Equilibrando el juego: Submuestreo y Sobremuestreo

INTRODUCCIÓN

Imagina que te encuentras implementando aprendizaje automático para crear un modelo de clasificación, y dentro de la variable objetivo te das cuenta que una de las clases es predominante con respecto a la otra clase en un porcentaje bastante mayor: A esto se le conoce como desbalanceo de datos, o el problema de datos desbalanceados, donde el número de instancias de las categorías de la variable objetivo no se distribuye de manera equitativa. Este problema puede tener un impacto significativo en la capacidad del modelo para generalizar y clasificar con precisión la clase minoritaria, el modelo puede inclinarse hacia la predicción de la clase mayoritaria, ignorando las otras clases, lo que resultaría en un mal rendimiento del modelo. Por lo tanto, es crucial aplicar diferentes estrategias para garantizar que el modelo aprenda de manera efectiva de todas las clases de la variable objetivo, y que pueda clasificar de manera correcta en situaciones del mundo real.

En esta ocasión, se pretende abordar el problema aplicando técnicas de balanceo de datos, específicamente: submuestreo y sobremuestreo.

SUBMUESTREO

El objetivo principal de esta técnica es balancear los datos eliminando instancias de manera aleatoria de la clase mayoritaria, con el fin de igualar el número de instancia de la clase minoritaria y así lograr que ambas clases tengan una distribución equitativa.

Este tipo de técnica ayuda a reducir el almacenamiento de datos y el tiempo de ejecución del código, ya que la cantidad de datos será mucho menor. Por otro lado, si la cantidad de datos no es suficientemente alta, el conjunto de datos se verá tan reducido que, al implementar un modelo de clasificación, este probablemente presente underfitting ya que no tendrá datos suficientes para obtener un buen resultado por parte del modelo entrenado.

SOBREMUESTREO

De la misma manera, esta técnica busca igualar las clases de la variable objetivo. Su principal diferencia es que busca replicar aleatoriamente instancias de la clase minoritaria, con el fin de igualar el número de instancia de la clase mayoritaria, en otras palabras, crea “nuevas” instancias que son copia de las ya existentes de la clase minoritaria para que la distribución de las clases sea equitativa.

La mayor ventaja de esta técnica, es que no existe perdida de información, ya que, en lugar de eliminar registros, se crean “nuevos”. Esto hace que el conjunto de datos tenga más registros con el fin de entrenar el modelo. Pero a su vez, el hecho de replicar los datos hace que nuestro modelo tenga datos repetidos en su entrenamiento, lo que puede conllevar a un overfitting ya que al ser datos iguales el modelo no generalice bien los datos nuevos, ya no aprenda a clasificarlos de manera óptima.

IMPLEMENTACIÓN

Para la implementación de las diferentes técnicas de balanceo de datos, se utilizará el conjunto de datos sobre el cáncer de mama. Este conjunto de datos cuenta con 31 columnas y 310 instancias. Para este caso, la variable objetivo cuenta con dos clases ‘0’ y ‘1’ que representan a pacientes sin cáncer y pacientes con cáncer.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

✓ 4.4s Python

```python
df = pd.read_csv('../data/unbalanced_cancer.csv', header= None)
df.head()
```

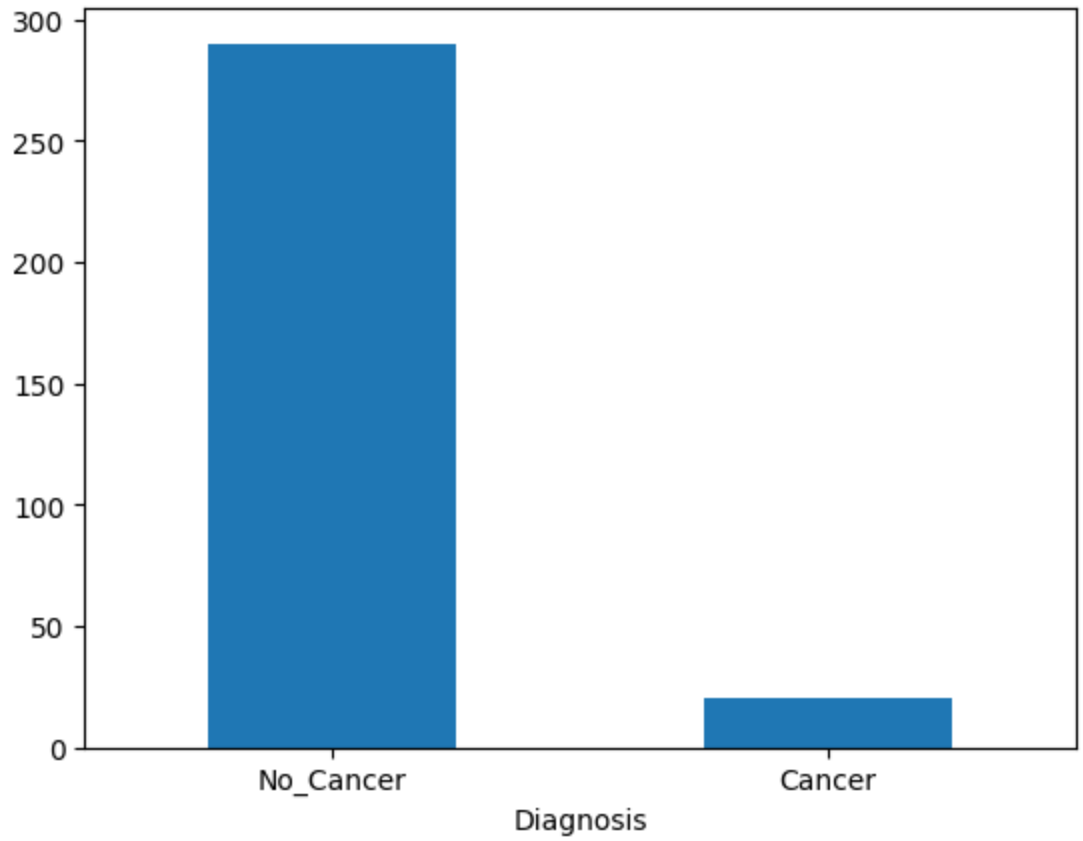
✓ 0.0s Python

	0	1	2	3	4	5	6	7	8	9	...	21	22	23	24	25	26	27	28	29	30
0	0.4163	0.4257	0.4095	0.2550	0.5721	0.5743	0.4603	0.5396	0.5163	0.3738	...	0.5843	0.4566	0.2885	0.6584	0.3997	0.4435	0.8080	0.4975	0.4999	1.0
1	0.7157	0.5216	0.7030	0.5359	0.5784	0.7947	0.6840	0.9248	0.6548	0.3433	...	0.5618	0.6056	0.4659	0.5368	0.5979	0.5633	0.9227	0.6943	0.4871	1.0
2	0.5347	0.4693	0.5633	0.3608	0.6353	1.0000	0.7763	0.7093	0.6257	0.4559	...	0.5296	0.4940	0.2750	0.7888	0.6198	0.7216	0.7482	0.5153	0.6916	1.0
3	0.5386	0.4544	0.5172	0.3598	0.4601	0.4660	0.2924	0.5224	0.5538	0.2245	...	0.4499	0.4367	0.2860	0.5232	0.3802	0.2637	0.7520	0.4586	0.3193	1.0
4	0.3987	0.8340	0.3698	0.2443	0.2760	0.1120	0.1141	0.1622	0.4706	0.0531	...	0.8364	0.2973	0.1834	0.2963	0.0784	0.1236	0.2505	0.5117	0.0690	1.0

5 rows × 31 columns

```python
df[30].value_counts().plot(kind='bar')
plt.xticks(labels=['No_Cancer', 'Cancer'], ticks=[0,1], rotation=0)
plt.xlabel('Diagnosis')
plt.show()
```

✓ 0.3s Python



Los 310 registros se encuentran divididos entre ‘0’ y ‘1’ con 290 y 20 registros respectivamente, siendo el 94% y 6% de los datos para cada cual, lo que puede considerarse como un desbalanceo de datos.

```
cancer = df[df[30]==1]
no_cancer = df[df[30]==0]
```
✓ 0.0s Python

La celda de código anterior está realizando una partición del dataframe ‘df’ basándose en los valores de la columna en la posición 30 (variable objetivo). La lógica de este código es la siguiente:

- *cancer = df[df[30]==1]*: Crea un nuevo dataframe llamado ‘cancer’ que contiene solo las filas donde la columna 30 tiene el valor 1, lo que indica casos de cáncer.
- *no_cancer = df[df[30]==0]*: Crea otro nuevo dataframe llamado ‘no_cancer’ que contiene las filas donde la columna 30 tiene el valor 0, lo que indica casos sin cáncer.

SUBMUESTREO

Como se explicó en la definición, la idea es reducir el número de registros de la clase ‘0’ (clase mayoritaria) para igualar los 20 registros de la clase ‘1’ (clase minoritaria), y se hace de la siguiente manera:

```
undersampling_no_cancer = no_cancer.sample(n=20, replace=False, random_state=0)
undersampling_no_cancer
```

✓ 0.0s Python

	0	1	2	3	4	5	6	7	8	9	...	21	22	23	24	25	26	27	28	29	30
75	0.0854	0.2539	0.0842	0.0371	0.5631	0.2629	0.1006	0.1139	0.3500	0.5704	...	0.2554	0.0701	0.0244	0.6876	0.2296	0.1962	0.4000	0.4174	0.4978	0.0
144	0.3449	0.4788	0.3214	0.1951	0.3437	0.2102	0.0356	0.1123	0.2715	0.1727	...	0.6136	0.2384	0.1303	0.3907	0.1362	0.0498	0.3034	0.2026	0.2244	0.0
201	0.2876	0.1498	0.2727	0.1533	0.5486	0.2895	0.0925	0.2013	0.2870	0.3145	...	0.3044	0.2111	0.1018	0.5673	0.2150	0.1431	0.4400	0.2505	0.3247	0.0
230	0.2827	0.3788	0.2661	0.1498	0.3984	0.2525	0.0654	0.1054	0.4153	0.2497	...	0.4694	0.1863	0.0970	0.3982	0.1720	0.1002	0.3166	0.3616	0.1839	0.0
291	0.2451	0.6299	0.2363	0.1263	0.4547	0.3056	0.1324	0.2530	0.4516	0.3973	...	0.6163	0.1697	0.0808	0.4071	0.1701	0.1285	0.4579	0.3171	0.2923	0.0
65	0.2544	0.4485	0.2336	0.1337	0.2483	0.0882	0.0580	0.1048	0.4034	0.1378	...	0.6996	0.1702	0.0891	0.2794	0.0674	0.0915	0.2831	0.3462	0.1453	0.0
294	0.2392	0.4909	0.2309	0.1224	0.1219	0.3653	0.2137	0.1413	0.1717	0.3446	...	0.5430	0.1648	0.0791	0.1589	0.3433	0.2570	0.3333	0.2268	0.4745	0.0
232	0.3141	0.3004	0.2913	0.1734	0.2414	0.1578	0.0531	0.0872	0.3506	0.1431	...	0.3253	0.2159	0.1176	0.3405	0.1541	0.1089	0.2662	0.3607	0.2207	0.0
35	0.3185	0.5224	0.3013	0.1771	0.3150	0.2418	0.1157	0.2004	0.4450	0.1469	...	0.6653	0.2420	0.1262	0.3058	0.1579	0.1823	0.4884	0.4006	0.1495	0.0
42	0.1560	0.2145	0.1451	0.0714	0.5486	0.2570	0.0264	0.0764	1.0000	0.4216	...	0.1825	0.0969	0.0451	0.3826	0.0784	0.0173	0.0982	0.6111	0.2676	0.0
142	0.2074	1.0000	0.1887	0.1032	0.2272	0.0687	0.0121	0.0381	0.4664	0.1712	...	1.0000	0.1396	0.0702	0.1954	0.0456	0.0185	0.1144	0.4128	0.1912	0.0
138	0.1692	0.2386	0.1595	0.0789	0.4755	0.2428	0.0162	0.0720	0.4326	0.3088	...	0.2614	0.1148	0.0517	0.4274	0.0966	0.0211	0.1701	0.3220	0.2942	0.0
32	0.2960	0.3614	0.2713	0.1613	0.3358	0.0767	0.0624	0.1731	0.2418	0.1789	...	0.3626	0.1696	0.0887	0.1755	0.0208	0.0386	0.1910	0.1292	0.0694	0.0
269	0.1186	0.4975	0.1105	0.0542	0.4674	0.1772	0.0498	0.0744	0.5734	0.3422	...	0.6458	0.0912	0.0428	0.5531	0.1010	0.0490	0.1436	0.4009	0.2980	0.0
83	0.2549	0.1485	0.2466	0.1325	0.4872	0.3178	0.0695	0.1706	0.4872	0.3134	...	0.1946	0.2040	0.0885	0.4858	0.2539	0.0792	0.3119	0.5841	0.3983	0.0
296	0.3693	0.1772	0.3499	0.2124	0.5170	0.3066	0.1678	0.3845	0.3506	0.2360	...	0.1650	0.2624	0.1388	0.4315	0.1323	0.1097	0.4072	0.3192	0.2451	0.0
248	0.1604	0.1041	0.1534	0.0755	0.4273	0.2352	0.0468	0.1165	0.4397	0.3351	...	0.1220	0.1137	0.0514	0.4288	0.1512	0.0688	0.2551	0.4208	0.2357	0.0
159	0.1824	0.4432	0.1797	0.0854	0.5017	0.4597	0.2056	0.1697	0.3613	0.3999	...	0.4432	0.1300	0.0553	0.4234	0.2501	0.2024	0.3276	0.3189	0.3405	0.0
202	0.2553	0.2282	0.2391	0.1334	0.3078	0.1934	0.0485	0.1002	0.3434	0.2378	...	0.3145	0.2027	0.0978	0.3697	0.1818	0.0932	0.2117	0.3361	0.2613	0.0
146	0.3258	0.2444	0.3046	0.1832	0.3756	0.1970	0.1145	0.2209	0.3904	0.1340	...	0.2362	0.2176	0.1224	0.3846	0.1118	0.1224	0.3513	0.2959	0.1059	0.0

20 rows × 31 columns

Esta parte del código está utilizando el método *sample* de pandas para realizar el submuestreo. Aquí están los detalles:

- *n=20*: Se están seleccionando aleatoriamente 20 muestras del dataframe ‘no_cancer’. Esto significa que se reducirá el tamaño de ‘no_cancer’ a 20 instancias.
- *replace=False*: Esto indica que las muestras se seleccionan sin reemplazo. En otras palabras, cada muestra seleccionada no se puede seleccionar nuevamente, asegurando que no haya duplicados en el conjunto de datos.
- *random_state=0*: Establece una semilla para la generación de números aleatorios, garantizando que, si ejecutas este código varias veces, obtendrás la misma selección aleatoria cada vez.
- *undersampling_no_cancer*: Esto almacena el resultado del submuestreo en la variable *undersampling_no_cancer*. Este nuevo dataframe contendrá solo 20 instancias aleatorias de ‘no_cancer’.

```
undersampling = pd.concat([cancer, undersampling_no_cancer])
X_undersampling = undersampling.iloc[:, :-1]
y_undersampling = undersampling.iloc[:, -1:]
```

Python

Por último, se concatena los datos de pacientes con cancer y el dataframe anterior para crear el conjunto de datos balanceado a través de submuestreo. Adicionalmente separamos el conjunto de datos en datos de entrada (X_undersampling) y datos de salida (y_undersampling) que corresponde a las respectivas clases para cada instancia.

SOBREMUESTREO

Como dice su definición, el objetivo es replicar instancias de la clase ‘1’ (clase minoritaria) hasta igualar el número de registros de la clase ‘0’ (clase mayoritaria) con 290 instancias cada una.

```
oversampling_cancer = cancer.sample(n=290, replace=True, random_state=0)
oversampling_cancer
```

✓ 0.0s Python

	0	1	2	3	4	5	6	7	8	9	...	21	22	23	24	25	26	27	28	29	30
12	0.5269	0.7602	0.5137	0.3555	0.4274	0.4702	0.4133	0.4910	0.3880	0.1907	...	0.8911	0.4731	0.3077	0.4682	0.3832	0.5111	0.7505	0.4316	0.3806	1.0
15	0.4941	0.7237	0.4710	0.3340	0.4184	0.3846	0.2451	0.3808	0.4355	0.2714	...	0.7550	0.5064	0.3595	0.6033	0.2384	0.2108	0.5924	0.4432	0.3789	1.0
0	0.4163	0.4257	0.4095	0.2550	0.5721	0.5743	0.4603	0.5396	0.5163	0.3738	...	0.5843	0.4566	0.2885	0.6584	0.3997	0.4435	0.8080	0.4975	0.4999	1.0
3	0.5386	0.4544	0.5172	0.3598	0.4601	0.4660	0.2924	0.5224	0.5538	0.2245	...	0.4499	0.4367	0.2860	0.5232	0.3802	0.2637	0.7520	0.4586	0.3193	1.0
3	0.5386	0.4544	0.5172	0.3598	0.4601	0.4660	0.2924	0.5224	0.5538	0.2245	...	0.4499	0.4367	0.2860	0.5232	0.3802	0.2637	0.7520	0.4586	0.3193	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13	0.6908	0.4485	0.6590	0.5262	0.3991	0.4119	0.3827	0.6838	0.2935	0.1343	...	0.6737	0.5866	0.4514	0.4458	0.3106	0.3514	0.8686	0.2155	0.2039	1.0
17	0.5195	0.5651	0.5255	0.3550	0.3635	0.7599	0.5200	0.5933	0.3779	0.5121	...	0.5662	0.4641	0.3077	0.3364	0.4955	0.4125	0.7604	0.2256	0.7195	1.0
0	0.4163	0.4257	0.4095	0.2550	0.5721	0.5743	0.4603	0.5396	0.5163	0.3738	...	0.5843	0.4566	0.2885	0.6584	0.3997	0.4435	0.8080	0.4975	0.4999	1.0
11	0.5406	0.2963	0.5151	0.3591	0.3326	0.4358	0.2653	0.3703	0.3922	0.1606	...	0.3360	0.4128	0.2720	0.3744	0.2628	0.3427	0.5848	0.3917	0.2895	1.0
4	0.3987	0.8340	0.3698	0.2443	0.2760	0.1120	0.1141	0.1622	0.4706	0.0531	...	0.8364	0.2973	0.1834	0.2963	0.0784	0.1236	0.2505	0.5117	0.0690	1.0

290 rows × 31 columns

- *.sample(n=290, replace=True, random_state=0)*: Esta parte de la línea de código utiliza el método sample de pandas para realizar el sobremuestreo. Aquí están los detalles:
- *n=290*: Se están extrayendo 290 muestras aleatorias del dataframe 'cancer'. Estas muestras serán replicadas para realizar el sobremuestreo.
- *replace=True*: Esto significa que las muestras se seleccionan con reemplazo, lo que implica que una misma observación del conjunto de datos original puede ser seleccionada varias veces para formar parte del nuevo conjunto de datos.
- *random_state=0*: Esto establece una semilla para la generación de números aleatorios. Usar una semilla garantiza que, si ejecutas este código varias veces, obtendrás el mismo conjunto de muestras aleatorias cada vez.

```
oversampling = pd.concat([oversampling_cancer, no_cancer])
X_oversampling = oversampling.iloc[:, :-1]
y_oversampling = oversampling.iloc[:, -1:]
```

Python

Finalmente, unimos los datos de pacientes sin cancer y el nuevo dataframe creado para tener un conjunto de datos balanceado por medio del sobremuestreo. Adicionalmente separamos el conjunto de datos en datos de entrada y datos de salida.

MODELO DE CLASIFICACIÓN

Para observar como este tipo de técnicas favorecen el rendimiento de un modelo, creamos 3 modelos de clasificación utilizando Regresión Logística para cada conjunto de datos. El primero utilizando el conjunto de datos desbalanceado, el segundo con los datos balanceados mediante submuestreo, y el último modelo utilizando el conjunto de datos balanceado a través de sobremuestreo.

```
unbalanced = pd.concat([cancer, no_cancer])
X_unbalanced = unbalanced.iloc[:, :-1]
y_unbalanced = unbalanced.iloc[:, -1:]
```

Python

Para validar el rendimiento del modelo, se cargó un conjunto de datos de prueba con 20 instancias (10 de pacientes con cáncer, y 10 de pacientes sin cáncer), las cuales no conocerá el modelo entrenado:

```
test = pd.read_csv('../data/test_cancer.csv', header= None)
X_test = test.iloc[:, :-1]
y_test = test.iloc[:, -1:]
```
✓ 0.0s Python

Modelo 1

El primer modelo se entrena con los datos desbalanceados:

```
model_unbalanced = LogisticRegression().fit(X_unbalanced.values,
                                            y_unbalanced.values.reshape(-1))
print(confusion_matrix(y_test.values,
                       model_unbalanced.predict(X_test.values)))
```
✓ 0.0s Python

```
[[10  0]
 [ 6  4]]
```

```
print(accuracy_score(y_test.values,
                     model_unbalanced.predict(X_test.values)))
```
✓ 0.0s Python

```
0.7
```

Haciendo uso de la matriz de confusión, se evidencia que el modelo entrenado logra clasificar de manera optima todos los pacientes que no tienen cancer, mientras que solo logra clasificar de manera optima 4 pacientes con cancer y falla en la clasificación de los 6 restantes. Adicionalmente, se observa que el accuracy es de un 70 %.

Modelo 2

El segundo modelo se entrena con los datos balanceados por medio de la técnica de submuestreo:

```
model_undersampling = LogisticRegression().fit(X_undersampling.values,
                                               y_undersampling.values.reshape(-1))
print(confusion_matrix(y_test.values,
                       model_undersampling.predict(X_test.values)))
```
✓ 0.0s Python

```
[[10  0]
 [ 1  9]]
```

```
print(accuracy_score(y_test.values,
                     model_undersampling.predict(X_test.values)))
```
✓ 0.0s Python

```
0.95
```

Para este caso, haciendo uso de la matriz de confusión, se evidencia que el modelo entrenado logra clasificar de manera óptima todos los pacientes que no tienen cancer, y falla únicamente en 1 paciente con cancer mientras que los 9 restantes los clasifica de manera óptima. Adicionalmente, se observa que el accuracy aumento a un 95%, siendo mejor que el modelo anterior.

Modelo 3

El último modelo se entrena con el conjunto de datos balanceado mediante sobremuestreo:

```
model_oversampling = LogisticRegression().fit(X_oversampling.values,
                                              y_oversampling.values.reshape(-1))
print(confusion_matrix(y_test.values,
                       model_oversampling.predict(X_test.values)))
```
✓ 0.0s Python

```
[[10  0]
 [ 1  9]]
```

```
print(accuracy_score(y_test.values, model_oversampling.predict(X_test.values)))
```
✓ 0.0s Python

```
0.95
```

En este ultimo modelo, haciendo uso de la matriz de confusión, se obtuvo los mismos resultados que el modelo anterior, se clasificaron bien todos los pacientes con cancer, mientras que para los pacientes sin cancer, falla con un dato y los 9 restantes lo hace de manera correcta. Adicionalmente, se obtiene un accuracy de 95% mejorando el puntaje obtenido con el conjunto de datos desbalanceado.

¿Cómo elegir entre submuestreo y sobremuestreo?

La elección entre submuestreo y sobremuestreo depende del caso específico. Submuestreo puede ser eficaz cuando el conjunto de datos es grande y la información redundante puede eliminarse sin afectar gravemente el rendimiento del modelo. Por otro lado, el sobremuestreo es útil cuando la cantidad de datos es limitada y necesitamos fortalecer la representación de clases minoritarias.

INFORMACION EXTRA

Existen otro tipo de técnicas avanzadas más allá del submuestreo y el sobremuestreo tradicionales. Un ejemplo de ellos es el SMOTE (Synthetic Minority Over-sampling Technique) que crea instancias sintéticas para las clases minoritarias, ampliando de manera inteligente el conjunto de datos sin replicar datos, lo que evita los datos repetidos o duplicados.

CONCLUSIONES

En el aprendizaje hacia modelos mas precisos y con mejor rendimiento, el balanceo de datos es un aliado indispensable. Ya sea de manera en que se reduzcan las instancias o que se amplíen las clases minoritarias, este tipo de técnicas se destacan como herramientas esenciales para resolver los problemas existentes en el mundo de la ciencia de datos.

Como se pudo observar, ambas técnicas mejoraron el rendimiento del modelo, evitando que este clasificara de manera errónea datos que con el desbalance de datos no era posible, además de que el accuracy aumento de una manera considerable. ¿Pero qué sucede cuando las técnicas de balanceo de datos no pueden implementarse? Existen otro tipo de soluciones, como tener en cuenta otro tipo de métricas como el f1-score o el recall que dan un entendimiento distinto a lo que se

busca resolver con el aprendizaje automático, o incluso dar pesos a los modelos entrenados para que tenga en cuenta ese desbalance de los datos. Pero todo esto será información para otra ocasión.

REFERENCIAS RELACIONADAS

Gutiérrez-García, J.O. [Código Máquina]. (2023, 7 de agosto). Qué son los Datos Desbalanceados y Cómo balancearlos usando Submuestreo y Sobremuestreo con Python [Video]. YouTube. [https://www.youtube.com/watch?v=2J90FG6QKL4&ab_channel=CodigoMaquina].

Cómo actuar ante el desbalance de datos | by Nicolás Arrioja Landa Cosio | Medium https://medium.com/@nicolasarrioja/c%C3%B3mo-actuar-ante-el-desbalance-de-datos-a0d64f2b9619

Gutiérrez-García, J.O. [Código Máquina]. (2022, 20 de Agosto). Métricas para Clasificadores de Machine Learning: Matriz de Confusión, Precision, Accuracy, Recall, F1 [Video]. YouTube. [https://www.youtube.com/watch?v=uaGMk43XTOw&ab_channel=CodigoMaquina]

By Daniel Augusto Muñoz Viveros