

Dataset1-Regression_output_11

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 model, 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 optimiser 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. $Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$
 2. $\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else $\beta_i \sim N(\beta_i^*, \sigma^*)$ where β_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
```

```

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)

```

[3]: n_features = 10
     n_samples= 100

     #ABC Generator Parameters
     mean = 1
     variance = 0.001

```

```

[4]: # Parameters
     n_samples = 100
     n_features = 10
     mean = 0
     variance = 0.01

```

1.4 Dataset

Generate a random regression problem

$Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$

```

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

```

| | X1 | X2 | X3 | X4 | X5 | X6 | X7 \ |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 0 | -0.860844 | 0.117922 | 0.163629 | 0.353067 | -0.744865 | -0.170500 | -1.213949 |
| 1 | -0.256920 | -1.333080 | -0.019197 | -1.011873 | -0.491135 | 0.997981 | -0.148086 |
| 2 | 0.902495 | 1.240818 | 0.895561 | 0.090845 | 0.261104 | 0.748207 | -0.139901 |
| 3 | 0.617556 | -1.157392 | -0.510753 | 1.029362 | 0.445784 | -1.146300 | -0.715458 |
| 4 | 1.627241 | -0.195752 | 1.217944 | -1.265895 | -1.084936 | 0.553177 | -2.077023 |
| | X8 | X9 | X10 | Y | | | |

```

0 -0.535152  1.474283  0.181919 -195.718771
1  1.367162 -1.746467  0.181863 -67.703916
2 -0.391223 -1.274797  2.203746  402.117598
3 -0.167168  1.279619  1.503284  63.679034
4  1.834429 -0.706555 -1.585716  38.923515

```

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:            5.122e+07
Date:                          Tue, 02 Nov 2021    Prob (F-statistic):      3.05e-296
Time:                          18:30:14    Log-Likelihood:          636.39
No. Observations:              100    AIC:                    -1251.
Df Residuals:                  89    BIC:                    -1222.
Df Model:                      10
Covariance Type:               nonrobust
=====

```

| | coef | std err | t | P> t | [0.025 | 0.975] |
|-------|--------|----------|----------|-------|-----------|----------|
| const | 0 | 4.42e-05 | 0 | 1.000 | -8.78e-05 | 8.78e-05 |
| x1 | 0.3996 | 4.57e-05 | 8736.232 | 0.000 | 0.400 | 0.400 |
| x2 | 0.2929 | 4.71e-05 | 6224.826 | 0.000 | 0.293 | 0.293 |
| x3 | 0.4439 | 4.73e-05 | 9380.337 | 0.000 | 0.444 | 0.444 |
| x4 | 0.2769 | 4.63e-05 | 5978.869 | 0.000 | 0.277 | 0.277 |
| x5 | 0.3501 | 4.69e-05 | 7470.425 | 0.000 | 0.350 | 0.350 |
| x6 | 0.0561 | 4.85e-05 | 1158.384 | 0.000 | 0.056 | 0.056 |
| x7 | 0.2872 | 4.67e-05 | 6145.078 | 0.000 | 0.287 | 0.287 |
| x8 | 0.4825 | 4.75e-05 | 1.02e+04 | 0.000 | 0.482 | 0.483 |
| x9 | 0.0034 | 4.66e-05 | 73.704 | 0.000 | 0.003 | 0.004 |
| x10 | 0.3743 | 4.8e-05 | 7799.616 | 0.000 | 0.374 | 0.374 |

```

=====
Omnibus:                      6.313    Durbin-Watson:            2.324
Prob(Omnibus):                0.043    Jarque-Bera (JB):          6.683
Skew:                         0.385    Prob(JB):                  0.0354
Kurtosis:                     4.005    Cond. No.                   1.78
=====

```

Notes:

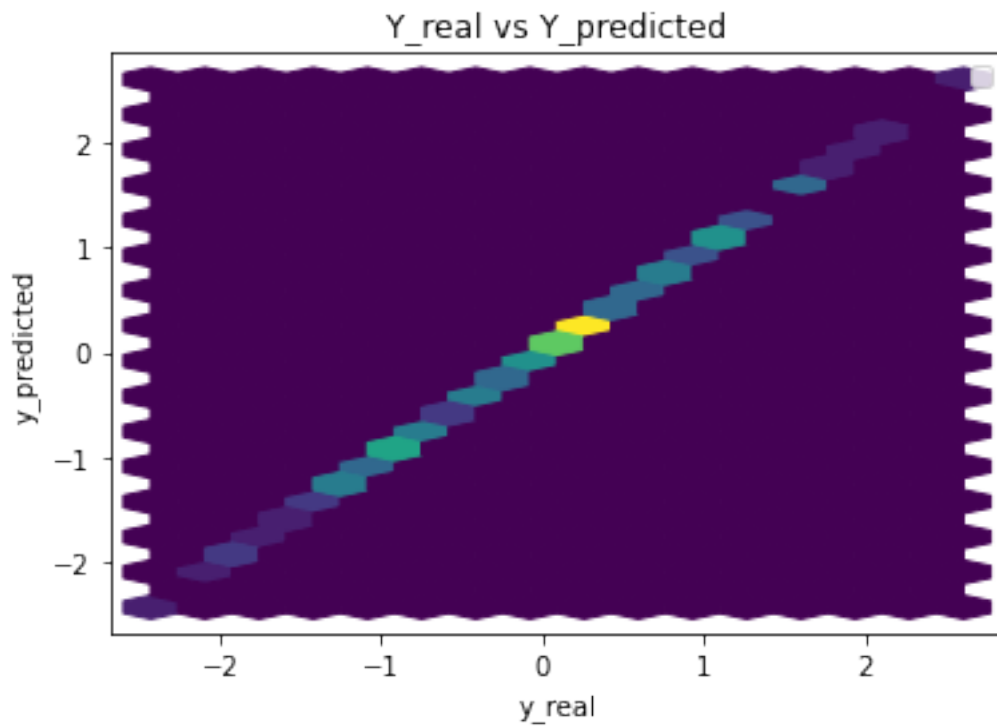
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```

Parameters:  const      0.000000
x1          0.399625

```

```
x2      0.292881
x3      0.443872
x4      0.276887
x5      0.350138
x6      0.056148
x7      0.287242
x8      0.482482
x9      0.003432
x10     0.374328
dtype: float64
```



Performance Metrics

```
Mean Squared Error: 1.737441979855676e-07
Mean Absolute Error: 0.00031478798065315977
Manhattan distance: 0.03147879806531598
Euclidean distance: 0.004168263403212034
```

1.6 Common Training Parameters (GAN & ABC_GAN)

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

1.7 GAN Model

```
[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

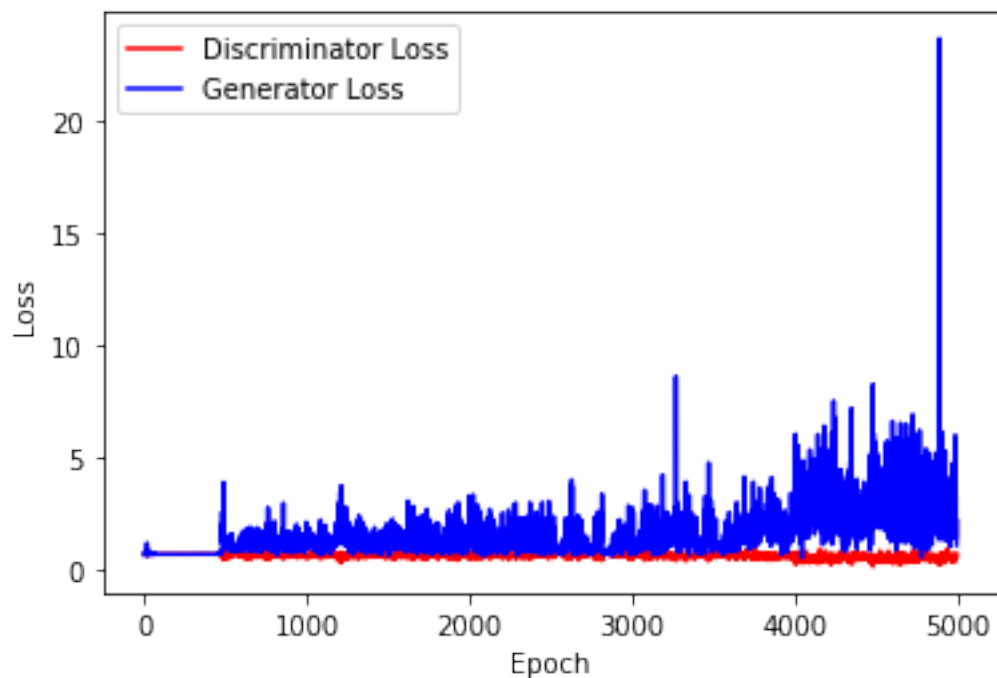
```
[9]: generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)

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))
```

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

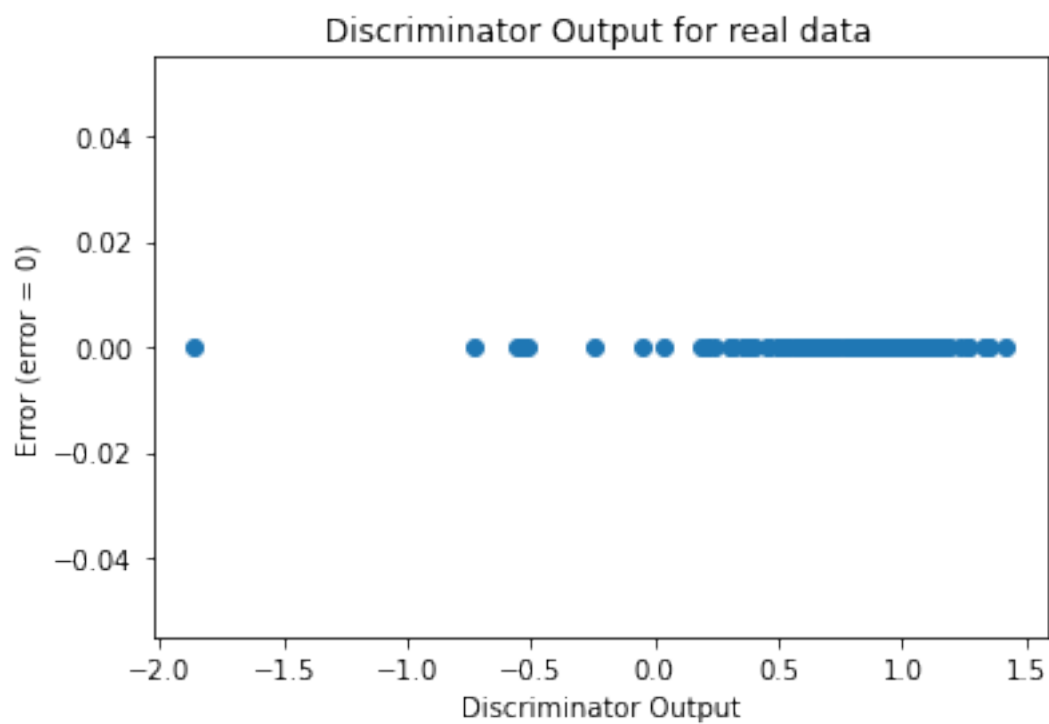
```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)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

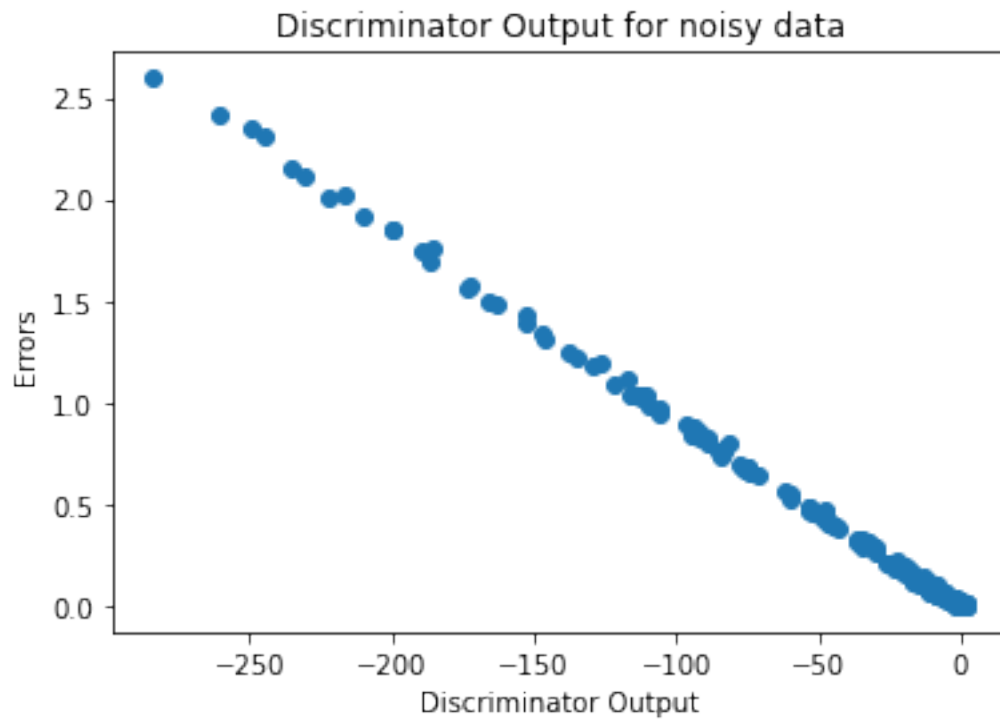
```
[11]: train_test.
→training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,
→n_epochs,criterion,device)
```



```
[12]: GAN1_metrics = train_test.test_generator(generator,real_dataset,device)
```

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



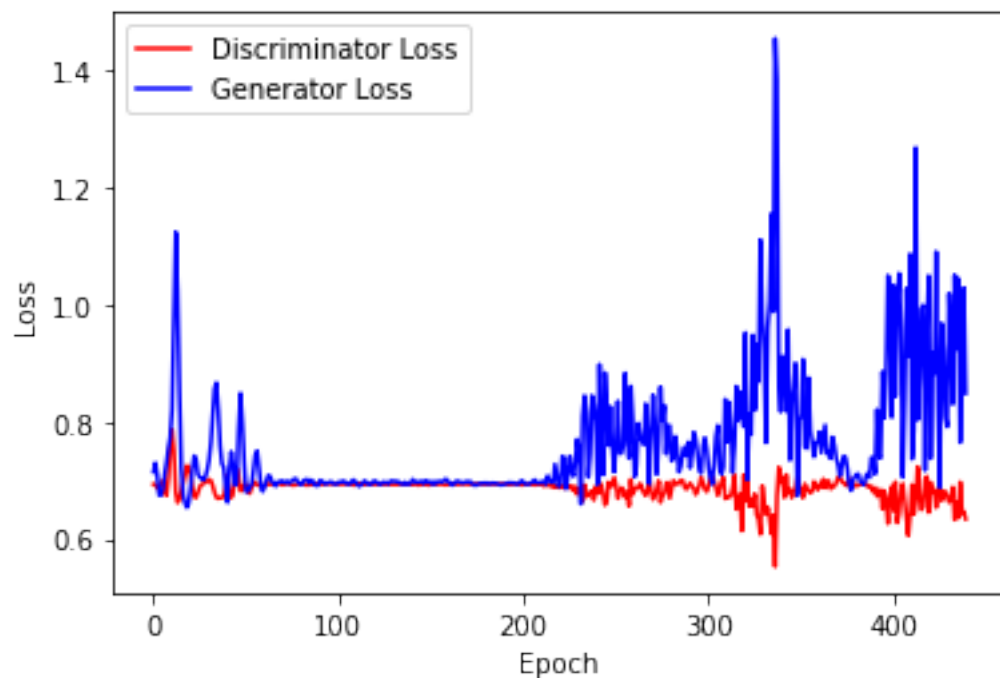


Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

```
[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=(0.5, 0.
↪999))
```

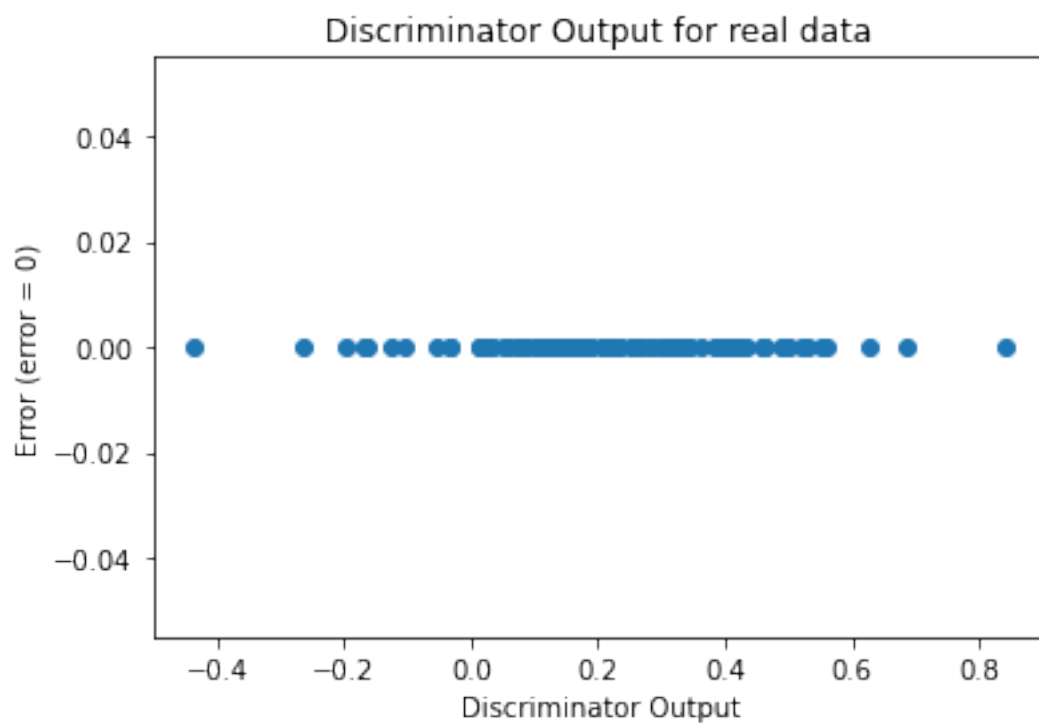
```
[15]: train_test.
↪training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,real_dataset,batch_size,error,crit
```

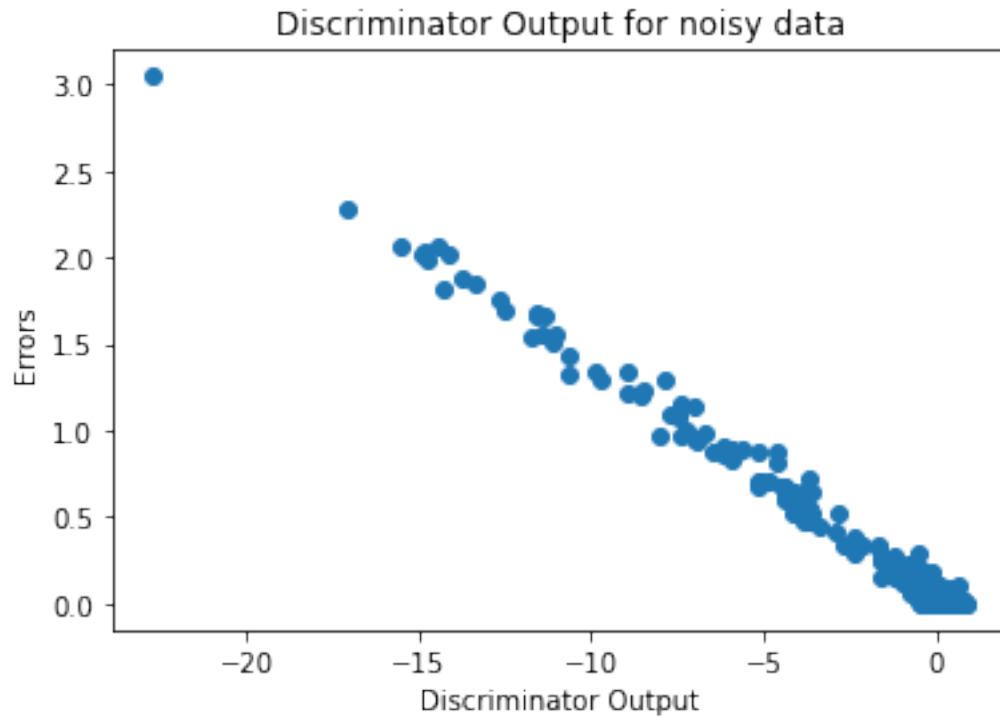
Number of epochs needed 440



```
[16]: GAN2_metrics=train_test.test_generator_2(generator2,real_dataset,device)
```

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





2 ABC GAN Model

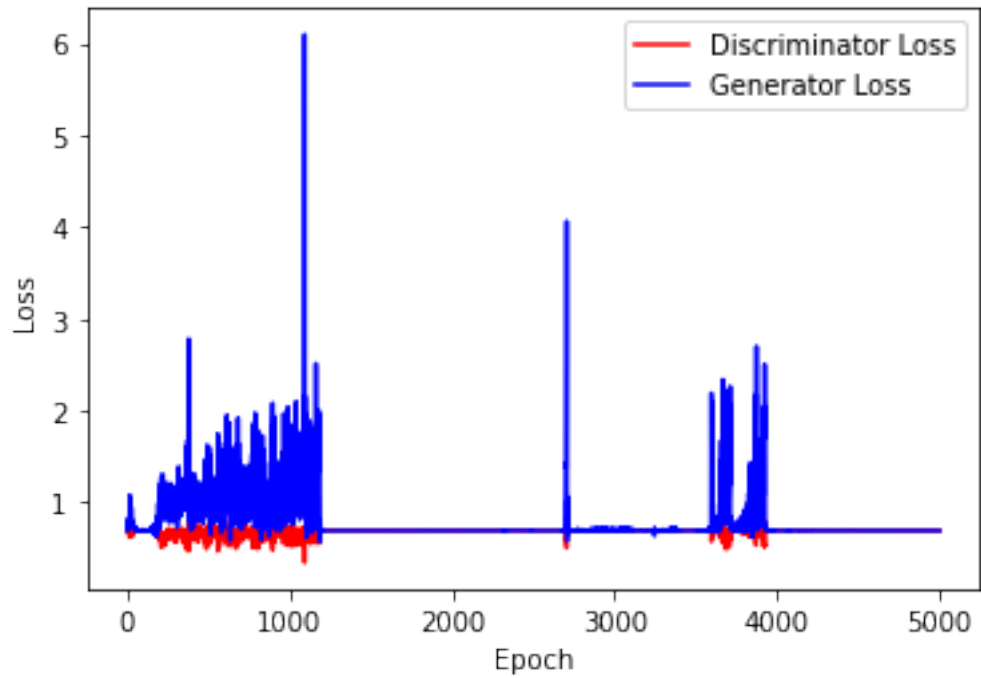
2.0.1 Training the network

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))

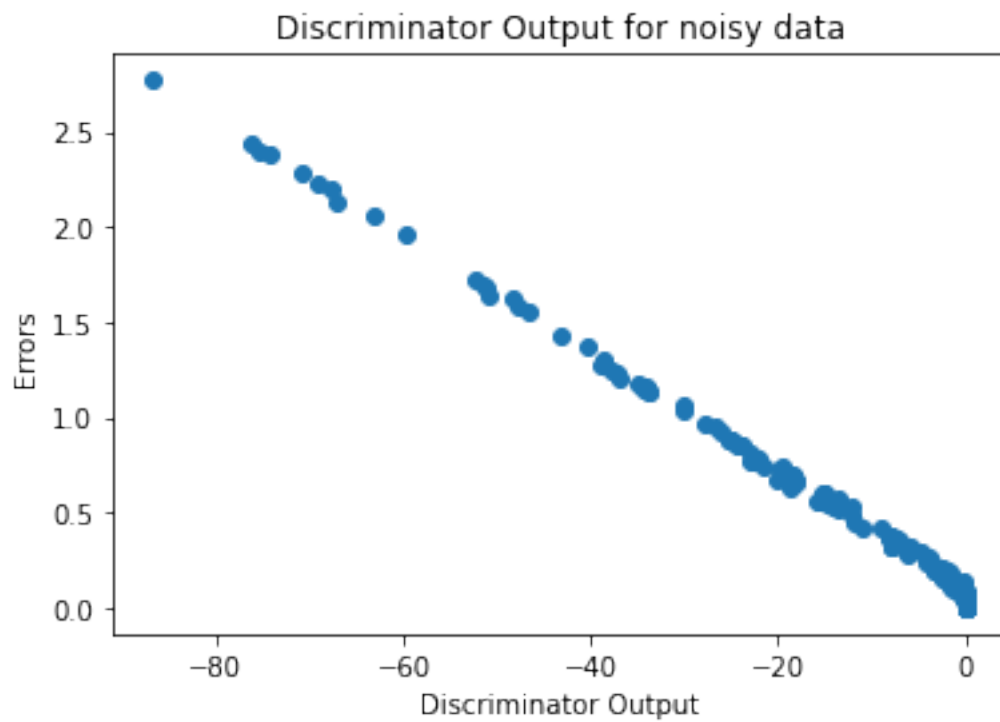
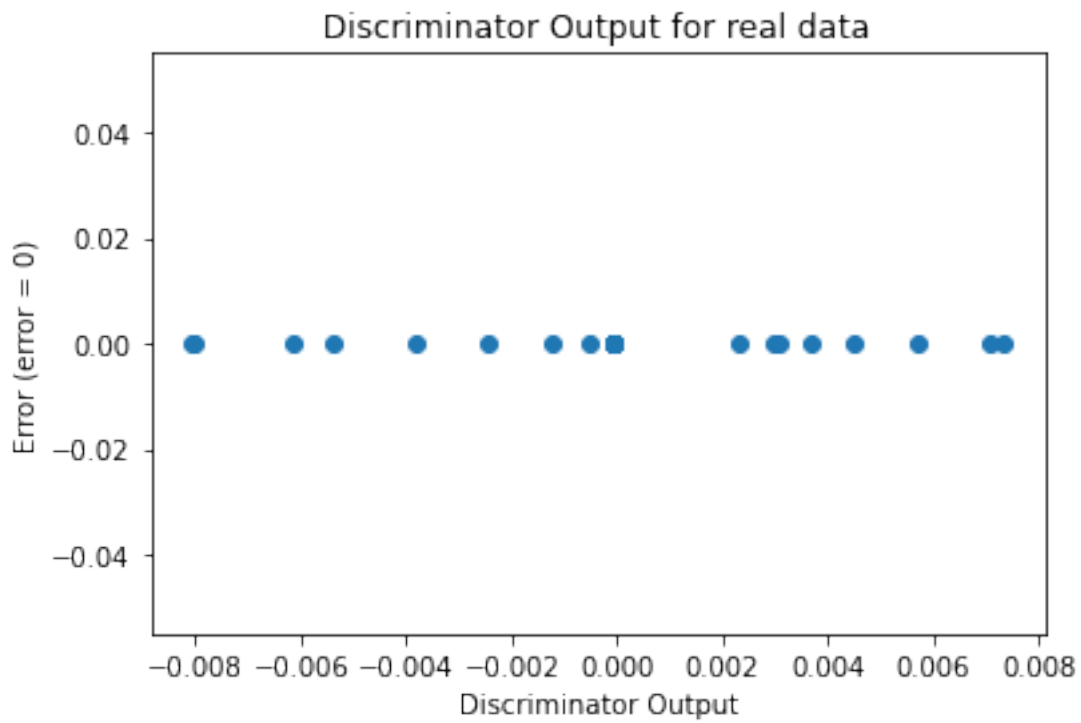
[19]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,
      ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```



```
[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)
```



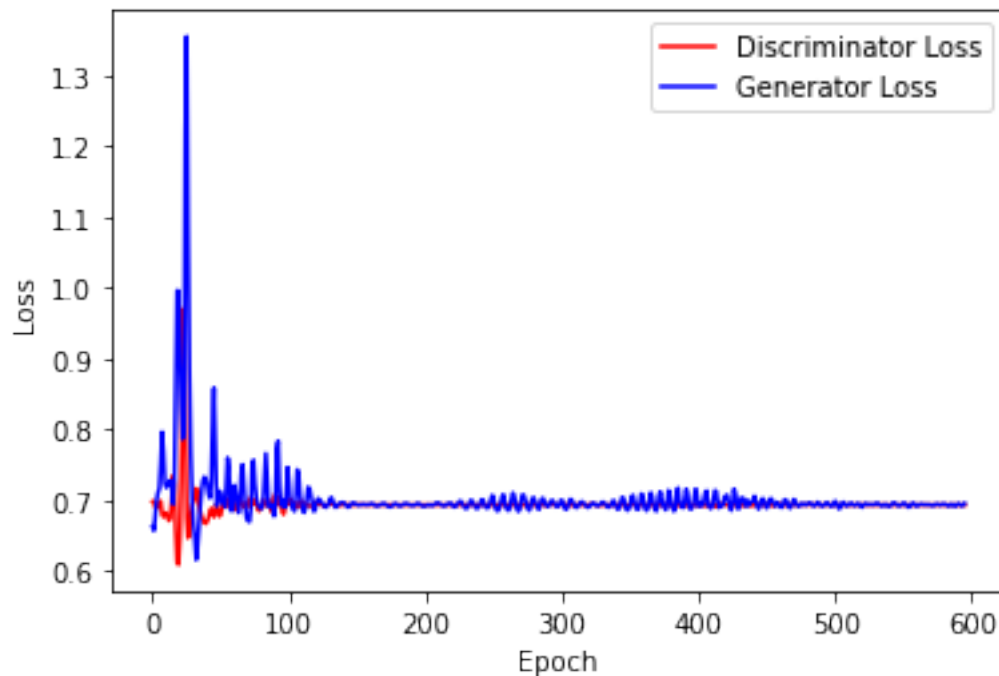
Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

```
[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))
```

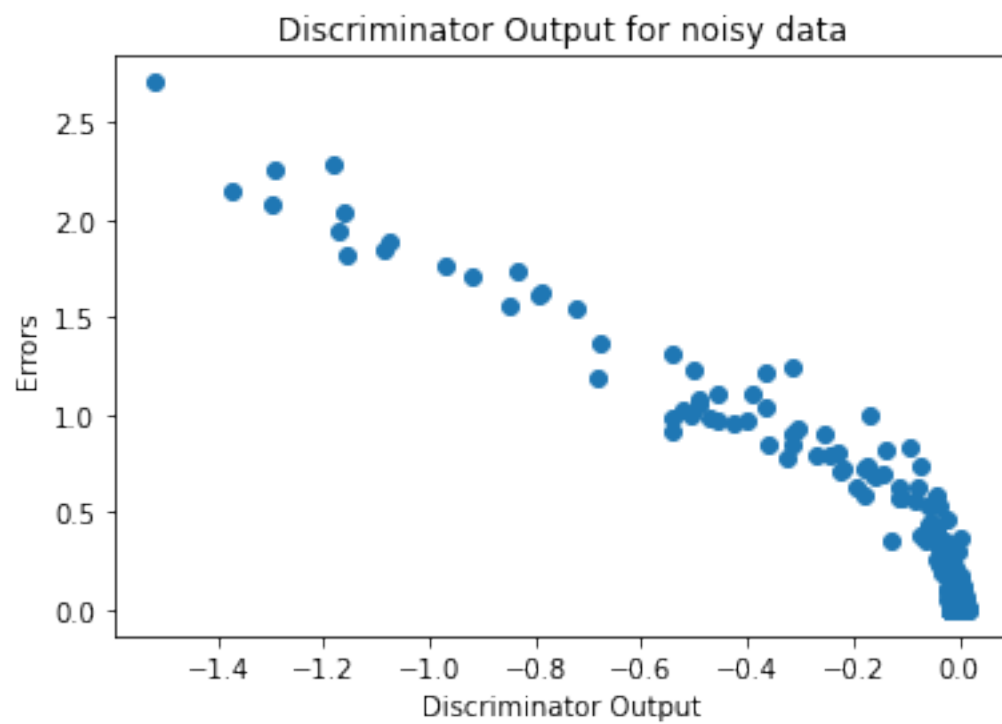
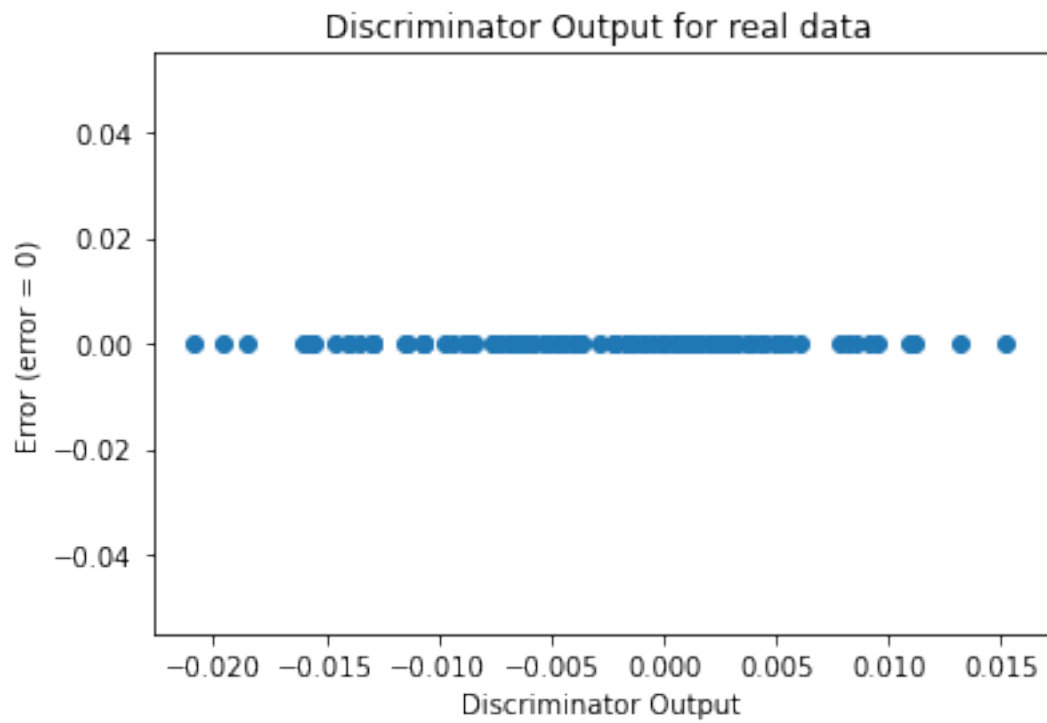
```
[23]: ABC_train_test.
      ↪ training_GAN_2(disc2,gen2,disc_opt,gen_opt,real_dataset,batch_size,
      ↪ error,criterion,coeff,mean,variance,device)
```

Number of epochs 596



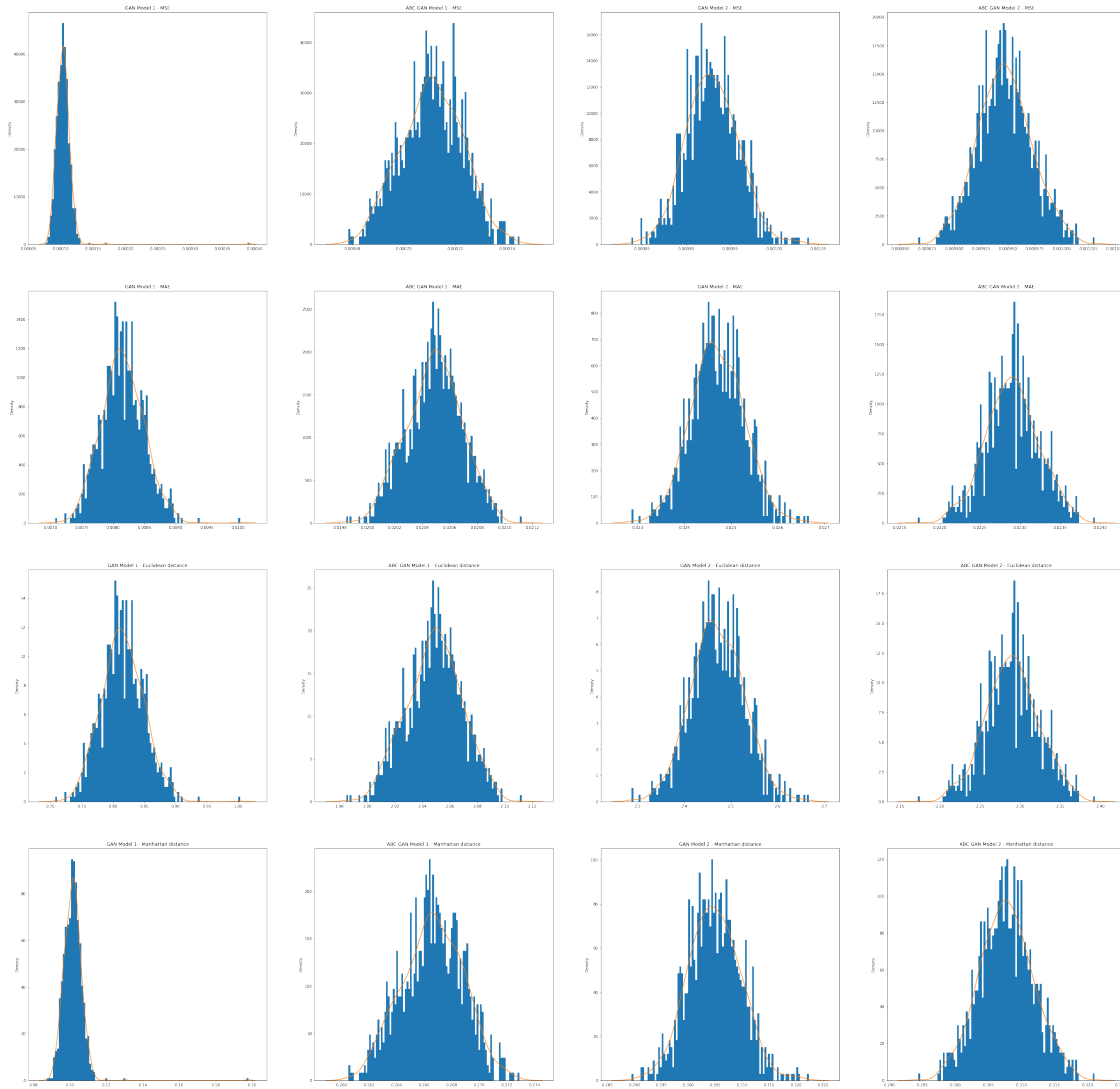
```
[24]: ABC_GAN2_metrics=ABC_train_test.
      ↪ test_generator_2(gen2,real_dataset,coeff,mean,variance,device)
```

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



3 Model Analysis

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



```
[ ]:
```