Makesh Srinivasan

Machine learning Lab FAT

I used screenshots because the PDF conversion is not working for my system, sir

Lab FAT: ML

Name: Makesh Srinivasan
Registration number: 19BCE1717
Course code: CSE4020
Faculty: Dr. Abdul Quadir
Slot: L31 + L32
Date: 22-November-2021 Monday

QUESTION 5

Create a program to address the class imbalance problem for a synthetic binary classification dataset. The dataset must have at least 4 features

a.The dataset should have 10,000 samples. Variate the class distribution of the samples to handle the implementation using different methods.

b. Change the sampling strategy by altering the class distribution to see if there is any lift in performance

c. Observe the change by calculating the accuracy. Also, display and plot the transformed dataset.

AIM: Create a program to address the class imbalance problem for a synthetic binary classification dataset. The dataset must have at least 4 features

Part a

Dataset generation:

The dataset should have 10,000 samples. Variate the class distribution of the samples to handle the implementation using different methods.

Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
import plotly.graph_objs as go
from numpy.random import seed
from numpy.random import randint
import plotly
import plotly.io as pio
import plotly.offline as pyo
pio.renderers.default='notebook'
pyo.init_notebook_mode(connected=True)
```

Synthesise data

The seed function makes sure that the creation of the dataset is same for all runs. The question says minimum four features, hence, there are 4 as shown below. The question also says to variate the class distribution of the label (binary classification - only two classes) so I made a difference in the distribution with the ration of class1 to class2 as 1:4

In [2]:
```
seed(1)
feature_1 = randint(0, 10, 10000)
feature_2 = randint(0, 20, 10000)
feature_3 = randint(0, 30, 10000)
feature_4 = randint(0, 40, 10000)
label_c1 = randint(1, 2, 8000)
label_c2 = randint(0, 1, 2000)
label = np.array(list(label_c1) + list(label_c2))
```

In [3]:
```
data = pd.DataFrame({'feature_1':feature_1, 'feature_2':feature_2,'feature_3':feature_3, 'feature_4':feature_4, 'label':label})
data
```

Out[3]:

	feature_1	feature_2	feature_3	feature_4	label
0	5	15	1	3	1
1	8	0	20	12	1
2	9	0	26	17	1
3	5	8	13	30	1
4	0	2	2	27	1
...	...	...	...	...	...
9995	9	18	16	18	0
9996	4	9	7	11	0
9997	5	0	4	28	0
9998	3	8	5	9	0
9999	5	9	26	39	0

10000 rows × 5 columns

The synthetic data is loaded!
There are 4 features, one label with two classes for binary classification

Part B

Change the sampling strategy by altering the class distribution to see if there is any lift in performance

Values of the label - Outcome

In [4]:
```
data.label.value_counts()
```

Out[4]:
```
1    8000
0    2000
Name: label, dtype: int64
```

There are 2 classes. The number of class 1 is 8000 while the number of class 2 is 2000

Splitting x and y:

In [5]:
```
X = data.drop(['label'],axis=1)
y = data['label']
X.head()
```

Out[5]:

	feature_1	feature_2	feature_3	feature_4
0	5	15	1	3
1	8	0	20	12
2	9	0	26	17
3	5	8	13	30
4	0	2	2	27

SOLUTIONS: Sampling strategies on imbalanced dataset

There are three ways to work on imbalanced dataset and they are as follows.

1) No strategy applied
2) Over-sampling
3) Under-sampling
4) SMOTE

I have decided to use KNN to measure the binary classification performance as the ML model is not specified in the question

1) No sampling strategy applied:

Without any appliation of sampling strategy:

In [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 20)
model = KNeighborsClassifier()
model.fit(X_train, y_train)

y_predict = model.predict(X_test)
acc_n = accuracy_score(y_test, y_predict)
print("Accuracy = ", acc_n)
pd.crosstab(y_test, y_predict)
```

```
Accuracy =  0.7545
```

Out[6]:

col_0	0	1
label		
0	25	384
1	107	1484

We get the accuracy as 0.75 with no strategy applied

1) label 0: 25/409 are correct predictions
2) label 1: 1484/1591 are correct predictions

2) Oversampling:

In [7]:
```
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

In [8]:
```
over=RandomOverSampler()
Over_X,Over_y=over.fit_resample(X,y)
```

In [9]:
```
print(Counter(Over_y))
Over_y
```

```
Counter({1: 8000, 0: 8000})
```

Out[9]:
```
0        1
1        1
2        1
3        1
4        1
        ..
15995    0
15996    0
15997    0
15998    0
15999    0
Name: label, Length: 16000, dtype: int64
```

In [10]:
```
train_x, test_x, train_y, test_y = train_test_split(Over_X,Over_y, test_size=0.2, random_state=10)
model = KNeighborsClassifier()
model.fit(train_x,train_y)
y_predict = model.predict(test_x)
over_acc = accuracy_score(test_y,y_predict)
print("Accuracy = ", over_acc)
pd.crosstab(test_y,y_predict)
```

```
Accuracy =  0.67375
```

Accuracy = 0.67375

Out[10]:

col_0	0	1
label		
0	1243	326
1	718	913

We get the accuracy as 0.67 with oversampling strategy applied

1) label 0: 1243/(1243+326) are correct predictions
2) label 1: 913/(913+718) are correct predictions

In [11]:
```
over_data = pd.DataFrame(Over_X)
over_data['label'] = Over_y
over_data
```

Out[11]:

	feature_1	feature_2	feature_3	feature_4	label
0	5	15	1	3	1
1	8	0	20	12	1
2	9	0	26	17	1
3	5	8	13	30	1
4	0	2	2	27	1
...	...	...	...	...	...
15995	2	10	17	2	0
15996	9	2	6	35	0
15997	8	5	10	20	0
15998	2	18	10	24	0
15999	4	11	9	8	0

16000 rows × 5 columns

The dataset is consisting of the four features, one label, and there are 16000 rows. This is more than 10000 because the dataset is oversampled - increased the size of the dataset

3) Undersampling:

In [12]:
```
from imblearn.under_sampling import RandomUnderSampler
```

In [13]:
```
under=RandomUnderSampler()
Under_X,Under_y=under.fit_resample(X,y)
print(Counter(Under_y))
```

```
Counter({0: 2000, 1: 2000})
```

In [14]:
```
Under_y
```

Out[14]:
```
0       0
1       0
2       0
3       0
4       0
       ..
3995    1
3996    1
3997    1
3998    1
3999    1
Name: label, Length: 4000, dtype: int64
```

In [15]:
```
train_x, test_x, train_y, test_y = train_test_split(Under_X,Under_y, test_size=0.2, random_state=10)
model = KNeighborsClassifier()
model.fit(train_x,train_y)
y_predict = model.predict(test_x)

under_acc = accuracy_score(test_y,y_predict)
print("Accuracy = ", under_acc)
pd.crosstab(test_y,y_predict)
```

```
Accuracy =  0.485
```

Out[15]:

col_0	0	1
label		
0	199	217
1	195	189

We get the accuracy as 0.485 with undersampling applied

1) label 0: 199/(199+217) are correct predictions
2) label 1: 189/(195+189) are correct predictions

In [16]:
```
under_data = pd.DataFrame(Under_X)
under_data['label'] = Under_y
under_data
```

Out[16]:

	feature_1	feature_2	feature_3	feature_4	label
0	4	12	1	25	0
1	7	1	0	23	0
2	6	13	26	19	0
3	7	14	28	35	0
4	2	3	6	36	0
...	...	...	...	...	...
3995	3	6	14	16	1
3996	1	5	5	22	1
3997	4	4	4	2	1
3998	5	6	17	6	1
3999	0	5	1	29	1

4000 rows × 5 columns

The dataset is consisting of the four features, one label, and there are 4000 rows. This is less than 10000 because the dataset is udnersampled

4) SMOTE:

In [17]:
```
from imblearn.over_sampling import SMOTE
```

In [18]:
```
smote=SMOTE()
SMOTE_X,SMOTE_y=smote.fit_resample(X,y)
print(Counter(SMOTE_y))
```

```
Counter({1: 8000, 0: 8000})
```

In [19]:
```
SMOTE_y
```

Out[19]:
```
0        1
1        1
2        1
3        1
4        1
        ..
15995    0
15996    0
15997    0
15998    0
15999    0
Name: label, Length: 16000, dtype: int64
```

In [20]:
```
train_x, test_x, train_y, test_y = train_test_split(SMOTE_X,SMOTE_y, test_size=0.2, random_state=20)
model = KNeighborsClassifier()
model.fit(train_x,train_y)
y_predict = model.predict(test_x)

smote_acc = accuracy_score(test_y,y_predict)
print("Accuracy = ", smote_acc)
pd.crosstab(test_y,y_predict)
```

```
Accuracy =  0.73625
```

Out[20]:

Accuracy = 0.73625

col_0	0	1
label		
0	1419	205
1	639	937

We get the accuracy as 0.736 with no strategy applied

1) label 0: 1419/(1419+205) are correct predictions
2) label 1: 937/(937+639) are correct predictions

In [21]:
```
SMOTE_data = pd.DataFrame(SMOTE_X)
SMOTE_data['label'] = SMOTE_y
SMOTE_data
```

Out[21]:

	feature_1	feature_2	feature_3	feature_4	label
0	5	15	1	3	1
1	8	0	20	12	1
2	9	0	26	17	1
3	5	8	13	30	1
4	0	2	2	27	1
...	...	...	...	...	...
15995	0	9	24	14	0
15996	2	6	8	18	0
15997	3	7	17	34	0
15998	6	3	23	25	0
15999	3	16	17	9	0

16000 rows × 5 columns

The size of the dataset is 16000 rows for five cols (4 features and 1 label). The size is greated than the original dimension of 10000x5 because of SMOTE

Part C

Observe the change by calculating the accuracy. Also, display and plot the transformed dataset.

ACCURACY:

In [22]:
```
print("The accuracy of the sampling strategies by altering the class distribution: ")
print("1) With no strategy applied: ", acc_n)
print("2) With Over sampling: ", over_acc)
print("3) With Under sampling: ", under_acc)
print("4) With SMOTE: ", smote_acc)
```

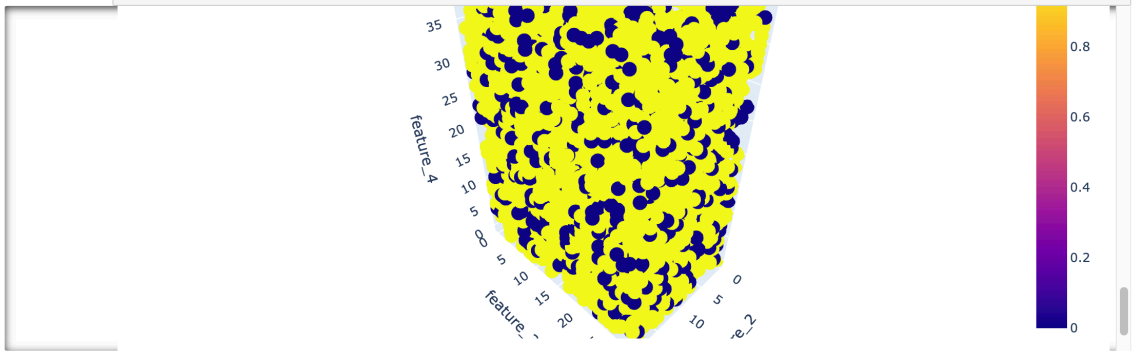
```
The accuracy of the sampling strategies by altering the class distribution:
1) With no strategy applied:  0.7545
2) With Over sampling:  0.67375
3) With Under sampling:  0.485
4) With SMOTE:  0.73625
```

Plotting the transformation:

Plotting the transformation:

The original data:

In [23]:
```
for i in range(1, 5):
    for j in range(i+1,5):
        for k in range(j+1,5):
            var = "feature_"
            vari = var + str(i)
            varj = var + str(j)
            vark = var + str(k)
            fig = px.scatter_3d(data,x=vari,y=varj,z=vark,color='label')
            fig.show()
```

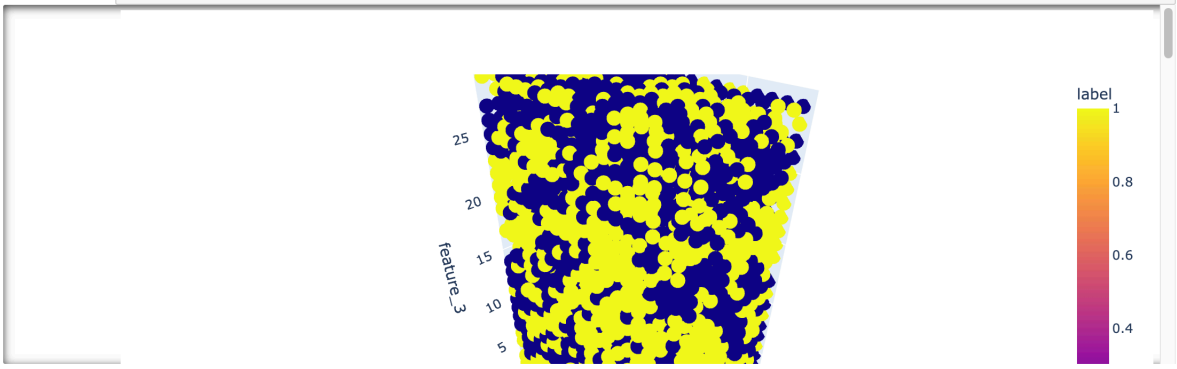


The dataset is not having any meaning, this is an arbitray dataset with random values. Thus, visualising this on the 3d plot above is not very useful especially considering that the number of tuples are 10000. If we use a meaningful dataset such as Iris or Cars we may be able to better visualise the plots and infer meaning from the diagrams. Given the plot above, the only useful conclusion we can draw is that classes are randomly dispersed and the blue points correspond to class 0, and yellow points correspond to class 1.

The transformation for SMOTE is given below.

In [24]:
```
for i in range(1, 5):
    for j in range(i+1,5):
        for k in range(j+1,5):
            var = "feature_"
            vari = var + str(i)
            varj = var + str(j)
            vark = var + str(k)
            fig = px.scatter_3d(SMOTE_data,x=vari,y=varj,z=vark,color='label')
            fig.show()
```

```
fig.show()
```



The plotting of the undersampling and oversampling is not shown below as the plots shown above disappeared when more plotly graphs were added. I have provided the code in comments in case you wish to run and check the plots, sir

Under sampling:

Under sampling:

In [25]:
```
# for i in range(1, 5):
#     for j in range(i+1,5):
#         for k in range(j+1,5):
#             var = "feature_"
#             vari = var + str(i)
#             varj = var + str(j)
#             vark = var + str(k)
#             fig = px.scatter_3d(under_data,x=vari,y=varj,z=vark,color='label')
#             fig.show()
```

Over sampling:

In [26]:
```
# for i in range(1, 5):
#     for j in range(i+1,5):
#         for k in range(j+1,5):
#             var = "feature_"
#             vari = var + str(i)
#             varj = var + str(j)
#             vark = var + str(k)
#             fig = px.scatter_3d(over_data,x=vari,y=varj,z=vark,color='label')
#             fig.show()
```

INFERENCE:

The accuracy of the sampling strategies by altering the class distribution:
With no strategy applied: 0.7545
With Over sampling: 0.67375
With Under sampling: 0.485
With SMOTE: 0.73625

The accuracy values of the above strategies are mentioned. We can see that the best performance is achieved when there is no sampling strategy applied. The worst is observed in case of undersampling. This is because the number of data points available for training the KNN model is reduced. In case of oversampling the performance is slightly better than undersampling but still not as good as SMOTE or "when no strategy applied". This is because the noise alone (along with replication of data is not helpful in increasing the size of the dataset). In SMOTE, the performance measured in terms of accuracy is very close to the no strategy classification. Generally, there is an increase in performance but since the dataset is a synthetically generated data with random entries for each features, there is no meaningful relation between the features and the class/label for a given tuple. Thus, using SMOTE resulted in a slightly reduced accuracy than "no strategy" classification. Nevertheless, it is still close to the performance of the "no strategy" classification. The reason for the drop in accuracy could be due to the increased noise in the dataset that might have caused the training to become less reliable.

We can also observe that the size of the dataset increased for SMOTE and over sampling (16000 rows and 5 cols) while decreased to 4000 (rows) for undersampling from the original of 10000 (rows). The plots are also shown above in terms of scatter and the classes are indicated by the colour. The plots are done in 3D because the notebook only displays limited number of plots at a time and 2D plots take up more space. Moreover, as mentioned before, the plots for under and over sampling are not displayed becasue of plotly display issues. But the code is given in comments which can be run by uncomenting.