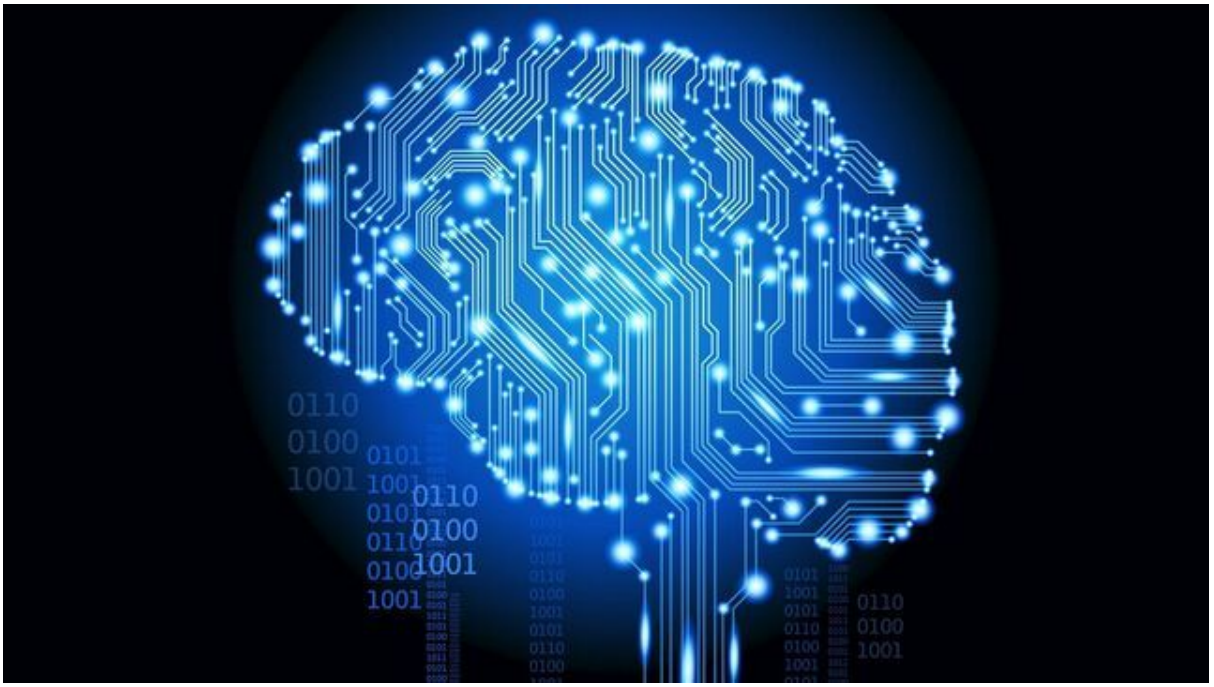

“Detección de deserción de clientes bancarios”

Red Neuronal Artificial

Inteligencia Artificial

Banchero Juan	De Felice Agustin	Hansen Ignacio
Haspert Fernando	Iasi Luciano	Grasso Martin

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Introducción

En este último año, cierta entidad bancaria ha sufrido la baja de un gran número de sus clientes por diversos motivos (mejor publicidad de otros bancos, promociones de traspaso, tasas, entre otros) y desea lanzar una campaña a todos aquellos clientes que están considerando dejar de serlo. Para lograrlo deben poder identificar/predecir quiénes son estos posibles clientes que tienen mayor probabilidad de cerrar sus cuentas.

La entidad mencionada nos brindó información acerca de sus 10.000 clientes activos pertenecientes al sistema. Dentro de la información recibida encontramos los siguientes datos:

- Propios del cliente (apellido, edad, nacionalidad, entre otros)
- Información Económica y financiera (Cuentas, Balances, Tarjetas, Resúmenes).

Para resolver la problemática de la entidad buscamos desarrollar e implementar una Inteligencia Artificial que permita predecir, utilizando la información suministrada, todos los clientes que podrían cerrar sus cuentas en la brevedad.

Resumen

El trabajo práctico consistió en el desarrollo e implementación de una **Red Neuronal Artificial (RNA) que permita la estimación de la cantidad de clientes que son plausibles de dejar de formar parte del Banco en el corto plazo.**

Este proyecto fue realizado con el lenguaje de programación **Python** en conjunto con las librerías **TensorFlow y Keras,** y utilizando como fuente de información la **Base de Clientes Activos** que la entidad bancaria nos ha provisto. La RNA construida se proveerá de ciertos atributos de los clientes activos de la entidad, realizará la evaluación de los mismos e informará si el mismo es candidato a dejar de utilizar los servicios brindados.

Elementos de Trabajo y Metodología

Modelo RNA

Para la resolución del problema se decidió utilizar una RNA del tipo Backpropagation.
Sabemos que entre otras cosas, este tipo de RNA permite identificar patrones, resolver problemas de predicción de resultados y clasificar datos.
En este caso se la utilizó como herramienta de predicción.

¿Por qué no elegimos una red Perceptrón?
Principalmente por los patrones que utilizamos, ya que estos no pueden ser aprendidos por Perceptrón debido a que no son linealmente separables.

Además de que este tipo de RNA (Perceptrón) está destinada principalmente a la clasificación.

Otra cuestión a destacar es que nuestra red no utiliza valores exclusivamente binarios, sino que posee valores reales, como por ejemplo el balance de una cuenta bancaria.
Esto no es compatible con un modelo Perceptrón.

Es importante destacar la virtud con la que cuenta la red Backpropagation de poder propagar los errores hacia atrás. Esto trae aparejado la modificación de los pesos entre las relaciones de cada par de neuronas que conforman la/las capa/s intermedia/s.

De esta manera la red puede ir disminuyendo su error y acercarse cada vez más a un mejor nivel de confiabilidad.

Si bien más adelante se explica con mayor detalle; destacamos que la RNA implementada es heteroasociativa. Esto implica que la red se ajusta por comparación en base a los valores de respuesta obtenidos contra los deseados.

Arquitectura y Topología Final

La RNA confeccionada se compone de 3 capas: capa de entrada, una capa oculta y la capa de salida. Para la capa de entrada se utilizan 6 neuronas, una por cada atributo en particular de cada cliente.

Estos atributos son:

- Edad: numérico entero
- Sexo: binario
- Balance: numérico real
- Salario estimado: numérico real
- Cliente posee tarjeta de crédito: binario
- Cliente es miembro activo: binario

En lo que respecta a los niveles ocultos de la red, fuimos probando diversas combinaciones y en base a los tiempos de entrenamiento y los resultados obtenidos, que se encuentran al final del documento, nos decidimos por utilizar un nivel (capa oculta) que cuenta con 8 neuronas.

La conformación de la salida de la RNA involucra a una única neurona de salida, la cual informa si el cliente evaluado seguirá formando parte de la cartera del banco o no.

Desde el punto de vista topológico, que implica el grado de conectividad de nuestra red, afirmamos que la misma tiene un grado de conectividad total, esto se debe a que todas las neuronas de entrada se encuentran conectadas con las neuronas de la capa oculta, y a su vez todas ellas con la neurona de salida.

En este tipo de red la conexión es del tipo hacia adelante, aunque la propagación del error se realiza en sentido contrario (desde la salida hacia los niveles intermedios previos). De esta forma cada neurona recibe el valor porcentual de su implicancia con el error cometido. Según ese valor de error recibido se realiza el reajuste de los pesos de la conexiones entre las neuronas.Finalmente, a través de estos ajustes, se minimiza el error de nuestra red.

Entrenamiento

Al utilizar un red neuronal Backpropagation el tipo de entrenamiento es OFF-LINE y supervisado. Esto quiere decir que el modelo solo aprende en una primer instancia previa a la puesta en producción.

Una vez que se pone en producción dicha RNA deja de entrenarse.

Desde el punto de vista del aprendizaje supervisado, a la red se le proporcionaron patrones con respuestas deseadas para que pueda realizar el ajuste del peso de sus conexiones y así aproximarse a los valores deseados.

Respecto a los patrones de entrenamiento, utilizamos un dataset que contaba con 10000 datos, de los cuales aplicamos 7500 para el entrenamiento, y el resto para la validación de nuestro modelo.

Cada registro del dataset se dividió en dos conjuntos: X e Y.
Donde X se compuso de todos los datos de entrada (X1, X2, X3, X4, X5, X6) e (Y1) de su respectiva salida.

Herramientas

Para la implementación se utilizaron:

- Lenguaje Python
- Librerías de Keras/Tensorflow
- Servicio Google Colab.
- Anaconda / IDE Spyder

Modelo de RNA

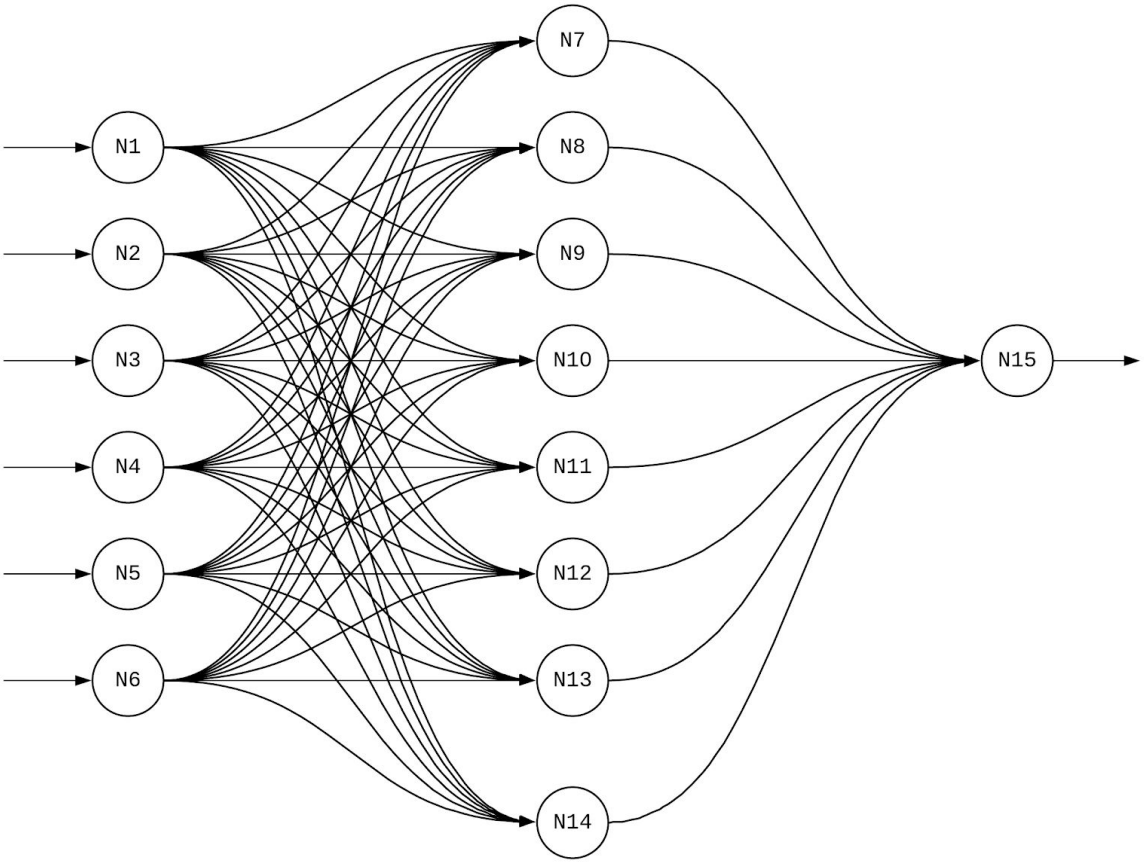

Figura 1. Topología RNA Implementada

Resultados

A partir de distintas configuraciones realizadas de la red neuronal a la hora de confección y entrenamiento, como por ejemplo: la cantidad de neuronas en la capa oculta, el tamaño de lote y cantidad de repeticiones armamos seis casos de estudio.

- **Caso de estudio 1:**

Las variables de prueba fueron las siguientes:1 capa intermedia con 4 neuronas, batch size de 30 y 100 epoch:

Exactitud = 0.7964
Precisión = 0.7964
Recuperación = 1.0

A esta composición la determinamos como nuestro peor resultado debido a que la exactitud y precisión fue del 80% aproximadamente

- **Caso de estudio 2:**

Las variables de prueba fueron las siguientes:1 capa intermedia con 4 neuronas, batch size de 15 y 300 epoch:

Exactitud = 0.8328
Precision = 0.8362548097477555
Recuperacion = 0.982420894023104

- **Caso de estudio 3:**

Las variables de prueba fueron las siguientes:1 capa intermedia con 8 neuronas, batch size de 30 y 100 epoch:

Exactitud = 0.826
Precision = 0.8279932546374368
Recuperacion = 0.9864389753892516

- **Caso de estudio 4:**

Las variables de prueba fueron las siguientes:1 capa intermedia con 8 neuronas, batch size de 15 y 300 epoch:

Primera ejecución

Exactitud = 0.8404
Precision = 0.8466898954703833
Recuperacion = 0.9763937719738824

Segunda ejecución

Exactitud = 0.8412
Precision = 0.8489492119089317
Recuperacion = 0.9738824711200402
F1-Score = 0.90713450292

Para este caso en particular, decidimos hacer dos corridas/entrenamientos ya que en comparación con todos los demás casos analizados vimos que para la primera corrida obtuvimos mejores resultados.
En la segunda ejecución actualizamos el Dropout[1] a 0.1 que significó una mejora del 0.01% aproximadamente.
A la segunda corrida de este caso la adoptamos como nuestro mejor resultado. Dicha ejecución puede visualizarse en el Colab

- **Caso de estudio 5:**

Las variables de prueba fueron las siguientes:1 capa intermedia con 8 neuronas, batch size de 6 y 200 epoch

Exactitud = 0.838
Precision = 0.8409286328460877
Recuperacion = 0.982420894023104

- **Caso de estudio 6:**

Las variables de prueba fueron las siguientes:1 capa intermedia con 8 neuronas, batch size de 15 y 800 epoch

Exactitud = 0.8404
Precision = 0.8448873483535528
Recuperacion = 0.9794073329984933

Los casos 5 y 6 son casos “extremos”, debido a que se realizaron modificaciones muy abruptas para verificar si los resultados cambiaban significativamente.

[1] El dropout (tasa de abandono) es un coeficiente utilizado para prevenir el sobreajuste.Consiste en ignorar neuronas al azar durante la etapa de entrenamiento

Para el caso 5 se tomó un batch size de 6, como consecuencia el tiempo que tuvo que utilizar la red para entrenarse fue muy extenso y como resultado no obtuvo una notable diferencia respecto a los anteriores. En cuanto al caso 6, se aumentó significativamente la cantidad de repeticiones y en cuanto a tiempo y resultados fue similar al caso 5.

Una vez elegido al caso 4 como nuestro mejor caso, si bien no pudimos obtener la matriz de confusión en el Colab, lo re-ejecutamos en el IDE Spyder en donde obtuvimos resultados muy aproximados a los de Colab y la matriz de confusión fue la siguiente:

	0	1
0	1938	53
1	347	162

Figura 2. Matriz de confusión.

Error general

En cuanto al error general, en la primera repetición fue del 21,28% y se fue minimizando hasta llegar a un 12.98% en su ultima repeticion (n°300).

El siguiente gráfico nos compara los porcentajes de errores que se obtuvieron a la hora de entrenamiento y testeo.

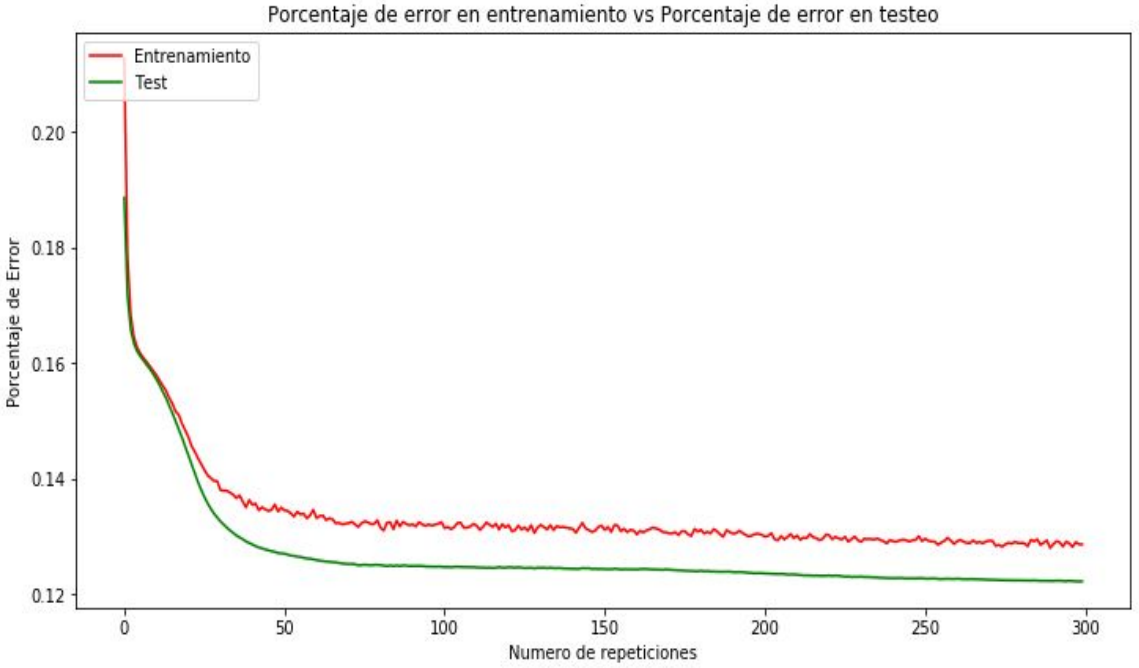

Figura 3. Porcentaje de error en entrenamiento vs Porcentaje de error en testeo

Además de las métricas del error, realizamos la siguiente comparación sobre los distintos porcentajes de exactitud a lo largo de las repeticiones tanto para entrenamiento como para testeo:

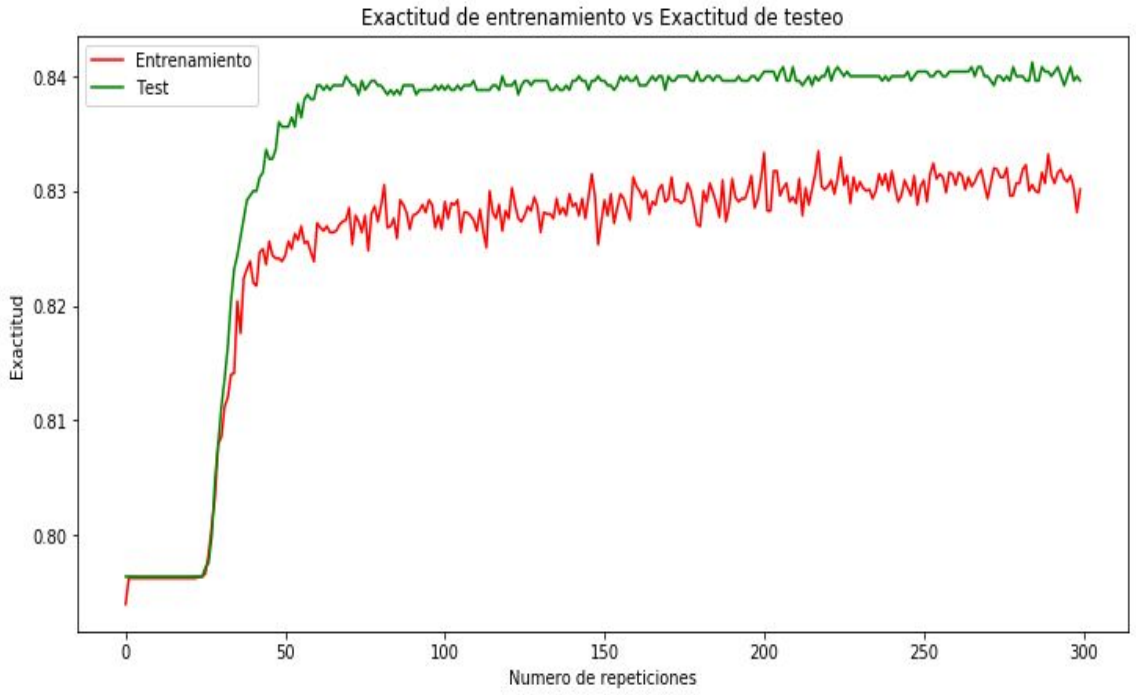

Figura 4. Exactitud de entrenamiento vs Exactitud de testeo

Patrones de entrenamiento

Si bien en el archivo (TP IA - Grupo 7.py) al ejecutarlo en IDE Spyder se puede apreciar completamente los distintos patrones utilizados al visualizar las distintas variables, se nos dificulta mostrar en una imagen el conjunto completo.

Por lo tanto optamos por mostrar una parcialidad de estos patrones normalizados, con algunas imágenes:

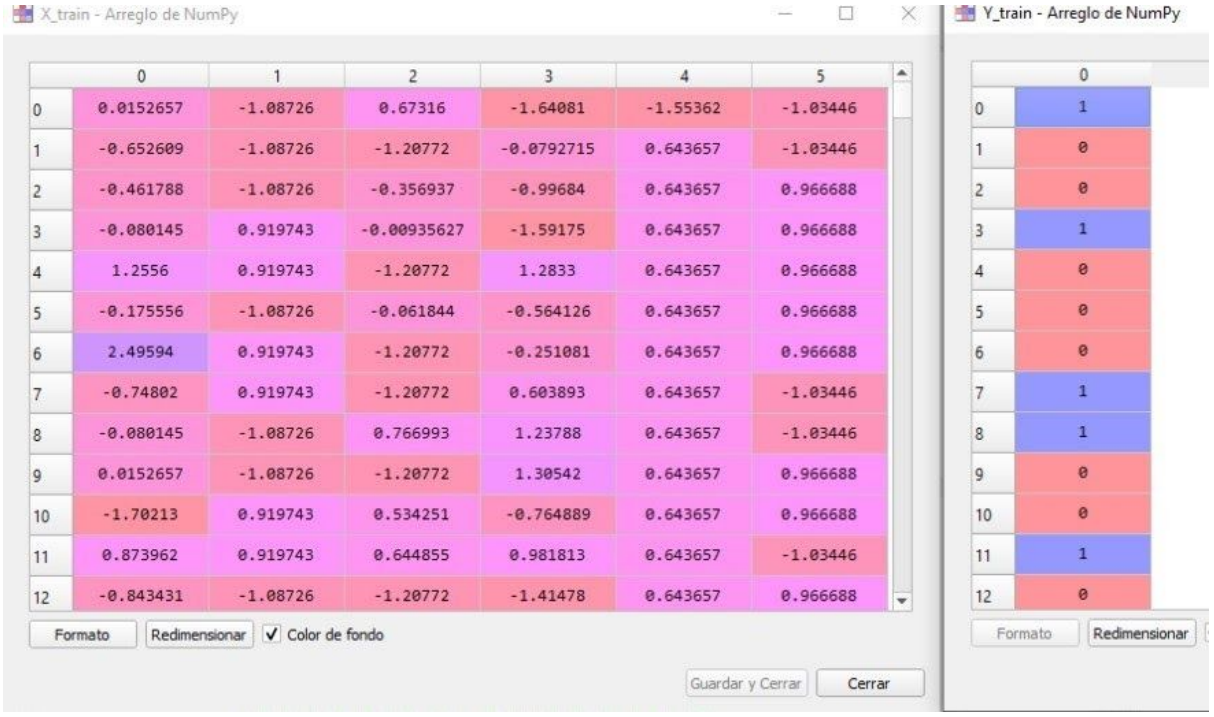

Figura 5. Patrones de entrenamiento

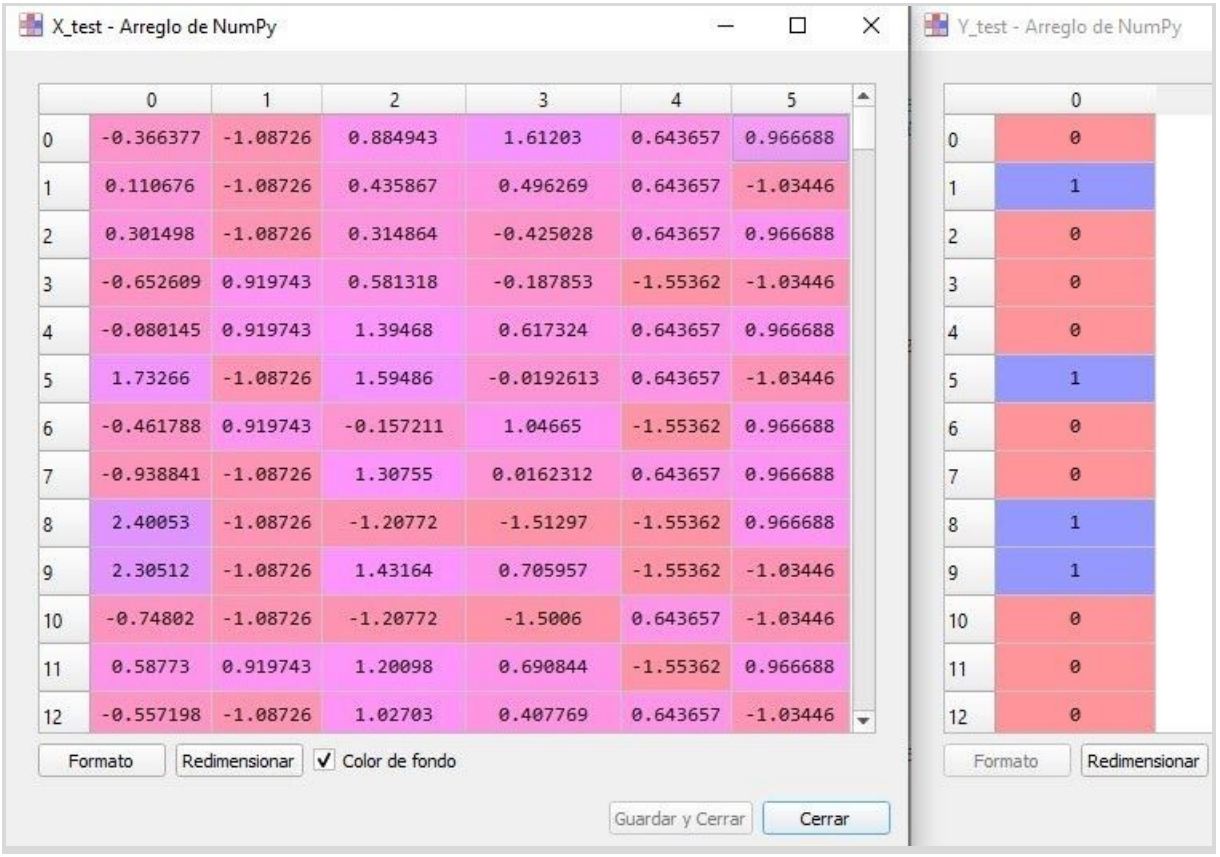

Figura 6. Conjuntos de validación

Discusión

A partir de los distintos resultados que hemos analizado y lo mencionado anteriormente, creemos que nuestro sistema inteligente resuelve satisfactoriamente el problema. También notamos que sumando todos nuestros casos de estudio, no pudimos superar una efectividad o precisión del 84%.

Por lo tanto, nos surgió la duda de que sucedería si agregamos más neuronas de entrada. Hicimos otro caso en donde agregamos dos nuevas neuronas de entrada que representan al puntaje y al número de productos:

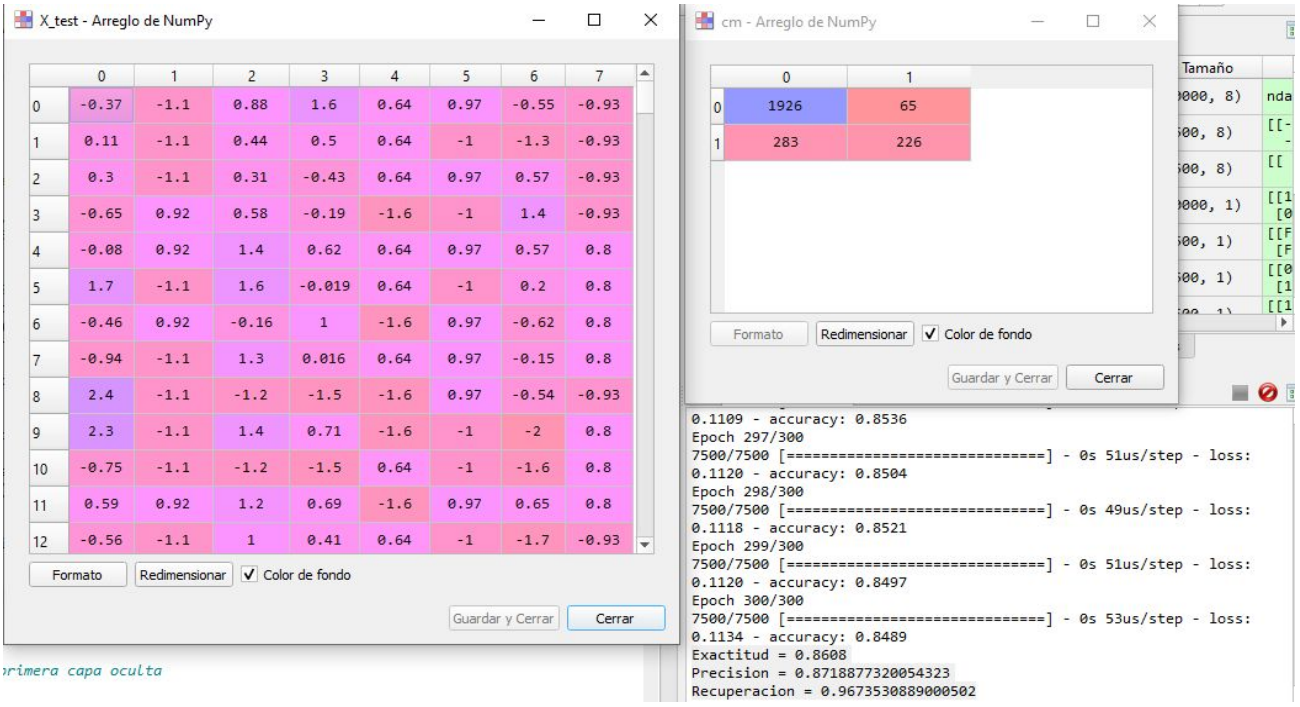

Figura 7. Resultados con 8 neuronas de entrada

Se puede visualizar en la imagen que se alcanzó una exactitud y precisión del 86% y 87% respectivamente con estos nuevos cambios.

Finalmente, podemos observar que se obtuvieron mejores resultados al agregar las dos neuronas en la capa de entrada, sin realizar cambio alguno en la capa oculta.

Conclusión

Relacionando el contenido de la teoría vista en clase y nuestra implementación, nos fueron útiles las distintas recomendaciones dadas por la cátedra.

En cuanto a las funciones de activación, utilizamos la función de rectificador lineal uniforme para la capa intermedia, lo que nos permite que solo los valores positivos sean los significativos, y la función sigmoidal para la capa de salida generando como resultado un valor representativo de la probabilidad de abandono del cliente (entre 0 y 1).
Para interpretar ese resultado, establecimos que si la probabilidad era mayor al 50%, el cliente efectivamente iba a abandonar el banco.

Otra recomendación que adoptamos fue la de la normalización de los datos de entrada para reducir las posibilidades de quedar atrapado en un óptimo local.
Es importante destacar que cuando nuestra RNA esté en producción será necesario que los datos de entrada se normalicen, de lo contrario arrojará resultados erróneos.

Respecto a la implementacion, utilizamos como metodo de optimizacion al gradiente descendiente estocástico (SGD) buscando como objetivo minimizar el error calculado por la función “Error cuadrático medio” a través del ajuste de los pesos de las conexiones.

Utilizamos dicho optimizador con las configuraciones dadas por defecto provenientes de la documentación de Keras. Dentro de ellas figura el coeficiente de entrenamiento, que para este caso su valor fue del 0.01.

Uno de nuestros principales problemas fue ver cómo podíamos tratar al Sexo como una entrada, ya que en nuestro dataset no está representado como un número sino como texto.
Es por ello que tuvimos que codificar este dato categórico para tomarlo como un número utilizando una librería de sklearn.

Bibliografía

- **K52-13_Redes_Neuronales_Artificiales_SI.PDF**
- https://es.wikipedia.org/wiki/Sobreajuste
- https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
- https://en.wikipedia.org/wiki/Stochastic_gradient_descent
- https://keras.io/api/
- https://enmilocalfunciona.io/deep-learning-basico-con-keras-parte-1/
- https://medium.com/@jayeshbahire/perceptron-and-backpropagation-970d752f4e44