UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Árboles auto-organizativos para la clasificación de datos atípicos

JAVIER IGNACIO AROS MENDOZA

Profesor Guía: CÉSAR ALEJANDRO ASTUDILLO HERNÁNDEZ

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
Diciembre, 2017

CONSTANCIA

La Dirección del Sistema de Bibliotecas a través de su encargado Biblioteca Campus Curicó certifica que el autor del siguiente trabajo de titulación ha firmado su autorización para la reproducción en forma total o parcial e ilimitada del mismo.

Curicó, 2019

A mis padres, hermanos, seres queridos y amigos por su apoyo incondicional.

AGRADECIMIENTOS

En estas últimas palabras que dejo en esta memoria quisiera agradece a mi familia por su apoyo incondicional, en especial a mis padres y hermanos por su cariño y aliento.

A todas las personas con las que disfrute estos años de universidad, a mis compañeros de carrera y amigos con los que me forme como profesional.

A la Fundación de Hogar de Estudiantes Universitarios, en especial a mis hermanos de Casa Rodriguez por acogerme en este proceso y hacerme sentir como en casa.

Por último, pero no por eso menos importante a mi profesor guía Cesar Astudillo y Colin Bellinger, por su constante apoyo y orientación en la realización de esta memoria.

RESUMEN

La agrupación y clasificación son problemas estudiados en Machine Learning. La agrupación corresponde al problema de identificar dentro de un conjunto de datos las clases involucradas, para aglutinar estas instancias no existe información de las clases y en ocasiones se desconoce la cantidad de categorías posible. Esto consiste en asignar elementos similares a un grupo y a la vez asignar a grupos distintos instancias distintas. La clasificación corresponde a determinar mediante una clase conocida, una nueva instancia sin clasificar, determinando si esta última corresponde a la clase ya mencionada o es un elemento extraño, esto se realiza utilizando información estadística de los datos.

Con este trabajo se presenta el desarrollo e implementación de Complete k-ary Tree SOM (CKTSOM), un algoritmo con la capacidad de agrupar y clasificar, esté utiliza una estructura de árbol capaz de aprender la distribución de un conjunto de datos de entrada. La estructura del árbol siempre es completa, esto significa que todos los nodos hojas tienen la misma altura. Estos nodos se pueden asociar a neuronas por su capacidad de aprender de la información presentada.

En este algoritmo las neuronas compiten para poder representar un dato de entrada, pero hay que destacar un cambio en el paradigma, donde solo competirán las neuronas hojas. Esto es un cambio drástico ya que comúnmente toda la estructura es utilizada para representar la instancia. La búsqueda generada con este cambio de paradigma produce una búsqueda logarítmica que nos entrega un resultado aproximado, este es analizado y verificado para comprender su calidad.

Por último se presenta la implementación del algoritmo para clasificar distintos conjuntos de datos que varían en cantidad de instancias y dimensiones. Esta implementación fue utilizada para comparar el algoritmo implementado con otros clasificadores. Para comparar el desempeño se utiliza la medida AUC que nos proporciona el rendimiento del algoritmo, un AUC menor a 0.5 indica que el clasificador es peor que una elección aleatoria.

Palabras Claves: SOM, TTOSOM, Aprendizaje automático, OCC, AUC.

ABSTRACT

Grouping and classification are problems studied in Machine Learning. The grouping corresponds to the problem of identifying within a set of data the classes involved, to agglutinate these instances there is no information of the classes and sometimes the number of possible categories is unknown. This consists of assigning similar elements to a group and at the same time assigning different instances to different groups. The classification corresponds to determine by means of a known class, a new instance without classifying, determining if this last one corresponds to the class already mentioned or it is a strange element, this is made using statistical information of the data.

This work presents the development and implementation of Complete k-ary Tree SOM (CKTSOM), an algorithm with the ability to group and classify, using a tree structure capable of learning the distribution of a set of input data. The tree structure is always complete, this means that all leaf nodes have the same height. These nodes can be associated with neurons because of their ability to learn from the information presented.

In this algorithm neurons compete to be able to represent an input data, but it is necessary to highlight a change in the paradigm, where only leaf neurons will compete. This is a drastic change since commonly the whole structure is used to represent the instance. The search generated with this paradigm shift produces a logarithmic search that gives us an approximate result, this is analyzed and verified to understand its quality.

Finally we present the implementation of the algorithm to classify different sets of data that vary in number of instances and dimensions. This implementation was used to compare the implemented algorithm with other classifiers. To compare the performance we use the AUC measure that provides the performance of the algorithm, an AUC less than 0.5 indicates that the classifier is worse than a random choice.

TABLA DE CONTENIDOS

página

ÍDICE DE FIGURAS

página

ÍNDICE DE TABLAS

página

1. Introducción

La agrupación de datos (clustering), es un problema típico al trabajar con gran cantidad de datos referentes a diversas áreas científicas y/o procesos de negocio tales como la bioinformática, informática, marketing, entre otros. Esto tiene como objetivo asociar elementos que son similares entre sí. Para lograr esto, existen distintos algoritmos que tratan de solucionar este problema utilizando variadas técnicas y paradigmas, pero ninguno logra una clasificación apropiada para todas las distribuciones de los datos, por esto es importante la investigación de nuevas técnicas que puedan solucionar este obstáculo y comprender en qué circunstancia este método entrega un resultado aceptable.

Dentro de estos algoritmo se encuentra Self Organizing Maps (SOM), que es una técnica bastante estudiada y utilizada [6]. Este procedimiento usa un aprendizaje no supervisado para producir una representación de los datos usando neuronas que aprenden de éstos, utiliza una especie de malla que se amolda a la forma en que los datos están distribuidos en el espacio, lo que permite obtener una estructura que representa los datos que usualmente tienen un menor tamaño que los datos originales.

Para ejemplificar, podemos mencionar una aplicación en particular de SOM en el análisis de datos bursátiles italianos que demostró un interesante poder para comprender las relaciones entre diferentes acciones. Para lograr esto se implementó una red neuronal capaz de capturar las relaciones de productos en el mercado, de este modo predecir o prever patrones financieros. Esto tiene como consecuencia una importante ventaja lucrativa para un inversor informado [10].

Como se mencionó, SOM se ha usado exitosamente en muchos ámbitos teniendo un impacto positivo, pero aún tiene deficiencias importantes que se tratan de solventar de distintas maneras. La primera que podríamos mencionar es que se debe configurar la malla antes de iniciar el entrenamiento, esto significa que para lograr una instrucción adecuada se debe ejecutar varias veces, lo que genera una configuración inicial tediosa y lenta. Además, se debe encontrar el tamaño adecuado de la red de neuronas donde una malla pequeña podría representar los datos de manera inadecuada o una grande podría ser costosa de generar. Existen publicaciones científicas que abordan estos problemas como Topology-oriented self-organizing maps: a survey [6], aunque no existe una solución definitiva que los resuelva todos.

La literatura aporta distintas variaciones de SOM que tratan de solucionar los problemas mencionados utilizando distintos tipos de estructuras topológicas. Las más utilizadas son: malla, árbol, grafo y malla irregular. Por lo general, la topología utilizada en las variantes son mezclas de las topologías mencionadas anteriormente. Dentro de esta literatura se encuentra Tree-based Topology Oriented SOM (TTOSOM) [2] el cual utiliza una estructura de árbol en lugar de la malla que puede identificar la distribución subyacente en los datos analizados.

El problema a resolver es de clasificación, donde se desea desarrollar una implementación basada en SOM que sea capaz de identificar ciertos objetos dentro de un conjunto de datos, este tipo de clasificador es denominado One-class classification (OCC) o en español Clasificación de una clase. El OCC tiene la capacidad de aprender la distribución subyacente de los datos analizados mediante la exposición de objetos de una clase en particular. Con un buen entrenamiento, este tipo de clasificador es capaz de determinar si un objeto en particular pertenece a la clase mostrada en el proceso de entrenamiento, de no reconocer este objeto éste es denominado como un dato atípico o también llamado outlier [14].

Para resolver esto, se propone el desarrollo de una nueva implementación de SOM que al igual que TTOSOM utilizará una topología de árbol, pero tendrá variaciones importantes. La estructura de éste tendrá que ser completa, es decir, que todos los hijos especificados inicialmente tendrán que estar definidos, además, cuando un dato entra al entrenamiento, solo los nodos hojas podrán competir por ser la unidad de

mejor correspondencia (BMU por sus siglas en inglés) que en el SOM tradicional y TTOSOM corresponde a toda la estructura.

1.1. Objetivos

El objetivo es diseñar e implementar el algoritmo CKTSOM, para ser utilizado en agrupación de datos reales con métodos de visualización.

1.1.1. Objetivos específicos

1. Investigar y desarrollar una implementación de SOM.
2. Diseñar e implementar la adaptación de SOM utilizando una topología de árbol (CKTSOM).
3. Desarrollar módulo de visualización gráfica para los algoritmos SOM y CKTSOM.
4. Analizar el desempeño de CKTSOM en relación al algoritmo SOM.
5. Análisis multidimensional del conjunto de datos.

1.2. Alcance del trabajo a realizar

1. Implementación del algoritmo SOM y CKTSOM en R [18] junto con los módulos de visualización de datos. Se espera que los algoritmos sean capaces de realizar aprendizaje automático, y que los módulos de visualización gráfica permitan realizar el análisis exploratorio respectivo. Las características del lenguaje R para el tratamiento estadístico de datos, lo transforma en una solución ideal para la implementación de estos algoritmos de aprendizaje automático.
2. Implementar en C++ los módulos internos de los algoritmos. Las características positivas del desempeño del código en C++ permitirán que los algoritmos de aprendizaje implementados puedan entregar soluciones en tiempos razonables.
3. Integrar las implementaciones en R y C++ respectivamente [11].

4. Se realizará una serie de experimentos para verificar la efectividad del modelo. Estos experimentos involucran un análisis estadístico a partir de la validación de los modelos en diversos conjuntos de datos.

2. Revisión Literaria

2.1. Introducción

En este capítulo se realiza una revisión literaria sobre los términos y conceptos necesarios para leer y entender esta memoria. Para esto se describen los conceptos de aprendizaje de máquina, luego de esto se detalla los algoritmos Self-Organizing Maps (SOM) y Tree-based Topology Oriented SOM (TTOSOM). En la sección subsecuente se incluyen métodos de visualización comúnmente utilizados en algoritmos SOM. Finalmente se explica el algoritmo central de esta tesis llamado Complete k-ary Tree SOM (CKTSOM).

2.2. Aprendizaje automático

El aprendizaje automático o aprendizaje de máquina, es un área de la ciencia de la computación y una rama de la inteligencia artificial que estudia e investiga técnicas que permita a la computadora aprender. Para lograr esto, se tienen distintos tipos de aprendizaje, los cuales se clasifican principalmente en aprendizaje supervisado, no supervisado y semi-supervisado [12]. Otros esquemas de aprendizaje automático involucran el paradigma aprendiendo a aprender y aprendizaje por reforzamiento [12]. En el aprendizaje supervisado, el modelo es creado a partir de ejemplos que contienen tanto las variables independientes como las variables a predecir. EL aprendizaje no supervisado se identifica como cualquier proceso de aprendizaje donde el entrenamiento se realiza en ausencia de una salida definida o retroalimentación. Aprendizaje semi-supervisado se caracteriza por utilizar datos etiquetados y no etiquetados para realizar un aprendizaje supervisado o aprendizaje no supervisado. Por último, en el aprendizaje por reforzamiento, se utiliza un agente de entrenamiento, cuyo objetivo

es aprender una política que permita maximizar la recompensa recibida. Dependiendo del tipo de respuesta, en este aprendizaje el agente es recompensado cuando el resultado es correcto, pero es penalizado cuando la respuesta no es la esperada.

2.3. Self-Organizing Maps (SOM)

Self-Organizing Maps (SOM) es un tipo de red neuronal capaz de realizar aprendizaje automático. SOM es utilizado para agrupar datos multidimensionales, para lo que utiliza una malla de neuronas interconectadas, cuya forma se va amoldando a la manera en que los datos están distribuidos en el espacio. En este mecanismo, la malla corresponde a una serie de neuronas conectadas entre sí, formando una especie de matriz neuronal de m filas y n columnas, lo que da un total de $m \times n$ neuronas interconectadas. SOM realiza un tipo de aprendizaje automático denominado "no supervisado", en el cual cada neurona de la malla va migrando dentro del espacio hasta formar una representación discreta de los datos. Cuando el algoritmo converge de manera correcta, la malla resultante posee dos propiedades: representa la distribución de datos de manera adecuada, y se produce un fenómeno denominado preservación de la topología. Este último concepto significa que elementos que se encuentran cerca en el espacio solución, también se encuentran asociados a neuronas que se encuentran cerca en la malla.

Para iniciar el algoritmo, tiene como entrada el conjunto $X \subseteq \mathbb{R}^d$ con el cual se debe inicializar la malla, definir la función de vecindad y las reglas de actualización especificando el factor de aprendizaje y el radio de vecindad.

La primera etapa del entrenamiento consiste en la fase de inicialización. En esta etapa, la estructura de malla es sobrepuesta de forma desordenada en el espacio donde se encuentran los datos. La malla es constituida por neuronas, las que, a su vez, tienen asociado un vector que posee la misma dimensión que los vectores del conjunto de datos original. La inicialización se materializa asignando un vector a cada neurona. Este vector puede ser generado de manera completamente aleatoria o puede usarse otro mecanismo más práctico. Típicamente, la inicialización ocurre a través de la selección aleatoria de instancias dentro del conjunto original (por ejemplo, usando selección aleatoria uniforme y con reemplazamiento) y copiando estos valores en la

neurona respectiva. Existen autores que han mencionado formas para asignar valores iniciales a las neuronas, de tal manera de acelerar el proceso de entrenamiento. Estos métodos generalmente envuelven el uso de técnicas algebraicas a través del uso de vectores y valores propios o la incorporación de técnicas bio-inspiradas como Algoritmos Genéticos o Estrategias Evolutivas [1]

Luego de tener la malla inicializada, se selecciona una instancia del conjunto de entrada X, la cual llamaremos estímulo, lo que provocará que las neuronas compitan entre sí para identificar la más cercana, llamándola Best Matching Unit (BMU). En términos geométricos, esta cercanía se asocia a una función de similitud, típicamente vinculada a la noción de distancia. La hipótesis es que la neurona que se encuentre más cercana al estímulo es la que mejor representa a ese dato. Con esto se determina el BMU y se denota como BMU(x) donde w_i son los pesos de una neurona y x es el estímulo, evidenciado la ecuación (2.1).

$$\mathsf{BMU}(x) = \min_i |x - w_i| \tag{2.1}$$

La búsqueda de la BMU corresponde a un proceso costoso en términos de tiempo computacional asociado al SOM y constituye una de las mayores dificultades conocidas del algoritmo. Varios autores han puesto atención a este problema y han sugerido mejoras al algoritmo para acelerar este proceso [9].

Teniendo identificada el BMU, se procede a determinar los vecinos. Este proceso se realiza mediante la función de vecindad, que utiliza el radio de vecindad determinando cuantas interconexiones existirán entre la BMU y una neurona para considerar a esta última como vecino. El concepto de vecindad está estrechamente vinculado a la distancia entre neuronas. En el caso del SOM, se define como el número mínimo de interconexión que separan dos neuronas. El radio, entonces, define la llamada burbuja de actividad, que involucra a todas las neuronas que se encuentran a una distancia con respecto de la BMU que es igual o menor al valor del radio especificado. A continuación en la Figura 2.1 se presenta un ejemplo de lo mencionado.

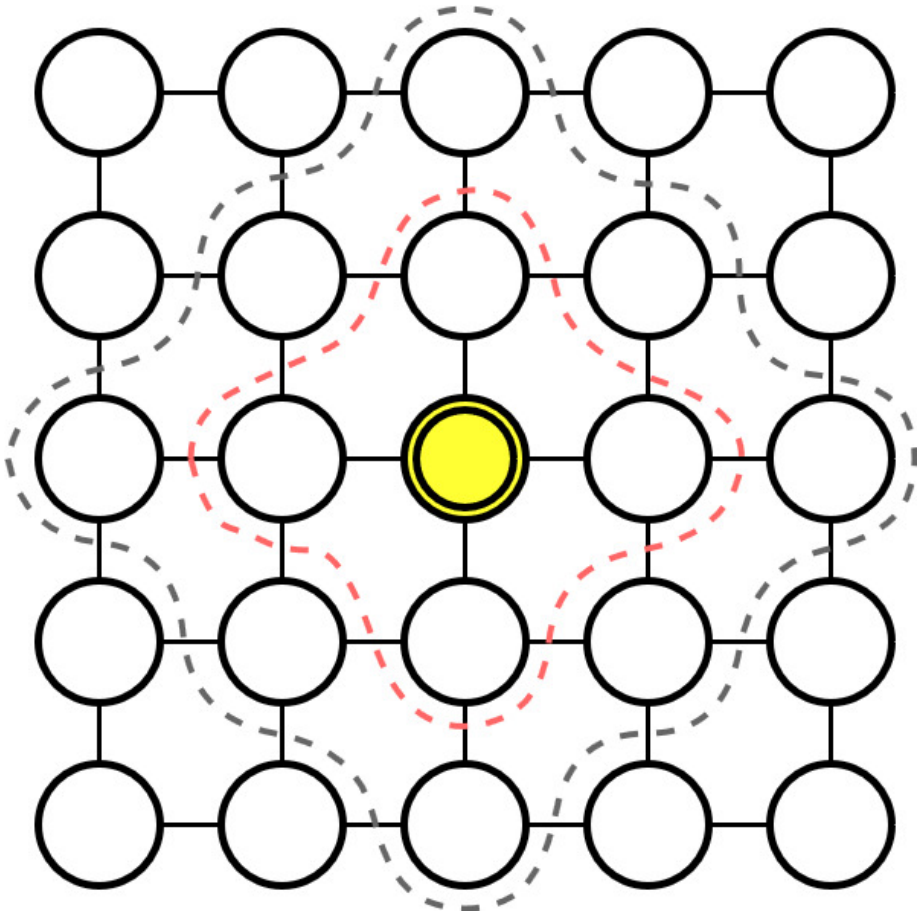


Figura 2.1: Representación de vecindad en SOM.

En la Figura 2.1, se simboliza la vecindad donde el círculo amarillo representa el BMU y los círculos blancos otras neuronas interconectadas por las aristas negras, mientras que la línea punteada roja encierra las neuronas que se encuentran a radio 1 del BMU y la línea punteada gris encierra a las neuronas a radio 2 del BMU.

Con el BMU y su vecindad identificada se inicia el proceso de migración, donde la información del estímulo es absorbida total o parcialmente por el BMU y sus vecinos dependiendo del factor de aprendizaje, que comúnmente se inicializa de tal forma que la absorción es completa en etapas tempranas del algoritmo, lo que provoca un desplazamiento mayor y una pérdida de información obtenida anteriormente. Al contrario, en las etapas finales el desplazamiento es menor, lo que permite que las neuronas recuerden lo aprendido, produciendo ajustes menores y refinados. Esta

migración se puede observar en la siguiente ecuación (2.2) donde w_i son los pesos de una neurona en un tiempo t y $\alpha(t)$ es el factor de aprendizaje.

$$w_i(t+1) = w_i(t) + \alpha(t)(x - w_i(t)) \tag{2.2}$$

Todo este proceso se repite de manera iterativa, seleccionando una instancia del conjunto de entrenamiento de manera estocástica. Durante cada iteración, los parámetros de factor de aprendizaje y el radio de vecindad son reducidos gradualmente de acuerdo a un esquema preestablecido. El criterio de término usualmente se asocia a un número máximo de iteraciones o a una medición de error. Los autores de [8] resumen las mediciones de error más usadas en el contexto de SOM.

2.4. TTOSOM

En [3] se propuso el algoritmo de Tree-based Topology Oriented SOM, en el cual la red neuronal utiliza una topología de árbol arbitrario con el objetivo de identificar la distribución subyacente en los datos analizados.

Para usar TTOSOM, se debe contar con anterioridad con la definición de la topología del árbol. El usuario tiene la capacidad de generar este árbol, lo que le permite reflejar el conocimiento a priori sobre la estructura de la distribución de los datos. Para esto se utiliza un arreglo en donde el elemento i corresponde al i-ésimo nodo del árbol recorriendo este en profundidad, específicamente en preorden. El valor encontrado en esta posición corresponde al número de hijos de este nodo i. En la siguiente Figura 2.2 se observa la representación de un árbol generado con el siguiente arreglo [3, 0, 4, 0, 2, 0, 0, 0, 0, 2, 0, 2, 0, 0].

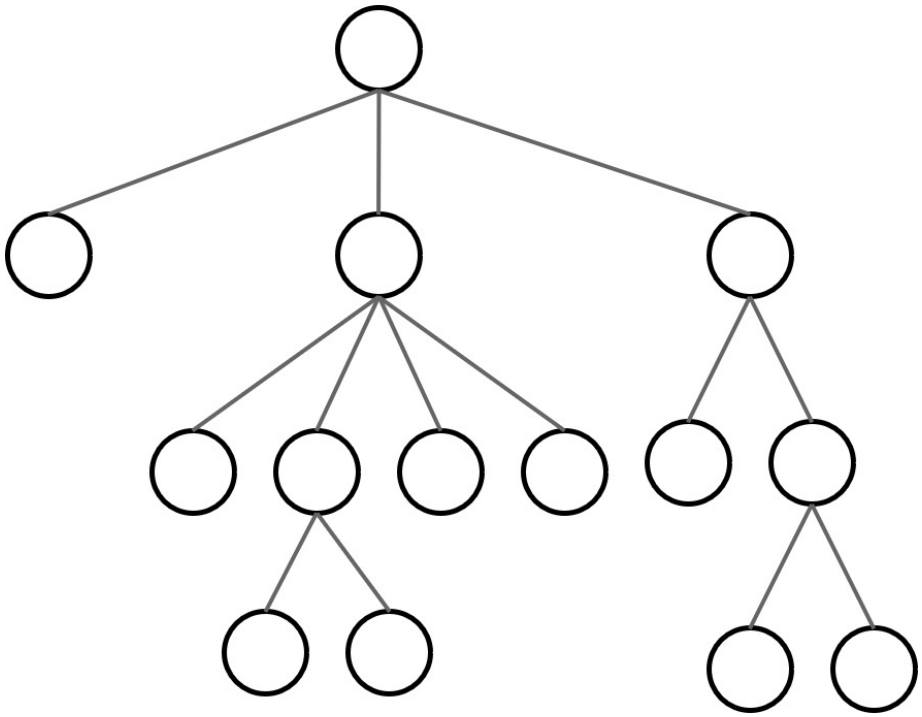


Figura 2.2: Árbol generado con TTOSOM.

En esta figura se observa la generación del árbol recorriendo éste en pre-orden, lo que significa que se procesa primero la raíz, luego el hijo izquierdo y por último los hijos derechos.

La configuración arbitraria de la topología de árbol genera una inmensa diversidad de estructuras posibles, esto no quiere decir que cualquier configuración puede ser aceptada, ya que hay que considerar esta estructura para para una búsqueda rápida del BMU y su vecindad.

Para buscar del BMU, se calcula la distancia $d(*,*)$, definiendo una función de distancia entre dos instancias no especificada explícitamente. La neurona ganadora es identificada luego de analizar todas las neuronas con el estímulo con la función recién descrita. Esta distancia se refiere en términos del espacio y no como distancia neuronal.

Para determinar la distancia neuronal existente entre dos neuronas, se utiliza el

número de arista en la ruta más corta para llegar de una neurona a otra. Con esto se determina que la distancia de una neurona a sí misma es cero y para su padre e hijos directos es uno. Para determinar la vecindad se utiliza esta distancia desde el BMU, en la siguiente Figura 2.3 se puede comprender esta técnica.

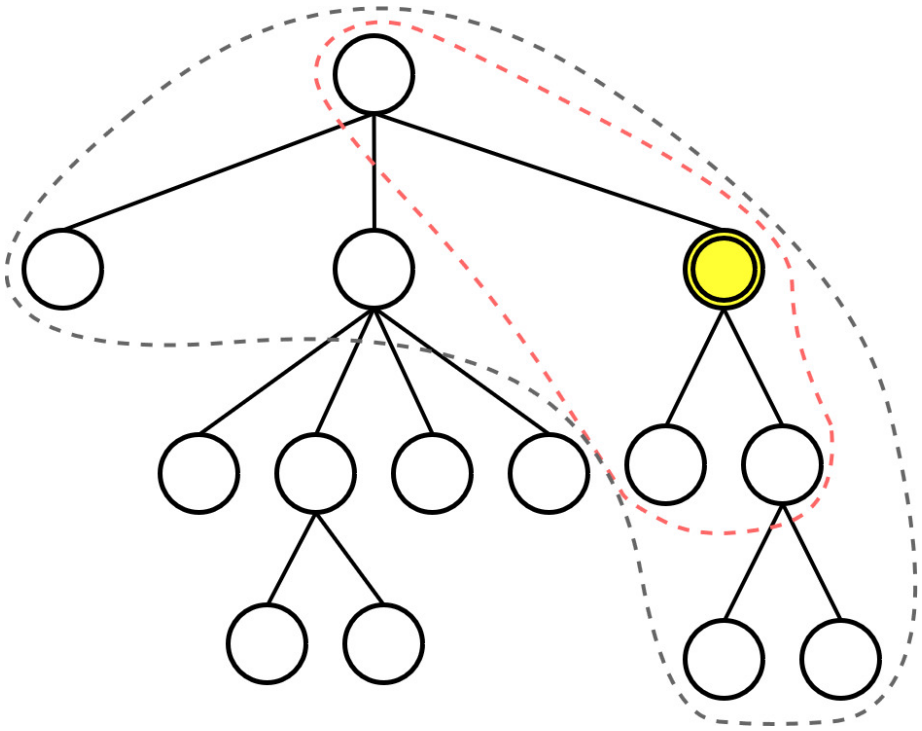


Figura 2.3: Representación de vecindad en TTOSOM.

La Figura 2.3 presenta la distancia entre las neuronas y el BMU, Éste último esta dibujado con un circulo amarillo y los círculos blancos son otras neuronas interconectadas por aristas negras. Las líneas punteadas roja y gris encierran las neuronas que se encuentran a radio uno y dos respectivamente del BMU.

Con lo anterior ya mencionado, se puede comprender el entrenamiento que se realiza. Resumiendo todo lo mencionado en las siguientes etapas: Primero se selecciona una instancia de forma aleatoria, segundo se encuentra el BMU, tercero se calcula la vecindad y cuarto se migra el BMU y los vecinos identificados hacia la instancia para terminar reduciendo el factor de aprendizaje y el radio de vecindad. En el Algoritmo

1 se detalla este entrenamiento. Además en [2] se puede encontrar el algoritmo y la definición específica.

Algoritmo 1 TTOSOM (X,*topologia*,α, *radio*,n)

Entrada:

X, datos de entrenamiento.

topologia, vector que describe la topología del árbol.

α, factor de aprendizaje.

radio, factor de vecindad.

n, numero total de iteraciones.

Salida:

T árbol que contiene los pesos de cada neurona

Método:

1: Inicializa el árbol T
2: **while** no se cumplan las iteraciones **do**
3: Seleccionar un elemento de X
4: Buscar el BMU
5: Identificar la vecindad
6: Migrar BMU y vecindad
7: Actualizar el factor de aprendizaje y vecindad
return T

En [4] , los autores extienden la propiedades de TTTOSOM para la clasificación semi-supervisada. Este tipo de clasificación utiliza datos con etiqueta y sin etiquetar de manera simultanea. Típicamente se asume que los datos sin etiquetar son mas abundantes. El algoritmo especificado en [4] opera en dos etapas: en la primera etapa se realiza clustering de datos utilizando los datos sin etiqueta, y en una segunda fase, se usan los datos etiquetados para conocer la identidad de las neuronas basados en un esquema de votación simple. La predicción de una instancia se efectúa a través de la asociación de la etiqueta del BMU respectivo.

La definición de la topología obliga al usuario a determinar una configuración inicial, para lo cual se requiere alguna información sobre la distribución de estos datos para una buena elección. Para mejorar esta configuración se define el algoritmo

TTOSOMCONROT [7]. Se plantea una propiedad auto organizativa estructural, que permite cambiar la configuración inicial de la topología. Para esto se tiene la noción de promoción neuronal, que mueve las neuronas localmente, reduciendo la longitud de la ruta ponderada. Esto se realiza guardando la información estadística del acceso a cada nodo. Esta rotación solo se realiza cuando se determina una menor longitud de ruta ponderada de la nueva topología para la red neuronal.

Para analizar el funcionamiento de TTOSOMCONROT, se comparó el funcionamiento experimental con diferentes clasificadores del estado del arte que utilizan el aprendizaje supervisado donde se comprueba que en ciertas condiciones puede tener resultados comparativamente mejores [5]. Además, los autores de [5] muestran que un esquema semi-supervisado como TTOSOM puede entregar resultados comparables con los métodos supervisados. Donde una red neuronal de pequeño tamaño en comparación a la cardinalidad del conjunto de entrada es capaz de reportar buenos resultados semejantes a los otros métodos.

En el artículo [10] se presenta la utilización de TTOSOM para un análisis del mercado bursátil italiano, donde se analizan 206 activos del mercado de valores. Este análisis tiene como propósito entender y comprender las relaciones entre varias compañías comerciales, agrupando las diferentes empresas según su comportamiento, lo que facilita la toma decisiones al proveer mayor información de los distintos sectores económico. La técnica presentada en [5] tiene el potencial de predecir y adelantarse al cambio del mercado, cualidad que es de gran valor para un inversionista.

En [16], se presenta una mecanismo basado en TTOSOM que realiza aprendizaje semi-supervisado para la predicción de la estabilidad de secuencias de aminoácidos. Los autores de [16] examinan un problema de dicotomización, donde las mutantes se consideran estables o inestables. El estudio concluye que es posible aprender de los datos usando el paradigma semi-supervisado y que se pueden obtener resultados adecuados incluso cuando se utilizan se conoce la estabilidad de muy pocas mutantes.

2.5. Visualización

2.5.1. Gráficos de dispersión

Un gráfico de dispersión es una representación de un conjunto de datos sobre un diagrama matemático que utiliza las coordenadas cartesianas, mostrando cada dato como un punto en el plano el cual puede ser en uno, dos o tres dimensiones.

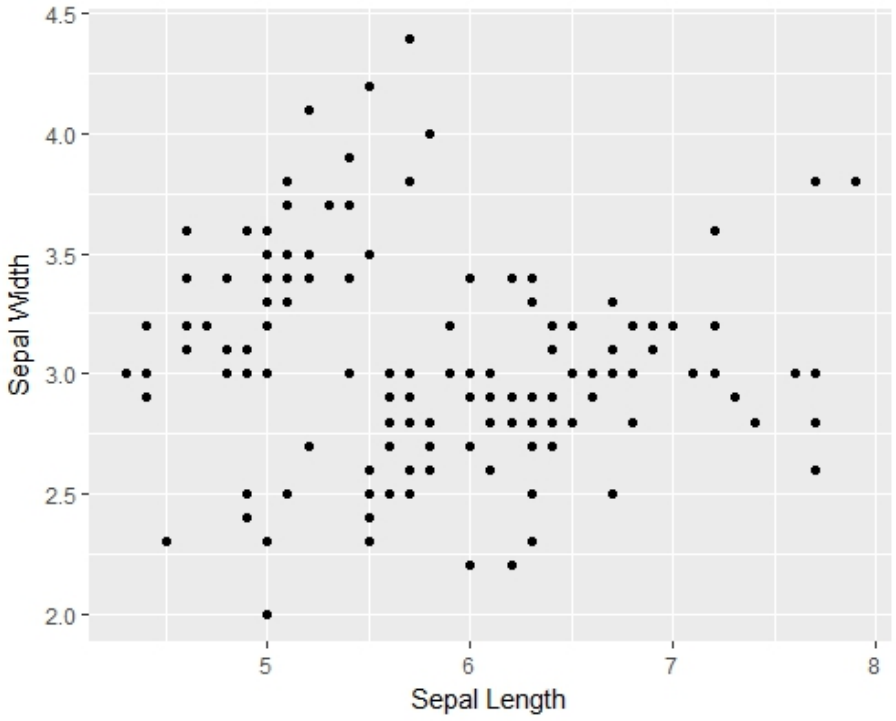


Figura 2.4: Ejemplo gráfico de dispersión.

2.5.2. Visualización del SOM

La visualización de la malla generada con SOM es difícil de comprender, y dependiendo de la dimensionalidad de ésta, la complejidad aumenta, por esto esté método de visualización es mayormente utilizado en el proceso de desarrollo del algoritmo, para poder ver y analizar el entrenamiento y como afectan los diferentes configuraciones de los parámetros en este entrenamiento.

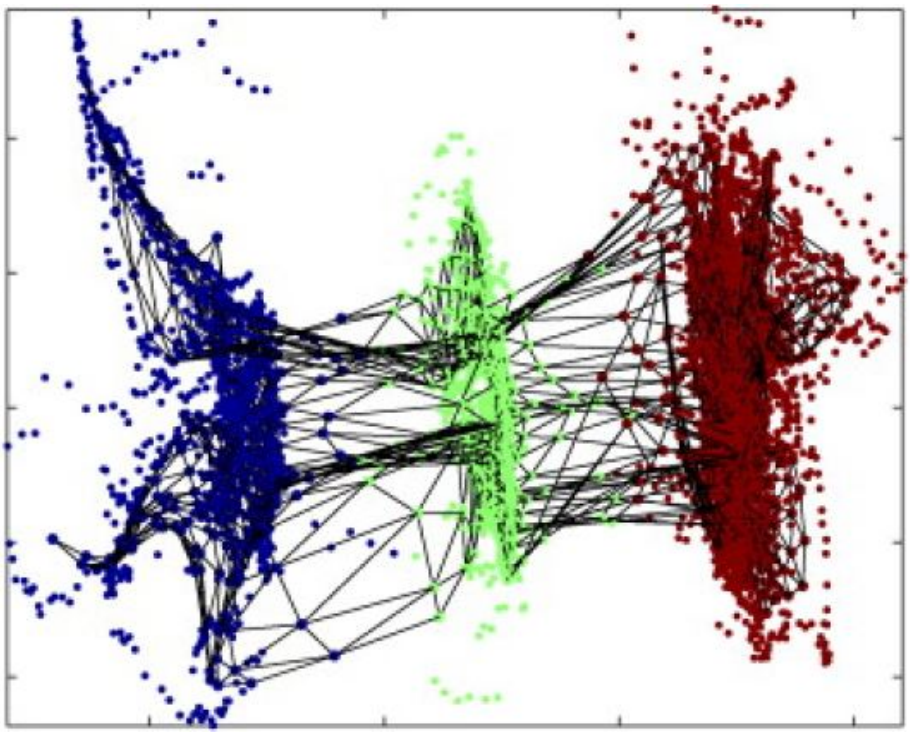
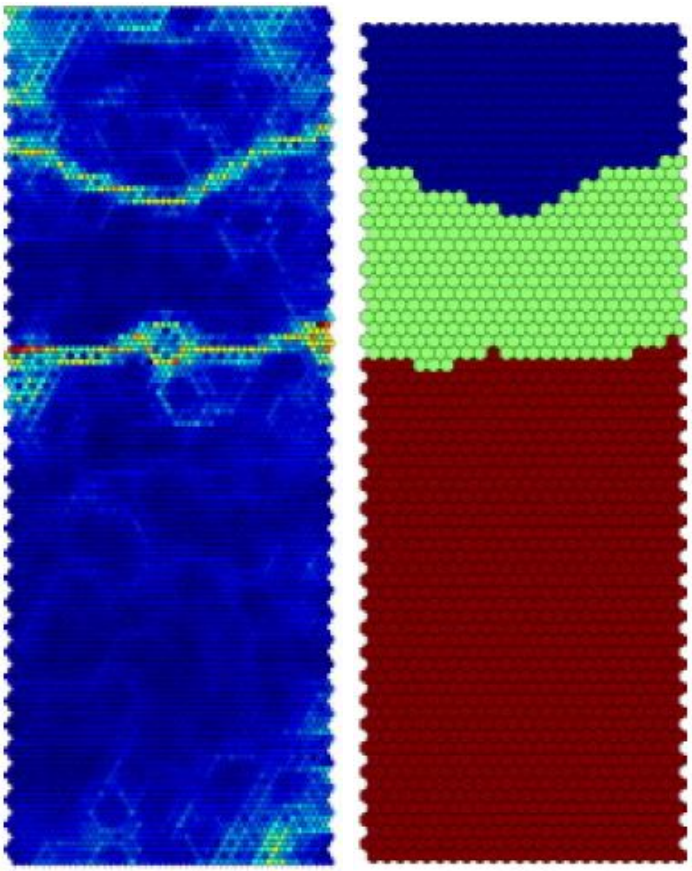


Figura 2.5: Ejemplo de visualización de la malla SOM junto a los datos.

2.5.3. U-matrix

El unified distance matrix, que se podría traducir como matriz de distancia unificada, es la forma más común de representar el SOM, donde las distancias entre las neuronas se pueden representar con colores. En la mayoría de los casos se puede identificar a simple vista líneas de un color distinto al que predomina en la imagen lo que identifica que en esa sección existe una mayor distancia entre neurona y nos ofrece los bordes que dividen los datos analizados, por lo tanto, esto permite visualizar zonas donde se encuentran los grupos de datos, separados por líneas, esto es útil para poder identificar los grupos a simple vista cuando no se tiene ninguna información sobre esto.



(a) (b)

Figura 2.6: Ejemplo de visualización de SOM usando (a) U-matrix, (b) la asociación de grupo.

2.6. Método propuesto

2.6.1. Complete k-ary Tree SOM (CKTSOM)

El algoritmo propuesto se llama Complete k-ary Tree Self-Organizing Maps (CKTSOM). Este es una variante del algoritmo Self-Organizing Maps que al igual que TTOSOM utiliza una estructura de árbol para interconectar las neuronas. La literatura reporta otros algoritmos basados en SOM que utilizan una estructura de árbol, que se encuentran descritos en [8].

Para iniciar con el algoritmo se define la altura y cantidad de hijos por nodo. Estos parámetros son la base para construir el árbol completo, esto significa que todos los hijos deben existir, la finalidad de esto es utilizar una estructura computacional simple que tenga la capacidad de representar el árbol.

Para la representación del árbol se utiliza un vector que contiene la información de cada neurona, y con el índice de éste se genera el árbol usando cálculos matemáticos para identificar su padre o sus hijos. Para ejemplificar lo mencionado, en la Figura 2.7 se presenta un árbol generado con una altura (h) 3 y 2 hijos por nodo (k).

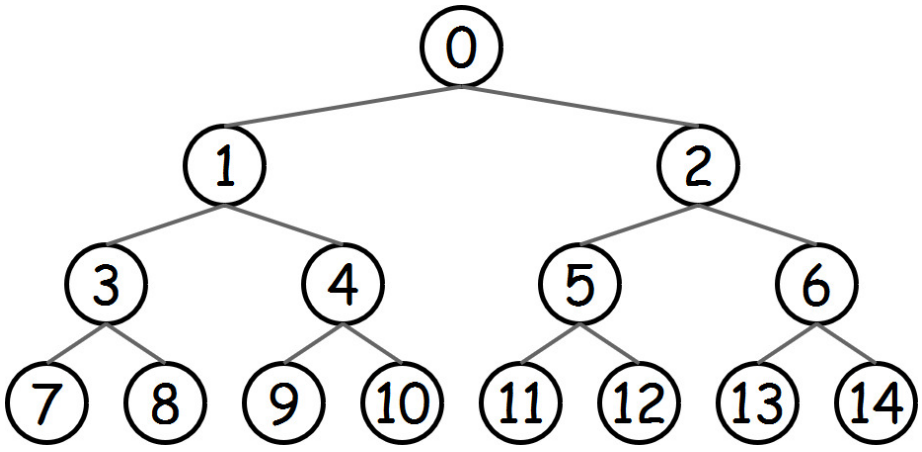


Figura 2.7: Árbol completo con $h = 3$ y $k = 2$.

Con la Figura 2.7 se representa la estructura de árbol donde el número de cada nodo marca su posición en el vector que lo almacena. A continuación se presentan

las ecuaciones (2.3) y (2.4) que buscan el padre o los hijos respectivamente de una neurona usando el índice de esta.

$$padre\,(neurona) = \frac{(neurona + k - 1)}{k} - 1 \tag{2.3}$$

La ecuación (2.3) muestra cómo encontrar al padre de una neurona en particular. Del resultado obtenido se utiliza la parte entera ya que es donde está el índice del padre buscado.

$$hijoMayor\,(neurona) = (neurona * k) + 1 \tag{2.4}$$

Con esta ecuación (2.4) se encuentra el primer hijo de una neurona. Con este resultado se puede obtener los demás hijos que se encontrarán consecutivos al primer hijo encontrado, siendo el último en la posición $hijoMayor\,(neurona) + k - 1$.

Hay que destacar que en esta propuesta no todos los nodos de la red neuronal son elegibles para ser denominados BMU, en este caso, solo los nodos hojas son elegibles como tal, como ejemplo en la Figura 2.7 los nodos del 7 al 14 son seleccionables a BMU. Para identificar esta neurona se utiliza la búsqueda en profundidad partiendo de la raíz hasta la hoja. Esto significa que se inicia en la raíz denominándola temporalmente como el BMU, luego se revisan los hijos directos, para buscar entre estos la neurona más cercana al estímulo para denominar a la neurona encontrada como nuevo BMU, este proceso se sigue repitiendo hasta llegar a un nodo hoja denominando a este último como BMU. Esto permite disminuir el costo computacional al tener un orden logarítmico en la búsqueda de la neurona ganadora.

Para determinar el vecindario o área de acción, se inicia desde el BMU que en este caso siempre será un nodo hoja, donde en sí mismo se encuentra a una distancia 0, sus hijos y padre directo a una distancia 1, este proceso es muy parecido al utilizado en TTOSOM [2].

Con el BMU identificado se iniciar el proceso de migración, donde esta neurona absorbe toda la información definida por el factor de aprendizaje, el padre absorbe

un 90 % de esta información y su hermano directo un 20 %, esto genera que a medida que la información sube hasta la raíz, la cantidad de información absorbida por cada neurona es menor. Este proceso se repite mientras el aprendizaje sube hasta la raíz o hasta que queda fuera de la vecindad.

La decisión de reducir la información aprendida por los hermanos fue tomada para que estos se dispersen, de lo contrario se genera que los hermanos sean prácticamente iguales. Esto se puede apreciar en el Anexo B donde se presentan imágenes de entrenamientos con diferentes porcentajes de absorción de la información.

Con la migración terminada se inicia el proceso de actualizar el factor de aprendizaje y el radio, en el Algoritmo 2 se muestra el funcionamiento CKTSOM.

Algoritmo 2 CKT-SOM($\mathcal{X}$,k,h, α_i, α_f,r_i,r_f,T)

Entrada:

$\mathcal{X}$, colección de datos $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \mathbb{R}^d$.

k, número de hijos por nodo.

h, altura del árbol.

α_i, α_f, factor de aprendizaje inicial y final, respectivamente.

r_i, r_f, radio de la vecindad inicial y final, respectivamente.

T, número total de iteraciones.

Salida:

A, Colección de neuronas $\{\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{m-1}\}$, donde $\mathbf{a}_i \in \mathbb{R}^d$ corresponde al vector de peso de las neuronas i.

Método:

$\alpha \leftarrow \alpha_f - \alpha_i$; $\epsilon_\alpha \leftarrow \alpha/T$; $r \leftarrow r_f - r_i$; $\epsilon_r \leftarrow r/T$

$m \leftarrow \sum_{i=0}^{h} k^h$ ▷ Número total de neuronas en el árbol k-ario completo.

for $i \leftarrow 0$ para $m-1$ **do** ▷ Por cada neurona

    $\mathbf{a}_i \leftarrow$ random-select($\mathcal{X}$) ▷ Inicializa la neurona con una entrada aleatoria.

for $t \leftarrow 1$ to T **do** ▷ Realizar T iteraciones de entrenamiento.

    $\mathbf{x} \leftarrow$ random-select($\mathcal{X}$) ▷ Escoger la entrada para ser procesada.

    $s \leftarrow 0$ ▷ El BMU actual es la raíz.

    $u \leftarrow m - k^h$ ▷ Índice del último nodo no hoja.

    while $s \leq u$ **do** ▷ Búsqueda logarítmica del árbol.

        $j \leftarrow k(s+1) - (k+1)$ ▷ Índice del actual hijo mayor.

        $s \leftarrow \text{argmin}_{b \in [j, j+(k-1)]}\, \phi(x, \mathbf{a}_b)$ ▷ Encontrar al mejor hijo.

    $\alpha_t \leftarrow \alpha$; $r_t \leftarrow r$

    while $s > 0$ y $r_t > 0$ **do** ▷ Actualización de la rama del árbol.

        $\mathbf{a}_s \leftarrow \mathbf{a}_s - \alpha_t(\mathbf{a}_s - \mathbf{x})$ ▷ Regla de actualización de SOM.

        $p \leftarrow ((s+k-1)/h) - 1$ ▷ Obtener el índice del padre.

        $\mathbf{a}_v \leftarrow \mathbf{a}_v - 0.2\alpha_t(\mathbf{a}_v - \mathbf{x})$, $v \in \{kp+1, \ldots, kp+k\}, v \neq s$ ▷ Actualizar hermanos.

        $r_t \leftarrow r_t - 1$; $\alpha_t \leftarrow 0.9\alpha_t$; $s \leftarrow p$

    $\mathbf{a}_p \leftarrow \mathbf{a}_p - \alpha_t(\mathbf{a}_p - \mathbf{x})$, $r_t > 0$ ▷ Actualizar la raíz.

    $\alpha \leftarrow \alpha - \epsilon_\alpha$; $r \leftarrow r - \epsilon_r$ ▷ Actualizar factor de aprendizaje y radio.

2.6.2. Exactitud búsqueda de BMU

La búsqueda del BMU es un proceso que se realiza en cada iteración, por lo tanto, se puede considerar como un procedimiento costoso para el algoritmo. Podemos mencionar al SOM donde todas las neuronas compiten entre sí para representar al estímulo, esto nos proporciona al mejor candidato a ser BMU, pero nos genera un costo en el tiempo que requiere este tipo de búsqueda [8]. Es por esto que para este algoritmo se utiliza una búsqueda en profundidad, lo que nos proporciona un resultado aproximado. A continuación se presentan las pruebas que verifican el funcionamiento de la búsqueda en profundidad y la calidad de esta respuesta.

Para comprobar qué tan buena fue la elección de la búsqueda en profundidad, a continuación se presenta la experimentación que se divide en dos partes. La primera tiene como objetivo comprender qué tan precisa es la elección del BMU con el algoritmo de búsqueda en profundidad, mediante el cual se comparará el resultado con una búsqueda exhaustiva para confeccionar el porcentaje de exactitud. Esto se realizará con distintos conjuntos de datos y en distintas iteraciones del algoritmo CKTSOM, ya que se sospecha que mientras la red neuronal esté mejor entrenada la búsqueda en profundidad tendrá una respuesta óptima. La segunda etapa consistirá en analizar los casos en que la búsqueda en profundidad no entregue el mejor resultado, analizando qué tan alejado o cercano se encuentra este resultado del óptimo, los conjuntos de datos serán normalizados para poder analizar esta distancia entre los distintos conjuntos de datos.

Los datos utilizados se encuentran en el repositorio del Profesor David Tax [19]. Estos datos originalmente son del repositorio de UCI Maching Learning [17] pero se prefiere trabajar con los datos proporcionados por el profesor ya que estos se encuentran tratados para no tener valores nulos. Estos datos tienen distintas distribuciones y dimensiones, para poder obtener una idea general del funcionamiento y qué tan bueno es el desempeño de esta búsqueda. A continuación en el Cuadro 2.1 se presenta el conjunto de datos utilizados

Conjunto de dato	Dimensión	Instancias
Biomed	5	194
Liver	6	345
Ecoli	7	336
Diabetes	8	768
Breast	9	699
Abalone	10	4177
Spectf	44	349
Sonar	60	208
Arrhythmia	278	420
Concordia	1024	4000
Colon	1908	66

Cuadro 2.1: Conjunto de datos.

Para iniciar esta prueba se definió para todos los conjuntos de datos, los mismos parámetros de entrenamiento.

```
Factor de aprendizaje inicial: 1
Factor de aprendizaje final: 0
Radio inicial: 7
Radio final: 1
Hijos por nodo: 3
Altura del árbol: 3
```

En la primera etapa, se entrenó la red neuronal utilizando diferentes iteraciones para luego buscar el BMU de cada uno de los datos comprendido en este conjunto. Esto se realizó utilizando la búsqueda en profundidad y el método exacto, esto ultimo significa que todos las neuronas hojas fueron revisadas. Estos resultados se compararon para luego calcular el porcentaje de similitud, siendo 1 cuando la búsqueda en profundidad respondió siempre con el mejor resultado y 0 cuando esta búsqueda no reportó el resultado óptimo. A continuación en el Cuadro 2.2 se presentan estos resultados.

Esto se realizó generando dos vectores que correspondían al resultado generado por la búsqueda exacta y en profundidad, luego se comparó los elementos de cada vector con igual índice, obteniendo 1 si se obtenía el mismo resultado y 0 en caso contrario. Los valores obtenidos son promediados para obtener el porcentaje de acierto.

	Iteraciones					
Conjunto de dato	1	10	100	1k	10k	100k
Biomed	0.20	0.31	0.53	0.85	0.93	0.95
Liver	0.16	0.30	0.33	0.79	0.90	0.93
Ecoli	0.29	0.36	0.64	0.87	0.96	0.96
Diabetes	0.10	0.15	0.51	0.81	0.88	0.92
Breast	0.20	0.16	0.66	0.84	0.92	0.94
Abalone	0.12	0.33	0.73	0.92	0.93	0.93
Spectf	0.19	0.37	0.41	0.61	0.91	0.95
Sonar	0.15	0.19	0.60	0.80	0.92	0.95
Arrhythmia	0.08	0.27	0.51	0.59v	0.83	0.92
Concordia	0.14	0.42	0.53	0.65	0.83	0.85
Colon	0.15	0.27	0.58	0.74	1.00	1.00

Cuadro 2.2: Porcentaje de veces que se encuentra la BMU óptima.

Con el Cuadro 2.2 se puede comprobar que en etapas tempranas el porcentaje de acierto es relativamente bajo, pero a medida que las iteraciones aumentan este resultado se incrementa, llegando, en la mayoría de los casos sobre el 90 %, con lo que podemos comprender que esta respuesta es aceptable.

Para la segunda etapa queda analizar los resultados de la búsqueda en profundidad, en donde esta respuesta no fue la óptima, para esto se normalizaron los datos entre 0 y 1, de esta forma comprender las distancias en términos de similitud donde los resultados más cercanos a 0 significa que la distancia fue mínima, en cambio, a resultados cercanos $\sqrt{dimension}$ podemos comprender que esta distancia es demasiada y que la búsqueda en profundidad nos proporcionó la peor solución. La distancia fue calculada entre las BMU obtenidas por el método exacto y la búsqueda

en profundidad. A continuación se presenta el Cuadro 2.3 en donde se muestra la distancia promedio por cada iteración.

Este peor resultado es determinado por la distancia euclidiana que se está utilizando, ya que queremos determinar la distancia entre dos puntos A y $B \in \mathbb{R}^d$, los cuales pueden tener valores entre 0 y 1. El peor caso nos deja el momento en que el punto A se encuentra en el origen y B tiene todas sus dimensiones en la posición 1, obtenemos que la distancia entre A y B es $\sqrt{d}$.

		Iteraciones					
Conjunto de dato	$\sqrt{d}$	1	10	100	1k	10k	100k
Biomed	2.24	1.08	0.95	0.72	0.34	0.26	0.19
Liver	2.45	1.05	0.92	0.99	0.48	0.34	0.28
Ecoli	2.65	1.15	1.05	0.70	0.51	0.20	0.24
Diabetes	2.83	1.44	1.46	1.04	0.60	0.50	0.44
Breast	3.00	1.44	1.48	0.93	0.55	0.49	0.38
Abalone	3.16	1.80	1.49	1.06	0.46	0.43	0.38
Spectf	6.63	2.61	2.28	2.30	1.94	0.83	0.64
Sonar	7.75	3.77	3.63	2.61	1.67	1.05	0.93
Arrhythmia	16.61	6.25	5.29	4.88	4.89	3.19	2.02
Concordia	32	16.78	13.41	11.32	9.63	6.84	6.56
Colon	43.68	21.14	18.54	14.96	11.51	0.00	0.00

Cuadro 2.3: Distancia promedio hasta BMU.

Con el Cuadro 2.3 podemos analizar los casos en los cuales la BMU reportada por la búsqueda en profundidad no fue la mejor elección, con la distancia relativa proporcionada se puede inferir que aunque la repuesta no sea la óptima ésta será cercana a lo esperado, por lo tanto nos confirma que la búsqueda por profundidad fue una buena decisión.

3. Clasificación

Este capítulo detalla los conceptos utilizados para definir un clasificador, específicamente un clasificador de una clase.

Si tuviéramos un grupo de perros y gatos, y nos pidieran separar los perros de este grupo, tenderíamos a pensar que es una acción sencilla, ya que todos podemos diferenciarlos y clasificarlos instantáneamente. Sin embargo, este proceso se complica si lo queremos hacer de forma automática, dado que, si solo quisiéramos los perros, no tendríamos una característica en particular que los identificara, es decir, tendríamos más de una, como lo es color del pelaje, el tipo de pelo, el olor, entre otros. Por otra parte, si se da el caso de que los perros estuvieran sucios, no podríamos usar estos aspectos directamente en la máquina. Otro caso aún más extraño, es lo que sucede con un peluche, ya que la maquina lo reconocería por su forma, color y/o pelaje, pero estaría cometiendo un error al reconocer como ser vivo algo que no lo es. Por estas razones, es que la clasificación automática tiene una elevada complejidad, la cual aumenta dependiendo de qué se desea clasificar, dado que para determinar si un elemento pertenece a una clase en particular, se requiere un conjunto de características que son procesadas de forma automática para nosotros, pero que, en la mayoría de los casos, no somos capaces de describir en su totalidad, las que además nos permiten determinar a qué grupo pertenece lo que estamos analizando [20].

Para clasificar, usamos el concepto de "objeto", pero esta palabra es amplia, dado que puede abarcar conjuntos de perros, gatos, plantas, sonidos de animales, notas musicales, entre otros. Por esto, en este estudio, objeto será definido como aquel que puede ser representado por una colección de números. Por lo tanto, un objeto será

representado mediante el vector $x = (x_1, x_2, x_3, ..., x_d), x_i \in \mathbb{R}$.Además, cada valor del vector será conocido y tenderá a no tener valores faltantes. Sin embargo, es posible que en ocasiones existan estos tipos de valores en las medidas de un objeto en particular, al cual no se le realizaron por su elevado costo, esfuerzo o relevancia. En esta experimentación, se supondrá que todos los elementos tendrán todas sus características, evitando las complicaciones adicionales que implican valores nulos [20].

3.1. One-Class Cassification

One-Class Classification (Clasificación de una clase) corresponde a un clasificador que solo conoce elementos pertenecientes al grupo objetivo, es decir, debe determinar de un conjunto de entrada, que instancias son las que pertenecen a este grupo, o de lo contrario, identificarla como un objeto extraño. Este tipo de clasificador, solo tiene información de la clase objetivo, lo que implica que desconoce toda la información de los objetos extraños en la etapa de entrenamiento. En resumen, la meta es identificar la mayor cantidad de elementos objetivos, minimizando la posibilidad de aceptar objetos exóticos. En la Figura 3.1 se presenta un ejemplo de clasificador de clase, donde se realiza el entrenamiento con los datos objetivos, sin presencia de datos atípicos. Definiendo con una línea donde se encuentran los datos objetivos y fuera de esta, donde están los objetos extraños [20].

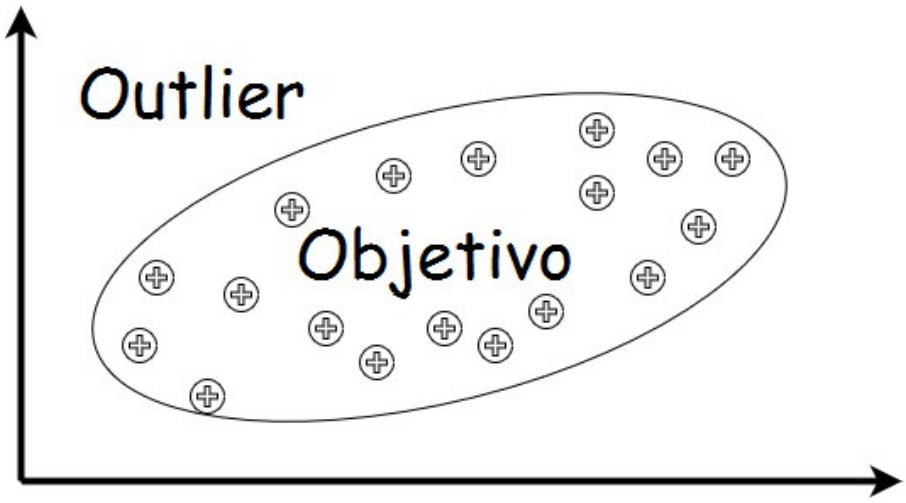


Figura 3.1: Ejemplo de clasificador.

Formalizando lo anterior mencionado, tendremos un conjunto de entrenamiento $X = (X_1, X_2, ..., X_n), X_i \in \mathbb{R}^d$, donde se desea buscar un clasificador de una clase, capaz de caracterizar los datos de entrenamiento que contienen solo a la clase objetivo. Por lo general, un clasificador de una clase se define de la siguiente manera [13].

$$h(x|X,\gamma) = \varphi(d(x|X,\gamma) < \theta) = \begin{cases} 1 & \text{si x es clasificado como objetivo.} \\ 0 & \text{si x es clasificado como outlier.} \end{cases} \tag{3.1}$$

donde θ es un umbral de aceptación y $\varphi()$ es la función indicadora. El modelo h se basa en la diferencia entre los datos de entrenamiento X y el objeto representado por el vector x. Además h puede ser utilizado para usar una medida de similitud utilizando el signo opuesto ($>$) en la ecuación (3.1). Por último γ determina la complejidad del modelo h [13].

4. Experimentación

En esta sección, se estudia el desempeño del clasificador OCC propuesto, utilizando AUC como medida de rendimiento. Se utilizaron objetos de la clase objetivo para entrenar el modelo, mientras que la validación se realiza con instancias pertenecientes a la clase objetivo y atípico. Para esto, se utilizó una curva Receiver Operating Characteristic (ROC). Esta es una función que utiliza el subconjunto de instancias reconocidas como pertenecientes a la clase objetivo para identificar los datos atípicos contenidos en los datos. Es por esto qué los datos atípicos que son conocidos solo intervienen en la fase de validación y consecuentemente, son excluidos de la fase de entrenamiento.

Se compara el desempeño en términos de AUC del algoritmo basado en CKTSOM con varios clasificadores definidos en el estado del arte [13]. El metodo propuesto se compara con otros 16 métodos, que son: Gauss, Mixture of Gaussian (MoG), Naïve Parzen, Parzen, k-Means, 1-Nearest Neighbour (1-NN), k-Nearest Neighbors (k-NN), Auto-Encoder, Principal Component Analysis (PCA), Self-Organizing Map (SOM), Minimum Spanning Tree Class Descriptor (MST_CD), k-Centres, Support Vector Domain Description (SVDD), Minimax Probability Machine (MPM), Linear Programming Dissimilarity-data Description (LPDD) y Chameleon. Entre más alto el valor del AUC, mejor es el rendimiento del algoritmo. Un AUC menor que 0.5 indica que el clasificador es peor que una elección aleatoria.

Para la realización del experimento se utilizaron los conjuntos de datos nombrados en el Cuadro 2.1. Para esto, se tomaron datos pertenecientes a la clase objetivo y se dividieron aleatoriamente en dos partes iguales para el entrenamiento y las prue-

bas. Todos los clasificadores fueron entrenados con el subconjunto de entrenamiento. Durante la fase de validación, se utilizó datos de la clase objetivo no usados en el entrenamiento y aquellos identificados como atípicos (datos no-objetivo). Este proceso se repitió 20 veces, reportando el promedio.

Los conjuntos de datos corresponden a problemas de clasificación múltiple, es decir, que contienen más de dos categorías, fueron transformados en problemas de clasificación binaria. Para esto, se designó una clase como objetivo, asumiendo que las demás corresponden a observaciones atípicas.

4.1. Implementación de prueba con CKTSOM

Las pruebas y el cálculo de AUC por cada conjunto de datos, se describen en la siguiente sección.

Para iniciar las pruebas se tomaron los datos objetivos y se dividieron aleatoriamente en partes iguales para el entrenamiento y las pruebas. En el entrenamiento se limpió este conjunto para obtener solo los datos objetivos con los cuales se entrenó el árbol neuronal T. La red neuronal es preparada con estos datos, usando diferentes parámetros de configuración del algoritmo.

Cuando el entrenamiento converge se inicia con los cálculos de estadística, para este fin se mide la distancia de cada dato hasta su BMU, para determinar la distancia promedio y la desviación estándar. Para esto se usa la ecuación root-mean-square deviation (RMSE) que en español sería error cuadrático medio que se presenta a continuación en la ecuación (4.1)

$$RMSE = \sqrt{\frac{1}{m} * \sum d(x_i, s_i)} \tag{4.1}$$

siendo m la cantidad de datos, x una instancia y s su BMU en particular. Al completar la etapa de entrenamiento se obtiene finalmente una tupla (T, μ, σ) que corresponde al arbol neuronal T, el $RMSE$ y la desviación estándar.

Para la fase de validación se utiliza el subconjunto de prueba. Para definir si un valor de este conjunto se considera como objetivo o un outlier, se utiliza un puntaje llamado Z-score [15] también conocido como standard score que podría ser traducido a puntuación estándar. Ésta es una medida numérica que relaciona un dato con una medida de un conjunto de éstos, utilizando la desviación estándar para determinar qué tan símil es. Este valor puede ser positivo o negativo, dependiendo donde esté el dato analizado. En este caso se usa el valor absoluto, ya que esta respuesta se analizará como una medida de distancia. El Z-score típicamente se calcula usando la ecuación (4.2).

$$Z_i = \frac{|d(x_i, s_i) - \mu|}{\sigma} \tag{4.2}$$

Con este valor obtenido se utiliza un nuevo parámetro θ que especifica las desviaciones estándar de la medida. Para esto se comparó con un umbral de 30 valores diferentes de θ, que varían entre 0,1 y 3,0. Luego si el Z-score obtenido de un dato es mayor que el θ este dato es considerado como outlier.

Con los resultados calculados por el clasificador y las respuestas previstas obtenidas de antemano, son utilizadas para generar una matriz de confusión. Esta es una tabla que contiene la información de las clases verdaderas y predichas, para el conjunto de elementos etiquetados. En el Cuadro 4.1, se encuentran el T_p y T_n siendo el número de verdaderos positivos y verdaderos negativos respectivamente, F_p y F_n corresponde al número de falso positivo y falso negativo respectivamente.

	Valor predicho		
Valor real	$-ve$	$+ve$	
$-ve$	T_n	F_p	C_n
$+ve$	F_n	T_p	C_p
	R_n	R_p	N

Cuadro 4.1: Matriz de confusión.

En esta tabla las filas C_n y C_p es el número de ejemplos real negativo y positivo. El total de las columnas son R_n y R_p es el número de ejemplos de predichos nega-

tivos y positivos, por ultimo N es el total de ejemplos, con esta matriz es posible extraer medidas significativas que expresan criterios de rendimiento, a continuación se presentan algunos de estos.

$$Exactitud(1 - Error) = \frac{T_p + T_n}{C_p + C_n} = P(C), \tag{4.3}$$

$$Sensibilidad(1 - \beta) = \frac{T_p}{C_p} = P(T_p), \tag{4.4}$$

$$Especificidad(1 - \alpha) = \frac{T_n}{C_n} = P(T_n), \tag{4.5}$$

$$Valor\ positivo\ predictivo = \frac{T_p}{R_p}, \tag{4.6}$$

$$Valor\ negativo\ predictivo = \frac{T_n}{R_n}. \tag{4.7}$$

Cuando se han obtenido varios puntos en la curva ROC $[P(F_p = \alpha, P(T_p) = 1-\beta]$ se puede obtener el área bajo la curva ROC usando la integración trapezoidal, que se expresa a continuación.

$$AUC = \sum_i \left\{ (1 - \beta_i \cdot \Delta\alpha) + \frac{1}{2}[\Delta(1 - \beta) \cdot \Delta\alpha] \right\}, \tag{4.8}$$

donde

$$\Delta(1 - \beta) = (1 - \beta_i) - (1 - \beta_{i-1}), \tag{4.9}$$

$$\Delta\alpha = \alpha_i - \alpha_{i-1}. \tag{4.10}$$

4.2. Resultados

A continuación se presentan las tablas comparativa con distintos algoritmos, además la procedencia de las puntaciones de los demás algoritmos.

Los algoritmos utilizados y su puntuación AUC serán obtenidos del artículo llamado Minimum spanning tree based one-class classifier [13]. En este artículo se propone un algoritmo llamado minimum spanning tree class descriptor (MST_CD) que al igual que CKTSOM, es un algoritmo que puede ser utilizado para clasificaciones de una clase.

El origen de los datos a analizar están disponible en la página web del profesor David Tax [19]. Para la clasificación de una clase se estableció que un grupo de datos será la clase objetivo y el resto son elementos atípicos o outliers.

En los Cuadros 4.2 y 4.3 se compara el rendimiento de CKTSOM con los demás algoritmos utilizando la medida AUC. Para esto la clase objetivo se dividió aleatoriamente en partes iguales para el entrenamiento y las pruebas. Todos los algoritmos clasificadores se entrenaron con los datos objetivos para luego ser probados con los datos objetivos y no objetivos. Estos experimentos se repitieron 20 veces promediando el resultado obtenido.

A continuación se presenta el Cuadro 4.2 comparativa con conjuntos de datos de baja dimensión, donde se presenta el AUC promedio obtenido junto con su desviación estándar en paréntesis.

Conjunto de dato	Biomed	Liver	Ecoli	Diabetes	Breast	Abalone
Clase objetivo	normal	healthy	periplasm	present	malignant	class 1-8
Objetivo/Outlier	127/67	145/200	52/284	500/268	241/458	1407/2770
Dimensión	5	6	7	8	9	10
Clasificador			AUC			
Gauss	90.0(0.4)	58.6(0.5)	**92.9(0.3)**	**70.5(0.3)**	82.3(0.2)	86.1(0.2)
MoG	91.2(0.9)	60.7(0.6)	**92.0(0.4)**	67.4(0.3)	78.5(1.3)	85.3(0.5)
Naïve Parzen	**93.1(0.2)**	61.4(0.7)	**93.0(0.8)**	67.9(0.3)	**96.5(0.4)**	85.9(0.4)
Parzen	90.0(1.1)	59.0(0.3)	92.2(0.4)	67.6(0.4)	72.3(0.5)	86.3(0.1)
k-Means	87.8(1.2)	57.8(1.0)	89.1(1.6)	65.9(0.7)	84.6(3.5)	79.2(1.1)
1-NN	89.1(0.8)	59.0(0.9)	90.2(0.9)	66.7(0.7)	69.4(0.6)	86.5(0.1)
k-NN	89.1(0.8)	59.0(0.9)	90.2(0.9)	66.7(0.7)	69.4(0.6)	86.5(0.1)
Auto-encoder	85.6(2.2)	56.4(0.9)	87.8(1.0)	59.8(1.8)	38.4(0.9)	82.6(0.3)
PCA	89.7(0.5)	54.9(0.5)	66.9(1.1)	58.7(0.2)	30.3(1.0)	80.2(0.1)
CKTSOM	83.0(2.0)	**64.4(12.7)**	75.0(12.4)	**68.1(2.4)**	77.0(8.2)	63.9(0.8)
SOM	88.7(0.8)	59.6(0.7)	89.0(1.1)	69.2(0.7)	79.0(2.3)	81.4(0.3)
MST_CD	89.8(1.0)	58.0(0.9)	89.7(0.9)	66.9(0.7)	75.6(1.8)	**87.5(0.1)**
k-Centres	87.8(2.4)	53.7(4.1)	86.3(1.2)	60.6(1.6)	71.5(12.4)	76.0(0.8)
SVDD	2.2(0.3)	4.7(1.4)	89.4(0.8)	57.7(9.8)	70.0(0.6)	80.6(0.1)
MPM	79.2(5.7)	58.7(0.9)	80.2(0.5)	65.6(0.7)	69.4(0.6)	59.4(0.1)
LPDD	86.5(2.6)	56.4(2.6)	89.6(0.5)	66.8(0.7)	80.0(0.5)	69.7(0.1)
CHAMELEON	72.7(1.9)	58.0(0.9)	75.8(1.6)	65.1(1.0)	66.9(0.8)	70.6(0.4)

Cuadro 4.2: Cuadro comparativo AUC baja dimensión.

A continuación se presenta el Cuadro 4.3 comparativa con conjuntos de datos de relativamente pequeñas pero con mayor dimensión que en la tabla anterior.

Conjunto de dato	Spectf	Sonar	Arrhythmia	Concordia	Colon
Clase objetivo	0	mines	normal	2	normal
Objetivo/Outlier	95/254	111/97	237/183	400/3600	22/44
Dimensión	44	60	278	1024	1908
Clasificador			AUC		
Gauss	83.3(3.3)	68.0(3.1)	60.6(0.6)	80.3(1.7)	70.4(1.1)
MoG	77.6(3.1)	70.4(3.5)	57.7(16.6)	50(1.1)	50.0(0.0)
Naïve Parzen	90.2(3.7)	53.2(3.9)	77.4(0.7)	84.6(0.7)	70.0(1.5)
Parzen	87.9(2.7)	80.5(3.1)	57.7(16.6)	50.2(2.2)	36.4(22.4)
k-Means	92.3(1.7)	69.8(3.7)	76.6(0.6)	86.2(2.5)	66.8(3.1)
1-NN	92.6(2.9)	76.3(4.3)	76.0(0.8)	90.1(0.8)	71.3(3.3)
k-NN	92.3(1.5)	69.6(4.8)	76.0(0.8)	90.1(0.9)	71.3(3.3)
Auto-encoder	81.7(6.2)	59.6(6.5)	52.2(2.1)	51.2(1.5)	50.0(0.0)
PCA	90.1(3.0)	69.6(3.3)	**80.7(1.0)**	82.4(0.4)	70.7(1.6)
CKTSOM	94.9(0.5)	**84.1(1.1)**	71.0(1.3)	62.9(5.4)	60.9(7.4)
SOM	97.5(2.1)	80.1(3.4)	77.2(0.7)	88.7(2.0)	68.2(2.6)
MST_CD	**98.1(2.6)**	81.1(3.1)	**79.6(0.6)**	91.1(0.1)	**73.3(3.0)**
k-Centres	90.9(1.6)	66.8(4.1)	76.7(1.6)	81.5(3.6)	68.4(2.9)
SVDD	**97.8(3.3)**	76.1(3.2)	58.1(16.4)	12.1(1.1)	36.4(22.4)
MPM	**98.0(7.4)**	78.5(3.0)	77.1(0.5)	**90.1(0.6)**	50.0(0.0)
LPDD	93.4(3.3)	63.6(2.7)	57.7(16.6)	86.4(0.4)	41.8(20.0)
CHAMELEON	94.4(0.7)	77.8(1.0)	76.0(0.8)	80.7(0.4)	39.1(5.1)

Cuadro 4.3: Cuadro comparativo AUC alta dimensión.

Podemos observar que CKTSOM en ciertos conjuntos de datos se obtuvo uno de los mejores desempeños, sin embargo esto no significa que es el mejor método. En el Cuadro 4.2 podemos observar que se obtuvieron buenos resultados para los conjuntos de Liver y Diabetes donde reporto el mejor AUC promedio, además para Liver se obtuvo una desviación estándar bastante elevada con lo que podemos deducir que en ocasiones se obtuvieron resultados muy superiores al reportado. Para los conjuntos de Ecoli, Breast y Abalone, el método propuesto obtuvo 20 puntos porcentuales más bajo que el mejor resultado, esto lo podemos asociar por la me-

nor cantidad de datos de entrenamiento con respecto al total de los datos. En los conjuntos de alta dimensionalidad, Cuadro 4.3 solo en Sonar se obtuvo el mejor resultado en comparación a los otros algoritmos, en los conjuntos de mayor dimensión Concordia y Colon no se obtuvieron resultados mejores que los reportados por los demás métodos, que mantiene la observación anterior de una menor proporción de datos objetivos en comparación al total de los datos, con esto último podemos decir que el algoritmo necesita una buena cantidad de instancias de entrenamiento para una buena clasificación.

Analizando los resultados del Cuadro 4.2 podemos interpretar que CKTSOM puede superar los resultados de otros OCC en problemas con baja cantidad de datos y dimensiones. El Cuadro 4.3 de alta dimensionalidad nos proporciona solo un resultado que logro superar los otros algoritmos.

5. Conclusión y trabajo futuro

5.1. Conclusión

La clasificación automática es un procedimiento costoso, que hasta el momento no asegura una detección perfecta en todas las distribuciones de datos, es por esto que la investigación de nuevos métodos, técnicas y paradigmas, influyen en la exploración de nuevas áreas que no han sido registradas. Dentro de esto podemos comentar el algoritmo CKTSOM, que cambia la elección de un posible BMU global por solo una porción de la red neuronal. Esto nos permite realizar una búsqueda aproximada que es más eficiente en términos de tiempo, donde el resultado es un 90 % de precisión en las 100k iteraciones y el resto de las ocasiones, en las que el resultado no fue el óptimo, sigue siendo bueno, ya que la distancia relativa es del 20 % del peor caso en la primera iteración, lo que baja drásticamente a medida que el entrenamiento avanza y se logra una convergencia adecuada. Este cambio permite que se pueda utilizar una búsqueda en profundidad con un costo logarítmico que es más eficiente que los métodos tradicionales y el utilizado en TTOSOM.

Se comprende que a medida que el tiempo avanza, la necesidad de clasificar datos aumentará, además estos últimos cada vez serán más grandes y complejos, ya que todos tratan de cuantificar su información para poder comprender que está sucediendo y lograr una ventaja ante los demás. Es por esto que algoritmos rápidos y eficientes son necesarios.

La estructura de vector para representar el árbol neuronal, nos proporciona una estructura simple y robusta que es capaz de representar todo el árbol, esto tiene una

gran ventaja por la rápida búsqueda del padre o los hijos, utilizando una operación matemática con los índices de estos. Pero nos trae la inflexibilidad de esta decisión al no poder cortar o agregar una nueva rama que podría influir en un adecuada representación.

En términos de la regla de actualización, la decisión de diferenciar el porcentaje de aprendizaje del padre y los hermanos de un nodo en particular, influye en el entrenamiento del árbol, (como se mencionó en anexos), ya que modificando estos parámetros, es posible cambiar la forma del árbol resultante. Un valor elevado en el porcentaje de aprendizaje de los hermanos genera un árbol con ramificaciones compactas, al contrario, cuando se reduce este factor, las ramas hojas se dispersan entre los datos, generando una mejor representación.

Se evidenció que la búsqueda en profundidad produce un resultado aceptable, y que el resultado mejora mientras el árbol esté entrenado, dado que los resultados obtenidos no superaron al 20 % del peor resultado posible, valor que se reduce notablemente a las pocas iteraciones.

Con los resultados, podemos observar que en general, el algoritmo está entre los primeros, pero no se logró encontrar una configuración adecuada para competir contra todos los demás algoritmos entre los diferentes conjuntos de datos. Es por esto que hasta el momento no se logró obtener un gran hallazgo, al no poder ganar de forma rotunda como se pretendía hacer.

Con los objetivos mencionados, se realizó una implementación del SOM, que luego se transformó en CKTSOM, al utilizar la topología de árbol y algunas variaciones ya mencionadas. Además, con las imágenes presentes en el informe, específicamente en el Anexo B se evidencia el modulo visual implementado para inspeccionar como se comporta el algoritmo CKTSOM.

Para comparar el desempeño del algoritmo propuesto se usó un conjunto de algoritmos mayor al planteado inicialmente en los objetivos, estos últimos fueron optimizados para cada conjunto de datos, obteniendo el mejor resultado posible para realizar la tabla comparativa entre los diferentes OCC.

Por último, mencionar que entre los conjuntos de datos analizados, existen diferencias de tamaño y dimensión, es por esto que se entiende que el tamaño del conjunto de datos influye en la cantidad de neuronas que debe tener el árbol, en otras palabras, en grandes volúmenes de datos se debe usar un árbol con una buena cantidad de nodos. Además, se comprende que el aumento de las dimensiones puede ser dificultoso para el algoritmo, ya que al utilizar la distancia euclidiana nos genera errores que no son positivos para la clasificación.

5.2. Trabajo futuro

El algoritmo implementado tiene una mejora importante que no se implementó por términos de tiempo, se notó esta mejora cuando el algoritmo estaba en la fase de validación, esta mejora tiene que ver con términos de rendimiento del algoritmo, ya que se identificó que es posible unir el proceso de búsqueda del BMU con el proceso de actualización de las neuronas, pues el camino que toma la búsqueda del BMU inicia desde la raíz hasta una hoja, que luego es repetido por el proceso de actualización en forma inversa. Ambos procesos recorren el mismo camino. Con esto queda claro que es posible realizar la actualización a medida que se desciende por el árbol, produciendo una disminución en los procesos que realiza el algoritmo, lo que generaría una disminución en el tiempo de entrenamiento. Esto es un cambio importante al reducir un procedimiento que se genera en cada iteración.

Una idea importante que surgió en el desarrollo del algoritmo (que no se implementó por términos de tiempo) fue la capacidad de trabajo distribuido entre distintas máquinas. Esto se pensó para los casos en que se desee utilizar el algoritmo en volúmenes gigantes de datos y el tiempo de respuesta esperado fuese rápido. Esta idea fue planteada en algún momento, pero no se realizó ningún análisis acabado para modificar el algoritmo de forma tal que pueda utilizar el sistema distribuido.

En la implementación del algoritmo, se utilizó la distancia euclidiana, donde se sabe que en alta dimensión no entrega un resultado adecuado. Una mejora en su desarrollo tendría que estar dirigida a este punto, ya que es posible dejar al usuario

que ingrese el método de cálculo para la distancia.

El algoritmo tiene el potencial de aprendizaje en tiempo real con algunas modificaciones, pudiendo estar detrás de una página web o servicio informático, aprendiendo de las entradas o acciones realizadas por un usuario, pudiendo ofrecer un reconocimiento de patrones, como es el caso de un sistema financiero el cual necesita conocer los patrones o movimientos del usuario para determinar si este es real o una suplantación. Esto tiene que ver con el hecho de que se realizan acciones parecidas, por ejemplo, una persona tiende a sacar dinero de los mismos cajeros automáticos. Es por esto que si detecta un cambio en la ubicación, cantidad o transacción, el algoritmo podría avisar al sistema o usuario para evitar un fraude.

Los porcentajes de actualización para los padres y hermanos no fueron modificados mayormente a lo largo del proyecto. Es posible que para un conjunto de datos en particular, cambiando estos parámetros, sea posible obtener una mejor representación de los datos o en un menor tiempo de entrenamiento.

Bibliografía

[1] Ayodeji A. Akinduko, Evgeny M. Mirkes y Alexander N. Gorban. «SOM: Stochastic initialization versus principal components». En: *Information Sciences* 364 (2016), págs. 213-221. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/j.ins.2015.10.013.

[2] C. A. Astudillo y B. J. Oommen. «Imposing tree-based topologies onto self organizing maps». En: *Inf. Sci.* 181.18 (2011), págs. 3798-3815. DOI: 10.1016/j.ins.2011.04.038. URL: http://dx.doi.org/10.1016/j.ins.2011.04.038.

[3] C. A. Astudillo y B. J. Oommen. «Imposing tree-based topologies onto self organizing maps». En: *Information Sciences* 181.18 (2011), págs. 3798-3815.

[4] C. A. Astudillo y B. J. Oommen. «On achieving semi-supervised pattern recognition by utilizing tree-based SOMs». En: *Pattern Recognition* 46.1 (2013), págs. 293-304. DOI: 10.1016/j.patcog.2012.07.006. URL: http://dx.doi.org/10.1016/j.patcog.2012.07.006.

[5] C. A. Astudillo y B. J. Oommen. «Pattern Recognition using the TTOCONROT». En: *Current Approaches in Applied Artificial Intelligence - 28th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2015, Seoul, South Korea, June 10-12, 2015, Proceedings*. Ed. por Moonis Ali y col. Vol. 9101. Lecture Notes in Computer Science. Springer, 2015, págs. 435-444. ISBN: 978-3-319-19065-5. DOI: 10.1007/978-3-319-19066-2_42. URL: http://dx.doi.org/10.1007/978-3-319-19066-2_42.

[6] C. A. Astudillo y B. J. Oommen. «Topology-oriented self-organizing maps: a survey». En: *Pattern Anal. Appl.* 17.2 (2014), págs. 223-248. DOI: 10.1007/

s10044-014-0367-9. URL: http://dx.doi.org/10.1007/s10044-014-0367-9.

[7] C. A. Astudillo y B. John Oommen. «Self-organizing maps whose topologies can be learned with adaptive binary search trees using conditional rotations». En: *Pattern Recognition* 47.1 (2014), págs. 96-113. DOI: 10.1016/j.patcog.2013.04.012. URL: http://dx.doi.org/10.1016/j.patcog.2013.04.012.

[8] César A. Astudillo, Matthew Bardeen y Narciso Cerpa. «Editorial: Data Mining in Electronic Commerce - Support vs. Confidence». En: *JTAER* 9.1 (2014). URL: http://www.jtaer.com/jan2014/Editorial_p0.pdf.

[9] César A. Astudillo y B. John Oommen. «Fast BMU Search in SOMs Using Random Hyperplane Trees». En: *PRICAI 2014: Trends in Artificial Intelligence: 13th Pacific Rim International Conference on Artificial Intelligence, Gold Coast, QLD, Australia, December 1-5, 2014. Proceedings*. Ed. por Duc-Nghia Pham y Seong-Bae Park. Cham: Springer International Publishing, 2014, págs. 39-51. ISBN: 978-3-319-13560-1. DOI: 10.1007/978-3-319-13560-1_4.

[10] César A. Astudillo y col. «A Cluster Analysis of Stock Market Data Using Hierarchical SOMs». En: *AI 2016: Advances in Artificial Intelligence - 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5-8, 2016, Proceedings*. Ed. por Byeong Ho Kang y Quan Bai. Vol. 9992. Lecture Notes in Computer Science. Springer, 2016, págs. 101-112. ISBN: 978-3-319-50126-0. DOI: 10.1007/978-3-319-50127-7_8. URL: https://doi.org/10.1007/978-3-319-50127-7_8.

[11] Dirk Eddelbuettel y Romain François. «Rcpp: Seamless R and C++ Integration». En: *Journal of Statistical Software* 40.8 (2011), págs. 1-18. URL: http://www.jstatsoft.org/v40/i08/.

[12] *Encyclopedia of Machine Learning*. SPRINGER PG, 11 de mayo de 2011. 1031 págs. ISBN: 0387307680. URL: http://www.ebook.de/de/product/8781134/encyclopedia_of_machine_learning.html.

[13] Piotr Juszczak y col. «Minimum spanning tree based one-class classifier». En: *Neurocomputing* 72.7 (2009). Advances in Machine Learning and Computational Intelligence, págs. 1859-1869. ISSN: 0925-2312. DOI: https://doi.org/

10.1016/j.neucom.2008.05.003. URL: http://www.sciencedirect.com/science/article/pii/S0925231208003238.

[14] Shehroz S. Khan y Michael G. Madden. «One-class classification: taxonomy of study and review of techniques». En: *The Knowledge Engineering Review* 29.3 (2014), págs. 345-374. DOI: 10.1017/S026988891300043X.

[15] Erwin Kreyszig, Herbert Kreyszig y E. J. Norminton. *Advanced Engineering Mathematics*. Tenth. Hoboken, NJ: Wiley, 2011, pág. 880. ISBN: 0470458364.

[16] Gonzalo Maldonado y col. «Predicting the stability of human lysozyme mutants using the tree-based classifier TTOSOM». En: *Chemometrics and Intelligent Laboratory Systems* 162 (2017), págs. 65-72. ISSN: 0169-7439. DOI: http://dx.doi.org/10.1016/j.chemolab.2017.01.002. URL: http://www.sciencedirect.com/science/article/pii/S0169743917300102.

[17] D.J. Newman y col. *UCI Repository of machine learning databases*. 1998. URL: http://www.ics.uci.edu/~mlearn/MLRepository.html.

[18] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2017. URL: https://www.R-project.org/.

[19] *Repositorio conjunto de dato*. 2017. URL: http://homepage.tudelft.nl/n9d04/occ/.

[20] David Martinus Johannes Tax. «One-class classification: Concept learning in the absence of counter-examples». Tesis doct. Technische Universiteit Delft, 2001.

Glosario

1-NN 1-Nearest Neighbour

AUC Area Under Curve

BMU Best Matching Unit

CKTSOM Complete k-ary Tree Self-Organizing Maps

k-NN k-Nearest Neighbors

LPDD Linear Programming Dissimilarity-data Description

MoG Mixture of Gaussian

MPM Minimax Probability Machine

MST_CD Minimum Spanning Tree Class Descriptor

OCC One-Class Classification

PCA Principal Component Analysis

RMSE Root-Mean-Square Deviation

ROC Receiver Operating Characteristic

SOM Self-Organizing Maps

SVDD Support Vector Domain Description

TTOSOM Tree-based Topology Oriented SOM

ANEXOS

A. Metodología de desarrollo

A.1. Metodología

La metodología utilizada corresponde a reuniones semanales donde el estudiante y el profesor se reúnen a trabajar y discutir sobre el tema, para esto primero se revisan las tareas asignadas la semana anterior, se revisan las tareas completadas con el fin de verificar su validez y para las incompletas se estudian los problemas y las posibles soluciones, terminándolas dentro de la semana. Luego de esto, se inicia un período en donde se trabaja y avanza en solucionar los problemas de mayor dificultad y tareas importantes, para luego programar tareas y analizar una posible solución. Todo esto es anotado en un sistema de control de versiones en donde se registran las tareas y las descripciones de estas para mantener el contexto del problema. Además, el estudiante avanza y registra su progreso dentro del sistema de control de versiones en momentos de trabajo autónomo, agregando la tarea realizada y el comentario correspondiente del trabajo terminado.

A.2. Herramientas

A.2.1. R

R es un lenguaje de programación enfocado al análisis estadístico, es un proyecto GNU que es similar a S pero con una implementación diferente, heredando la orientación a objeto de S. Proporciona un amplio abanico de herramientas estadísticas con modelos lineales y no lineales, análisis de series temporales, test estadísticos,

algoritmos de clasificación y agrupamiento, gráficas, etc. Además es posible la integración con bases de datos y con una documentación basada en látex [18].

A.2.2. C++

C++ es un lenguaje de programación multiparadigma desarrollado en 1980, se considera una extensión de C y por ello tiene el nombre de "C++" lo que significa "incremento de C", este nombre fue propuesto por Rick Mascitti en 1983. Este lenguaje se caracteriza por su gran rapidez de cómputo.

A.2.3. Librería Rcpp

Esta librería provee la interacción entre R y C/C++ utilizando la interfaz .call() proporcionada por R, permite la asignación de múltiples tipos de objetos de R como vectores, matrices, funciones, etc. a clases dedicadas en C++. La primera versión de Rcpp fue escrita por Dominick Samperi entre los años 2005 y 2006 [11].

A.3. Control de versiones

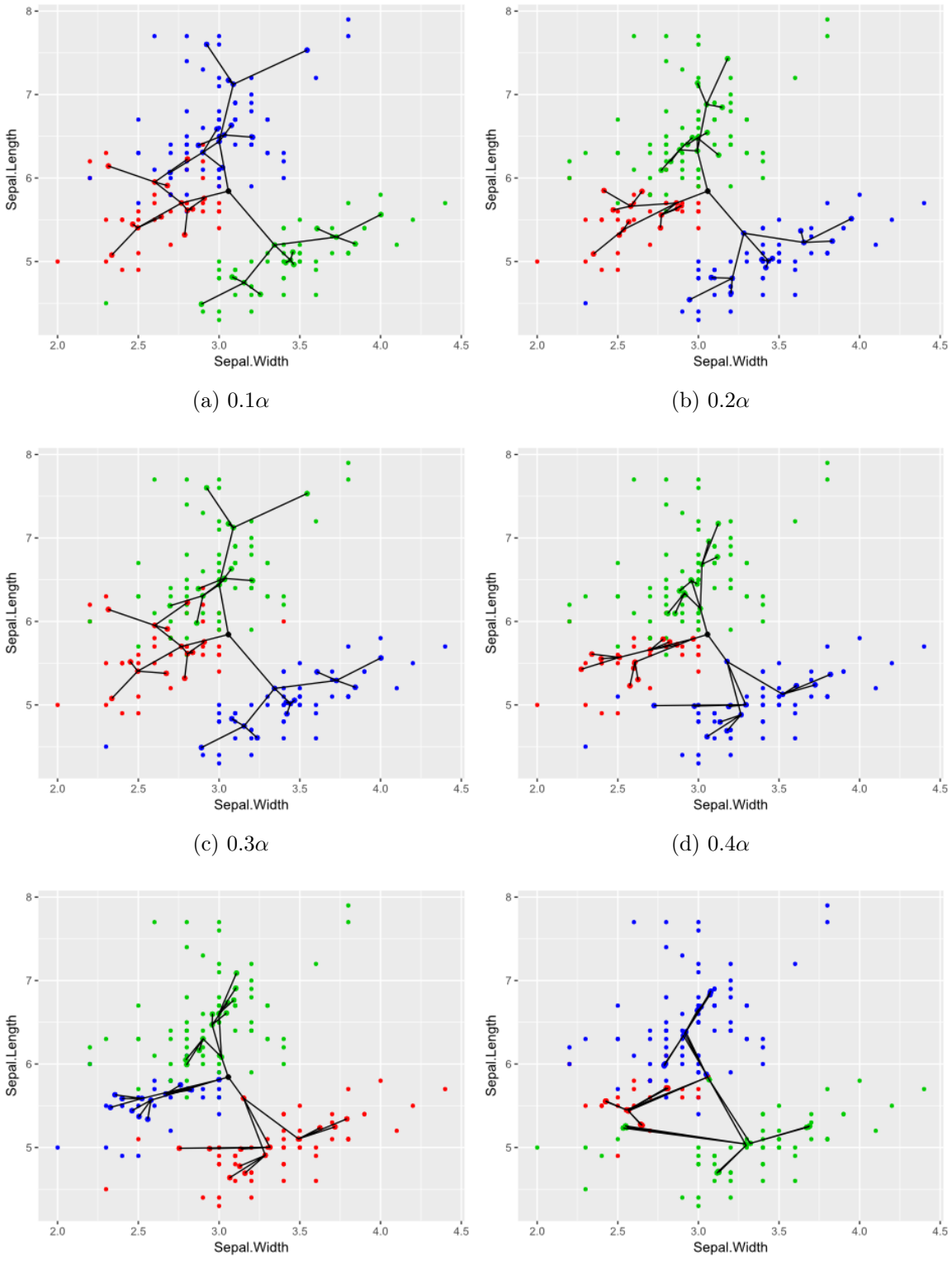
El control de versiones es un sistema que mantiene un registro de los cambios realizados sobre un archivo o un grupo de estos, proporciona un historial de los cambios, permite saber quién realizado un cambio y en que momento se hizo. Con esto se tiene un respaldo completo del avance del proyecto pudiendo retornar a versiones anteriores si fuese necesario, además todo el trabajo se encuentra solo en un repositorio lo que evita problemas serios de integración entre varios colaboradores ya que este proceso se realiza periódicamente. Esto permite un trabajo cooperativo fluido y eficiente.

B. Aprendizaje hermanos

B.1. Factor de aprendizaje

Las actualizaciones del árbol son definida por la iteración T, esta determina el factor de aprendizaje α_T, donde el BMU absorbe el 100 %, el padre directo el 90 % y el hermano un 20 % de la información, a continuación se presenta diferentes porcentaje de la absorción de información para los hermanos y como resulta el árbol luego del entrenamiento.

En la siguiente Figura B.1, se presenta distintos arboles configurados cada uno de la misma manera, solo cambiando el porcentaje de absorción para los hermanos, siendo 0.2α la configuración utilizada en la implementación, y 0.9α si el porcentaje aprendido por el padre y los hermanos fueran los mismos. Estos gráficos son construidos por datos de cuatro dimensiones, dibujando solo dos de estas, es por esto que en algunos gráficos no se observa un correcto clustering.



(a) 0.1α (b) 0.2α

(c) 0.3α (d) 0.4α

(e) 0.5α (f) 0.9α

Figura B.1: Diferentes configuraciones de aprendizaje de los hermanos.