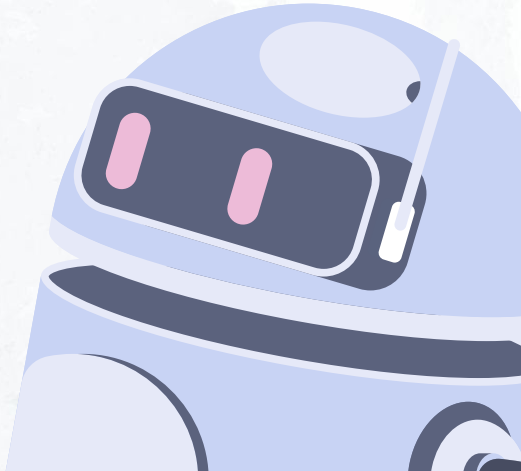
Clasificación de gestos

Proyecto Final Aprendizaje Automático

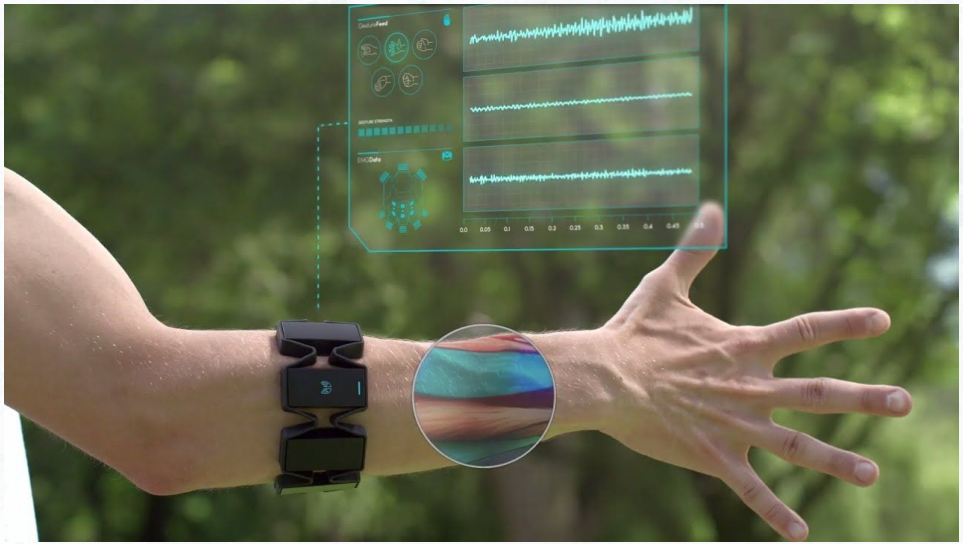


Grupo 5: Jorge Ortega y Daniela Vidal

Índice

1 Presentación del dataset

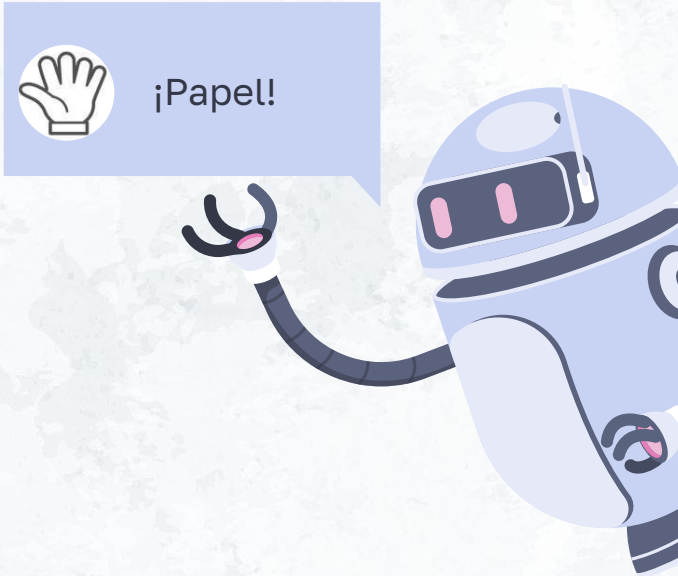
Reconocimiento de gestos con datos de actividad muscular



- Pulsera MYO armband que reconoce impulsos eléctricos en los músculos
- 8 sensores
- 8 medidas consecutivas de los 8 sensores (40ms)

Por cada gesto:

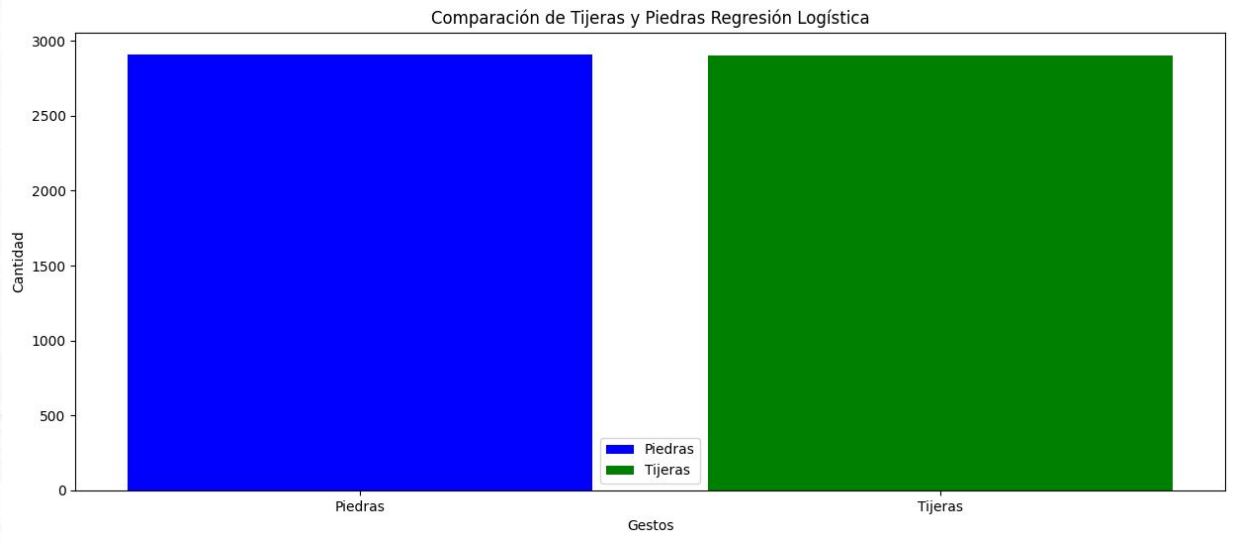
- 120s totales
- 2900 filas de 64 medidas
- Cada gesto se clasifica en:
  - 0: Piedra
  - 1: Tijeras
  - 2: Papel
  - 3: Okay
- Para la regresión logística y red neuronal se escalan

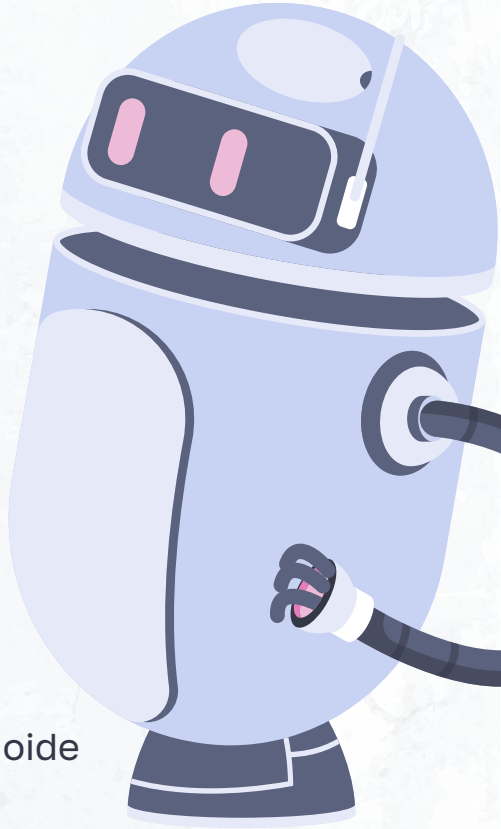


2 Regresión logística

P(piedra) = 0.61

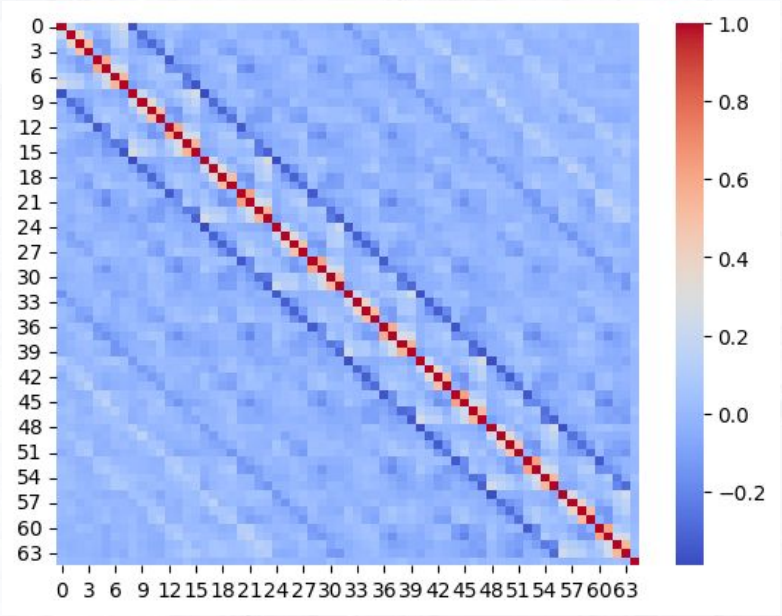
Usaremos la información de piedra y de tijeras:





La regresión logística distinguirá entre ambas aplicando la función sigmoide

2 Regresión logística



Matriz de correlación:

- Máxima correlación (entre variables distintas) de 0,6 en menos de 10 casos (de 64x64)
- Baja correlación lineal
- Parecen presentar una relación no lineal

Adaptación:

- Mapeo de los atributos a un grado 2
- De 64 variables → 2144

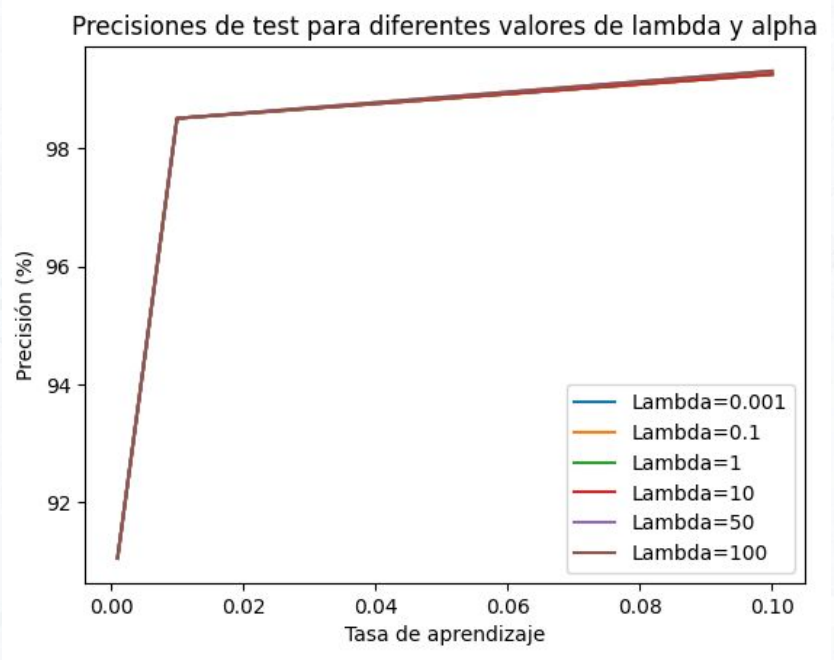
2 Regresión logística

Búsqueda de hiperparámetros:

- Lambda: tasa de regularización L2 → controlar el overfitting
- Alfa: tasa de aprendizaje → muy pequeña lleva a mayores tiempos, muy alta puede no encontrar la solución

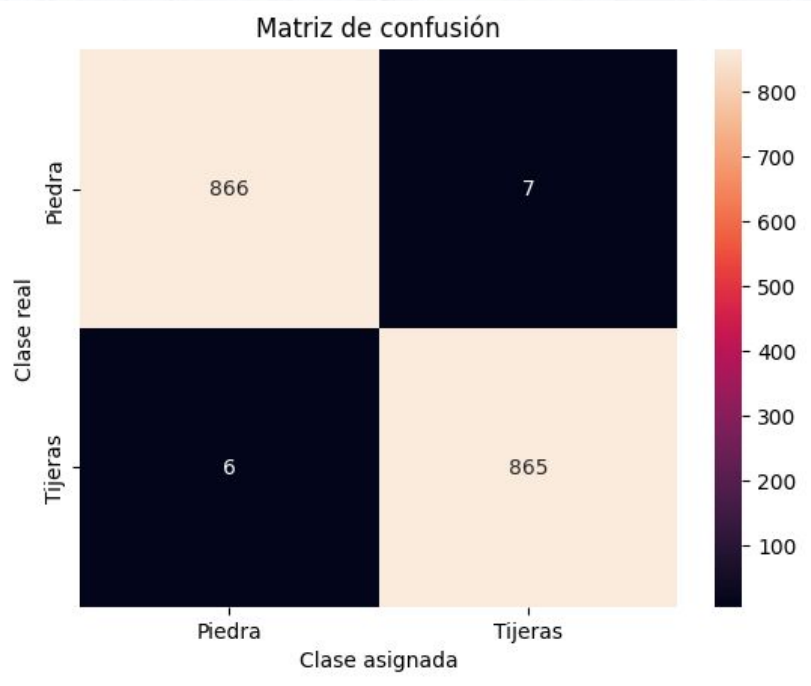
Los mejores resultados para:

- Lambda: 0 (vemos que no afecta los resultados del modelo)
- Alfa: 0.1



2 Regresión logística

Resultados:

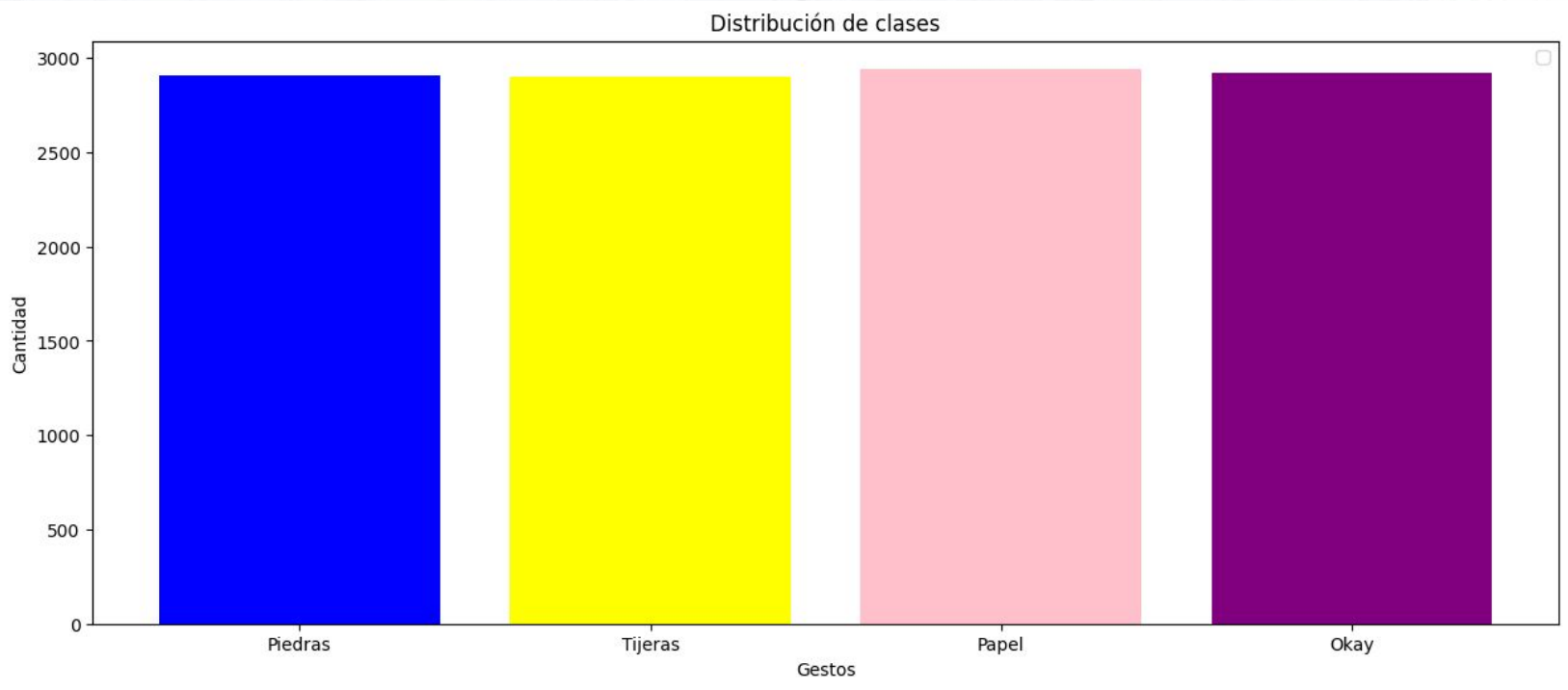


	Piedra	Tijeras	Exactitud
Precisión	0.99	0.99	
Recall	0.99	0.99	
F1-score	0.99	0.99	0.99
Instancias	873	871	1744

Precisión de
- Entrenamiento: 99.66%
- Test: 99.26%

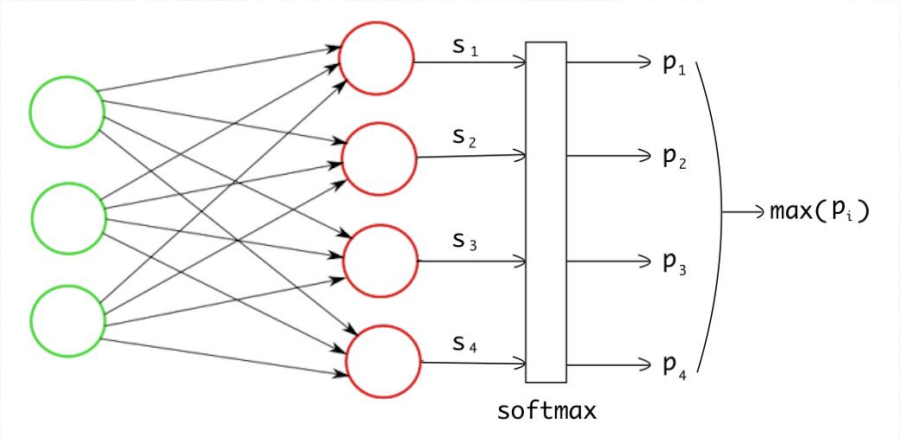
3 Red neuronal multiclase

Usaremos la información de las cuatro clases piedra, tijeras, papel y okay:



3 Red neuronal multiclase

- Función de activación **softmax** en la capa de salida
- Función de activación **ReLU** en las capas ocultas
- Optimizador **Adam**: eficiente computacionalmente y capacidad para manejar muchos datos
- Función de coste: **entropía cruzada categórica**
- División 60% entrenamiento (7006), 20% validación (2336), 20% test(2336).



3 Red neuronal multiclase

Búsqueda de hiperparámetros:

Tasa de aprendizaje tasa de aprendizaje Adam
Número de neuronas en la capa oculta
Número de iteraciones número de entrenamientos
Regularización parámetro de la regularización L2
Tamaño del batch número de muestras usadas para calcular el error y actualizar los pesos
Número de capas ocultas

Algoritmo minimización de la librería Hyperopt

3 Red neuronal multiclase

Algoritmo minimización de la librería Hyperopt

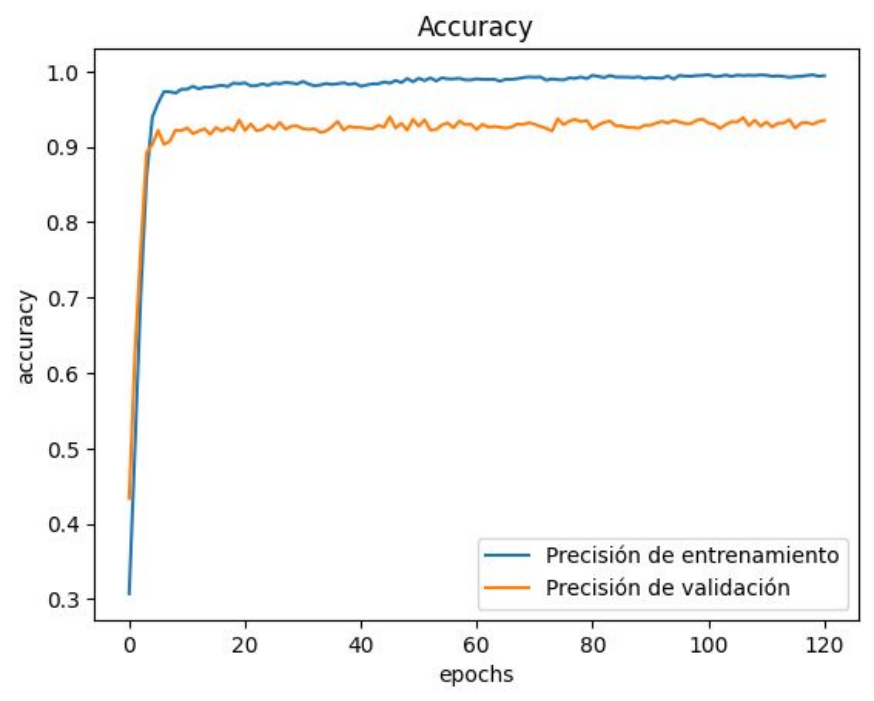
Con el algoritmo TPE (Tree-based Parzen Estimators):
Busca el siguiente parámetro en base a lo que “promete” → (Probabilidad (buena opción) / Probabilidad (mala opción))

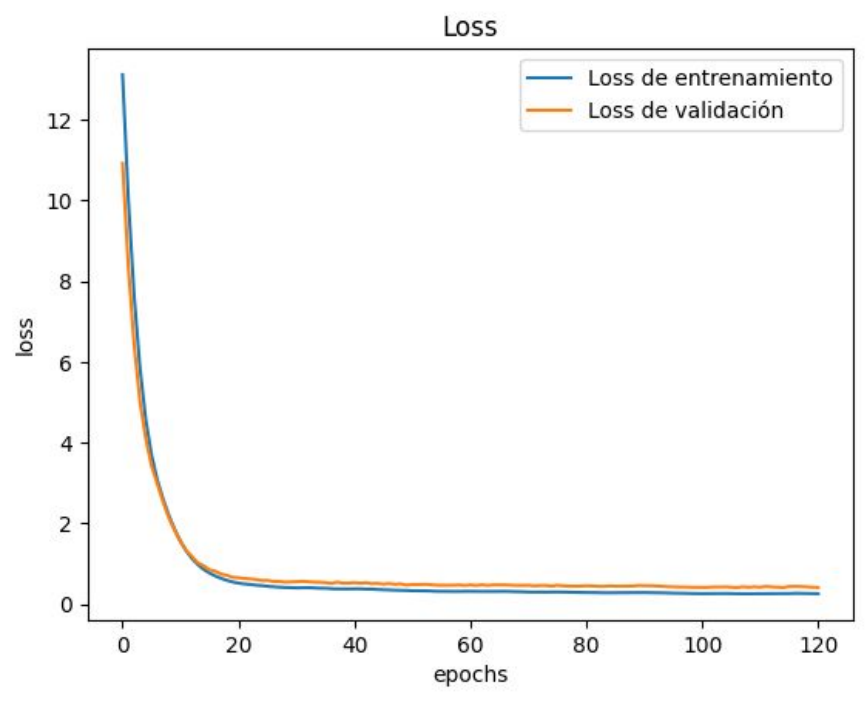
1. Se define **función a minimizar** (- precisión, en este caso, para maximizarla)
2. Se define un **espacio de búsqueda** para cada variable
3. Se ejecuta el algoritmo de búsqueda de parámetros que **maximiza la precisión**

3 Red neuronal multiclase

Tasa de aprendizaje 0.002 **Número de neuronas** 120 **Número de iteraciones** 40

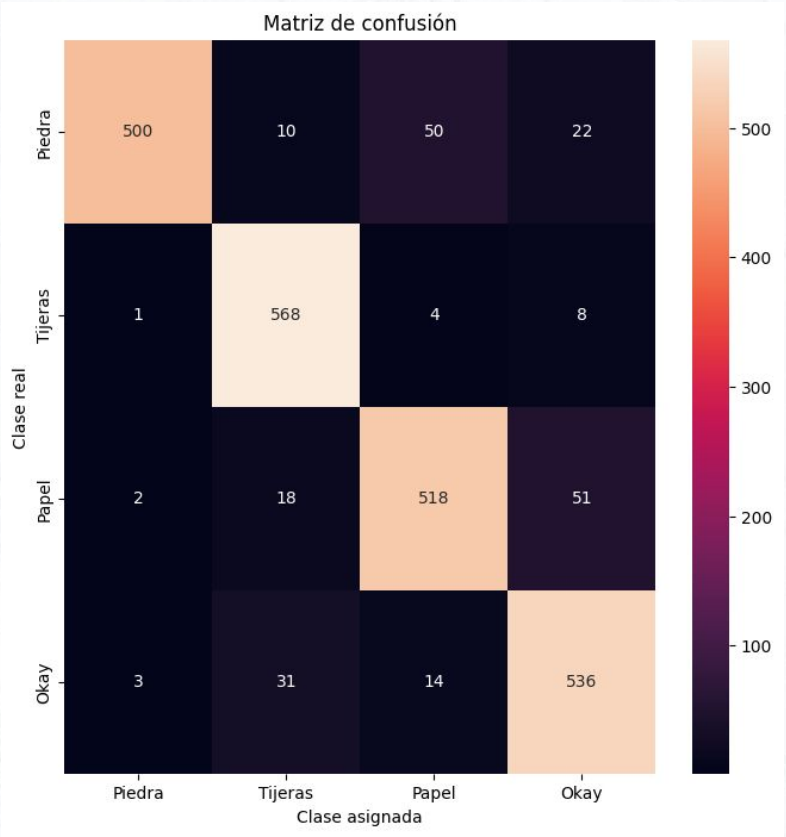
Regularización 0.02 **Tamaño del batch** 2040 **Número de capas ocultas** 3





3 Red neuronal multiclase

Resultados:



	Piedra	Tijeras	Papel	Okay	Exactitud
Precisión	0.96	0.90	0.95	0.86	
Recall	0.93	0.96	0.86	0.91	
F1-score	0.94	0.93	0.90	0.89	0.99
Instancias	582	581	589	584	1744

Precisión de

- Entrenamiento: 98%
- Test: 92%
- Validación 94%

4 Árboles de decisión

Con scikit-learn

64 columnas:

S1 S2 S3 S4 S5 S6 S7 S8 ... S1 S2 S3 S4 S5 S6 S7 S8

8 veces

8 columnas

media(S1) media(S2) media(S3) media(S4) media(S5)...

Con XGBoost (eXtreme Gradient Boosting)

Especialmente útil con muchos datos y muchas columnas

Muy eficiente computacionalmente

Ayuda a prevenir el overfitting

Usaremos las 64 columnas

Compararemos los resultados de ambos

4 Árbol de decisión con scikit-learn

Búsqueda de hiperparámetros:

Profundidad máxima del árbol

Mejor resultado: 8

Número mínimo de muestras para dividir un nodo

Mejor resultado: 30%

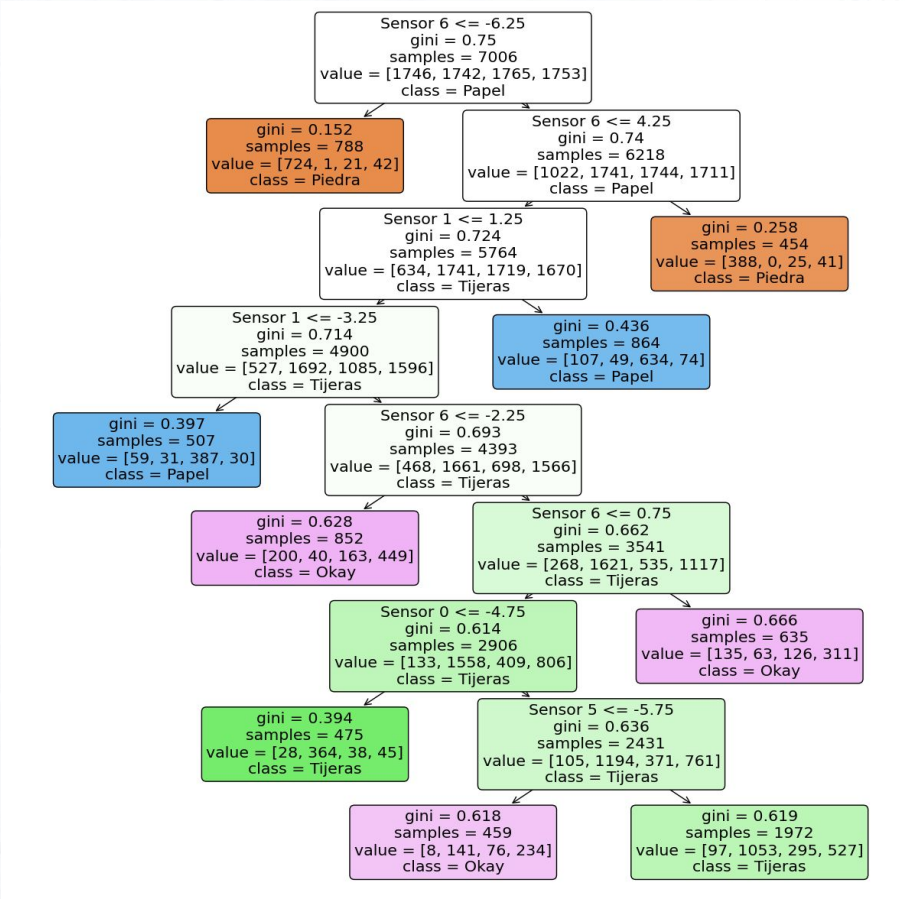
Criterio Gini o entropía

Mejor resultado Gini

4 Árbol de decisión con scikit-learn

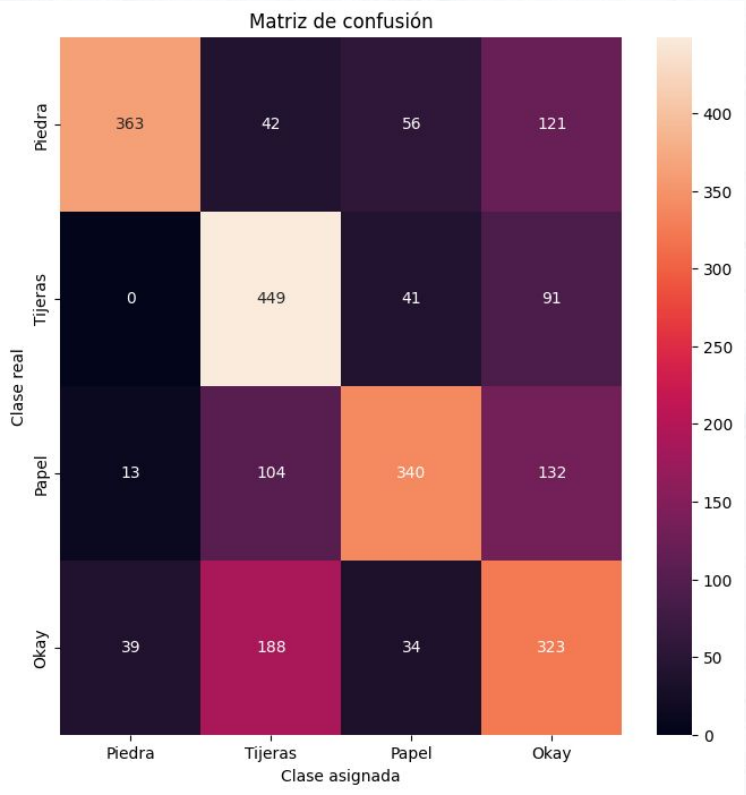
Resultados:

- Importancia de la media de las medidas del **sensor 6** y del **sensor 1**
- No consigue ningún nodo puro (todos gini > 0.0)
- Los que más difícilmente se distinguen son Tijeras y Okay



4 Árbol de decisión con scikit-learn

Resultados:



	Piedra	Tijeras	Papel	Okay	Exactitud
Precisión	0.87	0.57	0.72	0.48	
Recall	0.62	0.77	0.58	0.55	
F1-score	0.73	0.66	0.64	0.52	0.63
Instancias	582	581	589	584	2336

Precisión de

- Entrenamiento: 64.85%
- Test: 63.14%
- Validación 63.10%

4 Árbol de decisión con XGBoost

Eta análogo a tasa de aprendizaje

Mejor resultado: 0.45

Gamma reducción mín. de la función de coste para crear un nuevo nodo

Mejor resultado: 1.62

Máxima profundidad del árbol

Mejor resultado: 16

Número máximo de características a considerar en cada árbol

Mejor resultado: 77%

Lambda penalización L2 en los pesos de las hojas

Mejor resultado: 0.93

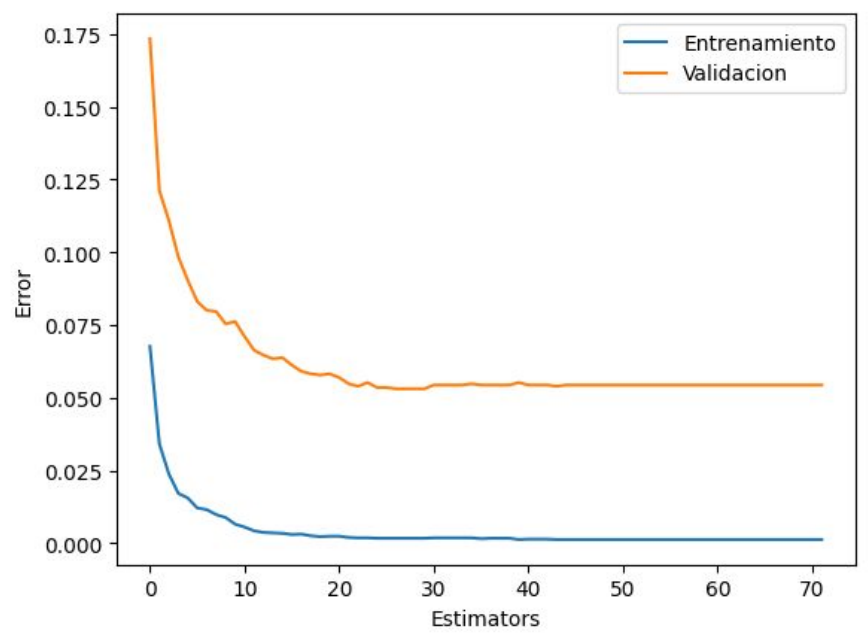
Mínimo peso de la hoja para crear una nueva hoja

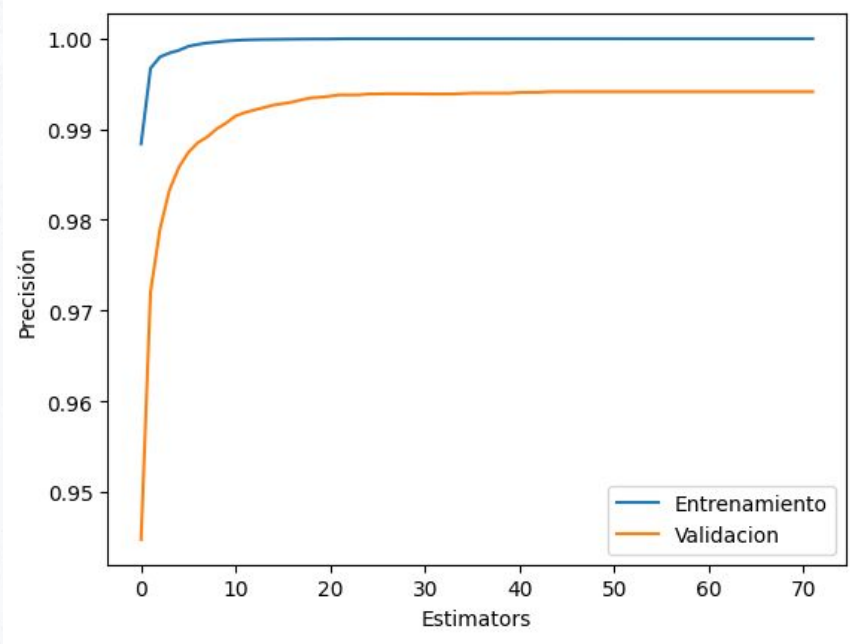
Mejor resultado: 2

4 Árbol de decisión con XGBoost

Número de estimadores número de árboles a crear

Mejor resultado: 52

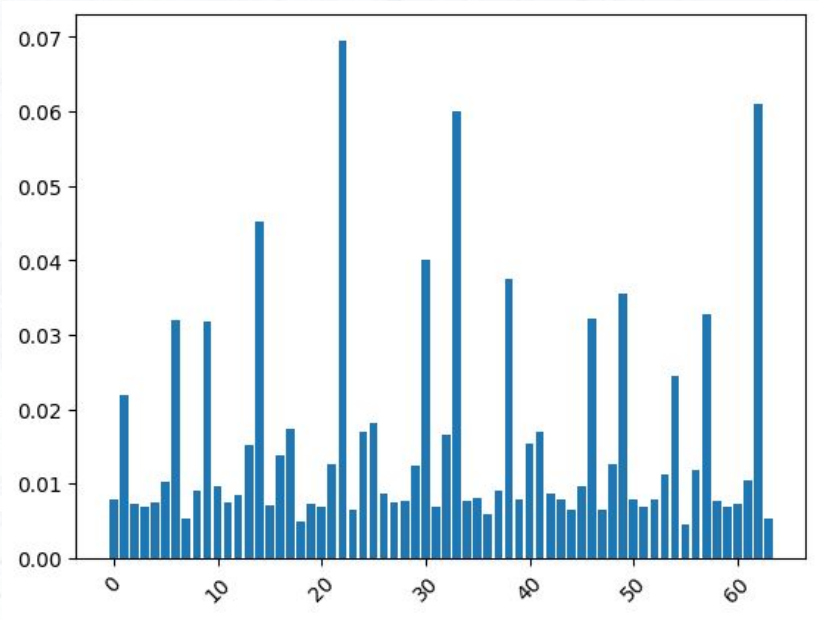




4 Árbol de decisión con XGBoost

Árbol final:

- Función de coste a optimizar: **multi softmax**
- Métrica de evaluación: “**merror**“ → Fallos / Casos totales

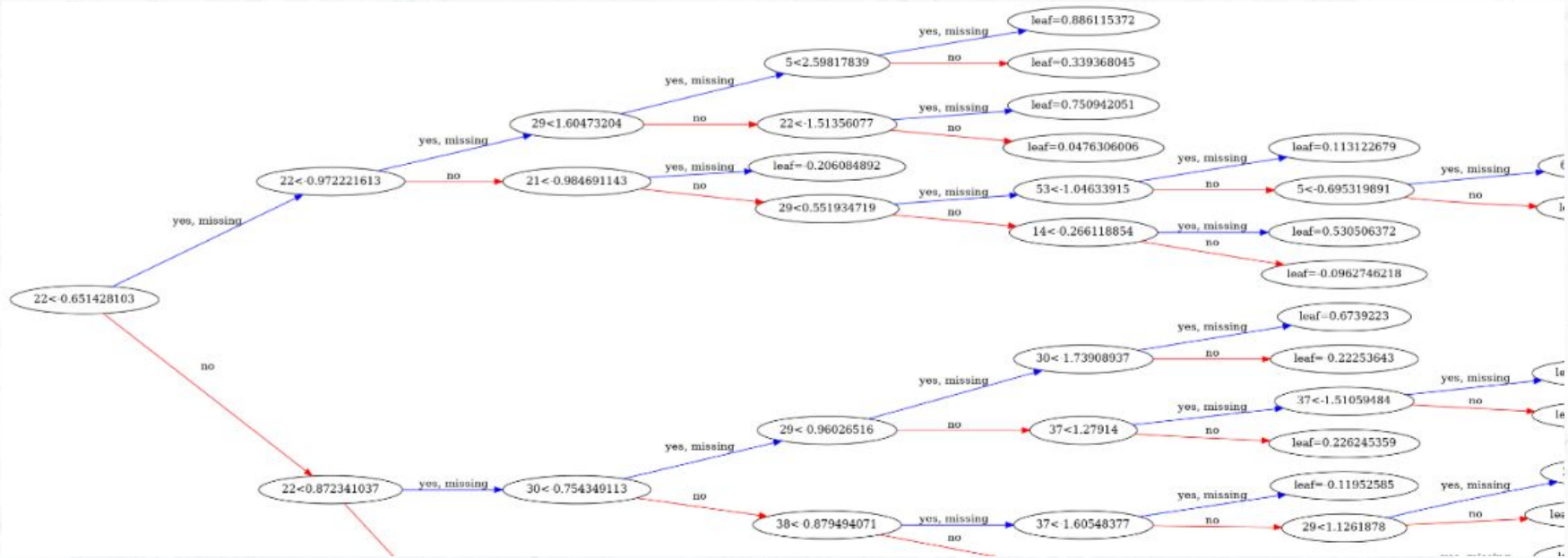


- Algunas más importantes que otras pero en un grado no demasiado grande
- Ninguna irrelevante → No podemos eliminar columnas

4 Árbol de decisión con XGBoost

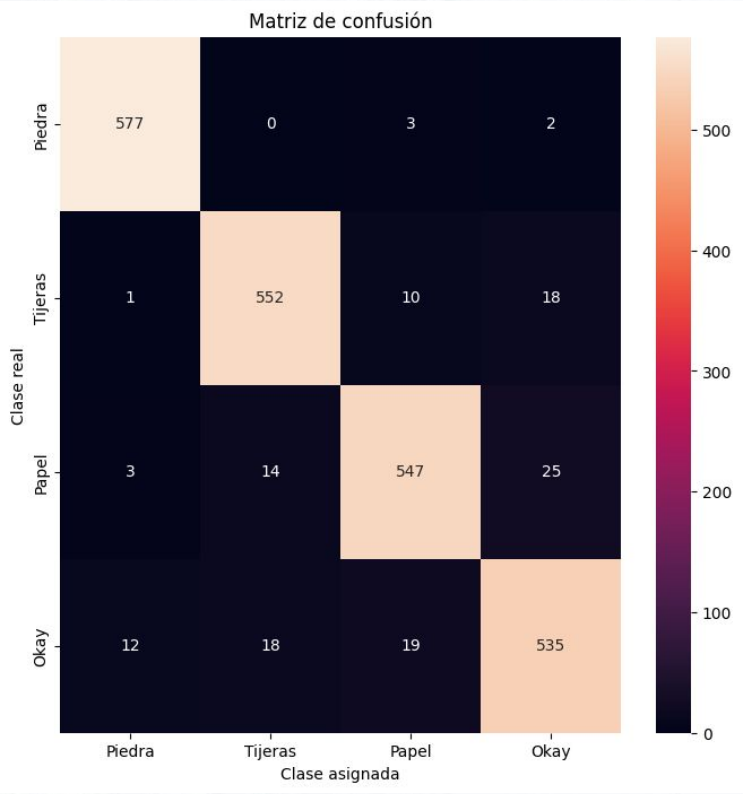
Árbol final:

- Función de coste a optimizar: **multi softmax**
- Métrica de evaluación: “**merror**“ → Fallos / Casos totales



4 Árbol de decisión con XGBoost

Resultados:



	Piedra	Tijeras	Papel	Okay	Exactitud
Precisión	0.97	0.95	0.94	0.92	
Recall	0.99	0.95	0.93	0.92	
F1-score	0.98	0.95	0.94	0.92	0.95
Instancias	582	581	589	584	2336

Precisión de

- Entrenamiento: 99.70%
- Test: 94.65%
- Validación 94.60%

5 Comparación de resultados

Regresión logística

- Distingue perfectamente Piedra de Tijeras
- Mejores resultados al mapear → no relación lineal
- 99% exactitud

Red neuronal

- Distingue perfectamente Piedra
- La que peor distingue es Okay que confunde con tijeras
- 92% exactitud (peor que XGBoost)

Árbol scikit-learn

- Peor resultados obtenidos
- 8 variables, a diferencia del resto
- 63% exactitud

Árbol XGBoost

- Mejor resultados obtenidos
- Mejor en capturar relaciones no lineales y complejas
- Distingue perfectamente Piedra
- La que peor distingue es Okay que confunde con tijeras
- 95% exactitud

5 Comparación de resultados

1. Mejor rendimiento

4 clases → XGBoost con 95% de exactitud

2 clases → Regresión logística 99% de exactitud

2. Capacidad generalización

Alta para XGBoost, regresión logística y red neuronal.

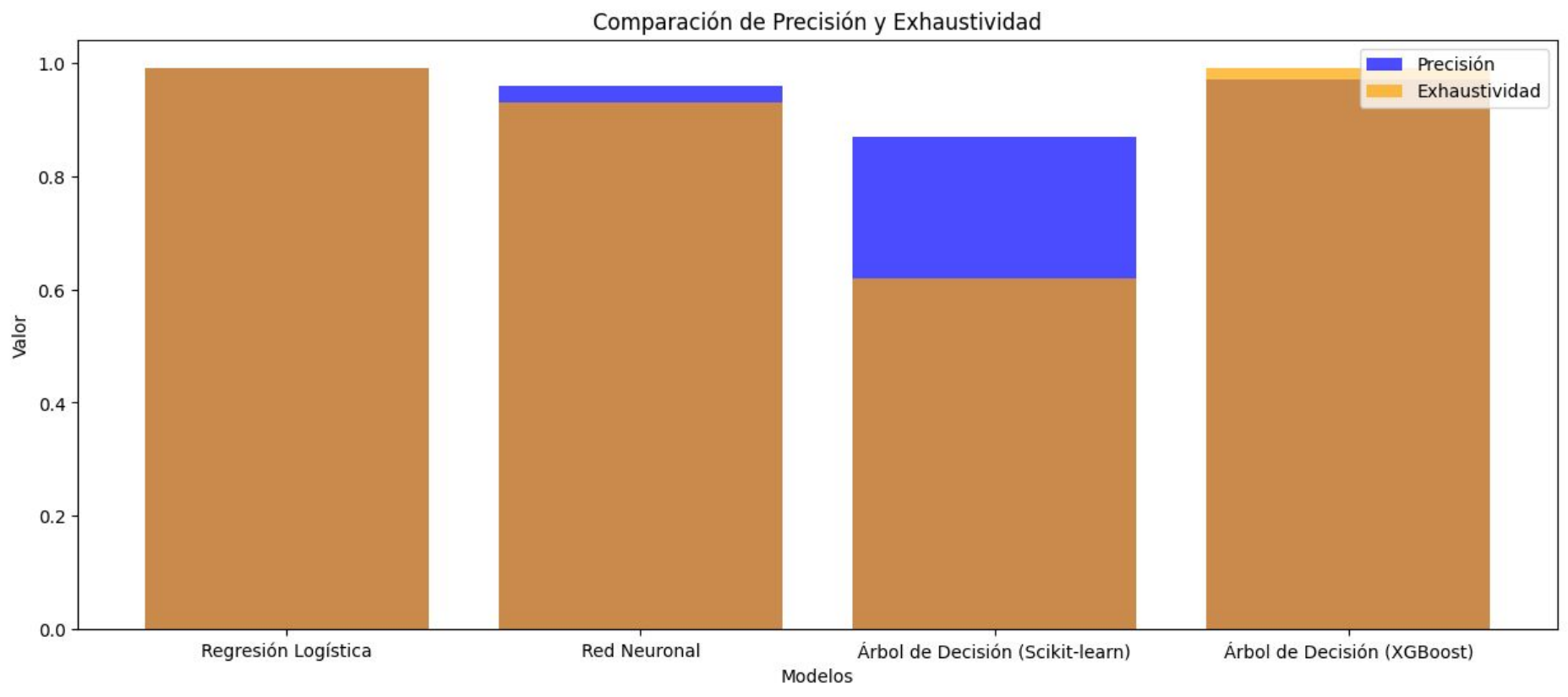
3. Tiempo de entrenamiento

- 15s para la regresión logística
- 5 min. para la red neuronal
- 2s para el árbol de Scikit-learn
- 3 min para el árbol de XGBoost

4. Interpretabilidad

Normalmente los árboles, al haber tantas variables → se vuelve más complicado

5 Comparación de resultados



6 Bibliografía

- Ng, A. “C1_W3”, "C2_W1 NN.", "C2_W2 NN training + multiclass." , "C2_W3 evaluation.", "C2_W4 Decision Trees."Stanford. Disponible en el Campus Virtual.

- Amat Rodrigo, J. (2020) Regresión Lineal con python, Regresión lineal con python. Cienciadedatos.net. Disponible en: https://www.cienciadedatos.net/documentos/py10-regresion-lineal-python.html (Visitado: 28 de abril, 2023).

- Amat Rodrigo, J. (2020) Regresión Lineal con python, Regresión lineal con python. Cienciadedatos.net. Disponible en: https://www.cienciadedatos.net/documentos/py10-regresion-lineal-python.html (Visitado: 28 de abril, 2023).

- Redacción KeepCoding (2022) Función Softmax en tensorflow, Función softmax en TensorFlow . KeepCoding Bootcamps. Disponible en: https://keepcoding.io/blog/funcion-softmax-tensorflow/#Funcion_softmax_en_TensorFlow (Visitado: 1 de mayo, 2023).

- Brownlee, J. (2021) Gentle introduction to the adam optimization algorithm for deep learning, MachineLearningMastery.com. machinelearningmastery.com. Disponible en: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (Visitado: 1 de mayo, 2023).

- Rendyk (2022) Tuning the hyperparameters and layers of neural network deep learning, Analytics Vidhya. Data Science Blogathon. Disponible en: https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/ (Visitado: 7 de mayo, 2023).

- Natsume, Y. (2022) Bayesian optimization with python, Medium. Towards Data Science. Disponible en: https://towardsdatascience.com/bayesian-optimization-with-python-85c66df711ec (Visitado: 7 de mayo, 2023).

- Banerjee, P. (2020) A Guide on XGBoost hyperparameters tuning, Kaggle. Disponible en: https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning (Visitado: 14 de mayo, 2023).