Generation of Synthetic examples for imbalanced tabular data

MSc Research Project
Data Analytics

Nirav Bharat Gala
Student ID: X21125261

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nirav Bharat Gala
Student ID:	X21125261
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	15/12/2022
Project Title:	Generation of Synthetic examples for imbalanced tabular data
Word Count:	557
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	1st February 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Generation of Synthetic examples for imbalanced tabular data

Nirav Bharat Gala
X21125261

1 Hardware Requirements

The hardware used for this research study is an Apple Macbook air laptop with 8Gb ram and MacOS operating system as shown in the figure.

Hardware Overview:

Model Name:	MacBook Air
Model Identifier:	Mac14,2
Chip:	Apple M2
Total Number of Cores:	8 (4 performance and 4 efficiency)
Memory:	8 GB
System Firmware Version:	7459.141.1
OS Loader Version:	7459.141.1
Serial Number (system):	R99X92GYQ2
Hardware UUID:	19BFD729-5A4C-5FB4-9D91-32977D9D88A8
Provisioning UDID:	00008112-001A445A1AF1401E
Activation Lock Status:	Disabled

Figure 1: Hardware Requirements

2 Software Requirements

The entire implementation of this research project was done in Google collaboratory using Python Programming language.As shown in the figure 2, Google Collab is browser based service to create and execute notebook with python and there is no need of installation on your computer.

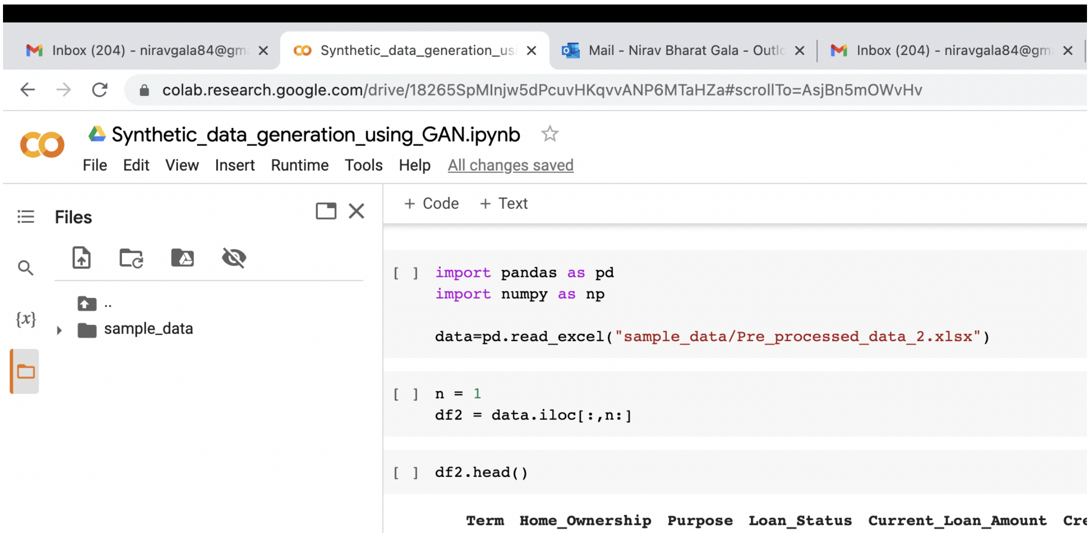

Figure 2: Programming Software

3 Implementation

The implementation of the entire research project is performed in 5 python notebooks which are as under.

- Automated_EDA.ipynb
- Data_Preparation.ipynb
- modelling.ipynb
- Synthetic_data_generation_using_gan.ipynb.
- Smote.ipynb.

The following libraries were used in implementation of this research study.

- Tensorflow.
- imblearn
- numpy.
- matplotlib.
- Pandas Profiler
- pandas.
- Tabgan.
- sklearn.

4 Dataset Description

- The dataset used in this research is bank loan status dataset available in Kaggle at the below URL. `https://www.kaggle.com/datasets/zaurbegiev/my-dataset`.
- It consists of 1,00,000 rows and 19 features where the target variable is Loan-Status.

5 Data pre-processing

- The dataset is uploaded to the google collab environment and pre-processed.The notebook that performs this operation is Data_preparation.ipynb as shown in the figure.Few of the data cleaning operations are shown in the below figures.The pre-processed data is stored in an excel file for further use.

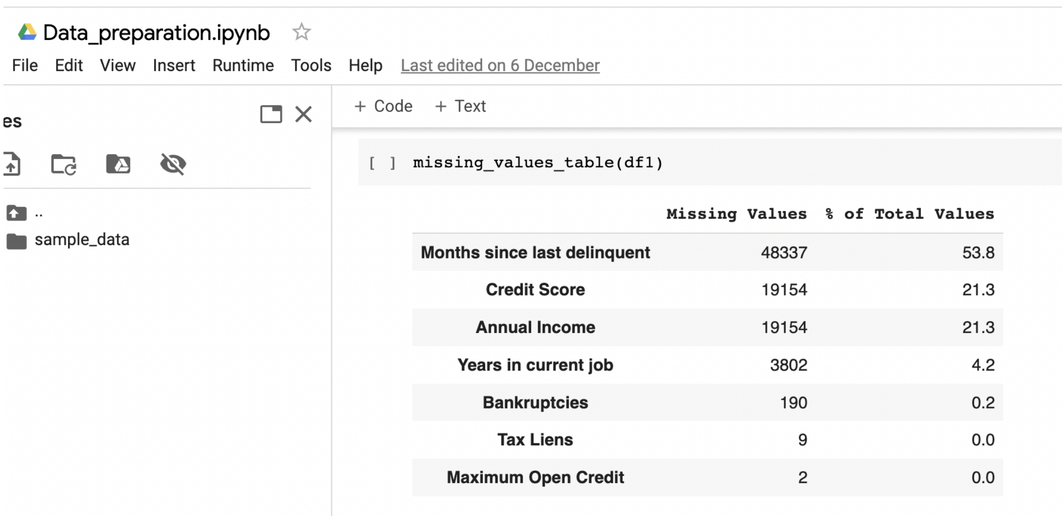

Figure 3: Missing values

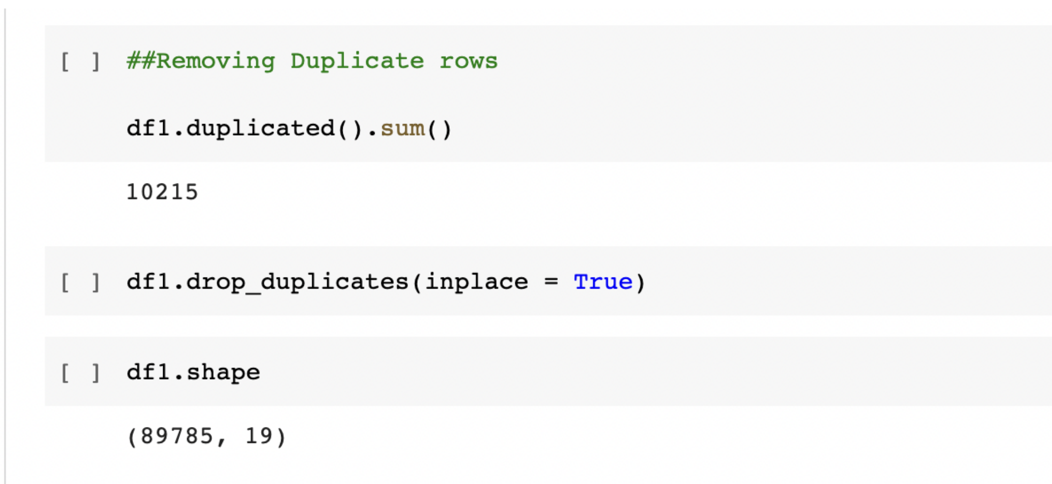

Figure 4: Duplicate rows

6 Synthetic Data Generation

After data-preprocessing , synthetic data is generated for tackling class imbalance using GAN.The notebook that implements this is called Synthetic_data_generation_using_gan.ipynb. The following figures display few of the important code snippets from this notebook.

```
from tabgan.sampler import GANGenerator
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

gen_x, gen_y = GANGenerator(gen_x_times=1.3, cat_cols=None,
           bot_filter_quantile=0.001, top_filter_quantile=0.999, \
              is_post_process=True,
           adversarial_model_params={
               "metrics": "AUC", "max_depth": 2, "max_bin": 100,
               "learning_rate": 0.01, "random_state": \
                42, "n_estimators": 500,
           }, pregeneration_frac=2, only_generated_data=False,\
           gan_params = {"batch_size": 500, "patience": 25, \
          "epochs" : 500,}).generate_data_pipe(df_x_train, df_y_train,\
          df_x_test, deep_copy=True, only_adversarial=False, \
          use_adversarial=True)
```

```
Fitting CTGAN transformers for each column: 100%  16/16 [01:57<00:00, 5.34s/it]
Training CTGAN, epochs:: 36%  181/500 [46:58<1:20:28, 15.14s/it]
```

Figure 5: Tabgan Data Generator

7 Modelling on balanced and imbalanced data

Before data is balanced,modelling is performed on imbalanced data using algorithms like Logistic Regression,Decision Tree and RandomForest.This phase is implemented in modelling.ipynb notebook.

```
[ ] from sklearn.preprocessing import RobustScaler

[ ] scaler = RobustScaler()

[ ] x_train_scaled=scaler.fit_transform(df_x_train)

[ ] x_test_scaled=scaler.transform(df_x_test)

[ ] from sklearn.linear_model import LogisticRegression

[ ] y=df_y_train.to_numpy().ravel()

[ ] model1 = LogisticRegression()
    model1.fit(x_train_scaled,y)

    LogisticRegression()

[ ] model1.score(x_train_scaled,y)

    0.7485699137762648
```

Figure 6: Model Building

```
[ ] y_pred = model1.predict(x_test_scaled)

[ ] from sklearn.metrics import classification_report

    print(classification_report(df_y_test, y_pred))

                  precision    recall  f1-score   support

               0       0.76      0.99      0.86     13448
               1       0.49      0.04      0.07      4471

        accuracy                           0.75     17919
       macro avg       0.62      0.51      0.46     17919
    weighted avg       0.69      0.75      0.66     17919
```

Figure 7: Model Building

7.1 Modelling on data balanced using GAN

Data is balanced using GAN by oversampling the minority class by adding the synthetic records generated.Hyperparameter tuning is performed and models are optimized for metric recall as shown in the following figures.This phase is implemented in the notebook file Synthetic_data_generation_using_gan.ipynb.

```
from sklearn.model_selection import GridSearchCV

params = {"C": np.logspace(-4, 4, 20),
          "solver": ["lbfgs"],
          "class_weight":["balanced"],
          "max_iter":[10000]}

lr_clf = LogisticRegression()

lr_cv = GridSearchCV(lr_clf, params, scoring=b, n_jobs=-1, verbose=1, cv=5)
lr_cv.fit(x_train_scaled, y)
best_params = lr_cv.best_params_
print(f"Best parameters: {best_params}")
lr_clf = LogisticRegression(**best_params)

lr_clf.fit(x_train_scaled, y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best parameters: {'C': 0.0001, 'class_weight': 'balanced', 'max_iter': 10000, 'solver': 'lbfgs'}
LogisticRegression(C=0.0001, class_weight='balanced', max_iter=10000)
```

Figure 8: Hyperparamter Tuning

7.2 Modelling on data balanced using SMOTE

Data is balanced using SMOTE by oversampling the minority class by adding the synthetic records generated.Hyperparameter tuning is performed and models are optimized for metric recall as shown in the following figures.This phase is implemented in the notebook file SMOTE.ipynb.

```
[ ] import imblearn
```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy= 'minority')
X_sm, y_sm = smote.fit_resample(df_x_train,df_y_train)
```

```
[ ] y_sm.value_counts()
```

```
0    53558
1    53558
Name: Loan_Status, dtype: int64
```

Figure 9: SMOTE

```
from sklearn.model_selection import GridSearchCV
n_estimators = [20,60,100, 120]
max_features = ['auto', 'sqrt']
max_depth = [2, 3, 5, 10, 15, None]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]

params_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf
              }

rf_clf = RandomForestClassifier(random_state=42)
rf_cv = GridSearchCV(rf_clf, params_grid, scoring='recall', cv=3, verbose=1, n_jobs=-1)
rf_cv.fit(x_train_scaled, y_sm)
best_params = rf_cv.best_params_
print(f"Best parameters: {best_params}")

rf_clf = RandomForestClassifier(**best_params)
rf_clf.fit(x_train_scaled, y_sm)
```

Figure 10: Hyperparameter Tuning

8 Evaluation of Implemented Methods

Three experiments were carried out.First was modelling on the imbalanced data and second was modelling on the data balanced data using GAN.The third experiement involved modelling on the data balanced using SMOTE.The results of these experiements were compared by classification report and ROC-AUC plots. The figures below show the logistic regression model built in three experiments.

```
print(classification_report(df_y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.76	0.99	0.86	13448
1	0.49	0.04	0.07	4471
accuracy			0.75	17919
macro avg	0.62	0.51	0.46	17919
weighted avg	0.69	0.75	0.66	17919

Figure 11: Experiment 1

```
report=classification_report(df_y_test,predict)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.58      0.68     13448
           1       0.33      0.62      0.43      4471

    accuracy                           0.59     17919
   macro avg       0.58      0.60      0.56     17919
weighted avg       0.70      0.59      0.62     17919
```

Figure 12: Experiment 2

```
report=classification_report(df_y_test,predict)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.85      0.50      0.63     13448
           1       0.33      0.74      0.45      4471

    accuracy                           0.56     17919
   macro avg       0.59      0.62      0.54     17919
weighted avg       0.72      0.56      0.58     17919
```

Figure 13: Experiment 3

```
fig = plt.figure(figsize=(8,6))
for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.2f}".format(i, result_table.loc[i]['auc']))

##plt.plot([0,1], [0,1], color='orange', linestyle='--')
plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)
plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)
plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')
plt.show()
```

Figure 14: Experiment 1:Code for Plotting ROC-AUC Curve

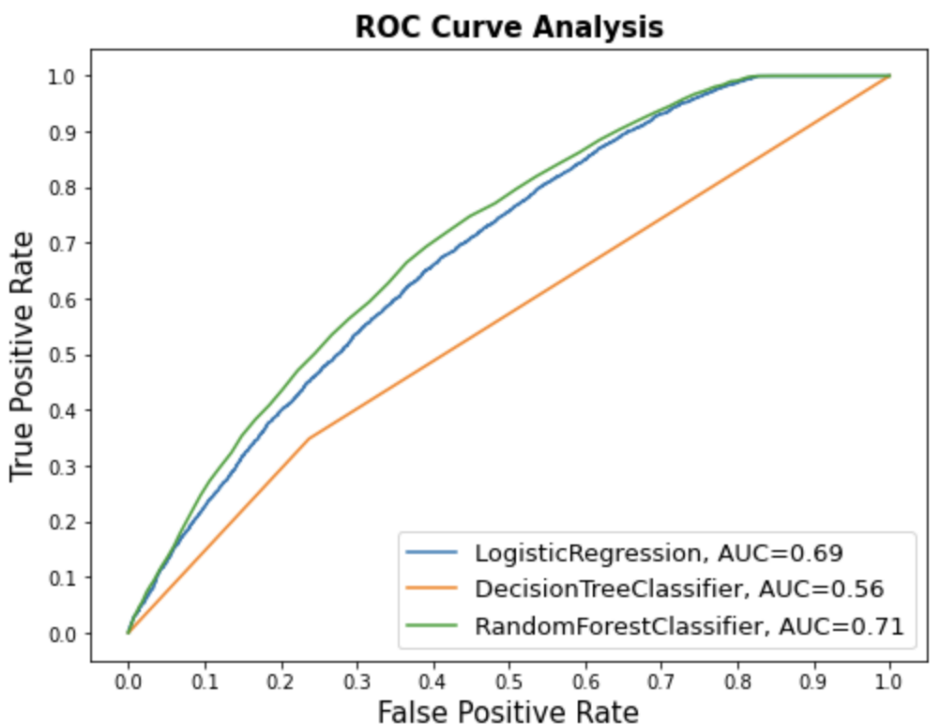

Figure 15: Experiment 1: ROC-AUC Curve

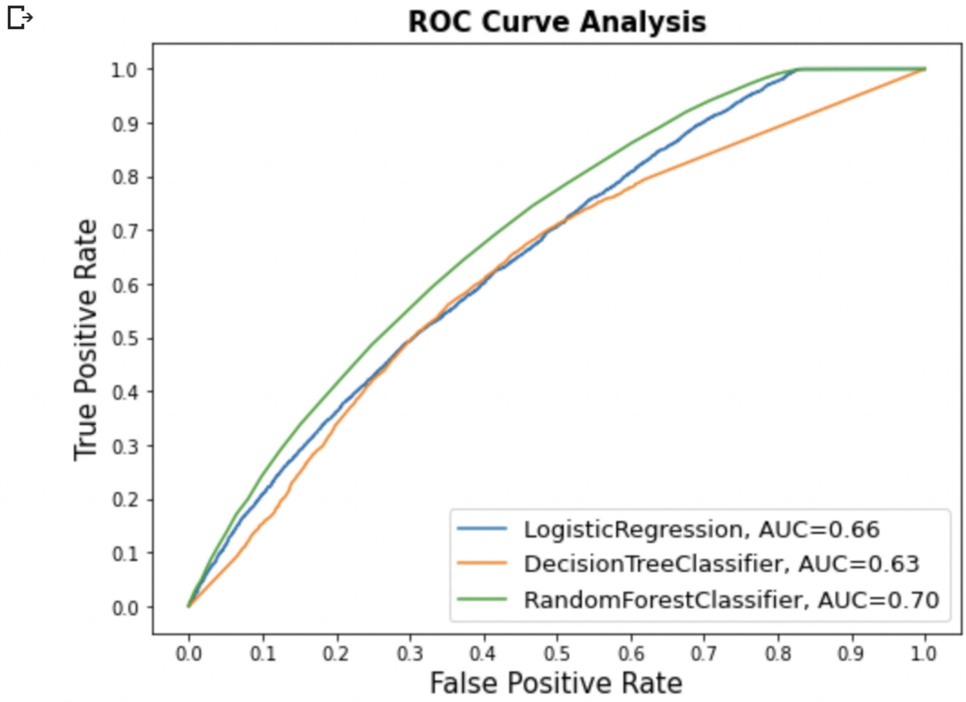

Figure 16: Experiment 2: ROC-AUC Curve

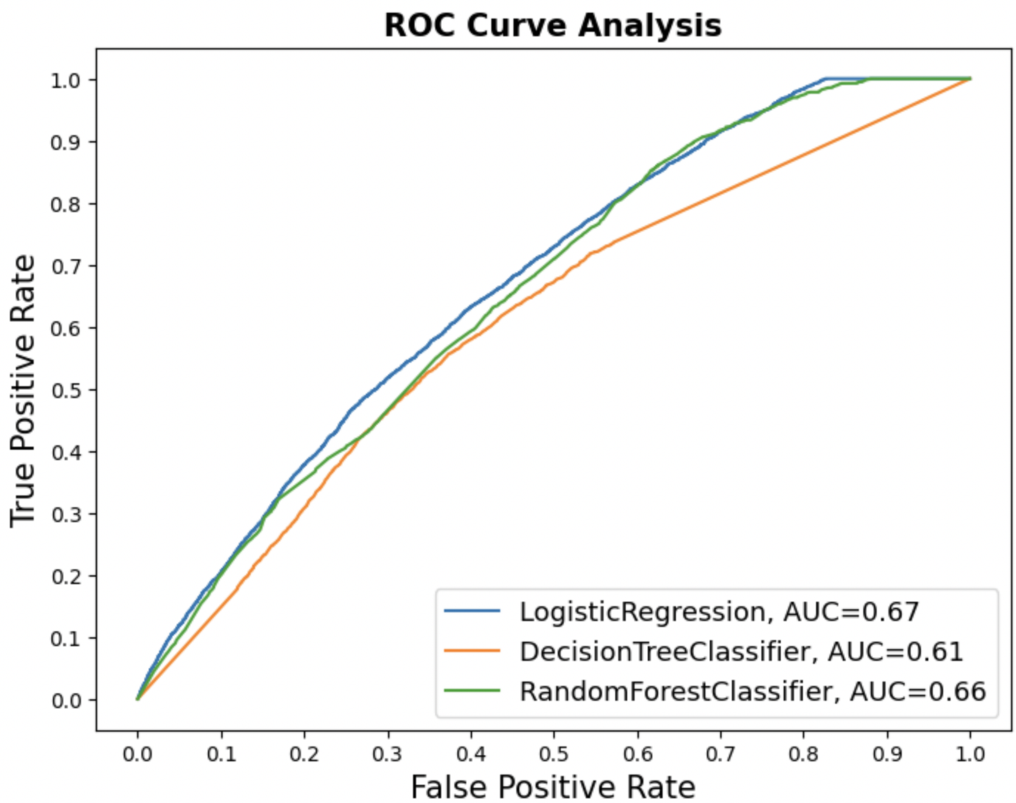

Figure 17: Experiment 3: ROC-AUC Curve