IMBALANCED HEALTHCARE DATA HANDLING

A Project Report submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Technology (B.Tech.)

in

Computer Science and Engineering

by

SHIPON NANDY, 191112015001

GAIRIK SAJJAN, 191112015014

SANDIP DAS, 191112015029

SAYANTAN BANERJEE, 191112015030

4th Semester

under the supervision of

Associate professor

Dr. NILANJAN DEY

Department of Computer Science and Engineering
JIS University, Kolkata
2021

Table of Contents

TOPIC PAGE NO.

1. ABSTRACT:

A considerable amount of health record data has been stored due to recent advances in the digitalization of medical systems. However, it is not always easy to analyze Health record data, as medical datasets are not well balanced in their class sets. In healthcare datasets to identify rare diseases in medical diagnostics etc. usually high-risk patients are tended to be in the minority class. For such reason traditional classification methods become biased towards majority class as they consider the minority class as noise. Thus, the model become useless. To build a good Machine Learning model a well-balanced dataset is very Important. Using the imblearn library from python we'll try over-sampling and under-sampling to balance the class in the healthcare dataset. And try to develop a better model which will give us a better accuracy than traditional models without considering the relative distribution of each class in the dataset.

2. OBJECTIVE:

Solving the problem related to class imbalance in healthcare datasets and building a prediction model with better accuracy**.**

3. INTRODUCTION:

An imbalanced dataset means instances of one of the two classes is higher than the other, in another way, the number of observations is not the same for all the classes in a classification dataset. This problem is faced not only in the binary class data but also in the multi-class data.

For example, suppose we're building a classifier to classify malignant and benign tumor we'll likely have 10,000 benign tumors for every 1 malignant tumor, that's quite an imbalance.

Feeding imbalanced data to any classifier can make it biased in favor of the majority class, simply because it did not have enough data to learn about the minority.

For example, from the above-mentioned imbalanced dataset the prediction model will always be biased towards the benign tumors as it is the majority class, the features of the malignant data may be treated as noise and often ignored as it is the minority class. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

A classifier which achieves an accuracy of 99 % with the imbalanced data is not accurate, if it classifies all instances as the majority class and eliminates the 1 % minority class observations as noise.

There is no exception of balanced dataset to make a good training set. The more the dataset is balanced we get more accurate prediction in terms of input. But most of the data available today is not well balanced. By balance we mean the difference in the numbers of the data in minority class and majority class and the ratio is significantly noticeable. As a result, the model built from the data is also inaccurate.
Medical datasets are not well balanced in their class sets as the cases of disease are rarer than the normal population. However, in the datasets, high risk patients are tended to be in the minority class. So existing classification methods perform poorly on minority class. Therefore, we need a good sampling technique to resolve those issues with imbalanced datasets.

Different sampling techniques are used to overcome the class imbalance problem by either eliminating some data from the majority class which is known as under-sampling or adding some synthetic data (artificially generated) to the minority class which is known as over-sampling. But there are many advantages and disadvantages. The disadvantage of under-sampling is it may ignore many important data from the majority class which leads to inappropriate prediction. But as it decreases the size of dataset it gives an advantage in terms of time and memory complexity. Where oversampling technique increases the minority class data in the training set. This leads to increase the size of dataset. And we need to be aware about the overfitting the data. Synthetic Minority over-sampling technique which is known as 'SMOTE' is a well-known over-sampling technique. It is a synthetic data generation over-sampling technique which generates synthetic minority class samples.

For now, we're trying to find a way to develop a new model that gives better performance than the previously available model using SMOTE or other over-sampling and under-sampling techniques like Random Under Sampling and Clustering Centroid.

4. TECHNIQUES TO HANDLE IMBALANCED DATA:

There are several ways to handle imbalanced dataset.

4.1 OVER-SAMPLING (SMOTE)

This technique is used to modify the unequal data classes to create balanced datasets.
SMOTE stands for 'Synthetic Minority Over Sampling Technique' which provides a technique to balance the classes in dataset by generating synthesized data for minority class from those that already exist.

When the quantity of data is insufficient, the oversampling method tries to balance by incrementing the size of rare samples.
It works randomly picking a point from the minority class and compute the k-nearest neighbours for the point. The synthetic points are added between the chosen point and its neighbours.

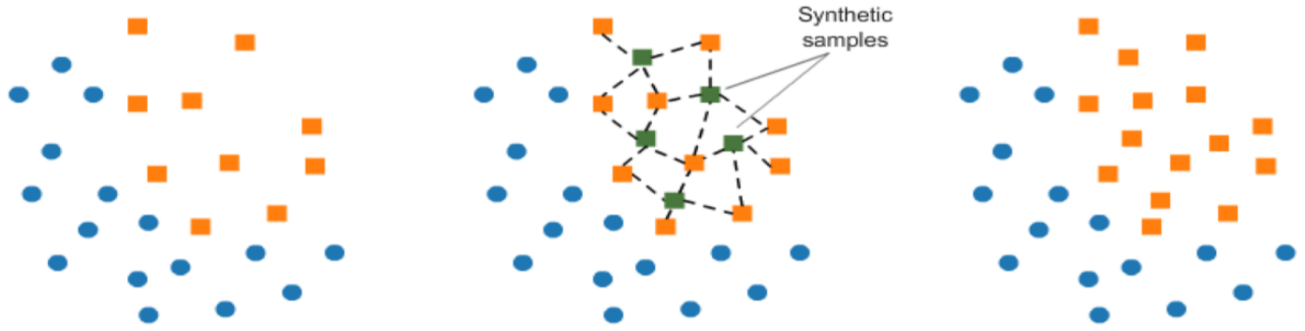
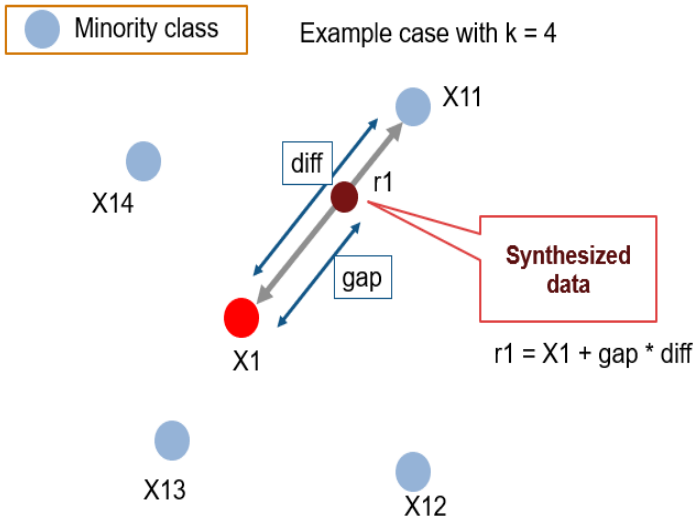


Fig: Blue = 'Majority', Yellow = 'Minority'
Green = 'Synthesized Data using K-Nearest neighbours'

WORKING PRINCIPLES:

- Find the k-nearest neighbours for each sample.
- Select samples randomly from a k-nearest neighbour.
- Find the new samples = original samples + difference * gap (0,1).
- Add new samples to the minority. Finally, a new dataset is created.



ADVANTAGES OF SMOTE -

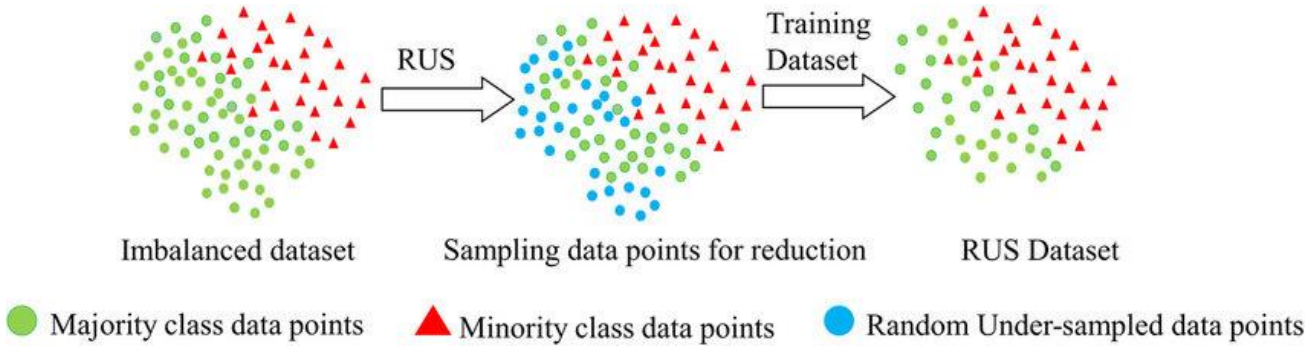
- Alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances.
- No loss of information.
- It's simple to implement and interpret.

DISADVANTAGES OF SMOTE -

- While generating synthetic examples, SMOTE does not take into consideration neighboring examples can be from other classes. This can increase the overlapping of classes and can introduce additional noise.
- SMOTE is not very practical for high dimensional data.

4.2 UNDER-SAMPLING

This resampling idea provides a naïve technique in rebalancing the class distribution for an imbalanced dataset.
It deletes random data from majority class thus, it makes the dataset smaller and balanced. That is why it is called under-sampling technique.



ADVANTAGES OF RANDOM UNDER-SAMPLING -

- Because of random selection we don't have to make decisions on which points are important and which are not: we simply let the random process do the work. Several studies have shown that random selection performs as well as, if not better than, processes where deliberate removal choices are made.

DISADVANTAGES OF RANDOM UNDER-SAMPLING -

- The process could remove important members.
- Problems tend to result in data that is non-smooth, has boundaries or small features

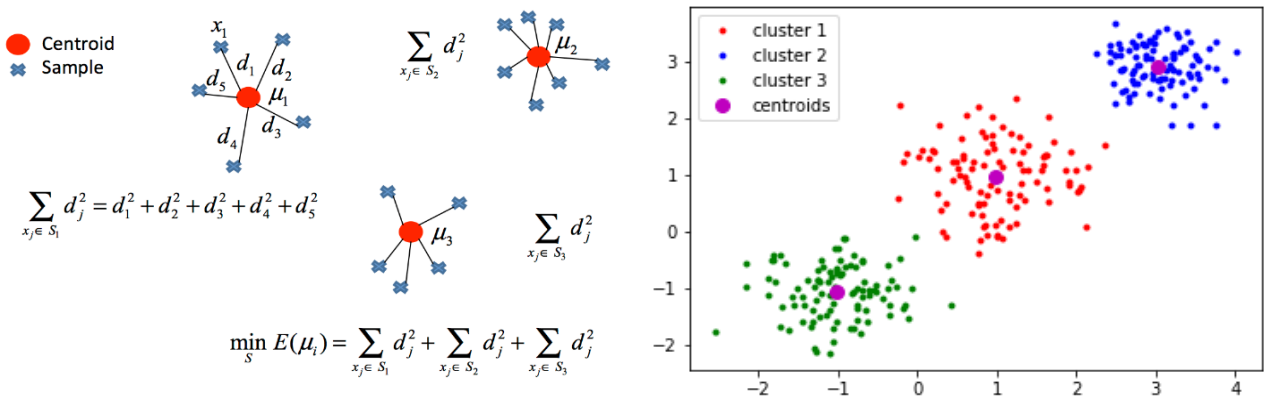
4.3 CLUSTERING CENTROIDS:

It is an under-sampling technique that uses cluster to balance the classes in the dataset. Like, we can cluster the records of the majority class, and do the

under-sampling by removing records from each cluster, thus seeking to preserve information.

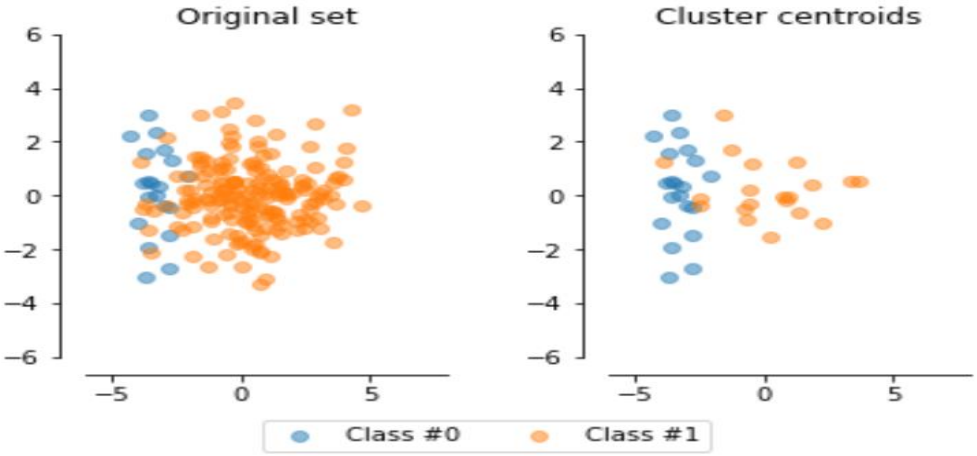
Given an original data set S, prototype generation algorithms will generate a new set S' where |S'| < |S| and S' ⊄ S. Prototype generation technique will reduce the number of samples in the targeted classes but the remaining samples are generated — and not selected — from the original set.

It makes use of K-means to reduce the number of samples. Prototype selection algorithms will select samples from the original set S. therefore, S' is defined as |S'| < |S| and S' ϵ S.



WORKING PRINCIPLES:

- Select k points at random as centroids/cluster centres.
- Assign data points to the closest cluster based on Euclidean distance.
- Calculate centroid of all points within the cluster Repeat iteratively till convergence. (Same points are assigned to the clusters in consecutive iterations)



ADVANTAGE OF CLUSTERING CENTROID:

- It is a controlled algorithm level approach unlike random under sampling. It uses K-means clustering to reduce the majority class.

DISADVANTAGE OF CLUSTERING CENTROID:

- It is sensitive to outliers.
- Uniform effect: Often produce clusters with relatively uniform size even if the input data have different cluster size.

5. PROPOSED METHODS:

5.0.1 IMBALANCE DATASET:

```
In [9]: X = data.drop(['stroke'], 1)
        y = data.stroke
```

We've set X as our independent data and y as our dependent data.

```
In [10]: X
```

Out[10]:

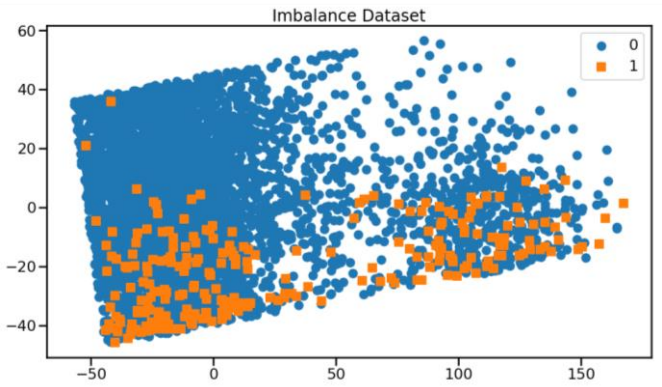
	age	hypertension	heart_disease	avg_glucose_level	bmi
0	66	0	0	76.46	21.2
1	57	0	0	197.28	34.5
2	68	0	0	233.94	42.4
3	68	1	1	247.51	40.5
4	57	0	0	84.96	36.7
...	...	...	...	...	...
195	64	0	0	82.34	31.9
196	62	0	0	180.63	31.8
197	2	0	0	92.48	18.0
198	53	0	0	116.66	28.5
199	65	1	0	112.09	29.5

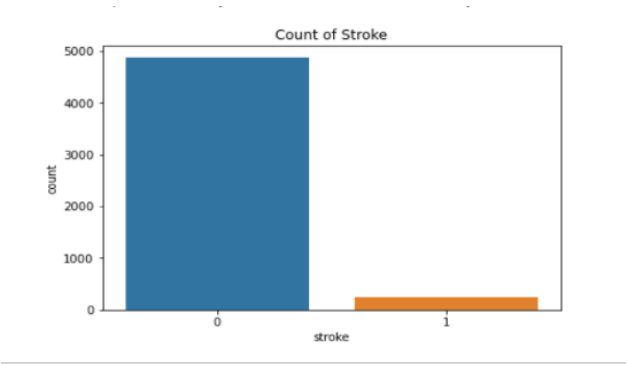
200 rows × 5 columns

```
In [58]: y
```

```
Out[58]: 0      1
         1      1
         2      1
         3      1
         4      1
               ..
         195    0
         196    0
         197    0
         198    0
         199    0
         Name: stroke, Length: 200, dtype: int64
```

The class imbalance in our dataset is showed in the below scatter plot and bar plot.





5.0.2 PYHTON LIBRARY TO BALANCE DATASET:

We're proposing some techniques that uses oversampling and under-sampling to resolve the issues of class distribution in imbalanced datasets. We'll use the Python library imbalanced-learn. It is compatible with scikit-learn and is part of scikit-learn-contrib project.

```
In [8]: import imblearn
```

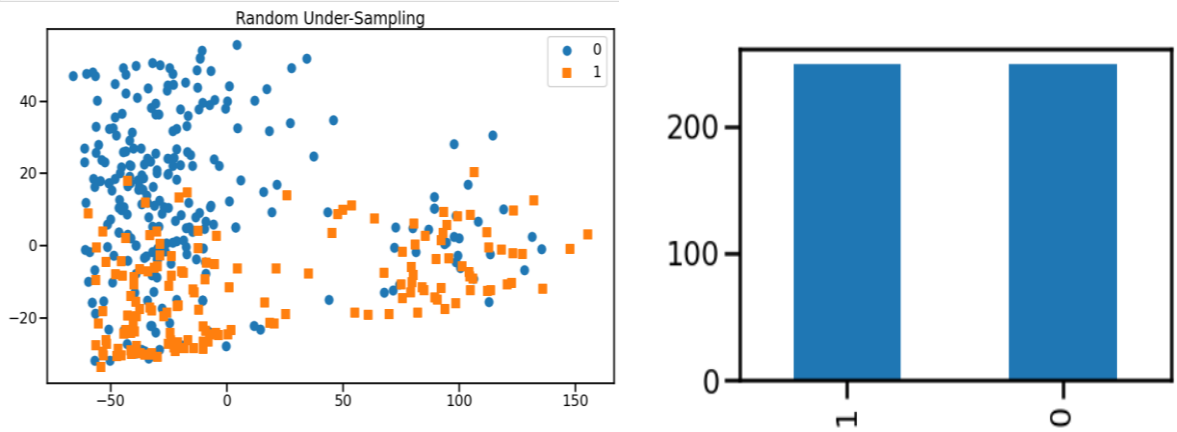
5.1 RANDOM UNDER-SAMPLING:

```
In [21]: from imblearn.under_sampling import RandomUnderSampler

         rus = RandomUnderSampler(random_state = 33, replacement = True)
```

We've imported the 'RandomUnderSampler' from 'imblearn.under_sampling' and declared an object 'rus' for the sampling function. Sampling strategy is set as default.

```
In [24]: # fit predictor and target variable
         X_rus, y_rus = rus.fit_resample(X, y)
```

To resample the data, we've taken the X and y as argument in the resample function and saved the resampled data in X_rus and y_rus. Thus, we get the imbalnced class as balanced.



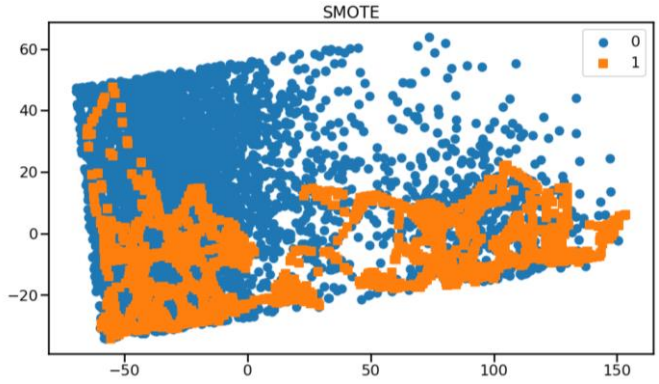
As we see from the above figure the data is balanced.

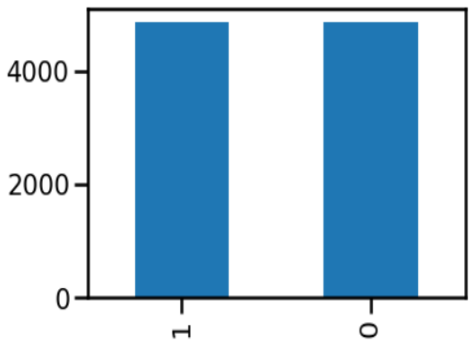
5.2 SMOTE:

```
In [35]: from imblearn.over_sampling import SMOTE

         sm  = SMOTE(random_state = 33, sampling_strategy='auto',k_neighbors=4)

In [36]: X_sm, y_sm = sm.fit_sample(X, y)
```

We've imported the 'SMOTE' from 'imblearn.over_sampling' and declared an object 'sm' for the sampling function. Sampling strategy is set as minority as we're creating synthetic data for minority class.

To resample the data, we've taken the X and y as argument in the resample function and saved the resampled data in X_sm and y_sm. Thus, we get the imbalanced class as balanced.





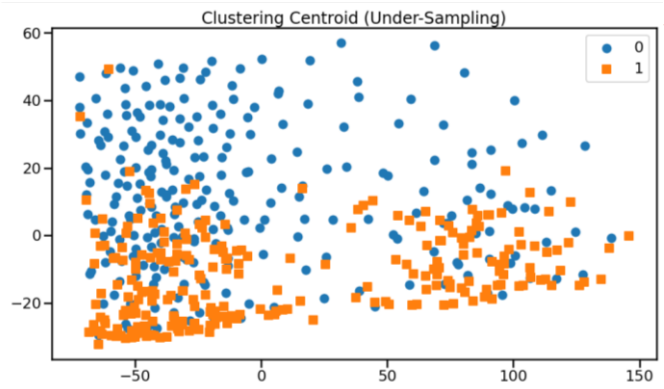
As we see from the above figure the data is balanced.

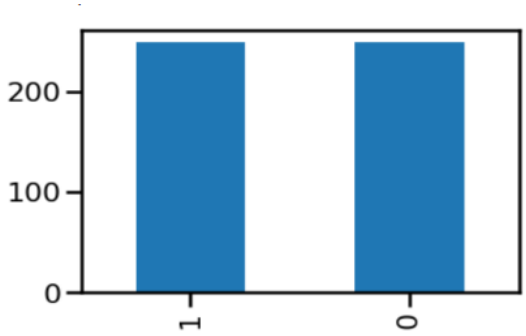
5.3 CLUSTERING CENTROIDS:

```
In [44]: from imblearn.under_sampling import ClusterCentroids

In [45]: cc = ClusterCentroids(sampling_strategy='majority', random_state=42, n_jobs= -1)
         X_cc, y_cc = cc.fit_sample(X, y)
```

We've imported the 'ClusteringCentroids' from 'imblearn.under_sampling' and declared an object 'cc' for the sampling function. Sampling strategy is set as 'majority' as we're under-sampling the majority class using cluster-based centroid.

To resample the data, we've taken the X and y as argument in the resample function and saved the resampled data in X_cc and y_cc. Thus, we get the imbalanced class as balanced.





6. REVIEW OF PREVIOUS YEARS RESEARCH PAPER:

Typically, real world data are usually imbalanced. And in terms of medical data high risk patients always tend to be in the minority class. So, while we are talking about the digitalization of medical systems, we have to build a model that is overall more accurate without considering the relative distribution of each class in the dataset.

In the paper (i) of M. Mostafizur Rahman and D. N. Davis which was published on 2013, titled as 'Addressing the Class Imbalance Problem in Medical dataset'; the authors discussed several techniques of under-sampling and over-sampling. They've proposed a better technique called Modified Cluster based Under-Sampling which can both balance the data and create a good quality training set.

In the paper (ii) of Joffrey L. Leevy, Taghi M. Khoshgoftaar, Richard A. Bauder, Naeem Seliya which published on 2018, titled as ' A survey on addressing high-class imbalance in big data-2018'; the authors have discussed about previous 8 years of research on high class imbalanced data (minority to majority class ratio between 100:1 and 10,000:1). They focused mainly on big data, which consists of 100,000 data. They found that Random Over Sampling yields better performance than Random Under Sampling or the SMOTE (Synthetic Minority Over-Sampling Technique).

A paper (iii) has been published titled as 'A Survey on Methods to Handle Imbalance Dataset by Apurva Sonak and R.A.Patankar. In this paper the authors have applied Random Under-Sampling, SMOTE oversampling and Cost Sensitive Learning on

different datasets that are relatively large and small. They have come to some conclusions that which technique works efficiently on different sizes of data. They have also pointed out some disadvantages or drawbacks of the techniques used in this paper.

A research paper (iv) on imbalanced dataset has been published on 2006 by Sotiris Kotsiantis, Dimitris Kanellopoulos, titled as 'Handling Imbalanced Dataset'. The authors of this paper have provided a review on handling imbalanced dataset. They have applied different data level approaches like, Random Under Sampling, SMOTE oversampling, and feature selection for imbalanced dataset. The Algorithm Level approaches used in this paper are Threshold method, One-class learning, and Cost Sensitive Learning.

7. IMPLEMENTATION DETAILS

7.1 SYSTEM INFORMATION

We've implemented Random Under Sampling, SMOTE and Under-Sampling using Clustering Centroid in 4 different machines. The configuration about those 4 machines and versions of different environments/platforms is given below in the table.

	OS	PROCESSOR	RAM	GPU	No. of CPU	Python	Anaconda	Platform
System 1	Windows 10	RYZEN 5 (4600)	8 GB	GTX 1650ti	6	3.8.5	3	Jupyter Notebook
System 2	Windows 10	INTEL i5 (9^{th} GEN.)	8 GB	GTX 1650ti	4	3.9.5	3	Jupyter Notebook
System 3	Windows 10	AMD A4 (7^{th} GEN.)	8 GB	AMD RADEON R3	4	3.8.2	3	Jupyter Notebook
System 4	Windows 10	INTEL i5 (5^{th} GEN.)	8 GB	GEFORCE 940mx (4 GB)	4	3.8.5	3	Jupyter Notebook

7.2 DATASET

7.2.1 SOURCE DATA

We are using a dataset that is imbalanced in its class level. We've performed Under-Sampling, SMOTE, Under-Sampling using Clustering Centroid techniques on the dataset.

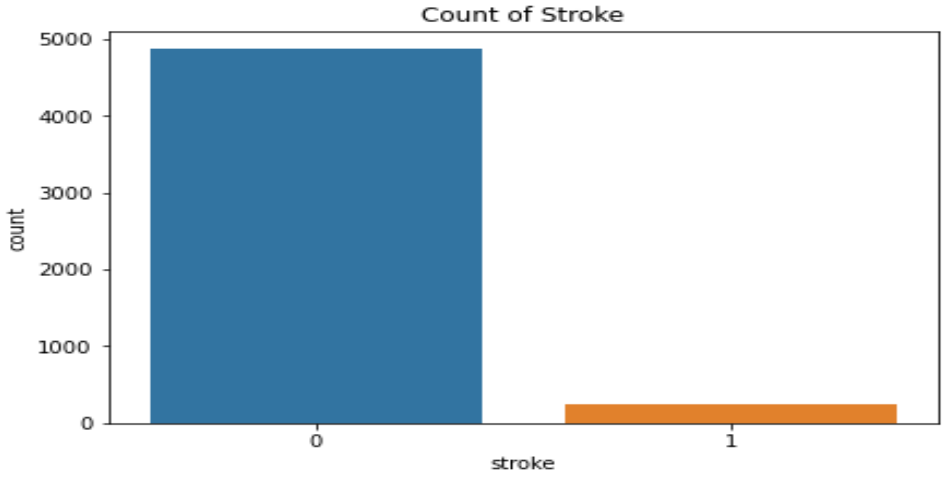
In the below table we've given all the attributes of our source dataset named "STROKE".

No. of Samples	No. of features	Target	No. of Class 2	No. of 0s' (Majority)	No. of 1s' (Minority)
5101	6	1	2	4861	249

Out[13]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
0	9046	67	0	1	228.69	36.6	1
1	51676	61	0	0	202.21	28.1	1
2	31112	80	0	1	105.92	32.5	1
3	60182	49	0	0	171.23	34.4	1
4	1665	79	1	0	174.12	24.0	1

The above figure shows the head of our source data. We're working with this "stroke" dataset where column 'stroke' is 'dependent' and the other columns are 'independent'. We'll predict that who can get a stroke based on the independent variables.



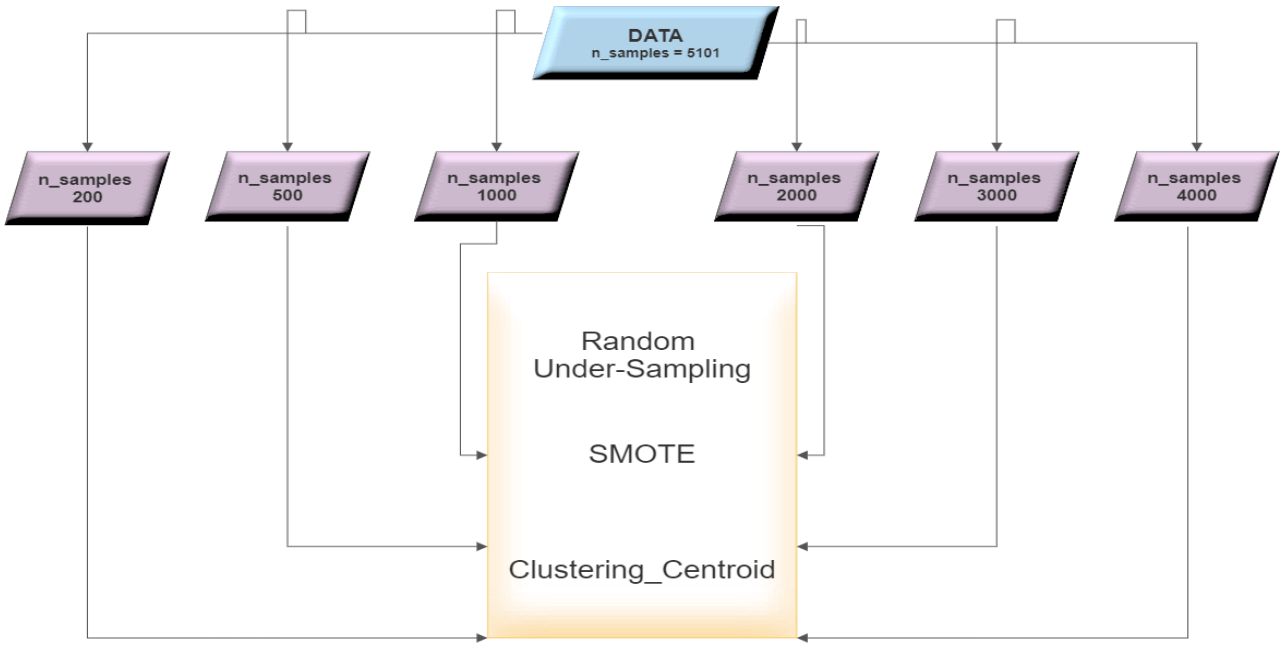
From the above figure we can see that in stroke data the ratio of 0 and 1 are too imbalanced and we've developed a prediction model, we'll balance the data with proposed techniques so that the model won't be biased towards majority class 0.

7.2.2 SYNTHETIC DATA

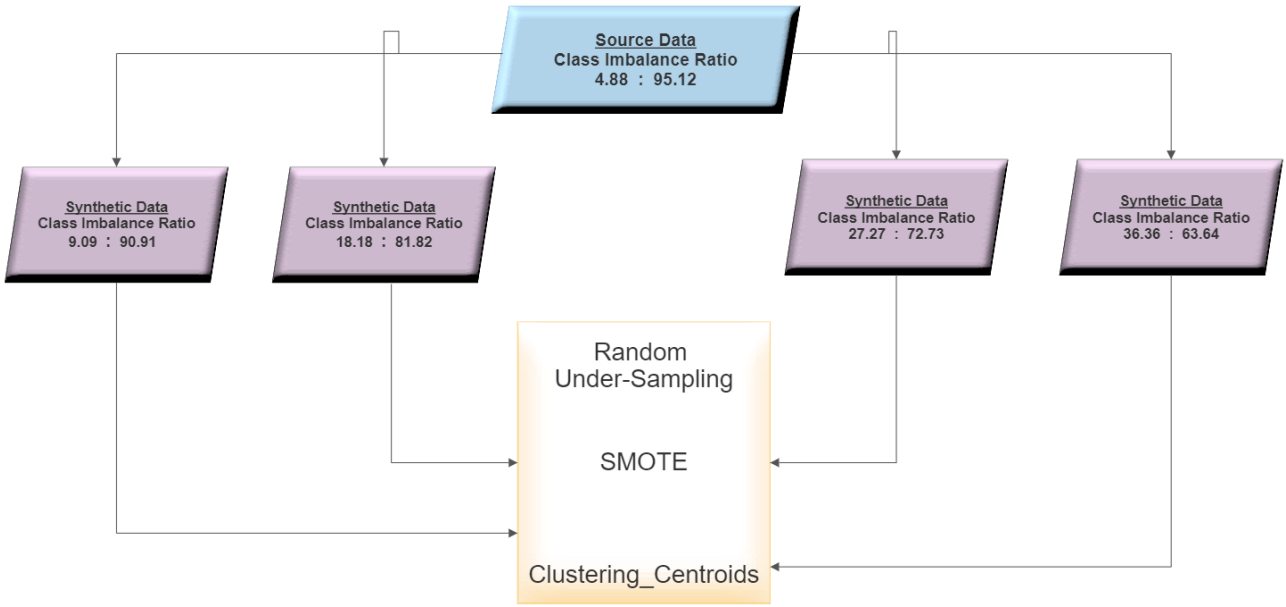
We've created 6 synthetic datasets and applied those 3 techniques proposed techniques on them.

No. of Samples	No. of features	Target	No. of Class 2	No. of 0s' (Majority)	No. of 1s' (Minority)	Ratio of 0 & 1
200	6	1	2	191	9	95.5 : 4.5
500	6	1	2	476	24	95.2 : 4.8
1000	6	1	2	952	48	95.2 : 4.8
2000	6	1	2	1903	97	95.2 : 4.8
3000	6	1	2	2854	146	95.2 : 4.8
4000	6	1	2	3805	195	95.2 : 4.8

We've implemented those 3 techniques on the above datasets to understand the effect of imbalance dataset on the size of data.



We've also created 4 synthetic datasets and applied those proposed techniques on them to see the effect on imbalance dataset for variable imbalance of minority class.



The attributes of those datasets are given in the below table:

No. of Samples	No. of features	Target	No. of Class 2	No. of 0s' (Majority)	No. of 1s' (Minority)
660	6	1	2	600	60
720	6	1	2	600	120
780	6	1	2	600	180
840	6	1	2	600	240

7.3 CLASSIFIER USED

We've used **Logistic Regression** as the classifier.

```
In [ ]: from sklearn.linear_model import LogisticRegression
        clf = LogisticRegression()
```

Logistic regression is a statistical method that finds an equation that predicts an outcome for a binary variable, *Y,* from one or more response variables, *X*. To predict group membership, it uses the log odds ratio rather than probabilities and an iterative maximum likelihood method rather than a least squares to fit the final model. This means the researcher has more freedom when using Logistic Regression and the method may be more appropriate for abnormally distributed data or when the samples have unequal covariance matrices.

Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modelled is a binary value (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

$$y = e^{(b0 + b1*x)} / (1 + e^{(b0 + b1*x)})$$

Where e is the base of the natural logarithms, y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

$$y = 0 \text{ if } < 0.5$$

$$y = 1 \text{ if } >= 0.5$$

In binary logistic regression analysis, it is essential that the categories of dependent variable should be encoded as 0 and 1 in the analysis. In our case, 0 shows low possibility of having stroke and 1 means high possibility of having stroke. Accordingly, the coefficients obtained reflect the effects of the variables of age, hypertension, heart disease, average glucose level Body Mass Index(bmi) on the possibility of having stroke, since the category encoded as 1 show “high” possibility of having stroke.

7.4 LIBRARIES USED

We’ve used python as language and Jupyter Notebook as our platform to develop the prediction model.

We've used different libraries and they are listed below.

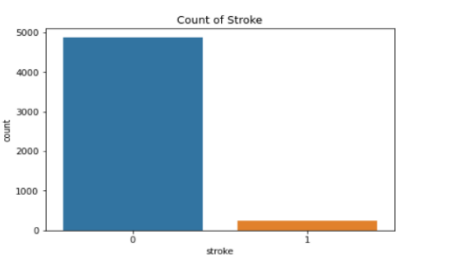
```
In [16]: import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from imblearn.under_sampling import RandomUnderSampler
         from imblearn.over_sampling import SMOTE
         from imblearn.under_sampling import ClusterCentroids
         from sklearn import metrics
         from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score, f1_score, plot_confusion_matrix
         from sklearn.decomposition import PCA
```

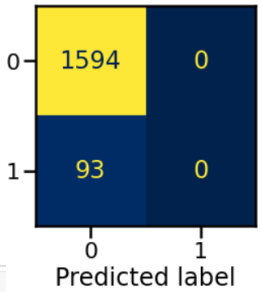
8. RESULTS AND DISCUSSIONS

8.1 USING LOGISTIC REGRESSION

We've performed simple Logistic Regression on the 'stroke' dataset to see the performance of the prediction model.

Though we get a high accuracy like 94%, but this model can be a disaster. As it neglects the minority class and produces a biased result.






```
In [13]: print('Accuracy Score for Testing Dataset = ',accuracy_score(y_test, test_pred))
         print('Accuracy Score for Training Dataset = ',accuracy_score(y_train, train_pred))

         Accuracy Score for Testing Dataset =  0.9448725548831061
         Accuracy Score for Training Dataset =  0.9544259421560035
```

Here, from the confusion matrix we can see that in the 'Testing Dataset' it has predicted 1594 as true negative

And 93 as false negative.

.

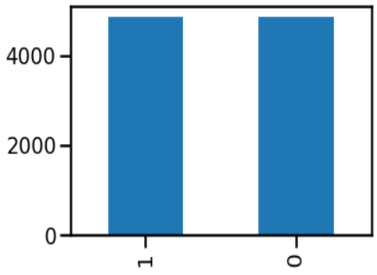
Here, we can see the precision, recall and f1-score. We can clearly see that how poor it has performed regarding the minority class.

```
In [19]: print(metrics.classification_report(y_true = y_test, y_pred = test_pred))
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	1594
1	0.00	0.00	0.00	93
accuracy			0.94	1687
macro avg	0.47	0.50	0.49	1687
weighted avg	0.89	0.94	0.92	1687

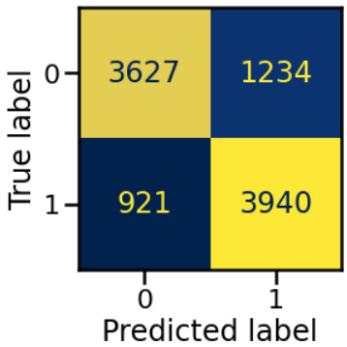
8.2 USING SMOTE:

Using SMOTE we've over-sample the minority class to balance the dataset.
In the figure we can see the ratio of 1 & 0 are same after applying SMOTE.
We can see that the accuracy has decreased but we shouldn't only consider the accuracy when we're designing a prediction model.




```
In [37]: print('Accuracy Score for Testing Dataset = ',accuracy_score(y_test, test_pred_sm))
         print('Accuracy Score for Training Dataset = ',accuracy_score(y_sm, train_pred_sm))

         Accuracy Score for Testing Dataset =  0.7696658097686375
         Accuracy Score for Training Dataset =  0.7783377905780704
```

As now we can calculate the TPR & FPR from that confusion matrix. Here,
True Negative = 3627
False Positive = 1234
False Negative = 921
True Positive = 3940



We can see the difference in precision, recall and f1-score after applying SMOTE.

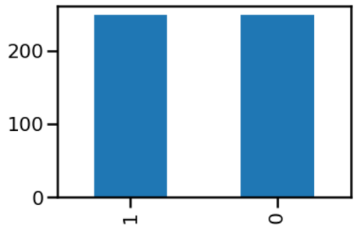
```
In [42]: print(metrics.classification_report(y_true = y_sm, y_pred = train_pred_sm))
```

	precision	recall	f1-score	support
0	0.80	0.75	0.77	4861
1	0.76	0.81	0.79	4861
accuracy			0.78	9722
macro avg	0.78	0.78	0.78	9722
weighted avg	0.78	0.78	0.78	9722

8.3 USING RANDOM UNDER-SAMPLING:

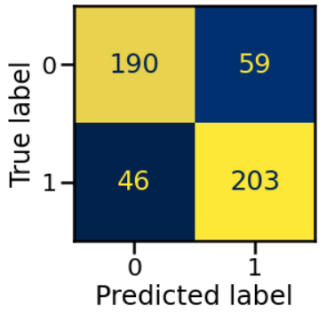
Here, we've under-sampled the majority class to balance the ratio of 0 & 1.

Now, both the counts for 0 and 1 = 249.




```
In [56]: print('Accuracy Score for Testing Dataset = ', accuracy_score(y_test, test_pred_rus))
         print('Accuracy Score for Training Dataset = ', accuracy_score(y_rus, train_pred_rus))

         Accuracy Score for Testing Dataset =  0.73
         Accuracy Score for Training Dataset =  0.7891566265060241
```

As now we can calculate the TPR & FPR from that confusion matrix. Here,
True Negative = 190
False Positive = 59
False Negative = 46
True Positive = 203

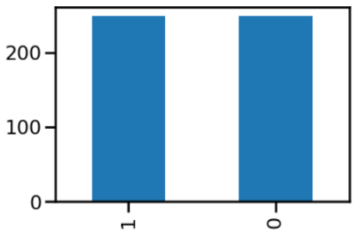


We can see the difference in precision, recall and f1-score after applying random under-sampling.

```
In [30]: print(metrics.classification_report(y_true = y_rus, y_pred = train_pred_rus))

                    precision    recall  f1-score   support

                 0       0.81      0.76      0.78       249
                 1       0.77      0.82      0.79       249

          accuracy                           0.79       498
         macro avg       0.79      0.79      0.79       498
      weighted avg       0.79      0.79      0.79       498
```

8.4 CLUSTERING CENTROIDS (UNDER-SAMPLING)

Here, we've under-sampled the majority class through 'clustering centroids' to balance the ratio of 0 & 1.
Now, both the counts for 0 and 1 = 249




```
In [51]: print('Accuracy Score for Testing Dataset = ', accuracy_score(y_test, test_pred_cc))
         print('Accuracy Score for Training Dataset = ', accuracy_score(y_cc, train_pred_cc))

         Accuracy Score for Testing Dataset =  0.75
         Accuracy Score for Training Dataset =  0.7751004016064257
```

As now we can calculate the TPR & FPR from that confusion matrix. Here,
True Negative = 197
False Positive = 52
False Negative = 61
True Positive = 188

We can see the difference in precision, recall and f1-score after applying 'Clustering-Centroids' under-sampling.

```
In [52]: print(metrics.classification_report(y_true = y_cc, y_pred = train_pred_cc))
```

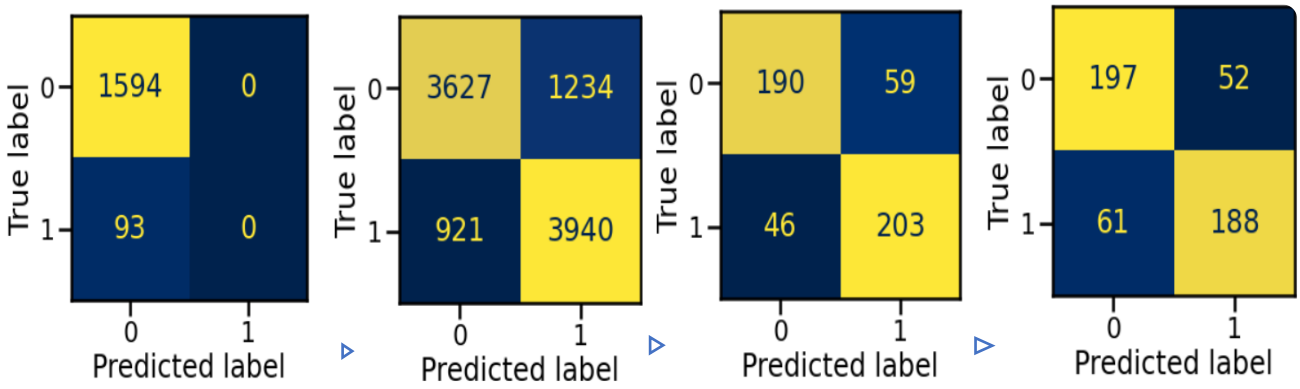
	precision	recall	f1-score	support
0	0.77	0.79	0.78	249
1	0.78	0.76	0.77	249
accuracy			0.78	498
macro avg	0.78	0.78	0.78	498
weighted avg	0.78	0.78	0.78	498

8.5 DISCUSSION

8.5.1 TABLE TO SHOW DIFFERENCE AMONG 4-TECHNIQUES:

Techniques→ Scores↓	Without Sampling	SMOTE	Random Under-Sampling	Clustering Centroid (Under-Sampling)
Accuracy (Testing)	0.95	0.77	0.73	0.75
Accuracy (Training)	0.95	0.78	0.79	0.77
Precision	0 = 0.95 1 = 0.00	0 = 0.80 1 = 0.76	0 = 0.81 1 = 0.77	0 = 0.77 1 = 0.78
Recall	0 = 1.00 1 = 0.00	0 = 0.75 1 = 0.81	0 = 0.76 1 = 0.82	0 = 0.79 1 = 0.76
F1-score	0 = 0.98 1 = 0.00	0 = 0.77 1 = 0.79	0 = 0.78 1 = 0.79	0 = 0.78 1 = 0.77

8.5.2 DIFFERENCE IN CONFUSION MATRICES AMONG 4-TECHNIQUES:



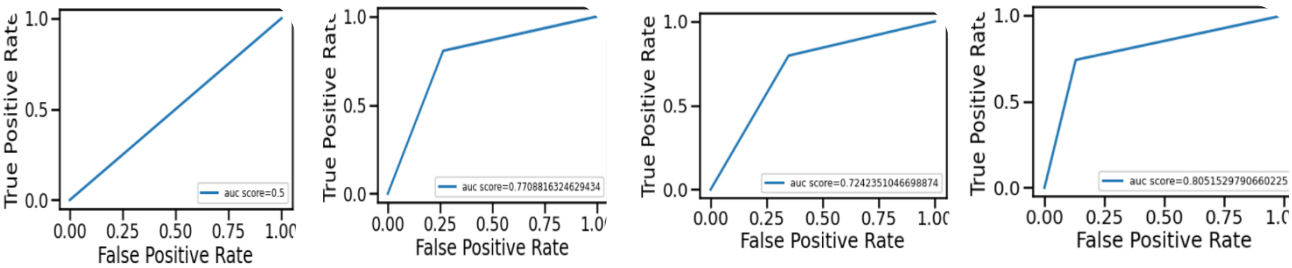
Without any Sampling

SMOTE
TPR = 81.1%
FPR = 25.4%

Random Under-Sampling
TPR = 81.5%
FPR = 24.7%

Clustering Centroid (Under-Sampling)
TPR = 75.5%
FPR = 20.9%

8.5.3 DIFFERENCE IN AUC SCORE AND ROC CURVE AMONG 4-TECHNIQUES:



Without any Sampling

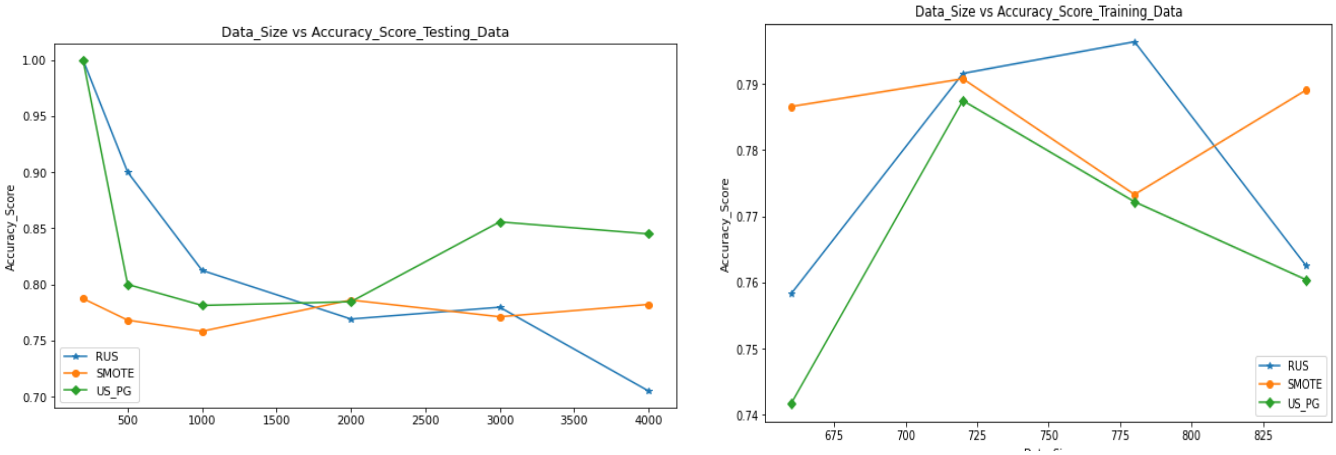
SMOTE

Random Under-Sampling

Clustering Centroids (Under-Sampling)

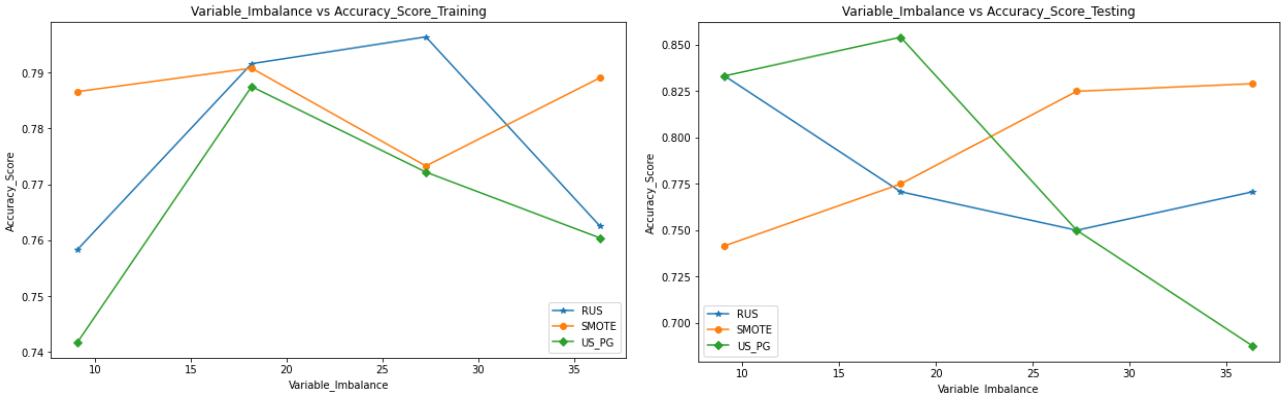
The higher the AUC, **the better the performance of the model at distinguishing between the positive and negative** classes. When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly.

8.6 EFFECT OF DATA SIZE ON IMBALANCE DATASET:



Data Size vs Accuracy Score (using synthetic data)

8.7 EFFECT OF VARIABLE IMBALANCE OF CLASS ON IMBALANCE DATASET:



Variable Imbalance in Class vs Accuracy Score
(Using Synthetic Data)

9. CONCLUSION

The data we've used in our project is an online data collected from google. Because of the shortage of medical data and the recent unavoidable circumstances of pandemic and lockdown, collecting field data was not possible. And online medical data which are imbalanced in their class is very rare. In future, when the situation becomes a little better, we plan to collect the data ourselves, either in person or from some healthcare institutes, so

the study becomes a bit more reliable. We are also planning to apply some new methods to our model or build a new imbalanced data learning technique, that yields more accurate results.

10. REFERENCES:

I. **http://www.ijmlc.org/papers/307-K0020.pdf**

II. **https://journalofbigdata.springeropen.com/articles/10.1186/s40537-018-0151-6**

III. **https://www.google.com/url?sa=t&source=web&rct=j&url=https://ijcsmc.com/docs/papers/November2015/V4I11201573.pdf&ved=2ahUKEwjft_Gk4YXxAhWUXisKHeuZBo8QFjABegQIBBAG&usg=AOvVaw3VMg0mUsNhAAvGIh9eGvYD&cshid=1623077137471**

IV. **https://www.researchgate.net/publication/228084509_Handling_imbalanced_datasets_A_review**

V. **https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html**

VI. **https://imbalanced-learn.org/stable/over_sampling.html**

VII. **http://scikit-learn.github.io/stable**

VIII. **https://medium.com/analytics-vidhya/balance-your-data-using-smote-98e4d79fcddb**

IX. **https://towardsdatascience.com/fraud-detection-with-cost-sensitive-machine-learning-24b8760d35d9**

X. **https://analyticsindiamag.com/5-important-techniques-to-process-imbalanced-data-in-machine-learning/**

XI. **https://www.hindawi.com/journals/jhe/2018/6275435/**

XII. **https://dl.acm.org/doi/fullHtml/10.1145/3357253**

XIII. **https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7248318/**

XIV. **https://www.mdpi.com/2076-3417/9/24/5287/htm**

XV. https://link.springer.com/article/10.1007/s13748-016-0094-0

XVI. https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/

XVII. https://www.wovenware.com/blog/2020/01/the-problems-with-imbalanced-datasets-and-how-to-solve-them/#.YImJqrfhUzY

XVIII. https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/

XIX. https://www.ijstr.org/final-print/mar2017/A-Review-Of-Synthetic-Data-Generation-Methods-For-Privacy-Preserving-Data-Publishing.pdf

XX. https://stackabuse.com/k-means-clustering-with-scikit-learn

XXI. https://www.intechopen.com/books/recent-trends-in-computational-intelligence/classification-problem-in-imbalanced-datasets

XXII. https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/

XXIII. https://imbalanced-learn.org/stable/under_sampling.html

XXIV. https://www.unioviedo.es/compnum/labs/new/kmeans.html