Dataset1-Regression_output_1

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  0.828190 -0.671401 -0.114409  0.321253  1.968355  0.415183  0.252660
1 -1.461843 -0.417667 -1.062421 -1.318094 -0.320996 -0.555302  0.973471
2  2.733327  0.582073 -0.420996 -0.201740 -1.217099 -0.953351 -0.641691
3 -2.464848  1.333694 -0.509637 -0.635727 -0.431631 -0.716983 -0.228871
4  0.624705  0.113061  0.687955 -0.020135  0.764722  1.286240 -0.305725

         X8        X9       X10           Y
0  1.300335 -0.167704  0.505798  225.217558
1  1.128970 -0.692240  0.030363 -124.349798
2 -1.486295  1.214144 -0.423282 -206.541175
3  0.189057 -0.421181  0.634204 -129.969607
4 -1.261808 -1.325728 -0.886753  -80.358712
```

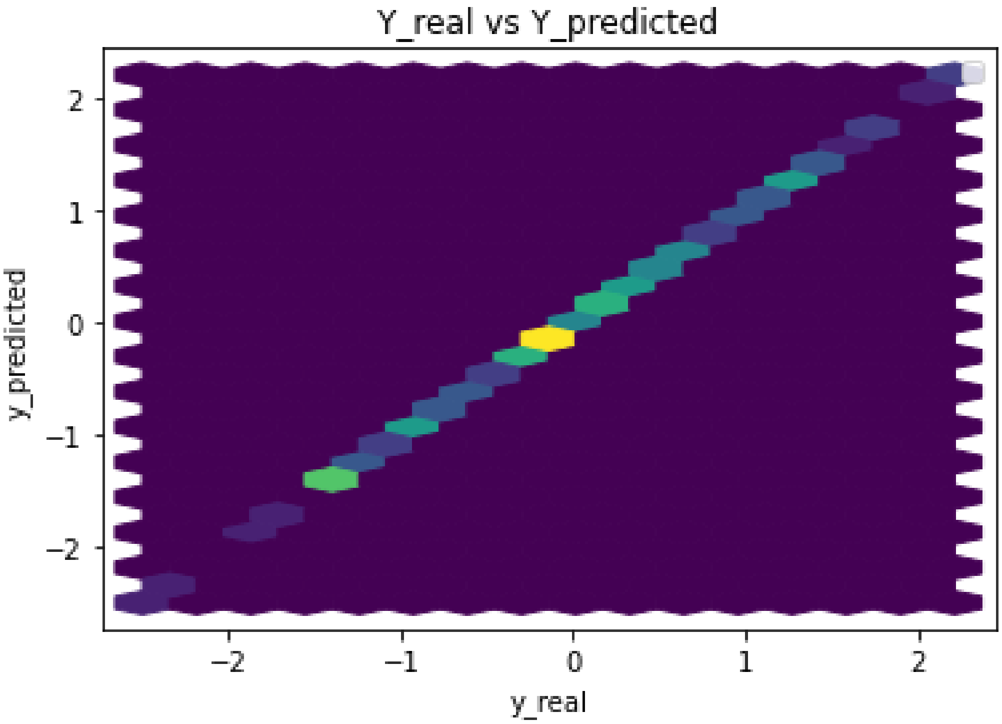
1.4 Stats Model

[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:                 2.703e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          6.86e-284
Time:                        18:50:15   Log-Likelihood:                 604.43
No. Observations:                 100   AIC:                            -1187.
Df Residuals:                      89   BIC:                            -1158.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -7.112e-17   6.08e-05  -1.17e-12      1.000      -0.000       0.000
x1             0.1223    6.3e-05   1940.987      0.000       0.122       0.122
x2             0.0350    6.4e-05    547.191      0.000       0.035       0.035
x3             0.6125   6.26e-05   9779.236      0.000       0.612       0.613
x4             0.3648   7.05e-05   5177.736      0.000       0.365       0.365
x5             0.1298   6.42e-05   2022.552      0.000       0.130       0.130
```

```
x6             0.2739   6.17e-05   4435.181      0.000       0.274       0.274
x7             0.3441   6.68e-05   5151.762      0.000       0.344       0.344
x8             0.5014   6.55e-05   7652.695      0.000       0.501       0.501
x9             0.2350   6.41e-05   3664.770      0.000       0.235       0.235
x10            0.4628   6.29e-05   7361.984      0.000       0.463       0.463
==============================================================================
Omnibus:                        0.276   Durbin-Watson:                   2.084
Prob(Omnibus):                  0.871   Jarque-Bera (JB):                0.453
Skew:                           0.049   Prob(JB):                        0.797
Kurtosis:                       2.685   Cond. No.                         1.93
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const   -7.112366e-17
x1       1.222886e-01
x2       3.501763e-02
x3       6.124679e-01
x4       3.648052e-01
x5       1.298098e-01
x6       2.738586e-01
x7       3.440973e-01
x8       5.013637e-01
x9       2.350319e-01
x10      4.628414e-01
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 3.2922662964844305e-07
Mean Absolute Error: 0.0004597128640476543
Manhattan distance: 0.04597128640476544
Euclidean distance: 0.005737827373217524
```

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]:

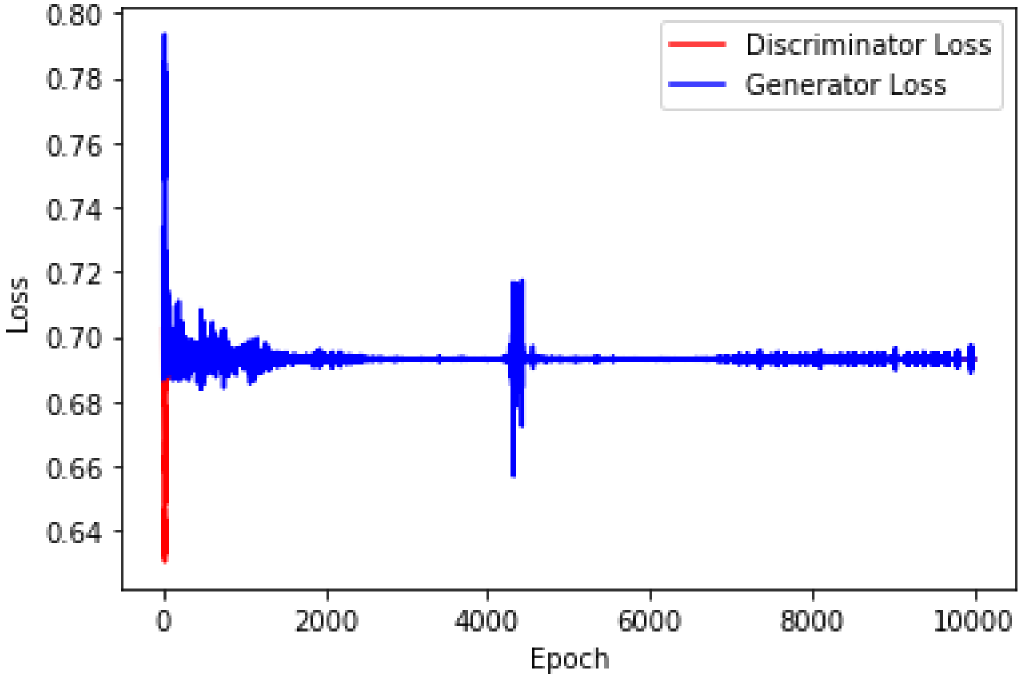
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:

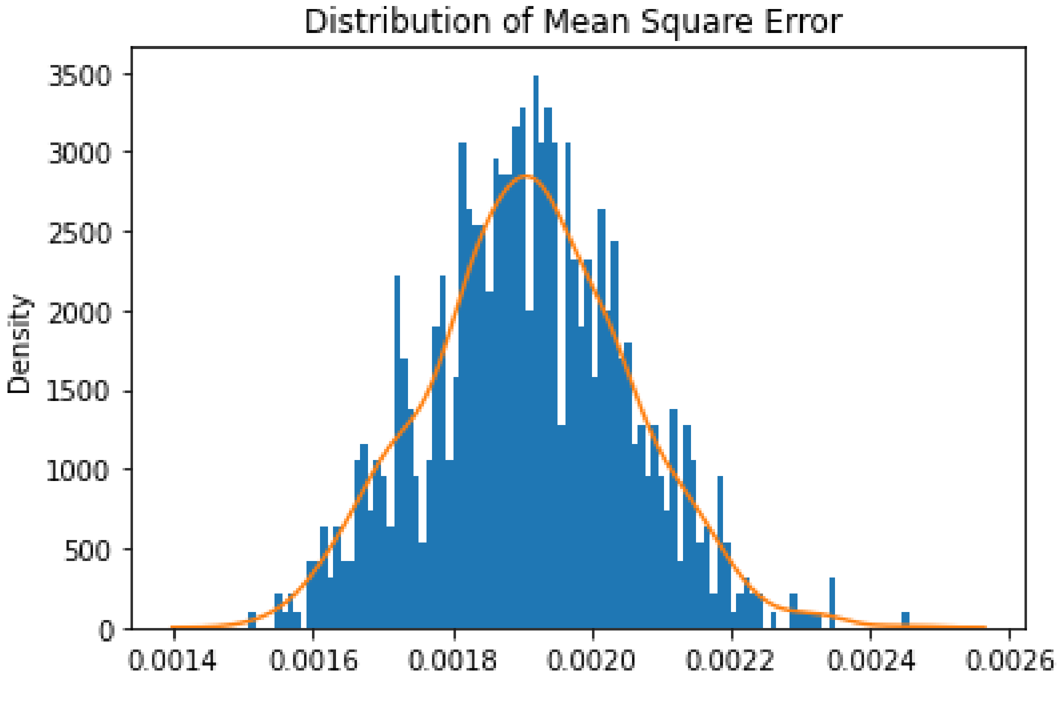
```
# Parameters
sample_size = 100
mean = 1
std = 1
```

[13]:

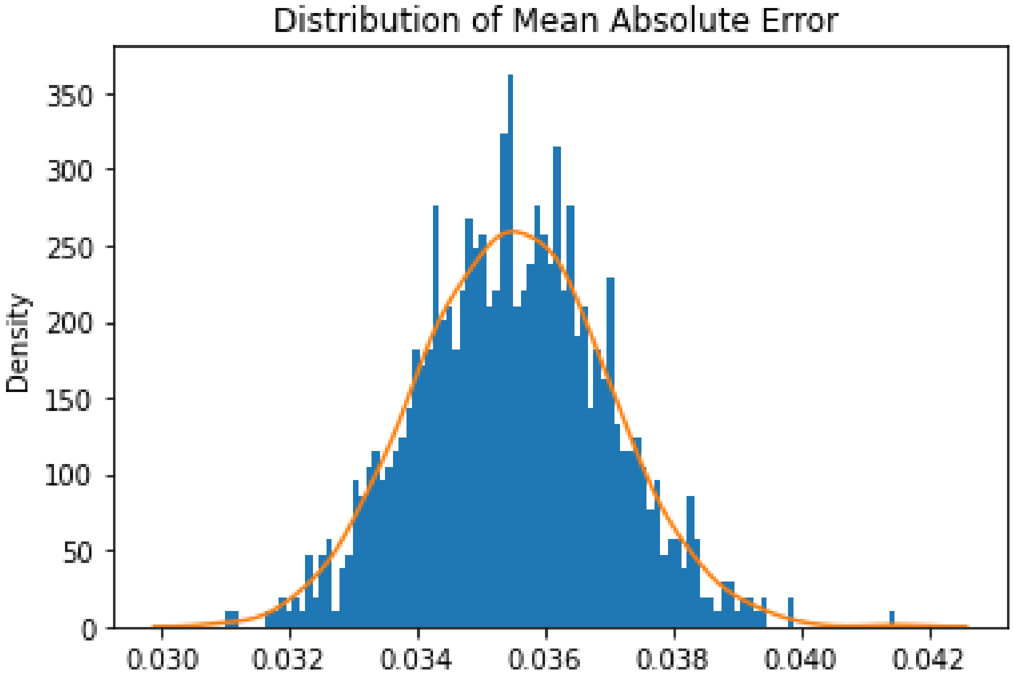
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



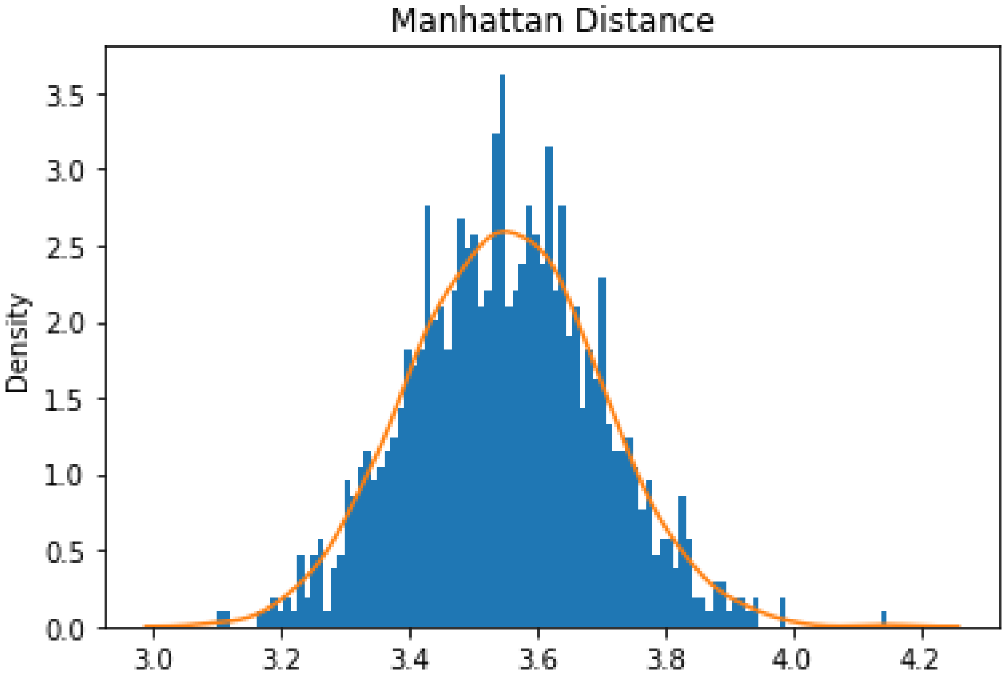
```
[14]: train_test.test_generator(generator,real_dataset,device)
```



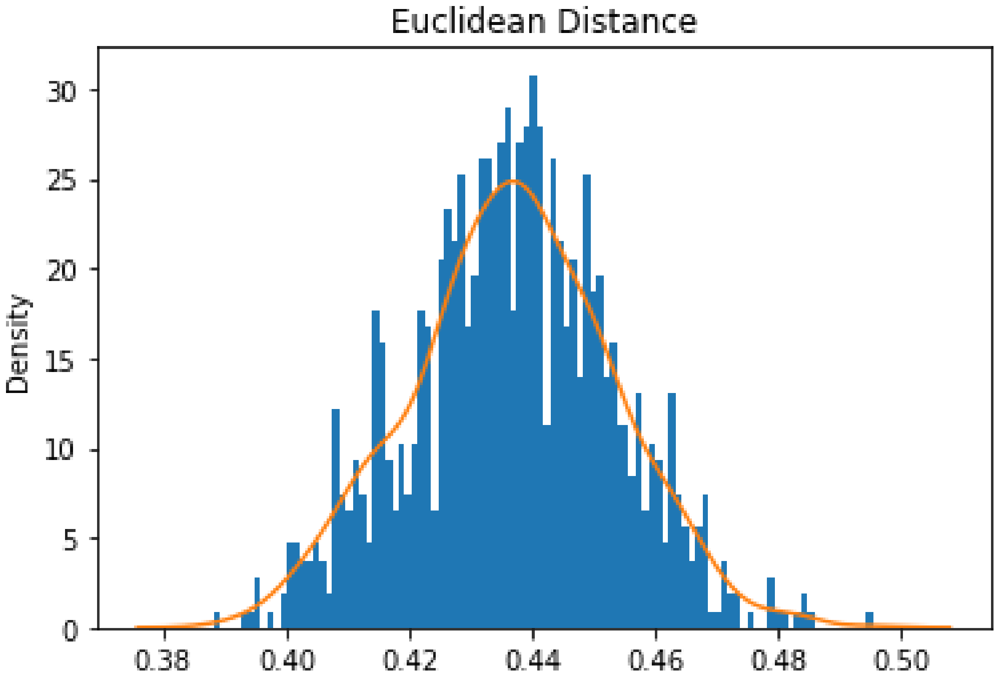
```
Mean Square Error: 0.0019104108167693284
```



```
Mean Absolute Error: 0.03551003866981715
```



```
Mean Manhattan Distance: 3.551003866981715
```



```
Mean Euclidean Distance: 3.551003866981715
```

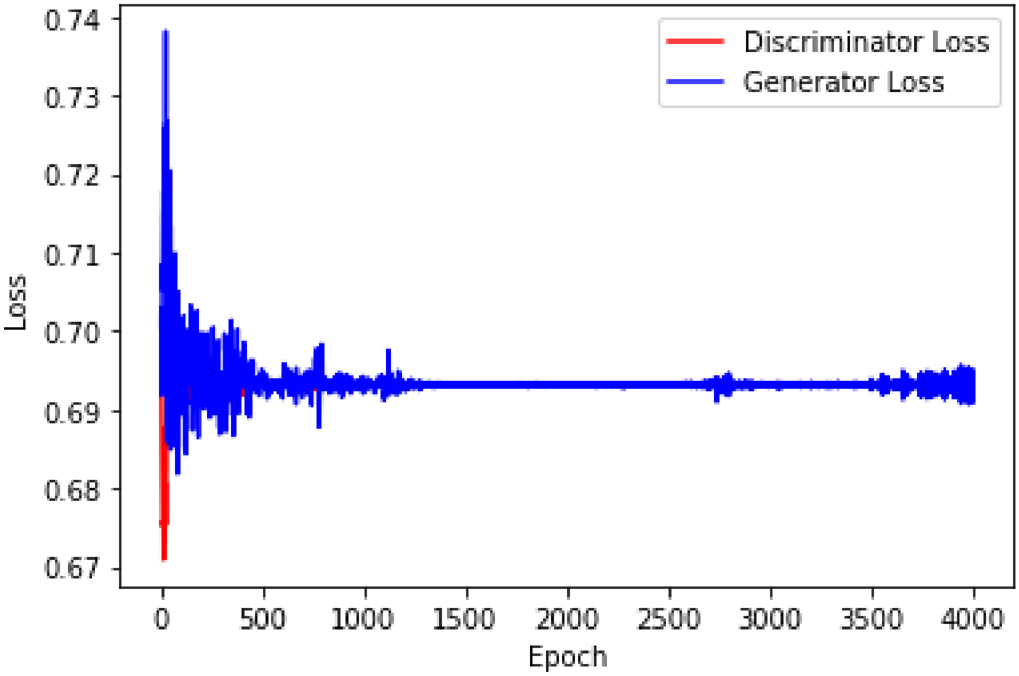
4 ABC GAN Model

Training the network

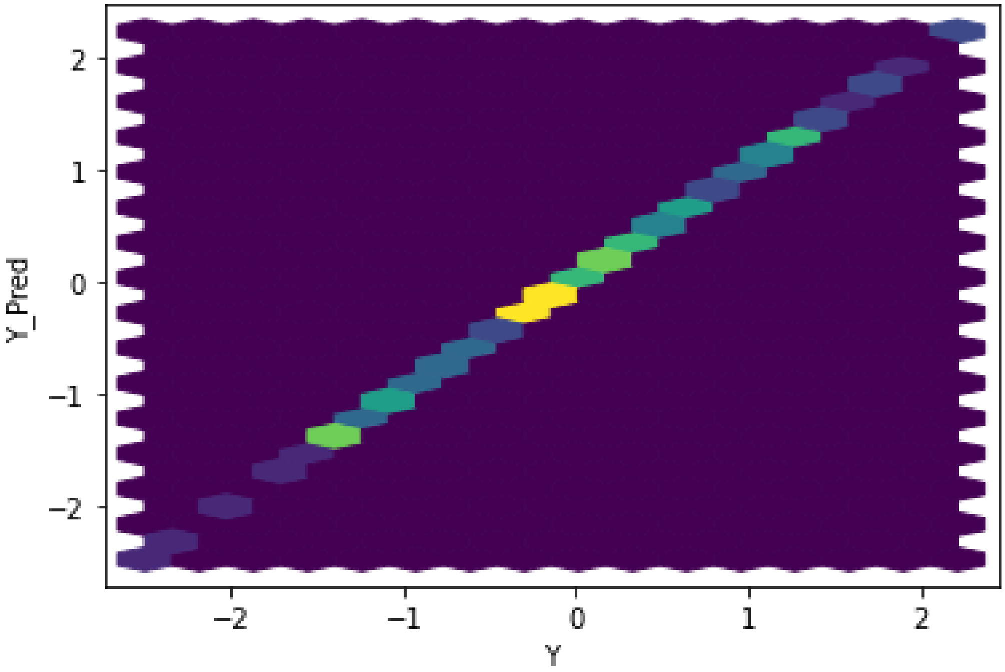
[15]:
```
gen = Generator(n_features+2)
disc = Discriminator(n_features+2,hidden_nodes)

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))
```

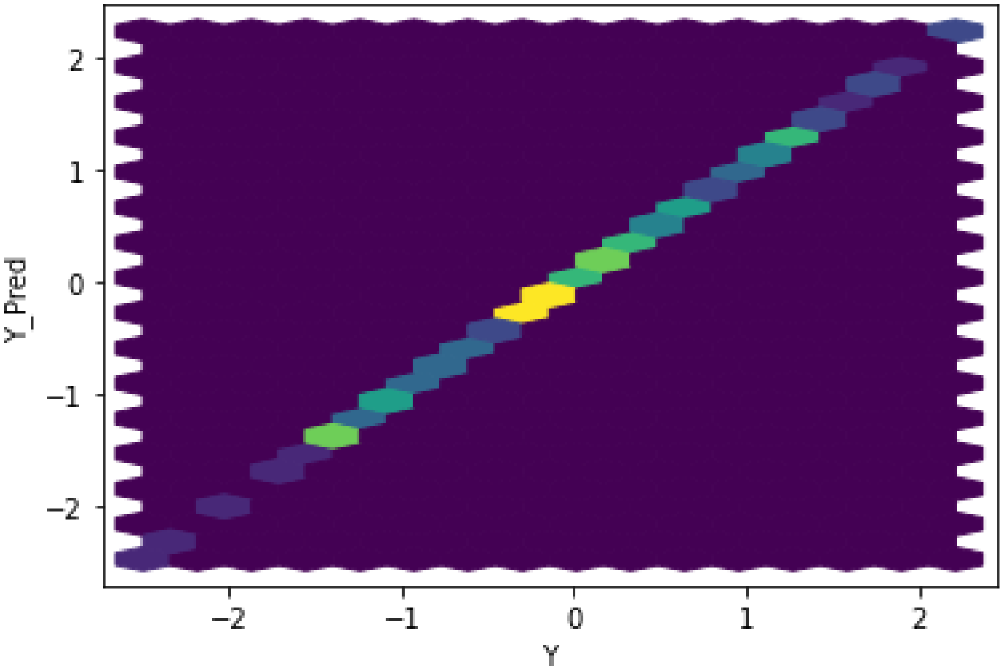
[16]:
```
n_epoch_abc = 2000
batch_size = sample_size//2
```

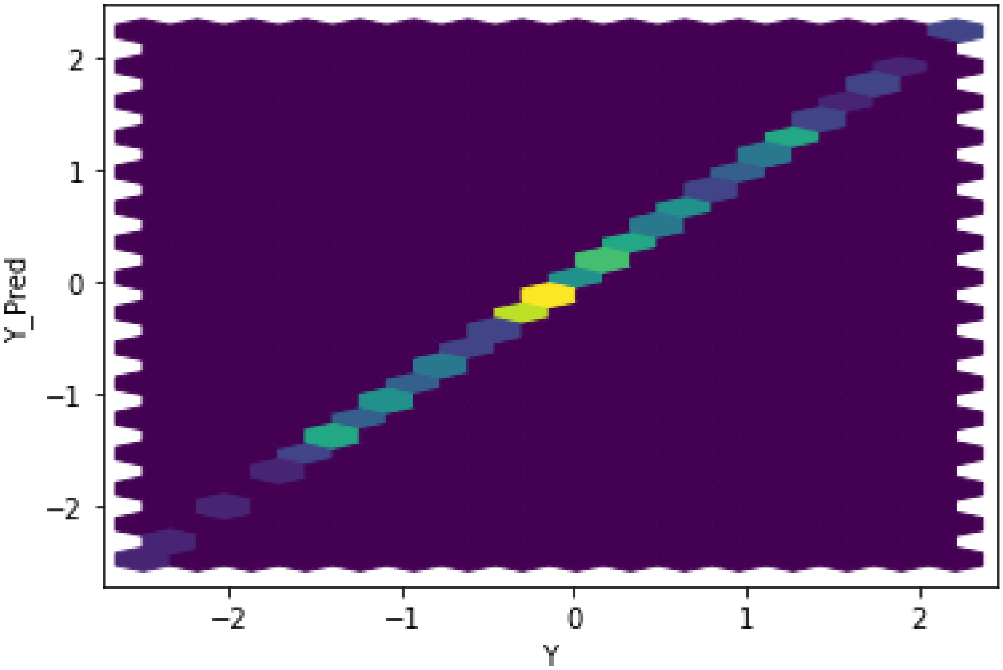
[17]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
 ↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

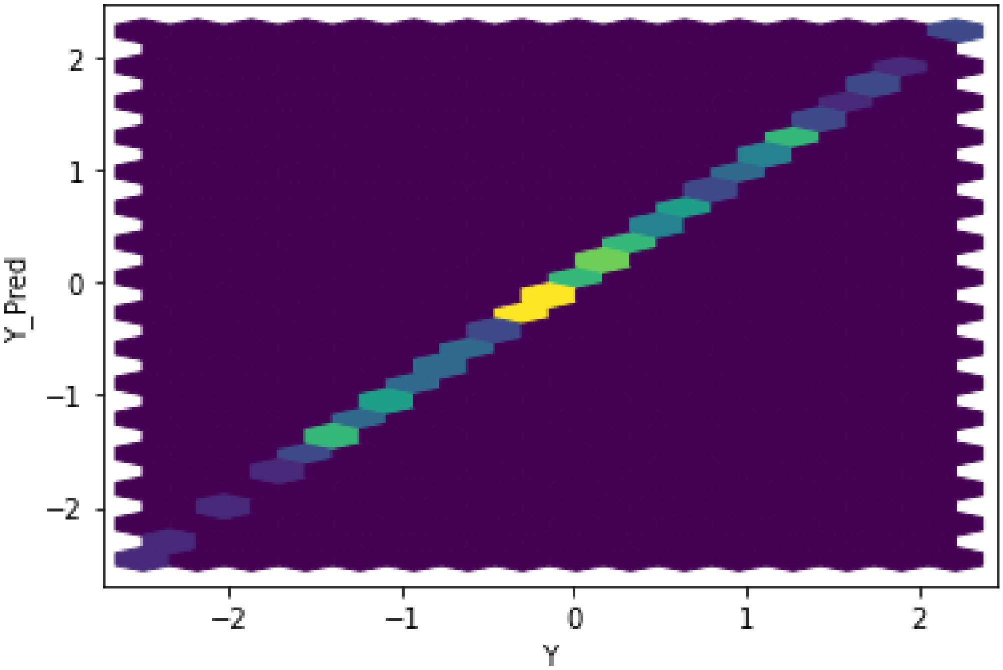


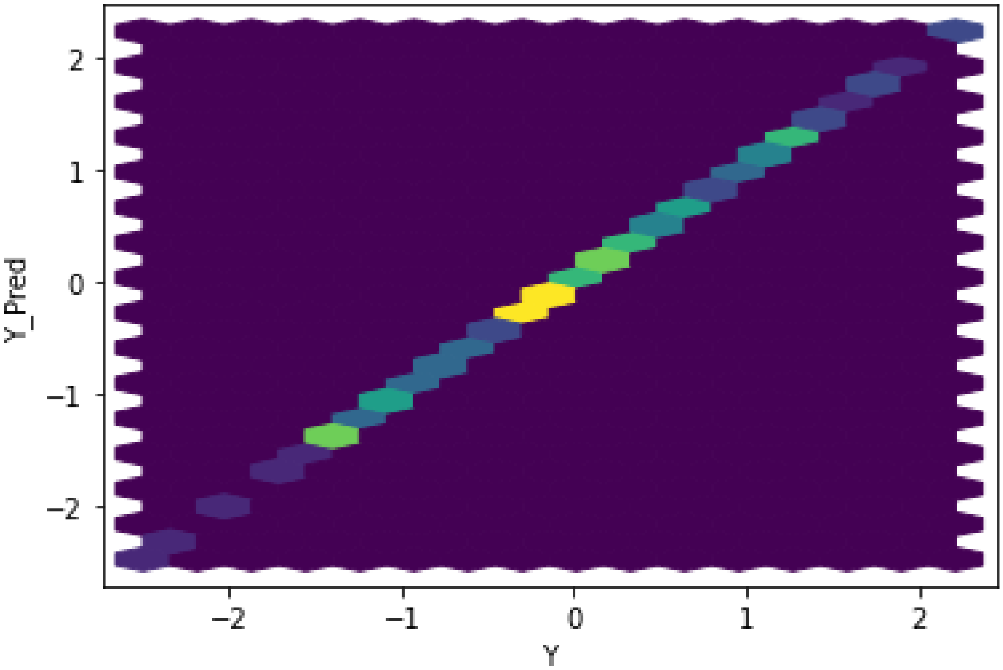
[18]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

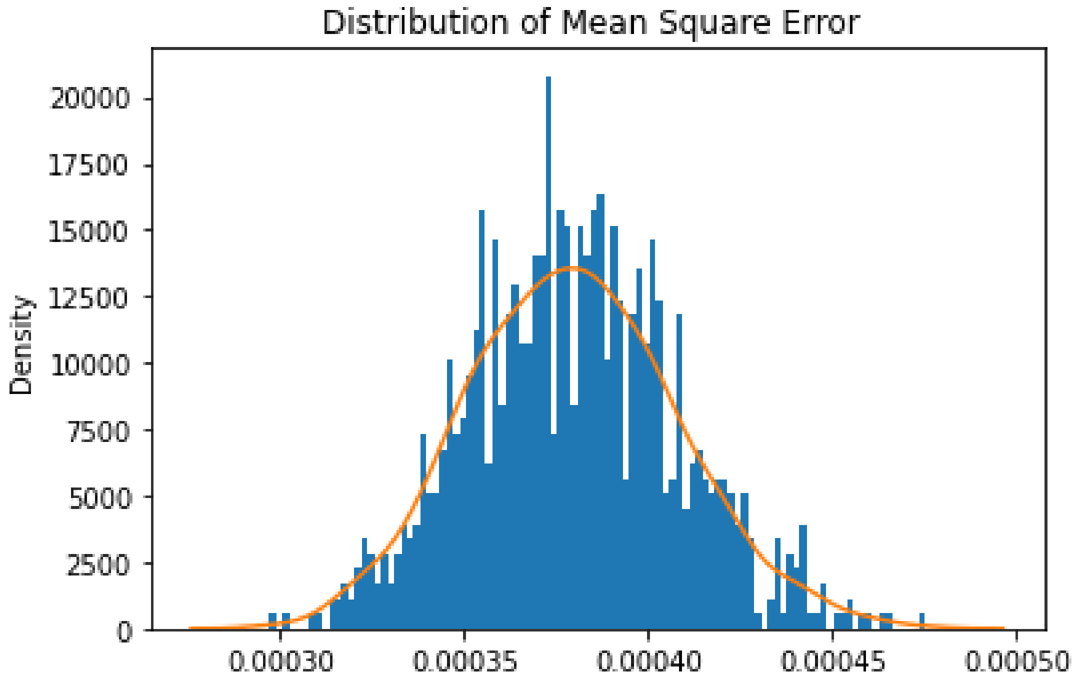



Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y

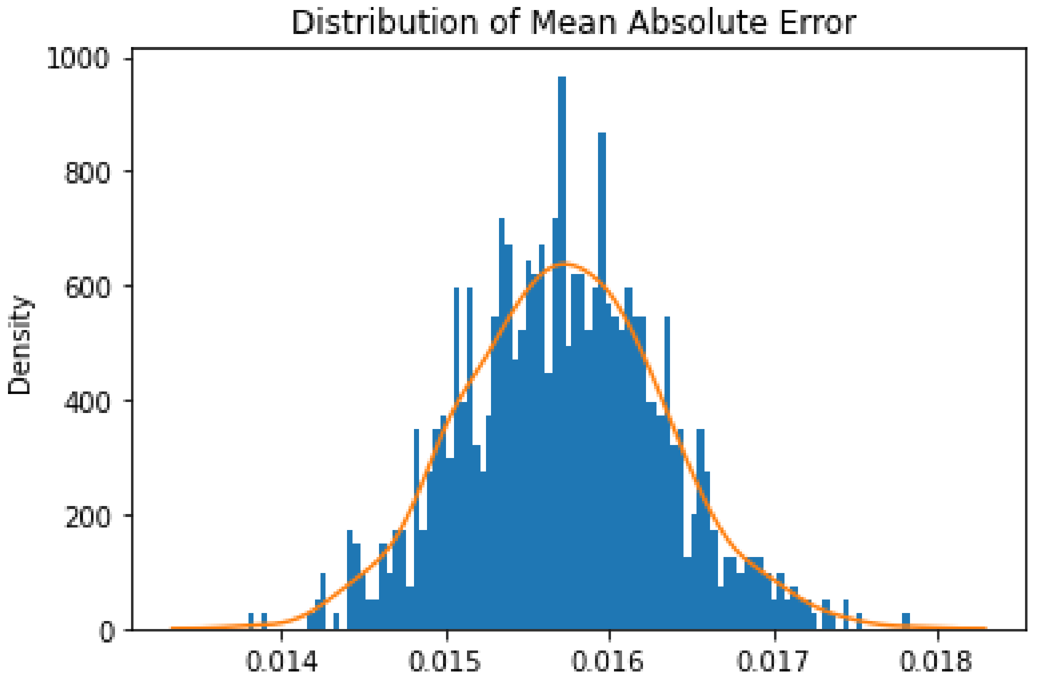

Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y


2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
Y

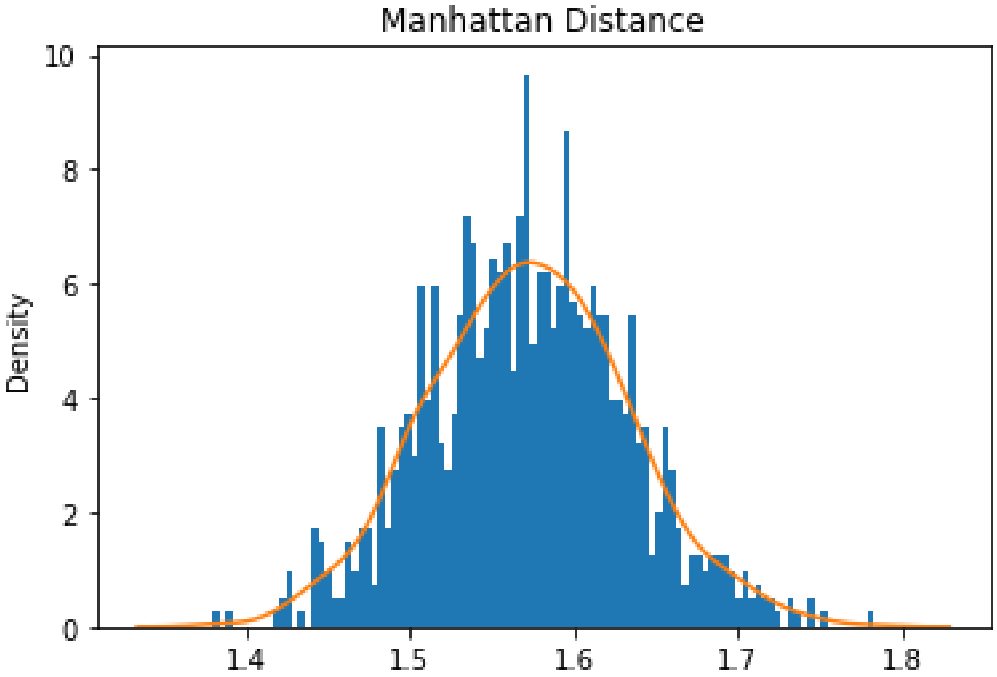

2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
Y



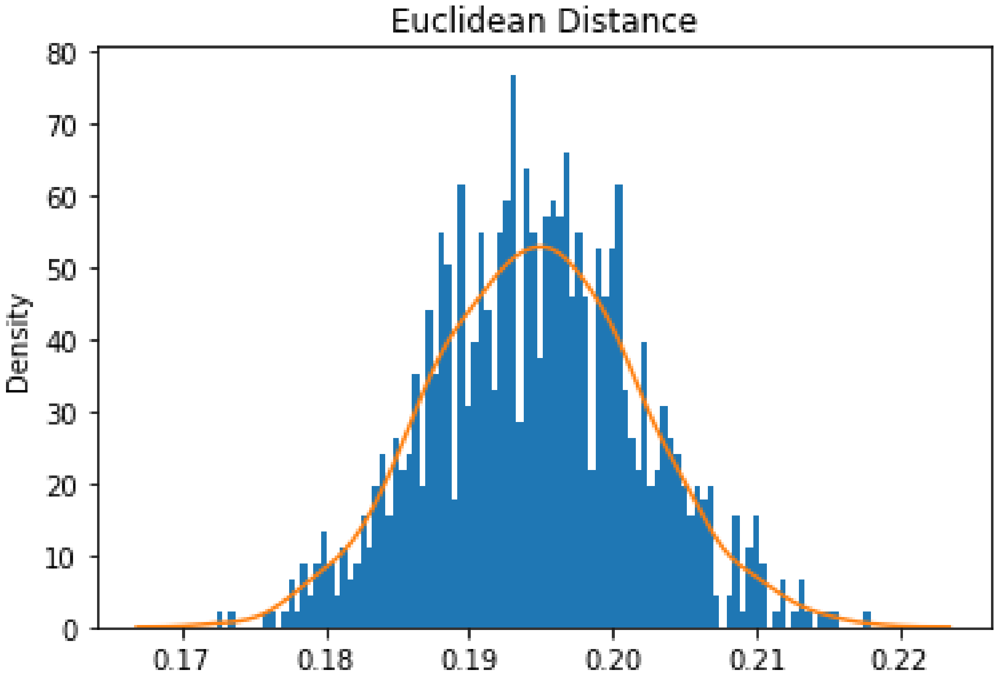
```
Mean Square Error: 0.00037939619624443335
```



```
Mean Absolute Error: 0.015724081892967225
Mean Manhattan Distance: 1.5724081892967223
```

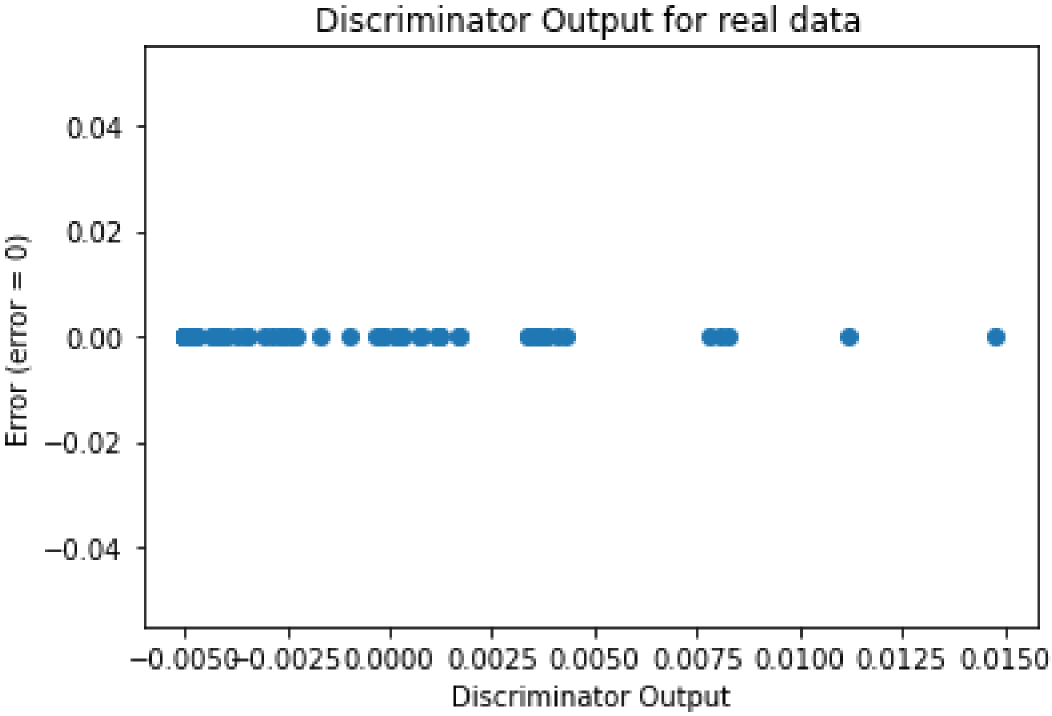


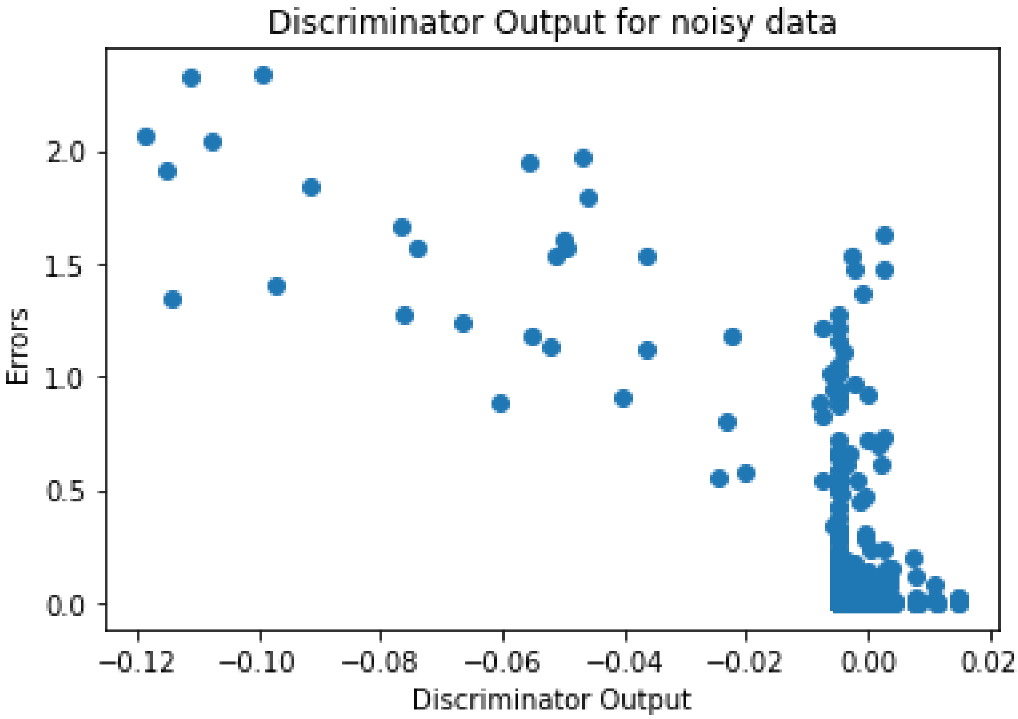
```
Mean Euclidean Distance: 0.19464564635511697
```



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([[-0.1462,  0.0331,  0.0069,  0.1857,  0.1207,  0.0345,  0.0945,  0.1032,
          0.1476,  0.0773,  0.1445,  0.6886]], requires_grad=True)
output.bias Parameter containing:
tensor([0.1520], requires_grad=True)
```