Dataset1-Regression_output_7

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
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 = 1
variance = 0.1
```

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 $\sigma = 0.1$

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

```
         X1        X2        X3        X4        X5        X6        X7  \
0 -2.201071 -0.949235 -1.710333 -0.733789  0.225914 -1.586271  0.360373
1 -0.878893 -0.695840 -0.581143  1.355534  0.626669  0.052689  0.884889
2 -1.401811  0.615003 -0.826774  0.009859  1.124991 -0.870255 -0.679171
3  1.487213 -0.980743 -0.474706  0.610891 -0.680719 -0.419187  0.525665
4 -0.400772 -2.163843  0.667464 -0.785645 -1.956020 -1.398659 -0.168380

         X8        X9       X10             Y
```

```
0  0.409588  1.180020  0.560986 -343.076308
1  0.909236  0.835298 -2.824454  -63.930628
2 -0.716709 -1.314967 -0.468699 -200.048609
3 -1.431232  0.173508 -0.696140  -57.252648
4 -0.331767  0.183969  1.108277 -404.282951
```

1.5 Stats Model

[6]:

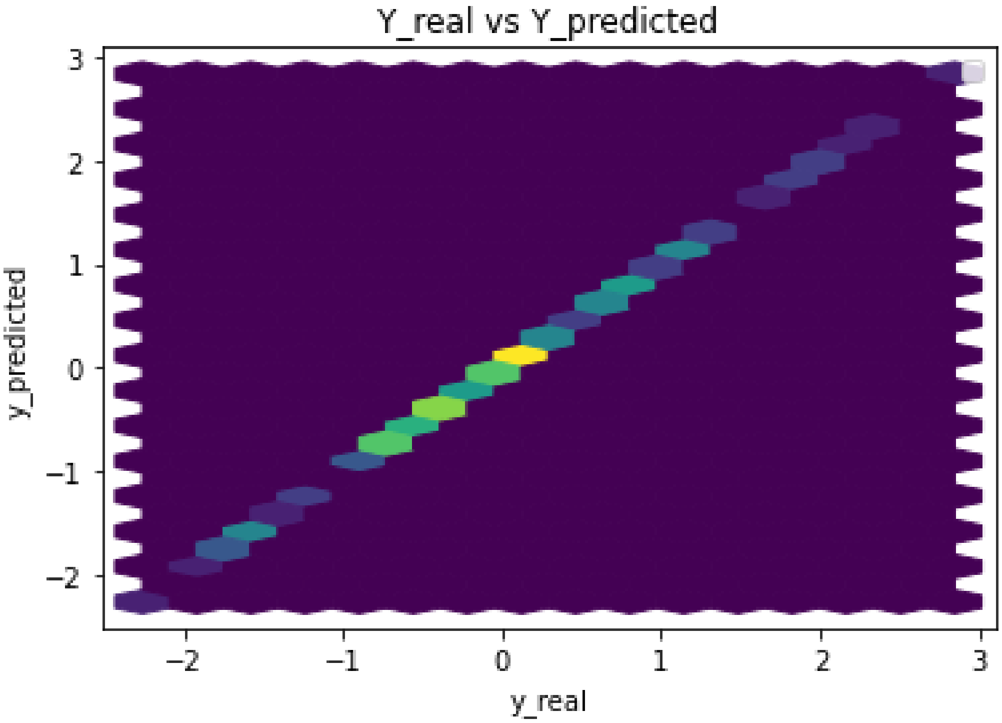
```
[coeff,y_pred] = statsModel.statsModel(X,Y)
```

```
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:                 3.355e+07
Date:                Tue, 02 Nov 2021   Prob (F-statistic):          4.62e-288
Time:                        18:25:15   Log-Likelihood:                 615.22
No. Observations:                 100   AIC:                            -1208.
Df Residuals:                      89   BIC:                            -1180.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -3.123e-17   5.46e-05  -5.72e-13      1.000      -0.000       0.000
x1             0.4842   5.72e-05   8463.913      0.000       0.484       0.484
x2             0.2951   5.81e-05   5076.255      0.000       0.295       0.295
x3             0.1377   5.88e-05   2344.277      0.000       0.138       0.138
x4             0.4339   5.82e-05   7457.312      0.000       0.434       0.434
x5             0.4341   5.65e-05   7684.290      0.000       0.434       0.434
x6             0.3759   5.84e-05   6434.277      0.000       0.376       0.376
x7             0.0425   5.83e-05    728.764      0.000       0.042       0.043
x8             0.1985   5.74e-05   3455.772      0.000       0.198       0.199
x9             0.2406   5.76e-05   4177.925      0.000       0.240       0.241
x10            0.3300   5.79e-05   5696.238      0.000       0.330       0.330
==============================================================================
Omnibus:                        2.631   Durbin-Watson:                   2.097
Prob(Omnibus):                  0.268   Jarque-Bera (JB):                2.633
Skew:                           0.377   Prob(JB):                        0.268
Kurtosis:                       2.747   Cond. No.                         1.65
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const   -3.122502e-17
x1       4.842265e-01
```

```
x2       2.950780e-01
x3       1.377265e-01
x4       4.339240e-01
x5       4.341308e-01
x6       3.759077e-01
x7       4.245644e-02
x8       1.984919e-01
x9       2.405761e-01
x10      3.300000e-01
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 2.653106135260218e-07
Mean Absolute Error: 0.0004143462646783684
Manhattan distance: 0.04143462646783684
Euclidean distance: 0.0051508311322156705
```

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))
```

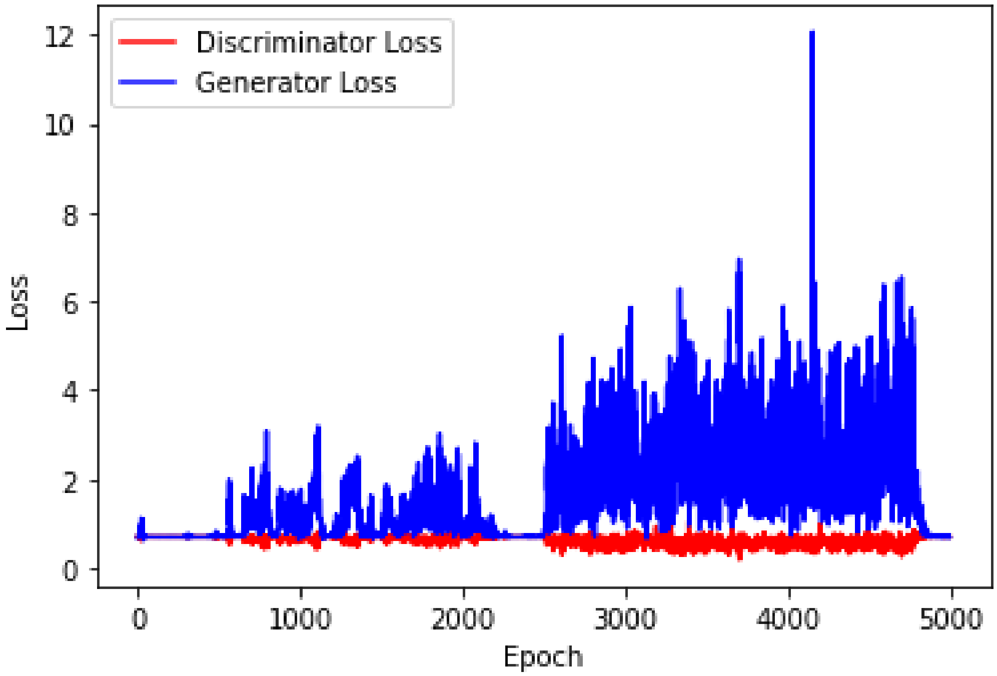
[10]:

```
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)
```

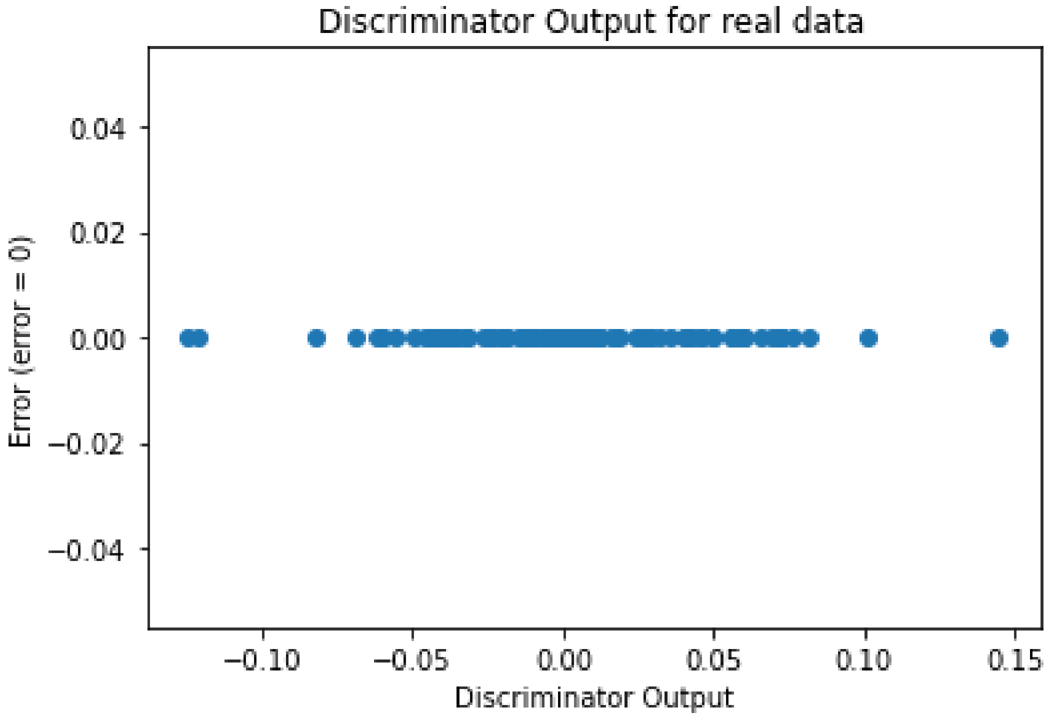


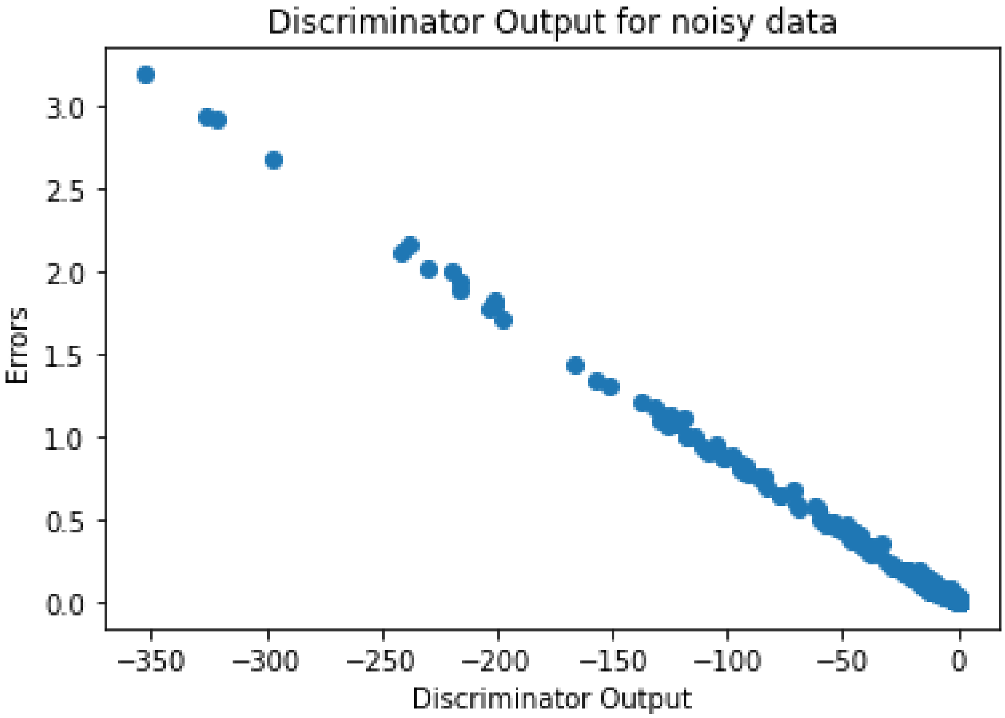
[12]:

```
GAN1_metrics = train_test.test_generator(generator,real_dataset,device)
```

[13]:

```
sanityChecks.discProbVsError(real_dataset,discriminator,device)
```


Discriminator Output for real data



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

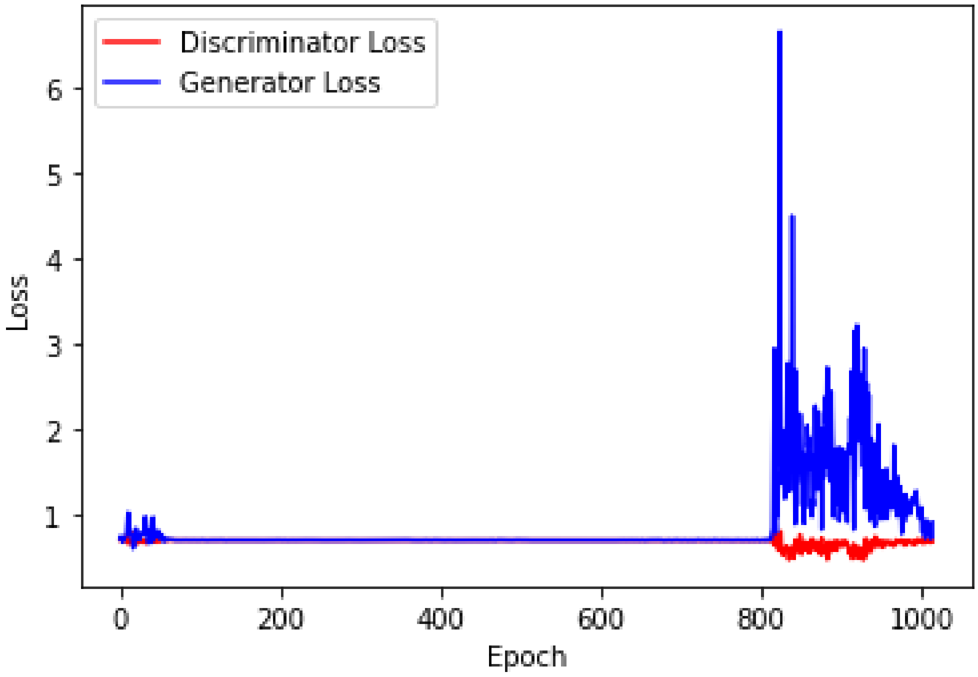
[14]:

```
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=(0.5, 0.
 ↪999))
```

[15]:

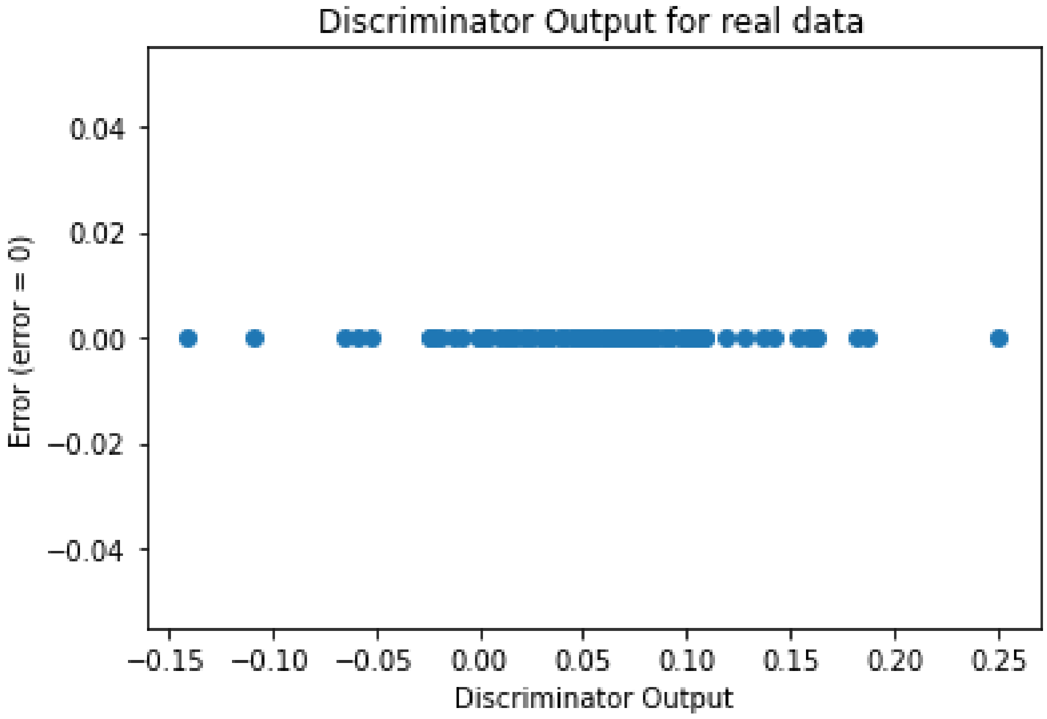
```
train_test.
 ↪training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,real_dataset,batch_size,error,cri
```

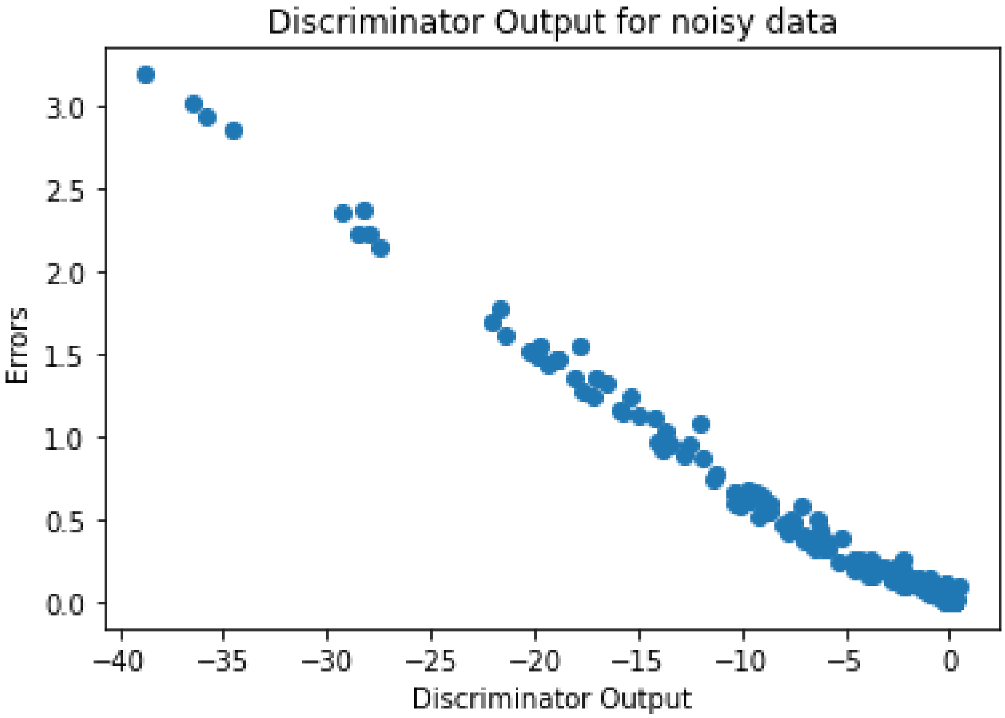
```
Number of epochs needed 1014
```



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

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





2 ABC GAN Model

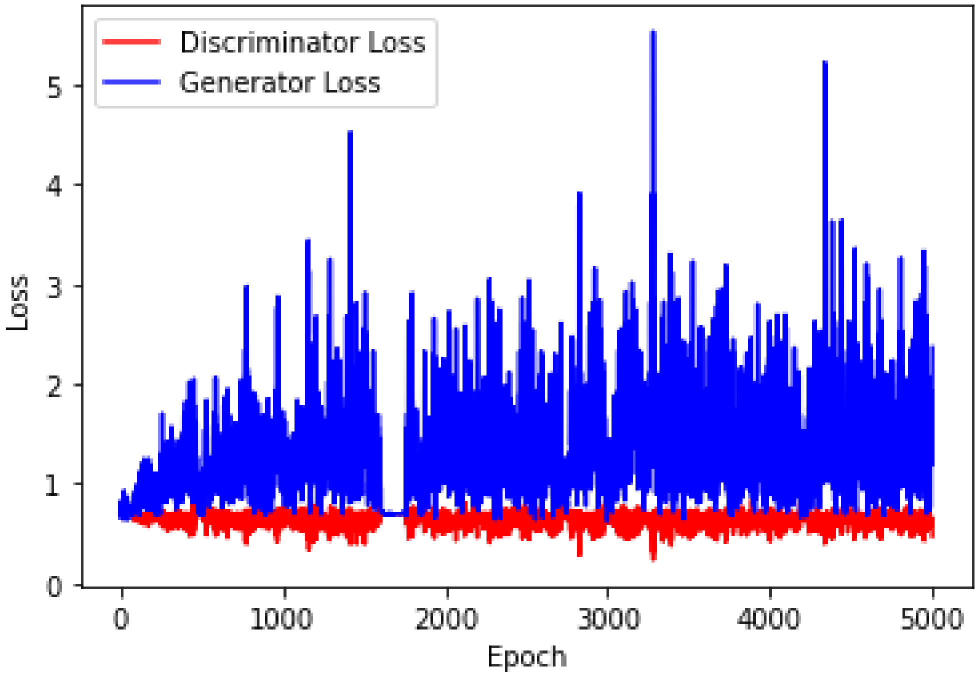
2.0.1 Training the network

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))
```

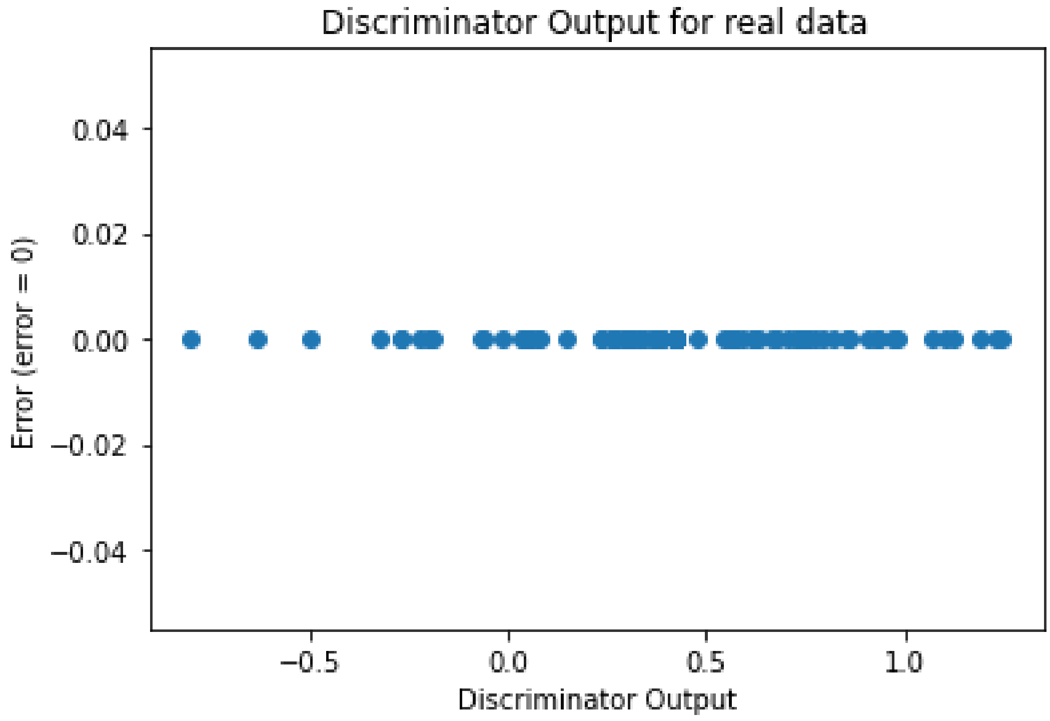
```
[19]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
       ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```

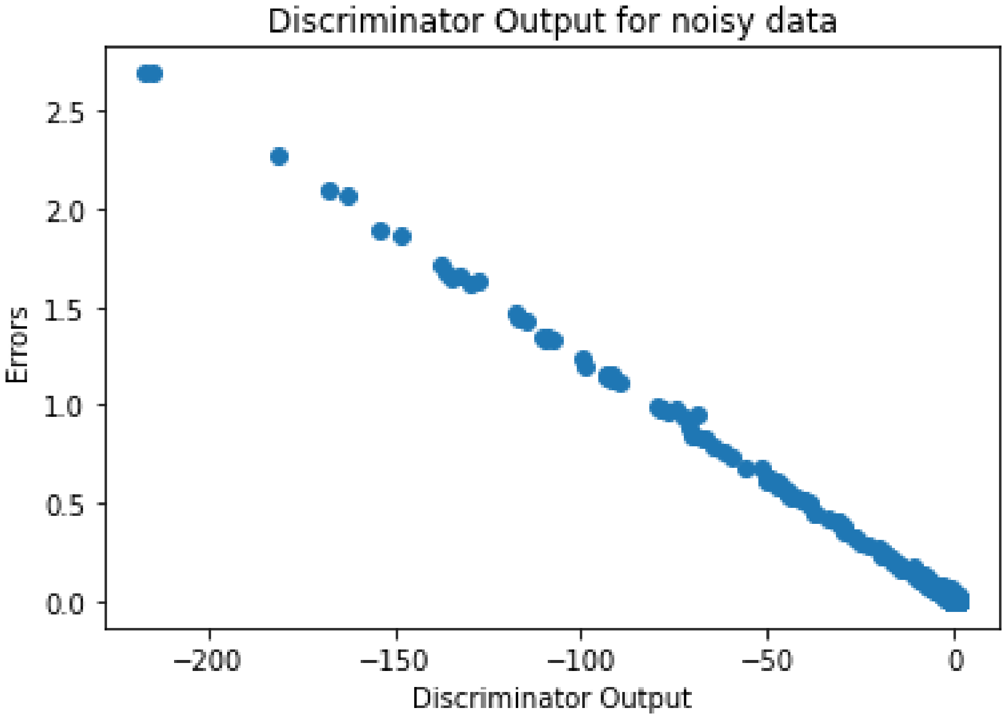


[20]:
```
ABC_GAN1_metrics=ABC_train_test.
 ↪test_generator(gen,real_dataset,coeff,mean,variance,device)
```

Sanity Checks

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


Discriminator Output for real data
0.04
0.02
0.00
−0.02
−0.04
Error (error = 0)
−0.5
0.0
0.5
1.0
Discriminator Output


Discriminator Output for noisy data
2.5
2.0
1.5
1.0
0.5
0.0
Errors
−200
−150
−100
−50
0
Discriminator Output

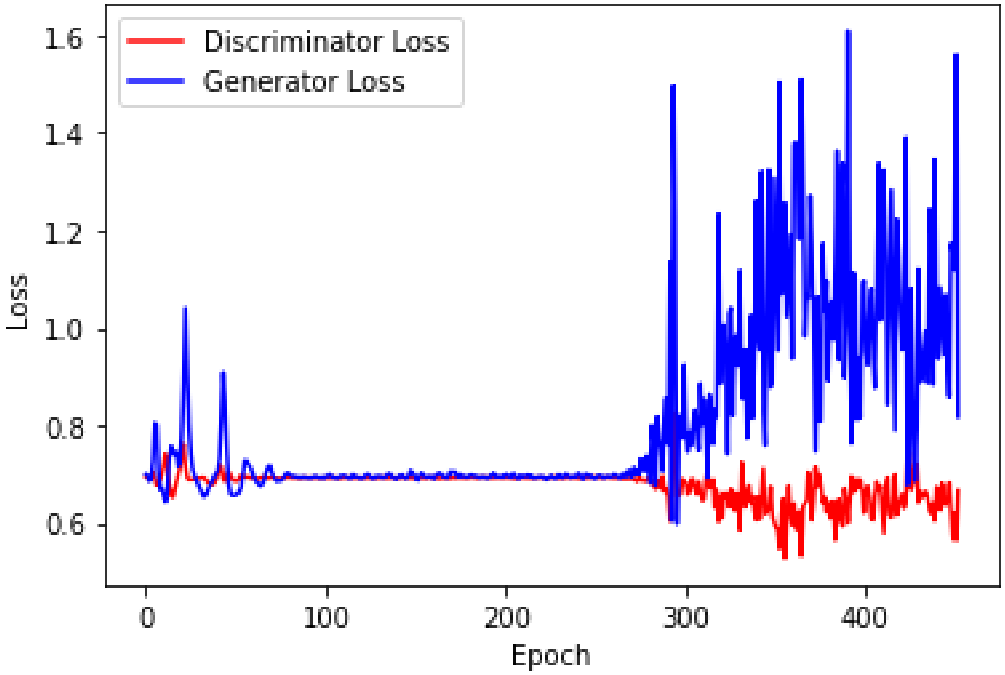
Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

[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))
```

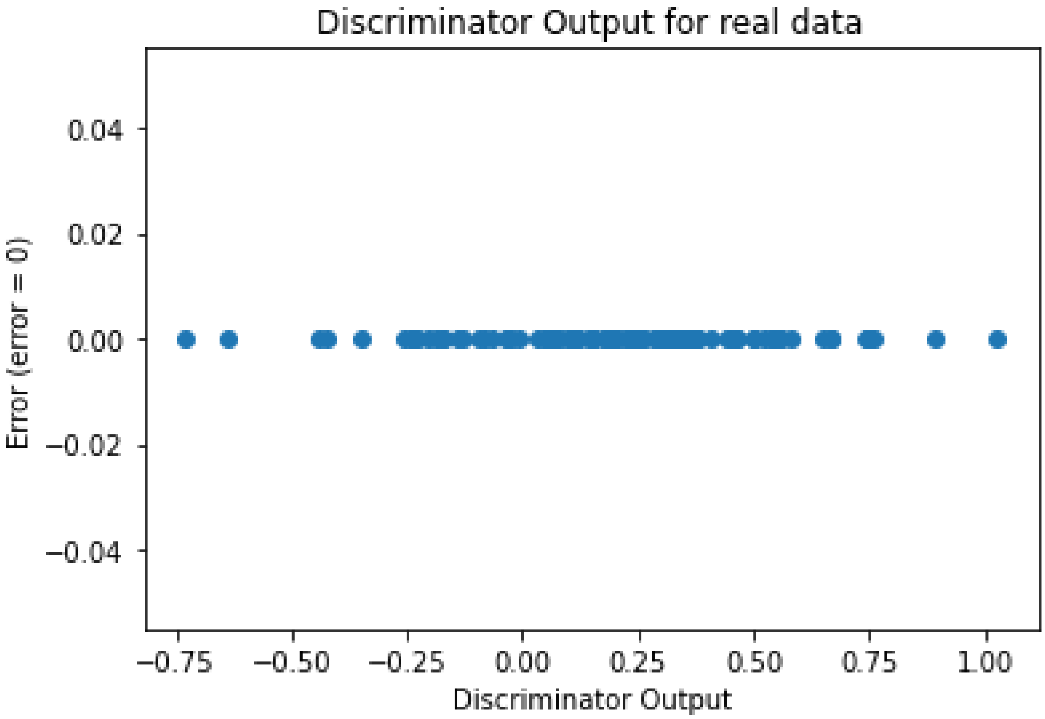
[23]:
```
ABC_train_test.
 ↪training_GAN_2(disc2,gen2,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪error,criterion,coeff,mean,variance,device)
```

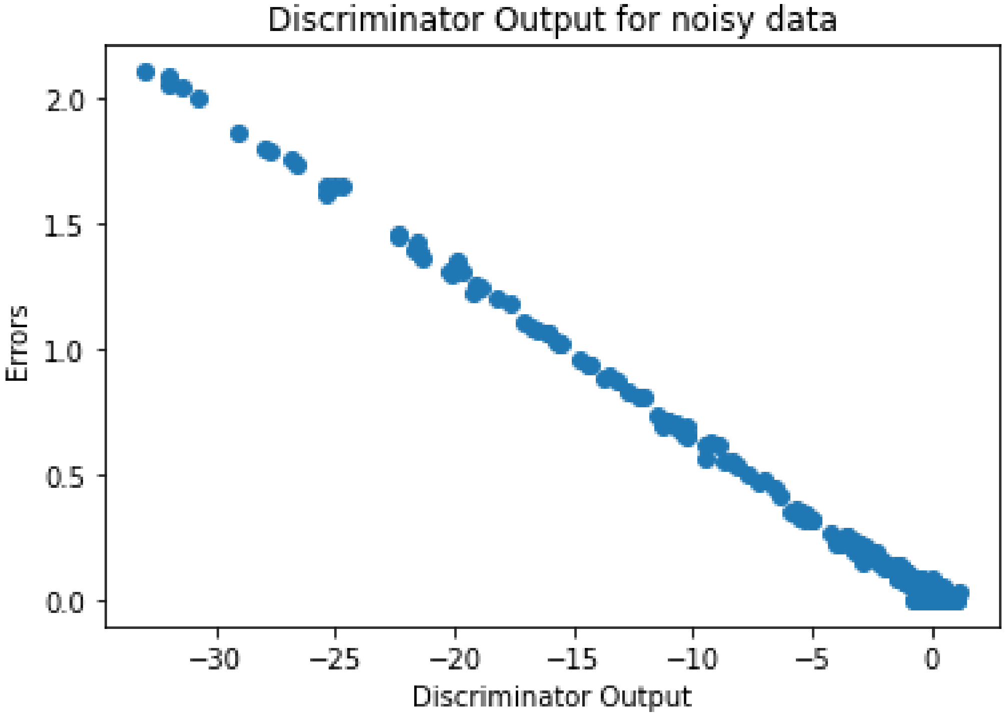
```
Number of epochs 452
```



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

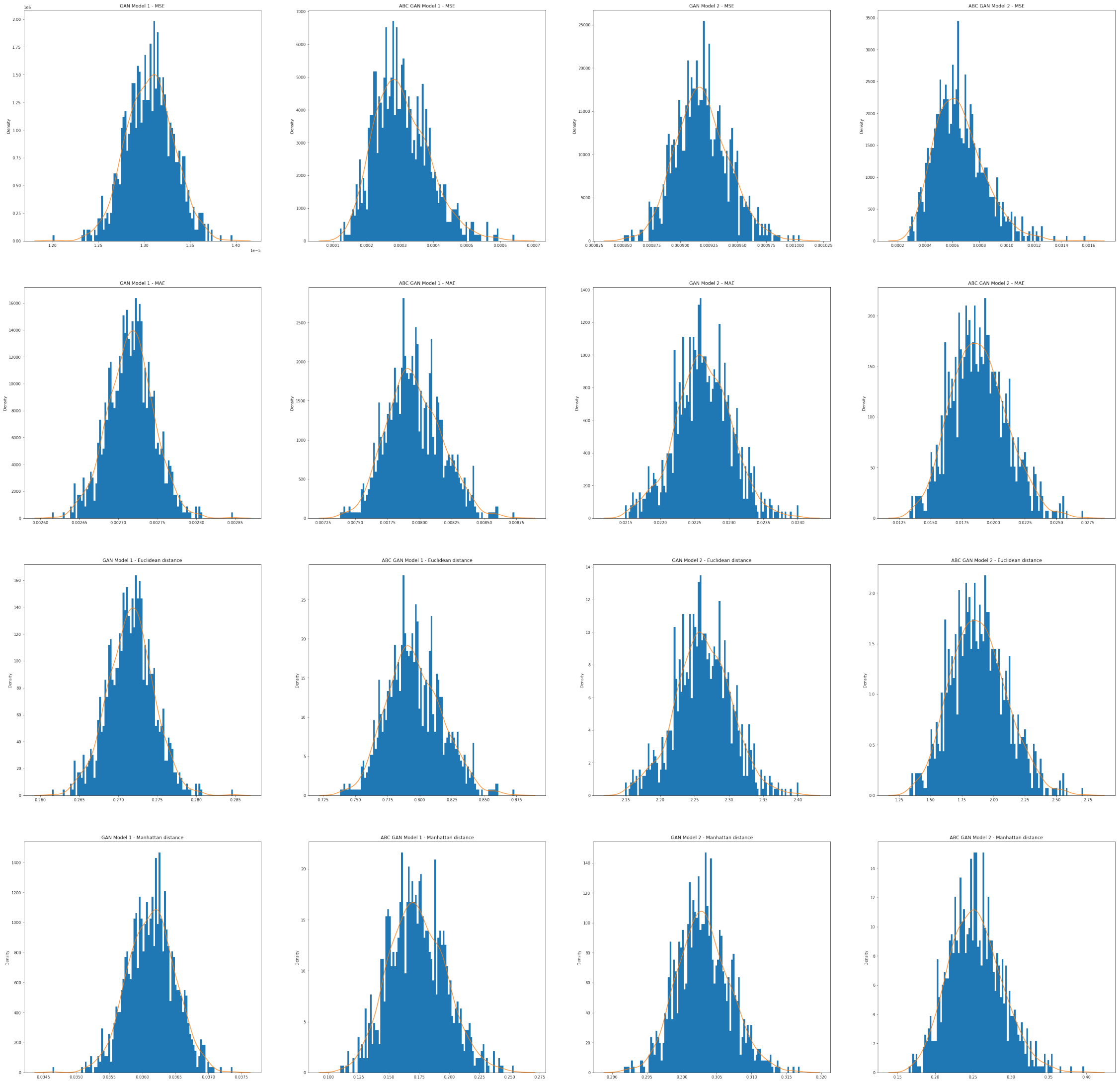
[25]:
```
sanityChecks.discProbVsError(real_dataset,disc2,device)
```


Discriminator Output for real data
0.04
0.02
0.00
−0.02
−0.04
Error (error = 0)
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Discriminator Output


Discriminator Output for noisy data
2.0
1.5
1.0
0.5
0.0
Errors
−30
−25
−20
−15
−10
−5
0
Discriminator Output

3 Model Analysis

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



```
[ ]:
```