Dataset1-Regression_output_7

October 7, 2021

1 Dataset 1 - Regression

1.1 Import Libraries

[1]:

```
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
import warnings
warnings.filterwarnings('ignore')
```

1.2 Parameters

General Parameters

1. Number of Samples

Discriminator Parameters

1. Size : number of hidden nodes

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)

[2]:

```
n_features = 10
sample_size = 100
#Discriminator Parameters
hidden_nodes = 25
#ABC Generator Parameters
mean = 1
```

```
variance = 0.001
```

1.3 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 $\sigma = 0.1$

[3]:

```
X,Y = regressionDataset.regression_data(sample_size,n_features)
```

```
         X1        X2        X3        X4        X5        X6        X7  \
0  0.300193  0.183195 -1.451439 -0.146473 -1.208747  2.192338 -1.463101
1 -1.535950 -0.153135 -0.628463  0.795716  0.445506 -0.486654  1.370390
2 -1.546688  0.140357 -0.179797  1.192841  0.493290  0.336355 -0.295996
3 -2.220979  1.325749 -0.084476 -1.662021 -1.850735  1.427069 -0.636311
4 -1.133980 -1.035369  0.776707 -0.488729 -0.291520 -0.379128  0.981149

         X8        X9       X10           Y
0  0.222585  1.218298 -0.584300  -95.006195
1  0.204180 -0.760542  0.677047   24.941899
2  0.075935 -0.333907 -1.141477  -41.500390
3  1.208603 -1.134527  0.988431 -263.046634
4  1.131512  1.456686 -0.307946   23.030752
```

1.4 Stats Model

[4]:

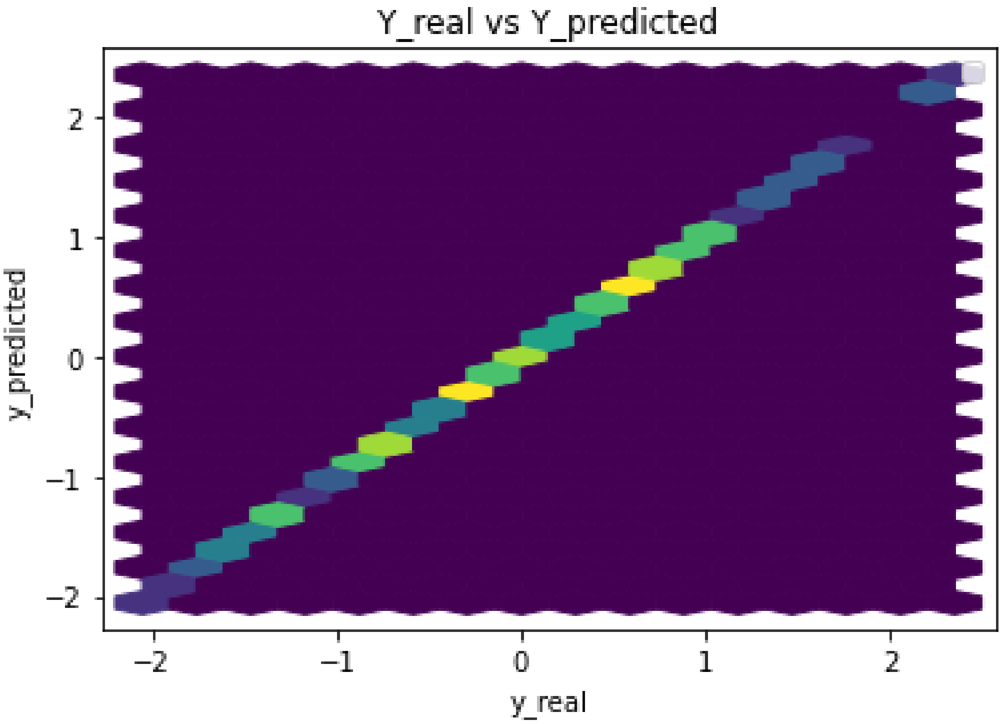
```
[coeff,y_pred] = statsModel.statsModel(X,Y)
```

```
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:                 3.317e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          7.66e-288
Time:                        07:41:05   Log-Likelihood:                 614.66
No. Observations:                 100   AIC:                            -1207.
Df Residuals:                      89   BIC:                            -1179.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.082e-17   5.49e-05   3.79e-13      1.000      -0.000       0.000
x1             0.4581   5.76e-05   7952.048      0.000       0.458       0.458
x2             0.2585    5.6e-05   4617.197      0.000       0.258       0.259
x3             0.2555   5.57e-05   4590.613      0.000       0.255       0.256
x4             0.4653   5.81e-05   8015.009      0.000       0.465       0.465
x5             0.2590   5.61e-05   4614.407      0.000       0.259       0.259
```

```
x6             0.1443   5.58e-05   2583.845      0.000       0.144       0.144
x7             0.4445   5.93e-05   7490.915      0.000       0.444       0.445
x8             0.5108   5.76e-05   8862.768      0.000       0.511       0.511
x9             0.1855   5.72e-05   3240.282      0.000       0.185       0.186
x10            0.1108   5.66e-05   1958.325      0.000       0.111       0.111
==============================================================================
Omnibus:                        1.046   Durbin-Watson:                   2.446
Prob(Omnibus):                  0.593   Jarque-Bera (JB):                0.554
Skew:                          -0.009   Prob(JB):                        0.758
Kurtosis:                       3.364   Cond. No.                         1.59
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    2.081668e-17
x1       4.580975e-01
x2       2.584810e-01
x3       2.554962e-01
x4       4.653325e-01
x5       2.590165e-01
x6       1.442808e-01
x7       4.445177e-01
x8       5.107817e-01
x9       1.854787e-01
x10      1.108222e-01
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 2.683376589256899e-07
Mean Absolute Error: 0.00039111815141975426
Manhattan distance: 0.039111815141975426
Euclidean distance: 0.005180131841234253
```

2 Generator and Discriminator Networks

GAN Generator

[5]:

```python
class Generator(nn.Module):

  def __init__(self,n_input):
    super().__init__()
    self.output = nn.Linear(n_input,1)

  def forward(self, x):
    x = self.output(x)
    return x
```

GAN Discriminator

[6]:

```python
class Discriminator(nn.Module):
  def __init__(self,n_input,n_hidden):

    super().__init__()
    self.hidden = nn.Linear(n_input,n_hidden)
    self.output = nn.Linear(n_hidden,1)
    self.relu = nn.ReLU()

  def forward(self, x):
    x = self.hidden(x)
    x = self.relu(x)
    x = self.output(x)
    return x
```

ABC Generator

The ABC generator is defined as follows:

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

$\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else

$\beta_i \sim N(\beta_i^*, \sigma^*)$ where $\beta_i^* s$ are coefficients obtained from stats model

Parameters : μ and σ^*

σ^* takes the values 0.01,0.1 and 1

[7]:

```
def ABC_pre_generator(x_batch,coeff,variance,mean,device):

  coeff_len = len(coeff)

  if mean == 0:
    weights = np.random.normal(0,variance,size=(coeff_len,1))
    weights = torch.from_numpy(weights).reshape(coeff_len,1)
  else:
    weights = []
    for i in range(coeff_len):
      weights.append(np.random.normal(coeff[i],variance))
    weights = torch.tensor(weights).reshape(coeff_len,1)

  y_abc =  torch.matmul(x_batch,weights.float())
  gen_input = torch.cat((x_batch,y_abc),dim = 1).to(device)
  return gen_input
```

3 GAN Model

[8]:

```
real_dataset = dataset.CustomDataset(X,Y)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[9]:

```
generator = Generator(n_features+2)
discriminator = Discriminator(n_features+2,hidden_nodes)

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(
  (output): Linear(in_features=12, out_features=1, bias=True)
)
Discriminator(
  (hidden): Linear(in_features=12, out_features=25, bias=True)
  (output): Linear(in_features=25, out_features=1, bias=True)
  (relu): ReLU()
)
```

[11]:

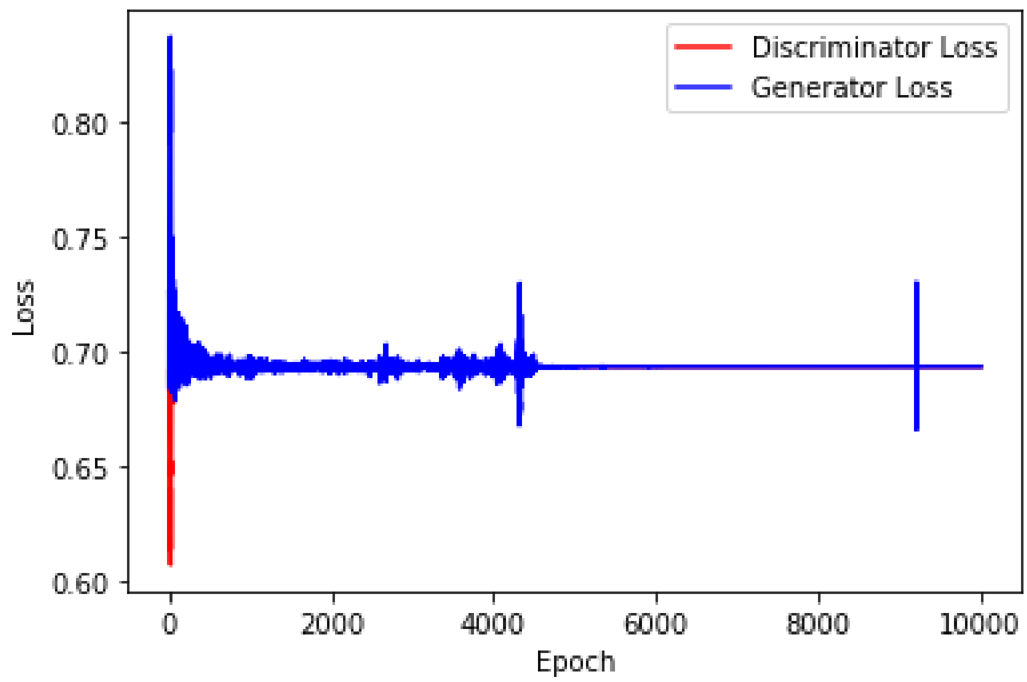
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:

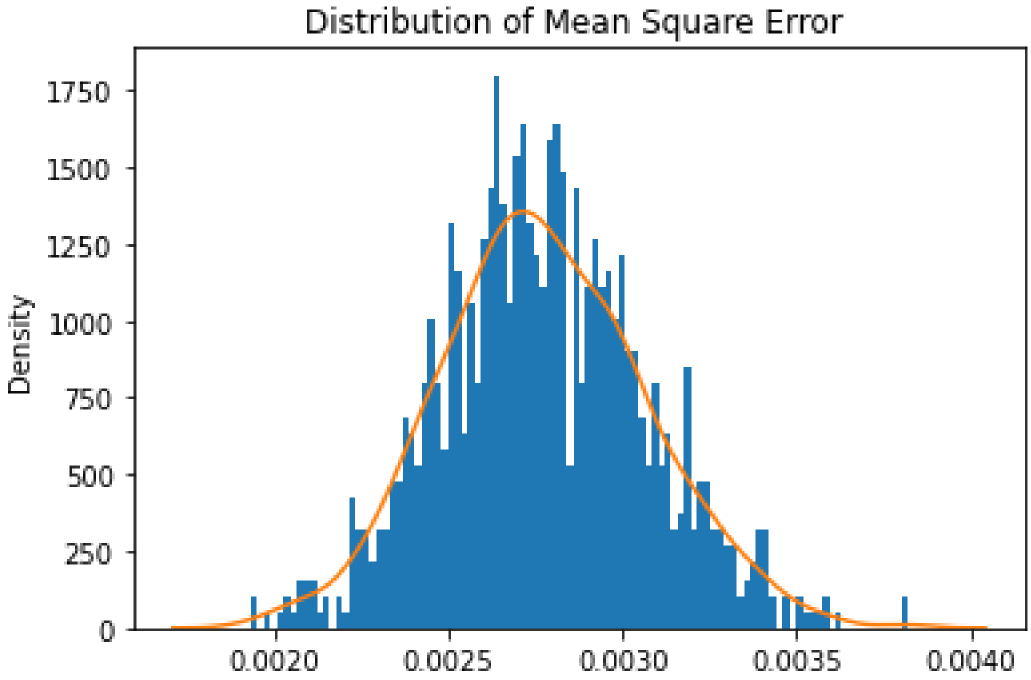
```
# Parameters
sample_size = 100
std = 1
mean = 0.01
```

[13]:

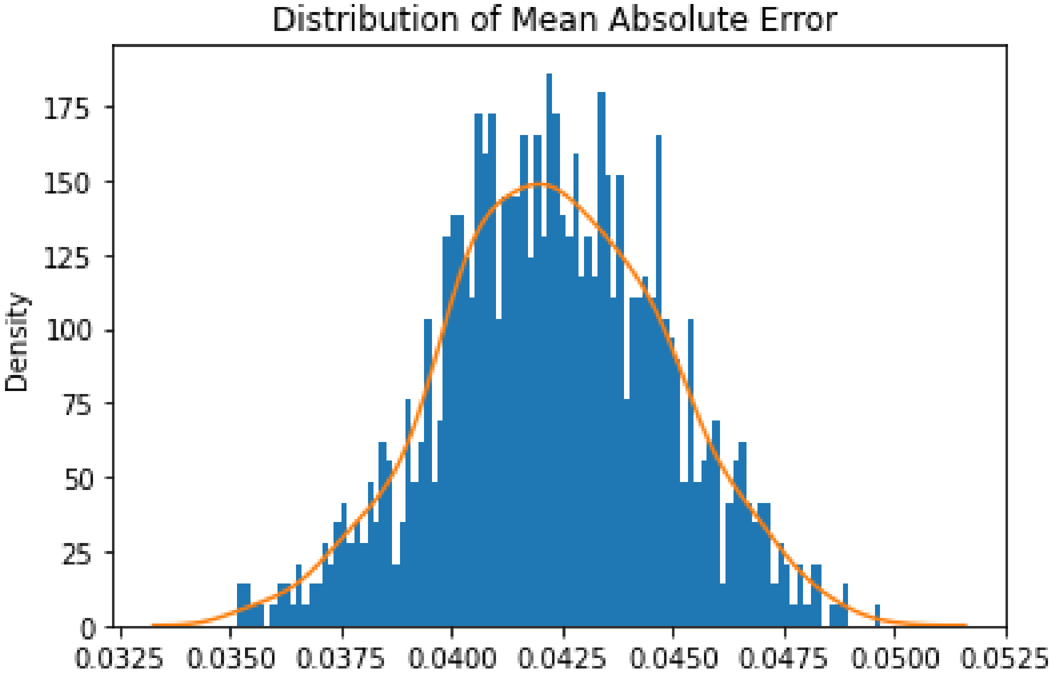
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



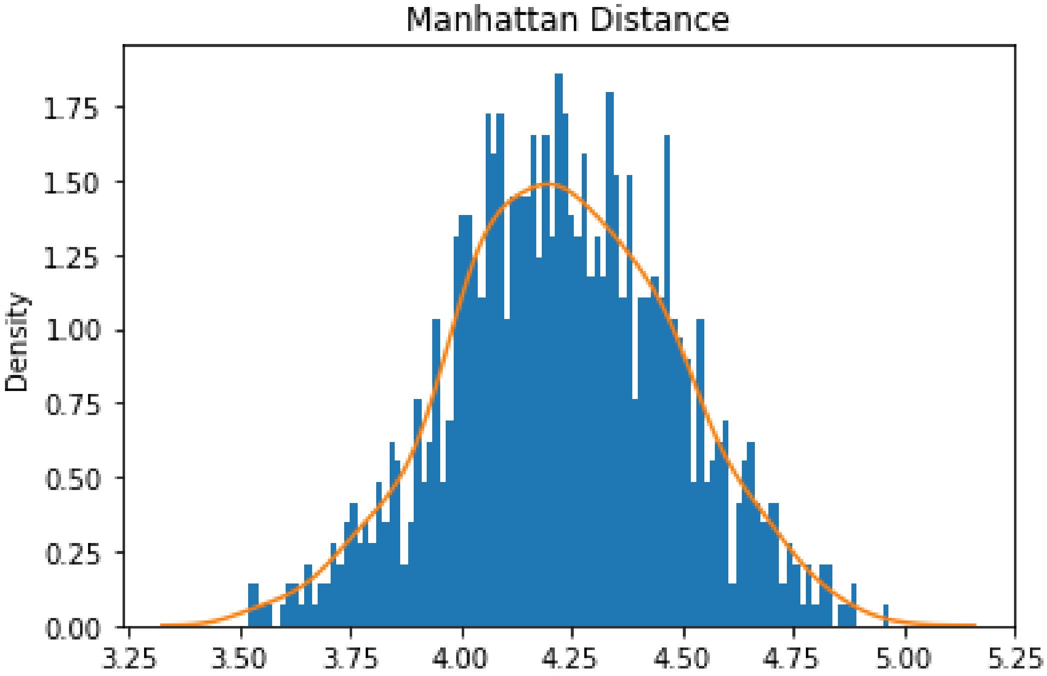
[14]:
```
train_test.test_generator(generator,real_dataset,device)
```



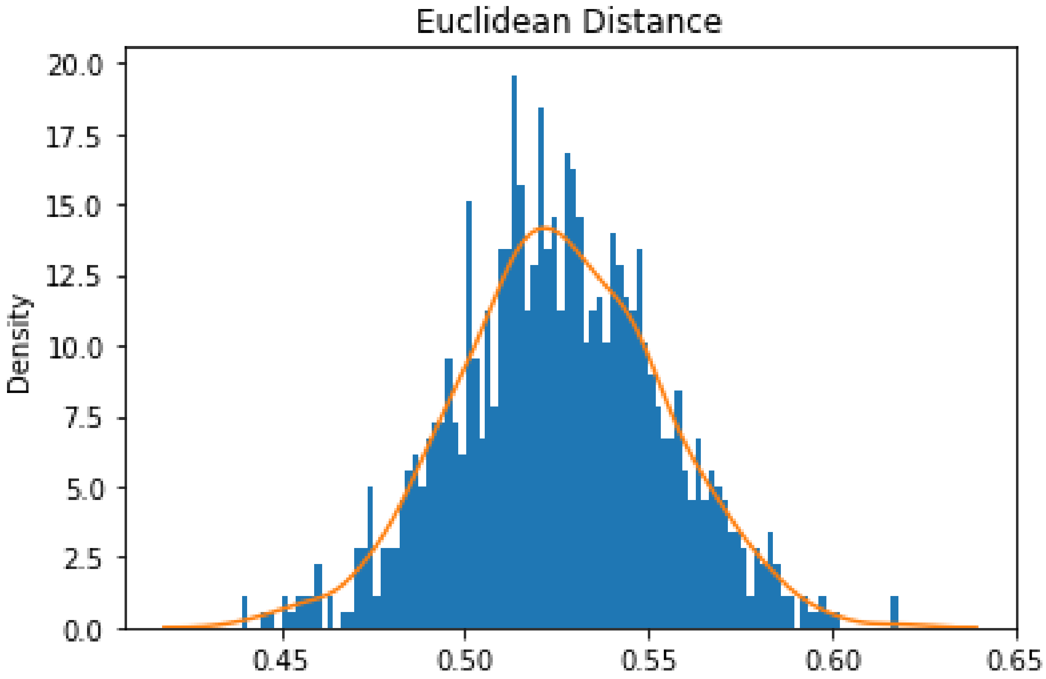
```
Mean Square Error: 0.0027712130148640312
```



```
Mean Absolute Error: 0.04231884976159781
```



```
Mean Manhattan Distance: 4.231884976159781
```



```
Mean Euclidean Distance: 4.231884976159781
```

4 ABC GAN Model

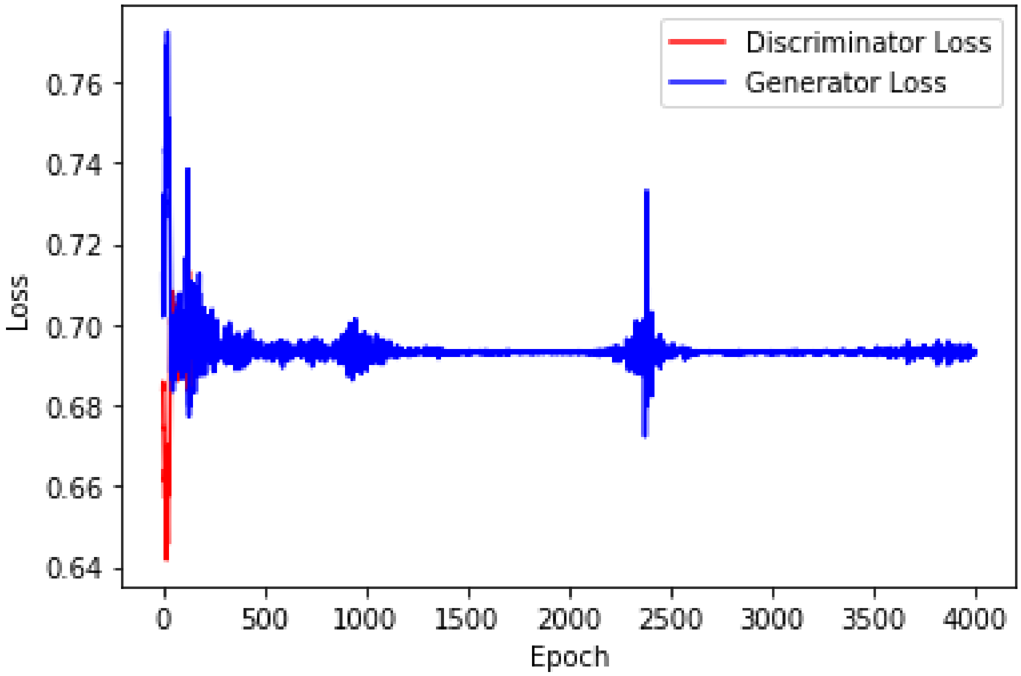
Training the network

[15]:
```python
gen = Generator(n_features+2)
disc = Discriminator(n_features+2,hidden_nodes)

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))
```

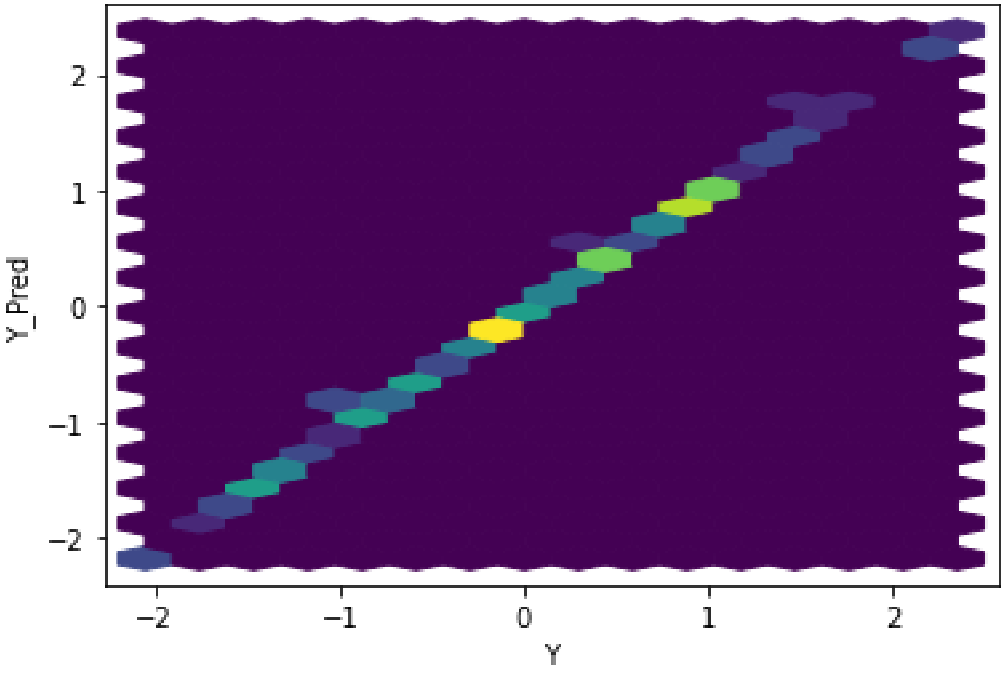
[16]:
```python
n_epoch_abc = 2000
batch_size = sample_size//2
```

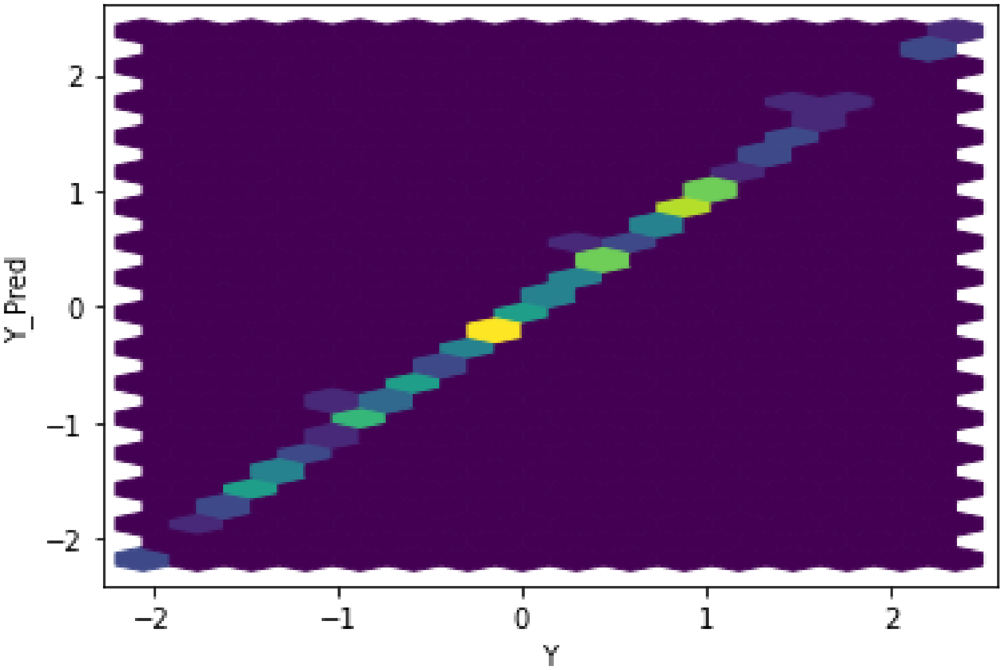
[17]:
```python
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset, batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

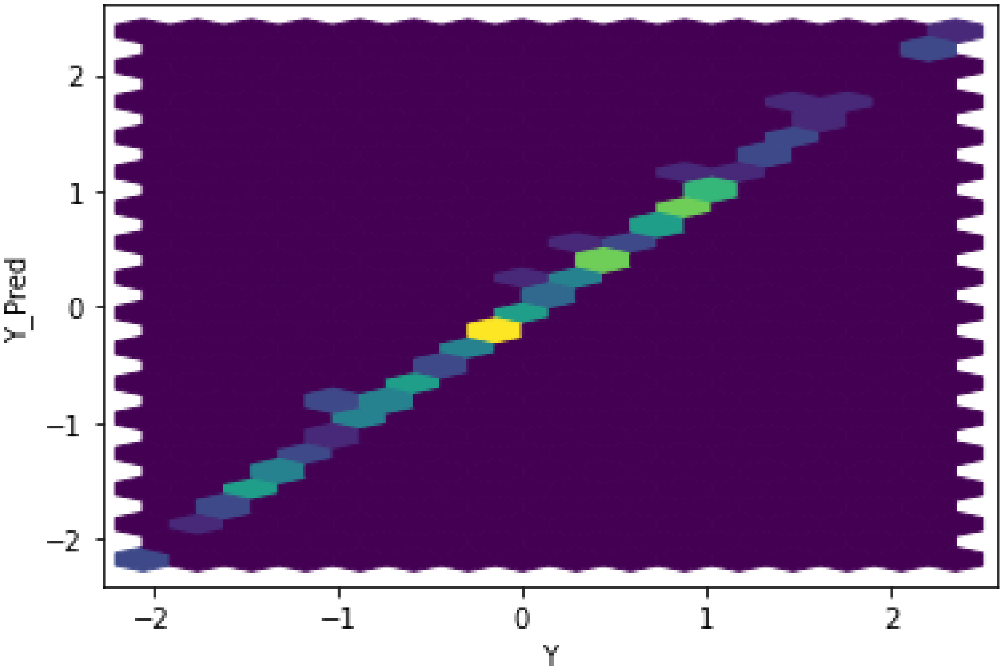


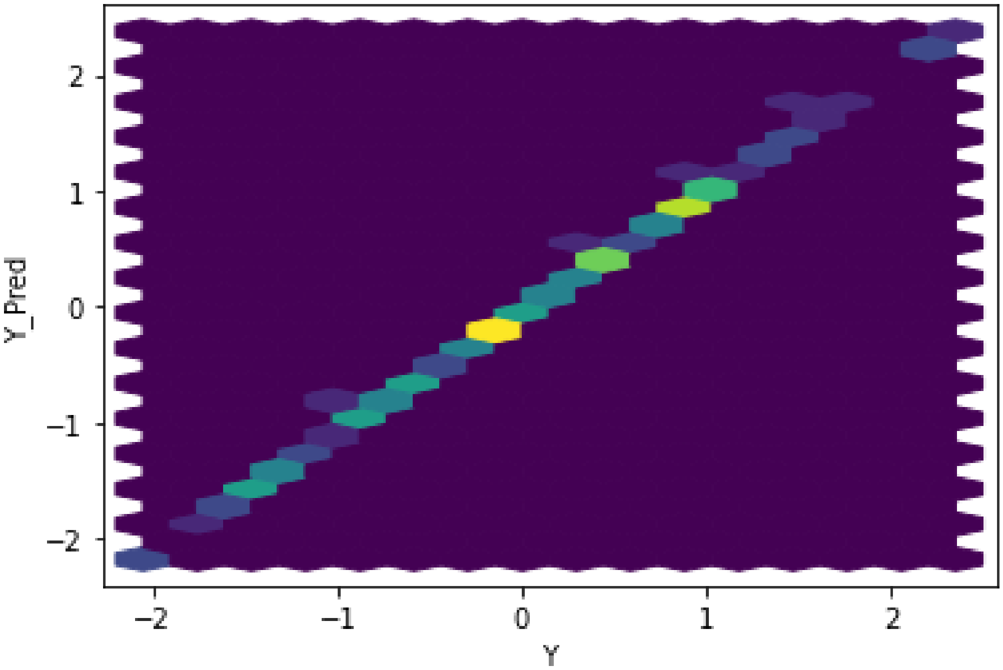
[18]:

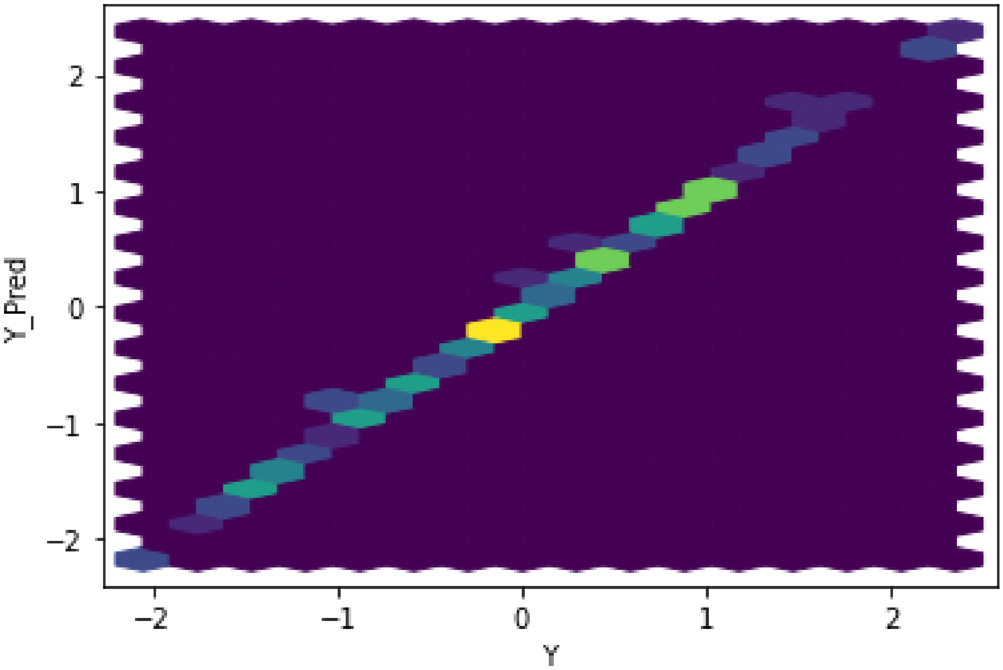
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

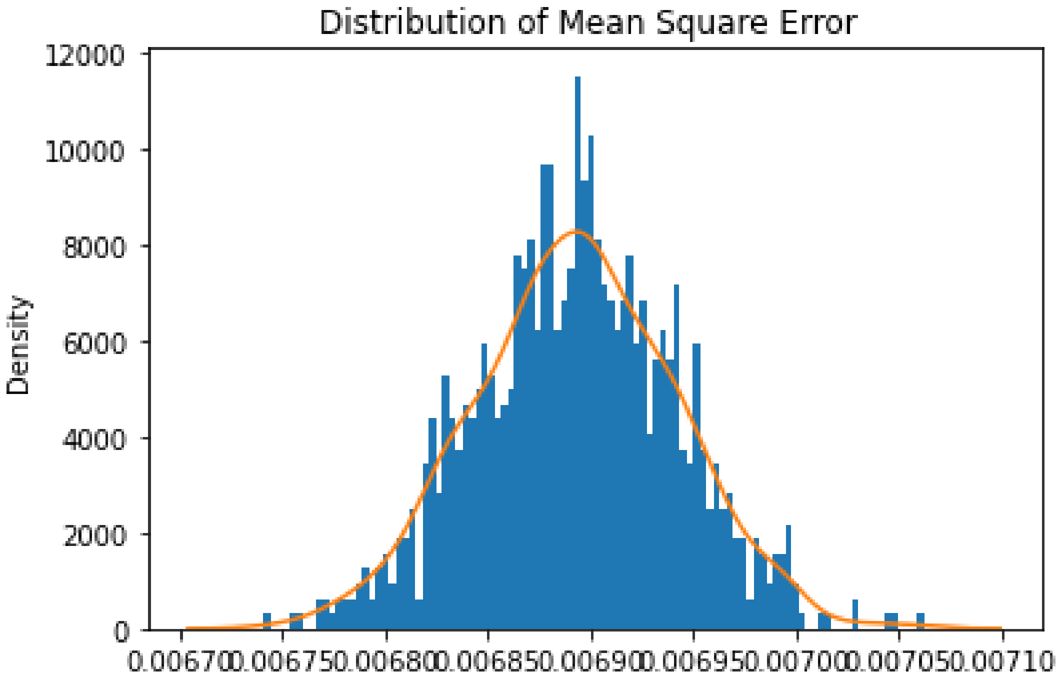



Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y

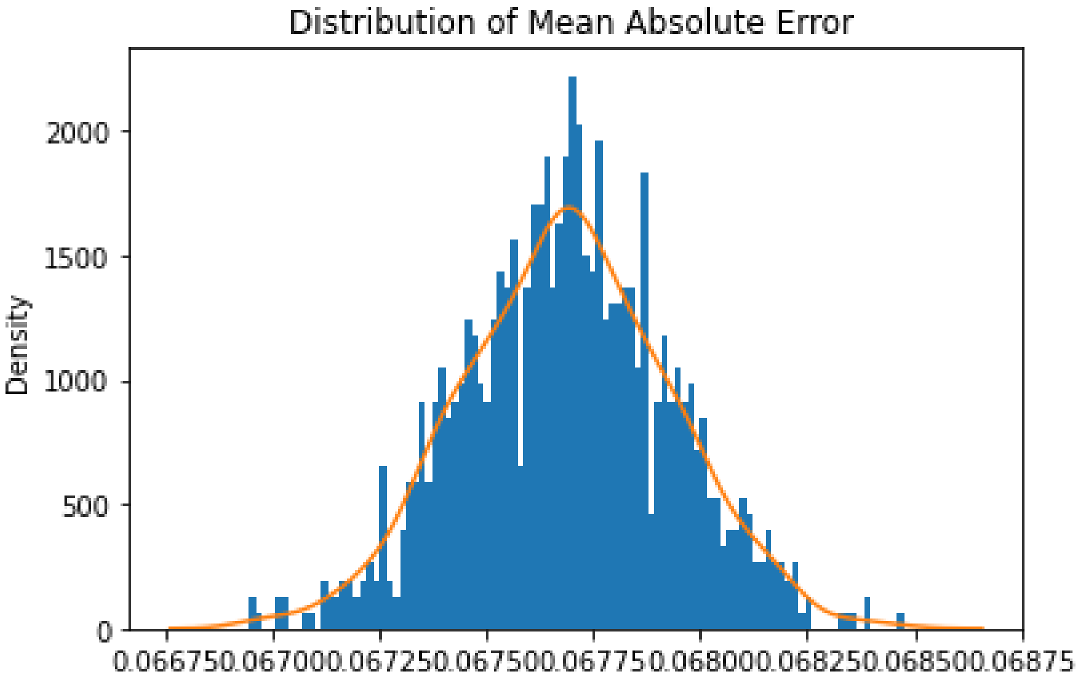

Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y


Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y

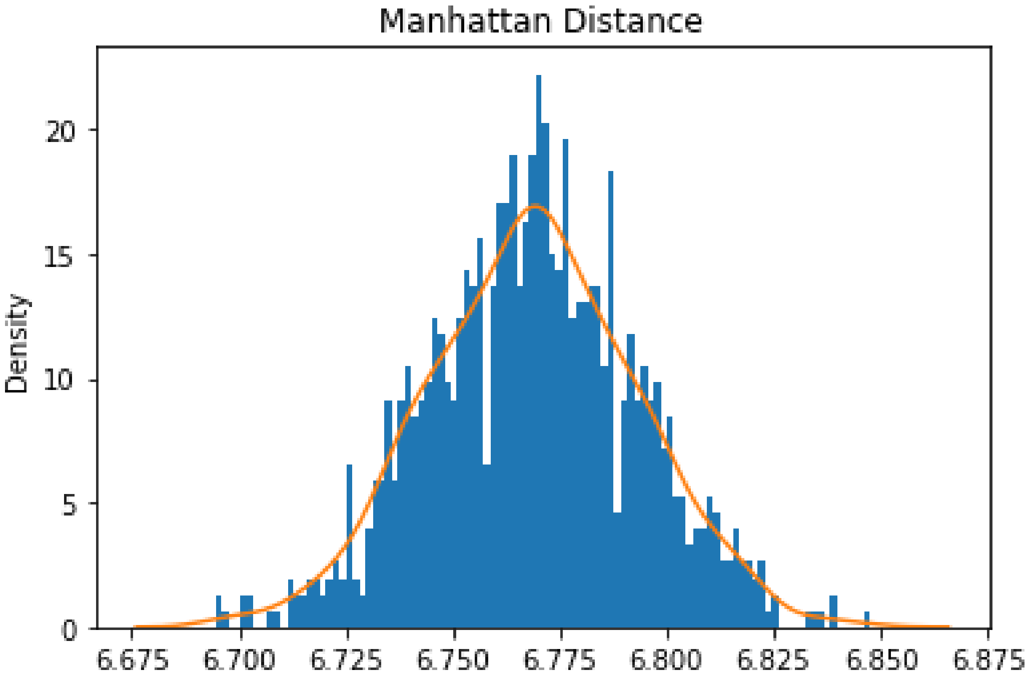

Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y



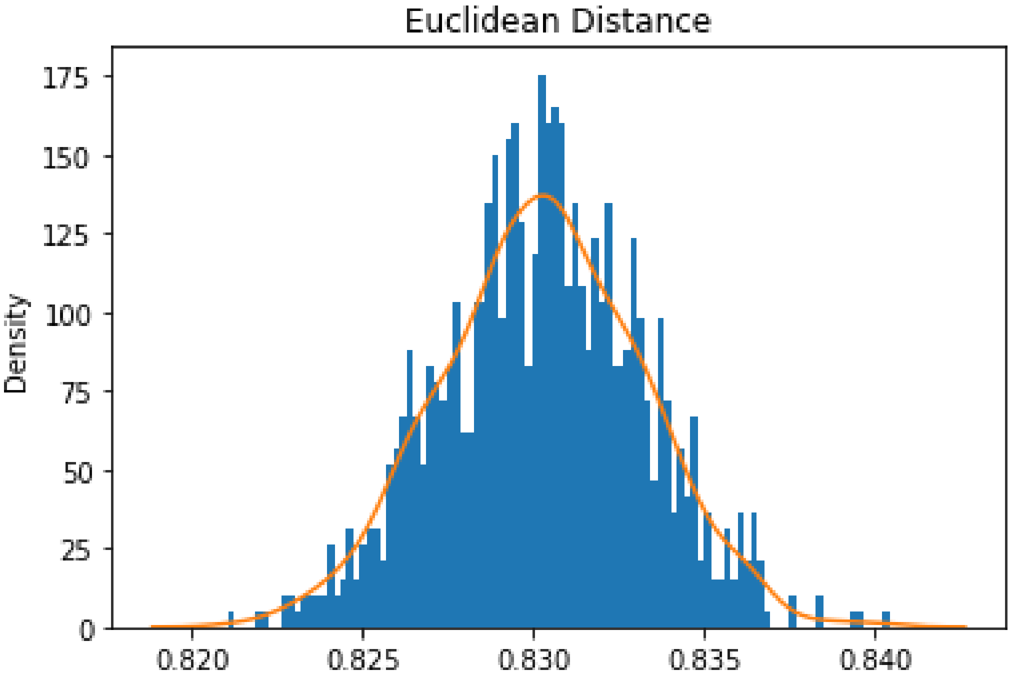
```
Mean Square Error: 0.006893201593445457
```



```
Mean Absolute Error: 0.06768699396684766
Mean Manhattan Distance: 6.768699396684766
```

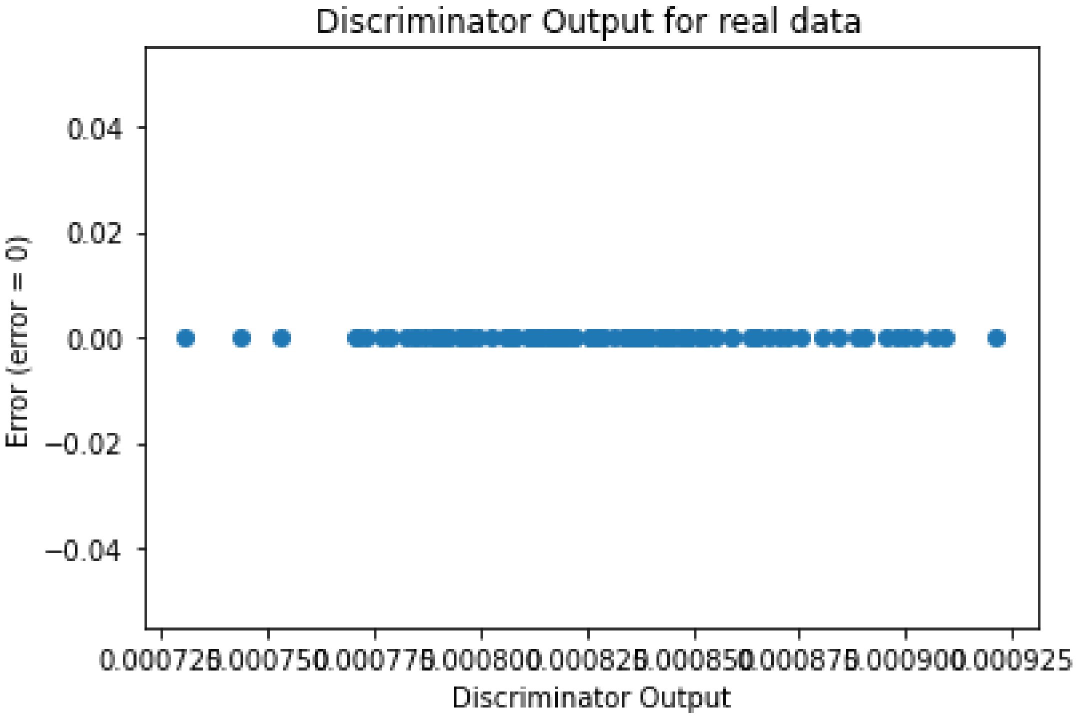


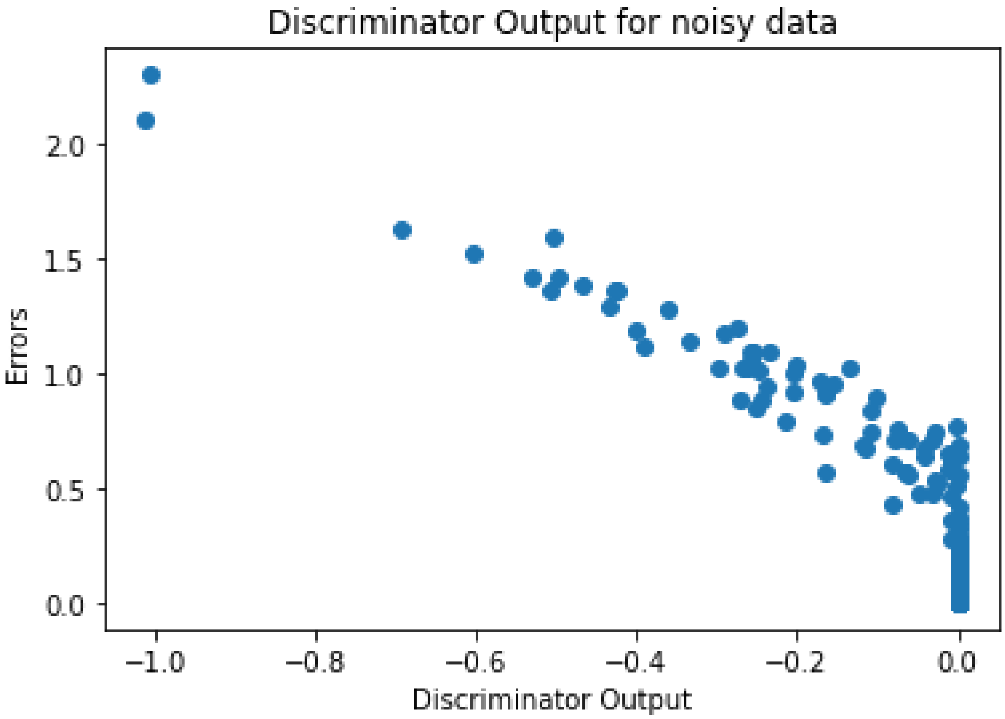
```
Mean Euclidean Distance: 0.8302478773152039
```



Sanity Checks

[19]:
```
sanityChecks.discProbVsError(real_dataset,disc,device)
```





4.1 Visualization of trained GAN generator

[20]:

```
for name, param in gen.named_parameters():
    print(name,param)
```

```
output.weight Parameter containing:
tensor([[-0.1209,  0.3050,  0.2030,  0.2402,  0.3336,  0.1884,  0.1306,  0.3340,
          0.3791,  0.1447,  0.0556,  0.2992]], requires_grad=True)
output.bias Parameter containing:
tensor([0.1003], requires_grad=True)
```