Balancing the Game: Undersampling and Oversampling

INTRODUCTION

Imagine that you are implementing machine learning to create a classification model, and you realize that one of the classes in the target variable predominates significantly over the other class by a considerably higher percentage: this is known as data imbalance or the imbalanced data problem, where the number of instances in the categories of the target variable is not distributed evenly. This issue can have a significant impact on the model's ability to generalize and accurately classify the minority class. The model may lean towards predicting the majority class, ignoring the other classes, resulting in poor model performance. Therefore, it is crucial to apply different strategies to ensure that the model effectively learns from all classes of the target variable and can classify accurately in real-world situations.

This time, we aim to address the problem by applying data balancing techniques, specifically: undersampling and oversampling.

UNDERSAMPLING

The main objective of this technique is to balance the data by randomly removing instances from the majority class, in order to equalize the number of instances in the minority class and thus achieve an equitable distribution for both classes.

This type of technique helps reduce data storage and code execution time, as the amount of data will be much smaller. On the other hand, if the amount of data is not high enough, the dataset will be so reduced that, when implementing a classification model, it will likely exhibit underfitting as it won't have enough data to produce a good result from the trained model.

OVERSAMPLING

Similarly, this technique aims to balance the classes of the target variable. Its main difference is that it seeks to randomly replicate instances of the minority class to equalize the number of

instances in the majority class. In other words, it creates "new" instances that are copies of the existing ones from the minority class so that the distribution of the classes becomes equitable.

The major advantage of this technique is that there is no loss of information since, instead of removing records, "new" ones are created. This results in the dataset having more records for training the model. However, the act of replicating data means that our model has repeated data in its training, which can lead to overfitting. Since the data is identical, the model may not generalize well to new data, failing to learn how to classify them optimally.

IMPLEMENTATION

For the implementation of the different data balancing techniques, the breast cancer dataset will be used. This dataset consists of 31 columns and 310 instances. In this case, the target variable has two classes, '0' and '1', representing patients without cancer and patients with cancer.

```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()
```

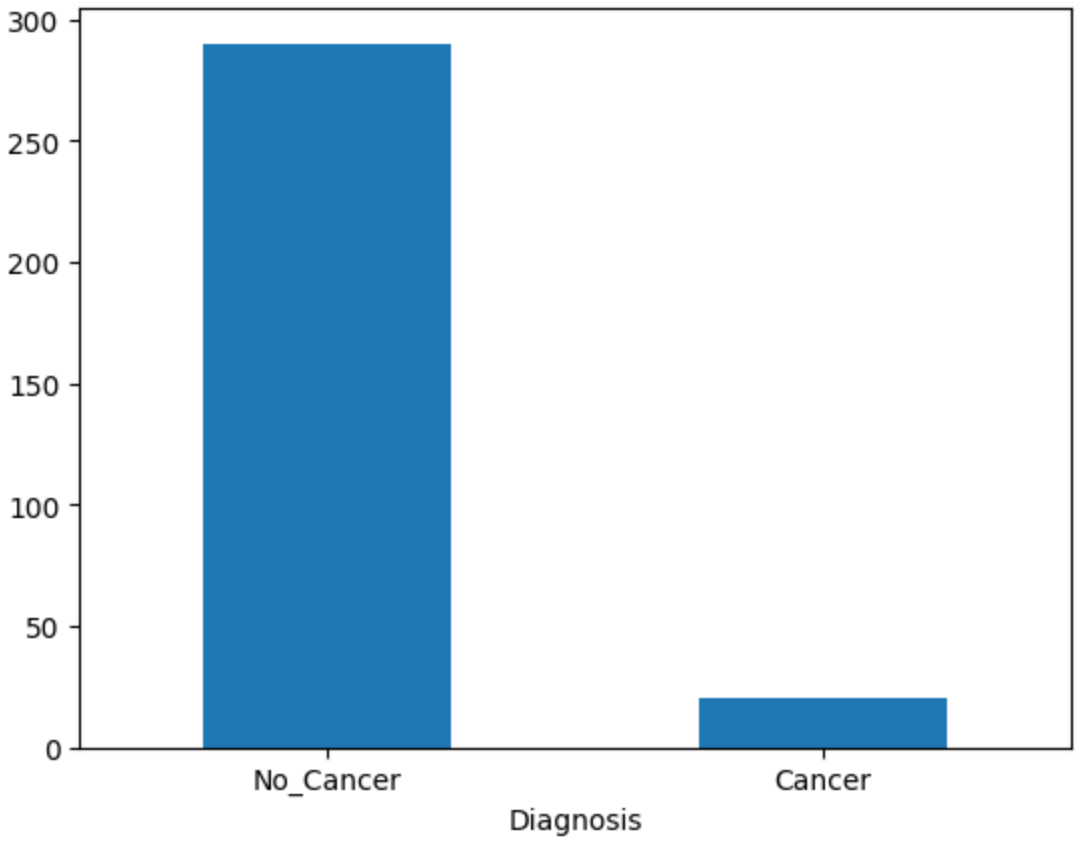
✓ 0.0s Python

	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



The 310 records are divided between '0' and '1' with 290 and 20 records respectively, representing 94% and 6% of the data for each class, which can be considered as data imbalance.

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

The previous code cell is performing a partition of the 'df' dataframe based on the values in the column at position 30 (target variable). The logic of this code is as follows:

- cancer = df[df[30]==1]: Creates a new dataframe called 'cancer' that contains only the rows where column 30 has the value 1, indicating cancer cases.
- no_cancer = df[df[30]==0]: Creates another new dataframe called 'no_cancer' that contains the rows where column 30 has the value 0, indicating cases without cancer.

UNDERSAMPLING

As explained in the definition, the idea is to reduce the number of records from class '0' (majority class) to match the 20 records of class '1' (minority class), and this is done as follows:

```
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

This part of the code is using the sample method from pandas to perform undersampling. Here are the details:

- n=20: 20 samples are randomly selected from the 'no_cancer' dataframe. This means the size of 'no_cancer' will be reduced to 20 instances.
- replace=False: This indicates that samples are selected without replacement. In other words, each selected sample cannot be chosen again, ensuring there are no duplicates in the dataset.
- random_state=0: Sets a seed for random number generation, ensuring that if you run this code multiple times, you will get the same random selection each time.
- undersampling_no_cancer: This stores the result of undersampling in the variable 'undersampling_no_cancer'. This new dataframe will contain only 20 random instances from 'no_cancer'.

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

Python

Finally, the data of patients with cancer is concatenated with the previous dataframe to create the balanced dataset through undersampling. Additionally, we separate the dataset into input data (X_undersampling) and output data (y_undersampling), corresponding to the respective classes for each instance.

OVERSAMPLING

As per its definition, the objective is to replicate instances of class '1' (minority class) until matching the number of records in class '0' (majority class) with 290 instances each.

```
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): This part of the code line uses the pandas sample method to perform oversampling. Here are the details:
- n=290: 290 random samples are extracted from the 'cancer' dataframe. These samples will be replicated to perform oversampling.
- replace=True: This indicates that samples are selected with replacement, meaning that the same observation from the original dataset can be selected multiple times to be part of the new dataset.
- random_state=0: Sets a seed for random number generation. Using a seed ensures that if you run this code multiple times, you will get the same set of random samples each time.

Finally, we combine the data of patients without cancer with the newly created dataframe to have a balanced dataset through oversampling. Additionally, we separate the dataset into input data and output data.

CLASSIFICATION MODEL

To observe how these types of techniques enhance the performance of a model, we created three classification models using Logistic Regression for each dataset. The first one using the imbalanced dataset, the second one with the data balanced through undersampling, and the last model using the dataset balanced through oversampling.

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

Python

To validate the model's performance, a test dataset with 20 instances (10 from patients with cancer and 10 from patients without cancer) was loaded, which the trained model will not have seen before:

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

✓ 0.0s Python

Model 1

The first model is trained with the imbalanced data:

```
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
```

Using the confusion matrix, it is evident that the trained model successfully classifies all patients without cancer optimally, while it only correctly classifies 4 patients with cancer and fails to classify the remaining 6. Additionally, it is observed that the accuracy is 70%.

Model 2

The second model is trained with data balanced using the undersampling technique:

```
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
```

In this case, using the confusion matrix, it is evident that the trained model successfully classifies all patients without cancer optimally. It only fails in classifying 1 patient with cancer, while correctly classifying the remaining 9. Additionally, it is observed that the accuracy has increased to 95%, which is an improvement compared to the previous model.

Model 3

The last model is trained with the dataset balanced through oversampling:

```
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
```

In this last model, using the confusion matrix, the same results as the previous model were obtained – all cancer patients were classified correctly, while for patients without cancer, there is an error in one data point, and the remaining nine are classified correctly. Additionally, an accuracy of 95% is achieved, improving the score obtained with the imbalanced dataset.

How to choose between undersampling and oversampling?

The choice between undersampling and oversampling depends on the specific case. Undersampling can be effective when the dataset is large, and redundant information can be removed without severely impacting the model's performance. On the other hand, oversampling is useful when the amount of data is limited, and there is a need to strengthen the representation of minority classes.

EXTRA INFORMATION

There are other advanced techniques beyond traditional undersampling and oversampling. One example is SMOTE (Synthetic Minority Over-sampling Technique), which creates synthetic instances for minority classes, intelligently expanding the dataset without replicating data, thereby avoiding repeated or duplicated data.

CONCLUSIONS

In the journey towards more precise and high-performance models, data balancing becomes an indispensable ally. Whether instances are reduced or minority classes are expanded, these techniques stand out as essential tools for solving challenges in the field of data science.

As observed, both techniques improved the model's performance, preventing it from misclassifying data that was not possible to achieve with imbalanced data. Additionally, the accuracy increased significantly. But what happens when data balancing techniques cannot be implemented? There are other types of solutions, such as considering alternative metrics like f1-score or recall, which provide a different understanding of what machine learning aims to address. Another approach involves assigning weights to trained models to account for data imbalance. However, these topics will be discussed on another occasion.

RELATED REFERENCES

Gutiérrez-García, J.O. [Código Máquina]. (2023, August 7). 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, August 20). 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