Práctica 2.1: Problema de clasificación desbalanceado PM

Redes de Neuronas Artificiales

Diciembre 2023

Grado en Ingeniería Informática

Pablo Hidalgo Delgado. NIA: 100451225
Marcos Caballero Cortés. NIA: 100451047

ÍNDICE

1. Introducción

El propósito de esta práctica es abordar un problema de clasificación clásico haciendo uso del Perceptrón Multicapa (PM). Para ello, se nos dispone de un conjunto de datos desbalanceado sobre las trayectorias de diferentes individuos en diferentes medios de transporte. Utilizando estos datos, debemos generar un modelo capaz de clasificar el medio de transporte en el que se desplaza un individuo.

2. Análisis exploratorio de datos (EDA)

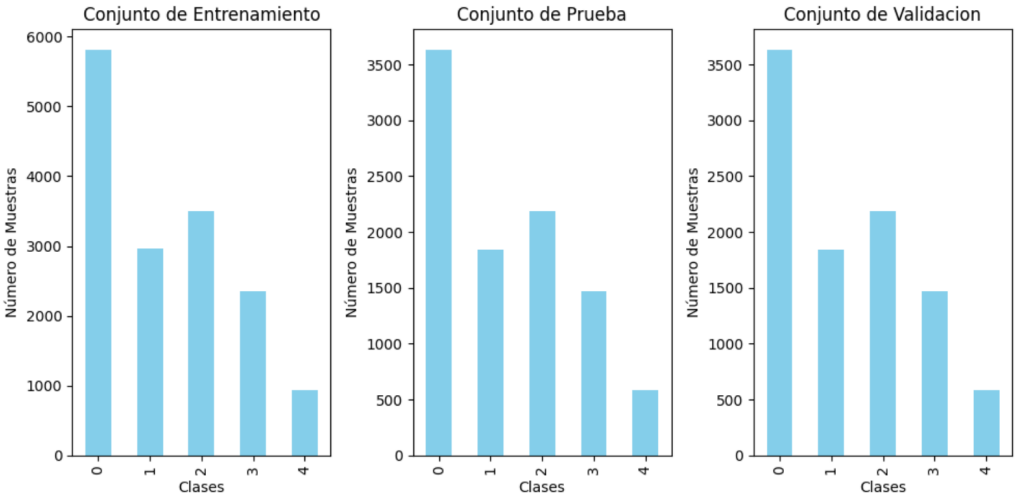
Primero, realizamos un breve Análisis Exploratorio de Datos (EDA) para resumir las características clave de nuestro conjunto de datos. Esto nos permite comprender mejor la naturaleza de los datos y optimizar la construcción de nuestro modelo. A través de este análisis, identificamos que el conjunto de datos consta de 29151 instancias y 22 atributos de tipo float64, sin valores nulos ni atributos constantes. También, hemos reafirmado el hecho de que estamos ante un conjunto de datos desbalanceado. Además, hemos identificado 2 columnas irrelevantes (identificadores) como son el user_id y track_id, las cuales decidimos eliminar para evitar que el modelo aprenda estos datos ya que podrían producir un sesgo en el modelo.

3. Preproceso

Previamente a entrenar los modelos, debemos realizar una transformación de los datos para que puedan ser interpretados por los algoritmos de manera eficiente. Las transformaciones de datos que realizamos son las siguientes:

3.1 División en datos de train, validation y test

Dividimos el conjunto de datos en train (2/3), y test (1/3). Posteriormente, dividimos los datos de entrenamiento en train train (80%) y train validation (20%), tal y como se especifica en el enunciado de la práctica. Esto lo realizamos mediante la función train_test_split() de la librería de scikit-learn. Además, al tratarse de un problema desbalanceado, utilizamos el parámetro stratify con el objetivo de realizar una partición estratificada, es decir, mantener la misma proporción de clases en ambos conjuntos. Así, la distribución de clases en cada conjunto es la siguiente:



3.2 Aleatorización

Aleatorizamos nuestros datos para evitar que el modelo aprenda patrones relacionados con la ordenación de los datos que nosotros no somos capaces de identificar pero la red neuronal sí, lo que afectaría negativamente a la capacidad de generalización del modelo.
La aleatorización la realizamos a la vez que la división de datos especificando el parámetro shuffle = True en la función train_test_split().

3.3 Codificación de la variable de salida

Aunque nuestra variable de salida ya está representada por valores numéricos, es crucial codificarla de tal manera que se evite cualquier suposición de relaciones ordinales entre los valores de diferentes clases. La razón es que, de lo contrario, el modelo podría interpretar e incluso aprender relaciones de orden entre las distintas clases, lo cual introduciría un sesgo en el modelo que no refleja la realidad del problema. Por tanto, codificamos nuestra variable de salida utilizando la clase LabelBinarizer(). De esta manera, contaremos con una columna binaria para cada clase (un total de 5 columnas).

3.4 Normalización

Algunos algoritmos de aprendizaje automático son sensibles a las diferencias en escala de los atributos y pueden dar resultados menos precisos por esta razón. En consecuencia, para evitar estas diferencias de escala, normalizamos nuestros datos haciendo uso de la fórmula Min-Max.
Cabe destacar que los valores máximos y mínimos utilizados en la normalización, se escogen exclusivamente a partir de los datos de entrenamiento. El propósito de esta práctica es evitar cualquier tipo de information leakage que se produzca al dar información al modelo sobre datos no utilizados para entrenar.

4. Experimentos realizados

Hemos realizado un gran número de experimentos iniciales utilizando el Perceptrón Multicapa, en los cuales hemos variado los valores de sus hiperparámetros (tasa de aprendizaje, función de activación, número de capas ocultas y número de neuronas en cada capa oculta). Los valores que hemos seleccionado son los siguientes: tasas de aprendizaje de [0.2, 0.1, 0.05], funciones de activación ReLU, sigmoide y softmax, y capas ocultas distribuidas de la siguiente manera (la clave hace referencia al número de capas ocultas y el valor al número de neuronas en cada capa oculta):

{ 1: [300],
1: [200],
1: [300],
2: [60,30],
2: [200,75],
2: [300,100],
3: [50,25,15],
3: [200,100,20],
3: [300,150,50] }

Todos los modelos tienen una capa de salida con 5 neuronas (5 clases) y con función de activación softmax, por lo que la salida indicará la probabilidad de pertenecer a cada clase.

Como resultado, para cada valor de la tasa de aprendizaje, hemos construido 27 modelos distintos con diferente distribución de las neuronas, dando lugar a un total de 81 modelos construidos. Cada modelo ha sido evaluado con el MSE en entrenamiento, validación y test. Esta variedad de experimentos nos permite examinar en detalle las diferencias en el rendimiento de cada modelo en función de los valores introducidos de sus hiperparámetros.

Es importante señalar que en todos estos experimentos empleamos un mismo número de épocas, 300. No obstante, se implementó la técnica de "Early Stopping" con un umbral de paciencia establecido en 20 épocas. Además, configuramos el parámetro restore_best_weights = True, permitiendo que el modelo final devuelto sea aquel que ha exhibido el menor error de validación durante el proceso de entrenamiento.

A partir de los resultados de estos experimentos, escogemos los 3 con menor error de validación. Entrenamos estos modelos utilizando un mayor número de épocas y una paciencia considerablemente alta (50 épocas) para asegurar la obtención del mejor rendimiento posible. También, en este caso, evaluamos estos modelos con métricas propias y adecuadas para los problemas desbalanceados como el accuracy, recall, precision y f1-score.

Posteriormente, entrenamos estos 3 modelos incluyendo balanceo de clases. Para ello, introducimos pesos de manera que sean inversamente proporcionales a la frecuencia de cada clase. Así, se dará más importancia a las clases minoritarias y menos a las mayoritarias. Estos valores de los pesos se han calculado automáticamente por medio de la función compute_class_weights.

5. Resultados obtenidos

pruebas

LR	Hidden Layers	Neurons	Activation Function	Epochs	Best Epoch	Tiempo Entrenamiento	Train MSE	Validation MSE	Test MSE
0,2	1	[100]	relu	300	298	312,022	4,66E-02	5,02E-02	5,05E-02
0,2	1	[100]	sigmoid	300	298	320,329	6,6E-02	6,72E-02	6,7E-02
0,2	1	[100]	softmax	300	300	304,283	8,36E-02	8,4E-02	8,37E-02
0,2	2	[60, 30]	relu	300	166	186,287	4,48E-02	4,94E-02	4,89E-02
0,2	2	[60, 30]	sigmoid	300	298	316,231	5,91E-02	6,03E-02	5,96E-02
0,2	2	[60, 30]	softmax	300	297	322,501	1,49E-01	1,49E-01	1,49E-01
0,2	3	[50, 25, 15]	relu	300	85	112,115	4,73E-02	5,01E-02	5,01E-02
0,2	3	[50, 25, 15]	sigmoid	300	298	324,438	6,3E-02	6,38E-02	6,29E-02
0,2	3	[50, 25, 15]	softmax	300	38	65,078	1,49E-01	1,49E-01	1,49E-01
0,2	1	[200]	relu	300	298	304,902	4,58E-02	4,93E-02	5E-02
0,2	1	[200]	sigmoid	300	295	322,422	6,62E-02	6,73E-02	6,71E-02
0,2	1	[200]	softmax	300	299	315,019	1,02E-01	1,02E-01	1,02E-01
0,2	2	[200, 75]	relu	300	148	190,375	4,22E-02	4,73E-02	4,74E-02
0,2	2	[200, 75]	sigmoid	300	295	338,092	6,13E-02	6,22E-02	6,16E-02
0,2	2	[200, 75]	softmax	300	38	67,583	1,49E-01	1,49E-01	1,49E-01
0,2	3	[200, 100, 20]	relu	300	130	176,896	3,91E-02	4,61E-02	4,66E-02
0,2	3	[200, 100, 20]	sigmoid	300	287	350,527	6,36E-02	6,39E-02	6,38E-02
0,2	3	[200, 100, 20]	softmax	300	38	70,005	1,49E-01	1,49E-01	1,49E-01
0,2	1	[300]	relu	300	298	322,409	4,57E-02	4,97E-02	5,01E-02
0,2	1	[300]	sigmoid	300	295	318,101	6,65E-02	6,74E-02	6,74E-02
0,2	1	[300]	softmax	300	300	329,227	1,07E-01	1,07E-01	1,07E-01
0,2	2	[300, 100]	relu	300	91	128,863	4,57E-02	4,97E-02	4,96E-02
0,2	2	[300, 100]	sigmoid	300	295	354,838	6,15E-02	6,23E-02	6,15E-02
0,2	2	[300, 100]	softmax	300	38	71,001	1,49E-01	1,49E-01	1,49E-01
0,2	3	[300, 150, 50]	relu	300	130	209,285	3,9E-02	4,56E-02	4,64E-02
0,2	3	[300, 150, 50]	sigmoid	300	287	414,388	6,43E-02	6,44E-02	6,42E-02
0,2	3	[300, 150, 50]	softmax	300	38	87,122	1,49E-01	1,49E-01	1,49E-01
0,1	1	[100]	relu	300	298	322,422	5,24E-02	5,47E-02	5,45E-02
0,1	1	[100]	sigmoid	300	300	297,312	7,12E-02	7,16E-02	7,18E-02
0,1	1	[100]	softmax	300	299	310,353	1,02E-01	1,02E-01	1,02E-01
0,1	2	[60, 30]	relu	300	217	249,425	4,62E-02	4,96E-02	4,98E-02
0,1	2	[60, 30]	sigmoid	300	298	322,496	7,08E-02	7,15E-02	7,15E-02
0,1	2	[60, 30]	softmax	300	289	322,495	1,49E-01	1,49E-01	1,49E-01
0,1	3	[50, 25, 15]	relu	300	84	109,896	5E-02	5,25E-02	5,15E-02
0,1	3	[50, 25, 15]	sigmoid	300	300	322,590	7,42E-02	7,47E-02	7,45E-02
0,1	3	[50, 25, 15]	softmax	300	54	83,006	1,49E-01	1,49E-01	1,49E-01
0,1	1	[200]	relu	300	297	301,873	5,17E-02	5,4E-02	5,41E-02
0,1	1	[200]	sigmoid	300	298	322,420	7,16E-02	7,19E-02	7,21E-02
0,1	1	[200]	softmax	300	300	322,418	1,19E-01	1,19E-01	1,19E-01
0,1	2	[200, 75]	relu	300	217	264,749	4,32E-02	4,84E-02	4,85E-02

(continuación)

0,1	2	[200, 75]	sigmoid	300	298	346,885	6,96E-02	7,01E-02	7,05E-02
0,1	2	[200, 75]	softmax	300	54	84,303	1,49E-01	1,49E-01	1,49E-01
0,1	3	[200, 100, 20]	relu	300	139	179,299	4,34E-02	4,73E-02	4,72E-02
0,1	3	[200, 100, 20]	sigmoid	300	293	382,582	7,29E-02	7,31E-02	7,34E-02
0,1	3	[200, 100, 20]	softmax	300	65	99,041	1,49E-01	1,49E-01	1,49E-01
0,1	1	[300]	relu	300	298	303,984	5,12E-02	5,4E-02	5,38E-02
0,1	1	[300]	sigmoid	300	298	306,048	7,14E-02	7,17E-02	7,19E-02
0,1	1	[300]	softmax	300	300	307,940	1,49E-01	1,49E-01	1,49E-01
0,1	2	[300, 100]	relu	300	202	245,390	4,36E-02	4,85E-02	4,84E-02
0,1	2	[300, 100]	sigmoid	300	298	382,782	7,16E-02	7,17E-02	7,24E-02
0,1	2	[300, 100]	softmax	300	56	86,142	1,49E-01	1,49E-01	1,49E-01
0,1	3	[300, 150, 50]	relu	300	139	202,130	4,35E-02	4,85E-02	4,79E-02
0,1	3	[300, 150, 50]	sigmoid	300	289	389,008	7,38E-02	7,37E-02	7,46E-02
0,1	3	[300, 150, 50]	softmax	300	56	104,158	1,49E-01	1,49E-01	1,49E-01
0,05	1	[100]	relu	300	300	323,101	5,94E-02	6,13E-02	6,13E-02
0,05	1	[100]	sigmoid	300	300	295,605	7,86E-02	7,86E-02	7,89E-02
0,05	1	[100]	softmax	300	300	298,101	1,17E-01	1,17E-01	1,17E-01
0,05	2	[60, 30]	relu	300	292	302,236	4,83E-02	5,11E-02	5,12E-02
0,05	2	[60, 30]	sigmoid	300	300	307,165	8,01E-02	8,03E-02	8,06E-02
0,05	2	[60, 30]	softmax	300	298	322,514	1,49E-01	1,49E-01	1,49E-01
0,05	3	[50, 25, 15]	relu	300	225	251,512	4,74E-02	5,16E-02	5,11E-02
0,05	3	[50, 25, 15]	sigmoid	300	299	322,562	9,83E-02	9,83E-02	9,81E-02
0,05	3	[50, 25, 15]	softmax	300	116	146,044	1,49E-01	1,49E-01	1,49E-01
0,05	1	[200]	relu	300	300	322,412	5,88E-02	6,06E-02	6,05E-02
0,05	1	[200]	sigmoid	300	300	322,408	7,85E-02	7,86E-02	7,88E-02
0,05	1	[200]	softmax	300	300	302,146	1,49E-01	1,49E-01	1,49E-01
0,05	2	[200, 75]	relu	300	217	254,547	4,78E-02	5,11E-02	5,13E-02
0,05	2	[200, 75]	sigmoid	300	298	322,526	7,92E-02	7,93E-02	7,96E-02
0,05	2	[200, 75]	softmax	300	92	122,608	1,49E-01	1,49E-01	1,49E-01
0,05	3	[200, 100, 20]	relu	300	225	269,666	4,37E-02	4,84E-02	4,87E-02
0,05	3	[200, 100, 20]	sigmoid	300	300	382,550	9,8E-02	9,79E-02	9,78E-02
0,05	3	[200, 100, 20]	softmax	300	95	130,493	1,49E-01	1,49E-01	1,49E-01
0,05	1	[300]	relu	300	300	296,297	5,91E-02	6,09E-02	6,08E-02
0,05	1	[300]	sigmoid	300	300	322,410	7,83E-02	7,82E-02	7,86E-02
0,05	1	[300]	softmax	300	300	323,255	1,49E-01	1,49E-01	1,49E-01
0,05	2	[300, 100]	relu	300	298	382,538	4,59E-02	4,96E-02	4,96E-02
0,05	2	[300, 100]	sigmoid	300	298	382,527	7,89E-02	7,9E-02	7,93E-02
0,05	2	[300, 100]	softmax	300	100	138,326	1,49E-01	1,49E-01	1,49E-01
0,05	3	[300, 150, 50]	relu	300	225	314,267	4,23E-02	4,74E-02	4,73E-02
0,05	3	[300, 150, 50]	sigmoid	300	300	442,845	8,88E-02	8,84E-02	8,91E-02
0,05	3	[300, 150, 50]	softmax	300	80	134,593	1,49E-01	1,49E-01	1,49E-01

Obtenemos los 3 modelos con menor error de validación.

mejores_modelos

LR	Neurons	Activation Function	Epochs	Best Epoch	Tiempo	Train MSE	Validation MSE	Test MSE	Accuracy	Recall	Precision	F1_score
0,2	[300, 150, 50]	relu	500	225	377	2,95E-02	4,35E-02	4,36E-02	0,858	[0,937, 0,776, 0,867, 0,778, 0,801]	[0,854, 0,875, 0,868, 0,868, 0,785]	[0,894, 0,823, 0,867, 0,821, 0,793]
0,2	[200, 100, 20]	relu	500	258	348	2,9E-02	4,41E-02	4,52E-02	0,852	[0,929, 0,762, 0,885, 0,728, 0,852]	[0,858, 0,876, 0,844, 0,898, 0,716]	[0,892, 0,815, 0,864, 0,804, 0,778]
0,1	[200, 100, 20]	relu	500	312	403	3,4E-02	4,48E-02	4,46E-02	0,853	[0,934, 0,75, 0,854, 0,78, 0,851]	[0,848, 0,874, 0,87, 0,861, 0,758]	[0,889, 0,808, 0,861, 0,818, 0,802]

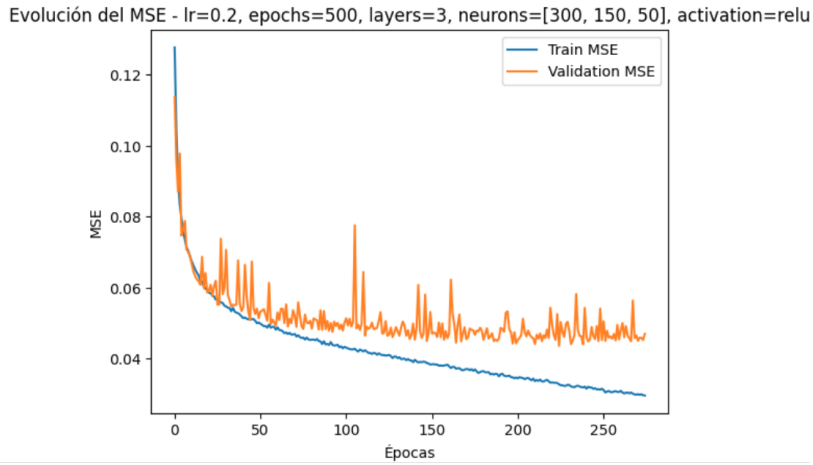
Incluimos balanceo de clases para estos modelos.

balanceo_mejores_modelos

LR	Neurons	Activation Function	Epochs	Best Epoch	Tiempo	Train MSE	Validation MSE	Test MSE	Accuracy	Recall	Precision	F1_score
0,2	[300, 150, 50]	relu	500	188	324	2,97E-02	4,45E-02	4,49E-02	0,853	[0,915, 0,789, 0,848, 0,785, 0,866]	[0,863, 0,84, 0,876, 0,86, 0,751]	[0,888, 0,814, 0,861, 0,821, 0,805]
0,2	[200, 100, 20]	relu	500	225	320	2,88E-02	4,5E-02	4,54E-02	0,852	[0,914, 0,786, 0,828, 0,804, 0,89]	[0,868, 0,847, 0,889, 0,825, 0,731]	[0,89, 0,815, 0,857, 0,814, 0,803]
0,1	[200, 100, 20]	relu	500	225	321	3,69E-02	4,57E-02	4,68E-02	0,847	[0,903, 0,785, 0,847, 0,77, 0,894]	[0,874, 0,844, 0,86, 0,841, 0,696]	[0,889, 0,813, 0,853, 0,804, 0,782]

- RELU, lr = 0,2, neuronas = [300, 150, 50], balance_classes = False

La evolución de su error de train y validation es la siguiente:



Seguidamente, lo entrenamos durante 225 épocas (best epoch) con los datos de train y validación conjuntamente para optimizar el rendimiento del modelo y garantizar una generalización adecuada. Al entrenar con ambos conjuntos de datos, el modelo tiene acceso a una variedad más amplia de patrones y características, lo que puede ayudar a mejorar su capacidad para reconocer y adaptarse a diferentes situaciones.
Evaluamos el modelo con los datos de test y los resultados obtenidos son los siguientes:

```
Accuracy: 0.841

Precision: [0.817, 0.864, 0.915, 0.793, 0.846]

Recall: [0.954, 0.771, 0.75, 0.82, 0.753]

F1 Score: [0.88, 0.815, 0.824, 0.806, 0.797]

Confusion Matrix:
[[3465  141   14    7    6]
 [ 397 1424    5   14    7]
 [ 212   36 1639  254   45]
 [ 136   36   70 1204   22]
 [  29   12   64   39  439]]
```

6. Análisis de resultados

6.1 Pruebas

En cuanto a los primeros experimentos realizados, podemos destacar una notable diferencia entre los modelos entrenados con una función de activación u otra en términos de error de validación. Observamos que la función de activación ReLU sobresale como la más efectiva para nuestros datos, logrando minimizar los errores de validación de manera significativa en comparación con las demás funciones. Además, comparando las otras dos funciones de activación, constatamos que la sigmoide exhibe un rendimiento superior respecto a la softmax.

Asimismo, percibimos que el aumento en el número de capas ocultas está asociado a una reducción en el error de validación para una misma tasa de aprendizaje. Esto se ve reflejado en los tres mejores modelos escogidos, (identificados por los errores de validación más bajos) los cuales presentan arquitecturas que incluyen tres capas ocultas.

6.2 Mejores modelos

Refiriéndonos a los mejores modelos, en contrario a lo que podíamos prever en un principio, los modelos en los que no se ha incluido balanceo dan unos resultados bastante similares a los modelos en los que sí se ha incluido el balanceo de clases. Esto lo vemos reflejado en las métricas de precision, recall, accuracy y f1-score, que son bastante parecidas.
El hecho de que la distribución de clases no es extremadamente desbalanceada, podría explicar por qué el balanceo con pesos no está generando mejoras sustanciales en los resultados. El balanceo de clases con pesos es más beneficioso cuando hay una gran disparidad en la cantidad de muestras entre las clases. Además, a esto se le puede añadir el hecho de que las clases tienen un tamaño lo suficientemente considerable como para permitir que el modelo aprenda de manera efectiva sin la necesidad de ajustar los pesos de clase.

Es relevante señalar que en todos los modelos, el recall de la clase mayoritaria (0) es bastante alto (por encima de 0.9), lo cual es comprensible dado que hay más instancias y las redes pueden aprender mejor estas clases. Esto indica una proporción elevada de instancias clasificadas como clase 0 ("andando") en relación con los valores reales de esa clase. Además, la precisión de esta clase también es notablemente alta.
Asimismo, observamos en estos modelos que tanto la precisión como el recall de la clase 2 ("en bus") también alcanzan valores elevados. Dado que esta es la segunda clase con mayor cantidad de instancias, este resultado es coherente.

En cuanto al resto de las clases, que son más minoritarias, aunque los resultados siguen siendo satisfactorios, ya sea la precisión o el recall tienden a ser ligeramente más bajos. Esto se ve reflejado en el f1-score, que es levemente menor para estas clases en comparación con las clases 0 y 2. Por ejemplo, para la clase 4, la más minoritaria, el recall arroja un buen resultado, pero la precisión es baja, tanto en modelos con balanceo como sin él. Esto indica que habrá una mayor proporción de instancias clasificadas incorrectamente como clase 4. En cambio, para las clases 1 y 3, el recall es más bajo que la precisión.

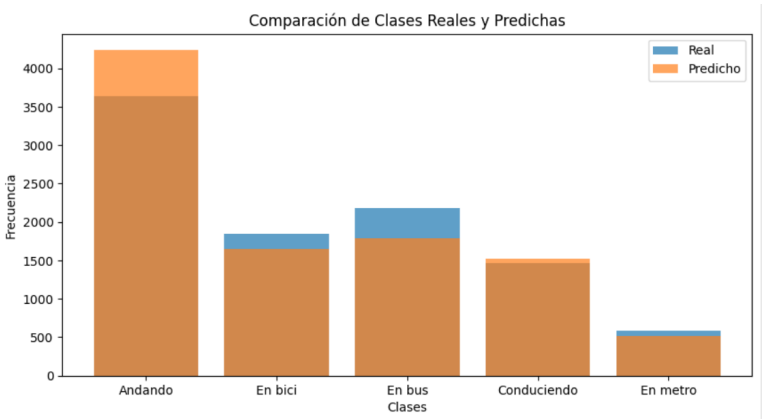
En cuanto al impacto del balanceo de clases, el principal cambio que observamos es que mejora el recall de las clases más minoritarias, especialmente de la clase 4 o "en metro". Sin embargo, empeora su precisión. Ocurre al contrario con las clases mayoritarias (0 y 2), el balanceo da como resultados valores más bajos del recall pero aumenta su precisión.

6.3 Modelo final

Ante esta semejanza de resultados, hemos escogido el mejor modelo basándonos en el objetivo de maximizar el accuracy y el f1-score, que relaciona la precisión con el recall.
Como hemos mencionado en los resultados obtenidos, la configuración del modelo final es la siguiente:

- RELU, lr = 0,2, neuronas = [300, 150, 50], balance_classes = False

Sorprendentemente, es un modelo que no experimenta un balanceo de clases por medio de la inclusión de pesos. Observando los resultados sobre los datos de test, hemos obtenido un 0.841 de accuracy, es decir, un 84.1 % de aciertos en los datos de test, lo que podemos considerar como un buen resultado.
Analizando la matriz de confusión, vemos como un considerable número de instancias que realmente pertenecen a clases distintas de la Clase 0 se están clasificando erróneamente como pertenecientes a esta clase. Este fenómeno impacta negativamente en el recall de estas clases y en la precisión de la Clase 0. Este comportamiento se atribuye principalmente a la naturaleza mayoritaria de la Clase 0 en los datos. La prevalencia de esta clase puede llevar al modelo a inclinarse hacia la predicción de instancias como pertenecientes a la Clase 0, lo que resulta en una disminución del recall para otras clases y una afectación en la precisión de la Clase 0.
Para visualizar de manera más clara este fenómeno, se presenta la siguiente gráfica de barras, la cual ilustra la distribución de las instancias reales y predichas para cada clase.



7. Conclusiones

En resumen, nuestros experimentos nos han permitido sacar diversas ideas sobre nuestros modelos y los datos utilizados. Primero, destacamos la eficacia de ReLU como función de activación y la tendencia positiva asociada con el aumento de capas ocultas en la arquitectura de red, lo que se ve reflejado en los 3 mejores resultados.
En segundo lugar, hemos comprobado que el balanceo de clases por medio de la inclusión de pesos no produce unos mejores resultados. Probablemente esto es debido a que los datos no presentan un desbalance significativo y cuentan con suficientes muestras para permitir que el modelo aprenda de manera efectiva sin la necesidad de ajustar los pesos de clase.
Finalmente, el modelo final seleccionado no presenta un balanceo de clases y aún así demuestra ser sólido, con un alto accuracy del 84.1% y un f1-score que supera el umbral del 0.8, considerado como un rendimiento notable.
En general, nuestros extensos experimentos nos han proporcionado conocimientos valiosos para el ajuste de hiperparámetros y la comprensión de la relación entre las características del modelo y el rendimiento en la resolución de problemas desbalanceados. Para una mejora adicional, se sugiere la implementación de oversampling en las clases minoritarias mediante técnicas como SMOTE. Este enfoque puede ayudar a reforzar la capacidad del modelo para reconocer y clasificar con precisión instancias pertenecientes a clases menos representadas.