Dataset3-Boston_output_11

November 17, 2021

1 Dataset 4 - Boston

1.1 Import Libraries

[1]:
```
import warnings
import sys
sys.path.insert(0, '../src')
warnings.filterwarnings('ignore')
```

[2]:
```
import train_test
import ABC_train_test
import bostonDataset
import network
import statsModel
import performanceMetrics
import dataset
import sanityChecks
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from torch.utils.data import random_split
#import pycuda.driver as cuda
```

1.2 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 = 13
n_samples= 506

#ABC Generator Parameters
mean = 1
variance = 0.01
```

```
#Hyper-parameters
n_epochs = 5000
```

[4]:
```
# Parameters
mean = 0
variance = 0.01
n_epochs = 8000
```

1.3 Dataset

[5]:
```
X,Y = bostonDataset.boston_data()
```

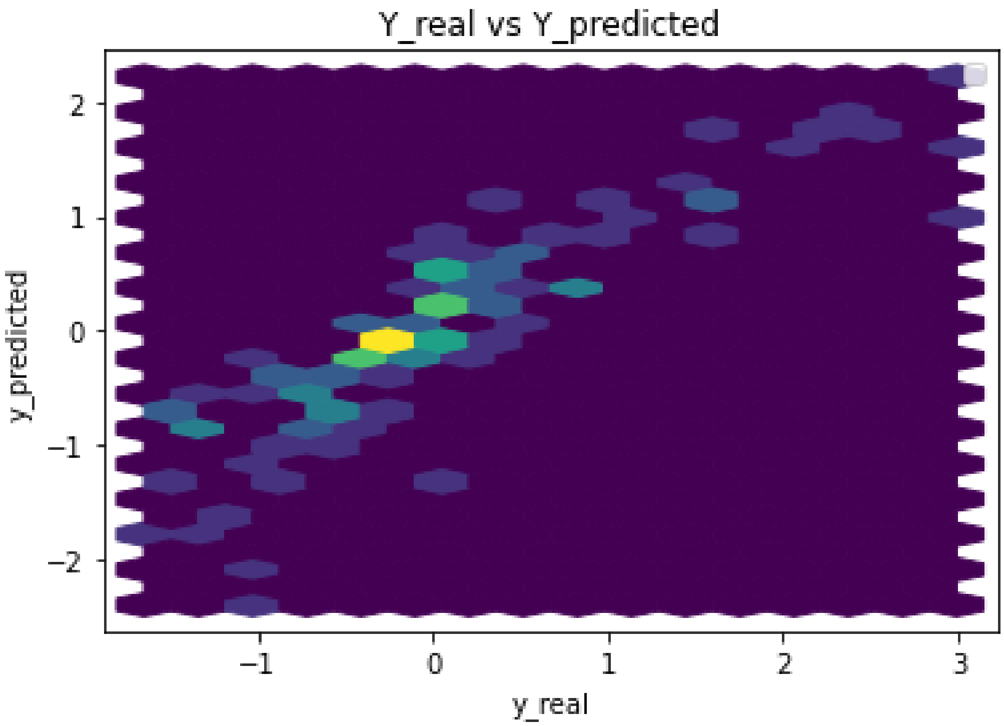
```
        X1    X2    X3   X4     X5     X6    X7      X8   X9    X10   X11  \
0  0.00632  18.0  2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7

      X12   X13     Y
0  396.90  4.98  24.0
1  396.90  9.14  21.6
2  392.83  4.03  34.7
3  394.63  2.94  33.4
4  396.90  5.33  36.2
```

1.4 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 (uncentered):
0.735
Model:                            OLS   Adj. R-squared (uncentered):
0.726
Method:                 Least Squares   F-statistic:
83.48
Date:                Wed, 17 Nov 2021   Prob (F-statistic):
4.48e-104
Time:                        21:02:35   Log-Likelihood:
-305.37
No. Observations:                 404   AIC:
636.7
Df Residuals:                     391   BIC:
```

```
688.8
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.1097      0.032     -3.401      0.001      -0.173      -0.046
x2             0.0935      0.039      2.375      0.018       0.016       0.171
x3             0.0177      0.054      0.326      0.745      -0.089       0.124
x4             0.0619      0.026      2.345      0.020       0.010       0.114
x5            -0.2197      0.054     -4.090      0.000      -0.325      -0.114
x6             0.3007      0.036      8.348      0.000       0.230       0.372
x7            -0.0063      0.045     -0.138      0.890      -0.096       0.083
x8            -0.3244      0.052     -6.205      0.000      -0.427      -0.222
x9             0.3294      0.072      4.574      0.000       0.188       0.471
x10           -0.2620      0.081     -3.227      0.001      -0.422      -0.102
x11           -0.2338      0.034     -6.805      0.000      -0.301      -0.166
x12            0.0763      0.030      2.528      0.012       0.017       0.136
x13           -0.4024      0.044     -9.179      0.000      -0.489      -0.316
==============================================================================
Omnibus:                      160.269   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              809.430
Skew:                           1.644   Prob(JB):                    1.72e-176
Kurtosis:                       9.105   Cond. No.                         10.1
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  x1    -0.109671
x2     0.093485
x3     0.017668
x4     0.061851
x5    -0.219732
x6     0.300694
x7    -0.006273
x8    -0.324388
x9     0.329439
x10   -0.261971
x11   -0.233818
x12    0.076281
x13   -0.402415
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 0.23996053976839402
Mean Absolute Error: 0.3573778496298103
Manhattan distance: 36.45254066224066
Euclidean distance: 4.947319987263426
```

1.5 Common Training Parameters (GAN & ABC_GAN)

[7]:
```
threshold_mse = 0.99
batch_size = 100
```

[8]:
```
# Train test split for dataset
real_dataset = dataset.CustomDataset(X,Y)
train_size = round(0.8 * n_samples)
test_size = n_samples - train_size
train_data, test_data = random_split(real_dataset,[train_size,test_size])
```

[9]:
```
# cuda.init()
# ## Get Id of default device
# torch.cuda.current_device()
# #0
# cuda.Device(0).name()
```

[10]:
```
#Select the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

1.6 GAN Model

Training GAN for n_epochs number of epochs

[11]:
```
generator = network.Generator(n_features+1).to(device)
discriminator = network.Discriminator(n_features+1).to(device)

criterion = torch.nn.BCELoss()
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))
```

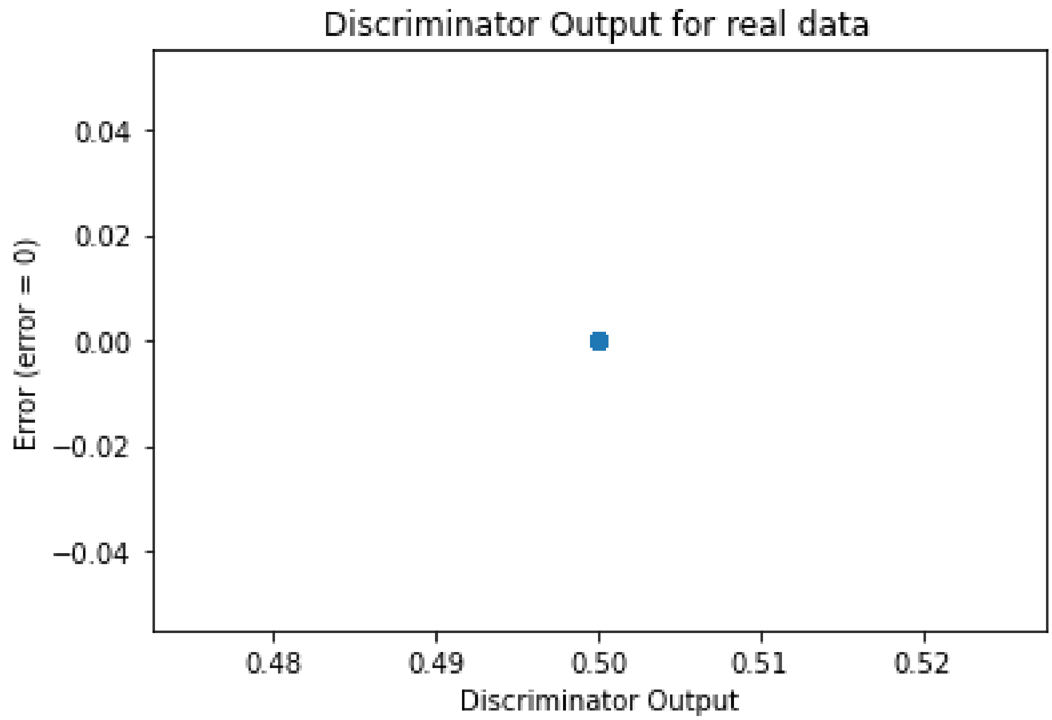
[12]:
```
print(generator)
print(discriminator)
```

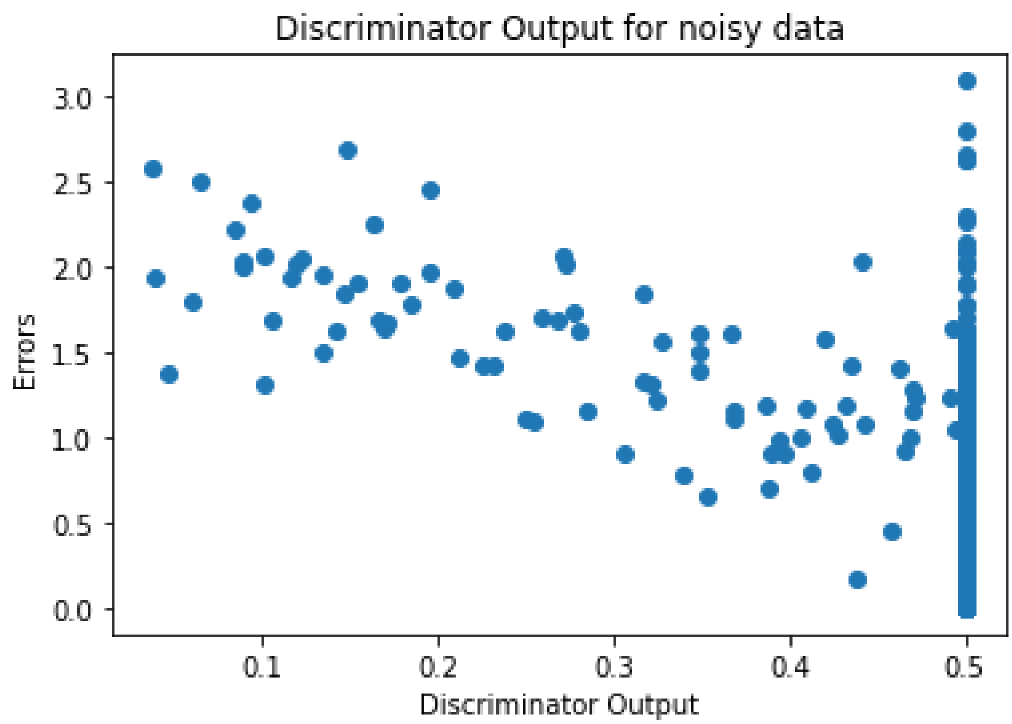
```
Generator(
  (hidden1): Linear(in_features=14, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=14, out_features=25, bias=True)
  (hidden2): Linear(in_features=25, out_features=50, bias=True)
  (output): Linear(in_features=50, out_features=1, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
)
```

[13]:
```
discLossG1,genLossG1 = train_test.
↪training_GAN(discriminator,generator,disc_opt,gen_opt,train_data,batch_size,␣
↪n_epochs,criterion,device)
```

[14]:
```
GAN1_metrics = train_test.test_generator(generator,test_data,device)
```

[15]:
```
sanityChecks.discProbVsError(real_dataset,discriminator,device)
```


Discriminator Output for real data
0.04
0.02
0.00
−0.02
−0.04
Error (error = 0)
0.48
0.49
0.50
0.51
0.52
Discriminator Output


Discriminator Output for noisy data
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Errors
0.1
0.2
0.3
0.4
0.5
Discriminator Output

Training GAN until mse of y__pred is > baseline__mse or n__epochs < 5000

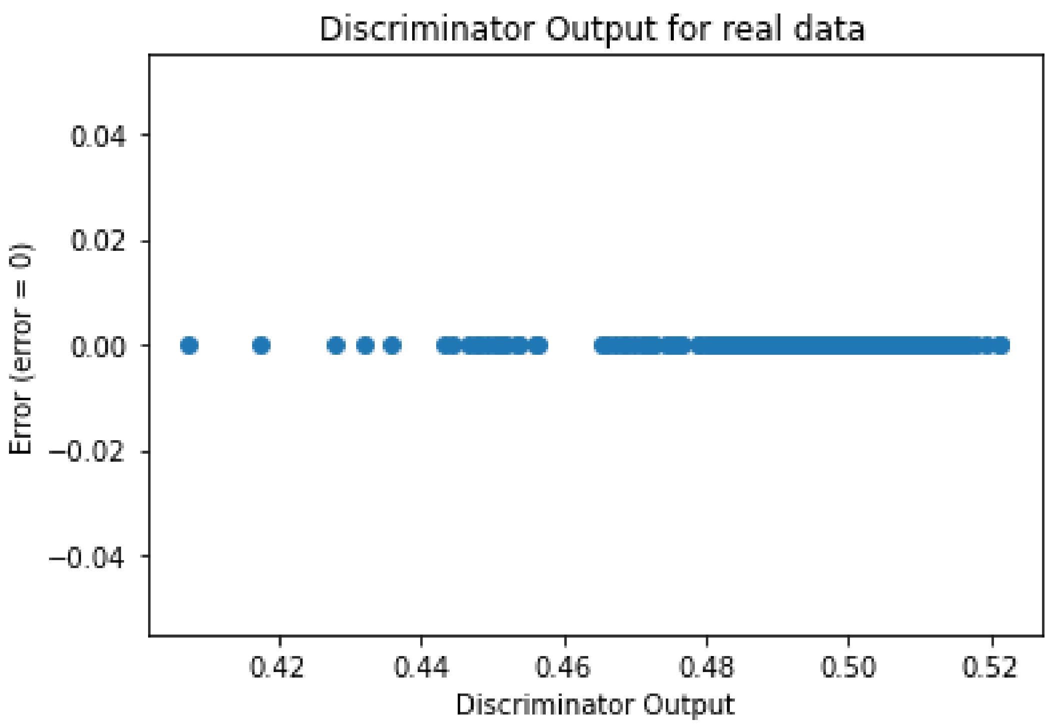
[16]:
```
generator2 = network.Generator(n_features+1).to(device)
discriminator2 = network.Discriminator(n_features+1).to(device)
criterion = torch.nn.BCELoss()
gen_opt = torch.optim.Adam(generator2.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator2.parameters(), lr=0.01, betas=(0.5, 0.
↪999))
```

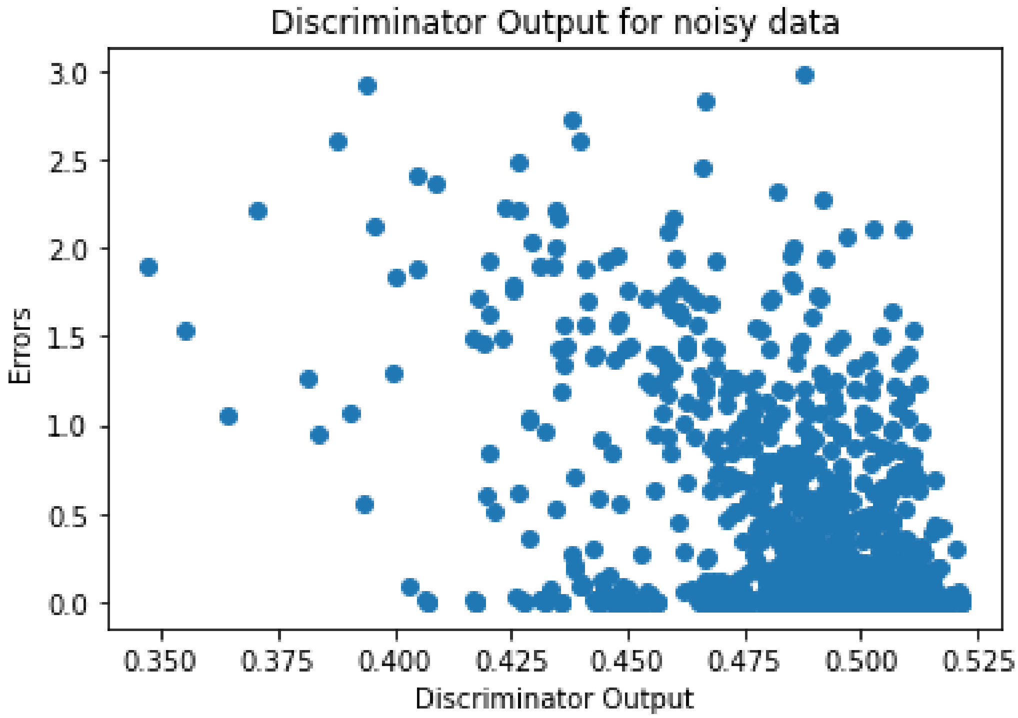
[17]:
```
discLossG2,genLossG2 = train_test.
↪training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,train_data,test_data,batch_size,t
```

```
Number of epochs needed 6
```

[18]:
```
GAN2_metrics=train_test.test_generator_2(generator2,test_data,device)
```

[19]:
```
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

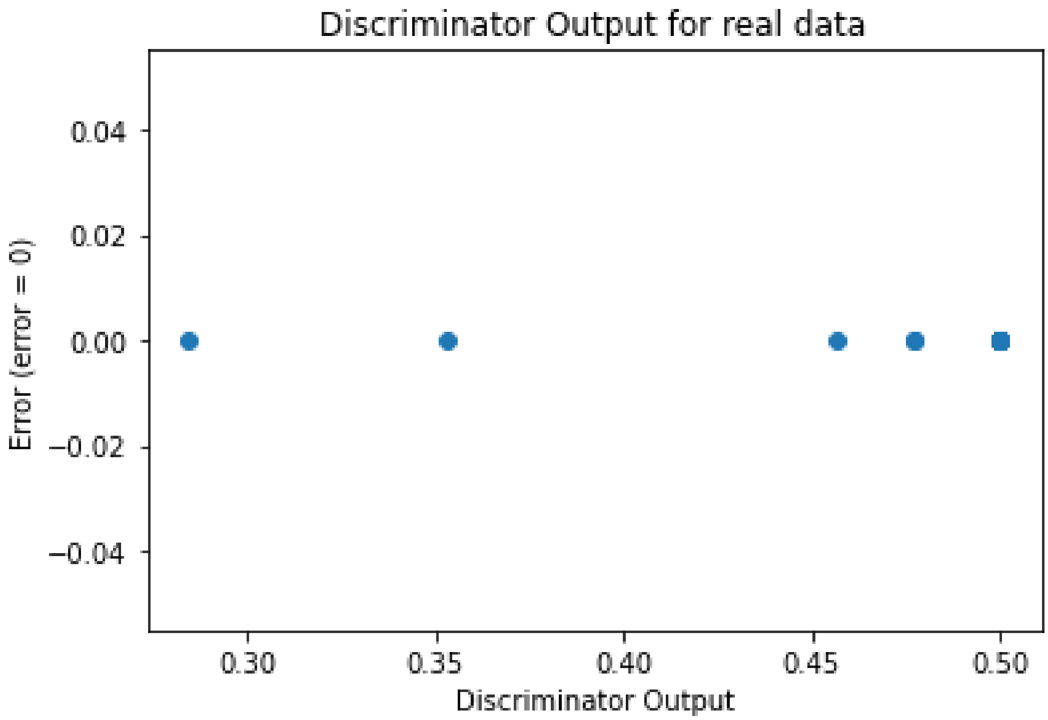
[20]:
```
gen = network.Generator(n_features+1).to(device)
disc = network.Discriminator(n_features+1).to(device)

criterion = torch.nn.BCELoss()
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))
```

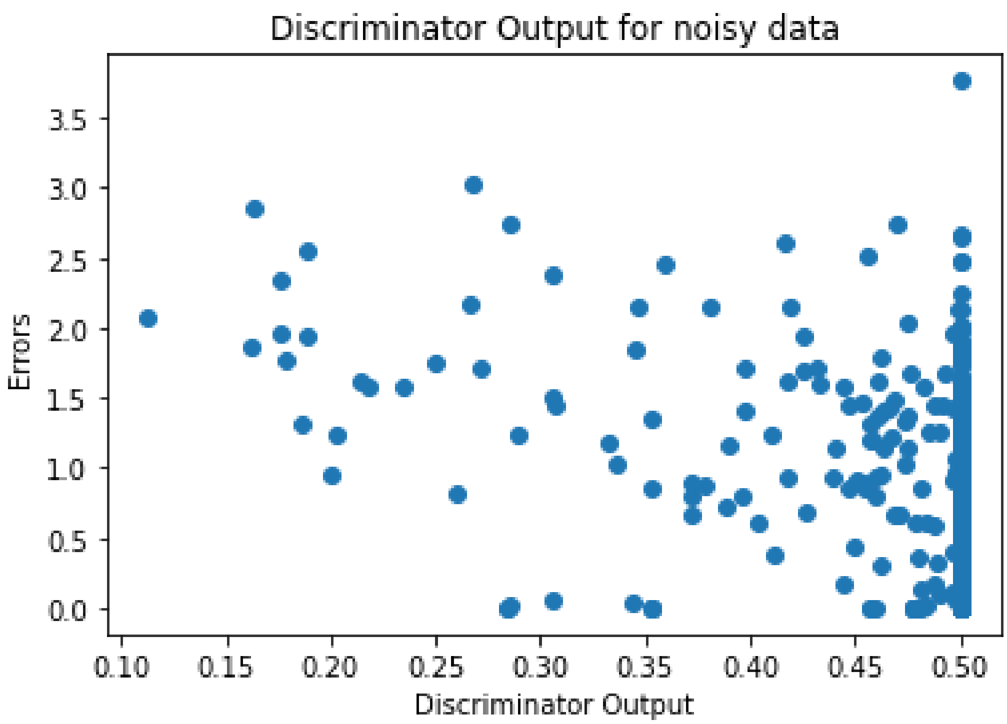
[21]:
```
discLossA1,genLossA1 = ABC_train_test.training_GAN(disc,␣
 ↪gen,disc_opt,gen_opt,train_data,batch_size,␣
 ↪n_epochs,criterion,coeff,mean,variance,device)
```

[22]:
```
ABC_GAN1_metrics=ABC_train_test.
 ↪test_generator(gen,test_data,coeff,mean,variance,device)
```

Sanity Checks

[23]:
```
sanityChecks.discProbVsError(real_dataset,disc,device)
```


Discriminator Output for real data
0.04
0.02
0.00
−0.02
−0.04
Error (error = 0)
0.30
0.35
0.40
0.45
0.50
Discriminator Output


Discriminator Output for noisy data
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Errors
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Discriminator Output

Training ABC-GAN until mse of y_pred is > baseline_mse or n_epochs < 5000

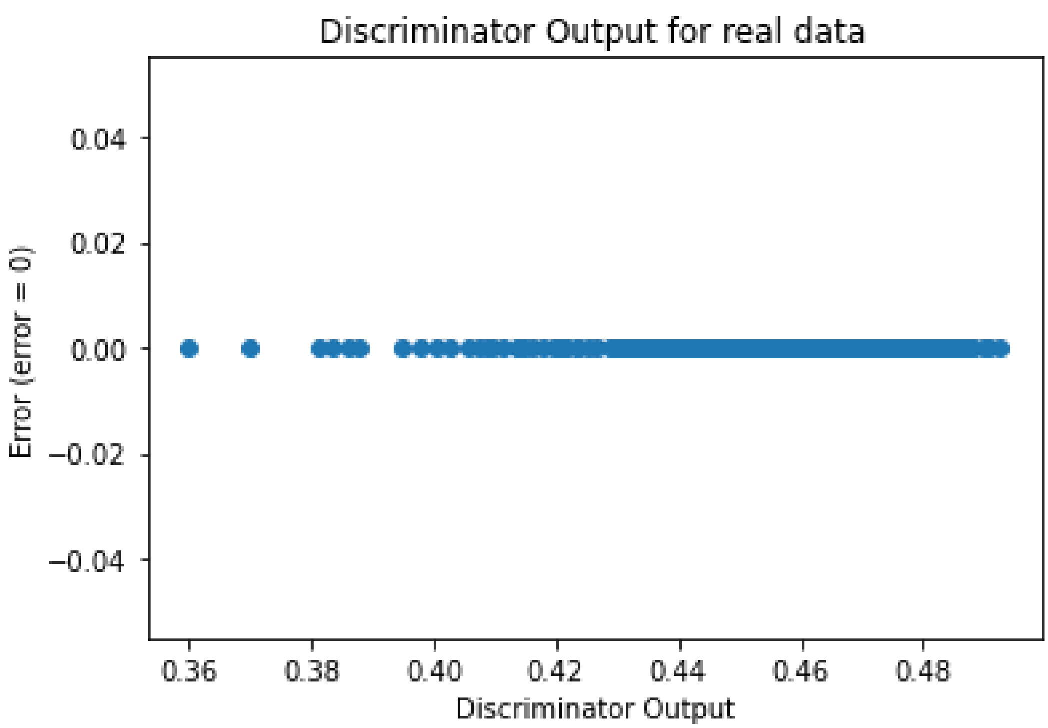
[24]:
```
gen2 = network.Generator(n_features+1).to(device)
disc2 = network.Discriminator(n_features+1).to(device)

criterion = torch.nn.BCELoss()
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))
```

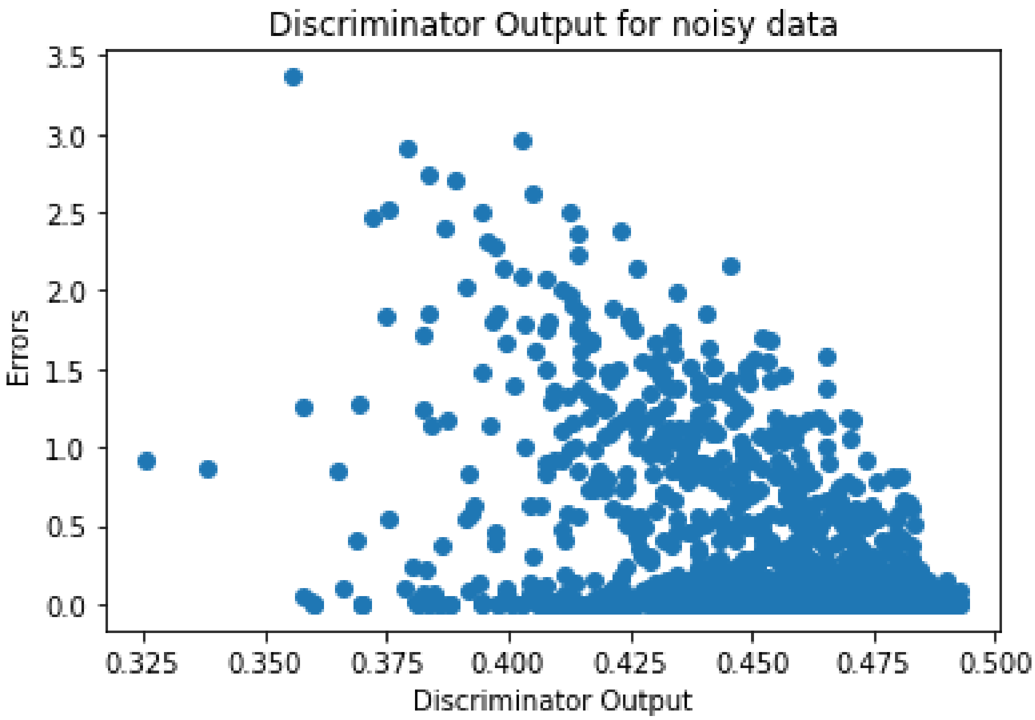
[25]:
```
discLossA2,genLossA2 = ABC_train_test.
↪training_GAN_2(disc2,gen2,disc_opt,gen_opt,train_data,test_data,batch_size,threshold_mse,cr
```

```
Number of epochs 5
```

[26]:
```
ABC_GAN2_metrics=ABC_train_test.
↪test_generator_2(gen2,test_data,coeff,mean,variance,device)
```

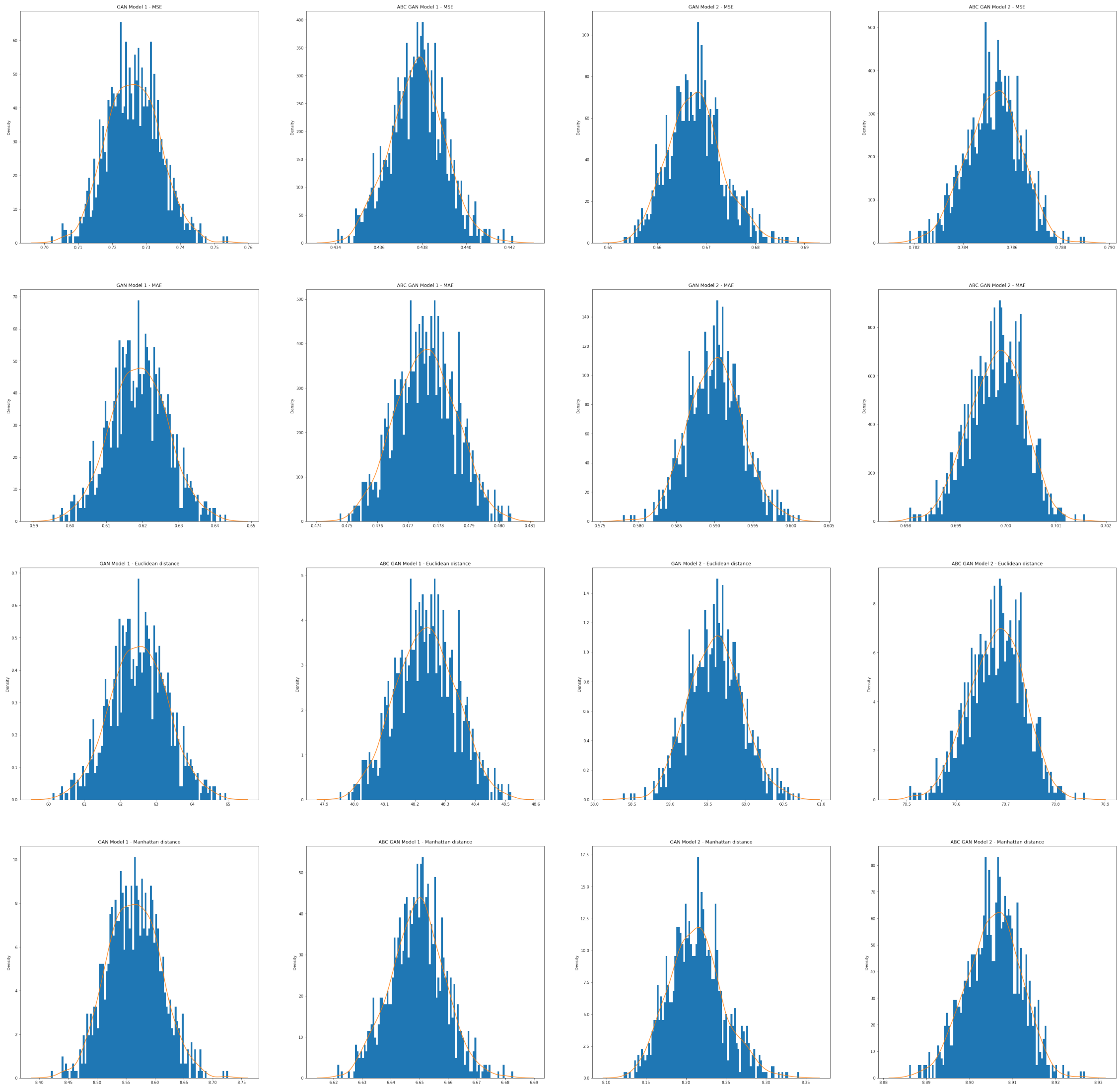
[27]:
```
sanityChecks.discProbVsError(real_dataset,disc2,device)
```



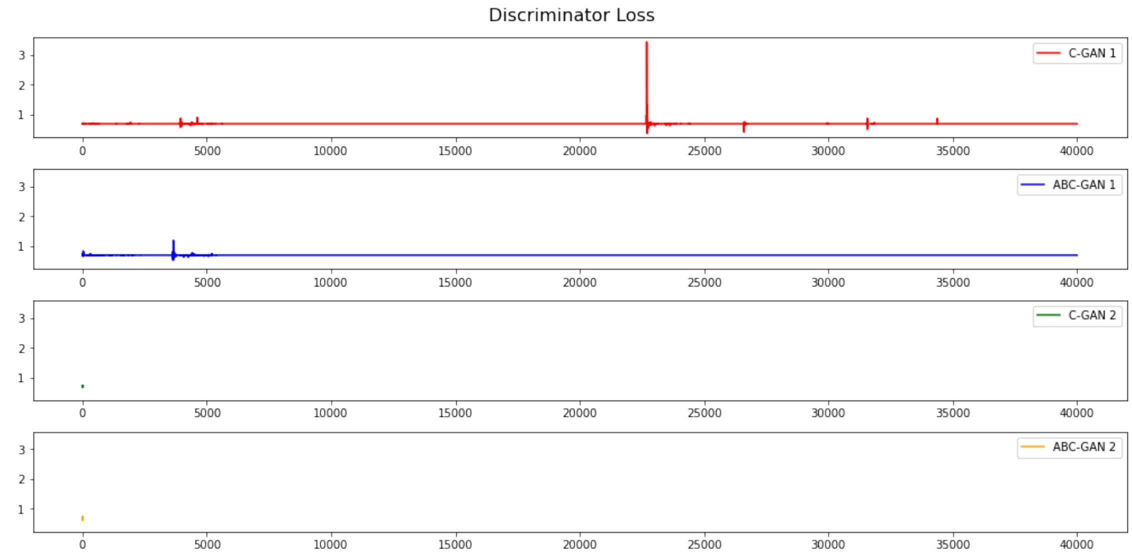


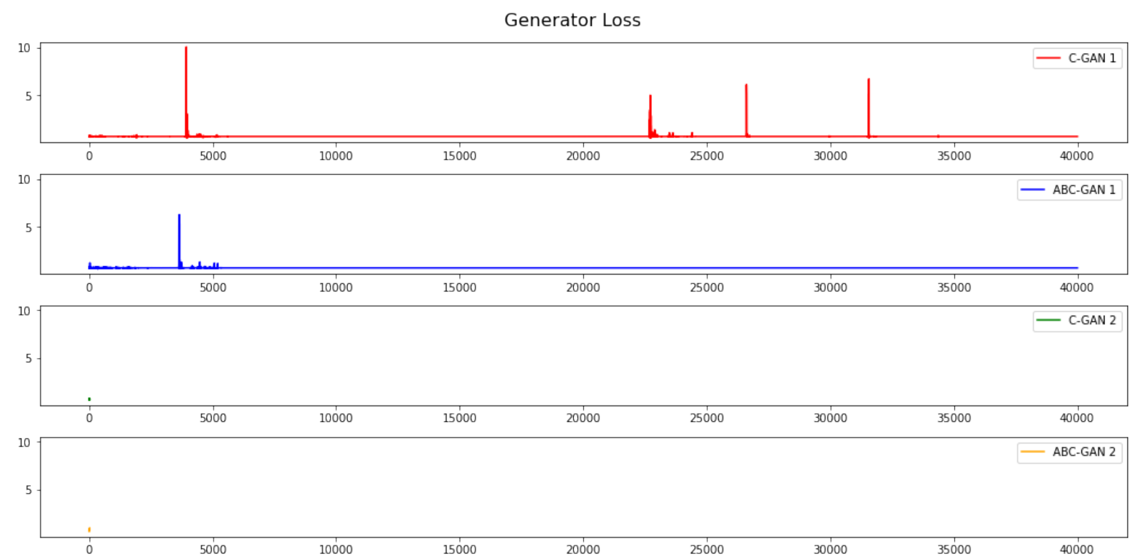
3 Model Analysis

[28]:
```
performanceMetrics.
↪modelAnalysis(GAN1_metrics,ABC_GAN1_metrics,GAN2_metrics,ABC_GAN2_metrics)
```



[29]:
```
performanceMetrics.
 ↪plotTrainingLoss(discLossG1,genLossG1,discLossA1,genLossA1,discLossG2,genLossG2,discLossA2,
```





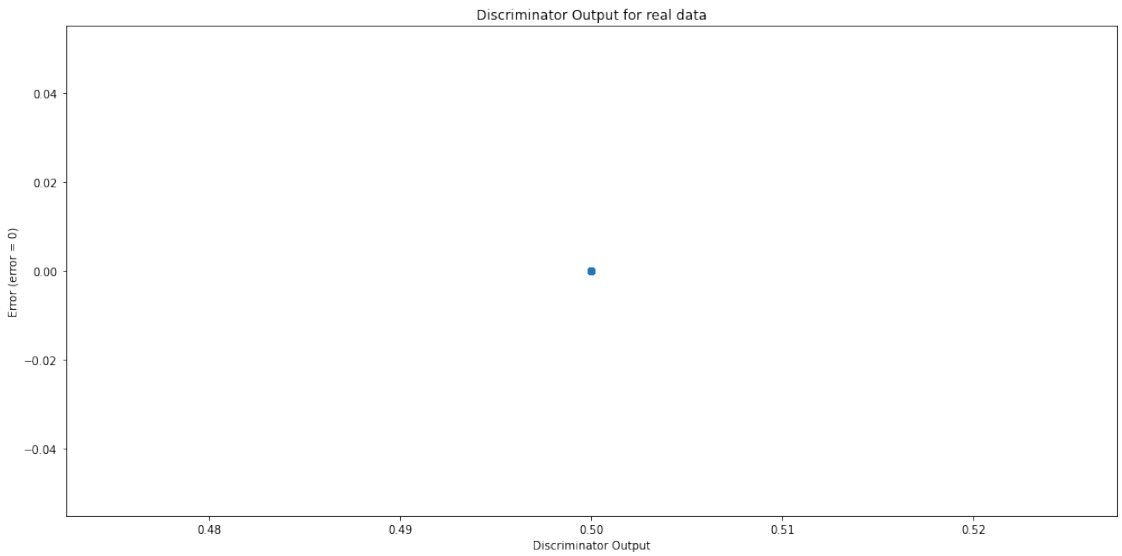
3.1 GAN Model with skip connection

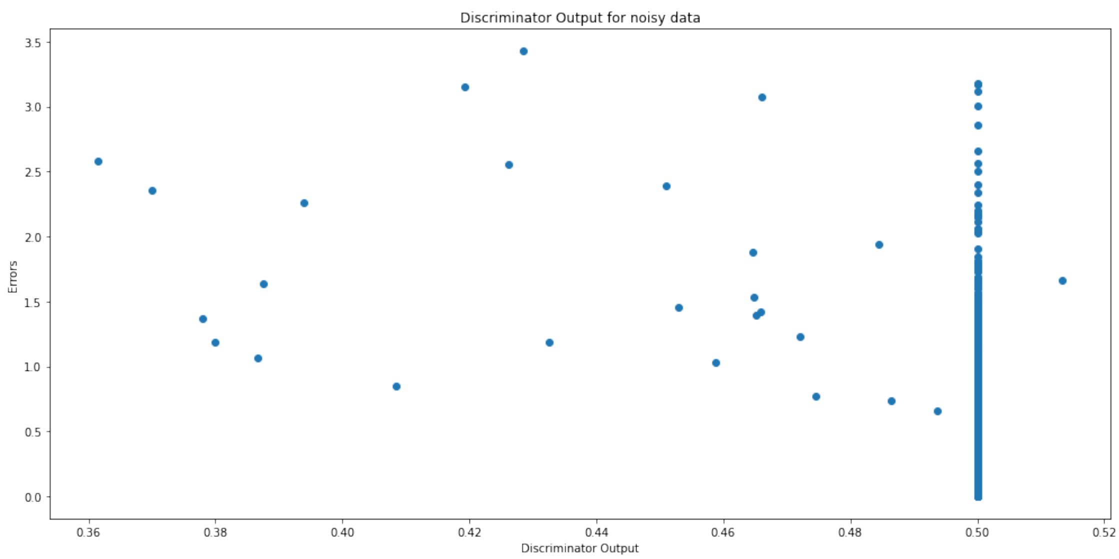
[30]:
```
generator3 = network.GeneratorWithSkipConnection(n_features+1).to(device)
discriminator3 = network.Discriminator(n_features+1).to(device)

criterion = torch.nn.BCELoss()
gen_opt = torch.optim.Adam(generator3.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator3.parameters(), lr=0.01, betas=(0.5, 0.
↪999))
```

[31]:
```
discLossG3,genLossG3 = train_test.training_GAN(discriminator3,generator3␣
↪,disc_opt,gen_opt,train_data,batch_size, n_epochs,criterion,device)
```

[32]:
```
GAN3_metrics=ABC_train_test.
↪test_generator(generator3,test_data,coeff,mean,variance,device)
```

[33]:
```
sanityChecks.discProbVsError(real_dataset,discriminator3,device)
```





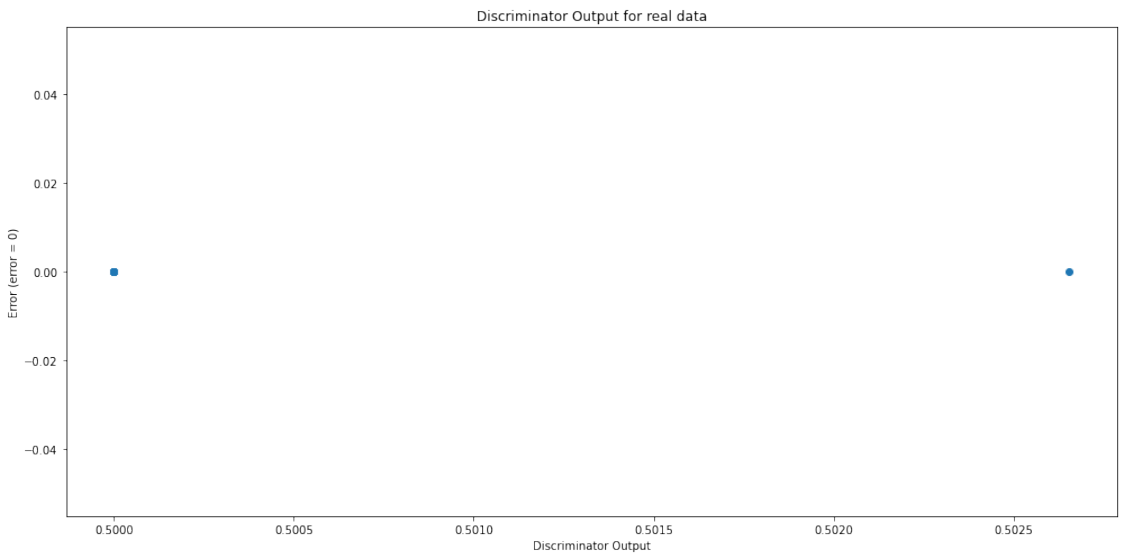
3.2 ABC - GAN Model with skip connection

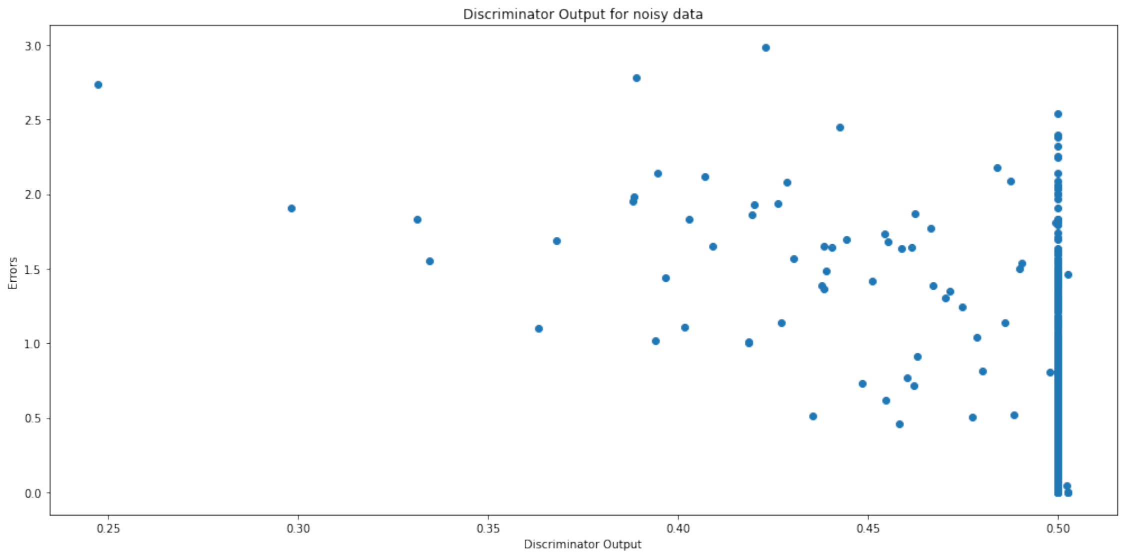
[34]:
```
gen3 = network.GeneratorWithSkipConnection(n_features+1).to(device)
disc3 = network.Discriminator(n_features+1).to(device)

criterion = torch.nn.BCELoss()
gen_opt = torch.optim.Adam(gen3.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(disc3.parameters(), lr=0.01, betas=(0.5, 0.999))
```

[35]:
```
discLossA3,genLossA3 = ABC_train_test.training_GAN(disc3,␣
↪gen3,disc_opt,gen_opt,train_data,batch_size,␣
↪n_epochs,criterion,coeff,mean,variance,device)
```

[36]:
```
ABC_GAN3_metrics=ABC_train_test.
↪test_generator(gen3,test_data,coeff,mean,variance,device)
```

[37]:
```
sanityChecks.discProbVsError(real_dataset,disc3,device)
```





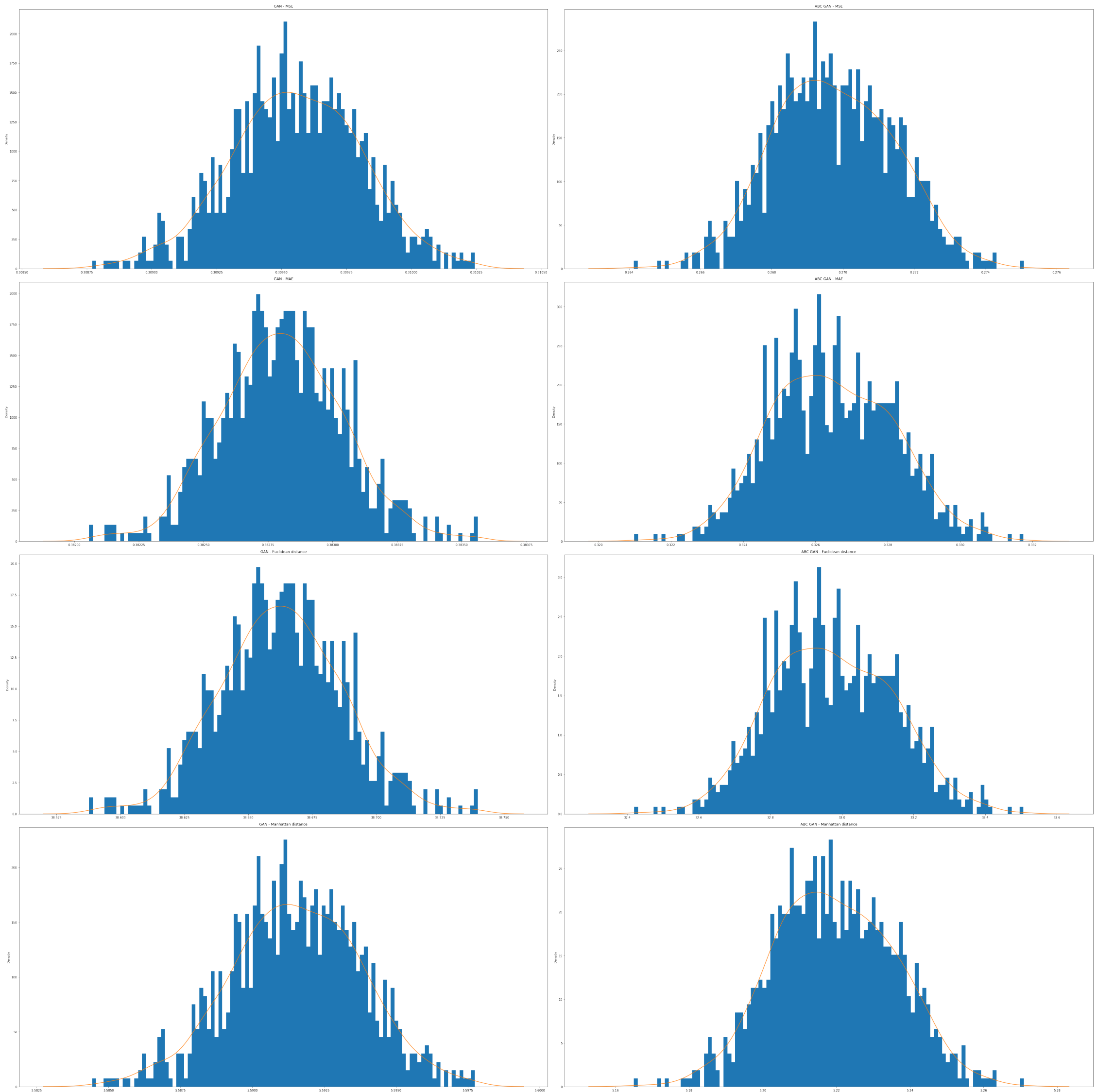
[38]:
```
## Skip Connection Model Analysis - GAN and ABC-GAN
```

[39]:
```
### Weight Analysis

##Study the weights of the skip connection layer
```

[40]:
```
print("GAN Weights")
for name,param in generator3.named_parameters():
    if(name == "skipNode.weight"):
        print(param)
print("ABC-GAN Weights")
for name,param in gen3.named_parameters():
    if(name == "skipNode.weight"):
        print(param)
```

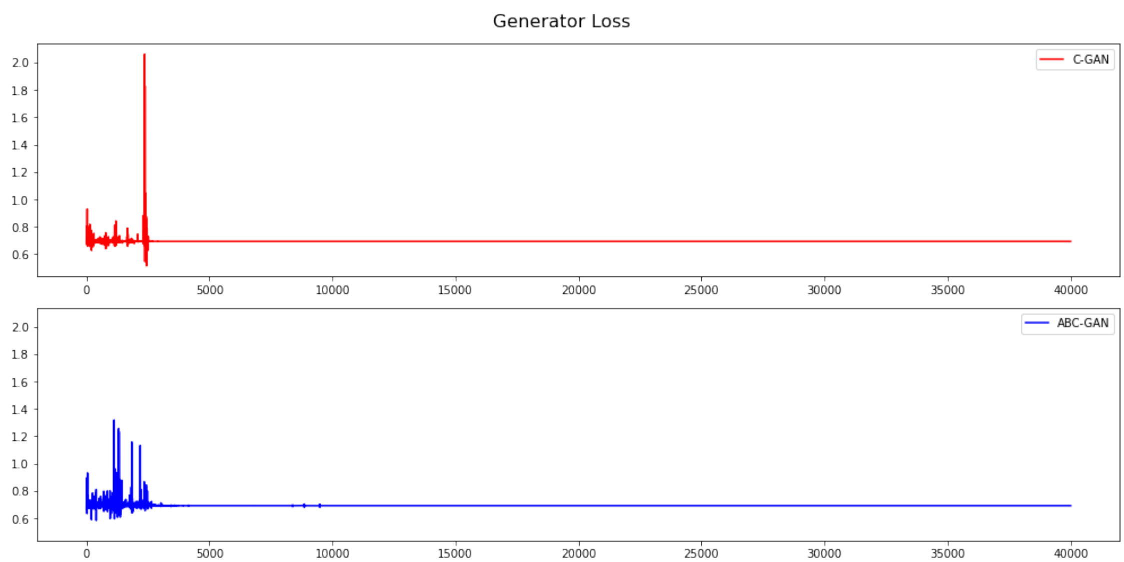
```
GAN Weights
Parameter containing:
tensor([[-0.0815, -0.0404]], requires_grad=True)
ABC-GAN Weights
Parameter containing:
tensor([[-0.0677, -0.1170]], requires_grad=True)
```

[41]:
```
performanceMetrics.modelAnalysis2(GAN3_metrics,ABC_GAN3_metrics)
```



[42]:
```
performanceMetrics.plotTrainingLoss2(discLossG3,genLossG3,discLossA3,genLossA3)
```


Discriminator Loss
C-GAN
ABC-GAN


Generator Loss
C-GAN
ABC-GAN

```
[ ]:
```