrito	x		x

CAPÍTULO 4

Construcción de un conjunto de datos sobre Dengue, Leptospirosis y Malaria

El uso efectivo de las técnicas de Machine Learning para resolver problemas de reconocimiento de patrones depende en gran medida de la disponibilidad de datos de calidad para el entrenamiento de los algoritmos a utilizar. Afortunadamente, dentro del ámbito médico, se recaba mucha información disponible a través de las historias clínicas. En la historia clínica de un paciente se guarda todo lo relacionado a la interacción médico paciente, y necesario para su correcta atención. Dentro de lo recogido en una historia clínica podemos encontrar antecedentes personales y/o familiares, enfermedades padecidas, tratamientos recibidos, complicaciones, síntomas presentados y aspectos relacionados con la eventual recuperación del paciente.

De acuerdo a lo anterior, las historias clínicas pueden ser usadas como fuente de información para la construcción de un conjunto de datos (dataset) que pueda ser usado para el entrenamiento, validación y prueba de técnicas de Machine Learning. En el caso particular de este trabajo, las historias de pacientes de Dengue, Leptospirosis y Malaria se usaron para la construcción de dicho dataset; y a continuación se describirán las características de los datos recabados. En primera instancia vale la pena resaltar que la investigación está circunscrita a los pacientes del Hospital Infantil Napoleón Franco Pareja - Casa del Niño, por lo que toda la información correspondiente a las historias clínicas viene de dicho hospital pediátrico.

4.1. Criterios de inclusión

Para la creación del conjunto de datos, se establecieron unos criterios de inclusión que determinaban si una historia era elegible para ser tenida en cuenta en la investigación e integrar el conjuntos. Dichos criterios fueron específicamente 2, y se relacionan a continuación.

TABLA 4.1. Distribución de los diagnósticos luego de la búsqueda inicial

Enfermedad	Número Historias Clínicas
Dengue	594
Leptospirosis	65
Malaria	43
Total	701

4.1.1. Horizonte temporal

Por sugerencia del personal médico del Hospital Infantil Napoleón Franco Pareja - Casa del Niño, se estableció un horizonte temporal de 10 años. De acuerdo a esto, el criterio de inclusión inicial es de que el paciente debía haber sido admitido en los últimos 10 años al hospital, presentando alguno de los 3 cuadros (Dengue, Leptospirosis o Malaria) a los que la investigación hace referencia.

4.1.2. Confirmación del diagnóstico

Los pacientes incluidos dentro del conjunto de datos debían tener diagnóstico de Dengue, Leptospirosis o Malaria confirmado por laboratorio. En caso de no contar con confirmación de laboratorio (lo cual tiende a ser frecuente en la ciudad de Cartagena debido a diferentes dificultades que se enfrentan a la hora de llevar a cabo este tipo de exámenes de laboratorio), el concepto favorable de los expertos médicos del Hospital Infantil Napoleón Franco Pareja - Casa del Niño era razón suficiente para la inclusión.

4.2. Búsqueda de la información

El proceso de búsqueda constó de 2 fases: En la primera se hizo una consulta al sistema de información del hospital para recabar los números de las historias clínicas de los pacientes con diagnóstico de Dengue, Leptospirosis o Malaria, ingresados en los últimos 10 años, y dicha consulta arrojó un número preliminar de 701 historias clínicas distribuidas acorde a la tabla 4.1. En la segunda fase del proceso se tomaron los números de las historias y se hizo una búsqueda en la sección de archivo y estadísticas del hospital de las historias clínicas ya identificadas, tomando de cada historia la información correspondiente.

Luego de aplicar el segundo criterio de inclusión, y dado que los resultados de la búsqueda inicial incluyeron pacientes para los que hubo sospecha de padecer alguna de las 3 enfermedades incluidas dentro de la investigación, pero que al final se confirmó presentaron otra patología, se obtuvieron finalmente 136 historias clínicas, con diagnósticos distribuidos de acuerdo a la figura 4.1.

4.3. Variables consideradas

De acuerdo a la revisión literaria realizada y la opinión del director científico del Hospital Infantil Napoleón Franco Pareja - Casa del Niño, infectólogo pediatra Hernando

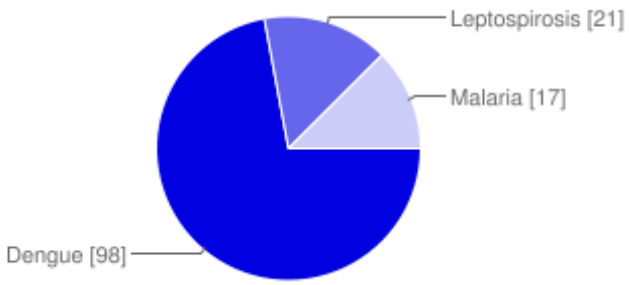

FIGURA 4.1. Distribución de los diagnósticos de las historias clínicas recabadas.

Pinzón, se determinó la estructura de un instrumento para la recolección de la información proveniente de las historias médicas seleccionadas. La estructura de dicho instrumento se relaciona en la tabla 4.2.

4.4. Descripción preliminar del dataset

Un análisis descriptivo del dataset recopilado muestra características interesantes. Inicialmente podemos destacar que se trata de un dataset donde las 3 clases se encuentran desbalanceadas entre sí. Cabe resaltar que esta distribución de diagnósticos es consistente con la experiencia del personal médico consultado. En las figuras de la 4.2 a la 4.13se muestran algunas estadísticas descriptivas relacionadas a las variables (features) del dataset.

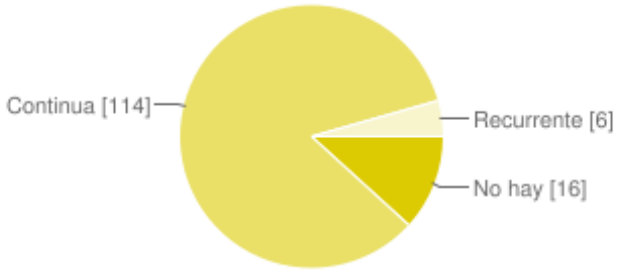

FIGURA 4.2. Distribución de la variable tipo de fiebre

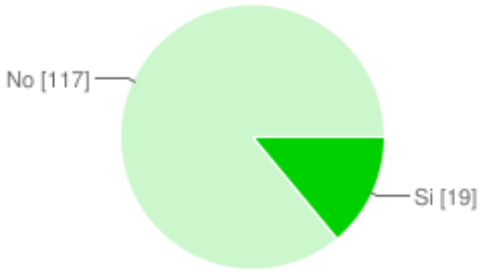

FIGURA 4.3. Distribución de la variable signo del torniquete

TABLA 4.2. Estructura del instrumento de recolección de datos

Variable	Tipo de Dato	Posibles Valores
Enfermedad	Categórico	Dengue, Malaria, Leptospirosis
Edad (Meses)	Numérico	N/A
Temperatura	Numérico	N/A
Tipo Fiebre	Categórico	Continua, Recurrente, No hay
Signo torniquete?	Categórico	Sí, No
Anorexia	Categórico	Sí,No
Escalofríos	Categórico	Sí, No
Tipo Cefalea	Categórico	Frontal, Posterior, Indeterminada, No hay
Dolor Retro-ocular?	Categórico	Sí, No
Exantema?	Categórico	Sí, No
Artralgia?	Categórico	Sí, No
Mialgia?	Categórico	Sí, No
Islas blancas en mar rojo?	Categórico	Sí, No
Prurito?	Categórico	Sí, No
Eritema Facial?	Categórico	Sí, No
Contacto con agua llovida/estancada?	Categórico	Sí, No
Infecciones oportunistas?	Categórico	Sí, No
CPK elevado?	Categórico	Sí, No
Compromiso Renal?	Categórico	Sí, No
Alteración ECG	Categórico	Bloqueo AV de primer grado, Aleteo Auricular, Fibrilación auricular, Taquicardia ventricular, Extrasístoles, No hay
Alteración Rx Tórax	Categórico	Derrame Pleural, Infiltrados, Cardiomegalia, No hay
Ictericia?	Categórico	Sí, No
Faringitis?	Categórico	Sí, No
Contacto con perros o roedores?	Categórico	Sí, No
Dolor Pantorrillas?	Categórico	Sí, No
Signo de Faget?	Categórico	Sí, No
Respiración de Kussmaul (dificultad respiratoria)	Categórico	Sí, No
Sodio (mEq/l)	Numérico	N/A
Potasio (mEq/l)	Numérico	N/A
Hemoglobina (gr/dl)	Numérico	N/A
Leucocitos (miles/mm3)	Numérico	N/A
Linfocitos(miles/mm3)	Numérico	N/A
Neutrófilos(miles/mm3)	Numérico	N/A
Mes de consulta	Categórico	[enero, diciembre]
Días de fiebre	Numérico	N/A
Hematocrito (%)	Numérico	N/A
Lugar del exantema	Categórico	Tronco, Extremidades, Generalizado, No hay

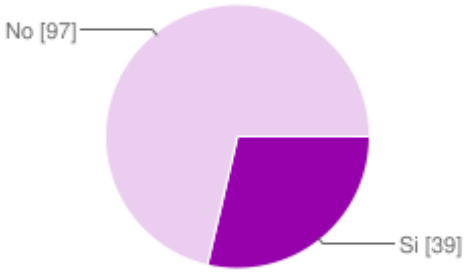

FIGURA 4.4. Distribución de la variable anorexia

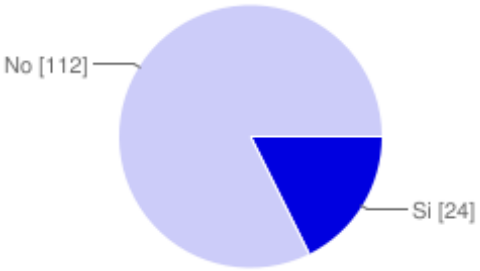

FIGURA 4.5. Distribución de la variable escalofrios

Un hallazgo interesante es la ausencia de variabilidad en algunas variables, que son consideradas en la literatura especializada como predictores importantes de dengue o leptospirosis, tales como el signo de Faget (dengue), Creatina Fosfocinasa - CPK (leptospirosis), y la falta de datos como los ionogramas (sodio y potasio), contacto con agua llovida o estancada, entre otros. De la figura 4.14 a la 4.16 se muestra el comportamiento de una de la variables más importantes según la literatura, la cantidad de leucocitos en los pacientes de cada enfermedad.

4.4.1. Datos faltantes

Dentro del dataset, se encontraron datos faltantes para algunas variables. La cantidad de datos faltantes se relaciona en la tabla 4.3. En algunos casos, ciertos pacientes registraban más de una variable faltante, incluyendo exámenes de laboratorio, lo cual es una complicación importante a la hora del entrenamiento y prueba de los clasificadores.

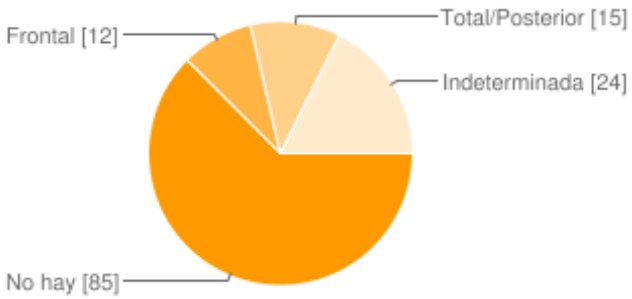

FIGURA 4.6. Distribución de la variable tipo cefalea

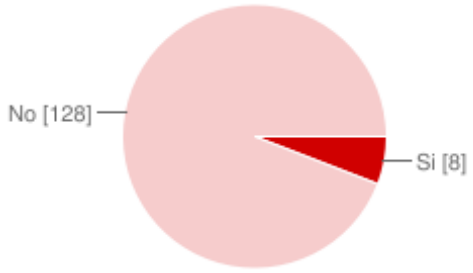

FIGURA 4.7. Distribución de la variable dolor retro-ocular

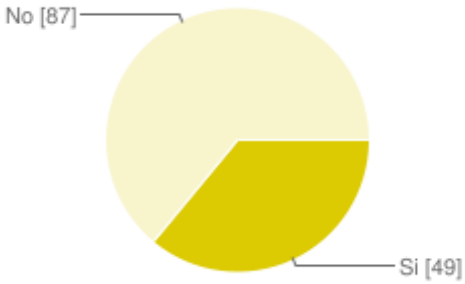

FIGURA 4.8. Distribución de la variable artralgia

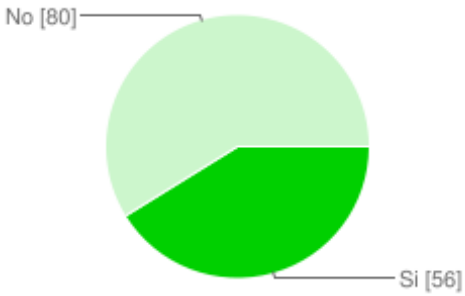

FIGURA 4.9. Distribución de la variable mialgia

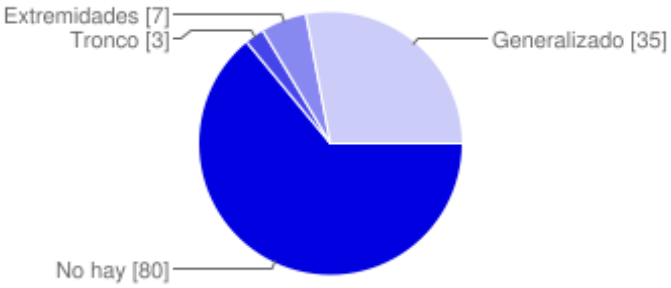

FIGURA 4.10. Distribución de la variable lugar del exantema

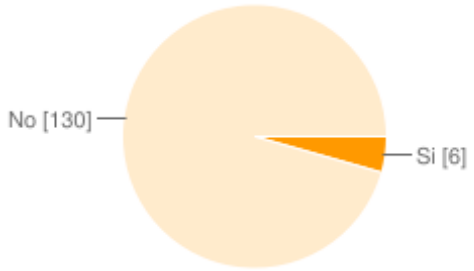

FIGURA 4.11. Distribución de la variable islas blancas en mar rojo

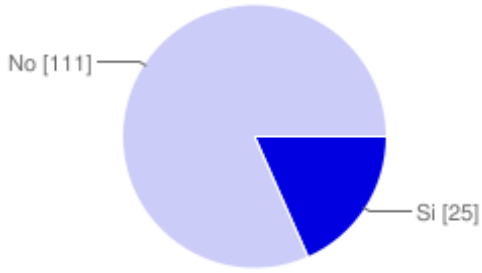

FIGURA 4.12. Distribución de la variable dolor en las pantorrillas

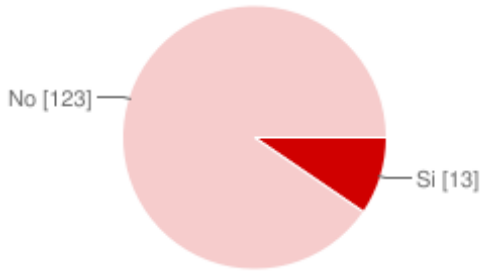

FIGURA 4.13. Distribución de la variable CPK elevado

TABLA 4.3. Cantidad de datos faltantes por variable

Variable	Cantidad
Temperatura de ingreso	7
Sodio	128
Potasio	128
Hemoglobina	5
Leucocitos	1
Linfocitos	6
Neutrófilos	6
Mes de consulta	11
Días de fiebre	11
Hematocrito	14
Lugar del exantema	11

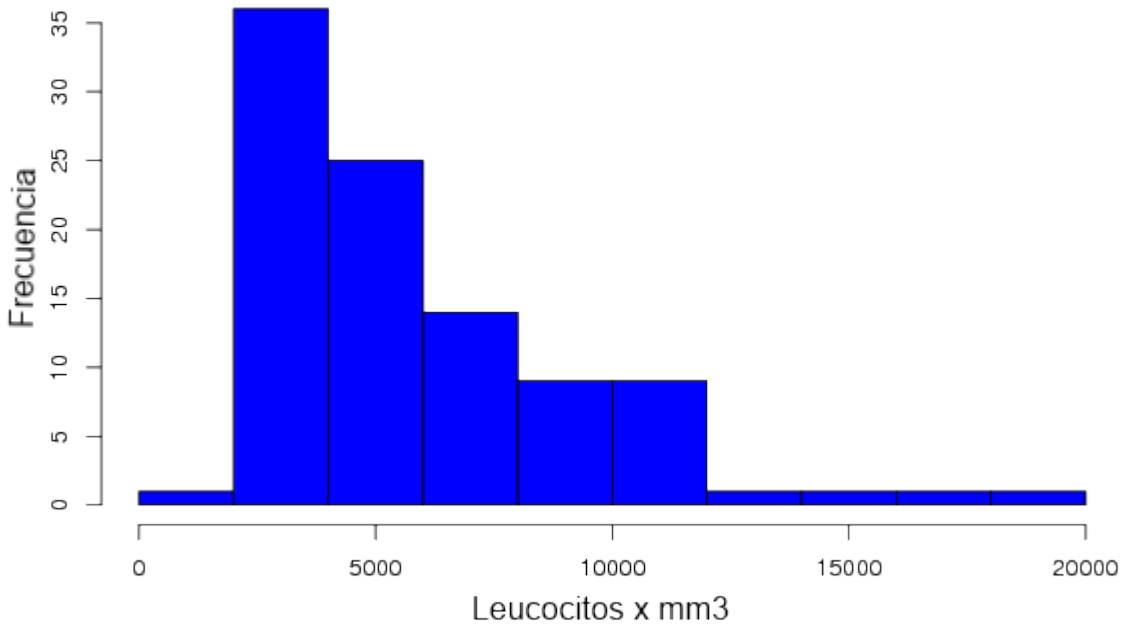

FIGURA 4.14. Distribución del número de leucocitos en los pacientes de Dengue

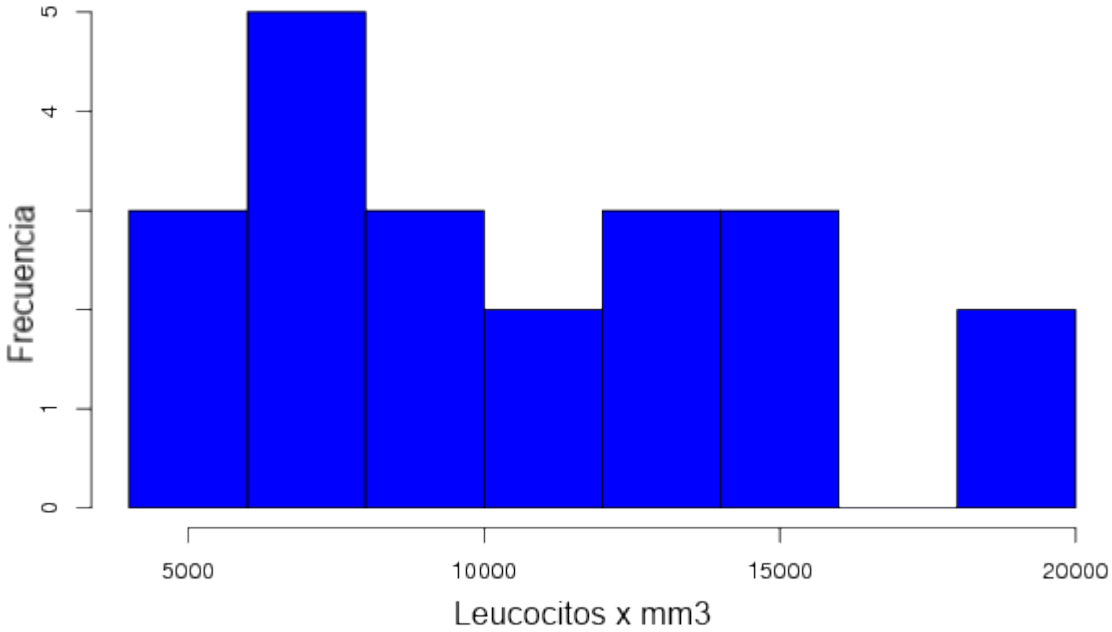

FIGURA 4.15. Distribución del número de leucocitos en los pacientes de Leptospirosis

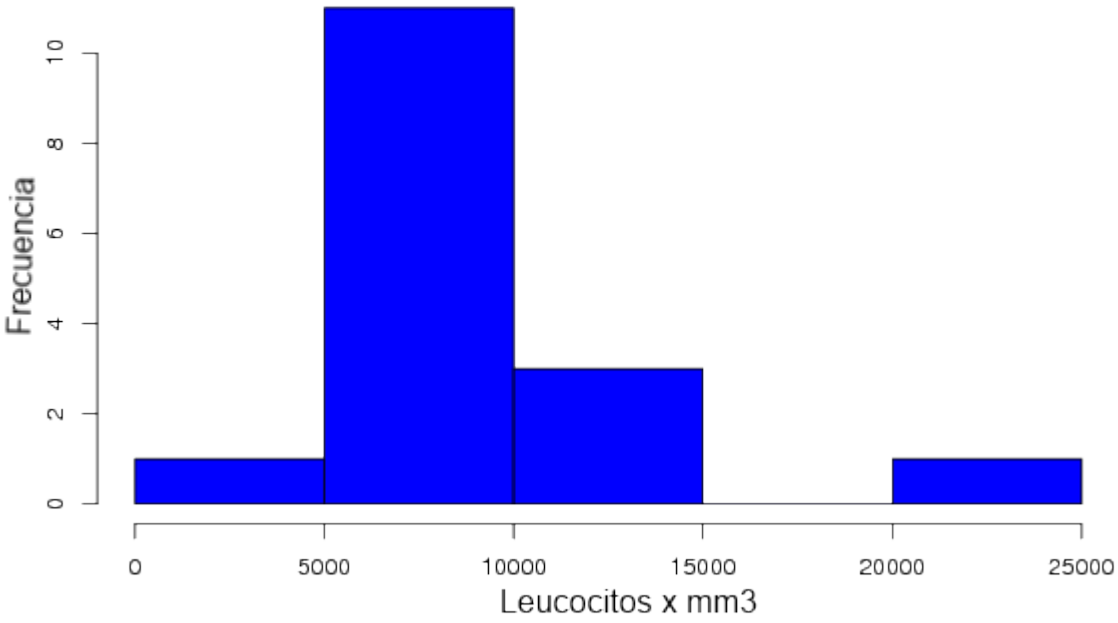

FIGURA 4.16. Distribución del número de leucocitos en los pacientes de Malaria

CAPÍTULO 5

Entrenamiento, validación y pruebas de los clasificadores ARTMAP

La característica fundamental de un algoritmo de Machine Learning es su capacidad de aprender de la experiencia, y para lograr tal efecto estos algoritmos requieren de datos de entrenamiento. Además, para la medición adecuada del desempeño de los algoritmos es necesario tener un conjunto adicional de datos diferente al conjunto de datos de entrenamiento, para darle validez estadística a los resultados.

Por otro lado, los algoritmos de Machine Learning tienen uno o más parámetros que pueden ser ajustados para variar su comportamiento y así mejorar su desempeño. La variación controlada de estos parámetros hace parte importante de todo el proceso, y para llevarla a cabo se pueden usar esquemas como la validación cruzada con K pliegues (K-Fold Cross Validation), donde el conjunto de datos de entrenamiento se fracciona en K subconjuntos disyuntos (pliegues) y el entrenamiento se hace usando K-1 pliegues, luego se prueba el desempeño sobre el pliegue restante y el proceso se repite hasta que todos los pliegues hayan sido usados para la prueba del algoritmo; y el desempeño de validación será igual al promedio de los desempeños sobre cada pliegue individual. Otro esquema de validación consiste en dividir el conjunto de datos en 3 partes: Un subconjunto de entrenamiento, un subconjunto de pruebas y adicionalmente un subconjunto específico de validación. De esta forma la variación de los parámetros se hace teniendo en cuenta el desempeño del algoritmo sobre el conjunto de validación. Usando cualquiera de las 2 estrategias, una vez determinados los valores de los parámetros que maximicen el desempeño del modelo se procede a aplicar el algoritmo de Machine Learning sobre el conjunto de pruebas, reportando estos resultados como el desempeño final del algoritmo.

Tabla 5.1. Distribución de los datos entre entrenamiento, validación y prueba

Conjunto	Dengue	Leptospirosis	Malaria	Total
Entrenamiento	51	11	8	70
Validación	24	5	4	33
Prueba	23	5	5	33

LISTADO 5.1. Script para generar los conjuntos de datos

```
'''
Created on 23/11/2011

@author: william
'''
import random
dataset = open('dataset.csv')

lines = dataset.readlines()
header = lines[0]
lines = lines[1:]
dengue = []
lepto = []
malaria = []
trainingSet = []
validationSet = []
testSet = []
# 91 casos de dengue
# 17 casos de leptospirosis
# 16 casos de malaria
# 124 casos en total
for line in lines:
    dis = line.split(';')[0]
    rep = line.replace(',','.')
    if dis == '1':
        dengue.append(rep)
    elif dis == '2':
        lepto.append(rep)
    else:
        malaria.append(rep)

#52 % para training 24 % para validation y 24 % para testing
random.shuffle(dengue);
random.shuffle(lepto);
random.shuffle(malaria);
#training
trainingSet.extend(dengue[:47])
trainingSet.extend(lepto[:9])
trainingSet.extend(malaria[:8])
print "training set size: {0} ".format(len(trainingSet))
random.shuffle(trainingSet)
trainingSet.insert(0, header)
trainingSetFile = open('trainingset.csv','w')
for line in trainingSet:
    trainingSetFile.write(line)
trainingSetFile.close()

#validation
validationSet.extend(dengue[47:69])
validationSet.extend(lepto[9:13])
validationSet.extend(malaria[8:12])
print "validation set size: {0} ".format(len(validationSet))
random.shuffle(validationSet)
validationSet.insert(0, header)
```

```
validationSetFile = open('validationset.csv','w')
for line in validationSet:
    validationSetFile.write(line)
validationSetFile.close()

#test
testSet.extend(dengue[69:])
testSet.extend(lepto[13:])
testSet.extend(malaria[12:])
print "test set size: {0} ".format(len(testSet))
random.shuffle(testSet)
testSet.insert(0, header)
testSetFile = open('testset.csv','w')
for line in testSet:
    testSetFile.write(line)
testSetFile.close()
```

En este trabajo, se empleó la segunda opción: Un conjunto especial para la validación, de tal manera que el conjunto de datos (dataset) recabado anteriormente se dividió en 3 partes de manera aleatoria, a través de muestreo estratificado con asignación proporcional. Para lo anterior se desarrolló un script en lenguaje Python. La distribución final de los datos se muestra en la tabla 5.1.

5.1. Entrenamiento

Una vez particionado el dataset, se procedió a entrenar los modelos previamente desarrollados. El modo de operación escogido para los clasificadores es Fast Learning, que asegura un aprendizaje perfecto (0 % de error) en pocas iteraciones. Para disminuir la posibilidad de sobre-entrenamiento, se llevó a cabo una sola iteración (Early Stopping). Los resultados sobre el conjunto de entrenamiento se muestran en la tabla 5.2. Los resultados

Tabla 5.2. Resultados (porcentajes de clasificación correcta) medidos sobre el conjunto de entrenamiento

Red neuronal	Dengue	Leptospirosis	Malaria
FuzzyARTMAP	96,07 %	100 %	87,5 %
ARTMAP-IC	94,12 %	90,9 %	87,5 %
DefaultARTMAP2	94,12 %	90,9 %	75 %

de las diferentes redes neuronales son similares, sin embargo en términos generales después de una iteración de entrenamiento en modo Fast-Learning, el clasificador FuzzyARTMAP lleva una ligera ventaja. Los parámetros (determinados a través de validación cruzada) usados en cada red aparecen en la tabla 5.3.

5.2. Validación cruzada

El proceso de validación se centró en encontrar en subconjunto de síntomas que maximizara el desempeño de los clasificadores, lo cual está establecido como objetivo

TABLA 5.3. Parámetros usados en cada Red Neuronal

Red neuronal	alfa	epsilon inicial	MT	Q	CAM Power
FuzzyARTMAP	0,1	0	-0,01	N/A	N/A
ARTMAP-IC	0,1	0	-0,001	9	N/A
DefaultARTMAP2	0,1	0	-0,001	N/A	4

específico de esta investigación. Es sabido que entre más complejo sea un clasificador, mayor será la cota superior de error en el conjunto de prueba, por lo que en la práctica se busca disminuir la complejidad tomando el menor número posible de variables de entrada para adelantar la clasificación. Este proceso, conocido como Feature Selection implica una búsqueda sobre el espacio de entradas, y su naturaleza claramente combinatoria le hace NP-Hard[46]. Para resolverlo, se tienen 2 aproximaciones principales: Los métodos de tipo filtro (Filter Approach) y los métodos de tipo envolvente (Wrapper Approach) [5]. En la primera aproximación se utilizan principalmente métodos estadísticos, y se observa la correlación entre los atributos disponibles y la variable a predecir; y en la segunda aproximación se utiliza un algoritmo de optimización combinatoria para buscar la combinación de variables de entrada más óptima, teniendo como evaluador de cada combinación al clasificador como tal. En este trabajo se utilizó la segunda aproximación, a través de un sistema inmune artificial (Clonalg).

Clonalg[47] es un sistema inmune artificial basado en la teoría de la Selección Clonal[48]. La respuesta del sistema inmune ante un antígeno desconocido es generar copias de células inmunes aceleradamente, de manera directamente proporcional a la afinidad entre la célula y el antígeno, y durante el proceso de copia se introducen pequeñas errores (mutaciones) que pueden traer potenciales incrementos en la afinidad. Algunas de las células que alcancen mayor afinidad se convierten en células memoria, listas para reaccionar en caso de que el antígeno vuelva a presentarse. Este principio se aplica a la optimización de funciones, donde la búsqueda se concentra en los lugares más prometedores y eventualmente se da la convergencia en un óptimo local (posiblemente global). Los agentes de búsqueda representan las células inmunes (linfocitos B o anticuerpos), y cada uno codifica una posible solución al problema de optimización de manera similar a un algoritmo genético.

Durante el proceso de validación (operando sobre el conjunto completo de predictores) se identificaron los valores de los diferentes parámetros que maximizaban el rendimiento sobre el conjunto de validación (tabla 5.3). El rendimiento de los modelos se encuentra relacionado en la tabla 5.4.

TABLA 5.4. Resultados (porcentajes de clasificación correcta) medidos sobre el conjunto de validación usando todos los predictores

Red neuronal	Dengue	Leptospirosis	Malaria
FuzzyARTMAP	91,67 % (22/24)	80 % (4/5)	50 % (2/4)
ARTMAP-IC	87,5 % (21/24)	100 % (5/5)	50 % (2/4)
DefaultARTMAP2	87,5 % (21/24)	100 % (5/5)	50 % (2/4)

Pseudocódigo 3 Pseudocódigo de CLONALG

población ← $crearLinfocitosAleatorios(númeroLinfocitos)$
mientras ¬*condicionDeParada* **hacer**
  para todo *p* en *población* **hacer**
    Afinidad(p)
  fin para todo
  $poblaciónSeleccionada \leftarrow selección(población, tamañoSelección)$
  para todo *p* en *poblacionSeleccionada* **hacer**
    $clones \leftarrow clonar(p, tasaDeClonación)$
  fin para todo
  para todo *c* en *clones* **hacer**
    Hipermutación($c, tasaMutación$)
    Afinidad(p)
  fin para todo
  $población \leftarrow selección(población, tamañoSelección)$
  $poblaciónAleatoria \leftarrow crearLinfocitosAleatorios(númeroLinfocitosAleatorios)$
  Reemplazar($población, poblaciónAleatoria$)
fin mientras
devolver *población*

Tabla 5.5. Resultados de validación para la clase dengue usando todos los predictores

Red neuronal	Precisión	Recall	F1-Score
FuzzyARTMAP	0,916	0,916	0,916
ARTMAP-IC	0,954	0,875	0,913
DefaultARTMAP2	0,954	0,875	0,913

Tabla 5.6. Resultados de validación para la clase leptospirosis usando todos los predictores

Red neuronal	Precisión	Recall	F1-Score
FuzzyARTMAP	0,8	0,8	0,8
ARTMAP-IC	0,625	1	0,769
DefaultARTMAP2	0,555	1	0,714

Tabla 5.7. Resultados de validación para la clase malaria usando todos los predictores

Red neuronal	Precisión	Recall	F1-Score
FuzzyARTMAP	0,5	0,5	0,5
ARTMAP-IC	0,666	0,5	0,571
DefaultARTMAP2	1	0,5	0,666

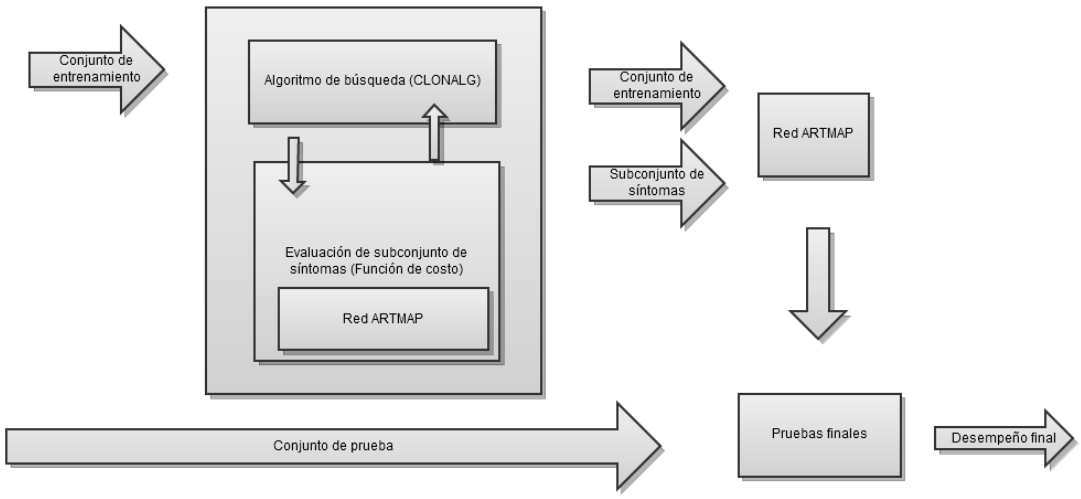

FIGURA 5.1. Proceso de feature selection. Basado en [5]

5.2.1. Procedimiento de Feature Selection

Los algoritmos de optimización buscan minimizar (o maximizar, según sea el caso) una función objetivo. Las soluciones pueden estar restringidas a cierta región factible (optimización con restricciones) o no, de acuerdo al problema. En el caso de esta investigación, la función objetivo es el desempeño del clasificador, y el espacio de búsqueda está dado por las posibles subconjuntos de síntomas que pueden ser usados como entrada; por lo que no se encontró la necesidad de plantear restricciones. Para representar lo anterior, la codificación utilizada para cada linfocito (posible solución) es binaria, utilizándose un arreglo de unos y ceros de tamaño igual al número de síntomas considerado inicialmente (cada posición representa un síntoma específico) luego de la revisión literaria. Dentro de dicho arreglo un 1 en la posición i representa que el respectivo síntoma será incluido, y un 0 representa la no inclusión de dicho síntoma. El sistema inmune realizó una busqueda en el espacio definido por la codificación escogida para identificar la permutación de unos y ceros (correspondiente a un subconjunto particular de síntomas y signos) que mejores resultados arrojase en el conjunto de validación.

La definición de la métrica usada para cuantificar el desempeño de los clasificadores es parte fundamental del proceso. En este caso particular, se escogieron la precisión, el recall y la medida F1 (F1-Score) como métricas de desempeño, debido a que estamos frente a un dataset con clases desbalanceadas (la clase Dengue es la abrumadora mayoría). La fórmula usada para calcular la medida F1 se muestra en la ecuación 5.3. Cabe destacar que la medida F1 se usa para problemas de clasificación binaria, y puesto que el problema formulado en esta investigación no es binario, se utilizó la técnica del macro-promedio (macro-averaging), propuesta en [49] para manejar problemas multi-clase.

$$precisión = \frac{verdaderospositivos}{verdaderospositivos + falsospositivos} \tag{5.1}$$

$$recall = \frac{verdaderospositivos}{verdaderospositivos + falsosnegativos} \tag{5.2}$$

$$F1 = 2 * \frac{precisión * recall}{precisión + recall} \tag{5.3}$$

Como primera medida, en el conjunto de datos se observa que varias variables carecen de variabilidad. Esto se debe principalmente a que los datos en cuestión no se encontraban en las historias clínicas analizadas. Por lo anterior se decidió dejar por fuera dichas variables. Los síntomas excluidos inicialmente por falta de variabilidad fueron presencia de infecciones oportunistas, alteración ECG, alteración, Rx Tórax, faringitis, signo Faget, dificultad respiratoria, nivel de sodio y nivel de potasio.

Luego de la ejecución del sistema inmune artificial (10 iteraciones, 30 linfocitos,

Pseudocódigo 4 Pseudocódigo de CLONALG aplicado al problema de feature selection

```
población ← crearMáscarasdeSintomasAleatorias(númeroConjuntos)
mientras ¬máximoIteraciones hacer
  para todo p en población hacer
    conjuntoEntrenamiento ← AplicarMáscara(p)
    red ← EntrenarRed(conjuntoEntrenamiento)
    CalcularMedidaF1(red, conjuntoValidación)
  fin para todo
  poblaciónSeleccionada ← selección(población, tamañoSelección)
  para todo p en poblacionSeleccionada hacer
    clones ← clonar(p, tasaDeClonación)
  fin para todo
  para todo c en clones hacer
    Hipermutación(c, tasaMutación)
    conjuntoEntrenamiento ← AplicarMáscara(p)
    red ← EntrenarRed(conjuntoEntrenamiento)
    CalcularMedidaF1(red, conjuntoValidación)
  fin para todo
  población ← selección(población, tamañoSelección)
  poblaciónAleatoria ← crearMáscarasdeSintomasAleatorias(númeroConjuntosAleatorios)

  Reemplazar(población, poblaciónAleatoria)
fin mientras
devolver población
```

300 clones por linfocito) para cada red neuronal, se obtuvieron varios subconjuntos de variables de entrada (síntomas y signos) que llevaron el desempeño de los clasificadores sobre el conjunto de validación al 100 % (porcentaje de diagnósticos correctos, precisión, recall y medida F1). Dentro de los resultados, fue particularmente llamativo un subconjunto de entradas (obtenido usando el desempeño de la red DefaultARTMAP como función de costo) que obtuvo el mejor desempeño general en los 3 clasificadores (Fuzzy, IC y Default). Dicho subconjunto de entradas consta de 19 variables que se relacionan en la tabla 5.8.

5.3. Pruebas finales

Una vez realizado el proceso de validación cruzada y de feature selection, se procedió a realizar las pruebas finales usando un conjunto de datos destinado específicamente para tal fin. Los parámetros usados para las pruebas fueron los mismos parámetros encontrados

TABLA 5.8. Subconjunto de entradas obtenido por el sistema inmune artificial

Variable	Tipo de Dato	Posibles Valores
Edad (Meses)	Numérico	N/A
Temperatura	Numérico	N/A
Tipo Fiebre	Categórico	Continua, Recurrente, No hay
Signo torniquete?	Categórico	Sí, No
Anorexia	Categórico	Sí,No
Escalofríos	Categórico	Sí, No
Dolor Retro-ocular?	Categórico	Sí, No
Localización del exantema?	Categórico	Tronco, Extremidades, Generalizado, No hay
Islas blancas en mar rojo?	Categórico	Sí, No
Prurito?	Categórico	Sí, No
Eritema Facial?	Categórico	Sí, No
Contacto con agua llovida/estancada?	Categórico	Sí, No
CPK elevado?	Categórico	Sí, No
Ictericia?	Categórico	Sí, No
Contacto con perros o roedores?	Categórico	Sí, No
Dolor Pantorrillas?	Categórico	Sí, No
Leucocitos (miles/mm3)	Numérico	N/A
Mes de consulta	Categórico	[enero, diciembre]
Hematocrito (%)	Numérico	N/A

TABLA 5.9. Resultados (porcentajes de clasificación correcta y Macro F1-Score) medidos sobre el conjunto de pruebas usando los predictores seleccionados por validación cruzada

Red neuronal	Dengue	Leptospirosis	Malaria	Macro F1-Score
FuzzyARTMAP	91,30 % (21/23)	80 % (4/5)	40 % (2/5)	0,725
ARTMAP-IC	91,30 % (21/23)	60 % (3/5)	80 % (4/5)	0,768
DefaultARTMAP2	91,30 % (21/23)	60 % (3/5)	60 % (3/5)	0,736

en el proceso de validación cruzada (tabla 5.3). Los resultados de las pruebas finales se muestran en las tablas 5.9 5.10 5.11 y 5.12.

TABLA 5.10. Resultados de prueba para la clase Dengue

Red neuronal	Precisión	Recall	F1-Score
FuzzyARTMAP	0,84	0,913	0,875
ARTMAP-IC	0,913	0,913	0,913
DefaultARTMAP2	0,84	0,913	0,875

TABLA 5.11. Resultados de prueba para la clase Leptospirosis

Red neuronal	Precisión	Recall	F1-Score
FuzzyARTMAP	0,8	0,8	0,8
ARTMAP-IC	0,75	0,6	0,666
DefaultARTMAP2	0,75	0,6	0,666

TABLA 5.12. Resultados de prueba para la clase Malaria

Red neuronal	Precisión	Recall	F1-Score
FuzzyARTMAP	0,666	0,4	0,5
ARTMAP-IC	0,666	0,8	0,7272
DefaultARTMAP2	0,75	0,6	0,666

5.3.1. Análisis de los resultados

Los resultados obtenidos por las redes en el conjunto de pruebas, muestran al modelo ARTMAP-IC por delante del resto en términos de la medida F1 y en porcentaje de clasificación correcta. En general se observa que el rendimiento de los 3 modelos en la clase Dengue es bastante bueno (91,30 % de diagnósticos correctos), lo cual se puede explicar por el hecho de que es la clase con mayor cantidad de datos para el entrenamiento. Por otro lado, para la clase Leptospirosis los modelos presentan una mayor variabilidad en el desempeño (60 % - 80 % de diagnósticos correctos), y el fenómeno se presenta en una forma más pronunciada en la clase Malaria (40 % - 60 % de diagnósticos correctos); y de igual manera la variabilidad en los resultados puede ser explicada por la limitada cantidad de datos para el entrenamiento y pruebas.

Es interesante notar que los modelos de predicción distribuida (ARTMAP-IC y Default ARTMAP) ofrecen los mejores resultados, por encima de FuzzyARTMAP. Esto es consistente con lo expresado en la literatura (la activación distribuida de la capa LTM (F2) es afín a aumentar el parámetro K en el algoritmo KNN o al uso de suavizado de Laplace en los clasificadores Bayesianos, lo que disminuye los efectos del sobre-entrenamiento). Además se puede notar que el desempeño de los modelos IC y Default es similar, a excepción de la clase Malaria, donde el modelo IC supera al modelo Default. Esto se puede atribuir a la capa F3 del modelo IC, donde las predicciones se sesgan en función del número de ejemplos en cada categoría. Por otro lado, el desempeño de los clasificadores es notable, dada la información faltante dentro del conjunto de datos.

En general la dinámica de los resultados en los diferentes conjuntos, muestra que la precisión (definida en la ecuación 5.1) se comporta de mejor manera que el recall (ecuación 5.2), es decir que en promedio, el número de falsos negativos es mayor que el número de falsos positivos. Otro aspecto a destacar es el tamaño de la capa LTM de los clasificadores ARTMAP, donde cada nodo corresponde a una categoría formada en el entrenamiento. Para los 3 clasificadores, la capa LTM tiene 13 categorías, lo cual representa una tasa de compresión del 81 %, lo que nos indica que los resultados alcanzados en la etapa de pruebas no son debidos a algún tipo de sobre-entrenamiento (over-fitting), sino que son representativos y ofrecen un estimador confiable del desempeño de los modelos para su uso a más grande escala.

Conclusiones

El Dengue, la Leptospirosis y la Malaria son zoonosis que prevalecen en nuestro medio, siendo factores que amenazan la salud pública de manera importante. Estas enfermedades, aun siendo de naturaleza muy distinta comparten varios síntomas y signos en diferentes estadios, lo que hace difícil distinguirlas entre si muchas veces, complicando el diagnóstico diferencial. Debido a lo anterior, las ayudas tecnológicas destinadas a ayudar con el diagnóstico temprano pueden desempeñar un rol importante en el cuidado de los pacientes, ya que la prognosis de estos mejora sustancialmente en la medida de que los cursos de tratamiento adecuado comiencen de manera rápida y oportuna. En este trabajo se presentó una aproximación al diagnóstico diferencial de estas fiebres hemorrágicas utilizando redes neuronales. Los resultados alcanzados muestran que el camino emprendido es promisorio, y que un clasificador ARTMAP-IC entrenado para efectuar el diagnóstico diferencial puede ser una herramienta de gran ayuda para el personal médico.

A través de esta investigación se llegó a varias conclusiones que vale la pena resaltar, puesto que revisten importancia en varios ámbitos. Dichas conclusiones se exponen a continuación.

- El diagnóstico diferencial es un problema de reconocimiento de patrones, y como tal puede ser atacado con el uso de herramientas de Machine Learning como las redes neuronales. Específicamente, las redes neuronales ARTMAP probaron ser capaces de llevar a cabo el diagnóstico diferencial de las enfermedades incluidas en esta investigación, con un grado de acierto especialmente satisfactorio en el caso del Dengue y la Malaria.
- El uso de un sistema inmune artificial permitió establecer un conjunto de síntomas y signos asociados con las 3 enfermedades consideradas en el estudio. El uso de estos síntomas y signos puede conducir a un diagnóstico diferencial acertado. Los signos y síntomas en cuestión son:
  - Edad (meses)
  - Temperatura de ingreso
  - Tipo de fiebre (continua, recurrente, ausente)
  - Presencia del signo del torniquete
  - Anorexia
  - Escalofríos
  - Dolor retro-ocular

  - Presencia de exantema
  - Signo de islas blancas en mar rojo
  - Prurito
  - Eritema facial
  - Contacto con agua llovida o estancada
  - CPK elevado
  - Ictericia
  - Contacto con perros
  - Dolor en las pantorrillas
  - Nivel de leucocitos
  - Mes de consulta
  - Nivel de hematocrito

- Debido a que los algoritmos de Machine Learning "aprenden" de la experiencia y dependen de la disponibilidad de adecuadas cantidades de datos para convertirse en clasificadores exitosos, en la medida en que se disponga de más datos acerca de pacientes con Leptospirosis (principalmente) y Malaria, la tasa de éxito de los clasificadores desarrollados aumentará. Por lo anterior, el uso de las herramientas desarrolladas es factible, y entre más sean usadas mejor será su desempeño.

- Es necesario mejorar la calidad de la información almacenada en las historias clínicas del Hospital Infantil Napoleón Franco Pareja - Casa del Niño, como producto de la interacción entre el personal médico y los pacientes admitidos por las enfermedades incluidas en esta investigación. Para diagnósticos más acertados y tempranos, es necesario buscar y registrar en las historias clínicas los síntomas y signos que la literatura especializada menciona. Por citar un ejemplo, sería de gran ayuda contar de manera consistente dentro de las historias médicas de pacientes de Dengue y/o leptospirosis con datos acerca del signo de Faget, valores séricos de la CPK, o los niveles de sodio y potasio. Por otra parte, muchas veces las pruebas de serología de pacientes altamente sospechosos para una enfermedad en particular estaban ausentes, lo que hace más difícil este tipo de investigaciones.

- En general, en nuestro país estamos en mora de aplicar de manera más amplia la inteligencia artificial en la medicina. Con los avances en computación de hoy y la ubicuidad de equipos de cómputo relativamente potentes, a través del despliegue de algoritmos de Machine Learning para la solución de este tipo de problemas de diagnóstico y muchos otros problemas relacionados a la práctica médica, se puede impactar de manera realmente sustancial la calidad de los servicios médicos, a muy bajo costo.

Trabajo futuro

Esta investigación solo marca el comienzo de un esfuerzo más general para acercar los desarrollos de la inteligencia artificial, y más específicamente el Machine Learning, a los hospitales de nuestro país. A continuación se relacionan las áreas a explorar a partir de este trabajo.

- Recabar más información y así construir un conjunto de datos de mayor tamaño y repetir los experimentos, con miras a obtener mejores resultados.
- Examinar y procesar las memorias de las redes ARTMAP desarrolladas en esta investigación para extraer conocimiento en forma de reglas difusas, y así enriquecer la comprensión de todo lo relacionado al diagnóstico diferencial del Dengue, Leptospirosis y Malaria, por parte del personal médico.
- Aplicar otros algoritmos de clasificación (SVM, Random Forest, KNN, Logistic Regression) al conjunto de datos construido y comparar los resultados obtenidos con los registrados para la familia ARTMAP en esta investigación.
- Aplicar la metodología utilizada en este trabajo en el diagnóstico de otras enfermedades en el Hospital Infantil Napoleón Franco Pareja - Casa del Niño, aprovechando la disponibilidad de datos en las historias clínicas.

Referencias

[1] Douglas (Editor) Mandell, John (Editor) Bennet, and Theodore (Autor) Tsai. *Principles and Practice of Infectious Diseases*, pages 1715–1736. Churchill Livingstone, 2000.

[2] Douglas (Editor) Mandell, John (Editor) Bennet, Jordan (Autor) Tappero, David (Autor) Ashford, and Bradley (Autor) Perkins. *Principles and Practice of Infectious Diseases*, pages 2495–2501. Churchill Livingstone, 2000.

[3] Douglas (Editor) Mandell, John (Editor) Bennet, and Donald (Autor) Kgrostad. *Principles and Practice of Infectious Diseases*, pages 2818–2831. Churchill Livingstone, 2000.

[4] G.A. Carpenter. Default artmap. *Proceedings of the International Joint Conference on Neural Networks 2003*, 2:1396–1401, 2003.

[5] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2):273–324, December 1997.

[6] Daniel H. Libraty, Khin S. A. Myint, Clinton K. Murray, Robert V. Gibbons, Mammen P. Mammen, Timothy P. Endy, Wenjun Li, David W. Vaughn, Ananda Nisalak, Siripen Kalayanarooj, Duane R. Hospenthal, Sharone Green, Alan L. Rothman, and Francis A. Ennis. A comparative study of leptospirosis and dengue in thai children. *PLoS Negl Trop Dis*, 1(3):e111, 12 2007.

[7] M.G. Brown, I.E. Vickers, R.A. Salas, and M.F. Smikle. Leptospirosis in suspected cases of dengue in Jamaica, 2002-2007. *Tropical doctor*, 40(2):92–4, April 2010.

[8] T. Ellis, A. Imrie, A.R. Katz, and P.V. Effler. Underrecognition of leptospirosis during a dengue fever outbreak in Hawaii, 2001-2002. *Vector borne and zoonotic diseases (Larchmont, N.Y.)*, 8(4):541–7, August 2008.

[9] P.N. Levett, S.L. Branch, and C.N. Edwards. Detection of dengue infection in patients investigated for leptospirosis in barbados. *Am J Trop Med Hyg*, 62(1):112–4, 2000.

[10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2:359–366, July 1989.

[11] J. Downs, R.F. Harrison, R.L. Kennedy, and S.S. Cross. Application of the fuzzy artmap neural network model to medical pattern classification tasks. *Artificial Intelligence in Medicine*, 8(4):403–428, 1996.

[12] N. Markuzon, S.A. Gaehde, A.S. Ash, G.A. Carpenter, and M.A. Moskowitz. Predicting risk of an adverse event in complex medical data sets using fuzzy artmap network. *Artificial Intelligence in Medicine: Interpreting Clinical Data. Technical Report Series*, pages 93–96, 1994.

[13] P.H. Goodman, V.G. Kaburlasos, D.D. Egbert, G.A. Carpenter, S. Grossberg, J.H. Reynolds, D.B. Rosen, and A.J. Hartz. Fuzzy artmap neural network compared to linear discriminant analysis prediction of the length of hospital stay in patients with pneumonia. *Proceedings of the IEEE 1992 Intl. Conf. on Systems, Man and Cybernetics*, 1:748–753, 1992.

[14] W. Caicedo. Diseño e implementación de una librería neuronal y de una suite didáctica para la enseñanza sobre redes neuronales, 2010.

[15] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Trans Neural Networks*, 3(5):698–713, 1992.

[16] M. Arbib. Turing machines, finite automata and neural nets. *J. ACM*, 8:467–475, October 1961.

[17] DARPA. *Neural Network Study.* 1990.

[18] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of Mathematical Biology*, 52(1-2):99–115; discussion 73–97, 1990.

[19] D. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8:1341–1390, October 1996.

[20] G.A. Carpenter and N. Markuzon. Artmap-ic and medical diagnosis: Instance counting and inconsistent cases. *Neural Networks*, pages 323–336, 1998.

[21] G. A. Carpenter and A. Tan. Rule extraction, fuzzy artmap, and medical databases. Technical report, Center for Adaptive Systems and Department of Cognitive and Neural Systems, Boston University, 1993.

[22] I. Modai, A. Israel, S. Mendel, E. Hines, and R. Weizman. Neural network based on adaptive resonance theory as compared to experts in suggesting treatment for schizophrenic and unipolar depressed in-patients. *Journal of Medical Systems*, 20:403–412, 1996. 10.1007/BF02257284.

[23] T. Faisal, F. Ibrahim, and M. Taib. Analysis of significant factors for dengue infection prognosis using the self organizing map. *Conference Proceedings of the International Conference of IEEE Engineering in Medicine and Biology Society*, 2008:5140–5143.

[24] T. Faisal, M. Taib, and F. Ibrahim. Neural network diagnostic system for dengue patients risk classification. *Journal of Medical Systems*, pages 1–16.

[25] F. Ibrahim, T. Faisal, M. Mohamad Salim, and M. Taib. Non-invasive diagnosis of risk in dengue patients using bioelectrical impedance analysis and artificial neural network. *Medical and Biological Engineering and Computing*, 48:1141–1148, 2010. 10.1007/s11517-010-0669-z.

[26] B. Andrade, A. Reis-Filho, A. Barros, S. Souza-Neto, L. Nogueira, K. Fukutani, E. Camargo, L. Camargo, A. Barral, A. Duarte, and M. Barral-Netto. Towards a precise test for malaria diagnosis in the brazilian amazon: comparison among field microscopy, a rapid diagnostic test, nested pcr, and a computational expert system based on artificial neural networks. *Malaria Journal*, 9(1):117, 2010.

[27] A. Barros, A. Duarte, M. Netto, and B. Andrade. Artificial neural networks and bayesian networks as supportting tools for diagnosis of asymptomatic malaria.

[28] G.A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37(1):54–115, 1987.

[29] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 10.1007/BF00337288.

[30] G.A. Carpenter and S. Grossberg. Art 2: self-organization of stable category recognition codes for analog input patterns. *Appl. Opt.*, 26(23):4919–4930, Dec 1987.

[31] G.A. Carpenter and S. Grossberg. Art3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3(2):129–152, 1990.

[32] G.A. Carpenter, S. Grossberg, and D.B. Rosen. Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Netw.*, 4:759–771, November 1991.

[33] L.A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[34] B. Moore. Art 1 and pattern clustering. *Proceedings of the 1988 Connectionist Models Summer School by David Touretzky, Geoffrey Hinton and Terrence Sejnowski*, pages 174–185, 1989.

[35] T. Kasuba. Simplified fuzzy artmap. *AI Expert*, 8:19–25, 1993.

[36] G.A. Carpenter, B.L. Milenova, and B.W. Noeske. Distributed artmap: a neural network for fast distributed supervised learning. *Neural Networks*, 11(5):793 – 813, 1998.

[37] P.W. Frey and D.J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6:161, 1991.

[38] P. M. Murphy and D. W. Aha. Uci repository of machine learning databases. *University of California at Irvine, Department of Information and Computer Science.*, 1992.

[39] S. (Editor) Halstead. *Dengue, Tropical Medicine: Science and Practice.* Imperial College Press, 2008.

[40] A. Rudnick and TW. Lim. Dengue fever studies in malaysia. *Inst Med Res Malaysia Bull*, (23):127–197, 1986.

[41] J.C. et al Roche. Isolement de 96 souches de virus dengue 2 partir de mosquiques captures en cote-d'ivoire et haute-volta. *Ann Virol*, 134:233–244, 1983.

[42] M. Traore-Lamizana, .H Zeller, E. Monlun, M. Mondo, J.P. Hervy, F. Adam, and J.P. Digoutte. Dengue 2 outbreak in southeastern senegal during 1990: virus isolations from mosquitoes (diptera: Culicidae). *Journal of Medical Entomology*, 31(4):623–627, 1994.

[43] R. Rico-Hesse. Molecular evolution and distribution of dengue viruses type 1 and 2 in nature. *Virology*, 174(2):479 – 493, 1990.

[44] J.A. Potts and A.L. Rothman. Clinical and laboratory features that distinguish dengue from other febrile illnesses in endemic populations. *Tropical medicine international health TM IH*, 13(11):1328–1340, 2008.

[45] D. Chadwick, B. Arch, A. Wilder-Smith, and N. Paton. Distinguishing dengue fever from other infections on the basis of simple clinical and laboratory features: Application of logistic regression analysis. *Journal of Clinical Virology*, 35(2):147–153, 2 2006.

[46] E. Amaldi and V. Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theor. Comput. Sci.*, 209:237–260, December 1998.

[47] L. N. De Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002.

[48] F.M. Burnet. *The clonal selection theory of acquired immunity*. Abraham Flexner lectures. Vanderbilt University Press, 1959.

[49] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, 1999. 10.1023/A:1009982220290.

APÉNDICE

Código fuente desarrollado

Listado 2. Neurona

```
1  package org.jane.core;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.Random;
6  import org.jane.core.functions.ITransferFunction;
7
8  /**
9   * This is the core class for JANE. Represents a McCulloch−Pitts Neuron with bias
10  * @author william
11  *
12  */
13 public class Neuron {
14      public List<Float> weights;
15      protected List<Float> inputs;
16      private Float bias;
17      protected Float output;
18      private Float sensitivity;
19      private Float sum;
20      //public int numberOfInputs;
21      private ITransferFunction transferFunctionObject;
22      /**
23       *
24       * This constructor takes a object implementing
25       * ITransferFunction representing a transfer function for this neuron.
26       */
27      public Neuron(ITransferFunction transferFunctionObject) {
28          this.transferFunctionObject = transferFunctionObject;
29          //this.numberOfInputs = numberOfInputs;
30          //this.initializeWeights(this.inputs.size());
31          this.bias = 0.0f;
32          this.sensitivity = 0.0f;
33          this.inputs = new ArrayList<Float>();
34      }
35      /**
36       *
```

```
37      * This method initializes neuron's weights randomly.
38      * @param numberOfWeights int representing number of weights for this neuron.
39      */
40     public void initializeWeights(int numberOfWeights) {
41
42         Random random = new Random();
43         this.weights = new ArrayList<Float>();
44         for (int i = 0; i < numberOfWeights; i++) {
45             this.weights.add(random.nextFloat());
46         }
47         //System.out.println("Pesos iniciales");
48         //System.out.println(this.weights);
49         //System.out.println("****************");
50          /* 1,-1,-1/0
51             1,1,-1/1
52             1,-1,-1/0
53             1,1,-1/1
54
55         this.weights = new ArrayList<Float>();
56         this.weights.add(0.5f);
57         this.weights.add(-1.0f);
58         this.weights.add(-0.5f);
59         */
60
61     }
62     /**
63 *
64 * This method performs a summation of weighted inputs and bias and stores the result.
65 @return a weighted sum of inputs and bias
66 */
67     protected float sum() {
68         if (this.weights == null)
69             this.initializeWeights(this.inputs.size());
70         float s = 0;
71         int index = 0;
72         for(Float input : this.inputs){
73             s += input*this.weights.get(index);
74             //System.out.println(input+"*"+this.weights.get(index));
75             index++;
76
77         }
78         s += this.bias;
79         this.sum = s;
80         return s;
81     }
82     /**
83      *
84      * This method takes the input, performs a weighted sum and computes
85      * it using neuron's transfer function.
86      * @return neuron output for the current input.
87      */
88     public float transfer() {
89         this.output = this.transferFunctionObject.compute(this.sum());
90         return this.output;
91     }
```

```
92
93      public Float getBias() {
94          return bias;
95      }
96
97      public void setBias(Float bias) {
98          this.bias = bias;
99      }
100
101     public List<Float> getWeights() {
102         return weights;
103     }
104
105     public void setWeights(List<Float> weights) {
106         this.weights = weights;
107     }
108
109     public List<Float> getInputs() {
110         return inputs;
111     }
112
113     public void setInputs(List<Float> inputs) {
114         this.inputs = inputs;
115
116     }
117     public Float getOutput() {
118         return this.output;
119     }
120     public Float getSensitivity() {
121         return this.sensitivity;
122     }
123     public void setSensitivity(Float sensitivity) {
124         this.sensitivity = sensitivity;
125     }
126 /**
127  *
128  * @return an object (Instance of ITransferFunction) representing neuron's
129  transfer function object. */
130     public ITransferFunction getTransferFunctionObject() {
131         return transferFunctionObject;
132     }
133
134     public Float getSum() {
135         return sum;
136     }
137 }
```

Listado 3. Neurona ART

```
1   package org.jane.core;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import org.jane.core.functions.ITransferFunction;
6
7  /**
```

```
8   *
9   * @author william
10  */
11  public class ART_LTMNeuron extends Neuron{
12      public List<Float> outputWeights;
13      private boolean inhibited;
14      private float l;
15      protected final int mapSize;
16      public ART_LTMNeuron(ITransferFunction transferFunctionObject, int mapSize) {
17          super(transferFunctionObject);
18          this.inhibited = false;
19          this.mapSize = mapSize;
20      }
21
22      @Override
23      public void initializeWeights(int numberOfWeights) {
24          this.weights = new ArrayList<Float>(numberOfWeights);
25          this.outputWeights = new ArrayList<Float>(numberOfWeights);
26          for (int i = 0; i < numberOfWeights; i++) {
27              this.weights.add(1/(l-1+numberOfWeights));
28              this.outputWeights.add(1f);
29          }
30      }
31
32      public List<Float> transferBack() {
33          List<Float> activation = new ArrayList<Float>();
34          for (Float weight : this.outputWeights) {
35              activation.add(weight*this.output);
36          }
37          return activation;
38      }
39
40      public boolean isInhibited() {
41          return inhibited;
42      }
43
44      public void setInhibited(boolean inhibited) {
45          this.inhibited = inhibited;
46      }
47
48      public List<Float> getOutputWeights() {
49          return outputWeights;
50      }
51
52      public void setOutputWeights(List<Float> outputWeights) {
53          this.outputWeights = outputWeights;
54      }
55
56      public float getL() {
57          return l;
58      }
59
60      public void setL(float l) {
61          this.l = l;
62      }
```

```
}
```

LISTADO 4. Neurona Fuzzy ART

```
package org.jane.core;

import java.util.ArrayList;
import java.util.List;
import org.jane.core.functions.PureLinFunction;
import org.jane.utils.VectorUtils;

/**
 *
 * @author william
 */
public class FuzzyARTNeuron extends ART_LTMNeuron {

    protected float alpha;
    private int M;

    public FuzzyARTNeuron(int mapSize) {
        super(new PureLinFunction(), mapSize);

    }

    @Override
    protected float sum() {
        if (this.weights == null) {
            this.initializeWeights(this.inputs.size());
            this.M = weights.size()/2;
            return alpha*(M);
        }
        float norm = VectorUtils.getL1Norm(VectorUtils.fuzzyAND(inputs, this.weights));

        float weightNorm = VectorUtils.getL1Norm(weights);
        float result = norm + (1-alpha)*(M - weightNorm);
        return result;
        //return norm / (this.alpha + VectorUtils.getL1Norm(weights));
    }

    @Override
    public void initializeWeights(int numberOfWeights) {
        this.weights = new ArrayList<Float>(numberOfWeights);
        //this.outputWeights = new ArrayList<Float>(numberOfWeights);
        for (int i = 0; i < numberOfWeights; i++) {
            this.weights.add(1f);
            //this.outputWeights.add(1f);
        }
        /*for (int i = 0; i < mapSize; i++) {
        this.outputWeights.add(1f);
        }*/
    }

    @Override
    public List<Float> getOutputWeights() {
        return weights;
```

```
53      }
54
55      @Override
56      public void setOutputWeights(List<Float> outputWeights) {
57          this.weights = outputWeights;
58      }
59
60      public List<Float> getMapWeights() {
61          return outputWeights;
62      }
63
64      public void setMapWeights(List<Float> weights) {
65          this.outputWeights = weights;
66      }
67
68      @Override
69      public void setL(float l) {
70          this.alpha = l;
71      }
72  }
```

Listado 5. Neurona ART IC

```
1   package org.jane.core;
2
3   import org.jane.utils.VectorUtils;
4
5   /**
6    *
7    * @author william
8    */
9   public class ART_ICNeuron extends FuzzyARTNeuron{
10      private int instanceCounter;
11      private float distributedOutput;
12      private int M;
13      public ART_ICNeuron(int mapSize) {
14          super(mapSize);
15          instanceCounter = 0;
16          distributedOutput = 0;
17      }
18
19      /*
20      @Override
21      protected float sum(){
22          if (this.weights == null){
23              this.initializeWeights(this.inputs.size());
24              this.M = weights.size()/2;
25              return alpha*(M);
26          }
27          //int M = weights.size()/2;
28          float weightNorm = VectorUtils.getL1Norm(weights);
29          float norm = VectorUtils.getL1Norm(VectorUtils.fuzzyAND(inputs, this.weights));
30          float result = norm + (1-alpha)*(M - weightNorm);
31          return result;
32         //return norm/(this.alpha + VectorUtils.getL1Norm(weights));
33      }
```

```
34          *
35          */
36      public void incrementInstanceCount(){
37          this.instanceCounter++;
38      }
39      public int getInstanceCount(){
40          return instanceCounter;
41      }
42
43      public float getDistributedOutput() {
44          return distributedOutput;
45      }
46
47      public void setDistributedOutput(float distributedOutput) {
48          this.distributedOutput = distributedOutput;
49      }
50  }
```

Listado 6. Capa LTM ART

```
1   package org.jane.core.layers;
2
3   import java.util.LinkedList;
4   import java.util.List;
5   import org.jane.core.ART_LTMNeuron;
6   import org.jane.core.Neuron;
7   import org.jane.core.TransferFunctions;
8   import org.jane.core.functions.ITransferFunction;
9   import org.jane.utils.ARTClusteringPolicy;
10
11  /**
12   *
13   * @author william
14   */
15  public class ART_LTMLayer extends AbstractNeuronLayer {
16
17      List<Neuron> layer;
18      private int winnerIndex;
19      protected float l;
20      protected final ARTClusteringPolicy policy;
21      protected final int mapSize;
22      private boolean newNode;
23
24      public ART_LTMLayer(int numberOfNeurons, float l,
25          ARTClusteringPolicy policy, int mapSize) {
26          super(numberOfNeurons, TransferFunctions.POSLIN);
27          this.l = l;
28          this.policy = policy;
29          this.mapSize = mapSize;
30      }
31
32      public void restoreLayer() {
33          for (int i = 0; i < layer.size(); i++) {
34              //((ART_LTMNeuron)layer.get(i)).setInhibited(false);
35              this.getNeuron(i).setInhibited(false);
36          }
```

```
37      }
38
39      @Override
40      protected Neuron injectNeuron(ITransferFunction f) {
41          ART_LTMNeuron neuron = new ART_LTMNeuron(f, mapSize);
42
43          return neuron;
44      }
45
46      @Override
47      public void addNeuron(Neuron neuron) {
48          if (this.layer == null) {
49              this.layer = new LinkedList<Neuron>();
50          }else{
51              this.size++;
52          }
53          this.layer.add((ART_LTMNeuron) neuron);
54      }
55
56      @Override
57      public List<Float> process(List<Float> inputs) {
58          this.newNode = false;
59          float max = -1;
60          float activation;
61          int index = -1;
62          List<Float> pattern = null;
63          for (int i = 0; i < this.layer.size(); i++) {
64              ART_LTMNeuron neuron = this.getNeuron(i);
65              neuron.setInputs(inputs);
66              neuron.setL(this.l);
67              activation = neuron.transfer();
68              if (activation > max && !checkNeuronInhibition(neuron)) {
69                  index = i;
70                  max = activation;
71                  pattern = neuron.getOutputWeights();
72              }
73          }
74
75          if (pattern == null && this.policy == ARTClusteringPolicy
76                  .VARIABLE_NUMBER_OF_CLUSTERS) {
77              this.addNeuron(this.injectNeuron(null));
78              this.size++;
79              index = this.size - 1;
80              this.getNeuron(index).setInputs(inputs);
81              this.getNeuron(index).setL(this.l);
82              this.getNeuron(index).transfer();
83              pattern = this.getNeuron(index).getOutputWeights();
84              this.newNode = true;
85          }
86          this.setWinnerIndex(index);
87          return pattern;
88      }
89
90      protected void setWinnerIndex(int index) {
91          this.winnerIndex = index;
```

```
92      }
93      protected boolean checkNeuronInhibition(ART_LTMNeuron neuron) {
94          return neuron.isInhibited();
95      }
96
97      public boolean isNewNode() {
98          return newNode;
99      }
100
101     @Override
102     public ART_LTMNeuron getNeuron(int index) {
103         return (ART_LTMNeuron) this.layer.get(index);
104     }
105
106     @Override
107     public void setNeuron(int index, Neuron neuron) {
108         throw new UnsupportedOperationException("Not supported yet.");
109     }
110
111     @Override
112     public Neuron getNeuron(int rowIndex, int columnIndex) {
113         throw new UnsupportedOperationException("Not supported yet.");
114     }
115
116     @Override
117     public void setLayerWeights(List<List<Float>> weights) {
118         throw new UnsupportedOperationException("Not supported yet.");
119     }
120
121     @Override
122     public void setLayerBiases(List<Float> biases) {
123         throw new UnsupportedOperationException("Not supported yet.");
124     }
125
126     @Override
127     public void setLayerBiases(Float bias) {
128         for (int i = 0; i < this.layer.size(); i++) {
129             this.layer.get(i).setBias(bias);
130         }
131     }
132
133     public int getWinnerIndex() {
134         return winnerIndex;
135     }
136 }
```

LISTADO 7. Capa LTM FuzzyART

```
1   package org.jane.core.layers;
2
3   import java.util.ArrayList;
4   import org.jane.core.FuzzyARTNeuron;
5   import org.jane.core.Neuron;
6   import org.jane.core.functions.ITransferFunction;
7   import org.jane.utils.ARTClusteringPolicy;
8
```

```
9  /**
10  *
11  * @author william
12  */
13 public class FuzzyART_LTMLayer extends ART_LTMLayer{
14
15     public FuzzyART_LTMLayer(int numberOfNeurons, float l,
16         ARTClusteringPolicy policy, int mapSize){
17         super(numberOfNeurons, l, policy,mapSize);
18     }
19
20     @Override
21      protected Neuron injectNeuron(ITransferFunction f){
22         FuzzyARTNeuron neuron = new FuzzyARTNeuron(mapSize);
23         //neuron.initializeWeights(layer.get(0).weights.size());
24         return neuron;
25     }
26
27     @Override
28     public void addNeuron(Neuron neuron) {
29         if (this.layer == null) {
30             this.layer = new ArrayList<Neuron>(this.size);
31         }
32         this.layer.add((FuzzyARTNeuron) neuron);
33     }
34     @Override
35     public FuzzyARTNeuron getNeuron(int index) {
36         return (FuzzyARTNeuron)this.layer.get(index);
37     }
38
39 }
```

Listado 8. Capa LTM DefaultART

```
1  package org.jane.core.layers;
2
3
4  import java.util.ArrayList;
5  import java.util.Collections;
6  import java.util.List;
7  import org.jane.core.ART_ICNeuron;
8  import org.jane.core.ART_LTMNeuron;
9  import org.jane.core.Neuron;
10 import org.jane.core.functions.ITransferFunction;
11 import org.jane.utils.ARTClusteringPolicy;
12
13 /**
14  *
15  * @author william
16  */
17 public class DefaultART_LTMLayer extends FuzzyART_LTMLayer {
18     private boolean inhibitionDisabled = false;
19
20     public DefaultART_LTMLayer(int numberOfNeurons, float l,
21       ARTClusteringPolicy policy, int mapSize) {
22         super(numberOfNeurons, l, policy, mapSize);
```

```
23      }
24
25      @Override
26      protected Neuron injectNeuron(ITransferFunction f) {
27          ART_ICNeuron neuron = new ART_ICNeuron(mapSize);
28          //neuron.initializeWeights(layer.get(0).weights.size());
29          return neuron;
30      }
31
32      @Override
33      public void addNeuron(Neuron neuron) {
34          if (this.layer == null) {
35              this.layer = new ArrayList<Neuron>(this.size);
36          }
37          this.layer.add((ART_ICNeuron) neuron);
38      }
39
40      @Override
41      protected void setWinnerIndex(int index) {
42          if (!this.inhibitionDisabled)
43              super.setWinnerIndex(index);
44      }
45
46      @Override
47      protected boolean checkNeuronInhibition(ART_LTMNeuron neuron) {
48          return (this.inhibitionDisabled)?false:neuron.isInhibited();
49      }
50
51      public boolean isInhibitionDisabled() {
52          return inhibitionDisabled;
53      }
54
55      public void setInhibitionDisabled(boolean inhibitionDisabled) {
56          this.inhibitionDisabled = inhibitionDisabled;
57      }
58
59      protected List<Float> IG_CAMRule(int power) {
60          //List<Float> result = new ArrayList<Float>();
61          //Increased Gradient Content Addressable Memory (IG-CAM) Rule
62          int M = this.getNeuron(0).getOutputWeights().size() / 2;
63          float alpha = this.l;
64          List<Neuron> lambda = new ArrayList<Neuron>();
65          List<Neuron> lambdaPrime = new ArrayList<Neuron>();
66          Neuron neuron = null;
67          for (int i = 0; i < this.layer.size(); i++) {
68              neuron = this.getNeuron(i);
69              //if (M == neuron.getOutput()) {
70              if ((M - neuron.getOutput()) < 0.000001) { //floating point equals
71                  lambdaPrime.add(neuron);
72              } else {
73                  //if (neuron.getOutput() > (alpha * M)) {
74                  if (neuron.getOutput() > (alpha * M - 0.000001)) {
75                  //compensating for floating point inaccuracies
76                      lambda.add(neuron);
77                  }
```

```
78              }
79
80          }
81          float activation;
82          ART_ICNeuron n = null;
83          //float countingTotal = 0;
84          if (lambdaPrime.isEmpty()) {
85              float total = 0;
86              for (int i = 0; i < lambda.size(); i++) {
87                  total += Math.pow((1f / (M - lambda.get(i).getOutput())), power);
88              }
89
90              for (int i = 0; i < lambda.size(); i++) {
91                  n = (ART_ICNeuron) lambda.get(i);
92                  activation = (float) (Math.pow(1f / (M - lambda.get(i).getOutput()), power) / total);
93                  //Instance Count biasing
94                  n.setDistributedOutput(activation);
95                  //countingTotal += n.getDistributedOutput();
96              }
97
98          } else {
99              for (int i = 0; i < lambdaPrime.size(); i++) {
100                 n = (ART_ICNeuron) lambdaPrime.get(i);
101                 activation = 1f / lambdaPrime.size();
102                 n.setDistributedOutput(activation);
103                 //countingTotal += n.getDistributedOutput();
104             }
105         }
106         //n = null;
107         //int pos;
108         List<Neuron> actualNodes = (lambdaPrime.isEmpty()) ? lambda : lambdaPrime;
109         /*for (int i = 0; i < actualNodes.size(); i++) {
110         n = (ART_ICNeuron) lambda.get(i);
111
112         }
113          *
114          */
115
116         //output = (ArrayList<Float>)Collections.
117                 nCopies(this.getNeuron(0).getMapWeights().size(), 0f);
118         /*for (int i = 0; i < this.getNeuron(0).getMapWeights().size(); i++) {
119         output.add(0f);
120         }
121          * */
122
123         return calculateFinalOutput(actualNodes);
124     }
125
126     protected List<Float> calculateFinalOutput(List<Neuron> actualNodes) {
127         ART_ICNeuron n = null;
128         int pos;
129         List<Float> output = new ArrayList<Float>(Collections.
130         nCopies(this.getNeuron(0).getMapWeights().size(), 0f));
131         for (int i = 0; i < actualNodes.size(); i++) {
132             n = ((ART_ICNeuron) actualNodes.get(i));
```

```
133                 //n.setDistributedOutput(n.getDistributedOutput()/countingTotal);
134                 pos = n.getMapWeights().indexOf(1f);
135                 output.set(pos, (output.get(pos) == 0f) ? n.
136                 getDistributedOutput() : output.get(pos)
137                 + n.getDistributedOutput());
138             }
139             return output;
140         }
141 
142         public List<Float> getDistributedActivation(int Q) {
143             return this.IG_CAMRule(Q);
144         }
145 }
```

Listado 9. Capa LTM ART IC

```
1   package org.jane.core.layers;
2 
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.List;
6 import org.jane.core.ART_ICNeuron;
7 import org.jane.core.Neuron;
8 import org.jane.core.functions.ITransferFunction;
9 import org.jane.utils.ARTClusteringPolicy;
10 
11 /**
12  *
13  * @author william
14  */
15 public class FuzzyART_LTM_ICLayer extends DefaultART_LTMLayer {
16 
17     private int CAM = 0;
18 
19     public FuzzyART_LTM_ICLayer(int numberOfNeurons,
20         float l, ARTClusteringPolicy policy, int mapSize, int CAM) {
21         super(numberOfNeurons, l, policy, mapSize);
22 
23     }
24 
25     @Override
26     protected Neuron injectNeuron(ITransferFunction f) {
27         ART_ICNeuron neuron = new ART_ICNeuron(mapSize);
28         //neuron.initializeWeights(layer.get(0).weights.size());
29         return neuron;
30     }
31 
32     @Override
33     public void addNeuron(Neuron neuron) {
34         if (this.layer == null) {
35             this.layer = new ArrayList<Neuron>(this.size);
36         }
37         this.layer.add((ART_ICNeuron) neuron);
38     }
39 
40     @Override
```

```
41      protected List<Float> calculateFinalOutput(List<Neuron> actualNodes) {
42          float countingTotal = 0;
43          int pos;
44          ART_ICNeuron n = null;
45          for (int i = 0; i < actualNodes.size(); i++) {
46              n = ((ART_ICNeuron) actualNodes.get(i));
47              countingTotal += n.getDistributedOutput() * n.getInstanceCount();
48          }
49          List<Float> output = new ArrayList<Float>(Collections
50                  .nCopies(this.getNeuron(0).getMapWeights().size(), 0f));
51          float out = 0;
52          for (int i = 0; i < actualNodes.size(); i++) {
53              n = ((ART_ICNeuron) actualNodes.get(i));
54              //n.setDistributedOutput(n.getDistributedOutput()/countingTotal);
55              pos = n.getMapWeights().indexOf(1f);
56              out = (n.getDistributedOutput() * n.getInstanceCount()) / countingTotal;
57              output.set(pos, (output.get(pos) == 0f) ? out : output.get(pos) + out);
58          }
59          return output;
60      }
61
62      @Override
63      public List<Float> process(List<Float> inputs) {
64          List<Float> output = super.process(inputs);
65          ((ART_ICNeuron) this.getNeuron(this.getWinnerIndex()))
66                  .incrementInstanceCount();
67          return output;
68      }
69
70      @Override
71      public List<Float> getDistributedActivation(int Q) {
72          if (CAM == 1)
73              return this.QmaxRule(Q);
74          else
75              return this.IG_CAMRule(Q);
76      }
77
78
79      private List<Float> QmaxRule(int Q) {
80
81          List<Integer> winners = new ArrayList<Integer>();
82          //int n = 0;
83          Float max;
84          int index;
85          Float output;
86          float total = 0;
87          //Q-MAX Rule
88          while (winners.size() < Q) {
89              max = 0f;
90              index = -1;
91              for (int i = 0; i < this.layer.size(); i++) {
92                  output = this.layer.get(i).getOutput();
93                  if ((output > max) && !winners.contains(i)) {
94                      max = output;
95                      index = i;
```

```
96                  }
97              }
98              winners.add(index);
99              total += this.getNeuron(index).getOutput()
100                 * ((ART_ICNeuron) this.getNeuron(index)).getInstanceCount();
101         }
102
103         List<Float> result = new ArrayList<Float>(Collections
104                 .nCopies(this.getNeuron(0).getMapWeights().size(), 0f));
105         for (int i = 0; i < winners.size(); i++) {
106             ART_ICNeuron neuron = (ART_ICNeuron) this.layer.get(winners.get(i));
107             int pos = neuron.getMapWeights().indexOf(1f);
108             //Normalized Instance−Counting Biasing
109             Float activation = neuron.getInstanceCount() * neuron.getOutput() / total;
110             result.set(pos, (result.get(pos) == 0f) ? activation : result.get(pos) + activation);
111         }
112         return result;
113
114     }
115 }
```

Listado 10. ART1

```
1  package org.jane.core.networks;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import org.jane.core.ART_LTMNeuron;
6  import org.jane.core.Neuron;
7  import org.jane.core.layers.ART_LTMLayer;
8  import org.jane.utils.ARTClusteringPolicy;
9  import org.jane.utils.VectorUtils;
10
11 /**
12  *
13  * @author william
14  */
15 public class ART1 extends AbstractUnsupervisedNeuralNetwork {
16
17     protected List<Float> STMaField;
18     protected List<Float> STMbField;
19     protected ART_LTMLayer LTMLayer;
20     protected float vigilance;
21     protected float l;
22     protected final ARTClusteringPolicy clusteringPolicy;
23     private boolean trainingMode;
24
25     public ART1(int numberOfClasses, int inputSize, float vigilance,
26         float l, ARTClusteringPolicy policy) {
27         this.clusteringPolicy = policy;
28         this.STMaField = new ArrayList<Float>(inputSize);
29         this.STMbField = new ArrayList<Float>(inputSize);
30         //this.LTMLayer = new ART_LTMLayer(numberOfClasses,l,policy);
31         this.vigilance = vigilance;
32         this.init(numberOfClasses, l);
33         trainingMode = true;
```

```
34
35      }
36
37      protected void init(int numberOfClasses, float l) {
38          this.LTMLayer = new ART_LTMLayer(numberOfClasses, l, clusteringPolicy, 0);
39          this.l = l;
40      }
41
42      @Override
43      public int getLayerSize() {
44          return this.LTMLayer.getSize();
45      }
46
47      @Override
48      public List<List<Float>> getWeightVectors() {
49          List<List<Float>> weights = new ArrayList<List<Float>>();
50          for (int i = 0; i < this.LTMLayer.getSize(); i++) {
51              ART_LTMNeuron neuron = this.LTMLayer.getNeuron(i);
52              weights.add(VectorUtils.floatListDeepCopy(neuron.getOutputWeights()));
53          }
54          return weights;
55      }
56
57      @Override
58      public List<Float> processData(List<Float> inputs) {
59          this.STMaField = inputs;
60          boolean reset = true;
61          float patternNorm = 0;
62          //float inputNorm = VectorUtils.getL1Norm(this.STMaField);
63          List<Float> pattern = null;
64          if (trainingMode) {
65              while (reset) {
66                  pattern = this.LTMLayer.process(this.STMaField);
67                  if (pattern == null) {
68                      return null;
69                  }
70                  this.updateSTMbField(pattern);
71                  patternNorm = VectorUtils.getL1Norm(this.STMbField);
72                  //reset = ((patternNorm/inputNorm)<this.vigilance);
73                  reset = this.testForReset();
74                  if (reset) {
75                      this.LTMLayer.getNeuron(this.LTMLayer.getWinnerIndex()).setInhibited(true);
76                  }
77              }
78              ART_LTMNeuron winnerNeuron = this.LTMLayer.getNeuron(this.LTMLayer.getWinnerIndex());
79              this.learn(winnerNeuron, patternNorm);
80              //this.LTMLayer.restoreLayer();
81          }else{
82              this.LTMLayer.process(this.STMaField);
83          }
84          //printBottomUpWeights();
85          //System.out.println("----------------");
86          //printTopDownWeights();
87          //System.out.println("********************");
88
```

```
89          return getOutputList();
90      }
91
92      protected List<Float> getOutputList() {
93          List<Float> result = new ArrayList<Float>();
94          for (int i = 0; i < this.LTMLayer.getSize(); i++) {
95              this.LTMLayer.getNeuron(i).setInhibited(false);
96              result.add((i == this.LTMLayer.getWinnerIndex()) ? 1f : 0f);
97          }
98          return result;
99      }
100     protected void learn(Neuron n, float patternNorm) {
101         ART_LTMNeuron neuron = (ART_LTMNeuron) n;
102         for (int i = 0; i < this.STMaField.size(); i++) {
103             neuron.getWeights().set(i, (l * this.STMbField.get(i) / (l - 1 + patternNorm)));
104             neuron.setOutputWeights(VectorUtils.floatListDeepCopy(this.STMbField));
105             //winnerNeuron.getOutputWeights().set(i, this.STMbField.get(i));
106         }
107     }
108
109     void printBottomUpWeights() {
110         for (int i = 0; i < this.LTMLayer.getSize(); i++) {
111             ART_LTMNeuron n = this.LTMLayer.getNeuron(i);
112             System.out.println(n.getWeights().toString());
113         }
114     }
115
116     void printTopDownWeights() {
117         for (int i = 0; i < this.LTMLayer.getSize(); i++) {
118             ART_LTMNeuron n = this.LTMLayer.getNeuron(i);
119             System.out.println(n.getOutputWeights().toString());
120         }
121     }
122
123     protected void updateSTMbField(List<Float> pattern) {
124         List<Float> result = new ArrayList<Float>();
125         for (int i = 0; i < pattern.size(); i++) {
126             result.add(pattern.get(i) * this.STMaField.get(i));
127         }
128         this.STMbField = result;
129     }
130
131     @Override
132     public Object getNeurons() {
133         throw new UnsupportedOperationException("Not supported yet.");
134     }
135
136     public float getVigilance() {
137         return vigilance;
138     }
139
140     public void setVigilance(float vigilance) {
141         this.vigilance = vigilance;
142     }
143
```

```
144     protected boolean testForReset() {
145         return
146                 ((VectorUtils.getL1Norm(this.STMbField) /
147         VectorUtils.getL1Norm(this.STMaField)) < this.vigilance); }
148 
149     public boolean isTrainingMode() {
150         return trainingMode;
151     }
152 
153     public void setTrainingMode(boolean trainingMode) {
154         this.trainingMode = trainingMode;
155     }
156 }
```

Listado 11. FuzzyART

```
1   package org.jane.core.networks;
2 
3 import java.util.List;
4 import org.jane.core.FuzzyARTNeuron;
5 import org.jane.core.Neuron;
6 import org.jane.core.layers.FuzzyART_LTMLayer;
7 import org.jane.utils.ARTClusteringPolicy;
8 import org.jane.utils.VectorUtils;
9 
10 /**
11  *
12  * @author william
13  */
14 public class FuzzyART extends ART1{
15 
16     public FuzzyART(int numberOfClasses, int inputSize,
17         float vigilance, float alpha,ARTClusteringPolicy policy) {
18         super(numberOfClasses, inputSize, vigilance, alpha, policy);
19     }
20 
21     @Override
22     protected void init(int numberOfClasses, float l) {
23         this.LTMLayer = new FuzzyART_LTMLayer(numberOfClasses,l,this.clusteringPolicy,0);
24         this.l = l;
25     }
26 
27     @Override
28     protected void learn(Neuron neuron, float patternNorm) {
29         ((FuzzyARTNeuron)neuron).setWeights(VectorUtils.floatListDeepCopy(this.STMbField));
30     }
31 
32     @Override
33     protected void updateSTMbField(List<Float> pattern) {
34         this.STMbField = VectorUtils.fuzzyAND(pattern,this.STMaField);
35     }
36 
37 }
```

Listado 12. FuzzyARTMAP

```
1   package org.jane.core.networks;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.List;
6  import org.jane.core.FuzzyARTNeuron;
7  import org.jane.core.layers.FuzzyART_LTMLayer;
8  import org.jane.utils.ARTClusteringPolicy;
9  import org.jane.utils.VectorUtils;
10
11 /**
12  *
13  * @author william
14  */
15 public class FuzzyARTmap extends AbstractSupervisedNeuralNetwork {
16
17     protected FuzzyART ARTa;
18     private FuzzyART ARTb;
19     private float mapVigilance;
20     protected float ARTa_vigilance;
21     protected float ARTa_alpha;
22     private float ARTb_alpha;
23     private float epsilon;
24     protected List<Float> currentCategory;
25     protected int M;
26     private int outSize;
27     //private SimpleLayer map;
28     //private final ARTClusteringPolicy clusteringPolicy;
29
30
31     public FuzzyARTmap() {
32         this.ARTa_alpha = 0.1f;
33         this.ARTb_alpha = 1;
34         this.ARTa_vigilance = 0f;
35         //this.clusteringPolicy = policy;
36         this.mapVigilance = 1;
37         this.epsilon = -0.001f;
38     }
39     public FuzzyARTmap(float ARTa_alpha, float ARTb_alpha,
40         float baselineVigilance, float mapVigilance, float epsilon) {
41         this.ARTa_alpha = ARTa_alpha;
42         this.ARTb_alpha = ARTb_alpha;
43         this.ARTa_vigilance = baselineVigilance;
44         //this.clusteringPolicy = policy;
45         this.mapVigilance = mapVigilance;
46         this.epsilon = epsilon;
47     }
48
49
50     protected FuzzyART getARTa(int inSize) {
51         return new ModifiedFuzzyART(1, inSize, ARTa_vigilance, ARTa_alpha,
52                 ARTClusteringPolicy.VARIABLE_NUMBER_OF_CLUSTERS, this);
53     }
54
55     private void init(int outSize, int inSize) {
```

```
56          this.outSize = outSize;
57          this.currentCategory = new ArrayList<Float>();
58          this.M = inSize / 2;
59          this.ARTa = this.getARTa(inSize);
60          this.ARTb = new FuzzyART(outSize, outSize, 1, ARTb_alpha,
61                  ARTClusteringPolicy.FIXED_NUMBER_OF_CLUSTERS);
62          this.currentCategory = new ArrayList<Float>(outSize);
63      }
64
65      @Override
66      public List<Float> train(List<List<List<Float>>> trainingSet) {
67          //List<Float> outputA;
68          //List<Float> outputB;
69          if (this.ARTa == null) {
70              this.init(trainingSet.get(0).get(1).size(), trainingSet.get(0).get(0).size());
71          } else {
72              if (!this.ARTa.isTrainingMode()) {
73                  this.ARTa.setTrainingMode(true);
74              }
75              if (!this.ARTb.isTrainingMode()) {
76                  this.ARTb.setTrainingMode(true);
77              }
78              //this.ARTa.setVigilance(this.ARTa_vigilance);
79          }
80          for (int i = 0; i < trainingSet.size(); i++) {
81
82              //if (i % 10 == 0) {
83              //System.out.println("Caso: " + i);
84              //}
85
86
87              //this.currentCategory = this.ARTb.processData(trainingSet.get(i).get(1));
88              this.currentCategory = this.getCurrentCategory(i, trainingSet);
89              this.ARTa.processData(trainingSet.get(i).get(0));
90
91          }
92          this.ARTa.setVigilance(this.ARTa_vigilance);
93          return null;
94      }
95
96      protected List<Float> getCurrentCategory(int i, List<List<List<Float>>> trainingSet) {
97          //return this.ARTb.processData(trainingSet.get(i).get(1));
98          return trainingSet.get(i).get(1);
99      }
100
101     @Override
102     public void setLearningRate(float learningRate) {
103         throw new UnsupportedOperationException("Not supported yet.");
104     }
105
106     @Override
107     public Float getBatchError() {
108         throw new UnsupportedOperationException("Not supported yet.");
109     }
110
```

```
111     @Override
112     public List<Float> processData(List<Float> inputs) {
113         if (this.ARTa.isTrainingMode()) {
114             this.ARTa.setTrainingMode(false);
115         }
116         return this.ARTa.processData(inputs);
117     }
118 
119     public List<Float> translateCategory(List<Float> inputs) {
120         if (this.ARTb.isTrainingMode()) {
121             this.ARTb.setTrainingMode(false);
122         }
123         return this.ARTb.processData(inputs);
124     }
125 
126     @Override
127     public Object getNeurons() {
128         throw new UnsupportedOperationException("Not supported yet.");
129     }
130 
131     public int getNumberOfCategories() {
132         return this.ARTa.getLayerSize();
133     }
134 
135     protected class ModifiedFuzzyART extends FuzzyART {
136 
137         FuzzyARTmap parent;
138 
139         public ModifiedFuzzyART(int numberOfClasses, int inputSize,
140         float vigilance, float alpha,
141                 ARTClusteringPolicy policy, FuzzyARTmap parent) {
142             super(numberOfClasses, inputSize, vigilance, alpha, policy);
143         }
144 
145         @Override
146         protected void init(int numberOfClasses, float l) {
147             this.LTMLayer = new FuzzyART_LTMLayer(numberOfClasses,
148                 l, this.clusteringPolicy, FuzzyARTmap.this.outSize);
149             this.l = l;
150         }
151 
152         @Override
153         protected boolean testForReset() {
154 
155             float r = VectorUtils
156                 .getL1Norm(this.STMbField) / VectorUtils.getL1Norm(this.STMaField);
157             if (this.LTMLayer.isNewNode()) {
158                 this.vigilance = 0;
159             }
160             if (r < this.vigilance) {
161                 return true;
162             }
163             FuzzyARTNeuron neuron = ((FuzzyART_LTMLayer) this.LTMLayer)
164                 .getNeuron(this.LTMLayer.getWinnerIndex());
165             if (this.LTMLayer.isNewNode() || neuron.getMapWeights() == null) {
```

```
166                 neuron.setMapWeights(Collections.nCopies(outSize, 1f));
167             }
168             List<Float> mapActivation = VectorUtils
169                 .fuzzyAND(currentCategory, neuron.getMapWeights());
170             if (VectorUtils.getL1Norm(mapActivation) == 0) {
171                 raiseVigilance();
172                 return true;
173             }
174             neuron.setMapWeights(currentCategory);
175             return false;
176         }
177 
178 
179         protected void raiseVigilance() {
180             float ratio = VectorUtils
181                 .getL1Norm(this.STMbField) / VectorUtils.getL1Norm(this.STMaField);
182             this.setVigilance(ratio == Float.NaN ? 0 : ratio + epsilon);
183         }
184 
185         @Override
186         protected List<Float> getOutputList() {
187             this.LTMLayer.restoreLayer();
188             if (!this.isTrainingMode()) {
189                 return ((FuzzyART_LTMLayer) this.LTMLayer)
190                         .getNeuron(this.LTMLayer.getWinnerIndex()).getMapWeights();
191             } else {
192                 return null;
193             }
194         }
195     }
196 }
```

Listado 13. DefaultARTMAP

```
1   package org.jane.core.networks;
2 
3 import java.util.Collections;
4 import java.util.List;
5 import org.jane.core.layers.DefaultART_LTMLayer;
6 import org.jane.utils.ARTClusteringPolicy;
7 
8 /**
9  *
10 * @author william
11 */
12 public class DefaultARTMAP extends FuzzyARTmap {
13 
14     protected int CAMPower;
15     public static final int CLASSIC = 1;
16     public static final int DEFAULT_ARTMAP2 = 2;
17     private int networkType;
18 
19     public DefaultARTMAP(){
20         super(0.1f, 0.01f, 0f, 1, 0.1f);
21         this.CAMPower = 2;
22         this.networkType = 2;
```

```
23      }
24      public DefaultARTMAP(float ARTa_alpha, float ARTb_alpha,
25      float baselineVigilance, float mapVigilance,
26          float epsilon, int CAMPower, int networkType) {
27          super(ARTa_alpha, ARTb_alpha, baselineVigilance, mapVigilance, epsilon);
28          this.CAMPower = CAMPower;
29          this.networkType = (networkType == 1 || networkType == 2) ? networkType : 1;
30
31      }
32
33      @Override
34      protected FuzzyART getARTa(int inSize) {
35          return new DefaultART(1, inSize, ARTa_vigilance,
36                  ARTa_alpha, ARTClusteringPolicy.VARIABLE_NUMBER_OF_CLUSTERS, this);
37      }
38
39      @Override
40      protected List<Float> getCurrentCategory(int i,
41          List<List<List<Float>>> trainingSet) {
42          return trainingSet.get(i).get(1);
43      }
44
45      protected class DefaultART extends FuzzyARTmap.ModifiedFuzzyART {
46
47          public DefaultART(int numberOfClasses, int inputSize,
48                  float vigilance, float alpha, ARTClusteringPolicy policy, FuzzyARTmap
49          parent) { super(numberOfClasses, inputSize, vigilance, alpha, policy, parent);
50          }
51
52          @Override
53          protected void init(int numberOfClasses, float l) {
54              this.LTMLayer = new DefaultART_LTMLayer(numberOfClasses, l, this.clusteringPolicy, 0);
55              this.l = l;
56          }
57
58          @Override
59          protected boolean testForReset() {
60              boolean reset = super.testForReset();
61              if (networkType == DefaultARTMAP.DEFAULT_ARTMAP2 && !reset) {
62                  DefaultARTMAP dartmap = DefaultARTMAP.this;
63                  this.learn(this.LTMLayer.getNeuron(this.LTMLayer.getWinnerIndex()), 0f);
64                  ((DefaultART_LTMLayer)this.LTMLayer).setInhibitionDisabled(true);
65                  List<Float> r = dartmap.processData(STMaField);
66                  ((DefaultART_LTMLayer)this.LTMLayer).setInhibitionDisabled(false);
67                  dartmap.ARTa.setTrainingMode(true);
68                  int index = r.indexOf(Collections.max(r));
69                  int realIndex = currentCategory.indexOf(1f);
70                  if (index != realIndex) {
71                      raiseVigilance();
72                      return true;
73                  }
74              }
75              return reset;
76          }
77
```

```
78
79
80
81          @Override
82          protected List<Float> getOutputList() {
83              if (!((DefaultART_LTMLayer)this.LTMLayer).isInhibitionDisabled())
84                  this.LTMLayer.restoreLayer();
85
86              if (!this.isTrainingMode()) {
87                  return ((DefaultART_LTMLayer) this.LTMLayer)
88                          .getDistributedActivation(CAMPower);
89              } else {
90                  return null;
91              }
92              //return ((FuzzyART_LTMLayer) this.LTMLayer)
93                  .getNeuron(this.LTMLayer.getWinnerIndex()).getMapWeights();
94          }
95
96      }
97
98  }
```

Listado 14. ARTMAP IC

```
1   package org.jane.core.networks;
2
3   import java.util.List;
4   import org.jane.core.layers.FuzzyART_LTM_ICLayer;
5   import org.jane.utils.ARTClusteringPolicy;
6
7   /**
8    *
9    * @author william
10   */
11  public class ARTMAP_IC extends DefaultARTMAP {
12      //private int Q;
13
14      public static final int IG = 0;
15      public static final int QMAX = 1;
16      private int CAM;
17
18      public ARTMAP_IC(){
19          super(0.1f, 0.01f, 0f, 1, 0.1f, 9,1);
20          this.CAM = ARTMAP_IC.QMAX;
21      }
22
23      public ARTMAP_IC(float ARTa_alpha, float ARTb_alpha,
24      float baselineVigilance, float mapVigilance, float epsilon, int Q, int CAM) {
25          super(ARTa_alpha, ARTb_alpha, baselineVigilance, mapVigilance, epsilon,Q,1);
26           this.CAM = CAM;
27
28      }
29
30      @Override
31      protected FuzzyART getARTa(int inSize) {
32          return new ART_IC(1, inSize, ARTa_vigilance,
```

```
33                  ARTa_alpha, ARTClusteringPolicy.VARIABLE_NUMBER_OF_CLUSTERS, this);
34      }
35
36
37      protected class ART_IC extends DefaultARTMAP.DefaultART {
38
39          public ART_IC(int numberOfClasses, int inputSize, float vigilance,
40                  float alpha, ARTClusteringPolicy policy, FuzzyARTmap parent) {
41              super(numberOfClasses, inputSize, vigilance, alpha, policy, parent);
42          }
43
44          @Override
45          protected void init(int numberOfClasses, float l) {
46              this.LTMLayer = new FuzzyART_LTM_ICLayer(numberOfClasses,
47                  l, this.clusteringPolicy, 0,CAM);
48              this.l = l;
49          }
50
51      }
52  }
```

Listado 15. Optimizador inmune Clonalg

```
1   package org.biopt.optimizers.immune;
2
3   import java.util.Collections;
4   import java.util.List;
5
6   import javolution.util.FastList;
7
8   import org.biopt.core.AbstractOptimizer;
9   import org.biopt.core.ICostFunction;
10  import org.biopt.core.IMutationProvider;
11  import org.biopt.utils.MersenneTwisterFast;
12  import org.biopt.utils.Utils;
13
14  /**
15   *
16   * @author william
17   */
18  public class ClonalgOptimizer extends AbstractOptimizer<Lymphocyte> {
19
20      //List<Lymphocyte> limphocytes;
21      //List<List<Lymphocyte>> clones;
22
23      private int dimensions;
24      private int numberOfCells;
25      private final double d;
26      private final double beta;
27      private MersenneTwisterFast mt;
28          private IMutationProvider mutationProvider;
29
30      public ClonalgOptimizer(int dimensions, int numberOfCells,
31          double beta, double d, List<Double> lowerBound, List<Double> upperBound,
32          String functionClassName, IMutationProvider mutationProvider,
33          boolean constrainedSearch)
```

```
34      {
35          super(lowerBound, upperBound,
36                  (ICostFunction)Utils.getInstanceByReflection(functionClassName));
37                  this.setCostFunction((ICostFunction)
38                  Utils.getInstanceByReflection(functionClassName));
39          this.mutationProvider =
40                  mutationProvider; this.dimensions = dimensions;
41          this.numberOfCells = numberOfCells;
42          this.beta = beta;
43          this.d = d;
44          mt = Utils.getMTInstance();
45          this.init();
46      }
47
48      protected void init() {
49
50
51          for (int i = 0; i < this.numberOfCells; i++) {
52              this.getPopulation().add(this.getRandomLymphocyte());
53          }
54      }
55
56      protected Lymphocyte getRandomLymphocyte() {
57          double v;
58          List<Double> temp = new FastList<Double>();
59
60          for (int j = 0; j < this.dimensions; j++) {
61                  v = (this.mt.nextBoolean())?1:0;
62                  temp.add(v);
63          }
64          return new Lymphocyte(temp);
65      }
66
67      @Override
68      public void evolve() {
69          clonalSelect();
70
71          suppress();
72      }
73
74      protected void clonalSelect() {
75
76
77          for (Lymphocyte cell : this.getPopulation()) {
78              cell.evaluate(getCostFunction());
79          }
80          Collections.sort(this.getPopulation());
81          double max = this.getPopulation().get(this
82                  .getPopulation().size() - 1).getValue();
83
84
85          Lymphocyte cell = null;
86          Lymphocyte mutant = null;
87          Lymphocyte bestMutant = null;
88
```

```
89          long nc;
90          double highestFitness;
91          for (int i = 0; i < this.getPopulation().size(); i++) {
92
93              highestFitness = Double.POSITIVE_INFINITY;
94              cell = this.getPopulation().get(i);
95              nc = Math.round(beta * this.getPopulation().size());
96
97              for (int j = 0; j < nc; j++) {
98                  mutant = (Lymphocyte) this.mutationProvider.mutate(cell, this);
99                  if (mutant.evaluate(getCostFunction()) < highestFitness) {
100                     highestFitness = mutant.getValue();
101                     bestMutant = mutant;
102                 }
103
104             }
105             if (bestMutant.getValue() < cell.getValue()) {
106                 this.getPopulation().set(i, bestMutant);
107             }
108         }
109
110     }
111
112
113     protected void suppress() {
114         Collections.sort(this.getPopulation());
115
116         Lymphocyte cell;
117         int n = this.getPopulation().size() - (int) Math
118                 .round(this.getPopulation().size() * this.getD());
119         for (int i = n; i < this.getPopulation().size(); i++) {
120             cell = this.getRandomLymphocyte();
121             cell.evaluate(getCostFunction());
122             this.getPopulation().set(i, cell);
123         }
124     }
125
126     public double getD() {
127         return d;
128     }
129
130         public double getBeta() {
131                 return beta;
132         }
133
134
135         public MersenneTwisterFast getMt() {
136                 return mt;
137         }
138
139
140         public void setMt(MersenneTwisterFast mt) {
141                 this.mt = mt;
142         }
143 }
```

LISTADO 16. Agente de búsqueda en Clonalg

```
1   package org.biopt.core;
2
3  import java.util.List;
4
5  import org.biopt.optimizers.evolutionary.Individual;
6  import org.biopt.utils.Utils;
7
8  /**
9   *
10  * @author william
11  */
12 public abstract class DefaultSearchAgent implements ISearchAgent{
13
14
15          private List<Double> position;
16      private Double value;
17      public DefaultSearchAgent(List<Double> position) {
18          this.position = position;
19      }
20
21      @Override
22      public Double evaluate(ICostFunction f) {
23          try {
24              this.value = f.evaluate(position);
25              return this.value;
26          }catch(Exception e){
27              System.out.println(e);
28          }
29          return Double.NaN;
30      }
31
32      @Override
33      public List<Double> getPosition() {
34          return position;
35      }
36
37      @Override
38      public void setPosition(List<Double> position) {
39          this.position = position;
40      }
41
42      @Override
43      public Double getValue() {
44          return value;
45      }
46
47      public void setValue(Double value) {
48          this.value = value;
49
50      }
51
52      @Override
53      public int compareTo(ISearchAgent agent) {
54          //System.out.println(agent);
```

```
55          if (this.value < agent.getValue()) {
56              return -1;
57          }
58          if (this.value > agent.getValue()) {
59              return 1;
60          } else {
61              return 0;
62          }
63          //return (int)(this.value - cell.getValue());
64      }
65
66      @Override
67          public Individual clone() {
68                  // TODO Auto-generated method stub
69                  Individual ind = new Individual(Utils.doubleListDeepCopy(getPosition()));
70                  ind.setValue(getValue());
71                  return ind;
72          }
73
74      @Override
75          public String toString() {
76                  // TODO Auto-generated method stub
77                  return "position: "+position.toString()+" value: "+value;
78          }
79
80  }
```

LISTADO 17. Función de costo para el sistema inmune artificial

```
1   package co.utb.ml.fiebreshemorragicas;
2
3   import java.util.ArrayList;
4   import java.util.Collections;
5   import java.util.List;
6
7   import org.biopt.core.ICostFunction;
8   import org.jane.core.networks.DefaultARTMAP;
9   import org.jane.core.networks.FuzzyARTmap;
10  import org.jane.core.networks.ARTMAP_IC;
11  import org.jane.utils.VectorUtils;
12
13  public class ARTMAPCostFunction implements ICostFunction {
14
15          @Override
16          public double evaluate(List<Double> inputs) throws Exception {
17                  // TODO Auto-generated method stub
18                  FuzzyARTmap ARTMAP = new DefaultARTMAP(0.1f, 0.01f, 0f, 1,
19                          -0.001f,4,DefaultARTMAP.DEFAULT_ARTMAP2);
20                  //FuzzyARTmap ARTMAP =
21                          //new ARTMAP_IC(0.1f, 0.01f, 0f, 1,-0.1f,9,ARTMAP_IC.QMAX);
22                  //FuzzyARTmap ARTMAP = new FuzzyARTmap(0.1f, 0.01f, 0f, 1, -0.001f);
23                  //ARTMAPEnsemble<FuzzyARTmap> ARTMAP =
24                  // new ARTMAPEnsemble<FuzzyARTmap>(FuzzyARTmap.class, 9);
25                  // for(int i=0; i < 5; i++)
26                  List<List<List<Float>>> training = applyMask(inputs,Main.trainingSet);
27
```

```
28              ARTMAP.train(training);
29              // System.out.println("Número de categorias: " +
30              // ARTMAP.getNumberOfCategories());
31
32              int[] truePositives = { 0, 0, 0 };
33              int[] falsePositives = { 0, 0, 0 };
34              int[] trueNegatives = { 0, 0, 0 };
35              int[] falseNegatives = { 0, 0, 0 };
36              int[] correctAnswers = { 0, 0, 0 };
37              int[] wrongAnswers = { 0, 0, 0 };
38              List<Float> r = null;
39              int category;
40              List<List<List<Float>>> validation = applyMask(inputs,Main.validationSet);
41              for (int i = 0; i < Main.validationSet.size(); i++) {
42                      // category =
43                      // artmap.processData(validationSet.get(i).get(0)).indexOf(1f);
44                      r = ARTMAP.processData(validation.get(i).get(0));
45                      category = r.indexOf(Collections.max(r));
46                      int realCategory = validation.get(i).get(1).indexOf(1f);
47                      //System.out.println(r + "----" + realCategory);
48                      if (category == realCategory) {
49                              for (int j = 0; j < 3; j++) {
50                                      if (j == category) {
51                                              truePositives[category]++;
52                                      } else {
53                                              trueNegatives[category]++;
54                                      }
55                              }
56                              correctAnswers[realCategory]++;
57                      } else {
58                              for (int j = 0; j < 3; j++) {
59                                      if (j == category) {
60                                              falsePositives[j]++;
61                                      } else {
62                                              if (j == realCategory) {
63                                                      falseNegatives[j]++;
64                                              } else {
65                                                      trueNegatives[j]++;
66                                              }
67                                      }
68                              }
69                              wrongAnswers[realCategory]++;
70                      }
71              }
72              float precision0 = ((float)truePositives[0])/(truePositives[0]+falsePositives[0]);
73              float recall0 = ((float)truePositives[0])/(truePositives[0]+falseNegatives[0]);
74              float f10 = 0;
75              if (precision0 != 0 && recall0 !=0)
76                      f10 = 2*precision0*recall0/(precision0+recall0);
77
78              float precision1 = ((float)truePositives[1])/(truePositives[1]+falsePositives[1]);
79              float recall1 = ((float)truePositives[1])/(truePositives[1]+falseNegatives[1]);
80              float f11 = 0;
81              if (precision1 != 0 && recall1 !=0)
82                      f11 = 2*precision1*recall1/(precision1+recall1);
```

```
83
84                  float precision2 = ((float)truePositives[2])/(truePositives[2]+falsePositives[2]);
85                  float recall2 = ((float)truePositives[2])/(truePositives[2]+falseNegatives[2]);
86                  float f12 = 0;
87                  if (precision2 != 0 && recall2 !=0)
88                          f12 = 2*precision2*recall2/(precision2+recall2);
89
90                  //System.out.println(f10+"--"+f11+"--"+f12);
91                  return -0.95*(f10 + f11 + f12)/3 + 0.05*VectorUtils.getDoubleL1Norm(inputs);
92                  //return 0.1*wrongAnswers[0] + 0.4*wrongAnswers[0] + 0.4*wrongAnswers[0];
93          }
94
95          private static List<List<List<Float>>> applyMask(List<Double> mask,
96                          List<List<List<Float>>> dataset) {
97                  List<List<List<Float>>> resultSet = new ArrayList<List<List<Float>>>();
98                  List<List<Float>> newRow = null;
99                  List<Float> inputs = null;
100                 for (List<List<Float>> row : dataset) {
101                         newRow = new ArrayList<List<Float>>();
102                         inputs = new ArrayList<Float>();
103                         for (int i = 0; i < mask.size()*2; i++) {
104                                 if(i < mask.size()) {
105                                         if (mask.get(i) == 1) {
106                                                 inputs.add(row.get(0).get(i));
107                                         }
108                                 }else{
109                                         if (mask.get(i - mask.size()) == 1) {
110                                                 inputs.add(row.get(0).get(i));
111                                         }
112                                 }
113                         }
114                         newRow.add(inputs);
115                         newRow.add(row.get(1));
116                         resultSet.add(newRow);
117                 }
118
119                 return resultSet;
120         }
121
122 }
```

LISTADO 18. Rutina principal

```
1   package co.utb.ml.fiebreshemorragicas;
2
3   import java.io.BufferedReader;
4   import java.io.FileReader;
5   import java.util.ArrayList;
6   import java.util.Arrays;
7   import java.util.Collections;
8   import java.util.List;
9
10  import org.biopt.optimizers.immune.BitFlipMutationProvider;
11  import org.biopt.optimizers.immune.ClonalgOptimizer;
12  import org.jane.core.networks.ARTMAPEnsemble;
13  import org.jane.core.networks.ARTMAP_IC;
```

```
14 import org.jane.core.networks.DefaultARTMAP;
15 import org.jane.core.networks.FuzzyARTmap;
16
17 public class Main {
18
19         public static List<List<List<Float>>> trainingSet;
20         public static List<List<List<Float>>> validationSet;
21         public static List<List<List<Float>>> testSet;
22         static {
23                 String line = "";
24                 String[] arr = null;
25                 List<Float> in = null;
26                 List<List<Float>> row = null;
27                 Main.trainingSet = new ArrayList<List<List<Float>>>();
28                 Main.validationSet = new ArrayList<List<List<Float>>>();
29                 Main.testSet = new ArrayList<List<List<Float>>>();
30                 try {
31                         FileReader fr = new FileReader("trainingSetNew.csv");
32                         BufferedReader br = new BufferedReader(fr);
33                         br.readLine();
34
35                         while ((line = br.readLine()) != null) {
36                                 arr = line.split(";");
37                                 in = new ArrayList<Float>();
38                                 row = new ArrayList<List<Float>>();
39                                 for (int i = 1; i < 31; i++) {
40
41                                         try {
42                                                 in.add(Float.parseFloat(arr[i]));
43                                         } catch (NumberFormatException e) {
44                                                 in.add(0f);
45                                         }
46                                 }
47                                 row.add(in);
48                                 row.add(toBinaryArray(Integer.parseInt(arr[0]), 3));
49                                 trainingSet.add(row);
50                         }
51                         Utils.complementCode(Main.trainingSet);
52                         /************************ Validation *****************/
53                         fr = new FileReader("validationsetNew.csv");
54                         br = new BufferedReader(fr);
55                         br.readLine();
56
57                         while ((line = br.readLine()) != null) {
58                                 arr = line.split(";");
59                                 in = new ArrayList<Float>();
60                                 row = new ArrayList<List<Float>>();
61                                 for (int i = 1; i < 31; i++) {
62
63                                         try {
64                                                 in.add(Float.parseFloat(arr[i]));
65                                         } catch (NumberFormatException e) {
66                                                 in.add(0f);
67                                         }
68                                 }
```

```
69                              row.add(in);
70                              row.add(toBinaryArray(Integer.parseInt(arr[0]), 3));
71                              validationSet.add(row);
72                      }
73                      Utils.complementCode(Main.validationSet);
74                      /****************** TEST ******************************/
75                      fr = new FileReader("testsetNew.csv");
76                      br = new BufferedReader(fr);
77                      br.readLine();
78
79                      while ((line = br.readLine()) != null) {
80                              arr = line.split(";");
81                              in = new ArrayList<Float>();
82                              row = new ArrayList<List<Float>>();
83                              for (int i = 1; i < 31; i++) {
84
85                                      try {
86                                              in.add(Float.parseFloat(arr[i]));
87                                      } catch (NumberFormatException e) {
88                                              in.add(0f);
89                                      }
90                              }
91                              row.add(in);
92                              row.add(toBinaryArray(Integer.parseInt(arr[0]), 3));
93                              testSet.add(row);
94                      }
95                      Utils.complementCode(Main.testSet);
96
97              } catch (Exception e) {
98              }
99
100
101     }
102
103     /**
104      * @param args
105      */
106     public static void main(String[] args) throws Exception {
107             // TODO Auto-generated method stub
108
109             //optimize();
110
111             testNetwork();
112
113     }
114
115     public static void optimize() {
116             ClonalgOptimizer opt = new ClonalgOptimizer(30, 10, 100, 0.3, null,
117                             null, "co.utb.ml.fiebreshemorragicas.ARTMAPCostFunction",
118                             new BitFlipMutationProvider(0.5), true);
119             for (int i = 0; i < 10; i++) {
120                     opt.evolve();
121                     System.out.println("********************");
122                     System.out.println(opt.getBestSolution().getPosition());
123                     System.out.println(opt.getBestSolution().getValue());
```

```
124             }
125         }
126
127         public static List<Float> toBinaryArray(int f, int size) {
128             List<Float> result = new ArrayList<Float>();
129             for (int i = 0; i < size; i++) {
130                 if (i == f) {
131                     result.add(1f);
132                 } else {
133                     result.add(0f);
134                 }
135             }
136             return result;
137         }
138
139         private static List<List<List<Float>>> applyMask(List<Double> mask,
140                 List<List<List<Float>>> dataset) {
141             List<List<List<Float>>> resultSet = new ArrayList<List<List<Float>>>();
142             List<List<Float>> newRow = null;
143             List<Float> inputs = null;
144             for (List<List<Float>> row : dataset) {
145                 newRow = new ArrayList<List<Float>>();
146                 inputs = new ArrayList<Float>();
147                 for (int i = 0; i < mask.size()*2; i++) {
148                     if(i < mask.size()) {
149                         if (mask.get(i) == 1) {
150                             inputs.add(row.get(0).get(i));
151                         }
152                     }else{
153                         if (mask.get(i - mask.size()) == 1) {
154                             inputs.add(row.get(0).get(i));
155                         }
156                     }
157                 }
158                 newRow.add(inputs);
159                 newRow.add(row.get(1));
160                 resultSet.add(newRow);
161             }
162
163             return resultSet;
164         }
165
166         @SuppressWarnings("unused")
167         public static void testNetwork() {
168
169             String[] predictors = { "Edad (Meses)", "Temperatura", "Tipo Fiebre",
170                     "Signo torniquete?", "Anorexia", "Escalofríos", "Tipo Cefalea",
171                     "Dolor Retro-ocular?", "Artralgia?", "Mialgia?", "Exantema?",
172                     "Islas blancas en mar rojo?", "Prurito?", "Eritema Facial?",
173                     "Contacto con agua llovida/estancada?", "Hiperkalemia?",
174                     "Hiponatremia?", "CPK elevado?", "Compromiso Renal?",
175                     "Ictericia?", "Contacto con perros o roedores?",
176                     "Dolor Pantorrillas?", "Hemoglobina (gr/dl)",
177                     "Leucocitos (miles/mm3)", "Linfocitos(miles/mm3)",
178                     "Neutrófilos(miles/mm3)", "Mes de consulta", "Dias de fiebre",
```

```
179                     "Hematocrito (%)", "Lugar del exantema" };
180             //FuzzyARTmap ARTMAP =
181                     //new DefaultARTMAP(0.1f, 0.01f, 0f, 1,−0.001f,4,
182                     //DefaultARTMAP.DEFAULT_ARTMAP2);
183             FuzzyARTmap ARTMAP =
184                     new ARTMAP_IC(0.1f, 0.01f, 0f,1,−0.001f,9,ARTMAP_IC.QMAX);
185             //FuzzyARTmap ARTMAP = new FuzzyARTmap(0.1f, 0.01f, 0f, 1, −0.01f);
186             //ARTMAPEnsemble<FuzzyARTmap> ARTMAP =
187                     //new ARTMAPEnsemble<FuzzyARTmap>(FuzzyARTmap.class, 9);
188             // Collections.shuffle(trainingSet);
189             Double[] baseline = new Double[]{1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
190                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
191                     1.0, 1.0, 1.0, 1.0,1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
192             Double[] baseline_no_K_Na =
193                     new Double[]{1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
194                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
195                      0.0, 0.0, 1.0, 1.0, 1.0, 1.0,1.0, 1.0, 1.0,
196                      1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
197
198             Double[] newFAM = new Double[]{1.0, 0.0, 0.0, 1.0,
199                     1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
200                     1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0,
201                     0.0, 1.0, 1.0, 0.0, 1.0, 1.0,0.0};
202             Double[] newDAM = new Double[]{1.0, 1.0, 1.0, 1.0, 1.0,
203                     1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0,
204                     0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0,
205                     1.0, 0.0, 1.0, 0.0};
206             Double[] regularizedDAM = new Double[]{0.0, 1.0, 0.0, 0.0, 0.0,
207                     1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0,
208                     1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
209             List<Double> mask = Arrays.asList(newDAM);
210             //List<Double> mask = Arrays.asList(baseline_no_K_Na);
211             List<List<List<Float>>> training = applyMask(mask, Main.trainingSet);
212             //System.out.println("Size: "+training.get(0).get(0).size());
213             List<List<List<Float>>> validation = applyMask(mask, Main.validationSet);
214             List<List<List<Float>>> test = applyMask(mask, Main.testSet);
215
216             //for(int i=0; i < 2; i++) {
217                     ARTMAP.train(training);
218                     ARTMAP.train(validation);
219             //}
220             System.out.println("Entrenamiento finalizado");
221             System.out.println("Número de categorias: "
222                     + ARTMAP.getNumberOfCategories());
223             /************************ Validation ***************/
224             int[] truePositives = { 0, 0, 0 };
225             int[] falsePositives = { 0, 0, 0 };
226             int[] trueNegatives = { 0, 0, 0 };
227             int[] falseNegatives = { 0, 0, 0 };
228             int[] correctAnswers = { 0, 0, 0 };
229             int[] wrongAnswers = { 0, 0, 0 };
230             List<Float> r = null;
231             int category;
232             if (false) {
233                     for (int i = 0; i < Main.validationSet.size(); i++) {
```

```
234                         // category =
235                         // artmap.processData(validationSet.get(i).get(0)).indexOf(1f);
236
237                         r = ARTMAP.processData(validation.get(i).get(0));
238                         category = r.indexOf(Collections.max(r));
239                         int realCategory = validation.get(i).get(1).indexOf(1f);
240                         System.out.println(r + "----" + realCategory);
241                         if (category == realCategory) {
242                                 for (int j = 0; j < 3; j++) {
243                                         if (j == category) {
244                                                 truePositives[category]++;
245                                         } else {
246                                                 trueNegatives[category]++;
247                                         }
248                                 }
249                                 correctAnswers[realCategory]++;
250                         } else {
251                                 for (int j = 0; j < 3; j++) {
252                                         if (j == category) {
253                                                 falsePositives[j]++;
254                                         } else {
255                                                 if (j == realCategory) {
256                                                         falseNegatives[j]++;
257                                                 } else {
258                                                         trueNegatives[j]++;
259                                                 }
260                                         }
261                                 }
262                                 wrongAnswers[realCategory]++;
263                         }
264                 }
265         }else{
266                 /********************* TEST *************/
267                 for (int i = 0; i < Main.testSet.size(); i++) {
268                         // category =
269                         // artmap.processData(validationSet.get(i).get(0)).indexOf(1f);
270
271                         r = ARTMAP.processData(test.get(i).get(0));
272                         category = r.indexOf(Collections.max(r));
273                         int realCategory = test.get(i).get(1).indexOf(1f);
274                         System.out.println(r + "----" + realCategory);
275                         if (category == realCategory) {
276                                 for (int j = 0; j < 3; j++) {
277                                         if (j == category) {
278                                                 truePositives[category]++;
279                                         } else {
280                                                 trueNegatives[category]++;
281                                         }
282                                 }
283                                 correctAnswers[realCategory]++;
284                         } else {
285                                 for (int j = 0; j < 3; j++) {
286                                         if (j == category) {
287                                                 falsePositives[j]++;
288                                         } else {
```

```
289                                                     if (j == realCategory) {
290                                                             falseNegatives[j]++;
291                                                     } else {
292                                                             trueNegatives[j]++;
293                                                     }
294                                             }
295                                     }
296                                     wrongAnswers[realCategory]++;
297                             }
298                     }
299             }
300             System.out.println("correctas: " + Arrays.toString(correctAnswers)
301                             + " errores: " + Arrays.toString(wrongAnswers));
302             System.out.println("True Positives: " + Arrays.toString(truePositives));
303             System.out.println("False Positives: "
304                             + Arrays.toString(falsePositives));
305             System.out.println("True Negatives: " + Arrays.toString(trueNegatives));
306             System.out.println("False Negatives: "
307                             + Arrays.toString(falseNegatives));
308
309
310             float precision0 = ((float)truePositives[0])/(truePositives[0]+falsePositives[0]);
311             float recall0 = ((float)truePositives[0])/(truePositives[0]+falseNegatives[0]);
312             float f10 = 0;
313             if (precision0 != 0 && recall0 !=0)
314                     f10 = 2*precision0*recall0/(precision0+recall0);
315
316             float precision1 = ((float)truePositives[1])/(truePositives[1]+falsePositives[1]);
317             float recall1 = ((float)truePositives[1])/(truePositives[1]+falseNegatives[1]);
318             float f11 = 0;
319             if (precision1 != 0 && recall1 !=0)
320                     f11 = 2*precision1*recall1/(precision1+recall1);
321
322             float precision2 = ((float)truePositives[2])/(truePositives[2]+falsePositives[2]);
323             float recall2 = ((float)truePositives[2])/(truePositives[2]+falseNegatives[2]);
324             float f12 = 0;
325             if (precision2 != 0 && recall2 !=0)
326                     f12 = 2*precision2*recall2/(precision2+recall2);
327
328             System.out.println("**************************");
329             System.out.println("Dengue");
330             System.out.println("Precision: "+precision0);
331             System.out.println("Recall: "+recall0);
332             System.out.println("F-Score: "+f10);
333             System.out.println("Leptospirosis");
334             System.out.println("Precision: "+precision1);
335             System.out.println("Recall: "+recall1);
336             System.out.println("F-Score: "+f11);
337             System.out.println("Malaria");
338             System.out.println("Precision: "+precision2);
339             System.out.println("Recall: "+recall2);
340             System.out.println("F-Score: "+f12);
341             double avg = (f10 + f11 + f12)/3;
342             System.out.println("Macro-Average: "+avg);
343             System.out.println("predictores");
```

```
344                     int c = 0;
345 //  for(Double d : mask) {
346 //   if (d == 1) System.out.println(predictors[c]);
347 //   c++;
348 //  }
349 //  System.out.println("predictores descartados");
350 //  c = 0;
351 //  for(Double d : mask) {
352 //   if (d == 0) System.out.println(predictors[c]);
353 //   c++;
354 //  }
355         }
356 }
```