

Memoria (Provisional. Resultados todavía incompletos y provisionales.)

Pablo Fernández Piñeiro

14 de noviembre de 2023

Índice

1	Introducción	3
2	Resultado pruebas base	3
3	Resultado pruebas con conjunto de datos asimétrico	6
4	Etiquetado de dataset supervisado	7
5	Red neuronal	7
5.1	Tunning red neuronal	7
5.2	Resultados tunning red neuronal	8
6	Inconvenientes	8
6.1	Problemas de implementación del recomendador	8
6.2	Problemas de predicción del modelo de IA	9
6.2.1	Prueba básica	9
6.2.2	Prueba de validación cruzada	9
6.2.3	Prueba con distintas distribuciones de muestras	10
6.3	Problemas en la paralelización de las simulaciones	11
6.4	Problemas con el uso de red neuronal preentrenada	11
7	Dudas	11
A	Apéndice I: Escenario	12
A.1	Escenario base Martín - Secuencial. Nodov1	12
A.2	Nodov2	12
A.3	Nodov3	13
A.4	Nodov4	13
B	Apéndice II: Datasets	13
C	Apéndice III: <i>GUI</i> para resultados de <i>tunning</i>	14

1. Introducción

El objetivo final de este trabajo es observar los beneficios o desventajas de añadir algún tipo de mecanismos atencionales al escenario descrito en A. Esto se ha hecho a partir del *paper* (REFERENCIA a paper de mecanismos atencionales en el que se comenta posibles aplicaciones de mecanismos atencionales). Se estudiaron varias aplicaciones explicadas en (REFERENCIA a mi otro documento inicial donde planteaba posibilidades y justificaba decisiones), para finalmente decantarse por la opción de implementar un recomendador basado en un transformer, conocidos como *transformers for recommendation*. Esto se utilizaría como recomendador sobre qué prototipos compartir y cuáles no con sus vecinos, lo que incrementaría la eficiencia de las comunicaciones (en situaciones donde se comparten muchos prototipos y no es capaz de entrenarse con todos, mejoraría también la eficacia en la predicción puesto que se entrenaría con el mismo número de muestras pero estas serían las más relevantes).

2. Resultado pruebas base

Como se ha explicado en la intrducción, utilizando el escenario descrito en el apéndice A, se establecen unas pruebas base de partida en las que la diferencia entre ellas es el conjunto de datos de partida, detallado en el apéndice B. Nota: Los resultados de las gráficas representan el promedio de los 5 nodos en cada simulación.



Figura 1: F1: Resultados del experimento base

En el primer caso se aprecia una considerable mejora en función de los parámetros de compartición, en el segundo, con el dataset de *Phishing*, mejora cuando no hay compartición ($T=0$), pero a partir de ahí no se observa mejora. En el último caso, *Electricity* con 250 muestras por nodo, no se aprecia la misma diferencia que en el primero, pero sí se puede intuir cierta tendencia, esto puede ser porque hay una mayor aleatoriedad en este dataset, como se explica el correspondiente apéndice. Siguen realizándose simulaciones a la espera de que se estabilice y se aprecien bien las tendencias como en el primer caso. Para el primer y segundo caso se han hecho 20 iteraciones para cada valor de T y s , en el tercer caso se han realizado 40 iteraciones (por la razón explicada previamente).

LAS ITERACIONES DESCRITAS SON LAS TOTALES QUE SE VAN A REALIZAR, AÚN QUEDAN ITERACIONES POR COMPLETARSE, SOBRE TODO EN EL ÚLTIMO CASO.

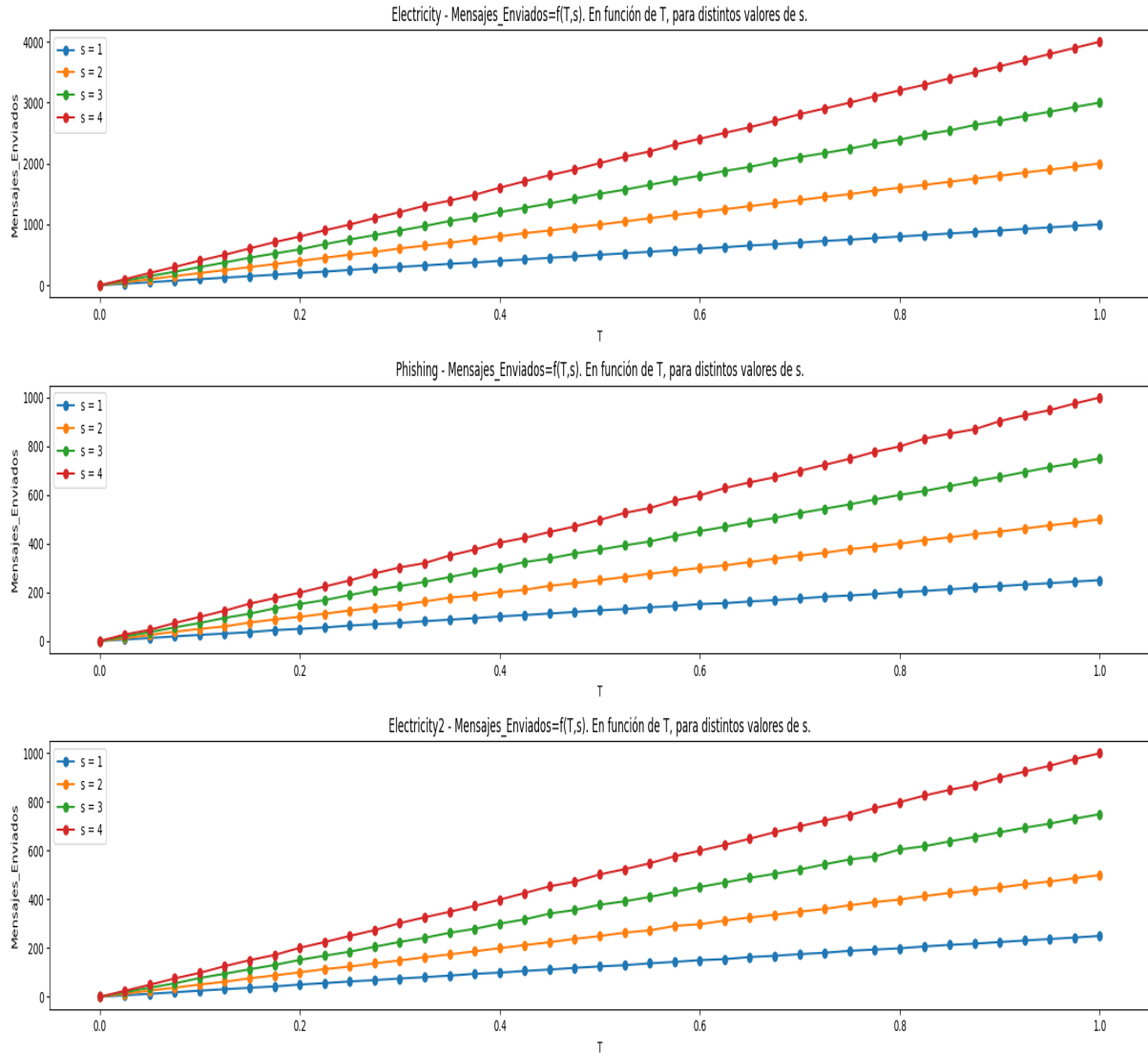


Figura 2: Número de mensajes enviados en el experimento base

En esta gráfica se representa el número de mensajes enviados por cada nodo en cada caso. En cada mensaje va incluido el conjunto de todos los prototipos del nodo que comparte en ese instante. Se puede observar que en el primer caso se envían alrededor de cuatro veces más mensajes, esto es porque tiene exactamente cuatro veces más muestras que los otros dos casos. Esto influye directamente en el número de mensajes enviados y en el de entrenados (mientras no se llegue a la saturación), puesto que con la misma media y distribución de llegada de muestras, dura en media cuatro veces más el experimento. Tener esto en cuenta es importante para entender la siguiente gráfica.

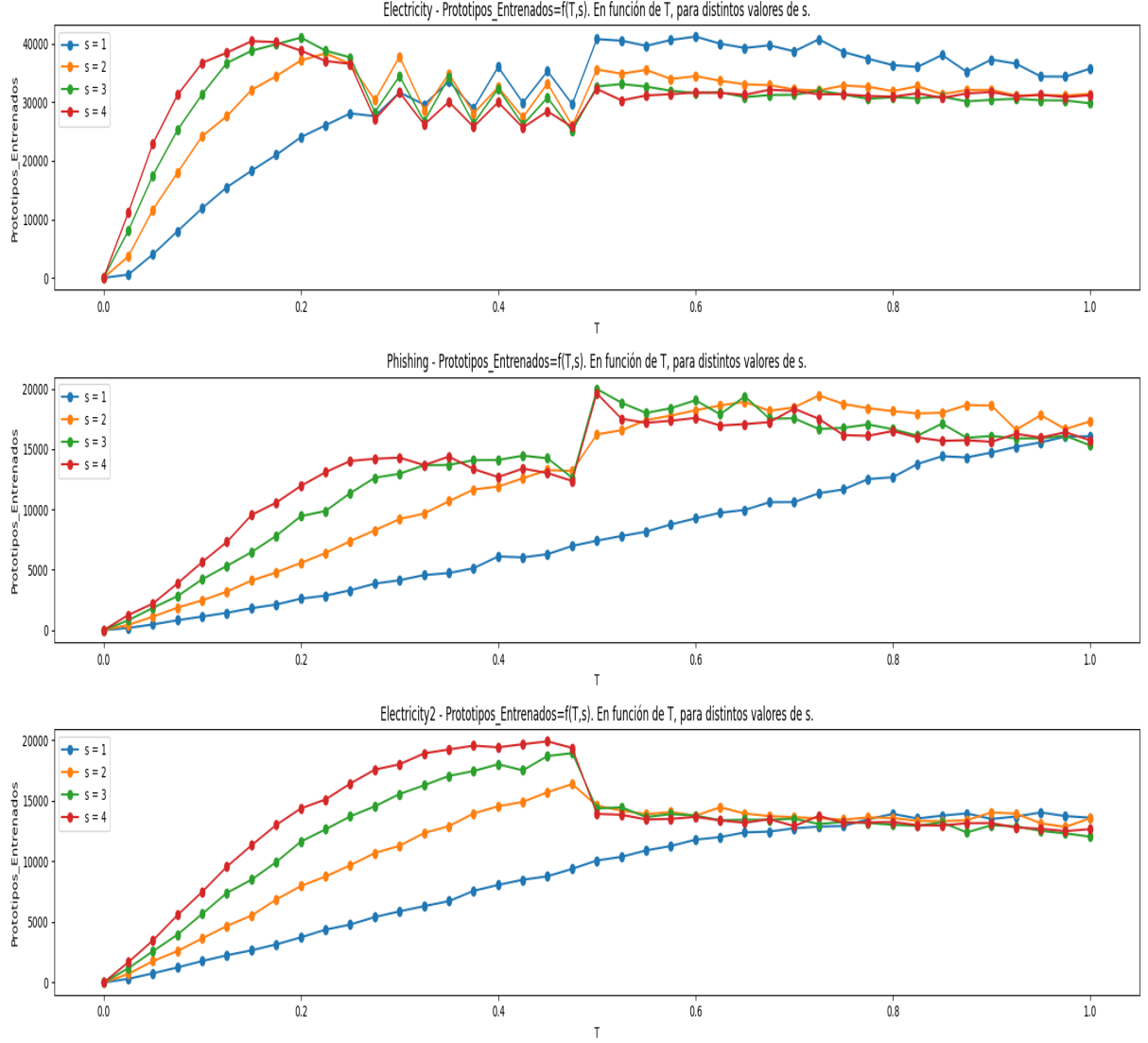


Figura 3: Número de prototipos entrenados en el experimento base

Observando el número de prototipos entrenados, estos convergen en un valor (casi) independiente de los valores de los parámetros de compartición. No es del todo independiente, en el primer caso se observa cuanto menor sea el parámetro “ s ”, con más prototipos se entrena. Este fenómeno sucede porque en el primer caso se entrena con muchos más prototipos (ver Figura 3) debido a que la primera prueba dura cuatro veces más. Como se explica en el apéndice A, los nodos, entre muestras se entrenan con prototipos y comparten su conjunto a los vecinos. El problema es que cuando los parámetros de compartición son muy altos invierte más tiempo en compartir y procesar las colas de prototipos lo que le hace tener un menor tiempo para entrenarse con prototipos.

A pesar de lo anterior, lo interesante es que aumentando los parámetros de compartición, llegando a esta saturación, es cuando se empieza a apreciar una mejora en el rendimiento, en los casos más altos (lo que hace que incluso comience a decaer el número de prototipos entrenados) al tener muchos prototipos más recientes mejora considerablemente su rendimiento. Esto se aprecia en el primer caso, donde la saturación se alcanza para un T aproximado de 0.25, que como se ha justificado previamente, es el valor para el que comienza la mejora del rendimiento del modelo (ver Figura 1).

En cuanto a los otros dos casos a pesar de tener unas gráficas idénticas en cuanto al número de mensajes enviados, se aprecia que en el segundo caso, se satura en $T=1$ prácticamente, lo que hace que no se vea esta mejora en la capacidad predictiva. En el último caso se satura antes y, aunque bastante más dispersa (tiene una mayor componente aleatoria, ver B), comienza a verse esa tendencia anteriormente

mencionada de mejora en el rendimiento. También cabe mencionar que a pesar de tener el mismo número de mensajes enviados y muestras en el dataset puede ser contradictorio que se entrene con más prototipos (y se sature antes), esto se justifica con la siguiente gráfica (ver Figura 4).

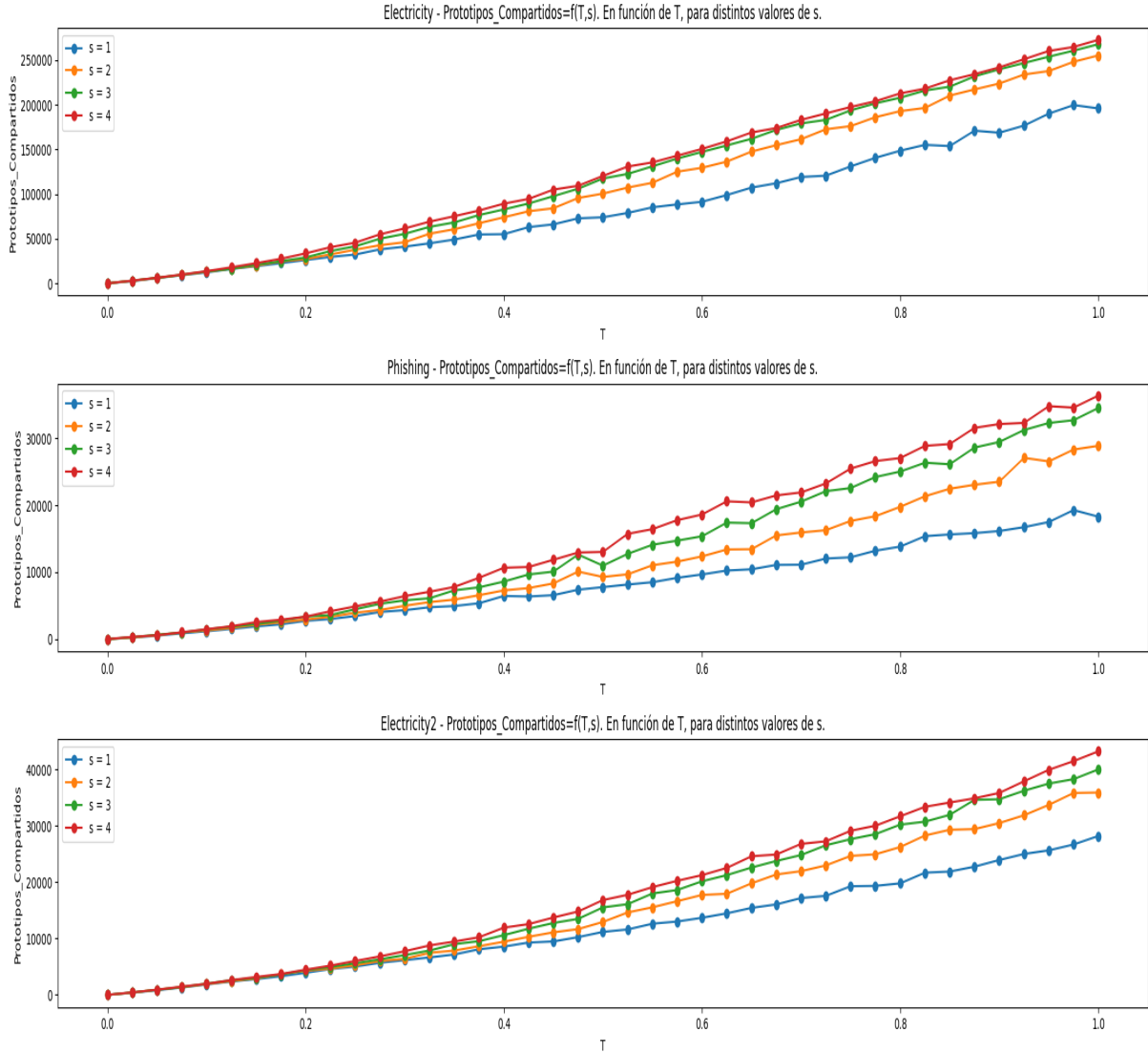


Figura 4: Número de prototipos compartidos en el experimento base

Esta Figura 4 sirve para justificar la conclusión previa. Comparando el segundo *dataset* con el tercero se ve un incremento aproximado del 33 % (30k vs 40k) en el número de prototipos compartidos a igualdad de mensajes. Esto lo que muestra es que el tercer *dataset* por su naturaleza hace que el modelo genere más prototipos, lo que conlleva a una más temprana saturación, y, finalmente, una más temprana mejora del rendimiento.

(MEDIR TIEMPO CONSUMIDO EN LA FUNCIÓN DE COMPARTIR Y PREOCESAR COLAS PARA JUSTIFICAR EL MENOR NÚMERO DE PROTOTIPOS ENTRENADOS CUANDO SE SATURA).

3. Resultado pruebas con conjunto de datos asimétrico

Aquí, la diferencia con las pruebas anteriores es que se prueba con un conjunto de deatos asimétrico respecto a los nodos, quiere decir, no tienen la misma cantidad de datos todos los nodos, en este caso, se procede a estudiar el escenario cuando un nodo tiene menos muestras que el resto, y comprobar si la compartición de prototipos le ayuda a mejorar su rendimiento considerablemente. Se estudia también

como afecta el recomendador en este caso.

SIMULACIONES Y SUS CONSECUENTES CONCLUSIONES A LA ESPERA DE QUE ACABEN LAS PRUEBAS BASE.

4. Etiquetado de dataset supervisado

Este problema parte de la base siguiente: El modelo ILVQ genera los prototipos a medida que se entrena con el dataset supervisado, estos prototipos son utilizados para predecir y son similares a las muestras, estos se van adaptando a las muestras con las que se va entrenando el modelo, por lo tanto, se sabe que las muestras son similares (en algunos casos iguales) a los prototipos, siendo siempre iguales en formato.

Para poder entrenar al modelo recomendador de prototipos se propone etiquetar el dataset existente según la relevancia de cada muestra. Este etiquetado se basa en la información que aporta cada muestra a un conjunto de muestras dado, partiendo de la idea que las muestras menos comunes aportan más información que las más comunes (según un ejemplo que puso Martín: Si todas las muestras dicen que cuando hace sol no llueve, daría más información al modelo una muestra nueva que diga que en el ecuador cuando hace sol puede llover, que otra muestra que diga lo mismo que todas las anteriores). Esta información se mide basándose en la entropía, más concretamente en la divergencia de Kullback-Leibler, que mide cuánto se aleja una distribución de otra de referencia.

En este caso se sigue un esquema de ventana deslizante, calculando la divergencia de Kullback-Leibler de la ventana respecto a la referencia que sería el resto del dataset. El tamaño de la ventana es de X muestras.

TODAVÍA REQUIERE ESTUDIO, REVISAR EN DUDAS, 7.

5. Red neuronal

Para el recomendador, se ha probado una red neuronal para poder evaluar si una muestra es relevante o no. Esta red neuronal se ha entrenado con el dataset generado según la sección 4. Es una red neuronal densa (*Feedforward Neural Network*, *FFN*), sus hiperparámetros se explican en el siguiente punto.

5.1. Tuning red neuronal

Una vez escogido el tipo de red neuronal, se procede a estudiar los mejores hiperparámetros siguiendo una estrategia *grid search*. En concreto se han hecho pruebas con los siguientes parámetros:

- Número de neuronas de entrada (en realidad este parámetro simplemente se adecuaba al número de variables predictoras de cada dataset). El número de neuronas de salida sólo es 1, pues sólo se desea predecir si es relevante o no una muestra.
- Número de neuronas por cada capa oculta. Rango [1,46] de 5 en 5.
- Número de capas ocultas. Rango [0, 10].
- Número de épocas de entrenamiento. [10, 100, 500, 1000].
- Umbral de decisión. Esto es, la red genera una salida entre 0 y 1 pero el resultado de la predicción final tiene que ser binario, este valor marca a partir de donde se considera 1 o 0 la salida. Rango [0.01, 0.1, 0.2, 0.3, 0.4...1]

Para realizar las pruebas, se han utilizado los datasets completos de *Electricity* y de *Phishing* con las muestras etiquetadas por relevancia.

Se ha elaborado un sistema de puntuación que valora la $F\beta$ -Score y los tiempos de entrenamiento y testeo. La fórmula es la siguiente:

$$Puntuacion = w1 \cdot F\beta - (w2 \cdot Tiempo_entrenamiento + w3 \cdot Tiempo_testeo) \quad (1)$$

Donde:

- w_1 : Es el peso que se le da al valor de $F\beta$, en este caso, $w_1 = 1$.
- $F\beta$: Métrica $F\beta$ -Score, para tener en cuenta tanto la *Precision* como el *Recall* pero haciendo una ponderación. En este caso $\beta = 0.2$, lo que le da un peso 25 veces mayor a la *Precision* que al *Recall*. Esto se ha hecho porque en este contexto el número de prototipos que se va a compartir es enorme, por lo tanto se desea que los que son compartidos sí sean importantes, no importa si se descartan algunos prototipos relevantes.
- w_2 : Peso que se le da al tiempo de entrenamiento. Tiene un valor de $w_2 = 0.05$. Se le da poco peso porque la implementación se hace con la red neuronal preentrenada, pero se pone porque, indirectamente tiene correlación con el tiempo de testeo.
- Tiempo de entrenamiento: Es el que ha tardado en entrenarse la red neuronal con una configuración dada, está en minutos.
- w_3 : Peso del tiempo de testeo. Mayor peso que el de entrenamiento porque simula mejor su uso final. $w_3 = 0.2$
- Tiempo de testeo. Análogo al tiempo de entrenamiento pero para la fase de test, en minutos también.

Las 3 con mayor configuración han sido las siguientes, con el siguiente formato:

(neuronas por capa oculta, número de capas ocultas, número de épocas de entrenamiento, valor umbral de decisión)

1. (21, 6, 100, 0.9), Puntuación promedio asociada: 0.92386
2. (46, 6, 100, 0.9), Puntuación promedio asociada: 0.91984
3. (46, 6, 100, 0.8), Puntuación promedio asociada: 0.91930

5.2. Resultados tuning red neuronal

Por comodidad a la hora de representar los resultados de las pruebas de *tunning* de los hiperparámetros de la red neuronal se ha implementado una sencilla interfaz gráfica. Esta se explica detalladamente en el apéndice C. La configuración que maximiza la función de puntuación sería con 21 neuronas por capa oculta, 6 capas ocultas, 100 épocas de entrenamiento y un umbral de decisión de 0.9. Sería óptima porque, aunque al tener menos neuronas y/o capas tenga una pequeña disminución en la $F\beta$, la fórmula de puntuación no sólo evalúa la eficacia, también tiene peso la eficiencia en términos del tiempo invertido, que está directamente correlacionado con la carga computacional requerida.

6. Inconvenientes

Han surgido varios problemas, se describen a continuación:

6.1. Problemas de implementación del recomendador

Adaptar un transformer a este caso es complejo porque son complejos de implementar, sobre todo de entrenar y están orientados a NLP, aun así la rama de transformers recomendadores está más cerca de lo que necesito, pero sigue siendo insuficiente.

ESTOY INTENTANDO IMPLEMENTARLO, ESTOY SIGUIENDO UN LIBRO SOBRE TRANSFORMERS(Transformer for Natural Language Processing de Denis Rothman).

Se ha llegado a la conclusión de que, en el fondo, si el dataset está etiquetado según relevancia, cualquier modelo de Machine Learning puede predecir si una muestra es relevante o no según este etiquetado.

6.2. Problemas de predicción del modelo de IA

Surgen problemas con el *dataset Electricity* de River, con el de *Phishing* no aparecen dado que tiene muchas menos muestras y la simulación se realiza con el dataset completo (además se intuye un menor *concept drift*), no extrayendo *subsets* como en el caso de *Electricity* (utilizar este *dataset* completo es inviable).

En cuanto al dataset de *Electricity*, se parte de que este modelo entrenado de forma *online* con todo el dataset da un resultado de F1 rondando el 80 % como se muestra más adelante en la *Prueba básica*, mientras que en las primeras pruebas, siguiendo el apéndice B, se consigue una media de F1 entre los nodos inferior al 50 %, como se puede apreciar en 2. Una pequeña reducción de su rendimiento podría ser comprensible debido a que cada modelo se entrena con un menor número de muestras, pero en este caso es demasiado grande.

Para intentar solucionar estos problemas y sobre todo entenderlos, se han realizado las siguientes pruebas:

6.2.1. Prueba básica

Esta prueba consiste, como se ha mencionado anteriormente, en realizar *Predict then Train* con todo el dataset para ver un resultado general, las métricas obtenidas son las siguientes:

Precisión: 0.766, Recall: 0.725, F1: 0.745

6.2.2. Prueba de validación cruzada

Esta prueba consiste en realizar un estudio de validación cruzada dividiendo en 5 partes el dataset, se han hecho pruebas con 1250 muestras en 5 *folds*, lo que se corresponde con 1000 muestras de entrenamiento y 250 de test que van alternándose. Además del modelo ILVQ, se crean otros dos modelos de comparación, un predictor aleatorio y un predictor por mayoría (siempre predice la etiqueta mayoritaria). Esta prueba no simula realmente cómo va a funcionar el modelo en el escenario más realista, pero se hace para estudiar mejor el dataset y entender el por qué del tan mal rendimiento del modelo (casi al nivel de un predictor aleatorio). Con la validación cruzada con K=5 pliegues, se dividen las 1250 muestras en 5 subconjuntos de 250 muestras, se entrena con 4 de estos y se testea con 1. Se realizan 1000 iteraciones para que la aleatoriedad presente en las pruebas sea despreciable, ya que en cada una de las iteraciones se escoge un *subset* de 1250 muestras distinto. Cada iteración conlleva la validación cruzada con cada uno de los subconjuntos.

■ Modelo: XuILVQ

- Precisión: Media = 0.558, Desviación Típica = 0.093
- Recall: Media = 0.480, Desviación Típica = 0.137
- F1: Media = 0.505, Desviación Típica = 0.106
- Accuracy: Media = 0.622, Desviación Típica = 0.033

■ Modelo: Random

- Precisión: Media = 0.424, Desviación Típica = 0.043
- Recall: Media = 0.500, Desviación Típica = 0.047
- F1: Media = 0.458, Desviación Típica = 0.039
- Accuracy: Media = 0.500, Desviación Típica = 0.031

■ Modelo: Majority

- Precisión: Media = 0.000, Desviación Típica = 0.000
- Recall: Media = 0.000, Desviación Típica = 0.000
- F1: Media = 0.000, Desviación Típica = 0.000
- Accuracy: Media = 0.577, Desviación Típica = 0.030

La desvaición típica se muestra para observar la variabilidad que presentan cada una de las medias.

Nota: El modelo por mayoría tiene precisión, recall y f1 a 0 porque en este dataset la mayoría de las etiquetas son negativas y estas métricas evalúan los verdaderos positivos, nunca predice positivo y por esa razón salen a 0.

6.2.3. Prueba con distintas distribuciones de muestras

En este caso se comparan 3 instancias de ILVQ pero con distintos conjuntos de datos, todos siguen el esquema *Test then Train*. En el primer caso, se representa el valor de la métrica F1 para ciertos números de muestras elegidas consecutivamente sobre el dataset entero, en el segundo caso la diferencia es que las muestras se obtienen de 5 en 5, como en el escenario, y en el tercer caso se cogen aleatoriamente.

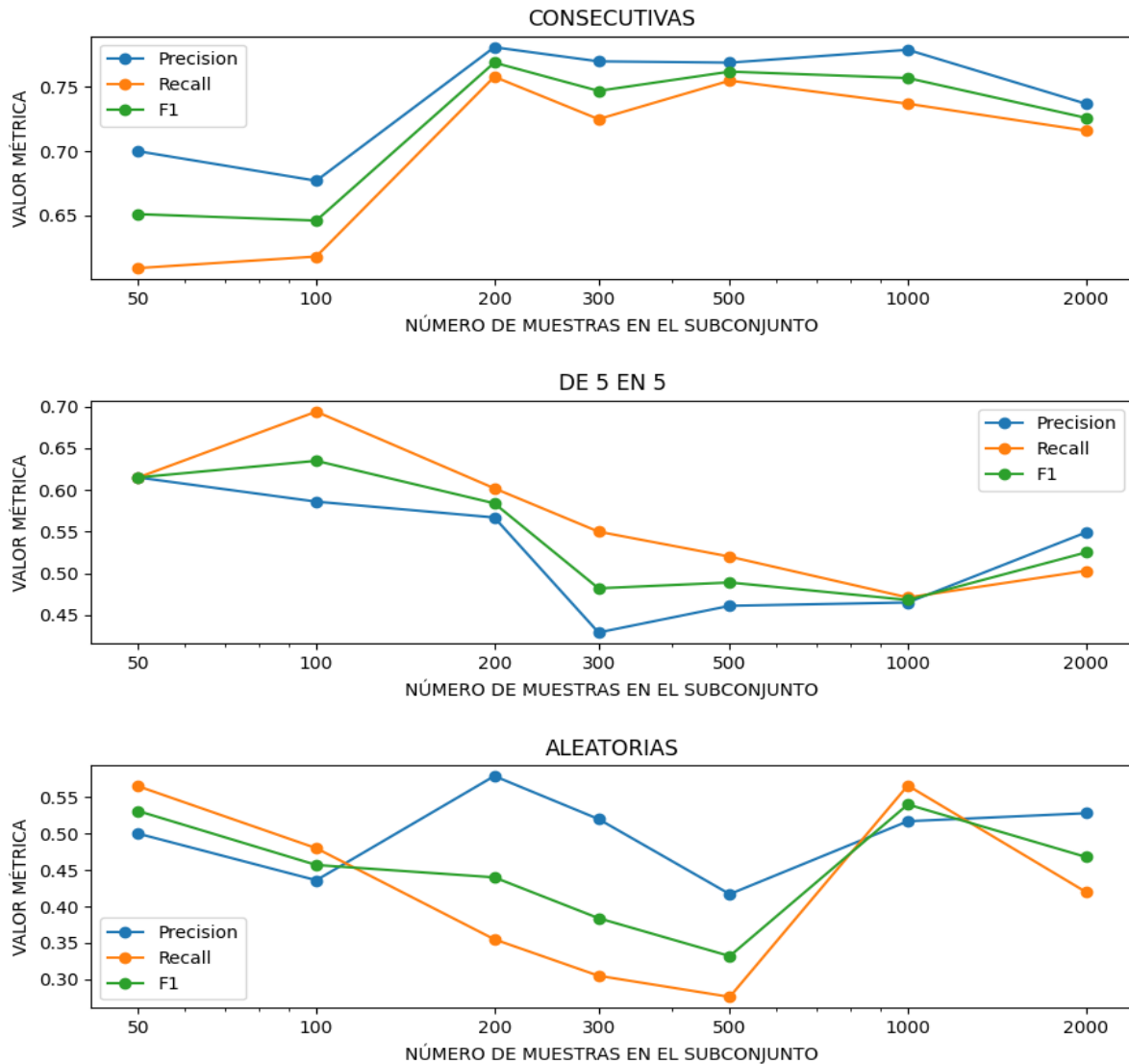


Figura 5: Gráficas para las distintas distribuciones de los conjuntos de datos

En el caso aleatorio, se puede apreciar que el rendimiento es deficiente, en cambio con las muestras de 5 en 5 ya comienza a ser aceptable, en el caso de consecutivas se aprecia que el rendimiento es muy superior a los otros dos casos.

Tras comparar los resultados previos, se procede a realizar una nueva prueba, que consiste en ir eligiendo muestras alternadas como en el segundo caso anterior, pero variando el paso en cada ejecución. El número de muestras en este caso, se fija a 1000 muestras.

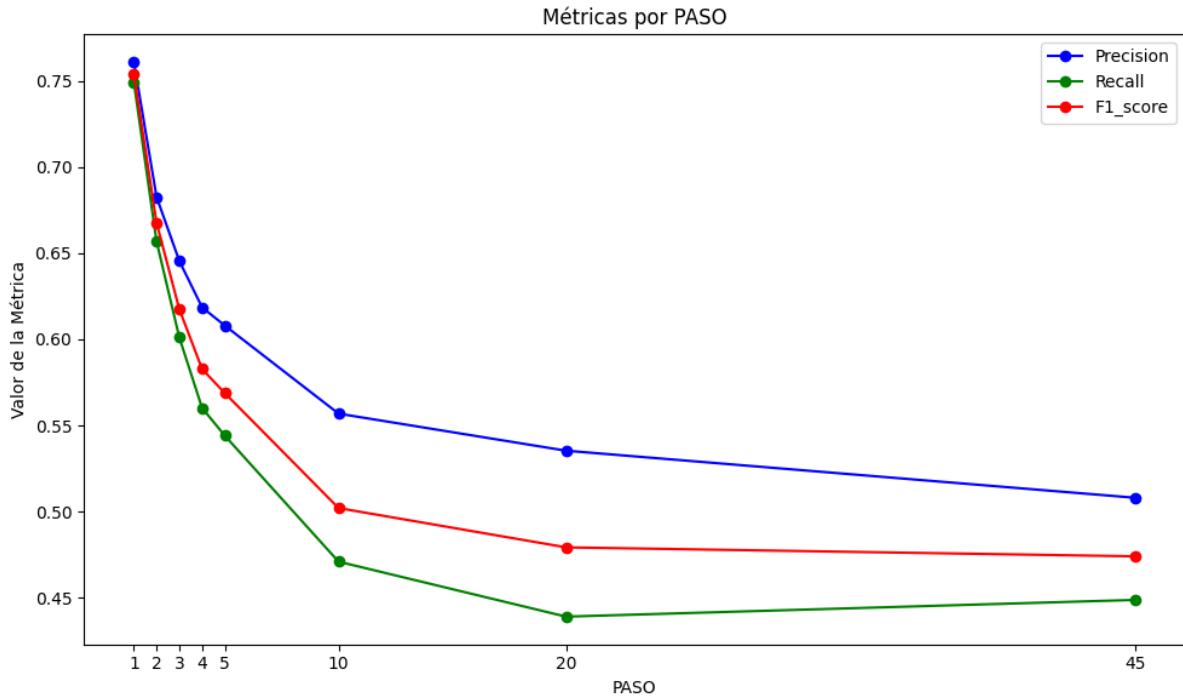


Figura 6: Gráfica para evaluar el rendimiento variando el paso entre muestra y muestra alterna.

Comparando los resultados, se puede concluir que todo el problema es debido al *concept drift*, con muestras aleatorias no predice absolutamente nada porque no es capaz de adaptarse, y cuanto menor sea el paso entre muestras alternas, mejor predice, se adapta mejor al *concept drift*.

6.3. Problemas en la paralelización de las simulaciones

Al paralelizar el experimento de Martín, que es secuencial, han surgido una serie de cuestiones acerca del flujo que debe seguir el programa. Esto está más detallado en A. El flujo previo carece de sentido cuando se pasa a paralelo y los nodos siguen entrenándose con prototipos cuando ya se acaban las muestras (al paralelizar), esto acababa derivando en problemas de sincronización entre los nodos.

6.4. Problemas con el uso de red neuronal preentrenada

El uso de una red neuronal preentrenada como recomendador podría tener algunos problemas, como por ejemplo, no adaptarse al *concept drift* porque sólo podría ser entrenada con los datos conocidos a priori. Estudiar si realmente es un problema porque la red neuronal debería de inferir si una muestra es relevante o no en términos de teoría de la información, y esta, evidentemente, es independiente del *concept drift* que pueda tener un dataset.

7. Dudas

1. En el apartado de **etiquetado del dataset**, para las entropías/divergencia Kullback-Leibler:
 - a) **Discretizar** y posteriormente utilizar cálculos de **entropía discreta**.
 - b) **Inferir fdp** (función de densidad de probabilidad) a partir de la función de masa de probabilidad discreta obtenida para cada variable predictora del dataset y utilizar cálculos de **entropía continua** (Esta última me parece a priori mejor pues hay menor pérdida de información por la ausencia de discretización y se adapta mejor al contexto del problema donde los datos son casi continuos).

En caso de utilizar cálculos discretos, la discretización es necesaria pues aunque el dataset tenga valores discretos (una CPU no puede manejar valores continuos), contienen muchos decimales, son “pseudocontinuos” habría que discretizarlos pues hay muchos valores que sólo se repiten una sola

vez (o muy pocas) en algunas variables.

Hay métodos de discretización sencillos ya implementados como el de igual ancho o igual frecuencia, pero para este caso utilizaría un método de discretización *ChiMerge* (basado en la prueba de independencia chi-cuadrado) o basado en **entropía mínima**. Este último resumidamente se basa en establecer unos puntos de corte entre los valores de la variable y se escogen los cortes que minimicen la entropía de los subconjuntos generados. Estos métodos de discretización son los que mejor se adaptan a distribuciones inusuales (como las que tienen algunas variables del dataset), si tuvieran distribuciones normales o incluso uniformes, un discretizador más simple como los de ancho y frecuencia serían suficientes. En específico el discretizador chi-cuadrado es recomendable para entornos de aprendizaje supervisado, y el de entropía cuando se quiere minimizar la pérdida de información por discretización.

2. Respecto a la **idea de implementar un recomendador**. Al fin y al cabo, lo que se están haciendo son cálculos sobre los cambios de información al añadir unas muestras a un conjunto ya existente (el de prototipos). Se podría implementar la ganancia de información, entropía, etc. con una función que calcule esto e implementar algunas reglas a seguir como superar cierto umbral de ganancia de información. Esto creo que tendría unos resultados similares, o incluso mejores puesto que el objetivo final de IA es saber si una muestra o prototipo aporta ganancia de información, este modelo puede fallar, su cálculo directo no pues es algo matemático sin “aleatoriedad” ninguna. Otra ventaja sería la reducción de capacidad de cómputo necesaria.
3. En cuanto a la **entropía enventanada** o de ventana deslizante, buscando información no entiendo como etiquetar cada muestra individualmente a partir de la entropía, en este caso divergencia de Kullback-Leibler. Quiero decir, por ejemplo, si tengo 100 muestras en total, una ventana de 10 muestras, calcularía la divergencia entre la distribución formada por las muestras de la ventana respecto al resto de muestras (este conjunto de referencia, sería con o sin muestras de la ventana, entiendo que sin ellas pero no estoy seguro). ¿Cómo voy etiquetando cada muestras, calculo la divergencia con la ventana de la muestra 0 a la 9, luego desplazo la ventana de la 1 a la 10 y si la divergencia es menor es porque la primera muestra era más relevante y se etiquetaría como tal (podría etiquetarse directamente con la diferencia de las divergencias entre los dos casos para tener un valor de relevancia)?

A. Apéndice I: Escenario

Partiendo de un esquema de aprendizaje federado descentralizado en el que cada nodo ejecuta el modelo ILVQ, se comparten prototipos aleatoriamente entre los nodos siguiendo un protocolo de compartición aleatorio con dos parámetros, s (número de vecinos a los que se le comparte), y T (probabilidad de que se compartan o no los prototipos) se ha probado lo siguiente:

En cuanto a la arquitectura, apenas hay modificaciones, en cambio, los nodos que la componen han sufrido pequeñas actualizaciones, cada una de ellas para simular un escenario más realista.

A.1. Escenario base Martín - Secuencial. Nodov1

Cada nodo, predice (y entrena) con la muestra que corresponde del dataset, comparte prototipos (siguiendo protocolo de compartición) y se entrena con todos los prototipos de la cola, por turnos (primero hace todo esto el nodo 0, luego el nodo 1, etc.).

A.2. Nodov2

Se procede a paralelizar la ejecución de los nodos, se utilizan colas IPC, para las comunicaciones inter-proceso de los nodos. El flujo del programa sigue siendo el mismo. Esto causa problemas, puesto que el enfoque previo tiene sentido en un escenario secuencial pero no en uno paralelo. No es realista entrenarse con todos los prototipos recibidos entre muestra y muestra independientemente de los parámetros de compartición.

Se implementa la serialización de los mensajes utilizando la librería Pickle de Python. Esto reduce el tamaño (en bytes) de los mensajes a intercambiar.

A.3. Nodov3

Se añaden colas LIFO para aprender siempre de los prototipos más recientes. Se establecen tiempos de llegada para cada una de las muestras de cada nodo siguiendo una distribución exponencial con media de 0.25 segundos entre muestras. Esta media se ha escogido por las siguientes razones:

- Para tener un compromiso entre tiempo de simulación y tiempo entre muestras que permita el entrenamiento con un número de prototipos aceptable.
- Para observar un fenómeno de saturación de las colas (estas no se llegan a vaciar) lo que da un resultado un tanto diferente.

Cada nodo tiene una cola LIFO para cada vecino, esto es para separar los prototipos de cada vecino e ir aprendiendo de todos los vecinos para conseguir una mejor adaptación al *concept drift*. El flujo que sigue es el siguiente:

1. Primero se comprueba si quedan más muestras para predecir, si quedan, avanza al paso 2., si no quedan, finaliza la ejecución.
2. Elige la cola del vecino con id inmediatamente siguiente a él mismo. (Si es el último, escoge el id = 0)
3. Comprueba si este vecino le ha dejado prototipos o no en esta cola. Si hay prototipos, se entrena con uno y avanza a la siguiente cola, si no, avanza a la siguiente cola directamente.
4. En la siguiente cola realiza lo mismo que en el paso 3.
5. Se repite cíclicamente hasta que llega la siguiente muestra, realiza su correspondiente *Predict, then Train* y vuelve al paso 2.

Esto conlleva un ligero cambio en el flujo de código a ejecutar por el nodo. Se añade también la opción de utilizar modelo recomendador añadido al principal ILVQ.

A.4. Nodov4

Aún no se han hecho pruebas con esta versión, la principal adición es la limitación del tamaño de las colas de prototipos. Esto se hace porque entre muestra y muestra el nodo se puede entrenar con X prototipos (depende del tiempo entre muestras y de la capacidad de cómputo del nodo en un caso real).

Al limitar las colas, lo que se hace es que en la cola sólo haya los prototipos más recientes. Antes se entrenaba con los prototipos más recientes primero, pero si tardaba mucho en llegar la siguiente muestra, el modelo acaba por entrenarse con los prototipos del final de la cola (más antiguos), esto produce que, cuando finalmente le llega la muestra e intenta predecir, los últimos prototipos con los que se ha entrenado están más desactualizados. Esto es muy importante en escenarios de aprendizaje incremental o aprendizaje en línea debido al *concept drift*.

B. Apéndice II: Datasets

En cuanto a los datasets se utilizan los dos datasets siguientes: Electricity y Phishing, ambos con *concept drift* puesto que son datasets para *online learning*. El dataset Electricity se utiliza de dos formas distintas, se explica a continuación:

1. **Electricity:** En este caso se obtienen las cinco mil primeras muestras del dataset y se asignan de 5 en 5 a cada nodo, es decir, para el nodo 0, se asignan las muestras 0,5,10...4995, para el nodo 1, las muestras 1,6,11...4996 y así hasta el nodo 4. De esta forma cada nodo tiene mil muestras intercaladas, esto se hace para simular mejor un entorno real donde las muestras tienen cierta relación temporal. Esto se hace porque el dataset es demasiado grande para realizar las simulaciones en un tiempo prudente, tiene un total de 45312 muestras.
2. **Phishing:** Para este dataset no hay problema en coger todas las muestras puesto que “sólo” tiene 1250, por lo tanto se dividen de forma intercalada como se explicó previamente y cada uno de los 5 nodos acabaría teniendo 250 muestras para su correspondiente *Predict then Train*.

3. **Electricity2**: Este dataset es el mismo que el primero, pero se escogen diferentes muestras, en este caso se cogen 1250 muestras consecutivas del dataset, el índice de la primera muestras es escogido aleatoriamente en el rango $[0, \text{tamaño}(\text{dataset}) - 1250]$. Esto se hace para tener en unas condiciones similares el dataset de Electricity y el de Phishing, a la vez que se descarta cualquier posible sesgo en las primeras cinco mil muestras que siempre se utilizan en el primer caso de Electricity.

C. Apéndice III: *GUI* para resultados de *tunning*

Esta interfaz gráfica se ha implementado con varios propósitos:

- Principalmente, hacer más cómoda la visualización de los resultados.
- Otra razón muy importante, es que se le han añadido filtrados por cualquiera parámetro, se pueden concatenar estos filtrados. Por ejemplo, se pueden ver los resultados de la métrica F1 en función del número de neuronas pero sólo para el *dataset* “Ejemplo” y para “N” capas ocultas.
- Este filtrado ha ayudado enormemente en la detección de anomalías o sesgos relacionados con un dataset o configuración concreta. Ha sido muy útil para depuración.

A continuación se muestran capturas de pantalla de la interfaz con algunos ejemplos:

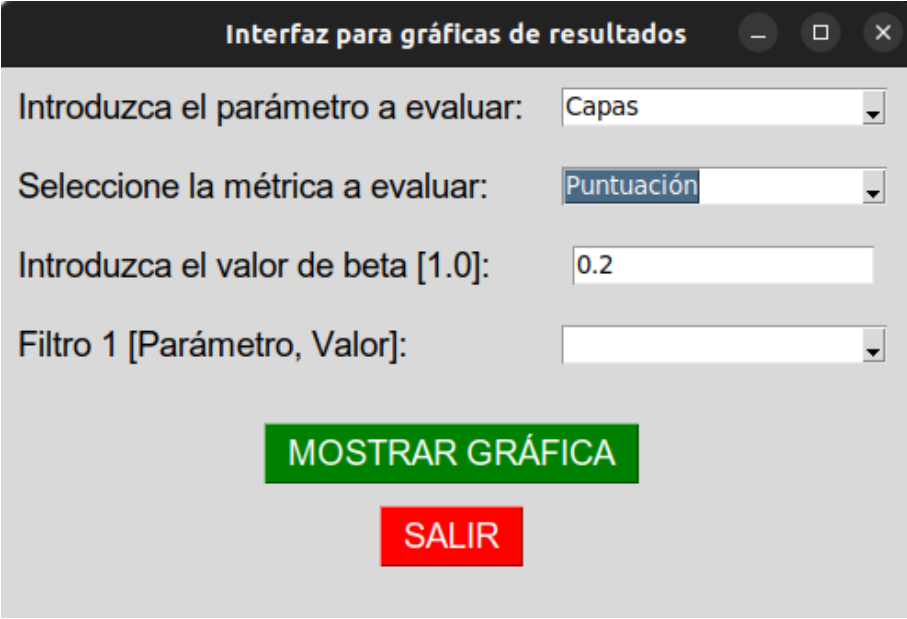


Figura 7: Captura menú. Ejemplo 1. Sin filtros.

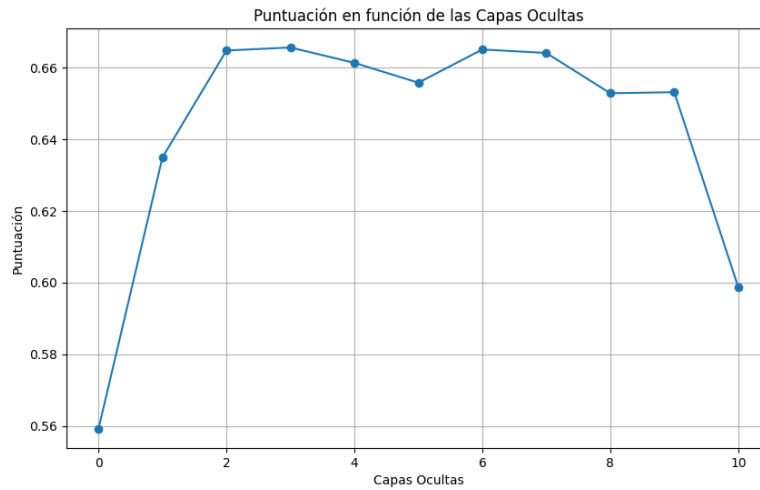


Figura 8: Captura gráfica. Ejemplo 1. Puntuación con $\beta=0.2$ en función de las capas.

Figura 9: Captura menú. Ejemplo 2. Filtrando para dataset="electricity" y capas=5

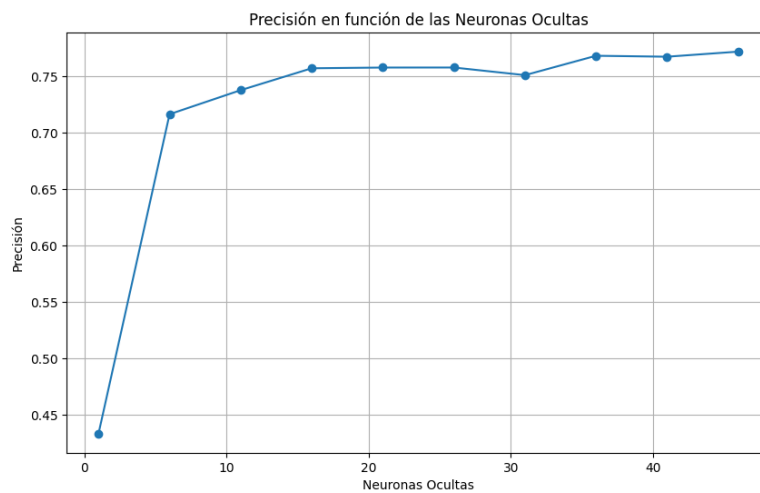


Figura 10: Captura gráfica. Ejemplo 2. Precisión en función de las neuronas por capa oculta

Interfaz para gráficas de resultados

Introduzca el parámetro a evaluar:

Seleccione la métrica a evaluar:

Filtro 1 [Parámetro, Valor]:

Filtro 2 [Parámetro, Valor]:

Filtro 3 [Parámetro, Valor]:

Filtro 4 [Parámetro, Valor]:

Filtro 5 [Parámetro, Valor]:

MOSTRAR GRÁFICA

SALIR

Figura 11: Captura menú. Ejemplo 3. Aplicando todos filtros posibles para un parámetro.

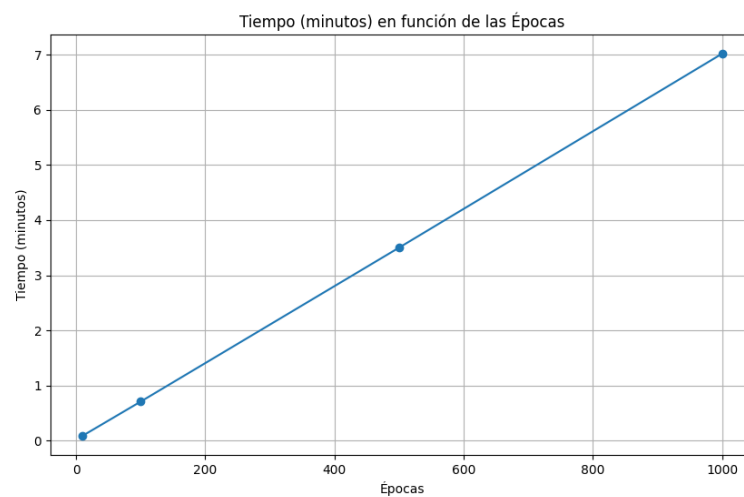


Figura 12: Captura menú. Ejemplo 3. Tiempo de ejecución en función de las épocas de entrenamiento.