34. Imbalanced Dataset

- Imbalanced dataset means that your data consists of multi categories and one category is repetitive in the data
- model is baised due to repitition of one category in the data
- Suppose your data consist of 500 rows:
- 400 rows for cat and 100 rows for dog,
- so the model will be biased towards cat

34.1 Techniques to handle imbalanced data

34.1.1 Random Under Sampling

- we will reduce the majority of the class so that it will have same number of as the minority
- for example out of 500 rows for cats and dogs, we will reduce (randomly) the rows to 100 for cats that is equal to 100 rows of dogs

34.1.2 Random Over Sampling

- We will increase the size of manority is inactive class to the size of majority calss i.e. active
- for example out of 500 rows for cats and dogs, we will repeat/duplicate (randomly) the rows for dogs to make it to 400 that is equal to 400 rows of cats

In [1]:

```
import pandas as pd
```

In [3]:

```
dataset = pd.read_csv(r'Data/Social_Network_Ads.csv')
dataset.head(3)
```

Out[3]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0

To check the data if it is imbalanced or not

In [4]:

```
dataset['Purchased'].value_counts()
```

Out[4]:

```
0    257
1    143
Name: Purchased, dtype: int64
```

So hence the data is **imbalanced** b/c both categories are not equal, so the data will be baised towards 0

In [12]:

```
x = dataset.iloc[:,:-1]
x
```

Out[12]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

In [13]:

```
y = dataset['Purchased']
y
```

Out[13]:

```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

In [14]:

```
from sklearn.model_selection import train_test_split
```

In [15]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
In [16]: from sklearn.linear_model import LogisticRegression
```

```
In [17]: lg = LogisticRegression()
         lg.fit(x_train, y_train)
```

```
Out[17]: ▾ LogisticRegression
         LogisticRegression()
```

```
In [22]: lg.score(x_test, y_test)*100
```

```
Out[22]: 65.0
```

```
In [23]: # y_true is 0
         lg.predict([[19, 19000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[23]: array([0], dtype=int64)
```

```
In [25]: # y_true is 1
         lg.predict([[45, 26000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[25]: array([0], dtype=int64)
```

It has given wrong prediction, Reason: B/c the input data is **imbalanced**

```
In [26]: # y_true is 1
         lg.predict([[46, 28000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[26]: array([0], dtype=int64)
```

Again, it has given wrong prediction, Reason: B/c the input data is **imbalanced**

```
In [ ]:
```

So hence we will balance our data by either:

- random under sampling, or
- random over sampling

34.2.1 Balacing the data by Random Under Sampling (Practical)

In [28]:

```
from imblearn.under_sampling import RandomUnderSampler
```

In [31]:

```
ru = RandomUnderSampler()
ru_x, ru_y = ru.fit_resample(x,y)
```

In [32]:

```
ru_x
```

Out[32]:

	Age	EstimatedSalary
224	35	60000
49	31	89000
153	36	50000
132	30	87000
359	42	54000
...	...	...
393	60	42000
395	46	41000
396	51	23000
397	50	20000
399	49	36000

286 rows × 2 columns

In [33]:

```
ru_y
```

Out[33]:

```
224    0
49     0
153    0
132    0
359    0
      ..
393    1
395    1
396    1
397    1
399    1
Name: Purchased, Length: 286, dtype: int64
```

Now after applying under sampling technique we will see, if 0 count is reduced to 143 or not

In [34]:

```
# Remember, out original data has following counts:
dataset['Purchased'].value_counts()
```

Out[34]:

```
0    257
1    143
Name: Purchased, dtype: int64
```

In [35]:

```
ru_y.value_counts()
```

Out[35]:

```
0    143
1    143
Name: Purchased, dtype: int64
```

So you can see 0 has reduced to 143 and now **our data is balanced!**

Now we have new data variables that are **ru_x, ru_y**

We will apply logitic regression on this new dataset that is balanced data, so we first split the data into train and test and then will apply logistic regression model

In [36]:

```
from sklearn.model_selection import train_test_split
```

In [37]:

```
x_train, x_test, y_train, y_test = train_test_split(ru_x, ru_y, test_size=0.20, ran
```

In [38]:

```
from sklearn.linear_model import LogisticRegression
```

In [39]:

```
ru_lg = LogisticRegression()
ru_lg.fit(x_train, y_train)
```

Out[39]:

```
▾ LogisticRegression
LogisticRegression()
```

ru_lg.score(x_test, y_test)*100

To check if the model has improved or not we will supply same value as were predicted wrongly

In [41]:

```
# y_true is 1
ru_lg.predict([[45, 26000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

Out[41]:

```
array([1], dtype=int64)
```

Hurrahh, now it has given accurate prediction, lets try second test..

```
In [43]: # y_true is 1
         ru_lg.predict([[46, 28000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[43]: array([1], dtype=int64)
```

Oh yes, the second prediction is also accurate!!!

```
In [44]: # y_true is 0
         lg.predict([[19, 19000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[44]: array([0], dtype=int64)
```

Great, accurate prediction again!!!

Conclusion is our model is not more biased

34.2.2 Balacing the data by Random Over Sampling (Practical)

```
In [46]: from imblearn.over_sampling import RandomOverSampler
```

```
In [47]: ro = RandomOverSampler()
         ro_x, ro_y = ro.fit_resample(x,y)
```

```
In [48]: ro_x
```

Out[48]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
509	42	73000
510	55	39000
511	46	28000
512	37	93000
513	46	79000

514 rows × 2 columns

In [49]:

```
ro_y
```

Out[49]:

```
0      0
1      0
2      0
3      0
4      0
      ..
509    1
510    1
511    1
512    1
513    1
Name: Purchased, Length: 514, dtype: int64
```

In [61]:

```
# Remember, out original data has following counts:
dataset['Purchased'].value_counts()
```

Out[61]:

```
0    257
1    143
Name: Purchased, dtype: int64
```

So 1 should be increased to 257 as well as we have applied random over sampling method

In [60]:

```
ro_y.value_counts()
```

Out[60]:

```
0    257
1    257
Name: Purchased, dtype: int64
```

Now input is ro_x and output is ro_y, we will split the data into test and train and then apply logistic regression model

```
In [50]: x_train, x_test, y_train, y_test = train_test_split(ro_x, ro_y, test_size=0.20, ran
```

```
In [52]: ro_lg = LogisticRegression()
         ro_lg.fit(x_train, y_train)
```

```
Out[52]: ▾ LogisticRegression
         LogisticRegression()
```

```
In [54]: ro_lg.score(x_test, y_test)*100
```

```
Out[54]: 88.3495145631068
```

Accuracy of the model is increased, impressive!!

```
In [56]: # y_true is 0
         lg.predict([[19, 19000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[56]: array([0], dtype=int64)
```

```
In [57]: # y_true is 1
         ru_lg.predict([[46, 28000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[57]: array([1], dtype=int64)
```

```
In [58]: # y_true is 1
         ru_lg.predict([[45, 26000]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

```
Out[58]: array([1], dtype=int64)
```

Conclusion:

- All predictions are accurate by even Random Over Sampling
- in case of Random Over Sampling, the model accuracy is also increased from 65% (on imbalanced data) to 88% (on balanced data)

- in case of Random Under Sampling, the model accuracy is also decreased from 65% (on imbalanced data) to 58% (on balanced data)
- However, the model is predicting accurately after making the data balanced by both methods, i.e, Random over sampling, Random under sampling

```
In [ ]:
```