UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

DISEÑO DE UN MODELO PARA LA DETECCIÓN DE FRAUDES BANCARIOS MEDIANTE EL USO DE TÉCNICAS DE APRENDIZAJE PROFUNDO

ROMERO LEON JUAN JOSE
INGENIERO DE SISTEMAS

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

DISEÑO DE UN MODELO PARA LA DETECCIÓN DE FRAUDES BANCARIOS MEDIANTE EL USO DE TÉCNICAS DE APRENDIZAJE PROFUNDO

ROMERO LEON JUAN JOSE
INGENIERO DE SISTEMAS

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

TRABAJO TITULACIÓN
PROPUESTAS TECNOLÓGICAS

DISEÑO DE UN MODELO PARA LA DETECCIÓN DE FRAUDES BANCARIOS MEDIANTE EL USO DE TÉCNICAS DE APRENDIZAJE PROFUNDO

ROMERO LEON JUAN JOSE
INGENIERO DE SISTEMAS

RIVAS ASANZA WILMER BRAULIO

MACHALA, 22 DE SEPTIEMBRE DE 2022

MACHALA
2022

juanromero

INFORME DE ORIGINALIDAD

8%	7%	1%	5%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad Técnica de Machala Trabajo del estudiante	3%
2	repositorio.utmachala.edu.ec Fuente de Internet	2%
3	hdl.handle.net Fuente de Internet	1%
4	Submitted to tec Trabajo del estudiante	<1%
5	Submitted to Escuela Politecnica Nacional Trabajo del estudiante	<1%
6	dspace.palermo.edu Fuente de Internet	<1%
7	Submitted to Universidad Internacional de la Rioja Trabajo del estudiante	<1%
8	www.mdpi.com Fuente de Internet	<1%
9	www.coursehero.com Fuente de Internet	

CLÁUSULA DE CESIÓN DE DERECHO DE PUBLICACIÓN EN EL REPOSITORIO DIGITAL INSTITUCIONAL

El que suscribe, ROMERO LEON JUAN JOSE, en calidad de autor del siguiente trabajo escrito titulado DISEÑO DE UN MODELO PARA LA DETECCIÓN DE FRAUDES BANCARIOS MEDIANTE EL USO DE TÉCNICAS DE APRENDIZAJE PROFUNDO, otorga a la Universidad Técnica de Machala, de forma gratuita y no exclusiva, los derechos de reproducción, distribución y comunicación pública de la obra, que constituye un trabajo de autoría propia, sobre la cual tiene potestad para otorgar los derechos contenidos en esta licencia.

El autor declara que el contenido que se publicará es de carácter académico y se enmarca en las dispociones definidas por la Universidad Técnica de Machala.

Se autoriza a transformar la obra, únicamente cuando sea necesario, y a realizar las adaptaciones pertinentes para permitir su preservación, distribución y publicación en el Repositorio Digital Institucional de la Universidad Técnica de Machala.

El autor como garante de la autoría de la obra y en relación a la misma, declara que la universidad se encuentra libre de todo tipo de responsabilidad sobre el contenido de la obra y que asume la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera exclusiva.

Aceptando esta licencia, se cede a la Universidad Técnica de Machala el derecho exclusivo de archivar, reproducir, convertir, comunicar y/o distribuir la obra mundialmente en formato electrónico y digital a través de su Repositorio Digital Institucional, siempre y cuando no se lo haga para obtener beneficio económico.

Machala, 22 de septiembre de 2022

ROMERO LEON JUAN JOSE
0705122075

DEDICATORIA

Este trabajo de titulación está dedicado con amor para mi papá, mamá y mis hermanos, que a lo largo de mis años de estudio me han acompañado, quienes día a día me brindan su apoyo incondicional y me motivan para alcanzar todas mis metas que me propongo, ya que sin su apoyo no hubiese podido cumplir con este objetivo.

Juan José Romero León

AGRADECIMIENTO

En primer lugar, agradezco a Dios por darme la dicha de poder culminar mis estudios de tercer nivel.

A mi papá, mamá y hermanos que han tenido paciencia y comprensión durante todo el trayecto de mi vida universitaria.

A los docentes que gracias a sus conocimientos impartidos me han ayudado a desarrollarme profesionalmente.

Y por último agradezco al docente y mi tutor del trabajo de titulación, el ingeniero Wilmer Rivas que ha sido un guía fundamental durante todo el proceso.

Juan José Romero León

RESUMEN

Los fraudes bancarios es un problema a nivel mundial, esto mantiene en alerta a todas las entidades financieras, pero gracias al avance de la tecnología en la actualidad, la mayoría de las mencionadas entidades han tomado medidas al respecto implementado sistemas informáticos robustos para que no sucedan este tipo de engaños. Es por eso por lo que es de importancia crear un modelo predictivo híbrido para poder detectar si un cliente está haciendo fraude a partir de los datos que han sido registrado en un formulario para una solicitud de tarjeta de crédito de un banco australiano mediante el uso de técnicas de Deep Learning.

Dentro de este trabajo de titulación gracias a la investigación exhaustiva, se describe la importancia de los avances tecnológicos en las empresas financieras, la importancia de la inteligencia artificial, el aprendizaje profundo y el aprendizaje automático que van de la mano para lograr soluciones a las empresas como, por ejemplo, detectar, predecir y prevenir los fraudes por parte de los clientes al momento de llenar un formulario.

Se propuso elaborar un modelo híbrido que mezcla dos técnicas de Deep Learning y gracias a la investigación bibliográfica de este trabajo se pudo entender cuál es el trabajo de un Analista de Datos en una entidad financiera, para ello, lo primero que se realizó fue la creación y entrenamiento de un Mapa Autoorganizado (modelo no supervisado) que nos dio como resultado una lista de los posibles clientes fraudulentos, luego se utilizó esa lista como la variable dependiente para poder trasladar el modelo no supervisado a uno supervisado, entonces se procedió a realizar la creación y el entrenamiento de una Red Neuronal Artificial (modelo supervisado) cuyo objetivo fue la de predecir el porcentaje de si un cliente está haciendo fraude al momento de llenar la solicitud para una tarjeta de crédito del banco.

Para evaluar el modelo se estableció el uso de una tabla comparativa con respecto a los resultados del Mapa Autoorganizado con el Dataset Original, para poder medir el porcentaje de precisión y de error. Cabe recalcar que no se hizo uso de métricas de evaluación ya que el modelo predictivo no es de clasificación, por ende, el modelo no utiliza datasets de testing ni de validación, tampoco se hizo uso de varios algoritmos dentro de la Red Neuronal Artificial.

Se realizaron cuatro pruebas de rendimiento del modelo con variaciones en ciertos parámetros del Mapa Autoorganizado para así escoger con que parámetros se obtiene el mejor resultado posible. El mejor resultado de las pruebas arrojó un porcentaje de 100

% de precisión del Mapa Autoorganizado con 18 aciertos y 0 errores y un 97 % de precisión y un 15.74 % de pérdida en el entrenamiento de la Red Neuronal Artificial.

En un futuro para que el modelo funcione mejor y robustamente, se debería experimentar con datasets de mayor cantidad, aplicar más capas ocultas a la red neuronal artificial y entrenarla con más épocas para así poder evaluar de mejor forma el modelo predictivo. También se puede experimentar probando otro tipo de Redes Neuronales dependiendo de los datos que nos proporciona el dataset original.

Palabras claves: aprendizaje profundo, mapas auto organizados, inteligencia artificial, algoritmos, modelo predictivo, sistema de fraude.

ABSTRACT

Bank fraud is a problem worldwide, this keeps all financial institutions on alert, but thanks to the advancement of technology today, most of the aforementioned entities have taken measures in this regard by implementing robust computer systems so that they do not happen. this type of deception. That is why it is important to create a hybrid predictive model to be able to detect if a customer is committing fraud from the data that has been registered in a form for a credit card application from an Australian bank through the use of Deep Learning techniques.

Within this degree work, thanks to exhaustive research, the importance of technological advances in financial companies is described, the importance of artificial intelligence, deep learning and machine learning that go hand in hand to achieve solutions for companies. such as detecting, predicting and preventing fraud by customers when filling out a form.

It was proposed to develop a hybrid model that mixes two Deep Learning techniques and thanks to the bibliographical research of this work it was possible to understand what the work of a Data Analyst is in a financial entity, for this, the first thing that was done was the creation and training of a Self-organized Map (unsupervised model) that gave us as a result a list of possible fraudulent clients, then that list was used as the dependent variable to be able to transfer the unsupervised model to a supervised one, then we proceeded to perform the creation and training of an Artificial Neural Network (supervised model) whose objective was to predict the percentage of whether a client is committing fraud when filling out the application for a bank credit card.

To evaluate the model, the use of a comparative table was established with respect to the results of the Self-organized Map with the Original Dataset, in order to measure the percentage of precision and error. It should be noted that evaluation metrics were not used since the predictive model is not a classification model, therefore, the model does not use testing or validation datasets, nor was use of various algorithms within the Artificial Neural Network.

Four performance tests of the model were carried out with variations in certain parameters of the Self-Organized Map in order to choose with which parameters the best possible result is obtained. The best result of the tests showed a percentage of 100% accuracy of the Self-Organized Map with 18 hits and 0 errors and 97% accuracy and 15.74% loss in the training of the Artificial Neural Network.

In the future, for the model to work better and more robustly, it should be experimented with larger datasets, apply more hidden layers to the artificial neural network and train it with more epochs in order to better evaluate the predictive model. It can also be experimented with by testing other types of Neural Networks depending on the data provided by the original dataset.

Keywords: Deep learning, Self-Organized Maps, Artificial Intelligence, algorithms, predictive model, fraud system.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN

En las últimas décadas, la inteligencia artificial ha evolucionado en las diferentes áreas de la vida cotidiana; en la actualidad, con los grandes volúmenes de datos que se producen y publican en la web, es de suma importancia contar con herramientas tecnológicas que permita a las personas obtener, procesar y distinguir información que le resulte útil en su formación profesional. [1]

Con la evolución tecnológica evidente en el campo empresarial y la fuerte presencia en temas de seguridad en los últimos años, el aprendizaje automático es una gran parte para obtener una comprensión más profunda de toda la información almacenada en la nube.

La inteligencia artificial es una tecnología basada en patrones, es capaz de aprender relaciones y tendencias de manera automática, permitiendo integrar técnicas de gran capacidad analítica como el Machine Learning, además es posible, entre otras cosas, monitorear y configurar parámetros que colaboren en la detección de acciones que antes eran más difíciles de prevenir. [2]

El sistema financiero ha evolucionado significativamente y, al mismo tiempo, las instituciones y el propio sistema financiero han sido capaces de diseñar diferentes sistemas para proteger su dinero; sin embargo, aún existen algunas personas inescrupulosas que aprovechan este medio para realizar sus actividades ilícitas, también conocidas como estafas financieras. [3]

La Detección de Fraude Financiero les da a las entidades un mejor control sobre su empresa y así puedan ingresar a nuevos mercados con la confianza de que todos los movimientos tengan un respaldo seguro en un sistema antifraude. Se espera que la información financiera sea resguardada y los profesionales encargados de la manipulación de estos datos mantengan los principios éticos de confidencialidad, si bien la información se ha extendido a más usuarios, lo que las organizaciones quieren lograr es que la información no sea manipulada por personas sin escrúpulos, y sea transparente e instrumental en la toma de decisiones financieras.

Existen algoritmos desarrollados a partir de aprendizaje automático que han ayudado en el área financiera de grandes empresas a detectar si existe un fraude por parte de los usuarios, por lo que se considera factible desarrollar un algoritmo híbrido de aprendizaje automático utilizando dos técnicas de aprendizaje profundo: Mapas Autoorganizados y

una Red Neuronal Artificial, el cual va a permitir detectar que clientes del dataset original han realizado un posible fraude en el banco.

El mapa autoorganizado será desarrollado con métodos de la clase MiniSom.py [4] para la obtención de una lista de clientes que será la variable independiente para la red neuronal artificial y así proceder a realizar los pasos correspondientes para lograr obtener una predicción adecuada y cumplir con los objetivos del trabajo de titulación.

El presente documento consta de la siguiente estructura:

Capítulo 1: En esta sección del trabajo se detalla el origen del problema, el contexto en que se va a aplicar, los requisitos del prototipo, la justificación y la solución al problema.

Capítulo 2: Aquí se especifican los conceptos relevantes para la comprensión del trabajo como son: la fundamentación teórica, objetivos, diseño, la ejecución y pruebas del prototipo.

Capítulo 3: Dentro de esta sección se evidencia la ejecución y los resultados obtenidos de la implementación del prototipo y se detallan las conclusiones y recomendaciones.

1. CAPÍTULO I: DIAGNÓSTICO DE NECESIDADES Y REQUERIMIENTOS

1.1 Ámbito de Aplicación: descripción del contexto y hechos de interés

Los grandes volúmenes de datos que es manejado por las organizaciones bancarias han crecido en gran cantidad y con ello el riesgo de ser más endebles a los ataques cibernéticos. Las entidades bancarias observan cada vez más la información como un recurso valioso que necesita ser protegido con la mejor tecnología y optimizando los costos. Los hackers incrementan así mismo su nivel de sofisticación, por lo tanto, las empresas financieras están avanzando para minimizar la amenaza cibernética.

Entre las herramientas modernas que nos permiten combatir diversas amenazas de los delincuentes informáticos se encuentran la inteligencia artificial y el aprendizaje automático. Los sistemas de aprendizaje automático pueden analizar información utilizando algoritmos matemáticos para predecir el comportamiento de los datos y obtener resultados sorprendentemente similares a los humanos. [5]

El fraude en las entidades bancarias es un gran problema que no solo ocasionan pérdida económica, si no también, pérdida de imagen y desconfianza hacia ese banco. El problema de la detección de fraudes se amplifica con algunas características y limitaciones. Cuando no se puede prevenir el fraude, es extremadamente importante detectarlo lo antes posible.

A la hora de determinar el riesgo de entidades financieras fraudulentas, es fundamental contar con herramientas informáticas que permitan identificar patrones de comportamiento en un gran número de registros de transacciones como anómalas y/o con potencial actividad fraudulenta respectivamente. La detección oportuna del fraude lo ayuda a evitar pérdidas, proteger la reputación y los activos de su empresa y aumentar la confianza del cliente. [6]

En este trabajo de titulación, se busca diseñar un modelo para la predicción de posibles fraudes por parte de los clientes de una entidad bancaria haciendo uso de técnicas de aprendizaje automático, utilizando un data set que contiene datos de los clientes, el modelo me va a generar una lista de los posibles clientes que están haciendo fraude en la entidad bancaria.

1.2 Establecimiento de requerimientos

Para crear el modelo de predicción para la detección de fraude, se necesitará los siguientes requerimientos:

- Recolección de información sobre las técnicas de Deep Learning para el desarrollo del modelo.
- Uso de un dataset constituido de datos sobre clientes de una entidad bancaria del país de Australia.
- Uso del lenguaje de programación Python para los algoritmos predictivos.
- Uso del entorno Spyder para el diseño del modelo, el entrenamiento y pruebas de este.
- Uso de librerías de Python para el diseño, entrenamiento y testeo del modelo predictivo.

1.3 Justificación del requerimiento a satisfacer

Los métodos de aprendizaje automático se utilizan en casi todos los aspectos de la informática hoy en día debido a sus funcionalidades altamente adaptables y extensibles. Capaz de adaptarse a retos nuevos y desconocidos. La ciberseguridad es actualmente un campo de rápido crecimiento debido al mayor interés en la seguridad de las redes y aplicaciones junto con el crecimiento de las redes sociales, la banca móvil, la computación en la nube, la tecnología de Internet, las redes inteligentes, etc. [7]

En el área de la ciberseguridad, por su diversidad y aplicaciones, genera grandes cantidades de datos de distintas fuentes. Estos datos ofrecen un gran alcance a los científicos de datos, los expertos de ciberseguridad y los entusiastas del aprendizaje automático, ya que tienen la capacidad de proporcionar información que puede ayudar en la lucha contra los ciberdelincuentes utilizando algoritmos de aprendizaje automático.

El fraude es una amenaza que crece rápidamente y tiene un impacto negativo en la economía, las instituciones cooperativas y la regulación. Las transacciones con tarjetas de crédito están creciendo cada vez más rápido gracias al avance de la tecnología de Internet, lo que genera una gran dependencia de Internet. [8]

Con el avance de la tecnología y el creciente uso de las tarjetas de crédito, las tasas de fraude se están convirtiendo en un problema para la economía. Además de agregar nuevas funciones de seguridad a las transacciones con tarjetas de crédito, los estafadores también desarrollan nuevos patrones o agujeros de seguridad para lograr un seguimiento más de cerca en cada transacción.

Además, el problema con la información de las tarjetas de crédito es que está muy sesgada, lo que conlleva a una predicción errónea de si el cliente está cometiendo fraude o no.

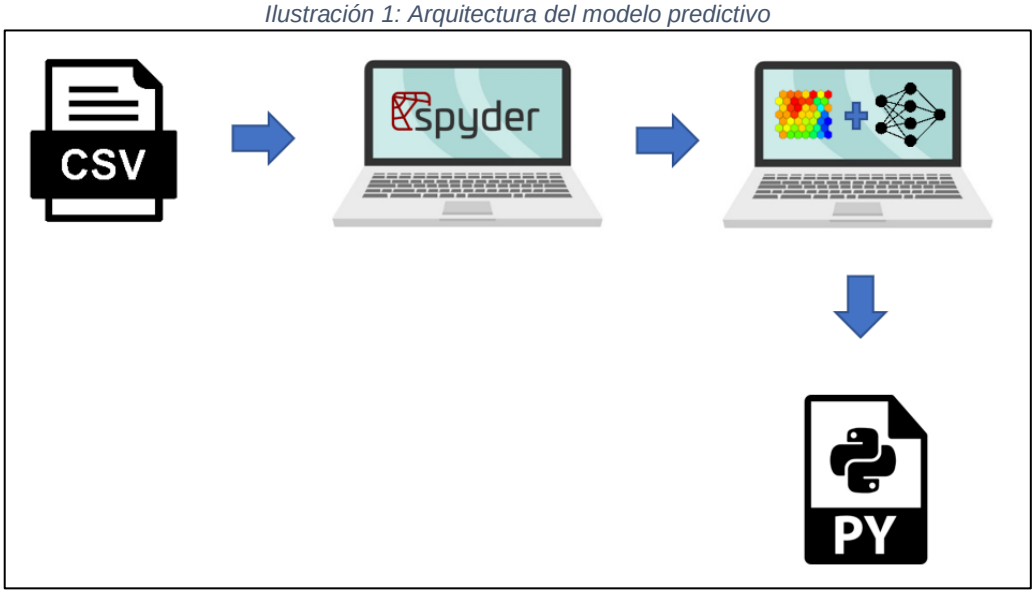
Por lo tanto, este trabajo de titulación integra el entrenamiento y testeo de un modelo utilizando un algoritmo híbrido de aprendizaje profundo, que permitirá predecir si un cliente del banco está haciendo fraude o no.

2. CAPÍTULO II. DESARROLLO DEL PROTOTIPO

2.1. Definición del prototipo tecnológico

El modelo de predicción elaborado en este trabajo de titulación hace uso de dos modelos: un modelo de mapas auto organizados (SOM) y una red neuronal artificial (RNA), se optó por usar un modelo no supervisado (Mapas Auto Organizados) para convertirlo en un modelo supervisado (Red Neuronal Artificial) y obtener nuestro modelo de predicción en un archivo .py para detectar si un cliente del banco está haciendo fraude o no.

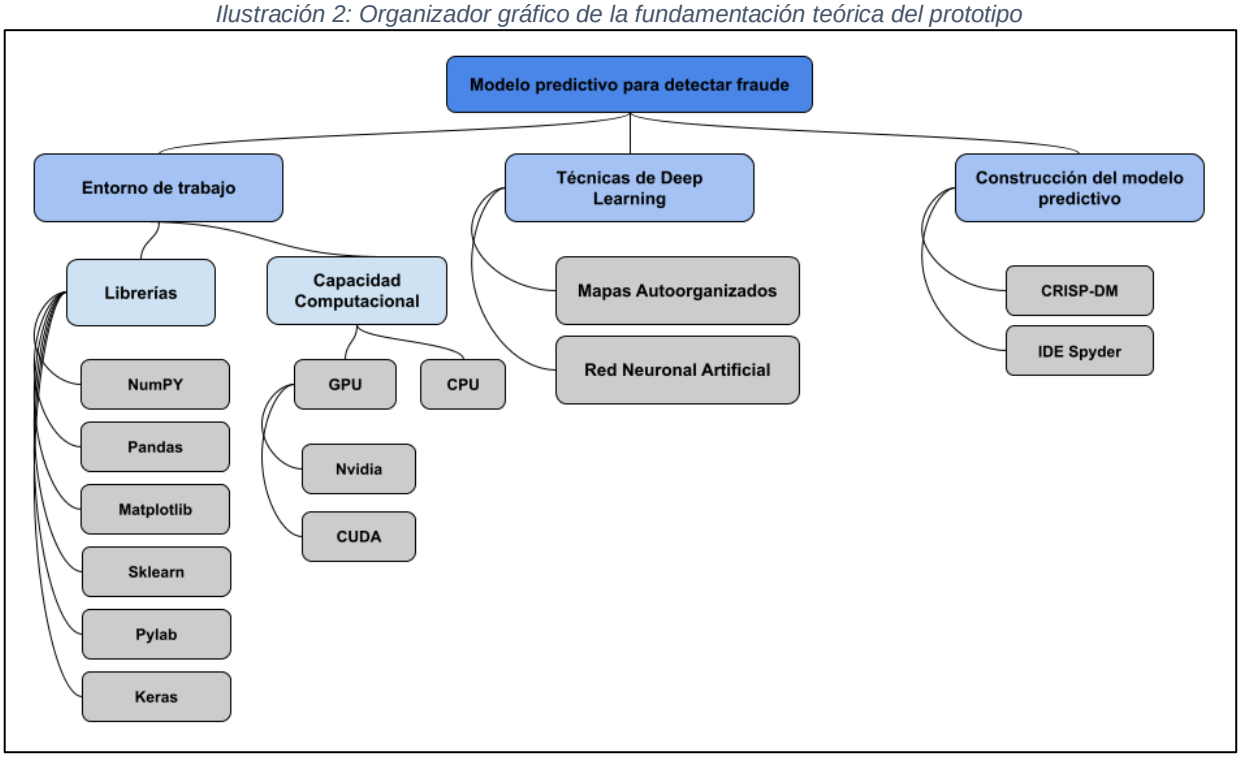
Ilustración 1: Arquitectura del modelo predictivo



Fuente: *Elaboración propia*

2.2. Fundamentación teórica del prototipo

Ilustración 2: Organizador gráfico de la fundamentación teórica del prototipo



Fuente: *Elaboración propia*

2.2.1. Entorno de trabajo

2.2.1.1. Librerías

2.2.1.1.1. NumPy

La librería NumPy (Numeric Python) se usa para manipular grandes arreglos y matrices de datos numéricos ya sean unidimensionales como multidimensionales. [9] NumPy es la librería más utilizada por los científicos de datos gracias a su versatilidad.

2.2.1.1.2. Pandas

Es una librería que se usa para el análisis de datos y proporciona unas estructuras de datos flexibles. Cuenta con la capacidad de transformar los datos a una tabla y poder trabajar con objetos de un Dataframe. [10]

2.2.1.1.3. Matplotlib

Es una librería de Python que se usa para crear gráficos e imágenes a partir de datos. La mayoría de estos gráficos o imágenes son representadas como matrices de pixeles, se la usa mucho con la combinación de la librería NumPy. [11] Con Matplotlib se puede crear varios tipos de gráficos: histogramas, series temporales, espectros de potencia, diagrama de errores, diagramas de barras, etc.

2.2.1.1.4. Sklearn

Es una biblioteca de aprendizaje automático ampliamente utilizada que es fácil de usar e incluye implementaciones de una lista amplia y completa de métodos de aprendizaje automático que siguen convenciones de datos uniformes y flujos de trabajo de modelado, lo que la convierte en una biblioteca muy fácil de usar para los profesionales del aprendizaje automático. estadísticas y análisis de datos. [12]

Los desarrolladores más la utilizan para la clasificación de datos gracias a sus algoritmos de aprendizaje automático.

2.2.1.1.5. Pylab

Es una librería que sirve para aportar funcionalidades extra a NumPy integrando funciones utilizadas en Matlab en entornos de aprendizaje automático. [13] Los desarrolladores utilizan esta librería como ayuda a la representación gráfica de los mapas auto organizados.

2.2.1.1.6. Keras

Es la librería más utilizada en redes neuronales. En el año 2017 fue el segundo marco de aprendizaje profundo más referenciado en la investigación científica, junto a TensorFlow de Google. Keras es ahora la interfaz oficial de TensorFlow. [14] Los desarrolladores utilizan Keras para ir elaborando los bloques de arquitectura de una red neuronal, incluyendo todos tipos de redes neuronales como, por ejemplo, redes artificiales, convolucionales y recurrentes, que son las que permiten entrenar los modelos de aprendizaje profundo.

2.2.1.2. Capacidad Computacional

Hoy en día, se puede implementar y ejecutar Machine Learning o Deep Learning desde cualquier computador, a la hora de entrenar las redes neuronales, va a depender mucho del hardware que se esté utilizando porque condicionará el tiempo de aprendizaje de la red neuronal.

Una alternativa para los desarrolladores que no cuentan con buenos recursos de hardware, es utilizar servicios en la nube como por ejemplo Google Colab [15] que permite a los usuarios tipear código Python desde el navegador. También utiliza recursos de CPU y GPU de los servidores de Google para la codificación de los algoritmos.

La GPU y CPU son los principales componentes de un computador, estos ayudan a mejorar el rendimiento en los computadores y de esta forma facilitar la creación de las redes neuronales y los entrenamientos de estos.

2.2.1.2.1. GPU

La GPU (Graphics Processing Unit, por sus siglas en inglés) se utiliza para acelerar el tiempo de entrenamiento y mejorar el funcionamiento de los algoritmos de aprendizaje automático, el aprendizaje profundo, el análisis de datos y las aplicaciones de ingeniería. Esto es de gran importancia en las áreas de inteligencia artificial, desarrollo de automóviles, sistemas no tripulados y robótica.

Dentro del mundo del aprendizaje profundo, diseñar un algoritmo paralelo en una GPU es bastante diferente que diseñarlo utilizando solo CPU. En una arquitectura de GPU, se tiene cientos de miles de subprocesos ejecutándose simultáneamente a diferencia de una CPU que se ejecuta en serie. [16]

2.2.1.2.1.1. Nvidia

Nvidia es una compañía tecnológica que diseña unidades de procesamiento gráficos (GPUs). También proporciona capacidades de procesamiento en paralelo a científicos e investigadores, para que puedan ejecutar aplicaciones de alto rendimiento de una manera eficaz. [17]

2.2.1.2.1.2. CUDA

CUDA (Compute Unified Device Arquitecture, por sus siglas en inglés) es una de las opciones más populares para la programación de GPU, pero solo se puede ejecutar en GPUs de NVIDIA. La ejecución de CUDA no solo beneficia a la comunidad de hardware, sino que también permite el cálculo paralelo de datos en sistemas heterogéneos. [18] Toda tarjeta gráfica de NVIDIA cuenta con núcleos CUDA, estos son procesadores paralelos que procesan todos los datos de entrada y salida de la GPU, su objetivo primordial es realizar cálculos gráficos y renderizar objetos 3D.

Dentro del mundo del aprendizaje automático, se utiliza estos núcleos para la clasificación de imágenes para obtener mejores resultados de los algoritmos.

2.2.1.2.2. CPU

CPU (Central Processing Unit, por sus siglas en inglés) es el encargado de procesar todos los datos del computador y realizar cálculos matemáticos-informáticos.

Dentro del ámbito de la inteligencia artificial para el procesamiento de imágenes, se recomienda paralelizar múltiples núcleos del CPU, ya que el CPU consta de unos pocos núcleos que están optimizados para el procesamiento secuencial. [19]

2.2.2. Técnicas de Deep Learning

Deep Learning o Aprendizaje Profundo se encarga de entrenar a un computador para que ejecute tareas como las que hacen los seres humanos, como el reconocimiento de imágenes, reconocimiento de voz o hacer predicciones.

Las técnicas de aprendizaje profundo están superando a las técnicas actuales de aprendizaje automático. Permite que los modelos computacionales aprendan características progresivamente a partir de datos en múltiples niveles. La popularidad del aprendizaje profundo se amplificó a medida que aumentaba la cantidad de datos disponibles, así como el avance del hardware que proporciona computadoras poderosas. [20]

Una de las bases importantes de la Inteligencia Artificial es el Deep Learning, gracias a sus técnicas han mejorado la capacidad de clasificar, detectar, describir y reconocer imágenes, palabras y audios.

2.2.2.1. Mapas Auto Organizados

Es una técnica del Deep Learning que es utilizada para la detección de características, fue creada en los años 80 por el señor Teuvo Kohonen. Los mapas autoorganizados están inspirados en el hecho de que el cerebro humano puede extraer características relevantes del mundo. [21]

Un mapa autoorganizado es una red neuronal artificial en la cual su entrenamiento es un aprendizaje no supervisado cuyo objetivo es presentar un mapa discreto con los datos de entrada.

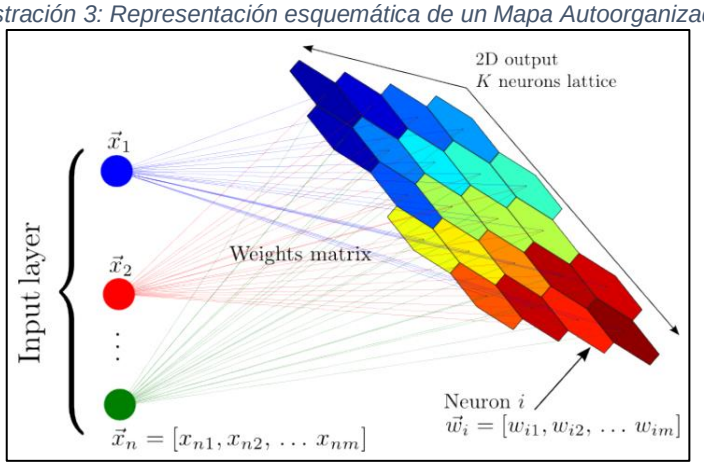
La red neuronal de un mapa autoorganizado detecta automáticamente las relaciones (autoorganizadas) dentro del conjunto de datos de entrada. Esta propiedad se puede aplicar al problema del mapeo de imágenes desde dimensiones más altas a un espacio de características bidimensionales para realizar la clasificación de imágenes. [22]

El algoritmo de un mapa autoorganizado solo utiliza dos capas:

- Una capa de **nodos de aprendizaje,** cuyas relaciones topológicas son importantes, es decir, la estructura y las conexiones que hacen, y que abarcarán información sobre la representación resultante. El propósito de estos nodos es generar representaciones cuantificadas de los datos predichos.
- La capa de **nodos de entrada**, que su objetivo es alimentar a los nodos de aprendizaje por medio de vectores que tienen la información original de los datos de entrada durante el proceso de entrenamiento.

Los componentes de la primera capa están conectados a los componentes de la segunda capa, por lo que estos enlaces tienen pesos variables para que los nodos de entrenamiento cumplan su función de aprender información de los botones de entrada. Los parámetros de estos pesos varían según el modelo utilizado.

Ilustración 3: Representación esquemática de un Mapa Autoorganizado



Fuente: *Matías Carrasco Kind y Robert J. Brunner. [23]*

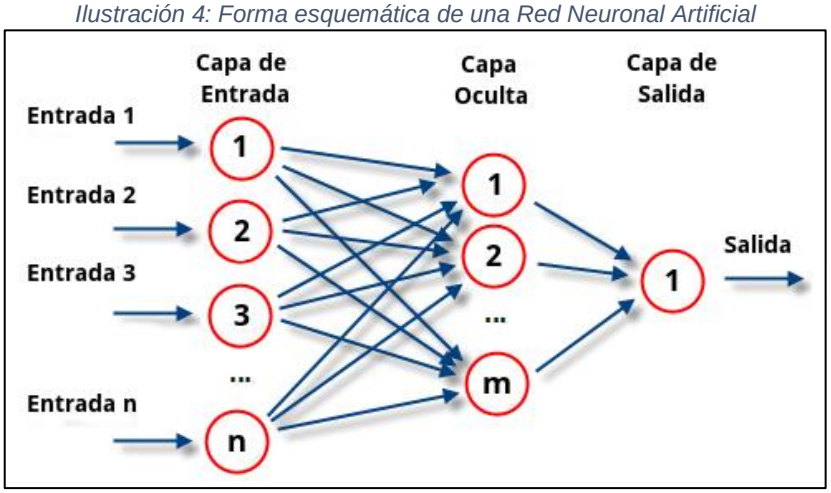
Durante el entrenamiento del mapa autoorganizado que se puede visualizar en la ilustración 3 cada nodo se puede representar mediante vectores de peso de la misma dimensión que el tamaño de la muestra de los datos de entrenamiento. En este proceso iterativo, los datos del conjunto de entrenamiento se utilizan individualmente para corregir los vectores de los pesos de modo que se modifique la neurona (o nodo). En un momento dado, los datos de entrada se representan mejor junto con los vectores de peso de sus neuronas vecinas, para convertirse en una mejor representación. Este proceso se repite para cada dato, y el mapa autoorganizado generalmente converge en unas pocas iteraciones a su forma final donde los datos de entrenamiento se separan en grupos de características similares por colores.

2.2.2.2. Red Neuronal Artificial

La Red Neural Artificial (ANN) es un modelo computacional basado en la estructura neurobiológica del cerebro; y cada enlace tiene un peso que indica la importancia de ese

enlace. La red consta de una capa de entrada, una capa oculta y una capa de salida. En la capa de entrada, el número de neuronas es el mismo que el número de entradas al proceso si no hay señal de polarización. Si se agrega un desplazamiento en la capa de entrada, se trata como una nueva neurona en la capa de entrada. Puede haber más de una capa oculta en una red neuronal artificial, pero usar una capa oculta es una aproximación universal. [24]

Ilustración 4: Forma esquemática de una Red Neuronal Artificial



Fuente: *ATRIA Innovation [25]*

Cada neurona de la capa oculta recibe una señal de cada neurona de la capa de entrada. Lo calcula combinándolos y aplicando una función de activación, y la capa final es la capa de salida. La red neuronal aprende a través de cambios internos para que ajuste su comportamiento al entorno. Cuando un agente externo está involucrado en el proceso de aprendizaje, se denomina aprendizaje supervisado. Este tipo de red se denomina relé de perceptrón de múltiples capas porque no tiene capas de retroalimentación repetidas y es el tipo de red más común para tratar en el caso de múltiples entradas y múltiples salidas.

En una red multicapa, el aprendizaje se realiza con el algoritmo retro propagación, un algoritmo de aprendizaje supervisado basado en la función de error que se minimiza en el espacio de los pesos. Este proceso funciona en dos pasos. El primer paso se llama entrenamiento, donde la red se inicializa con un pequeño número aleatorio de pesos. Es importante encontrar un conjunto de valores ponderados para minimizar el error global del sistema. Por el contrario, el segundo paso se llama generalización. El sistema ya ha aprendido una representación interna de los patrones presentados anteriormente y puede clasificar patrones nuevos presentados como entrada. La RNA ofrece muchas ventajas: neurobiológico, no lineal, proceso adaptativo, mapeo de entrada-salida,

tolerante a fallas, diseño uniforme, análisis, etc. Aparece en la mayoría de las clasificaciones de estilos de aprendizaje. [26]

2.2.3. Construcción del modelo

2.2.3.1. Metodología CRISP–DM

CRISP-DM (Cross Industry Standard Process for Data Mining, por sus siglas en inglés) es una metodología de proceso independiente para el Data Mining (minería de datos). [27]

Consta de seis pasos iterativos desde la comprensión del negocio hasta la implementación. La Tabla 1 resume la idea principal, el objetivo y el resultado de estos pasos.

Tabla 1: Descripciones del proceso CRISP-DM

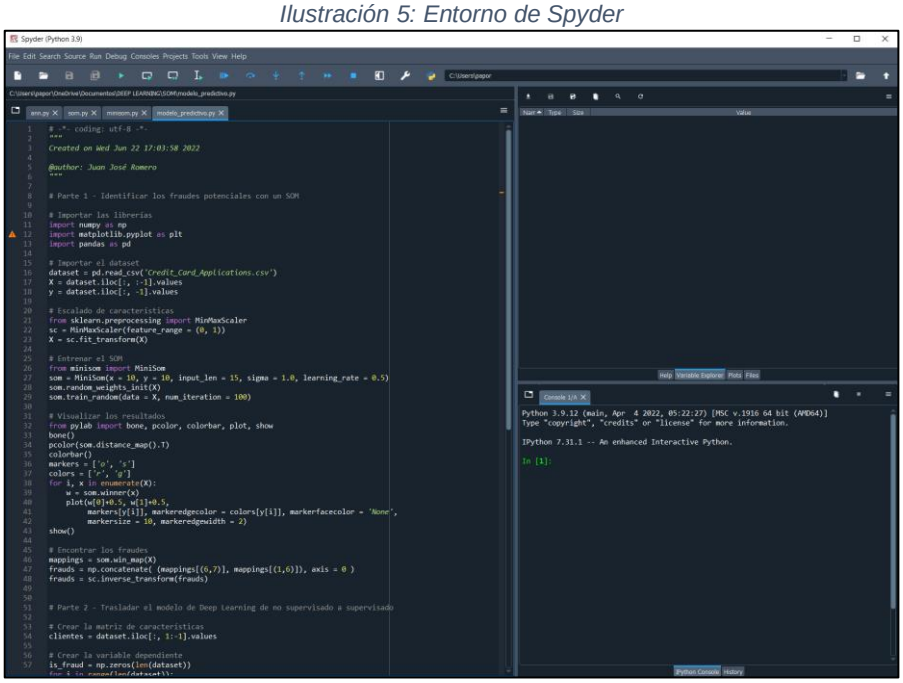
Fase	Breve descripción
Comprensión del Negocio	La comprensión del negocio debe evaluarse para obtener una visión general de los recursos disponibles y necesarios. La determinación del objetivo de la minería de datos es uno de los aspectos más importantes en esta fase. En primer lugar, se debe explicar el tipo de minería de datos (p. ej., clasificación) y los criterios de éxito de la minería de datos (como la precisión). Se debe crear un plan de proyecto obligatorio.
Comprensión de los Datos	La recopilación de datos de las fuentes de datos, la investigación, la caracterización y la verificación de la calidad de los datos son tareas importantes en esta fase. Para aclarar esto, la Guía del usuario describe la tarea de describir los datos utilizando análisis estadísticos y definir sus atributos y relaciones.
Preparación de los Datos	La selección de datos debe hacerse definiendo criterios de inclusión y exclusión. La mala calidad de los datos se puede eliminar mediante la limpieza de datos. Según el modelo utilizado (definido en el primer paso), debe crear propiedades derivadas. Se pueden realizar diferentes métodos para todos estos pasos según el modelo.
Modelado	Este paso incluye elegir un método de modelado, construir un caso de prueba y construir un modelo. Se pueden utilizar todos los métodos de minería de datos. En general, la elección depende de la misión empresarial y de los datos. Más importante es cómo interpretar la elección. Para construir un modelo, necesita establecer ciertos parámetros. Para evaluar el modelo, es necesario evaluar el modelo de acuerdo con los criterios de evaluación y seleccionar el mejor.
Evaluación	En este paso, los resultados se comparan con los objetivos comerciales definidos. Por lo tanto, es necesario interpretar los resultados y definir acciones posteriores. Otro punto es que todo el proceso necesita ser revisado.
Despliegue	La fase de implementación se describe de forma general en la guía del usuario. Podría ser un informe final o un componente de software. La guía del usuario describe que la fase de implementación consiste en planificar la implementación, la supervisión y el mantenimiento.

Fuente: *Procedia Informática [27]*

2.2.3.2. IDE Spyder

Spyder (Scientific Python Development Enviroment, por sus siglas en inglés) es un potente IDE (Integrated Development Environment, por sus siglas en inglés) también es una multiplataforma open source (código abierto) ideal para los científicos y analista de datos en el lenguaje Python. Este IDE se integra con una gran cantidad de paquetes de software protuberantes y sorprendentes en la pila científica de Python, que incluyen bibliotecas adicionales como Numpy, Editor, SciPy, Matplotlib, Pandas y Cython, seguidas de otras capacidades adicionales de fuentes abiertas. [28]

Ilustración 5: Entorno de Spyder



Fuente: *Elaboración propia*

Spyder ofrece una manera conveniente de trabajar con archivos de Python, presenta tres paneles:

- Editor: utilizado para escribir y editar programas. (es compatible con la opción Ejecutar para obtener mejores resultados).

- Consola de IPython: se utiliza para escribir provisiones de Python usando la ventana de la consola.

- Explorador de ayuda, variables y archivo.

2.3. Objetivos del prototipo

2.3.1. Objetivo General

Desarrollar un modelo predictivo para detectar fraudes bancarios aplicando técnicas de Deep Learning.

2.3.2. Objetivos Específicos

- Investigar y revisar las técnicas de aprendizaje profundo para la construcción del modelo predictivo.
- Investigar que dataset es el óptimo para el preprocesamiento de datos y entrenamiento del modelo.
- Diseñar un modelo no supervisado para el preprocesamiento de los datos.
- Diseñar, entrenar y mejorar una red neuronal artificial a partir de un modelo no supervisado.

2.4. Diseño del prototipo

Para el diseño del prototipo se utilizó cuatro fases del modelo CRISP-DM [27]:

Comprensión de los Datos: se obtuvo la data del sitio web UCI Machine Learning Repository [29] y se determinó qué características del dataset se va a utilizar para el pre procesado de datos.

Preparación de los Datos: La segunda parte del diseño consiste en realizar el pre procesado de los datos del modelo no supervisado, ya que Python no cuenta con una implementación para un Mapa Autoorganizado se recurrió a utilizar la clase ‘minisom.py’ [4] que fue elaborada por el desarrollador Giuseppe Vettigli [30].

Modelado: En esta fase se seleccionó los métodos para poder construir un mapa autoorganizado, entrenar el mapa, obtener resultados y a partir de los resultados se obtuvo una lista de clientes para convertirla en una variable dependiente y así poder crear la red neuronal artificial, la cual también fue entrenada y mejorada.

Evaluación: En esta fase se evaluó y se comparó los resultados obtenidos del modelo predictivo.

2.4.1. Requisitos

2.4.1.1. Hardware

Tabla 2: Características del equipo

Características del equipo	
Placa madre	Micro-Star International Co., Ltd. Z370 GAMING PLUS (MS-7B61)
Procesador	Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
Memoria RAM	32 GB
Tarjeta Gráfica	NVIDIA GeForce RTX 3080 (10 GB)
Disco Duro #1	Samsung SSD 970 EVO 1TB
Disco Duro #2	Seagate Barracuda de 2TB
Monitor	Dell Computer Corp. Dell S2417DG (2560x1440 / 27 Inch)
Fuente de poder	Corsair RM850x de 850 Watts
Sistema Operativo	Microsoft Windows 11 Pro

Fuente: *Elaboración propia*

2.4.1.2. Software

Tabla 3: Softwares utilizados para la elaboración del modelo

Software (librerías, programas, etc.)	Versión
Anaconda Navigator	2.2.0
Spyder	5.2.2
Python	3.9.12
Numpy	1.21.5
Pandas	1.4.2
Matplotlib	3.5.1
Sklearn	1.1.2
Keras	2.9.0
CUDA Toolkit	11.2
cuDNN	8.1.0

Fuente: *Elaboración propia*

2.4.2. Dataset

El dataset usado para el modelo predictivo se llama ‘Credit_Card_Applications.csv’, este dataset fue encontrado en el repositorio de Machine Learning de la Universidad Pública de Irvine, California [29]

Este dataset contiene información de clientes que solicitan una tarjeta de crédito de un banco australiano que se han obtenido al través de rellenar un formulario. Todos los nombres y valores de los atributos han sido modificados a símbolos sin sentido para proteger la confidencialidad de los datos.

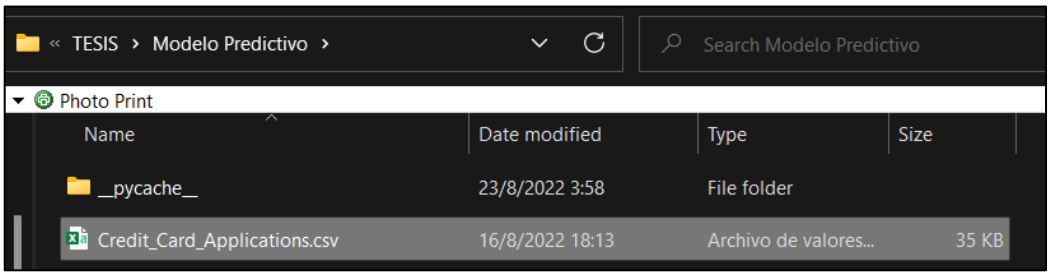
Este dataset es interesante porque cuenta con una combinación óptima de atributos: continuo, nominal con una pequeña cantidad de valores y nominal con una gran cantidad de valores. También hay algunos valores perdidos. Contiene las siguientes características:

Tabla 4: Características del Dataset

Características del conjunto de datos	Multivariado
Características de los atributos	Categórico, Entero, Real
Tareas asociadas	Clasificación
Número de instancias (filas)	690
Número de atributos (columnas)	14
¿Valores faltantes?	Sí
Área	Financiero

Fuente: *Elaboración propia*

Ilustración 6: Archivo del Dataset



Fuente: *Elaboración propia*

El conjunto de datos ‘Credit_Card_Applications.csv’ cuenta con 690 registros de clientes de un banco. Existen 6 atributos numéricos y 8 categóricos. Las etiquetas han sido cambiadas para conveniencia de los algoritmos estadísticos. Por ejemplo, el atributo 4 originalmente tenía 3 etiquetas p, g, gg y estas se cambiaron a las etiquetas 1, 2, 3.

Tabla 5: Información de los atributos del conjunto de datos

Atributos	Tipo de Atributo
A-1	0,1 CATEGÓRICO (antes: a,b)
A-2	Continuo
A-3	Continuo
A-4	1,2,3 CATEGÓRICO (antes: p,g,gg)
A-5	1,2,3,4,5,6,7,8,9,10,11,12,13,14 CATEGÓRICO (antes: ff,d,i,k,j,aa,m,c,w,e,q,r,cc,x)
A-6	1,2,3,4,5,6,7,8,9 CATEGÓRICO (antes: ff,dd,j,bb,v,n,o,h,z)
A-7	Continuo
A-8	1, 0 CATEGÓRICO (antes: t, f)
A-9	1, 0 CATEGÓRICO (antes: t, f)
A-10	Continuo
A-11	1, 0 CATEGÓRICO (antes: t, f)
A-12	1, 2, 3 CATEGÓRICO (antes: s, g, p)
A-13	Continuo
A-14	Continuo
A-15	Atributo de clase 1, 2 (antes: +, -)

Fuente: *Elaboración propia*

2.4.3. Diseño del modelo predictivo

Para el diseño del modelo predictivo se hizo uso del IDE Spyder, en el cual se dividió en cuatro partes:

Parte1: Identificar los potenciales fraudes con la creación de un Mapa Autoorganizado

Dentro de esta etapa se siguió una secuencia de pasos para lograr cumplir con lo planteado en la parte 1, primero se procedió a importar las librerías a utilizar para la creación del mapa autoorganizado, luego se procedió a importar el dataset con el método 'read_csv' de la librería Pandas [31], se hizo un escalado de características con el método 'MinMaxScaler' [32] de la librería de Sklearn, se entrenó el mapa autoorganizado con métodos de la clase 'minisom.py', se visualizó los resultados y por último se encontró los potenciales fraudes haciendo un mapeo.

Ilustración 7: Parte 1 - Creación del Mapa Autoorganizado

D:\PAPO\UNIVERSIDAD\TESIS\Modelo Predictivo\modelo_predictivo.py

modelo_predictivo.py | minisom.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 22 17:03:58 2022

@author: Juan José Romero
"""

# Parte 1 - Identificar los fraudes potenciales con un SOM

# Importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el dataset
dataset = pd.read_csv('Credit_Card_Applications.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Escalado de características
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

# Entrenar el SOM
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 15, sigma = 1.0, learning_rate = 0.5)
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)

# Visualizar los resultados
from pylab import bone, pcolor, colorbar, plot, show
bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's']
colors = ['r', 'g']
for i, x in enumerate(X):
    w = som.winner(x)
    plot(w[0]+0.5, w[1]+0.5,
         markers[y[i]], markeredgecolor = colors[y[i]], markerfacecolor = 'None',
         markersize = 10, markeredgewidth = 2)
show()

# Encontrar los fraudes
mappings = som.win_map(X)
frauds = np.concatenate( (mappings[(6,7)], mappings[(1,6)]), axis = 0 )
frauds = sc.inverse_transform(frauds)
```

Fuente: *Elaboración propia*

Parte 2: Trasladar el modelo no supervisado a supervisado

En esta parte lo primero que se hizo fue crear la matriz de características, luego se determinó cual iba a ser la variable dependiente y por último se hizo un escalado de variables utilizando el método 'fit_transform' de la librería sklearn. [33]

Ilustración 8: Parte 2 - Traslado del modelo no supervisado a supervisado

```
# Parte 2 - Trasladar el modelo no supervisado a supervisado

# Crear la matriz de características
clientes = dataset.iloc[:, 1:-1].values

# Crear la variable dependiente
is_fraud = np.zeros(len(dataset))
for i in range(len(dataset)):
    if dataset.iloc[i, 0] in frauds:
        is_fraud[i] = 1

# Escalado de variables
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
clientes = sc.fit_transform(clientes)
```

Fuente: *Elaboración propia*

Parte 3: Construcción de la RNA (red neuronal artificial)

En la creación de la red neuronal primero se importó las librerías a utilizar, se inicializó la red neuronal con una variable 'classifier' de un modelo 'Sequential' de la librería Keras [34] luego se procedió a agregar las capas de entrada y la única capa oculta, luego se añadió la capa de salida, para las capas de la red se utilizó el método 'Dense' [35] de la librería Keras.layers con la activación 'sigmoidal' [36]. Por último, se procedió a compilar la red neuronal con el método 'compile' donde se utilizó el optimizador = 'adam' [37] y ajustarla al conjunto de entrenamiento con el método 'fit' [38] se utilizó un 'batch_size = 1' y con 'epochs = 2'.

Ilustración 9: Parte 3 - Construcción de la Red Neuronal Artificial

```
68  # Parte 3 - Construir la RNA
69
70  # Importar Keras y librerías adicionales
71  from keras.models import Sequential
72  from keras.layers import Dense
73
74  # Inicializar la RNA
75  classifier = Sequential()
76
77  # Añadir las capas de entrada y primera capa oculta
78  classifier.add(Dense(units = 2, kernel_initializer = "uniform",
79                       activation = "relu", input_dim = 14))
80
81  # Añadir la capa de salida
82  classifier.add(Dense(units = 1, kernel_initializer = "uniform",
83                       activation = "sigmoid"))
84
85  # Compilar la RNA
86  classifier.compile(optimizer = "adam", loss = "binary_crossentropy",
87                     metrics = ["accuracy"])
88
89  # Ajustamos la RNA al conjunto de Entrenamiento
90  classifier.fit(clientes, is_fraud, batch_size = 1, epochs = 2)
91
```

Fuente: *Elaboración propia*

Parte 4: Evaluar el modelo y calcular predicciones finales

En la última parte del diseño del modelo predictivo se evaluó el modelo y se calcularon las predicciones finales con el método 'predict' [39] de la clase 'Sequential ()' que nos proporciona la librería Keras.

Ilustración 10: Parte 4 - Evaluación del modelo y cálculo de predicciones finales

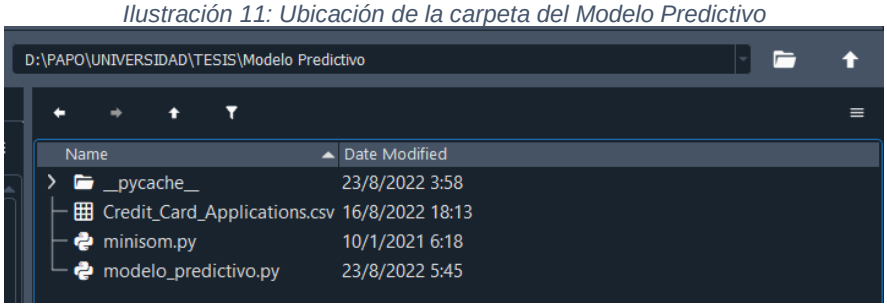
```
92
93  # Parte 4 - Evaluar el modelo y calcular predicciones finales
94
95  # Predicción de los resultados con el Conjunto de Testing
96  y_pred = classifier.predict(clientes)
97  y_pred = np.concatenate((dataset.iloc[:,0:1].values, y_pred), axis = 1)
98  y_pred = y_pred[y_pred[:,1].argsort()]
99
```

Fuente: *Elaboración propia*

2.5. Ejecución y/o ensamblaje del prototipo

Para lograr el funcionamiento del modelo diseñado, se debe ejecutar cada parte del diseño del modelo para eso lo primero que debemos de hacer en el IDE Spyder es ubicarnos en la carpeta donde se encuentre las clases y el conjunto de datos.

Ilustración 11: Ubicación de la carpeta del Modelo Predictivo



Fuente: *Elaboración propia*

Luego se ejecuta en orden las líneas de código de la clase 'modelo_predictivo.py':

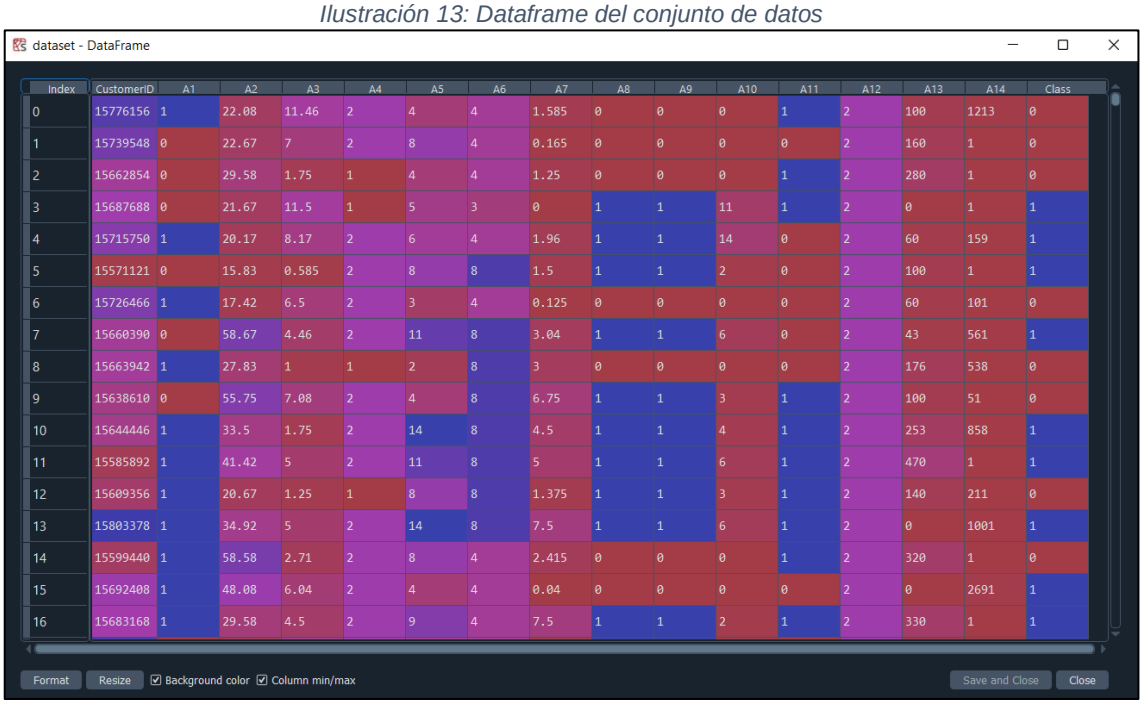
Ilustración 12: Importación de librerías y del Dataset

```
In [1]: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: import pandas as pd

In [2]: dataset = pd.read_csv('Credit_Card_Applications.csv')
```

Fuente: *Elaboración propia*

Ilustración 13: Dataframe del conjunto de datos



Index	CustomerID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	Class
0	15776156	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213	0
1	15739548	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1	0
2	15662854	0	29.58	1.75	1	4	4	1.25	0	0	0	1	2	280	1	0
3	15687688	0	21.67	11.5	1	5	3	0	1	1	11	1	2	0	1	1
4	15715750	1	20.17	8.17	2	6	4	1.96	1	1	14	0	2	60	159	1
5	15571121	0	15.83	0.585	2	8	8	1.5	1	1	2	0	2	100	1	1
6	15726466	1	17.42	6.5	2	3	4	0.125	0	0	0	0	2	60	101	0
7	15660390	0	58.67	4.46	2	11	8	3.04	1	1	6	0	2	43	561	1
8	15663942	1	27.83	1	1	2	8	3	0	0	0	0	2	176	538	0
9	15638610	0	55.75	7.08	2	4	8	6.75	1	1	3	1	2	100	51	0
10	15644446	1	33.5	1.75	2	14	8	4.5	1	1	4	1	2	253	858	1
11	15585892	1	41.42	5	2	11	8	5	1	1	6	1	2	470	1	1
12	15609356	1	20.67	1.25	1	8	8	1.375	1	1	3	1	2	140	211	0
13	15803378	1	34.92	5	2	14	8	7.5	1	1	6	1	2	0	1001	1
14	15599440	1	58.58	2.71	2	8	4	2.415	0	0	0	1	2	320	1	0
15	15692408	1	48.08	6.04	2	4	4	0.04	0	0	0	0	2	0	2691	1
16	15683168	1	29.58	4.5	2	9	4	7.5	1	1	2	1	2	330	1	1

Fuente: *Elaboración propia*

Ahora lo que se procede a realizar es separar la columna del atributo clase que se encuentra a final que es donde nos indica si la solicitud de la tarjeta de crédito por parte del cliente fue aprobada o no, un 1 nos indica que si fue aprobada y un 0 nos indica que no fue aprobada, por lo tanto, se va a separar todas y cada una de las variables que se

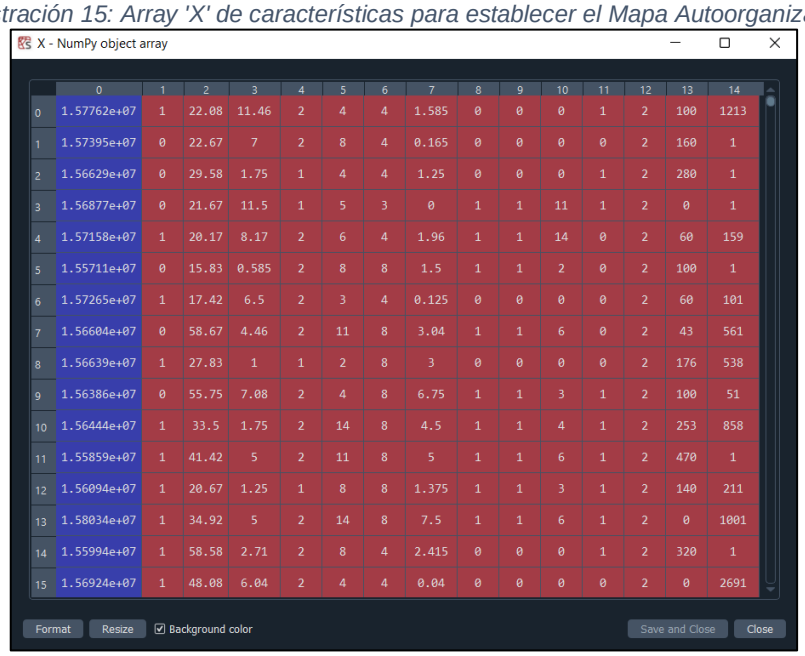
van a utilizar para medir distancias entre clientes y evidentemente la clase si se le otorga o no se le otorga la tarjeta de crédito sería la predicción que queremos que haga el mapa autoorganizado en última instancia para ver si el formulario que llenó el cliente era o no fraudulento.

Ilustración 14: Código para separar la columna 'class' del Dataframe

```
In [3]: X = dataset.iloc[:, :-1].values
   ...: y = dataset.iloc[:, -1].values
```

Fuente: *Elaboración propia*

Ilustración 15: Array 'X' de características para establecer el Mapa Autoorganizado



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.57762e+07	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213
1	1.57395e+07	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1
2	1.56629e+07	0	29.58	1.75	1	4	4	1.25	0	0	0	1	2	280	1
3	1.56877e+07	0	21.67	11.5	1	5	3	0	1	1	11	1	2	0	1
4	1.57158e+07	1	20.17	8.17	2	6	4	1.96	1	1	14	0	2	60	159
5	1.55711e+07	0	15.83	0.585	2	8	8	1.5	1	1	2	0	2	100	1
6	1.57265e+07	1	17.42	6.5	2	3	4	0.125	0	0	0	0	2	60	101
7	1.56604e+07	0	58.67	4.46	2	11	8	3.04	1	1	6	0	2	43	561
8	1.56639e+07	1	27.83	1	1	2	8	3	0	0	0	0	2	176	538
9	1.56386e+07	0	55.75	7.08	2	4	8	6.75	1	1	3	1	2	100	51
10	1.56444e+07	1	33.5	1.75	2	14	8	4.5	1	1	4	1	2	253	858
11	1.55859e+07	1	41.42	5	2	11	8	5	1	1	6	1	2	470	1
12	1.56094e+07	1	20.67	1.25	1	8	8	1.375	1	1	3	1	2	140	211
13	1.58034e+07	1	34.92	5	2	14	8	7.5	1	1	6	1	2	0	1001
14	1.55994e+07	1	58.58	2.71	2	8	4	2.415	0	0	0	1	2	320	1
15	1.56924e+07	1	48.08	6.04	2	4	4	0.04	0	0	0	0	2	0	2691

Fuente: *Elaboración propia*

Ilustración 16: Array 'y' que obtiene la columna 'class'

	0
0	0
1	0
2	0
3	1
4	1
5	1
6	0
7	1
8	0
9	0
10	1
11	1
12	0

Fuente: *Elaboración propia*

Como se aprecia en las ilustraciones 14, 15 y 16 se procedió a separar en dos partes, el array ‘X’ serán los datos utilizados para obtener las características y observaciones que se van a utilizar para la creación del mapa autoorganizado. Y en el array ‘y’ estará la última columna que es la que nos indica si el cliente recibió o no la tarjeta de crédito.

Esto se hizo con la intención para hacer la distinción entre los clientes que fueron aprobados y los que no fueron aprobados para obtener la tarjeta de crédito para luego poder evaluar si la técnica para detectar los valores alejados y con pocos vecinos funciona realmente en este caso, esto es una técnica no supervisada porque no se considera una variable dependiente para la elaboración del mapa autoorganizado.

Para lograr esto, se requiere hacer un escalado de características:

Ilustración 17: Código para el escalado de características

```
In [4]: from sklearn.preprocessing import MinMaxScaler
   ...: sc = MinMaxScaler(feature_range = (0, 1))
   ...: X = sc.fit_transform(X)
```

Fuente: *Elaboración propia*

Ilustración 18: Dataset escalado y normalizado con valores de 0 a 1

X - NumPy object array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.842681	1	0.125263	0.409286	0.5	0.230769	0.375	0.055614	0	0	0	1	0.5	0.05	0.01212
1	0.696091	0	0.134135	0.25	0.5	0.538462	0.375	0.00578947	0	0	0	0	0.5	0.08	0
2	0.388982	0	0.238045	0.0625	0	0.230769	0.375	0.0438596	0	0	0	1	0.5	0.14	0
3	0.488425	0	0.119098	0.410714	0	0.307692	0.25	0	1	1	0.164179	1	0.5	0	0
4	0.600795	1	0.0965414	0.291786	0.5	0.384615	0.375	0.0687719	1	1	0.208955	0	0.5	0.03	0.00158
5	0.0216515	0	0.0312782	0.0208929	0.5	0.538462	0.875	0.0526316	1	1	0.0298507	0	0.5	0.05	0
6	0.643706	1	0.055188	0.232143	0.5	0.153846	0.375	0.00438596	0	0	0	0	0.5	0.03	0.001
7	0.379115	0	0.675489	0.159286	0.5	0.769231	0.875	0.106667	1	1	0.0895522	0	0.5	0.0215	0.0056
8	0.393338	1	0.211729	0.0357143	0	0.0769231	0.875	0.105263	0	0	0	0	0.5	0.088	0.00537
9	0.2919	0	0.631579	0.252857	0.5	0.230769	0.875	0.236842	1	1	0.0447761	1	0.5	0.05	0.0005
10	0.31527	1	0.296992	0.0625	0.5	1	0.875	0.157895	1	1	0.0597015	1	0.5	0.1265	0.00857
11	0.0807996	1	0.41609	0.178571	0.5	0.769231	0.875	0.175439	1	1	0.0895522	1	0.5	0.235	0
12	0.174757	1	0.10406	0.0446429	0	0.538462	0.875	0.0482456	1	1	0.0447761	1	0.5	0.07	0.0021
13	0.951688	1	0.318346	0.178571	0.5	1	0.875	0.263158	1	1	0.0895522	1	0.5	0	0.01
14	0.13505	1	0.674135	0.0967857	0.5	0.538462	0.375	0.0847368	0	0	0	1	0.5	0.16	0
15	0.507326	1	0.516241	0.215714	0.5	0.230769	0.375	0.00140351	0	0	0	0	0.5	0	0.0269
16	0.470326	1	0.238045	0.160714	0.5	0.615385	0.375	0.263158	1	1	0.0298507	1	0.5	0.165	0

Format Resize Background color Save and Close Close

Fuente*: Elaboración propia*

Se escaló todas las variables en el rango (0, 1) para poder aplicar nuestro mapa autoorganizado.

Ilustración 19: Entrenamiento del Mapa Autoorganizado

```
In [5]: from minisom import MiniSom
   ...: som = MiniSom(x = 10, y = 10, input_len = 15, sigma = 1.0,
learning_rate = 0.5)
   ...: som.random_weights_init(X)
   ...: som.train_random(data = X, num_iteration = 100)
```

Fuente: *Elaboración propia*

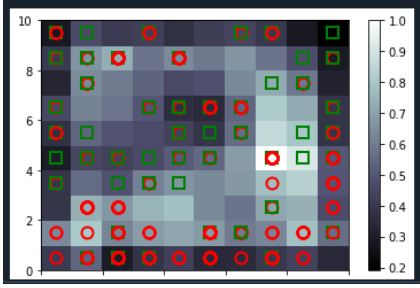
Como nuestro dataset cuenta con 690 registros de clientes, se utilizó una matriz de 10x10 para obtener la mayoría de valores concentrados y los valores atípicos ligeramente separados , también se usó un 'input_len' (longitud de entrada) de 15 que quiere decir que empezaremos desde el espacio vectorial 15 para poder establecer las proyecciones del mapa autoorganizado, también se usó un valor de sigma = 1.0 (radio inicial) para que los nodos vecinos se vean atraídos por todos los que están dentro de ese radio inicial y luego ese valor se va actualizando automáticamente dentro del algoritmo de la clase 'minisom.py' [4], por último se usó un valor de 0.5 en el 'learning_rate' (tasa de aprendizaje) que es la capacidad de ir adaptando los valores de los pesos de una iteración a la siguiente.

Después se inicializa los pesos con la función 'random_weights_init' de la clase 'minisom.py' [4], esta función sirve para inicializar los pesos aleatoriamente, lo único que hay que hacer en este método es darle los datos, en nuestro caso los datos es la matriz 'X' que se va a utilizar para entrenar y para terminar una vez que tengamos el método preparado para entrenar el mapa autoorganizado tiene que entrenar en base a los datos de 'X' con el método 'train_random', en este caso, solo se ingresan dos argumentos: el dataset y el número de iteraciones. En nuestro caso, se utilizan 100 iteraciones para hacer que nuestro conjunto de datos colapse automáticamente un mapa autoorganizado.

Ahora se procede a ejecutar el paso de la visualización del mapa autoorganizado para ver como quedó el entrenamiento.

Ilustración 20: Código para visualizar los resultados del Mapa Autoorganizado

```
In [6]: from pylab import bone, pcolor, colorbar, plot, show
   ...: bone()
   ...: pcolor(som.distance_map().T)
   ...: colorbar()
   ...: markers = ['o', 's']
   ...: colors = ['r', 'g']
   ...: for i, x in enumerate(X):
   ...:     w = som.winner(x)
   ...:     plot(w[0]+0.5, w[1]+0.5,
   ...:          markers[y[i]], markeredgecolor = colors[y[i]],
markerfacecolor = 'None',
   ...:          markersize = 10, markeredgewidth = 2)
   ...: show()
```



Fuente: *Elaboración propia*

Para la visualización, primero se utilizó la función ‘bone’ de la librería pylab que nos permite establecer la ventana del dibujo, lo siguiente que se ejecutó fue detectar los nodos ganadores del mapa, o sea, los nodos que tienen alguna de las observaciones más cercanas. Para poder detectarlos se hizo una iteración sobre todas las neuronas ganadoras que el mapa autoorganizado ha detectado utilizando un rango de colores que corresponderán a diferentes valores del rango de distancia media entre las neuronas, para ello se utilizó la función ‘pcolor’ que permitirá hacer un gradiente para poder agregar los colores de distancias medias entre neuronas de forma automática en el mapa autoorganizado y para la obtención de la distancia media se utilizó el método específico ‘distance_map()’ de la clase ‘minisom.py’ [4]. Al método se le aplicó la función propia de Python para Trasponer una matriz ya que el método ‘distance_map()’ por defecto las observaciones vienen en filas y queremos que se refleje en columnas, por ende, se aplicó la traspuesta: ‘distance_map().T’.

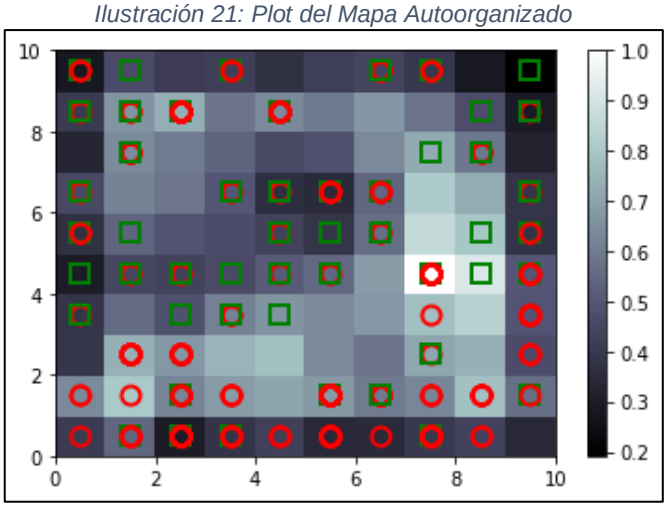
Para saber si los nodos de colores más oscuros son los que más vecinos cerca de ellos o son los que tienen menos vecinos cerca, se procedió a ejecutar la línea de código #35 que se muestra en la ilustración 20 para agregar una leyenda al dibujo de la ilustración 21 que nos indica que los nodos que tienen los vecinos más alejados son los colores más blancos y los nodos que tienen los vecinos más cercanos son los de color negro.

Ahora para saber la lista de clientes especifica se aplicó ingeniería inversa para saber si un cliente cae en una zona de pocos vecinos y si es así, poder asociar ese nodo ganador a una serie de clientes determinados o a un cliente determinado. Para lograr esto, primero se marcó dos marcadores con la ejecución de las líneas de código #36 y #37

respectivamente que se visualiza en la ilustración 20; un círculo de color rojo (para los clientes que no han tenido aprobación) y un cuadrado de color verde (para los clientes que si han tenido aprobación). Luego se asoció los clientes en los nodos ganadores mediante el uso de una iteración 'for' que se puede visualizar en la línea de código #38 de la ilustración 20.

Una vez ejecutado cada línea de código de la parte de la visualización, se finaliza con el método 'show()' para visualizar en una ventana lo que se había ejecutado con anterioridad, dándonos como resultado el dibujo que se muestra en la ilustración 21.

Ilustración 21: Plot del Mapa Autoorganizado



***Fuente**: Elaboración propia*

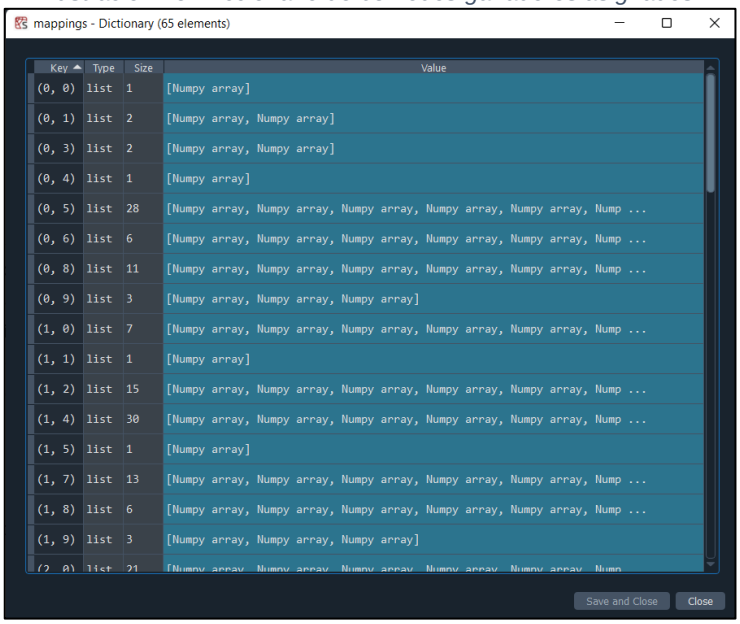
En esta ilustración podemos observar que los clientes asociados a las tarjetas fraudulentas están de color rojo con un círculo y los que si obtuvieron la tarjeta están con un cuadrado de color verde. Como podemos darnos cuenta los círculos rojos se encuentran cercanos a los nodos de color blanco (valores de 0.8 a 1.0), esto quiere decir, que estos nodos claramente son clientes que no obtuvieron la aprobación y todos los cuadrados verdes que se encuentran en los nodos de color más oscuros (valores de 0.1 a 0.4) son los clientes que si obtuvieron la aprobación de la tarjeta de crédito.

Ilustración 22: Mapeo para obtener el diccionario de los posibles fraudes

```
In [7]: mappings = som.win_map(X)
```

***Fuente:** Elaboración propia*

Ilustración 23: Diccionario de 65 nodos ganadores asignados



Fuente: *Elaboración propia*

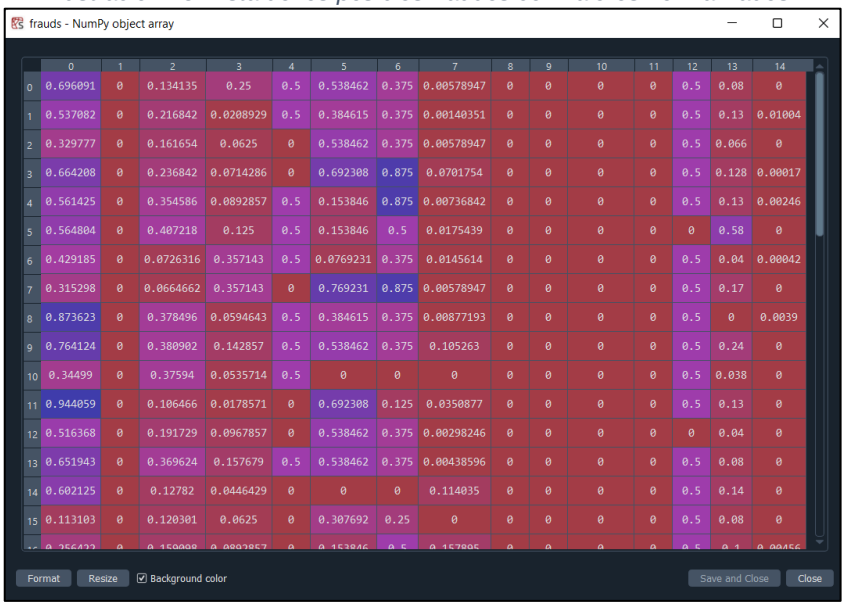
Ahora se procedió a ejecutar la parte de encontrar los posibles fraudes; como no existe una función de mapeo inverso que nos permita obtener directamente la lista de los clientes fraudulentos a partir de las coordenadas de los nodos ganadores, se procedió a utilizar el método ‘win_map’ que viene incluido en la clase ‘minisom.py’ [4] que se muestra en la ilustración 22, este método nos devuelve un diccionario con todas las asignaciones diferentes de los nodos ganadores directamente a los clientes.

Ilustración 24: Código para obtener la lista de los posibles fraudes

```
In [8]: frauds = np.concatenate( (mappings[(7,4)],
mappings[(1,1)]), axis = 0 )
```

Fuente: *Elaboración propia*

Ilustración 25: Lista de los posibles fraudes con valores normalizados



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.696091	0	0.134135	0.25	0.5	0.538462	0.375	0.00578947	0	0	0	0	0.5	0.08	0
1	0.537082	0	0.216842	0.0208929	0.5	0.384615	0.375	0.00140351	0	0	0	0	0.5	0.13	0.01004
2	0.329777	0	0.161654	0.0625	0	0.538462	0.375	0.00578947	0	0	0	0	0.5	0.066	0
3	0.664208	0	0.236842	0.0714286	0	0.692308	0.875	0.0701754	0	0	0	0	0.5	0.128	0.00017
4	0.561425	0	0.354586	0.0892857	0.5	0.153846	0.875	0.00736842	0	0	0	0	0.5	0.13	0.00246
5	0.564804	0	0.407218	0.125	0.5	0.153846	0.5	0.0175439	0	0	0	0	0	0.58	0
6	0.429185	0	0.0726316	0.357143	0.5	0.0769231	0.375	0.0145614	0	0	0	0	0.5	0.04	0.00042
7	0.315298	0	0.0664662	0.357143	0	0.769231	0.875	0.00578947	0	0	0	0	0.5	0.17	0
8	0.873623	0	0.378496	0.0594643	0.5	0.384615	0.375	0.00877193	0	0	0	0	0.5	0	0.0039
9	0.764124	0	0.380902	0.142857	0.5	0.538462	0.375	0.105263	0	0	0	0	0.5	0.24	0
10	0.34499	0	0.37594	0.0535714	0.5	0	0	0	0	0	0	0	0.5	0.038	0
11	0.944059	0	0.106466	0.0178571	0	0.692308	0.125	0.0350877	0	0	0	0	0.5	0.13	0
12	0.516368	0	0.191729	0.0967857	0	0.538462	0.375	0.00298246	0	0	0	0	0	0.04	0
13	0.651943	0	0.369624	0.157679	0.5	0.538462	0.375	0.00438596	0	0	0	0	0.5	0.08	0
14	0.602125	0	0.12782	0.0446429	0	0	0	0.114035	0	0	0	0	0.5	0.14	0
15	0.113103	0	0.120301	0.0625	0	0.307692	0.25	0	0	0	0	0	0.5	0.08	0

Fuente: *Elaboración propia*

Como podemos observar en las ilustraciones 24 y 25 se procedió a recorrer todas esas asignaciones una por una y utilizar las coordenadas de los nodos ganadores de los valores atípicos que hemos identificado anteriormente y así poder obtener una lista de los nodos ganadores con respecto a los clientes que caen en ese nodo, para eso se utilizó la función ‘concatenate’ de la librería NumPy, manualmente se colocó las coordenadas observando los valores atípicos en nuestro mapa autoorganizado y se utilizó la función ‘axis = 0’ para mostrar la lista de los posibles fraudes en filas.

Ilustración 26: Código para invertir la lista de los posibles fraudes

```
In [9]: frauds = sc.inverse_transform(frauds)
```

Fuente: *Elaboración propia*

Ilustración 27: Lista de los posibles fraudes con sus valores originales

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.57395e+07	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1
1	1.56998e+07	0	28.17	0.585	2	6	4	0.04	0	0	0	0	2	260	1005
2	1.56481e+07	0	24.5	1.75	1	8	4	0.165	0	0	0	0	2	132	1
3	1.57316e+07	0	29.5	2	1	10	8	2	0	0	0	0	2	256	18
4	1.57059e+07	0	37.33	2.5	2	3	8	0.21	0	0	0	0	2	260	247
5	1.57068e+07	0	40.83	3.5	2	3	5	0.5	0	0	0	0	1	1160	1
6	1.56729e+07	0	18.58	10	2	2	4	0.415	0	0	0	0	2	80	43
7	1.56445e+07	0	18.17	10	1	11	8	0.165	0	0	0	0	2	340	1
8	1.57839e+07	0	38.92	1.665	2	6	4	0.25	0	0	0	0	2	0	391
9	1.57565e+07	0	39.08	4	2	8	4	3	0	0	0	0	2	480	1
10	1.56519e+07	0	38.75	1.5	2	1	1	0	0	0	0	0	2	76	1
11	1.58015e+07	0	20.83	0.5	1	10	2	1	0	0	0	0	2	260	1
12	1.56947e+07	0	26.5	2.71	1	8	4	0.085	0	0	0	0	1	80	1
13	1.57285e+07	0	38.33	4.415	2	8	4	0.125	0	0	0	0	2	160	1
14	1.57161e+07	0	22.25	1.25	1	1	1	3.25	0	0	0	0	2	280	1
15	1.5594e+07	0	21.75	1.75	1	5	3	0	0	0	0	0	2	160	1

Fuente: *Elaboración propia*

Por último, en la elaboración de nuestro mapa autoorganizado, se tuvo que invertir la lista (escalado inverso de los valores) de los posibles fraudes obtenida anteriormente para obtener sus valores originales del dataset y saber que clientes son los que han hecho fraude, para ello se utilizó la función ‘inverse_transform’ de la librería Sklearn.

Dándonos como resultado 45 potenciales clientes fraudulentos.

Ahora procederemos a ejecutar la segunda parte del modelo predictivo, que es pasar de un mapa autoorganizado (aprendizaje no supervisado) a una red neuronal artificial (aprendizaje supervisado).

Ilustración 28: Creación de la matriz de características

```
In [10]: clientes = dataset.iloc[:, 1:-1].values
```

Fuente: *Elaboración propia*

Ilustración 29: Matriz de Características

clientes - NumPy object array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213
1	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1
2	0	29.58	1.75	1	4	4	1.25	0	0	0	1	2	280	1
3	0	21.67	11.5	1	5	3	0	1	1	11	1	2	0	1
4	1	20.17	8.17	2	6	4	1.96	1	1	14	0	2	60	159
5	0	15.83	0.585	2	8	8	1.5	1	1	2	0	2	100	1
6	1	17.42	6.5	2	3	4	0.125	0	0	0	0	2	60	101
7	0	58.67	4.46	2	11	8	3.04	1	1	6	0	2	43	561
8	1	27.83	1	1	2	8	3	0	0	0	0	2	176	538
9	0	55.75	7.08	2	4	8	6.75	1	1	3	1	2	100	51
10	1	33.5	1.75	2	14	8	4.5	1	1	4	1	2	253	858
11	1	41.42	5	2	11	8	5	1	1	6	1	2	470	1
12	1	20.67	1.25	1	8	8	1.375	1	1	3	1	2	140	211

Format Resize Background color Save and Close Close

Fuente: *Elaboración propia*

Para comprobar el porcentaje de efectividad del Mapa Autoorganizado se comparó los 45 potenciales clientes fraudulentos de la lista ‘frauds’ con el data set original y comprobar si acertó o no en la predicción, dándonos como resultado:

43 aciertos y 2 errores de los 45 clientes de la lista ‘frauds’, así que podemos decir que al mapa autoorganizado tiene un porcentaje de efectividad de un 95.56 %

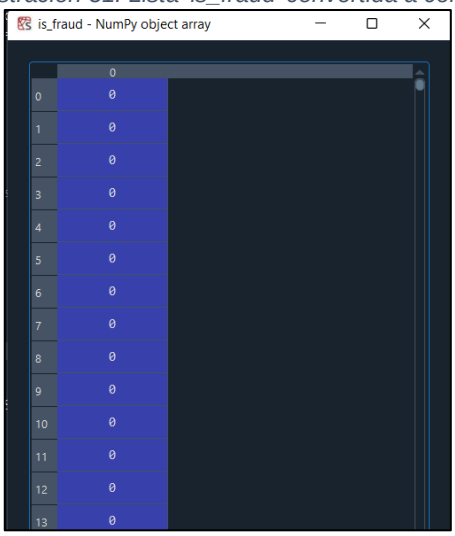
Para la creación de la matriz de características, como partimos de un mapa autoorganizado (modelo no supervisado) por ende, no existe la variable dependiente (en este caso el dataset original si tiene una variable dependiente llamada ‘class’) se creó la lista ‘clientes’ excluyendo la primera y última columna del dataset original para obtener solo las variables que describen si el cliente consiguió o no consiguió la tarjeta de crédito.

Ilustración 30: Creación de la variable dependiente 'is_fraud'

```
In [11]: is_fraud = np.zeros(len(dataset))
```

Fuente: *Elaboración propia*

Ilustración 31: Lista 'is_fraud' convertida a ceros



Fuente: *Elaboración propia*

Para obtener la lista que se muestra en la ilustración 31, se creó un vector con 690 ceros utilizando la función ‘zeros’ de NumPy suponiendo que todos los clientes del dataset original no han hecho fraude para luego buscar a que filas corresponden los clientes que aparecen dentro de la lista de ‘frauds’ obtenida del mapa autoorganizado.

Ilustración 32: Iteración 'for' para obtener la variable dependiente

```
In [12]: for i in range(len(dataset)):
    ...:     if dataset.iloc[i, 0] in frauds:
    ...:         is_fraud[i] = 1
```

Fuente: *Elaboración propia*

Para la creación de la variable dependiente ‘is_fraud’ vamos a suponer que no existe la variable ‘y = dataset.iloc[:, -1].values’ de la data set original que se puede visualizar en la ilustración 13 y vamos a crear una variable dependiente a partir de los resultados que nos dio el mapa autoorganizado en base si es una observación que procede del mapa autoorganizado entonces deberá ser marcada como fraudulenta y si es una observación que no proceda del mapa autoorganizado será marcada como no fraudulenta, por ende, el fraude será marcado como 1 si esa observación pertenece al mapa autoorganizado (variable ‘frauds’) y si los clientes que no se encuentran en esa salida del mapa, será marcado como un 0. Para lograr esto, se ejecutó una iteración ‘for’ que podemos observar en la ilustración 32.

Ilustración 33: Lista de la variable dependiente 'is_fraud'

Fuente: *Elaboración propia*

Para la obtención de la lista mostrada en la ilustración 33, se procedió a realizar un recorrido de forma inversa debido a que como hay menos fraudes (en este caso 45) que observaciones (690 en total), es decir, recorrer todas las observaciones y si la observación en cuestión en la fila actual está en la lista de fraudes ‘frauds’ se marcará con un valor de 1.

Ilustración 34: Escalado de variables

```
In [13]: from sklearn.preprocessing import
StandardScaler
    ...: sc = StandardScaler()
    ...: clientes = sc.fit_transform(clientes)
```

Fuente: *Elaboración propia*

Ilustración 35: Dataset 'clientes' con valores escalados

Fuente: *Elaboración propia*

El primer paso para crear la red neuronal artificial fue el de realizar el escalado de variables y la normalización de los datos que se pueden apreciar en las ilustraciones 34 y 35.

Ilustración 36: Creación de la Red Neuronal Artificial

```
In [14]: from keras.models import Sequential
    ...: from keras.layers import Dense
    ...:
    ...: # Inicializar la RNA
    ...: classifier = Sequential()
    ...:
    ...: # Añadir las capas de entrada y una capa oculta
    ...: classifier.add(Dense(units = 2, kernel_initializer = "uniform",
    ...:                       activation = "relu", input_dim = 14))
    ...:
    ...: # Añadir la capa de salida
    ...: classifier.add(Dense(units = 1, kernel_initializer = "uniform",
    ...:                       activation = "sigmoid"))
    ...:
    ...: # Compilar la RNA
    ...: classifier.compile(optimizer = "adam", loss =
"binary_crossentropy",
    ...:                    metrics = ["accuracy"])
    ...:
    ...: # Ajustamos la RNA al conjunto de Entrenamiento
    ...: classifier.fit(clientes, is_fraud, batch_size = 1, epochs = 2)
```

Fuente: *Elaboración propia*

Como podemos observar en la ilustración 36 se ejecutó la red neuronal artificial con dos capas de entrada, una capa oculta y una capa de salida. Como no son muchos datos (solo 690) se optó por entrenar esta red neuronal con un ‘batch_size’ con valor de 1 y solo con 2 épocas, esto quiere decir, que la red neuronal va a recorrer los datos de uno en uno y solo va hacer el recorrido dos veces.

Ilustración 37: Resultados del entrenamiento de la RNA

```
Epoch 1/2
690/690 [==============================] - 2s 2ms/step -
loss: 0.5095 - accuracy: 0.9304
Epoch 2/2
690/690 [==============================] - 1s 2ms/step -
loss: 0.2082 - accuracy: 0.9348
Out[15]: <keras.callbacks.History at 0x211b8ed55e0>
```

Fuente: *Elaboración propia*

Como se aprecia en la ilustración 37, el valor de precisión 'accuracy' se mantuvo en las 2 épocas en un 93% mientras que el valor de pérdida 'loss' se redujo de un 0.5095 a un 0.2082.

Por último, se ejecutó la predicción de los resultados de fraude.

Ilustración 38: Código para la predicción de los resultados

```
In [16]: y_pred = classifier.predict(clientes)
22/22 [==============================] - 0s 1ms/step

In [17]: y_pred = np.concatenate((dataset.iloc[:,
0:1].values, y_pred), axis = 1)
    ...: y_pred = y_pred[y_pred[:,1].argsort()]
```

Fuente: *Elaboración propia*

Ilustración 39: Resultados de la predicción

Fuente: *Elaboración propia*

Para la obtención de los resultados de la predicción, se ejecutó el método 'predict' de la librería Keras y se concatenó el id de los clientes del dataset original con la columna de 'y_pred' como se puede observar en la ilustración 39 para mostrar en la columna izquierda de color azul los clientes y en la otra columna de color rojo la probabilidad de que el formulario del cliente sea fraudulento. Se ejecutó la función propia de Python 'argsort' para ordenar las predicciones de menor a mayor.

3. CAPÍTULO III. EVALUACIÓN DEL PROTOTIPO

3.1. Plan de evaluación

Para la evaluación del modelo predictivo se elaboró una tabla comparativa de los resultados, comparando el porcentaje de precisión de los mapas autoorganizados y el porcentaje de precisión y pérdida del entrenamiento de la red neuronal artificial.

Cabe recalcar que para la evaluación del modelo predictivo como no es un modelo para clasificación de datos no se hizo uso de la matriz de confusión.

Para la evaluación del prototipo se realizó 4 pruebas modificando algunos parámetros del mapa autoorganizado.

Para la primera prueba se utilizó una matriz de 10x10 para el mapa autoorganizado con un mapeo de 2 coordenadas, valor de longitud de entrada igual a 15, valor de radio de 1.0, tasa de aprendizaje de 0.5 y un número de 100 iteraciones.

Para la segunda prueba se utilizó una matriz de 15x15 con un mapeo de 3 coordenadas, con un valor de longitud de entrada de 15, valor de radio de 1.0, tasa de aprendizaje de 0.5 y un número de 100 iteraciones.

Para la tercera prueba se utilizó una matriz de 20x20 con un mapeo de 3 coordenadas, con un valor de longitud de entrada de 15, valor de radio de 1.0, tasa de aprendizaje de 0.5 y un número de 150 iteraciones.

Para la cuarta y última prueba se utilizó una matriz de 30x30 con un mapeo de 5 coordenadas, con un valor de longitud de entrada de 15, valor de radio de 1.0, tasa de aprendizaje de 0.5 y un número de 200 iteraciones.

3.2. Resultados de la evaluación

Prueba 1

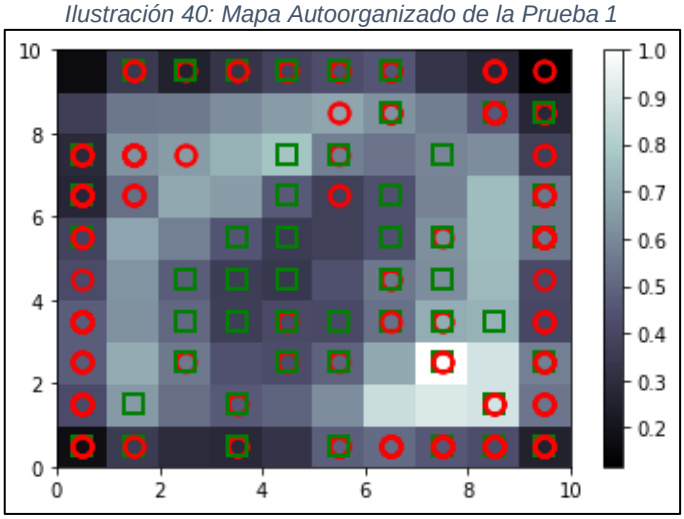
Tabla 6: Parámetros de la Prueba 1

Parámetros	Valor
Datos de entrada	690
Dimensión del Mapa Autoorganizado	10x10
Coordenada #1	(7, 2)
Coordenada #2	(8, 1)
Longitud de entrada	15
Valor de radio	1.0
Tasa de aprendizaje	0.5
Número de iteraciones	100

Fuente: *Elaboración propia*

Resultados obtenidos de la Prueba 1:

Ilustración 40: Mapa Autoorganizado de la Prueba 1



Fuente: *Elaboración propia*

Ilustración 41: Array de los posibles 11 fraudes de la Prueba 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.58087e+07	0	46	4	2	5	3	0	1	0	0	0	2	100	961
1	1.57765e+07	0	25	11	1	6	4	4.5	1	0	0	0	2	120	1
2	1.57921e+07	0	33.92	1.585	1	1	1	0	1	0	0	0	2	320	1
3	1.58132e+07	0	24.92	1.25	2	1	1	0	1	0	0	0	2	80	1
4	1.57628e+07	1	30.67	2.5	2	13	8	2.25	0	0	0	1	1	340	1
5	1.57478e+07	1	35	3.375	2	8	8	8.29	0	0	0	1	2	0	1
6	1.57163e+07	1	35.17	4.5	2	14	8	5.75	0	0	0	1	1	711	1
7	1.57655e+07	1	36.75	0.125	1	8	4	1.5	0	0	0	1	2	232	114
8	1.57546e+07	1	32.08	4	1	13	4	1.5	0	0	0	1	2	120	1
9	1.56572e+07	1	21.42	0.75	1	12	7	0.75	0	0	0	1	2	132	3
10	1.57769e+07	1	26.33	13	2	10	2	0	0	0	0	1	2	140	1111

Fuente: *Elaboración propia*

Comparación de resultados de la Prueba 1

Tabla 7: Comparación de resultados de la Prueba 1

Dataset Original		Array 'Frauds' (Posibles Fraudes)
CustomerID	**Atributo 'Class'**	**Valor Obtenido de la predicción**
15808662	1	0
15776545	0	0
15792107	0	0
15813192	0	0
15762799	0	0
15747757	0	0
15716347	1	0
15765487	1	0
15754578	0	0
15657228	0	0
15776921	0	0

Fuente: *Elaboración propia*

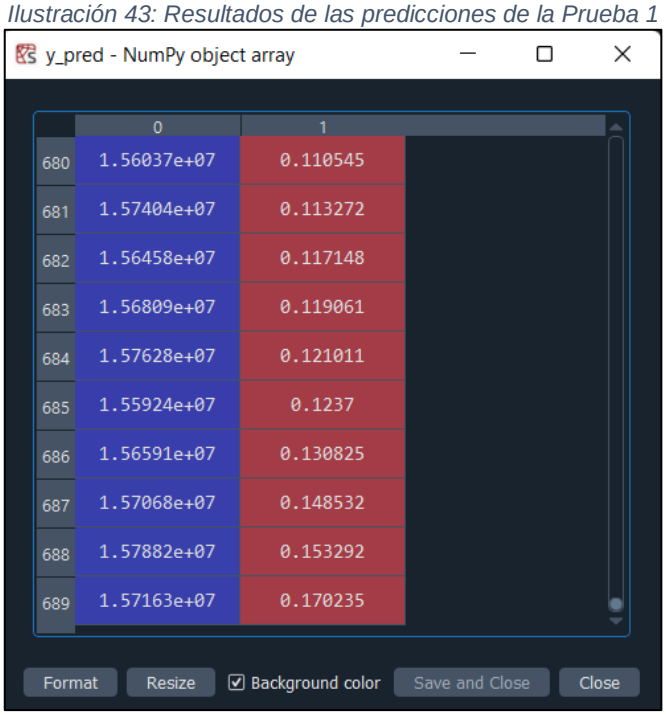
Como se puede apreciar en la tabla 7, se obtuvo una lista de 11 posibles fraudes de los cuales se obtuvo 3 errores y 8 aciertos en comparación al dataset original, dándonos un porcentaje de precisión de un 72.73 % y un porcentaje de error del 27.27 %.

Ilustración 42: Resultados del entrenamiento de la RNA de la Prueba 1

```
Epoch 1/2
550/690 [=====================>.......] - ETA: 0s -
loss: 0.5244 - accuracy: 0.9782
2022-08-24 21:55:32.728701: I tensorflow/
stream_executor/cuda/cuda_blas.cc:1786] TensorFloat-32
will be used for the matrix multiplication. This will
only be logged once.
690/690 [==============================] - 2s 2ms/step -
loss: 0.4709 - accuracy: 0.9812
Epoch 2/2
690/690 [==============================] - 1s 2ms/step -
loss: 0.1347 - accuracy: 0.9841
Out[9]: <keras.callbacks.History at 0x204dc306f10>
```

Fuente: *Elaboración Propia*

Ilustración 43: Resultados de las predicciones de la Prueba 1



Fuente: *Elaboración propia*

Como se puede observar en la ilustración 42, en la primera época en comparación a la segunda se redujo el porcentaje de pérdida de 47.09 % a 13.47% mientras que en el porcentaje de precisión se mantuvo en un 98%.

En la ilustración 43 se muestra los 10 valores más altos que se obtuvo de la predicción de la red neuronal, se puede decir que los valores más altos para un posible fraude van de un 11 – 17 %.

Prueba 2

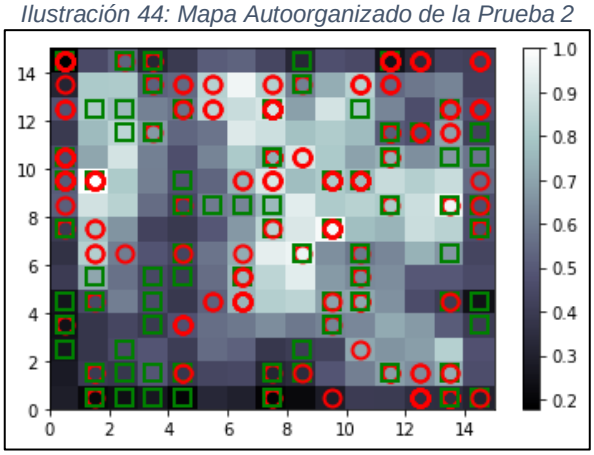
Tabla 8: Parámetros de la Prueba 2

Parámetros	Valor
Datos de entrada	690
Dimensión del Mapa Autoorganizado	15x15
Coordenada #1	(9, 7)
Coordenada #2	(1, 9)
Coordenada #3	(8, 10)
Longitud de entrada	15
Valor de radio	1.0
Tasa de aprendizaje	0.5
Número de iteraciones	100

Fuente: *Elaboración propia*

Resultados obtenidos de la Prueba 2:

Ilustración 44: Mapa Autoorganizado de la Prueba 2



Fuente: *Elaboración propia*

Ilustración 45: Array de los 38 posibles fraudes de la Prueba 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.56481e+07	0	24.5	1.75	1	8	4	0.165	0	0	0	0	2	132	1
1	1.57316e+07	0	29.5	2	1	10	8	2	0	0	0	0	2	256	18
2	1.55756e+07	0	24.75	13.665	2	11	8	1.5	0	0	0	0	2	280	2
3	1.56445e+07	0	18.17	10	1	11	8	0.165	0	0	0	0	2	340	1
4	1.56466e+07	0	15.92	2.875	2	11	4	0.085	0	0	0	0	2	120	1
5	1.56992e+07	0	22.5	8.46	1	14	4	2.46	0	0	0	0	2	164	1
6	1.5594e+07	0	21.75	1.75	1	5	3	0	0	0	0	0	2	160	1
7	1.56298e+07	0	24.33	2.5	1	3	5	4.5	0	0	0	0	2	200	457
8	1.5593e+07	0	31.75	3	1	5	3	0	0	0	0	0	2	160	21
9	1.55778e+07	0	29.5	1.085	1	14	4	1	0	0	0	0	2	280	14
10	1.55938e+07	0	16.33	0.21	2	6	4	0.125	0	0	0	0	2	200	2
11	1.55993e+07	0	33.67	0.375	2	13	4	0.375	0	0	0	0	2	300	45
12	1.56114e+07	0	23.58	0.585	1	1	1	0.125	0	0	0	0	2	120	88
13	1.57215e+07	0	24.5	2.415	1	8	4	0	0	0	0	0	2	120	1
14	1.56117e+07	0	20.08	1.25	2	8	4	0	0	0	0	0	2	0	1
15	1.56089e+07	0	17.67	0	1	5	1	0	0	0	0	0	2	86	1

Fuente: *Elaboración propia*

Comparación de resultados de la Prueba 2

Tabla 9: Comparación de los resultados de la Prueba 2

Dataset Original		Array 'frauds' (Posibles fraudes)
CustomerID	**Atributo 'class'**	**Valor obtenido de la predicción**
15648069	0	0
15629750	0	0
15721504	0	0
15731586	0	0
15575605	0	0
15644453	0	0
15646594	0	0
15699238	1	0
15593959	0	0
15592999	0	0
15577771	0	0
15593834	0	0
15599272	1	0
15611409	0	0
15660144	0	0
15611682	0	0
15608916	0	0
15610042	0	0
15568162	0	0
15662152	0	0
15734578	0	0
15641733	0	0
15603565	0	0
15802869	0	0
15664720	0	0
15645820	1	0
15800554	0	0
15696120	0	0
15567919	0	0
15687634	0	0
15638751	0	0
15665014	0	0
15606554	0	0
15719940	0	0
15761733	0	0
15736399	0	0
15766734	0	0
15667934	0	0

Fuente: *Elaboración propia*

Como se logra apreciar en la tabla 9, se obtuvo una lista de 38 posibles fraudes de los cuales se obtuvo 3 errores y 35 aciertos en comparación al dataset original, dándonos un porcentaje de precisión del 92.10 % y un porcentaje de error del 7.90 %.

Ilustración 46: Resultados del entrenamiento de la RNA de la Prueba 2

```
Epoch 1/2
334/690 [=============>.............] - ETA: 0s -
loss: 0.6144 - accuracy: 0.9431
2022-08-25 01:00:09.959737: I tensorflow/
stream_executor/cuda/cuda_blas.cc:1786] TensorFloat-32
will be used for the matrix multiplication. This will
only be logged once.
690/690 [==============================] - 2s 2ms/step -
loss: 0.5126 - accuracy: 0.9406
Epoch 2/2
690/690 [==============================] - 1s 2ms/step -
loss: 0.2204 - accuracy: 0.9449
```

Fuente: *Elaboración propia*

Ilustración 47: Resultados de las predicciones en la Prueba 2

Fuente: *Elaboración propia*

Como se aprecia en la ilustración 46, en la primera época en comparación a la segunda se redujo el porcentaje de pérdida de 51.26 % a 22.04% mientras que en el porcentaje de precisión se mantuvo en un 94%.

En la ilustración 47 se muestra los 15 valores más altos que se obtuvo de la predicción de la red neuronal, se puede decir que los valores más altos para un posible fraude van de un 20 – 34 %.

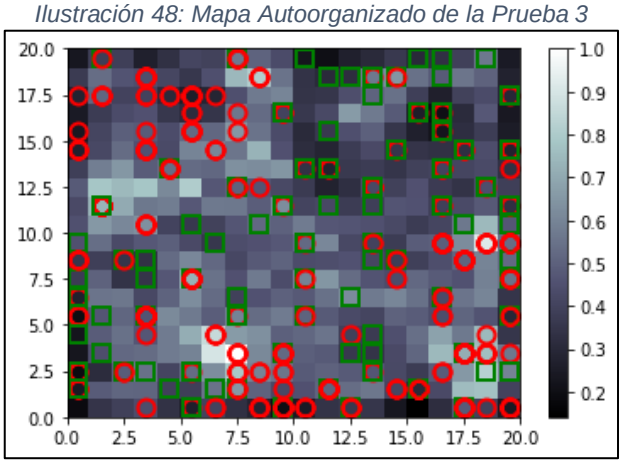
Prueba 3

Tabla 10: Parámetros de la Prueba 3

Parámetros	Valor
Datos de entrada	690
Dimensión del Mapa Autoorganizado	20x20
Coordenada #1	(5, 15)
Coordenada #2	(8, 15)
Coordenada #3	(16, 14)
Longitud de entrada	15
Valor de radio	1.0
Tasa de aprendizaje	0.5
Número de iteraciones	150

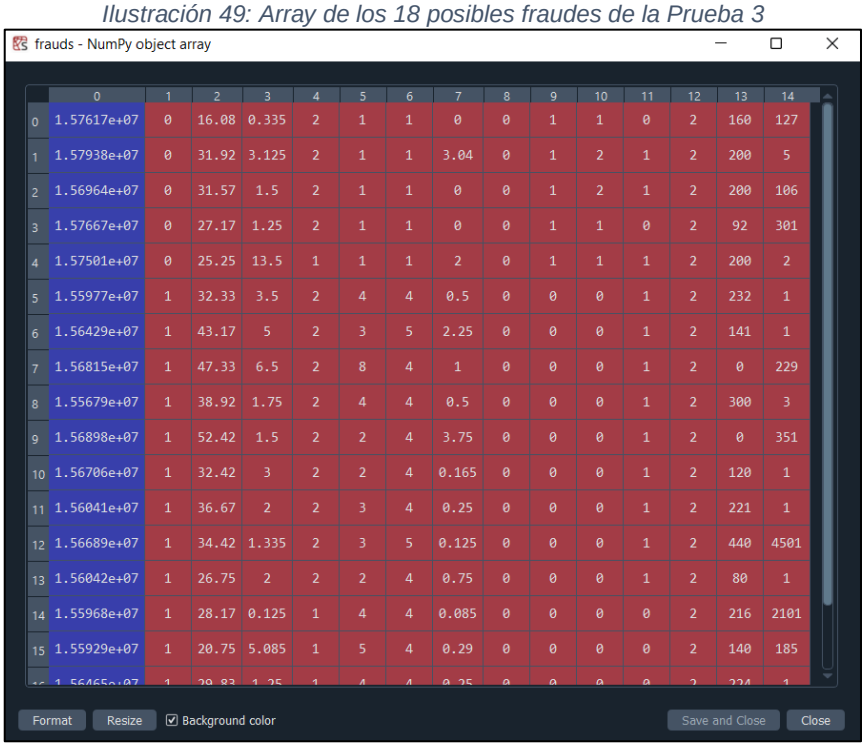
Fuente: *Elaboración propia*

Ilustración 48: Mapa Autoorganizado de la Prueba 3



Fuente: *Elaboración propia*

Ilustración 49: Array de los 18 posibles fraudes de la Prueba 3



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.57617e+07	0	16.08	0.335	2	1	1	0	0	1	1	0	2	160	127
1	1.57938e+07	0	31.92	3.125	2	1	1	3.04	0	1	2	1	2	200	5
2	1.56964e+07	0	31.57	1.5	2	1	1	0	0	1	2	1	2	200	106
3	1.57667e+07	0	27.17	1.25	2	1	1	0	0	1	1	0	2	92	301
4	1.57501e+07	0	25.25	13.5	1	1	1	2	0	1	1	1	2	200	2
5	1.55977e+07	1	32.33	3.5	2	4	4	0.5	0	0	0	1	2	232	1
6	1.56429e+07	1	43.17	5	2	3	5	2.25	0	0	0	1	2	141	1
7	1.56815e+07	1	47.33	6.5	2	8	4	1	0	0	0	1	2	0	229
8	1.55679e+07	1	38.92	1.75	2	4	4	0.5	0	0	0	1	2	300	3
9	1.56898e+07	1	52.42	1.5	2	2	4	3.75	0	0	0	1	2	0	351
10	1.56706e+07	1	32.42	3	2	2	4	0.165	0	0	0	1	2	120	1
11	1.56041e+07	1	36.67	2	2	3	4	0.25	0	0	0	1	2	221	1
12	1.56689e+07	1	34.42	1.335	2	3	5	0.125	0	0	0	1	2	440	4501
13	1.56042e+07	1	26.75	2	2	2	4	0.75	0	0	0	1	2	80	1
14	1.55968e+07	1	28.17	0.125	1	4	4	0.085	0	0	0	0	2	216	2101
15	1.55929e+07	1	20.75	5.085	1	5	4	0.29	0	0	0	0	2	140	185

Fuente: *Elaboración propia*

Comparación de resultados de la Prueba 3

Tabla 11: Comparación de resultados de la Prueba 3

Dataset Original		Array 'frauds' (Posibles fraudes)
CustomerID	**Atributo 'class'**	**Valor obtenido de la predicción**
15761733	0	0
15793825	0	0
15696361	0	0
15766734	0	0
15750104	0	0
15597709	0	0
15642934	0	0
15681509	0	0
15567860	0	0
15689786	0	0
15670646	0	0
15604130	0	0
15668889	0	0
15604196	0	0
15596797	0	0
15592914	0	0
15646521	0	0
15615670	0	0

Fuente: *Elaboración propia*

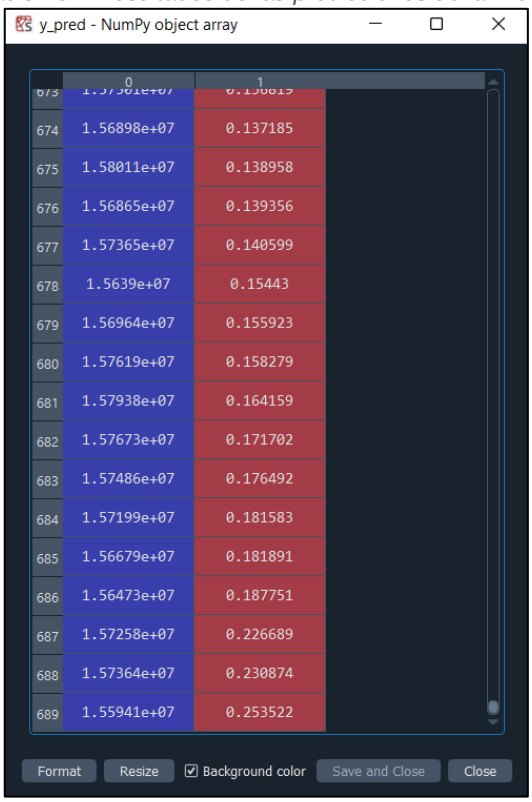
Como podemos apreciar en la tabla 11, se obtuvo una lista de 18 posibles fraudes de los cuales se obtuvo 0 errores y 18 aciertos en comparación al dataset original, dándonos un porcentaje de precisión del 100 % y un porcentaje de error del 0 %.

Ilustración 50: Resultados del entrenamiento de la RNA de la Prueba 3

```
In [23]: classifier.fit(clientes, is_fraud, batch_size = 1,
epochs = 2)
Epoch 1/2
206/690 [=======>......................] - ETA: 1s - loss:
0.6316 - accuracy: 0.9757
2022-08-25 01:28:45.816268: I tensorflow/stream_executor/cuda/
cuda_blas.cc:1786] TensorFloat-32 will be used for the matrix
multiplication. This will only be logged once.
690/690 [==============================] - 4s 4ms/step - loss:
0.4495 - accuracy: 0.9739
Epoch 2/2
690/690 [==============================] - 2s 2ms/step - loss:
0.1574 - accuracy: 0.9739
Out[23]: <keras.callbacks.History at 0x1c887e24520>
```

Fuente: *Elaboración propia*

Ilustración 51: Resultados de las predicciones de la Prueba 3



Fuente: *Elaboración propia*

Como se logra visualizar en la ilustración 50, en la primera época en comparación a la segunda se redujo el porcentaje de pérdida de 44.95 % a 15.74% mientras que en el porcentaje de precisión se mantuvo en un 97%.

En la ilustración 51 se muestra los 16 valores más altos que se obtuvo de la predicción de la red neuronal, se puede decir que los valores más altos para un posible fraude van de un 13 – 25 %.

Prueba 4

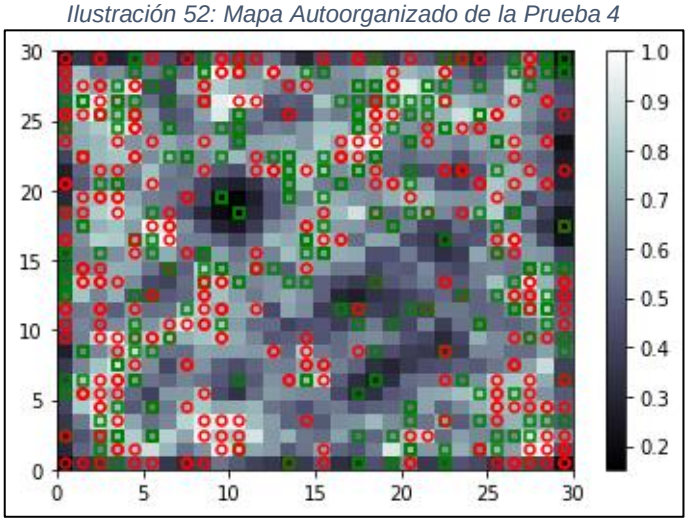
Tabla 12: Parámetros de la Prueba 4

Parámetros	Valor
Datos de entrada	690
Dimensión del Mapa Autoorganizado	30x30
Coordenada #1	(8, 3)
Coordenada #2	(18, 23)
Coordenada #3	(6, 16)
Coordenada #4	(15, 6)
Coordenada #5	(22, 26)
Longitud de entrada	15
Valor de radio	1.0
Tasa de aprendizaje	0.5
Número de iteraciones	200

Fuente: *Elaboración propia*

Resultados obtenidos de la Prueba 4:

Ilustración 52: Mapa Autoorganizado de la Prueba 4



Fuente: *Elaboración propia*

Ilustración 53: Array de los 6 posibles fraudes de la Prueba 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.56992e+07	0	22.5	8.46	1	14	4	2.46	0	0	0	0	2	164	1
1	1.55778e+07	0	29.5	1.085	1	14	4	1	0	0	0	0	2	280	14
2	1.5639e+07	0	56.83	4.25	1	1	1	5	0	0	0	1	2	0	5
3	1.58132e+07	0	24.92	1.25	2	1	1	0	1	0	0	0	2	80	1
4	1.56298e+07	0	24.33	2.5	1	3	5	4.5	0	0	0	0	2	200	457
5	1.57921e+07	0	33.92	1.585	1	1	1	0	1	0	0	0	2	320	1

Fuente: *Elaboración propia*

Comparación de resultados de la Prueba 4

Tabla 13: Comparación de resultados de la Prueba 4

Dataset Original		Array 'frauds' (Posibles fraudes)
CustomerID	**Atributo 'class'**	**Valor obtenido de la Predicción**
15792107	0	0
15629750	0	0
15813192	0	0
15699238	1	0
15577771	0	0
15638983	0	0

Fuente: *Elaboración propia*

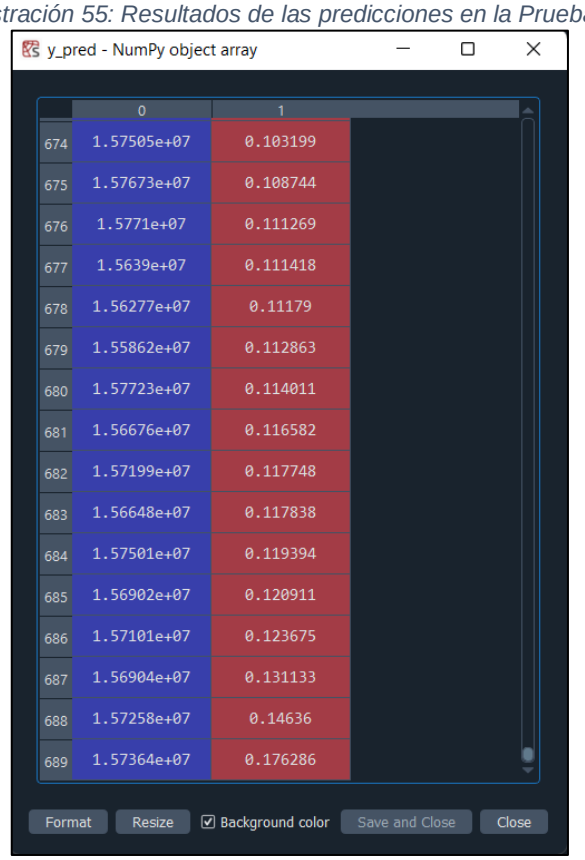
Como se puede visualizar en la tabla 13, se obtuvo una lista de 6 posibles fraudes de los cuales se obtuvo 1 error y 5 aciertos en comparación al dataset original, dándonos un porcentaje de precisión del 83.33 % y un porcentaje de error del 16.67 %.

Ilustración 54: Resultados del entrenamiento de la RNA de la Prueba 4

```
Epoch 1/2
 84/690 [==>...........................] - ETA: 1s - loss:
0.6689 - accuracy: 1.0000
2022-08-25 02:10:35.122949: I tensorflow/stream_executor/cuda/
cuda_blas.cc:1786] TensorFloat-32 will be used for the matrix
multiplication. This will only be logged once.
690/690 [==============================] - 2s 2ms/step - loss:
0.4474 - accuracy: 0.9913
Epoch 2/2
690/690 [==============================] - 1s 2ms/step - loss:
0.1133 - accuracy: 0.9913
Out[26]: <keras.callbacks.History at 0x23e337ca970>
```

Fuente: *Elaboración propia*

Ilustración 55: Resultados de las predicciones en la Prueba 4



Fuente: *Elaboración propia*

En la ilustración 54 se logra apreciar que en la primera época en comparación a la segunda se redujo el porcentaje de pérdida de 44.74 % a 11.33 % mientras que en el porcentaje de precisión se mantuvo en un 99%.

En la ilustración 55 se muestra los 16 valores más altos que se obtuvo de la predicción de la red neuronal, se puede decir que los valores más altos para un posible fraude van de un 10 – 17 %.

Finalizada las pruebas del modelo predictivo híbrido, se puede decir que en todas las pruebas el mapa autoorganizado tuvo más aciertos que errores, con un porcentaje mínimo de 72 % de efectividad (obtenido en la primera prueba) hasta un 100 % de efectividad (obtenido en la prueba 3). Y los valores de predicción de la red neuronal artificial con respecto a si un cliente haya hecho fraude llenando el formulario para la solicitud de una tarjeta de crédito no pasa más del 34 % (obtenido en la prueba 2), esto quiere decir que mientras más grande sea el mapa autoorganizado y más coordenadas se utiliza para entrenar el mapa, se obtiene mejores resultados de predicción, por ende, la red neuronal artificial va a tener un mayor porcentaje de precisión al momento del entrenamiento y contará con resultados de predicciones muy óptimas.

3.3. Conclusiones

Como resultado del modelo predictivo para la detección de fraudes se concluye que:

- Mediante la revisión bibliográfica de artículos y revistas de carácter científico se obtuvo mayor conocimiento sobre el aprendizaje profundo e inteligencia artificial, cómo las técnicas de Deep Learning que se escogió para la elaboración del modelo.
- Aplicando las técnicas de Deep Learning investigadas se logró realizar el modelo híbrido predictivo que permite predecir si un cliente hizo o no fraude al momento de llenar una solicitud para una tarjeta de crédito.
- Se concluyó que en el mundo de la inteligencia artificial se pueden mezclar algoritmos y técnicas para la realización de modelos predictivos.

3.4. Recomendaciones

En base a lo realizado en este trabajo de titulación se recomienda:

- Al momento de hacer un modelo de inteligencia artificial, ver si se cuenta con los recursos de hardware necesarios para la implementación y pruebas de este.
- Experimentar al momento de hacer las pruebas modificando los valores de los parámetros para así obtener mejores resultados.
- Se recomienda también utilizar otros datasets con más datos para probar el modelo predictivo y obtener mejores resultados.
- Y, por último, se recomienda investigar información en fuentes bibliográficas de revistas académicas con contenido redactado por profesionales.

BIBLIOGRAFÍA

[1] J. Calvo, M. A. Guzmán, y D. Ramos, “Machine Learning, una pieza clave en la transformación de los modelos de negocio”. ManagementSolutions, 2018. [En línea]. Disponible en: https://www.managementsolutions.com/sites/default/files/publicaciones/esp/machine-learning.pdf

[2] J. Alvarado Zabala, I. Martillo Alchundia, y G. Guzman Seraquive, “Revisión de literatura sobre las técnicas de Machine Learning en la detección de fraudes bancarios”, *Sapienza Int. J. Interdiscip. Stud.*, vol. 3, núm. 1, pp. 719–727, feb. 2022, doi: 10.51798/sijis.v3i1.257.

[3] C. Onwubiko, “Fraud matrix: A morphological and analysis-based classification and taxonomy of fraud”, *Comput. Secur.*, vol. 96, p. 101900, sep. 2020, doi: 10.1016/j.cose.2020.101900.

[4] G. Vettigli, “MiniSom 1.1.2”, agosto de 2018. https://test.pypi.org/project/MiniSom/

[5] J. F. Correa Wachter, C. F. Henao Villas, F. Henao Villa, y D. A. García Arango, “Análisis del aporte del aprendizaje de máquinas a la seguridad de la información”, *InGente Am.*, vol. 1, núm. 1, pp. 9–20, nov. 2021, doi: 10.21803/ingecana.1.1.407.

[6] F. Alvarez, “Machine Learning en la detección de fraudes de comercio electrónico aplicado a los servicios bancarios”, *Cienc. Tecnol.*, núm. 20, pp. 81–95, dic. 2020, doi: 10.18682/cyt.vi0.4310.

[7] R. Jain y R. Bhatnagar, “Applications of Machine Learning in Cyber Security - A Review and a Conceptual Framework for a University Setup”, en *The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2019)*, Cham, 2020, pp. 599–608. doi: 10.1007/978-3-030-14118-9_60.

[8] F. Itoo, Meenakshi, y S. Singh, “Comparison and analysis of logistic regression, Naïve Bayes and KNN machine learning algorithms for credit card fraud detection”, *Int. J. Inf. Technol.*, vol. 13, núm. 4, pp. 1503–1511, ago. 2021, doi: 10.1007/s41870-020-00430-y.

[9] J. Ranjani, A. Sheela, y K. P. Meena, “Combination of NumPy, SciPy and Matplotlib/Pylab -a good alternative methodology to MATLAB - A Comparative analysis”, en *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, abr. 2019, pp. 1–5. doi: 10.1109/ICIICT1.2019.8741475.

[10] D. A. Carrillo-Torres, A. Ocampo-Escobar, y R. D. Estrada-Esponda, “Databio, galería de biodiversidad del Jardín Botánico Juan María Céspedes”, *Rev. UIS Ing.*, vol. 20, núm. 3, pp. 155–166, jun. 2021, doi: 10.18273/revuin.v20n3-2021011.

[11] K. Oviedo Rodríguez y B. Jiménez Oviedo, “Ejemplos básicos de álgebra lineal con python: Basic examples of linear algebra with python”, *Rev. Digit. Matemática Educ. E Internet*, vol. 21, núm. 1, ago. 2020, doi: 10.18845/rdmei.v21i1.5340.

[12] J. Hao y T. K. Ho, “Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language”, *J. Educ. Behav. Stat.*, vol. 44, núm. 3, pp. 348–361, feb. 2019, doi: 10.3102/1076998619832248.

[13] P. Parth Singh, M. Dhananjay, y D. Pankaj, “OBTAINING EXACT SOLUTIONS OF VISCOINCOMPRESSIBLE PARALLEL FLOWS USING PYTHON”, *Int. J. Eng. Appl. Sci. Techonology*, vol. 6, núm. 11, pp. 213–217, mar. 2022, doi: 10.33564/IJEAST.2022.v06i11.040.

[14] M. Si, T. J. Tarnoczi, B. M. Wiens, y K. Du, “Development of Predictive Emissions Monitoring System Using Open Source Machine Learning Library – Keras: A Case Study on a Cogeneration Unit”, *IEEE Access*, vol. 7, pp. 113463–113475, 2019, doi: 10.1109/ACCESS.2019.2930555.

[15] M. Kuroki, “Using Python and Google Colab to teach undergraduate microeconomic theory”, *Int. Rev. Econ. Educ.*, vol. 38, p. 100225, 2021, doi: https://doi.org/10.1016/j.iree.2021.100225.
[16] J. R. Cheng y M. Gen, “Accelerating genetic algorithms with GPU computing: A selective overview”, *Comput. Ind. Eng.*, vol. 128, pp. 514–525, feb. 2019, doi: 10.1016/j.cie.2018.12.067.
[17] J. Choquette y W. Gandhi, “NVIDIA A100 GPU: Performance & Innovation for GPU Computing”, en *2020 IEEE Hot Chips 32 Symposium (HCS)*, Los Alamitos, CA, USA, ago. 2020, pp. 1–43. doi: 10.1109/HCS49909.2020.9220622.
[18] R. Han, J. Chen, B. Garg, J. Young, J. Sim, y H. Kim, “CuPBoP: CUDA for Parallelized and Broad-range Processors”, *arXiv*, 2022, doi: 10.48550/arXiv.2206.07896.
[19] D. Datta, D. Mittal, N. P. Mathew, y J. Sairabanu, “Comparison of Performance of Parallel Computation of CPU Cores on CNN model”, en *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, feb. 2020, pp. 1–8. doi: 10.1109/ic-ETITE47903.2020.142.
[20] A. Mathew, P. Amudha, y S. Sivakumari, “Deep Learning Techniques: An Overview”, en *Advanced Machine Learning Technologies and Applications*, Singapore, 2021, pp. 599–608. doi: 10.1007/978-981-15-3383-9_54.
[21] C. Aggarwal, *Neural Networks and Deep Learning*, 1a ed. Springer Cham, 2018. [En línea]. Disponible en: https://link.springer.com/book/10.1007/978-3-319-94463-0#about-this-book
[22] D. Ruiz Hidalgo, B. Bacca Cortés, y E. Caicedo Bravo, “Dimensionality reduction of hyperspectral images of vegetation and crops based on self-organized maps”, *Inf. Process. Agric.*, vol. 8, núm. 2, pp. 310–327, jun. 2021, doi: 10.1016/j.inpa.2020.07.002.
[23] M. Carrasco Kind y R. J. Brunner, “SOMz: photometric redshift PDFs with self-organizing maps and random atlas”, *Mon. Not. R. Astron. Soc.*, vol. 438, núm. 4, pp. 3409–3421, mar. 2014, doi: 10.1093/mnras/stt2456.
[24] Y. Gambo y M. Shakir, “An Artificial Neural Network (ANN)-Based Learning Agent for Classifying Learning Styles in Self-Regulated Smart Learning Environment”, *Int. J. Emerg. Technol. Learn. IJET*, vol. 16, núm. 18, pp. 185–199, sep. 2021, doi: 10.3991/ijet.v16i18.24251.
[25] I. Atria, “Qué son las redes neuronales y sus funciones”, *ATRIA INNOVATION*, octubre de 2019. https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/
[26] J. C. R. Whittington y R. Bogacz, “Theories of Error Back-Propagation in the Brain”, *Trends Cogn. Sci.*, vol. 23, núm. 3, pp. 235–250, mar. 2019, doi: 10.1016/j.tics.2018.12.005.
[27] C. Schröer, F. Kruse, y J. M. Gómez, “A Systematic Literature Review on Applying CRISP-DM Process Model”, *CENTERIS 2020 - Int. Conf. Enterp. Inf. Syst. ProjMAN 2020 - Int. Conf. Proj. Manag. HCist 2020 - Int. Conf. Health Soc. Care Inf. Syst. Technol. 2020 CENTERISProjMANHCist 2020*, vol. 181, pp. 526–534, ene. 2021, doi: 10.1016/j.procs.2021.01.199.
[28] Z. Hussain y M. S. Khan, “Introducing python programming for engineering scholars”, *Int. J. Comput. Sci. Netw. Secur.*, vol. 18, núm. 12, pp. 26–33, dic. 2018.
[29] D. Dua y C. Graff, “UCI Machine Learning Repository”. University of California, Irvine, School of Information and Computer Sciences, 2017. [En línea]. Disponible en: http://archive.ics.uci.edu/ml
[30] G. Vettigli, “Giuseppe Vettigli - Senior Data Scientist”. 2022. [En línea]. Disponible en: https://www.linkedin.com/in/giuseppevettigli/

[31] C. Gierlich y S. Palkovits, “Featurizing chemistry for machine learning — methods and a coded example”, *Curr. Opin. Chem. Eng.*, vol. 37, p. 100840, sep. 2022, doi: 10.1016/j.coche.2022.100840.
[32] X. Yan, Z. Cao, A. Murphy, y Y. Qiao, “An ensemble machine learning method for microplastics identification with FTIR spectrum”, *J. Environ. Chem. Eng.*, vol. 10, núm. 4, p. 108130, ago. 2022, doi: 10.1016/j.jece.2022.108130.
[33] D. Parbat y M. Chakraborty, “A python based support vector regression model for prediction of COVID19 cases in India”, *Chaos Solitons Fractals*, vol. 138, p. 109942, sep. 2020, doi: 10.1016/j.chaos.2020.109942.
[34] R. Conlin, K. Erickson, J. Abbate, y E. Kolemen, “Keras2c: A library for converting Keras neural networks to real-time compatible C”, *Eng. Appl. Artif. Intell.*, vol. 100, p. 104182, abr. 2021, doi: 10.1016/j.engappai.2021.104182.
[35] N. Pancino, P. Bongini, F. Scarselli, y M. Bianchini, “GNNkeras: A Keras-based library for Graph Neural Networks and homogeneous and heterogeneous graph processing”, *SoftwareX*, vol. 18, p. 101061, jun. 2022, doi: 10.1016/j.softx.2022.101061.
[36] L. Parisi, R. Ma, N. RaviChandran, y M. Lanzillotta, “hyper-sinh: An accurate and reliable function from shallow to deep learning in TensorFlow and Keras”, *Mach. Learn. Appl.*, vol. 6, p. 100112, dic. 2021, doi: 10.1016/j.mlwa.2021.100112.
[37] V.-H. Nhu *et al.*, “Effectiveness assessment of Keras based deep learning with different robust optimization algorithms for shallow landslide susceptibility mapping at tropical area”, *CATENA*, vol. 188, p. 104458, may 2020, doi: 10.1016/j.catena.2020.104458.
[38] E. Haghighat y R. Juanes, “SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks”, *Comput. Methods Appl. Mech. Eng.*, vol. 373, p. 113552, ene. 2021, doi: 10.1016/j.cma.2020.113552.
[39] S. Osah, A. A. Acheampong, C. Fosu, y I. Dadzie, “Deep learning model for predicting daily IGS zenith tropospheric delays in West Africa using TensorFlow and Keras”, *Adv. Space Res.*, vol. 68, núm. 3, pp. 1243–1262, ago. 2021, doi: 10.1016/j.asr.2021.04.039.

ANEXOS

ANEXO I. Código fuente de la clase modelo_predictivo.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 22 17:03:58 2022

@author: Juan José Romero
"""

# Parte 1 - Identificar los fraudes potenciales con un SOM

# Importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el dataset
dataset = pd.read_csv('Credit_Card_Applications.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Escalado de características
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

# Entrenar el SOM
from minisom import MiniSom
som = MiniSom(x = 30, y = 30, input_len = 15, sigma = 1.0, learning_rate =
0.5)
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)

# Visualizar los resultados
from pylab import bone, pcolor, colorbar, plot, show
bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's']
colors = ['r', 'g']
for i, x in enumerate(X):
    w = som.winner(x)
    plot(w[0]+0.5, w[1]+0.5,
         markers[y[i]], markeredgecolor = colors[y[i]], markerfacecolor =
'None',
         markersize = 5, markeredgewidth = 1.5)
show()

# Encontrar los fraudes
mappings = som.win_map(X)
frauds = np.concatenate( (mappings[(22,26)],
mappings[(15,6)],mappings[(8,3)], mappings[(18,23)], mappings[(6,16)]), axis
= 0 )
frauds = sc.inverse_transform(frauds)


# Parte 2 - Trasladar el modelo no supervisado a supervisado
```

```
# Crear la matriz de características
clientes = dataset.iloc[:, 1:-1].values

# Crear la variable dependiente
is_fraud = np.zeros(len(dataset))
for i in range(len(dataset)):
    if dataset.iloc[i, 0] in frauds:
        is_fraud[i] = 1

# Escalado de variables
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
clientes = sc.fit_transform(clientes)


# Parte 3 - Construir la RNA

# Importar Keras y librerías adicionales
from keras.models import Sequential
from keras.layers import Dense

# Inicializar la RNA
classifier = Sequential()

# Añadir las capas de entrada y una capa oculta
classifier.add(Dense(units = 2, kernel_initializer = "uniform",
                     activation = "relu", input_dim = 14))

# Añadir la capa de salida
classifier.add(Dense(units = 1, kernel_initializer = "uniform",
                     activation = "sigmoid"))

# Compilar la RNA
classifier.compile(optimizer = "adam", loss = "binary_crossentropy",
                   metrics = ["accuracy"])

# Ajustamos la RNA al conjunto de Entrenamiento
classifier.fit(clientes, is_fraud, batch_size = 1, epochs = 2)


# Parte 4 - Evaluar el modelo y calcular predicciones finales

# Predicción de los resultados
y_pred = classifier.predict(clientes)
y_pred = np.concatenate((dataset.iloc[:,0:1].values, y_pred), axis = 1)
y_pred = y_pred[y_pred[:,1].argsort()]
```

ANEXO II. Código fuente de la clase minisom.py

```
from math import sqrt

from numpy import (array, unravel_index, nditer, linalg, random, subtract,
                   power, exp, pi, zeros, arange, outer, meshgrid, dot,
                   logical_and)
from collections import defaultdict
from warnings import warn

# for unit tests
from numpy.testing import assert_almost_equal, assert_array_almost_equal
from numpy.testing import assert_array_equal
import unittest

"""
    Minimalistic implementation of the Self Organizing Maps (SOM).
"""


def fast_norm(x):
    """Returns norm-2 of a 1-D numpy array.

    * faster than linalg.norm in case of 1-D arrays (numpy 1.9.2rc1).
    """
    return sqrt(dot(x, x.T))


class MiniSom(object):
    def __init__(self, x, y, input_len, sigma=1.0, learning_rate=0.5,
                 decay_function=None, neighborhood_function='gaussian',
                 random_seed=None):
        """Initializes a Self Organizing Maps.

        Parameters
        ----------
        decision_tree : decision tree
        The decision tree to be exported.

        x : int
            x dimension of the SOM

        y : int
            y dimension of the SOM

        input_len : int
            Number of the elements of the vectors in input.

        sigma : float, optional (default=1.0)
            Spread of the neighborhood function, needs to be adequate
            to the dimensions of the map.
            (at the iteration t we have sigma(t) = sigma / (1 + t/T)
            where T is #num_iteration/2)
            learning_rate, initial learning rate
            (at the iteration t we have
            learning_rate(t) = learning_rate / (1 + t/T)
            where T is #num_iteration/2)

        decay_function : function (default=None)
```

```
            Function that reduces learning_rate and sigma at each iteration
            default function:
            lambda x, current_iteration, max_iter :
                        x/(1+current_iteration/max_iter)

        neighborhood_function : function, optional (default='gaussian')
            Function that weights the neighborhood of a position in the map
            possible values: 'gaussian', 'mexican_hat', 'bubble'

        random_seed : int, optiona (default=None)
            Random seed to use.
        """
        if sigma >= x/2.0 or sigma >= y/2.0:
            warn('Warning: sigma is too high for the dimension of the map.')
        if random_seed:
            self._random_generator = random.RandomState(random_seed)
        else:
            self._random_generator = random.RandomState(random_seed)
        if decay_function:
            self._decay_function = decay_function
        else:
            self._decay_function = lambda x, t, max_iter: x/(1+t/max_iter)
        self._learning_rate = learning_rate
        self._sigma = sigma
        # random initialization
        self._weights = self._random_generator.rand(x, y, input_len)*2-1
        for i in range(x):
            for j in range(y):
                # normalization
                norm = fast_norm(self._weights[i, j])
                self._weights[i, j] = self._weights[i, j] / norm
        self._activation_map = zeros((x, y))
        self._neigx = arange(x)
        self._neigy = arange(y)  # used to evaluate the neighborhood function
        neig_functions = {'gaussian': self._gaussian,
                          'mexican_hat': self._mexican_hat,
                          'bubble': self._bubble}
        if neighborhood_function not in neig_functions:
            msg = '%s not supported. Functions available: %s'
            raise ValueError(msg % (neighborhood_function,
                                    ', '.join(neig_functions.keys())))
        if neighborhood_function == 'bubble' and sigma % 2 != 1:
            warn('sigma should be an odd value when bubble \
                 is used as neighborhood function')
        self.neighborhood = neig_functions[neighborhood_function]

    def get_weights(self):
        """Returns the weights of the neural network"""
        return self._weights

    def _activate(self, x):
        """Updates matrix activation_map, in this matrix
           the element i,j is the response of the neuron i,j to x"""
        s = subtract(x, self._weights)  # x - w
        it = nditer(self._activation_map, flags=['multi_index'])
        while not it.finished:
            # || x - w ||
            self._activation_map[it.multi_index] =
fast_norm(s[it.multi_index])
```

```
            it.iternext()

    def activate(self, x):
        """Returns the activation map to x"""
        self._activate(x)
        return self._activation_map

    def _gaussian(self, c, sigma):
        """Returns a Gaussian centered in c"""
        d = 2*pi*sigma*sigma
        ax = exp(-power(self._neigx-c[0], 2)/d)
        ay = exp(-power(self._neigy-c[1], 2)/d)
        return outer(ax, ay)  # the external product gives a matrix

    def _mexican_hat(self, c, sigma):
        """Mexican hat centered in c"""
        xx, yy = meshgrid(self._neigx, self._neigy)
        p = power(xx-c[0], 2) + power(yy-c[1], 2)
        d = 2*pi*sigma*sigma
        return exp(-p/d)*(1-2/d*p)

    def _bubble(self, c, sigma):
        """Constant function centered in c with spread sigma.
           sigma should be an odd value,
        """
        ax = logical_and(self._neigx > c[0]-sigma/2.,
                         self._neigx < c[0]+sigma/2.)
        ay = logical_and(self._neigy > c[1]-sigma/2.,
                         self._neigy < c[1]+sigma/2.)
        return outer(ax, ay)*1.

    def winner(self, x):
        """Computes the coordinates of the winning neuron for the sample x"""
        self._activate(x)
        return unravel_index(self._activation_map.argmin(),
                             self._activation_map.shape)

    def update(self, x, win, t):
        """Updates the weights of the neurons.

        Parameters
        ----------
        x : np.array
            Current pattern to learn
        win : tuple
            Position of the winning neuron for x (array or tuple).
        t : int
            Iteration index
        """
        eta = self._decay_function(self._learning_rate, t, self.T)
        # sigma and learning rate decrease with the same rule
        sig = self._decay_function(self._sigma, t, self.T)
        # improves the performances
        g = self.neighborhood(win, sig)*eta
        it = nditer(g, flags=['multi_index'])
        while not it.finished:
            # eta * neighborhood_function * (x-w)
            x_w = (x - self._weights[it.multi_index])
            self._weights[it.multi_index] += g[it.multi_index] * x_w
```

```
            # normalization
            norm = fast_norm(self._weights[it.multi_index])
            self._weights[it.multi_index] =
self._weights[it.multi_index]/norm
            it.iternext()

    def quantization(self, data):
        """Assigns a code book (weights vector of the winning neuron)
        to each sample in data."""
        q = zeros(data.shape)
        for i, x in enumerate(data):
            q[i] = self._weights[self.winner(x)]
        return q

    def random_weights_init(self, data):
        """Initializes the weights of the SOM
        picking random samples from data"""
        it = nditer(self._activation_map, flags=['multi_index'])
        while not it.finished:
            rand_i = self._random_generator.randint(len(data))
            self._weights[it.multi_index] = data[rand_i]
            norm = fast_norm(self._weights[it.multi_index])
            self._weights[it.multi_index] =
self._weights[it.multi_index]/norm
            it.iternext()

    def train_random(self, data, num_iteration):
        """Trains the SOM picking samples at random from data"""
        self._init_T(num_iteration)
        for iteration in range(num_iteration):
            # pick a random sample
            rand_i = self._random_generator.randint(len(data))
            self.update(data[rand_i], self.winner(data[rand_i]), iteration)

    def train_batch(self, data, num_iteration):
        """Trains using all the vectors in data sequentially"""
        self._init_T(len(data)*num_iteration)
        iteration = 0
        while iteration < num_iteration:
            idx = iteration % (len(data)-1)
            self.update(data[idx], self.winner(data[idx]), iteration)
            iteration += 1

    def _init_T(self, num_iteration):
        """Initializes the parameter T needed to adjust the learning rate"""
        # keeps the learning rate nearly constant
        # for the last half of the iterations
        self.T = num_iteration/2

    def distance_map(self):
        """Returns the distance map of the weights.
        Each cell is the normalised sum of the distances between
        a neuron and its neighbours."""
        um = zeros((self._weights.shape[0], self._weights.shape[1]))
        it = nditer(um, flags=['multi_index'])
        while not it.finished:
            for ii in range(it.multi_index[0]-1, it.multi_index[0]+2):
                for jj in range(it.multi_index[1]-1, it.multi_index[1]+2):
                    if (ii >= 0 and ii < self._weights.shape[0] and
```

```
                            jj >= 0 and jj < self._weights.shape[1]):
                        w_1 = self._weights[ii, jj, :]
                        w_2 = self._weights[it.multi_index]
                        um[it.multi_index] += fast_norm(w_1-w_2)
            it.iternext()
        um = um/um.max()
        return um

    def activation_response(self, data):
        """
            Returns a matrix where the element i,j is the number of times
            that the neuron i,j have been winner.
        """
        a = zeros((self._weights.shape[0], self._weights.shape[1]))
        for x in data:
            a[self.winner(x)] += 1
        return a

    def quantization_error(self, data):
        """Returns the quantization error computed as the average
        distance between each input sample and its best matching unit."""
        error = 0
        for x in data:
            error += fast_norm(x-self._weights[self.winner(x)])
        return error/len(data)

    def win_map(self, data):
        """Returns a dictionary wm where wm[(i,j)] is a list
        with all the patterns that have been mapped in the position i,j."""
        winmap = defaultdict(list)
        for x in data:
            winmap[self.winner(x)].append(x)
        return winmap


class TestMinisom(unittest.TestCase):
    def setup_method(self, method):
        self.som = MiniSom(5, 5, 1)
        for i in range(5):
            for j in range(5):
                # checking weights normalization
                assert_almost_equal(1.0, linalg.norm(self.som._weights[i,
j]))
        self.som._weights = zeros((5, 5))  # fake weights
        self.som._weights[2, 3] = 5.0
        self.som._weights[1, 1] = 2.0

    def test_decay_function(self):
        assert self.som._decay_function(1., 2., 3.) == 1./(1.+2./3.)

    def test_fast_norm(self):
        assert fast_norm(array([1, 3])) == sqrt(1+9)

    def test_unavailable_neigh_function(self):
        with self.assertRaises(ValueError):
            MiniSom(5, 5, 1, neighborhood_function='boooom')

    def test_gaussian(self):
        bell = self.som._gaussian((2, 2), 1)
```

```
        assert bell.max() == 1.0
        assert bell.argmax() == 12  # unravel(12) = (2,2)

    def test_mexican_hat(self):
        bell = self.som._mexican_hat((2, 2), 1)
        assert bell.max() == 1.0
        assert bell.argmax() == 12  # unravel(12) = (2,2)

    def test_bubble(self):
        bubble = self.som._bubble((2, 2), 1)
        assert bubble[2,2] == 1
        assert sum(sum(bubble)) == 1

    def test_win_map(self):
        winners = self.som.win_map([5.0, 2.0])
        assert winners[(2, 3)][0] == 5.0
        assert winners[(1, 1)][0] == 2.0

    def test_activation_reponse(self):
        response = self.som.activation_response([5.0, 2.0])
        assert response[2, 3] == 1
        assert response[1, 1] == 1

    def test_activate(self):
        assert self.som.activate(5.0).argmin() == 13.0  # unravel(13) = (2,3)

    def test_quantization_error(self):
        self.som.quantization_error([5, 2]) == 0.0
        self.som.quantization_error([4, 1]) == 0.5

    def test_quantization(self):
        q = self.som.quantization(array([4, 2]))
        assert q[0] == 5.0
        assert q[1] == 2.0

    def test_random_seed(self):
        som1 = MiniSom(5, 5, 2, sigma=1.0, learning_rate=0.5, random_seed=1)
        som2 = MiniSom(5, 5, 2, sigma=1.0, learning_rate=0.5, random_seed=1)
        # same initialization
        assert_array_almost_equal(som1._weights, som2._weights)
        data = random.rand(100, 2)
        som1 = MiniSom(5, 5, 2, sigma=1.0, learning_rate=0.5, random_seed=1)
        som1.train_random(data, 10)
        som2 = MiniSom(5, 5, 2, sigma=1.0, learning_rate=0.5, random_seed=1)
        som2.train_random(data, 10)
        # same state after training
        assert_array_almost_equal(som1._weights, som2._weights)

    def test_train_batch(self):
        som = MiniSom(5, 5, 2, sigma=1.0, learning_rate=0.5, random_seed=1)
        data = array([[4, 2], [3, 1]])
        q1 = som.quantization_error(data)
        som.train_batch(data, 10)
        assert q1 > som.quantization_error(data)

    def test_train_random(self):
        som = MiniSom(5, 5, 2, sigma=1.0, learning_rate=0.5, random_seed=1)
        data = array([[4, 2], [3, 1]])
        q1 = som.quantization_error(data)
```

```
        som.train_random(data, 10)
        assert q1 > som.quantization_error(data)

    def test_random_weights_init(self):
        som = MiniSom(2, 2, 2, random_seed=1)
        som.random_weights_init(array([[1.0, .0]]))
        for w in som._weights:
            assert_array_equal(w[0], array([1.0, .0]))
```