ML 2 - Handling Imbalanced Dataset By Virat Tiwari

November 7, 2023

1 Handling Imbalanced Dataset - With the help of handling imbalanced dataset we have to get correct accuracy from the dataset otherwise it will give biased accuracy with imbalanced dataset

There are two techniques for solving " Handling The Imabalced Dataset" - :

1) Upsampling

2) Down Sampling

[64]:
```
# For making a dataset we have to import these two libraries with that we have␣
↪to create the dataset

import numpy as np
import pandas as pd
```

[65]:
```
# With the have help numpy we have to create a seed so that our values are not␣
↪going to be change

# set the random seed for reproductivity

np.random.seed(123)

# Here we Create a dataframe with two classes

# Basically we create a imbalanced dataset in two categories

n_sample=1000
class_0_ratio=0.9
n_class_0=int(n_sample*class_0_ratio)
n_class_1=n_sample-n_class_0
```

[66]:
```
# Here we see the imbalaced datapoints value in two diffrent catergories

n_class_0,n_class_1
```

[66]:
```
(900, 100)
```

[67]:
```
# Here we create two classes for two dataframes with "feature 1" and "feature 2"

# np.random.normal is nothing but a normal distribution

# scale is similar to standard deviation and loc is similar to mean

# target [0] gives 900 zeroes
# target [1] gives the 100 zeroes

class_0=pd.DataFrame({
    "feature_1":np.random.normal(loc=0,scale=1,size=n_class_0),
    "feature_2":np.random.normal(loc=0,scale=1,size=n_class_0),
    "target":[0]*n_class_0
})

class_1=pd.DataFrame({
    "feature_1":np.random.normal(loc=2,scale=1,size=n_class_1),
    "feature_2":np.random.normal(loc=2,scale=1,size=n_class_1),
    "target":[1]*n_class_1
})
```

[68]:
```
# Here we get the complete dataframe

df=pd.concat([class_0,class_1]).reset_index(drop=True)
```

[69]:
```
df.head()
```

[69]:
```
   feature_1  feature_2  target
0  -1.085631   0.551302       0
1   0.997345   0.419589       0
2   0.282978   1.815652       0
3  -1.506295  -0.252750       0
4  -0.578600  -0.292004       0
```

[70]:
```
# Total no of "0" is 900
# Total no of "1" is 100

# 900 datapoints have 0 output
# 100 datapoints have 1 output

df["target"].value_counts()
```

[70]:
```
0    900
1    100
Name: target, dtype: int64
```

2 UPSAMPLING - In Upsampling those section have less datapoints so we have to manage or balance that datapoints by creating some more datapoints

[71]:
```
df_minority=df[df["target"]==1]
df_majority=df[df["target"]==0]
```

[72]:
```
df_minority
```

[72]:
```
     feature_1  feature_2  target
900   1.699768   2.139033       1
901   1.367739   2.025577       1
902   1.795683   1.803557       1
903   2.213696   3.312255       1
904   3.033878   3.187417       1
..         …          …     …
995   1.376371   2.845701       1
996   2.239810   0.880077       1
997   1.131760   1.640703       1
998   2.902006   0.390305       1
999   2.697490   2.013570       1

[100 rows x 3 columns]
```

[73]:
```
df_majority
```

[73]:
```
     feature_1  feature_2  target
0    -1.085631   0.551302       0
1     0.997345   0.419589       0
2     0.282978   1.815652       0
3    -1.506295  -0.252750       0
4    -0.578600  -0.292004       0
..         …          …     …
895   0.238761  -0.003155       0
896  -1.106386  -0.430660       0
897   0.366732  -0.146416       0
898   1.023906   1.160176       0
899  -0.210056  -0.641512       0

[900 rows x 3 columns]
```

[74]:
```
df_minority.head()
```

[74]:
```
     feature_1  feature_2  target
900   1.699768   2.139033       1
901   1.367739   2.025577       1
```

```
902   1.795683   1.803557       1
903   2.213696   3.312255       1
904   3.033878   3.187417       1
```

[75]:
```
df_majority.head()
```

[75]:
```
   feature_1  feature_2  target
0  -1.085631   0.551302       0
1   0.997345   0.419589       0
2   0.282978   1.815652       0
3  -1.506295  -0.252750       0
4  -0.578600  -0.292004       0
```

PERFORMING UPSAMPLING -

[76]:
```
# for upsampling we have to use sklearn
# reshape library also help in upsampling

from sklearn.utils import resample
```

[77]:
```
# Here we increase the datapoints of minority part

# replace ( ) function is used for "sample with replacement"

# n_sample = len ( ) function is used for "match the majority class"

df_minority_upsample=resample(df_minority,replace=True,n_samples=len(df_majority),random_state
```

[78]:
```
df_minority_upsample.shape
```

[78]:
```
(900, 3)
```

[79]:
```
df_minority_upsample.head()
```

[79]:
```
     feature_1  feature_2  target
951   1.125854   1.843917       1
992   2.196570   1.397425       1
914   1.932170   2.998053       1
971   2.272825   3.034197       1
960   2.870056   1.550485       1
```

[80]:
```
df_minority_upsample["target"].value_counts()
```

[80]:
```
1    900
Name: target, dtype: int64
```

[81]:
```
df_upsampled=pd.concat([df_majority,df_minority_upsample])
```

[82]:
```
df_upsampled["target"].value_counts()
```

[82]:
```
0    900
1    900
Name: target, dtype: int64
```

[83]:
```
df_upsampled.head()
```

[83]:
```
   feature_1  feature_2  target
0  -1.085631   0.551302       0
1   0.997345   0.419589       0
2   0.282978   1.815652       0
3  -1.506295  -0.252750       0
4  -0.578600  -0.292004       0
```

[84]:
```
df_upsampled.shape
```

[84]:
```
(1800, 3)
```

3 DOWN SAMPLING - In downsapmling we have to reduce the datapoints from the higher sections of categoricatl dataset for balancing the datapoints , We simply reduce the more datapoints for balancing the catergorical dataset

[85]:
```
class_0=pd.DataFrame({
    "feature_1":np.random.normal(loc=0,scale=1,size=n_class_0),
    "feature_2":np.random.normal(loc=0,scale=1,size=n_class_0),
    "target":[0]*n_class_0
})

class_1=pd.DataFrame({
    "feature_1":np.random.normal(loc=2,scale=1,size=n_class_1),
    "feature_2":np.random.normal(loc=2,scale=1,size=n_class_1),
    "target":[1]*n_class_1
})
```

[86]:
```
df=pd.concat([class_0,class_1]).reset_index(drop=True)
```

[87]:
```
df_minority=df[df["target"]==1]
df_majority=df[df["target"]==0]
```

[88]:
```
# Here we increase the datapoints of majority part

# replace ( ) function is used for "sample with replacement"

# n_sample = len ( ) function is used for "match the majority class"

df_majority_downsample=resample(df_majority,replace=False,n_samples=len(df_minority),random_st
```

[89]:
```
df_majority_downsample.shape
```

[89]:
```
(100, 3)
```

[90]:
```
df_downsample=pd.concat([df_minority,df_majority_downsample])
```

[91]:
```
df_downsample["target"].value_counts()
```

[91]:
```
1    100
0    100
Name: target, dtype: int64
```

THANK YOU SO MUCH !!

YOURS VIRAT TIWARI :)