Dataset1-Regression_output_9

November 2, 2021

1 Dataset 1 - Regression

1.1 Experiment Details

The aim of the experiment is to verify if the: 1. ABC_GAN model corrects model misspecification 2. ABC_GAN model performs better and converges faster than a simple C-GAN model

In the experiment we predict the distribution that represents the real data and simulate realistic fake data points using statistical mode, C-GAN and ABC-GAN model with 3 priors. We analyze and compare their performance using metrics like mean squared error, mean absolute error , manhattan distance and euclidean distance between y_{real} and y_{pred}

The models are as follows:

1. The statistical model assumes the distribution $Y = \beta X + \mu$ where $\mu \sim N(0, 1)$
2. The Conditional GAN consists of
    1. Generator with 2 hidden layers with 100 nodes each and ReLu activation.
    2. Discriminator with 2 hidden layers with 25 and 50 nodes and ReLu activation. We use Adam's optimser and BCE Logit Loss to train the model. The input to the Generator of the GAN is (x,e) where x are the features and $e \sim N(0, 1)$. The discriminator output is linear.
3. The ABC GAN Model consists of
    1. ABC generator is defined as follows:
        1. $Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_2 x_3 + ... + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$
        2. $\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else $\beta_i \sim N(\beta_i^*, \sigma^*)$ where $\beta_i^* s$ are coefficients obtained from statistical model
        3. σ^* takes the values 0.01,0.1 and 1

    2. C-GAN network is as defined above. However the input to the Generator of the GAN is (x, y_{abc}) where y_{abc} is the output of the ABC Generator.

1.2 Import Libraries

```
[1]: import warnings
     warnings.filterwarnings('ignore')
```

```
[2]: import train_test
     import ABC_train_test
     import regressionDataset
     import network
```

```python
import statsModel
import performanceMetrics
import dataset
import sanityChecks
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from torch.utils.data import Dataset,DataLoader
from torch import nn
```

1.3 Parameters

General Parameters

1. Number of Samples
2. Number of features

ABC-Generator parameters are as mentioned below: 1. mean : 1 ($\beta \sim N(\beta^*, \sigma)$ where β^* are coefficients of statistical model) or 1 ($\beta \sim N(0, \sigma)$ 2. std : $\sigma = 1, 0.1, 0.01$ (standard deviation)

```python
[3]: n_features = 10
     n_samples= 100

     #ABC Generator Parameters
     mean = 1
     variance = 0.001
```

```python
[4]: # Parameters
     n_samples = 100
     n_features = 10
     mean = 0
     variance = 1
```

1.4 Dataset

Generate a random regression problem

$Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_2 x_3 + ... + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$

```python
[5]: X,Y = regressionDataset.regression_data(n_samples,n_features)
```

```
          X1        X2        X3        X4        X5        X6        X7  \
0  -0.517925 -0.320773  0.014421  0.651013  1.037562  0.107556 -0.646690
1   0.813374 -0.208519 -0.309006 -1.029033 -1.072201 -1.601325 -1.369895
2   1.289511 -0.789571 -0.286003 -1.561090 -0.972157  0.208757  0.215313
3  -2.410901 -2.653887  1.417954 -1.772042  1.225399  1.759394  0.128076
4  -0.493614  0.540132  0.684174  2.231490  1.887989  0.189140 -0.592425

          X8        X9       X10              Y
```

```
0  -1.144631   2.083538  -0.056830    59.120547
1   1.751645  -1.081315   0.457012  -154.254285
2   0.171749  -0.053740  -1.293846   -87.564555
3   0.071760   0.281419   0.714336  -318.284523
4   0.942683  -0.815419  -0.849842   172.744112
```

1.5 Stats Model

```
[6]:  [coeff,y_pred] = statsModel.statsModel(X,Y)
```

No handles with labels found to put in legend.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.826e+07
Date:                Tue, 02 Nov 2021   Prob (F-statistic):          9.54e-285
Time:                        18:27:43   Log-Likelihood:                 606.65
No. Observations:                 100   AIC:                            -1191.
Df Residuals:                      89   BIC:                            -1163.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        6.939e-18   5.95e-05   1.17e-13      1.000      -0.000       0.000
x1              0.3783   6.06e-05   6246.069      0.000       0.378       0.378
x2              0.5512   6.04e-05   9132.240      0.000       0.551       0.551
x3              0.0292   6.37e-05    459.033      0.000       0.029       0.029
x4              0.2235   6.21e-05   3599.723      0.000       0.223       0.224
x5              0.3441   6.17e-05   5577.336      0.000       0.344       0.344
x6              0.1074   6.07e-05   1769.877      0.000       0.107       0.108
x7              0.3352   6.14e-05   5455.445      0.000       0.335       0.335
x8              0.3582   6.32e-05   5667.712      0.000       0.358       0.358
x9              0.3601   6.15e-05   5856.357      0.000       0.360       0.360
x10             0.0167   6.41e-05    260.594      0.000       0.017       0.017
==============================================================================
Omnibus:                        0.146   Durbin-Watson:                   1.711
Prob(Omnibus):                  0.930   Jarque-Bera (JB):                0.325
Skew:                           0.030   Prob(JB):                        0.850
Kurtosis:                       2.727   Cond. No.                         1.62
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    6.938894e-18
x1          3.783372e-01
```
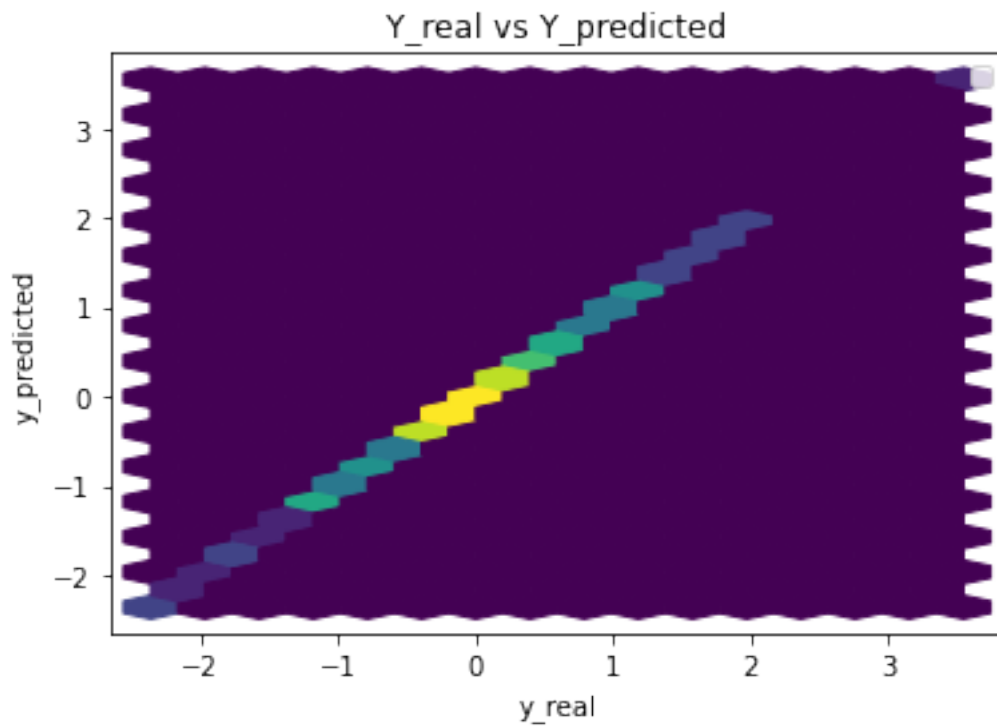
```
x2      5.512438e-01
x3      2.921850e-02
x4      2.234658e-01
x5      3.440829e-01
x6      1.074100e-01
x7      3.351972e-01
x8      3.581973e-01
x9      3.601080e-01
x10     1.669499e-02
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 3.14953091949632e-07
Mean Absolute Error: 0.0004475444042220275
Manhattan distance: 0.04475444042220275
Euclidean distance: 0.005612068174475717
```

1.6 Common Training Parameters (GAN & ABC_GAN)

```
[7]: n_epochs = 5000
     error = 0.001
     batch_size = n_samples
```

1.7 GAN Model

```python
[8]: real_dataset = dataset.CustomDataset(X,Y)
     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Training GAN for n_epochs number of epochs

```python
[9]: generator = network.Generator(n_features+2)
     discriminator = network.Discriminator(n_features+2)

     criterion = torch.nn.BCEWithLogitsLoss()
     gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
     disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5, 0.
      →999))
```

```python
[10]: print(generator)
      print(discriminator)
```

```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)
  (hidden2): Linear(in_features=25, out_features=50, bias=True)
  (output): Linear(in_features=50, out_features=1, bias=True)
  (relu): ReLU()
)
```

```python
[11]: train_test.
       →training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
       →n_epochs,criterion,device)
```
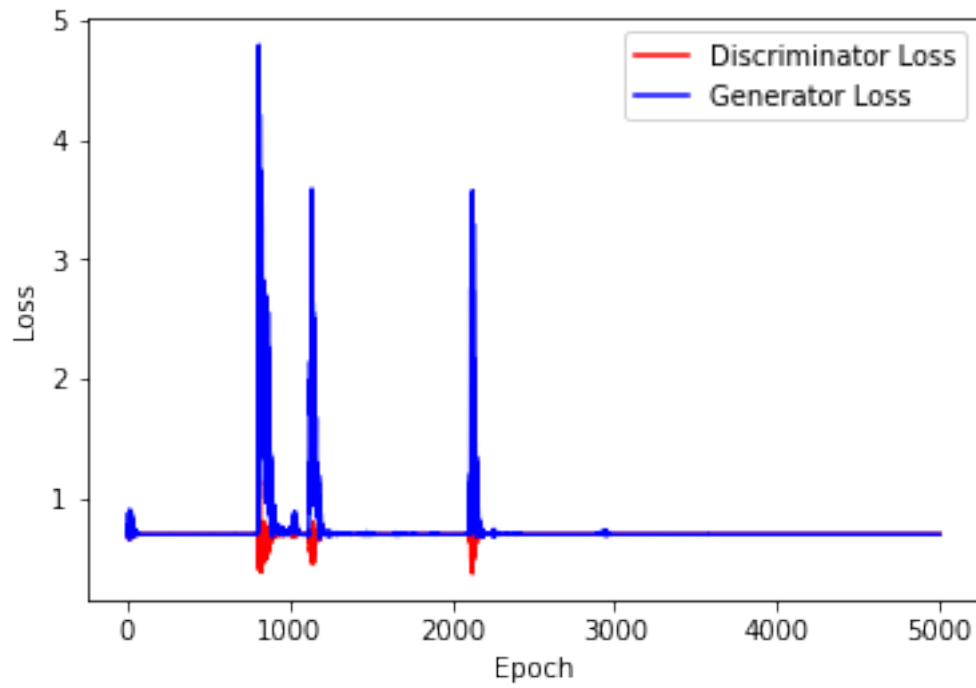
```
[12]: GAN1_metrics = train_test.test_generator(generator,real_dataset,device)
```

```
[13]: sanityChecks.discProbVsError(real_dataset,discriminator,device)
```

Discriminator Output for noisy data
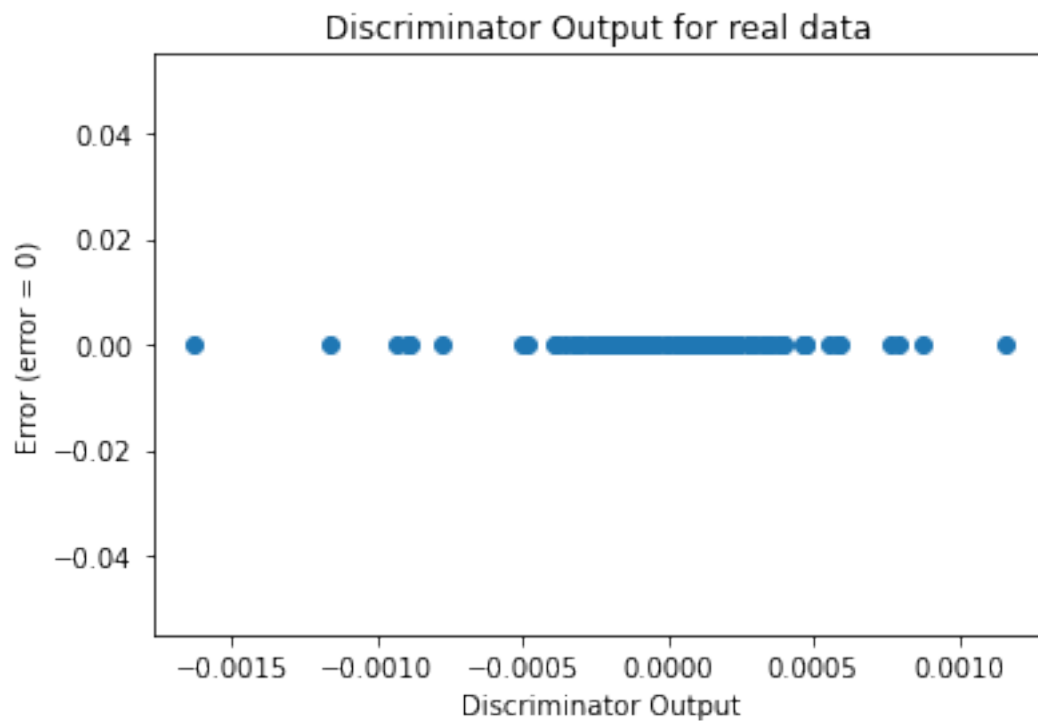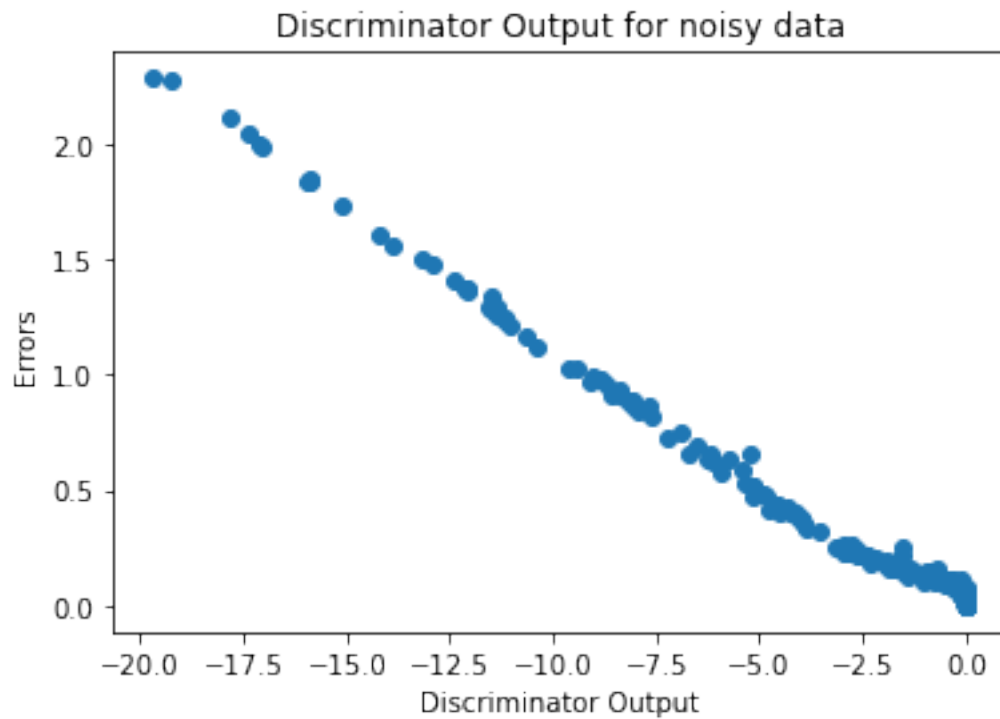
Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

```
[14]: generator2 = network.Generator(n_features+2)
      discriminator2 = network.Discriminator(n_features+2)
      criterion = torch.nn.BCEWithLogitsLoss()
      gen_opt = torch.optim.Adam(generator2.parameters(), lr=0.01, betas=(0.5, 0.999))
      disc_opt = torch.optim.Adam(discriminator2.parameters(), lr=0.01, betas=(0.
       →999))
```

```
[15]: train_test.
       →training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,real_dataset,batch_size,error,cri
```

Number of epochs needed 381
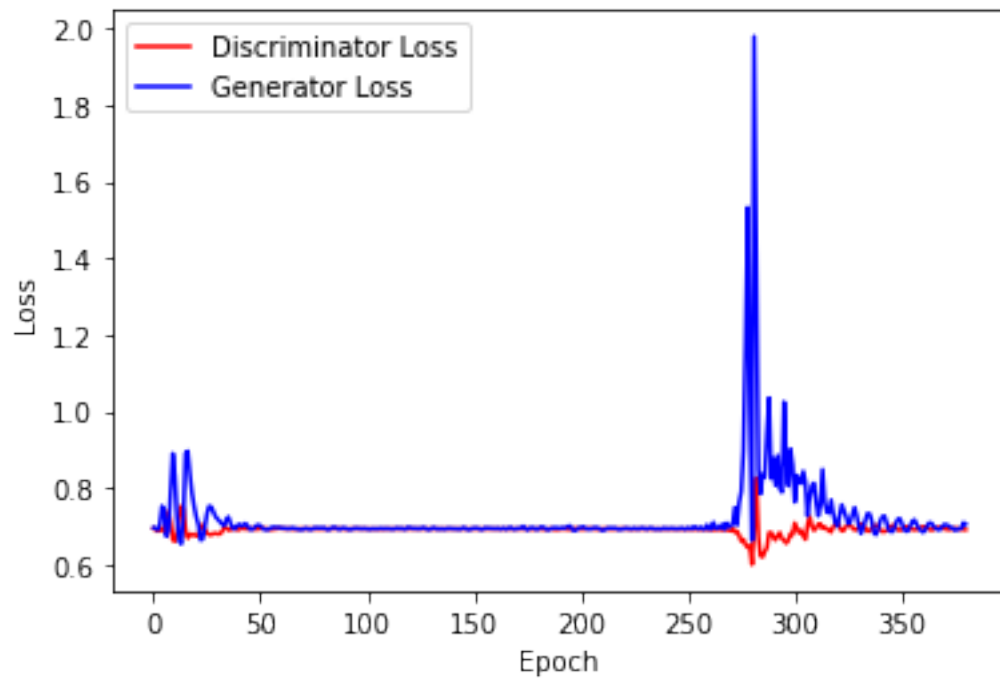
```
[16]: GAN2_metrics=train_test.test_generator_2(generator2,real_dataset,device)
```

```
[17]: sanityChecks.discProbVsError(real_dataset,discriminator2,device)
```

Discriminator Output for noisy data
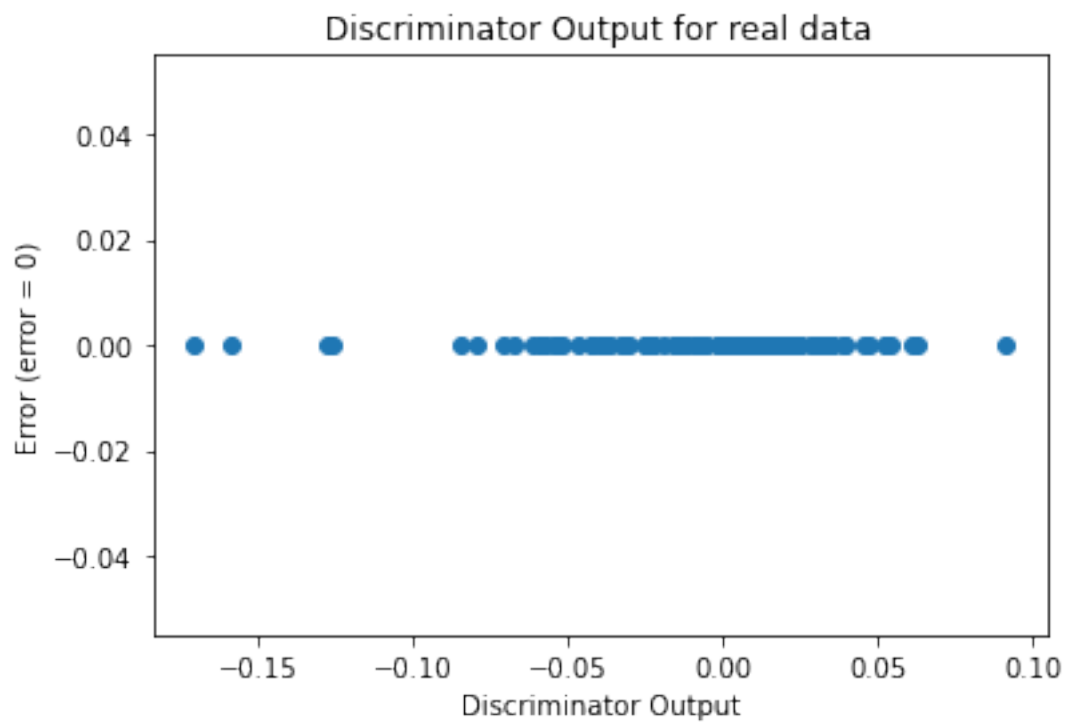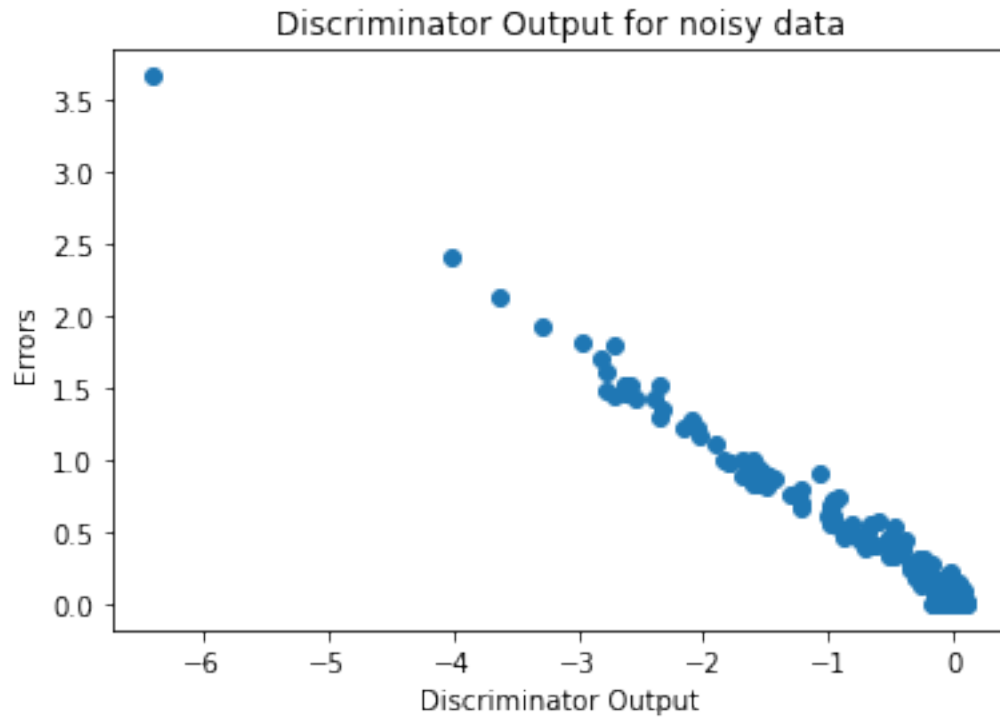
2 ABC GAN Model

2.0.1 Training the network

Training ABC-GAN for n_epochs number of epochs

```
[18]: gen = network.Generator(n_features+2)
      disc = network.Discriminator(n_features+2)

      criterion = torch.nn.BCEWithLogitsLoss()
      gen_opt = torch.optim.Adam(gen.parameters(), lr=0.01, betas=(0.5, 0.999))
      disc_opt = torch.optim.Adam(disc.parameters(), lr=0.01, betas=(0.5, 0.999))
```
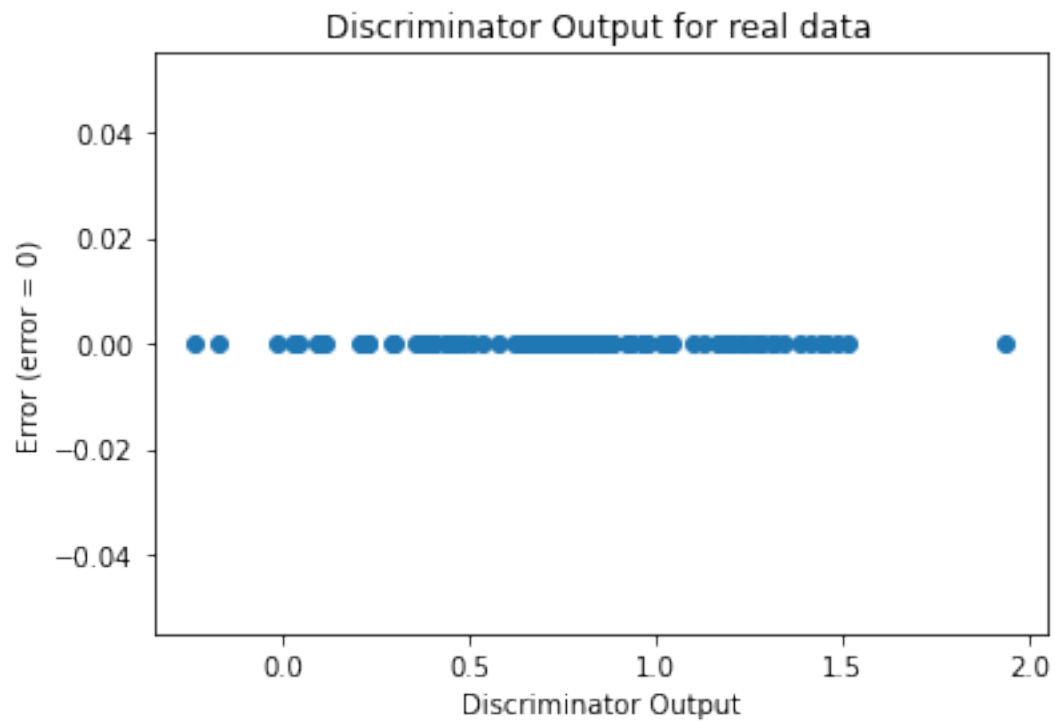
```
[19]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
      ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```
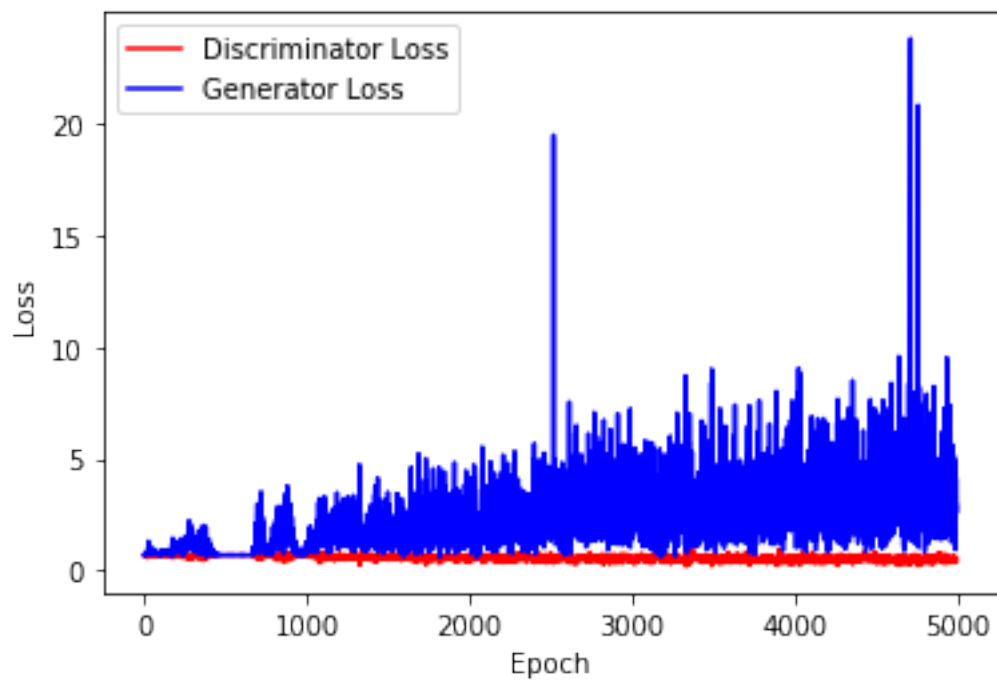
```
[20]: ABC_GAN1_metrics=ABC_train_test.
      ↪test_generator(gen,real_dataset,coeff,mean,variance,device)
```

Sanity Checks

```
[21]: sanityChecks.discProbVsError(real_dataset,disc,device)
```

Discriminator Output for real data


Discriminator Output for noisy data
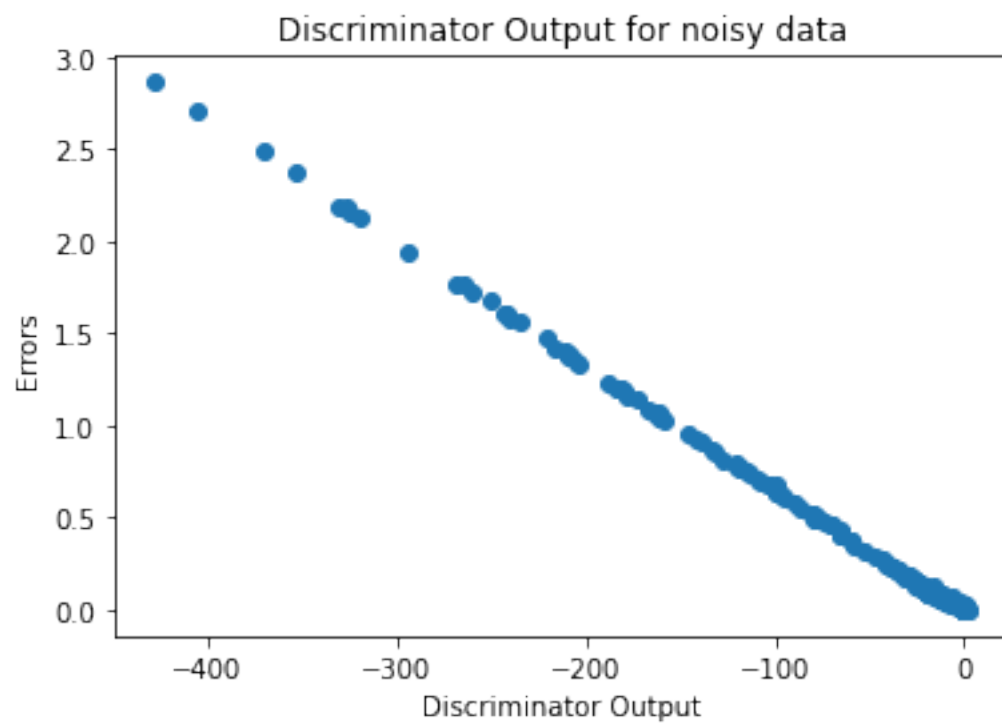
Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

```
[22]: gen2 = network.Generator(n_features+2)
      disc2 = network.Discriminator(n_features+2)

      criterion = torch.nn.BCEWithLogitsLoss()
      gen_opt = torch.optim.Adam(gen2.parameters(), lr=0.01, betas=(0.5, 0.999))
      disc_opt = torch.optim.Adam(disc2.parameters(), lr=0.01, betas=(0.5, 0.999))
```

```
[23]: ABC_train_test.
      ↪training_GAN_2(disc2,gen2,disc_opt,gen_opt,real_dataset,batch_size,␣
      ↪error,criterion,coeff,mean,variance,device)
```
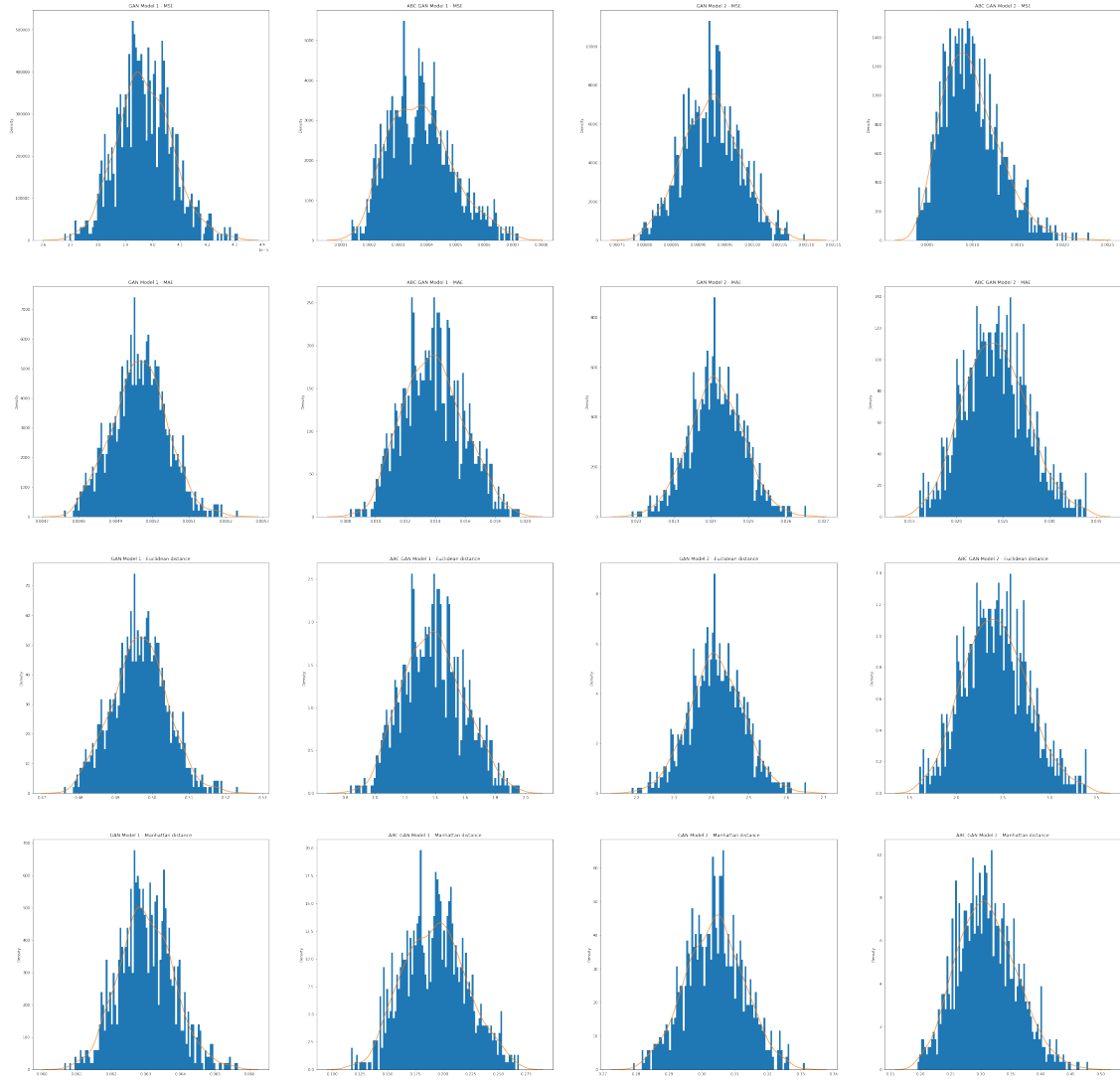
Number of epochs 356
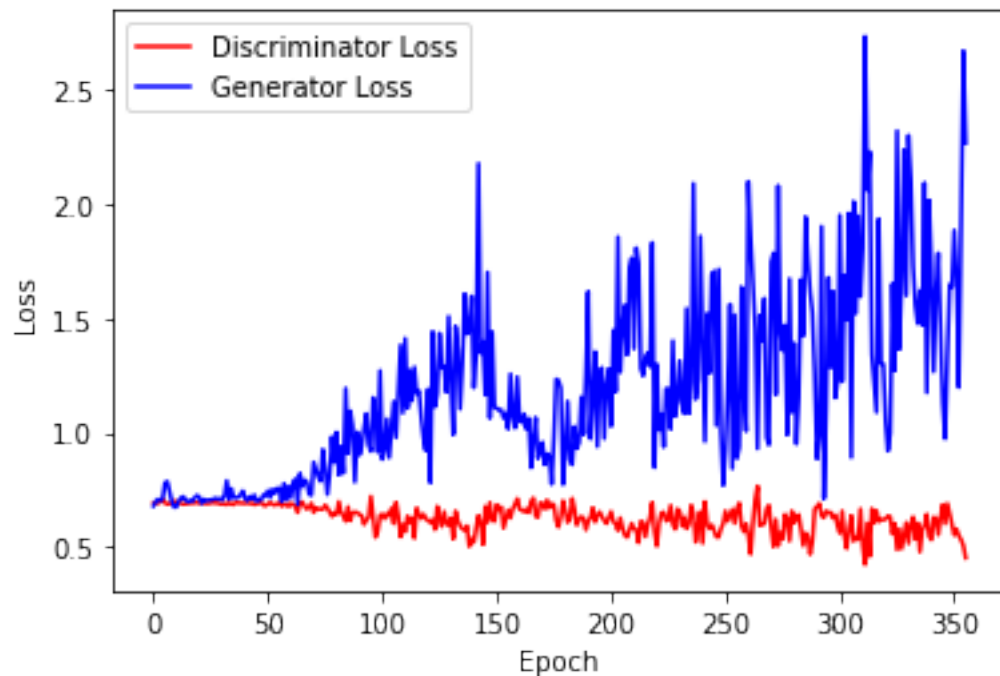

```
[24]: ABC_GAN2_metrics=ABC_train_test.
      ↪test_generator_2(gen2,real_dataset,coeff,mean,variance,device)
```

```
[25]: sanityChecks.discProbVsError(real_dataset,disc2,device)
```

Discriminator Output for real data

Discriminator Output for noisy data
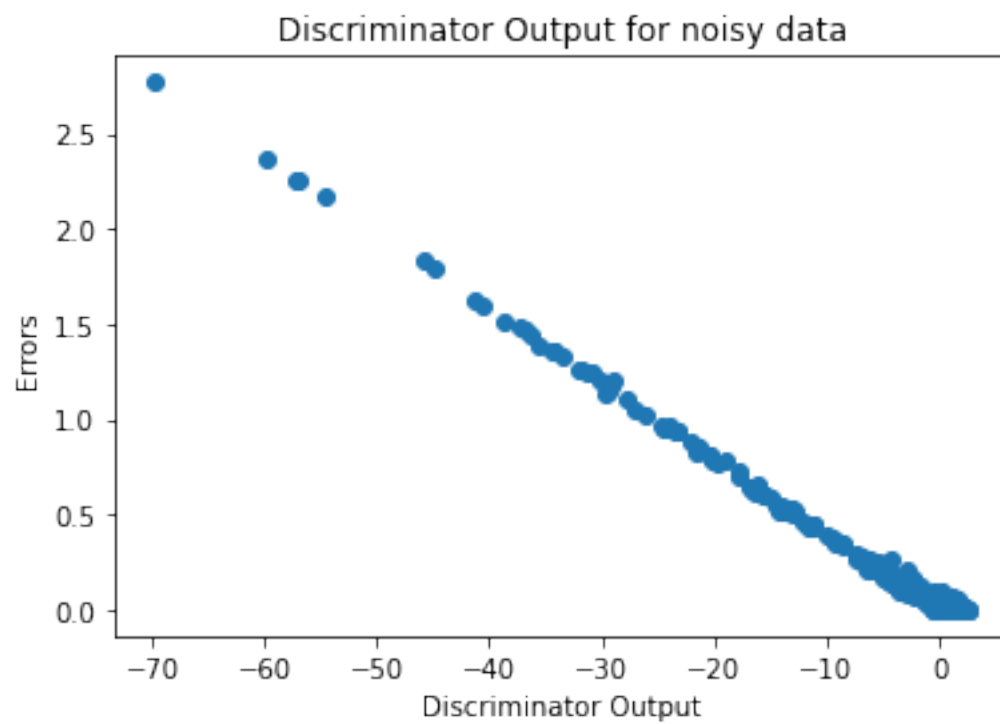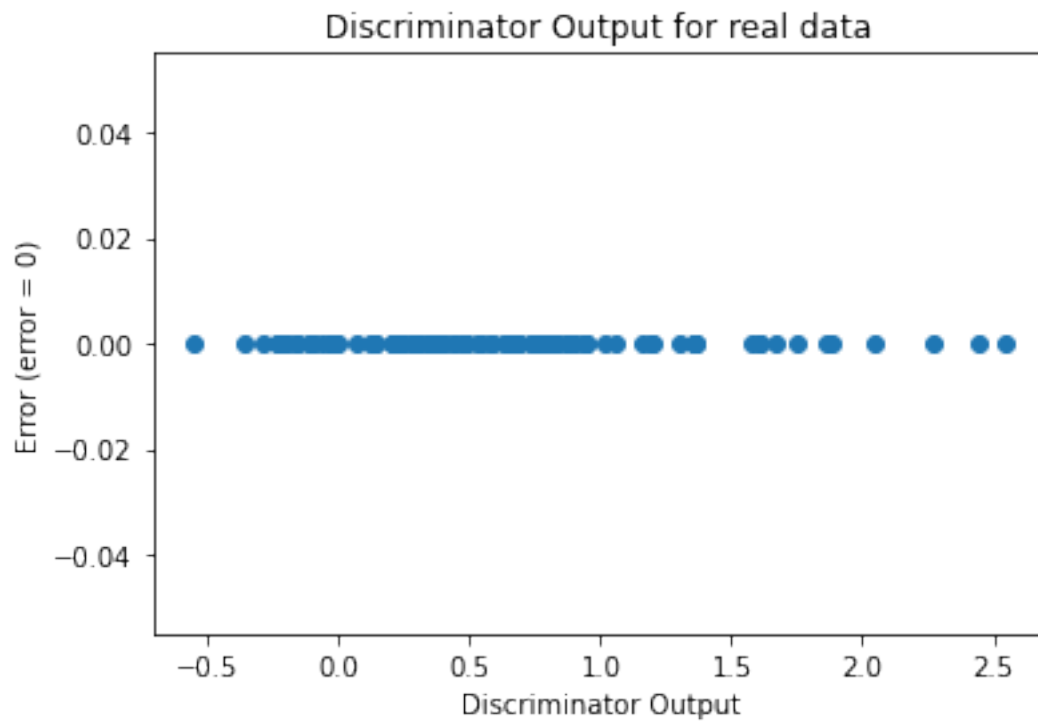
3   Model Analysis

```
[26]: performanceMetrics.
      ↪modelAnalysis(GAN1_metrics,ABC_GAN1_metrics,GAN2_metrics,ABC_GAN2_metrics)
```



```
[ ]:
```