Dataset1-Regression_output_12

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)
```

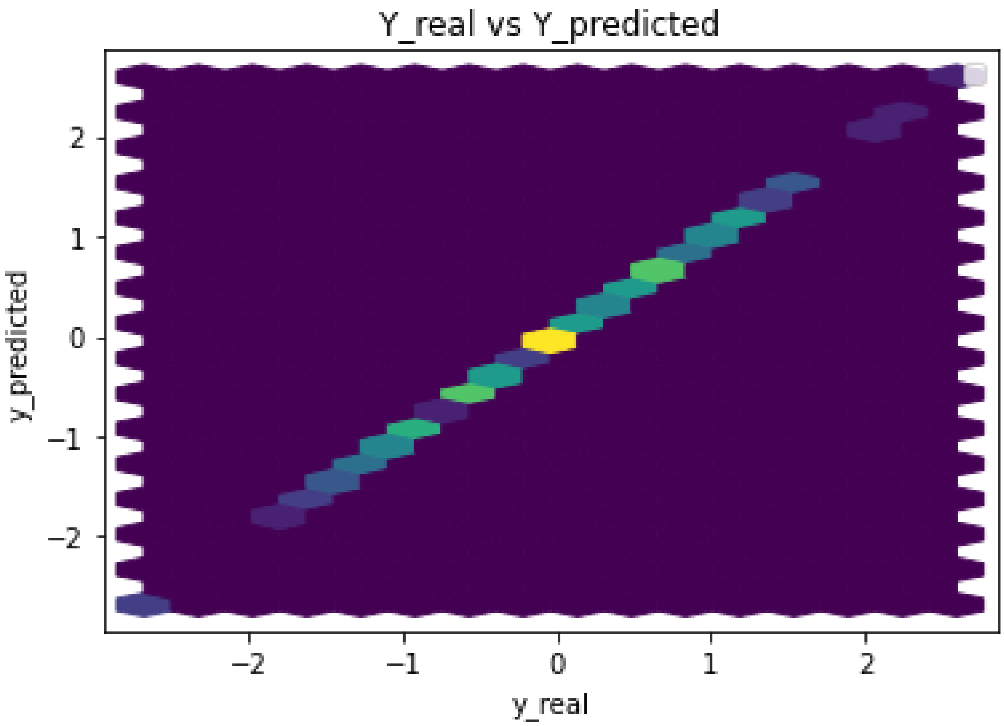
```
         X1        X2        X3        X4        X5        X6        X7  \
0  0.511652  1.124947 -0.712013  1.836362 -1.476262 -0.303459  0.753767
1  1.362499 -1.414153  0.663315  0.125101  0.025895 -2.591533  0.046855
2  1.619812  1.626317 -0.069758 -0.658972  0.616078  0.878996  0.122417
3 -0.267271  0.921697  0.045051  0.397481  0.454955  0.249371  0.516169
4 -0.300797 -0.382979 -1.379667  0.737230  1.067886  1.357061 -0.896643

         X8        X9       X10           Y
0 -0.769144 -1.022808  0.865857  -52.133705
1 -0.487969  0.249752  1.681512 -235.087084
2  0.355046  1.310133 -0.799510  336.082251
3  1.537083 -0.519017 -2.170549  138.728748
4 -0.114048 -0.540825  1.851047  248.993577
```

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:                 4.435e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          1.86e-293
Time:                        07:44:49   Log-Likelihood:                 629.18
No. Observations:                 100   AIC:                            -1236.
Df Residuals:                      89   BIC:                            -1208.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       6.939e-18   4.75e-05   1.46e-13      1.000   -9.44e-05    9.44e-05
x1             0.0368    4.8e-05    768.431      0.000       0.037       0.037
x2             0.3138    4.9e-05   6400.171      0.000       0.314       0.314
x3             0.0890   5.29e-05   1681.964      0.000       0.089       0.089
x4             0.3601   4.81e-05   7480.951      0.000       0.360       0.360
x5             0.4418   5.15e-05   8584.836      0.000       0.442       0.442
```

```
x6             0.3995      5e-05    7985.676      0.000       0.399       0.400
x7             0.1333   5.06e-05    2637.423      0.000       0.133       0.133
x8             0.4058   4.88e-05    8310.221      0.000       0.406       0.406
x9             0.4882   5.13e-05    9517.766      0.000       0.488       0.488
x10            0.2976   4.89e-05    6084.736      0.000       0.298       0.298
==============================================================================
Omnibus:                        2.228   Durbin-Watson:                   2.002
Prob(Omnibus):                  0.328   Jarque-Bera (JB):                1.883
Skew:                          -0.004   Prob(JB):                        0.390
Kurtosis:                       3.672   Cond. No.                         1.69
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    6.938894e-18
x1       3.684833e-02
x2       3.138090e-01
x3       8.900745e-02
x4       3.601077e-01
x5       4.417722e-01
x6       3.995456e-01
x7       1.333239e-01
x8       4.058453e-01
x9       4.882306e-01
x10      2.976259e-01
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 2.006856112358927e-07
Mean Absolute Error: 0.0003527331224100709
Manhattan distance: 0.03527331224100709
Euclidean distance: 0.004479794763556615
```

2 Generator and Discriminator Networks

GAN Generator

[5]:
```
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]:
```
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))
```

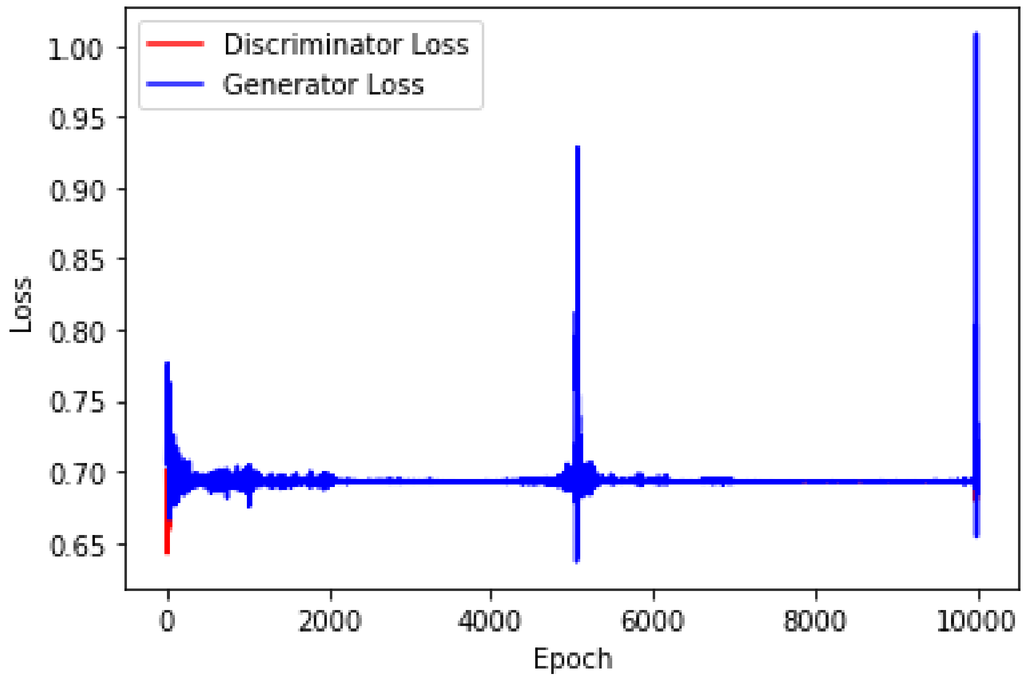
[10]:
```
print(generator)
print(discriminator)
```

```
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()
)
```

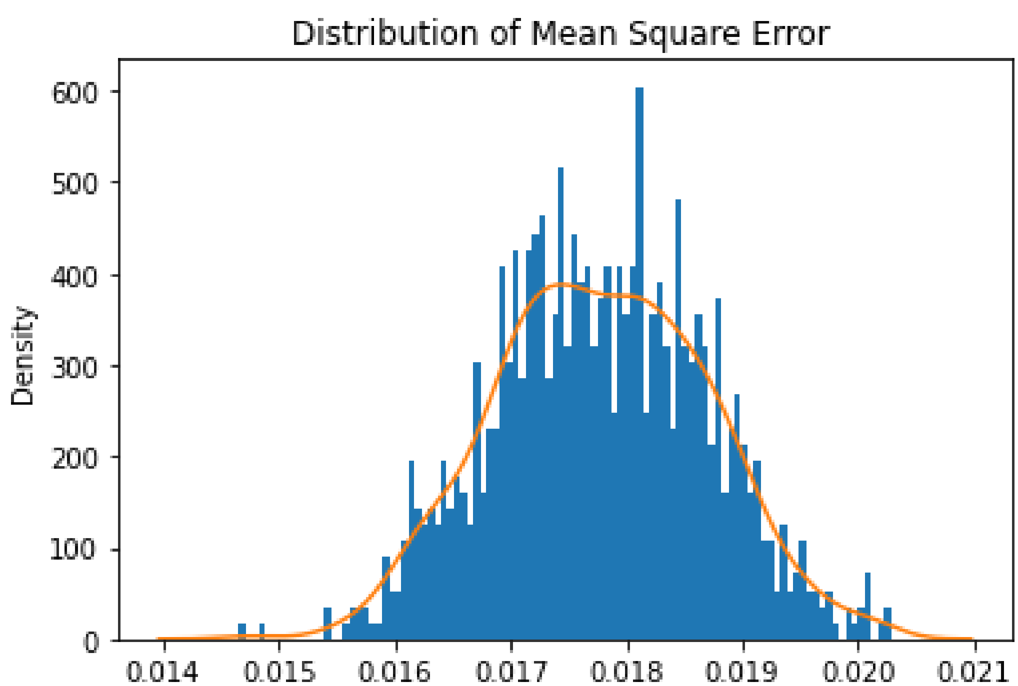
[11]:
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:
```
# Parameters
sample_size = 1000000
std = 1
mean = 1
```

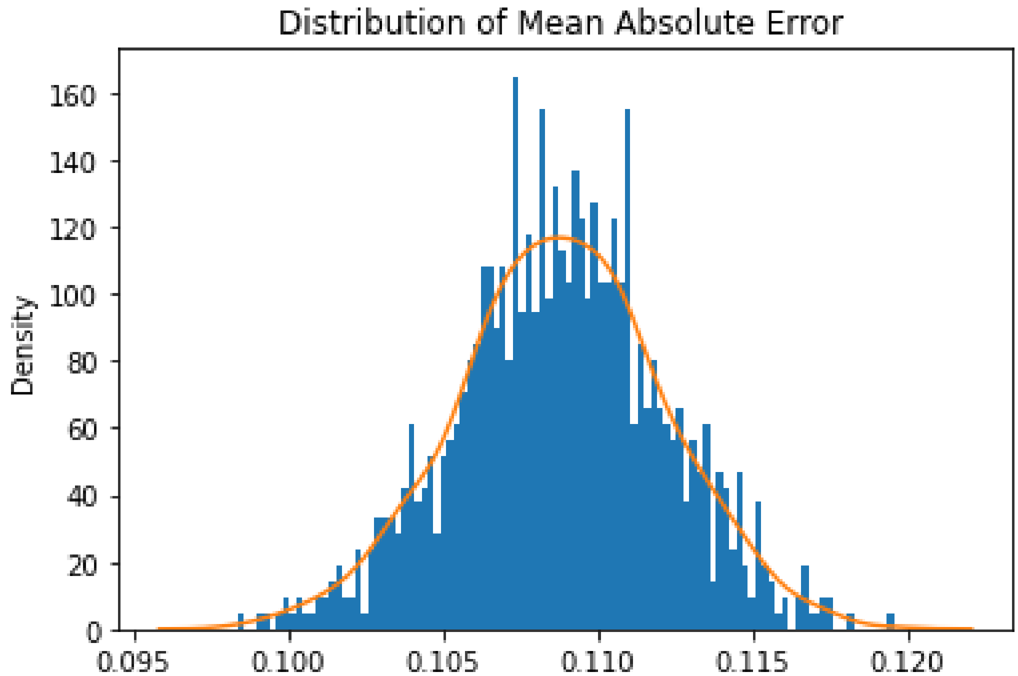
[13]:
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



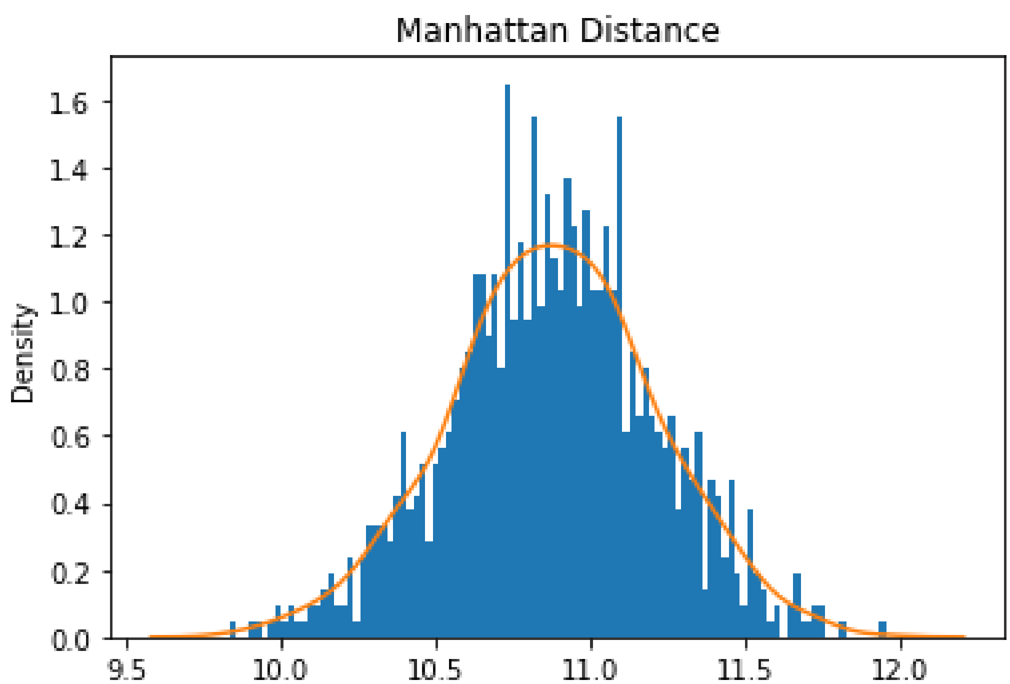
[14]:
```
train_test.test_generator(generator,real_dataset,device)
```



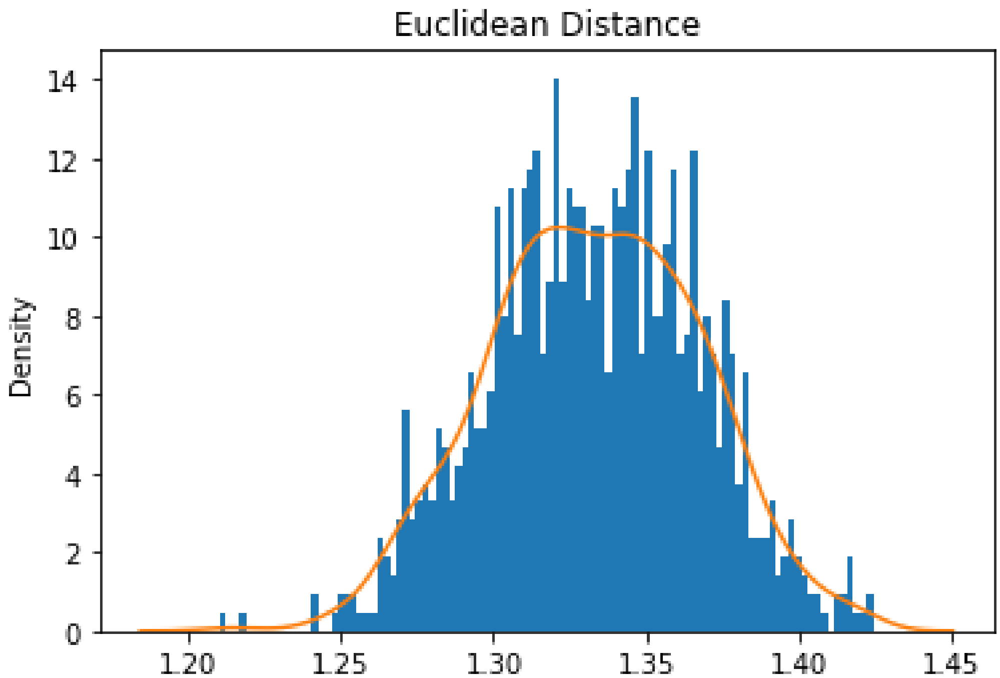
Mean Square Error: 0.017773335584112345



Mean Absolute Error: 0.10880879941302818



```
Mean Manhattan Distance: 10.880879941302817
```




```
Mean Euclidean Distance: 10.880879941302817
```

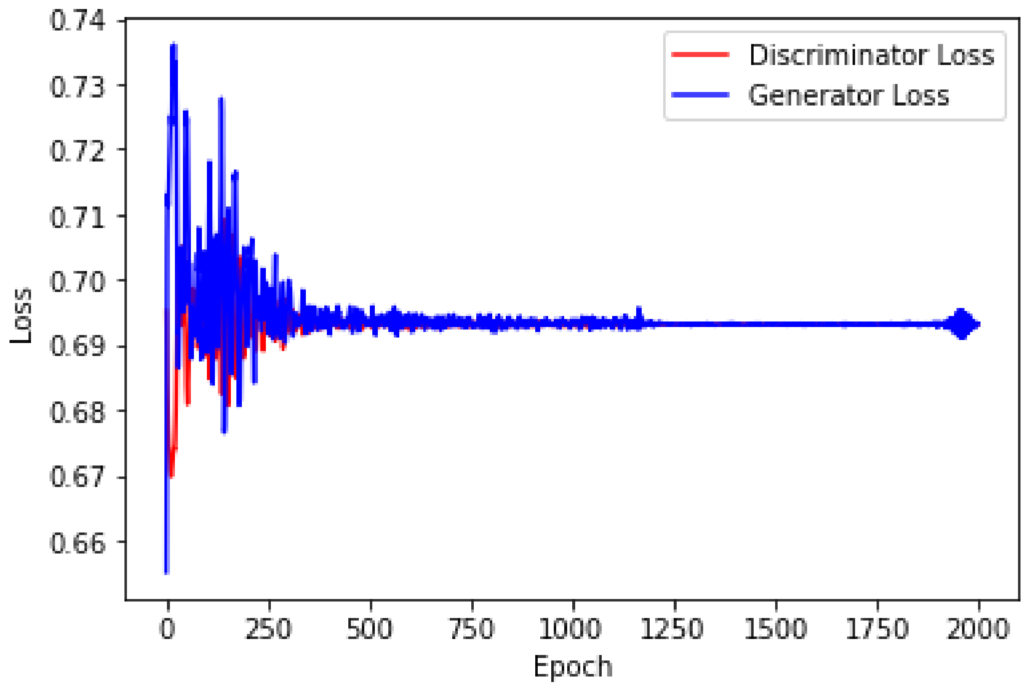
4 ABC GAN Model

Training the network

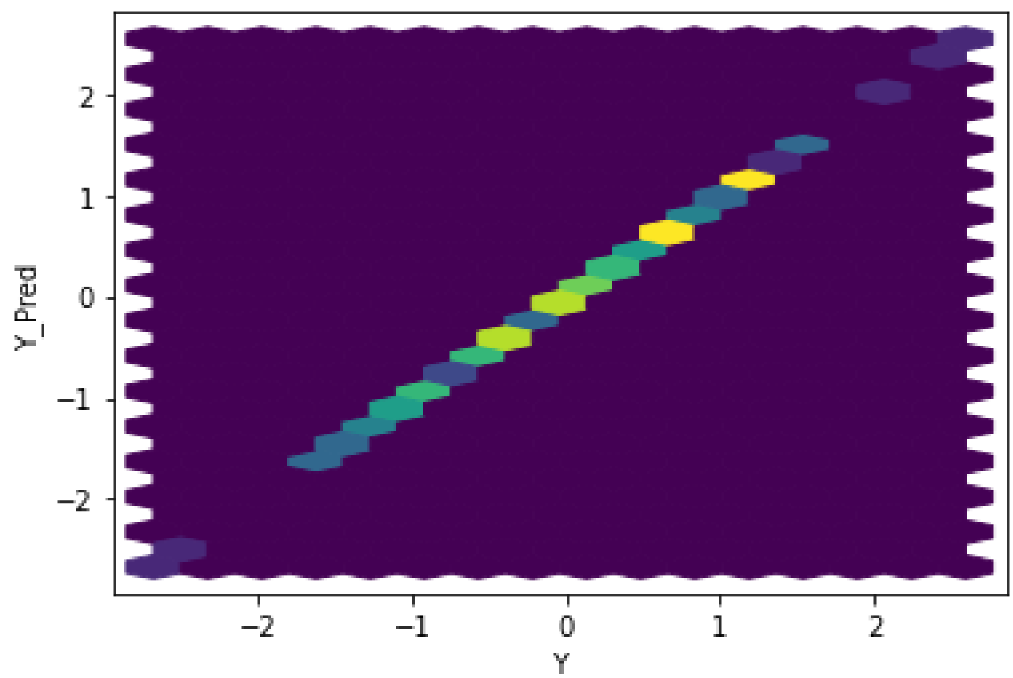
[15]:
```
gen = Generator(n_features+2)
disc = Discriminator(n_features+2,hidden_nodes)

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))
```

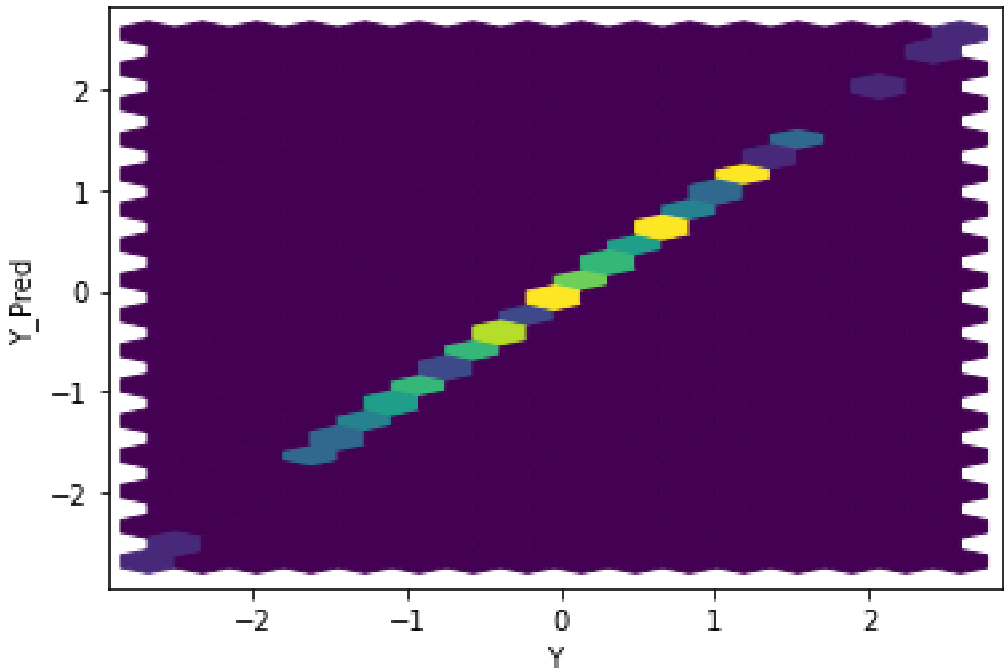
[16]:
```
n_epoch_abc = 2000
batch_size = sample_size//2
```

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

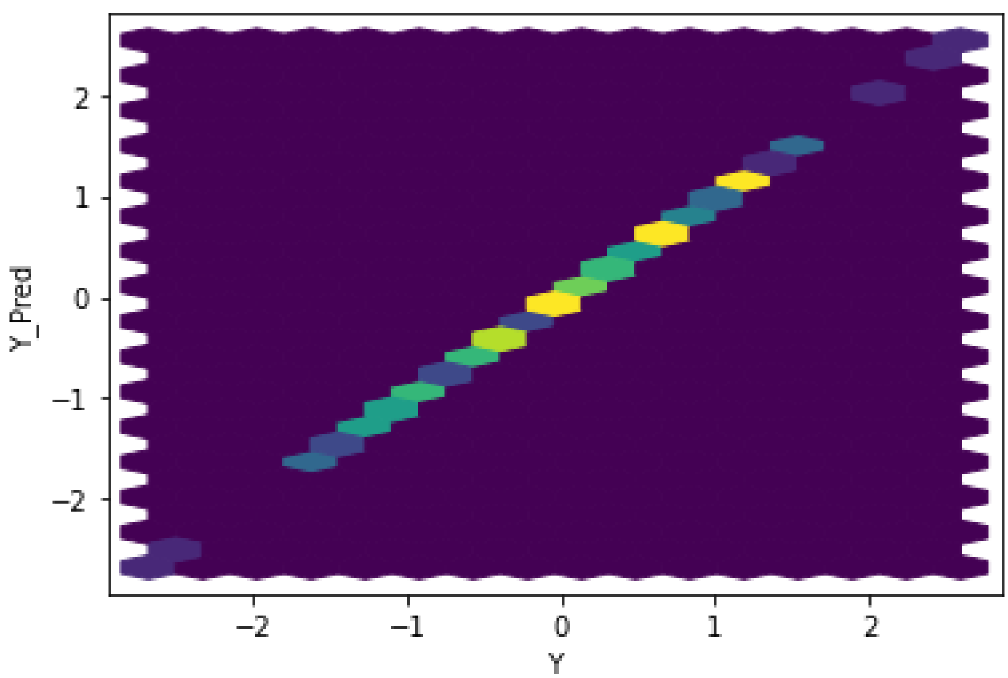


[18]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

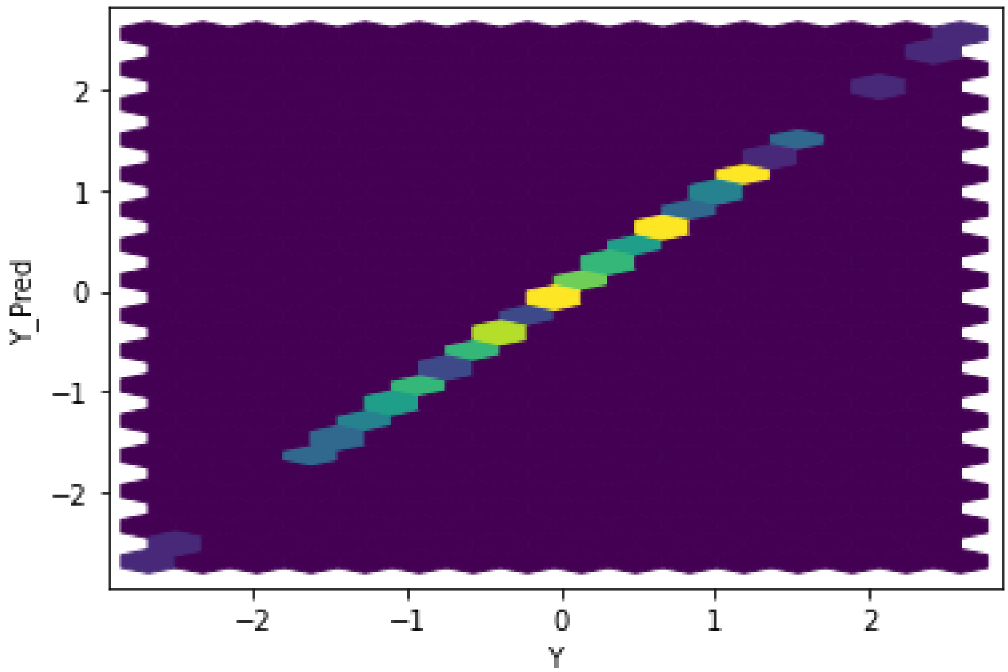




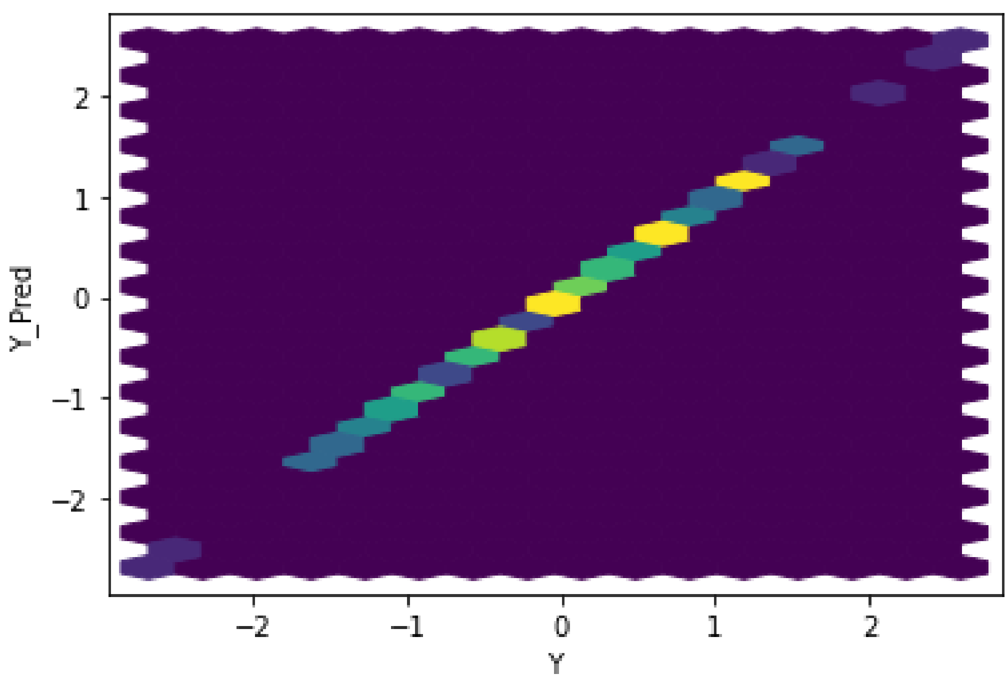
Y_Pred
Y



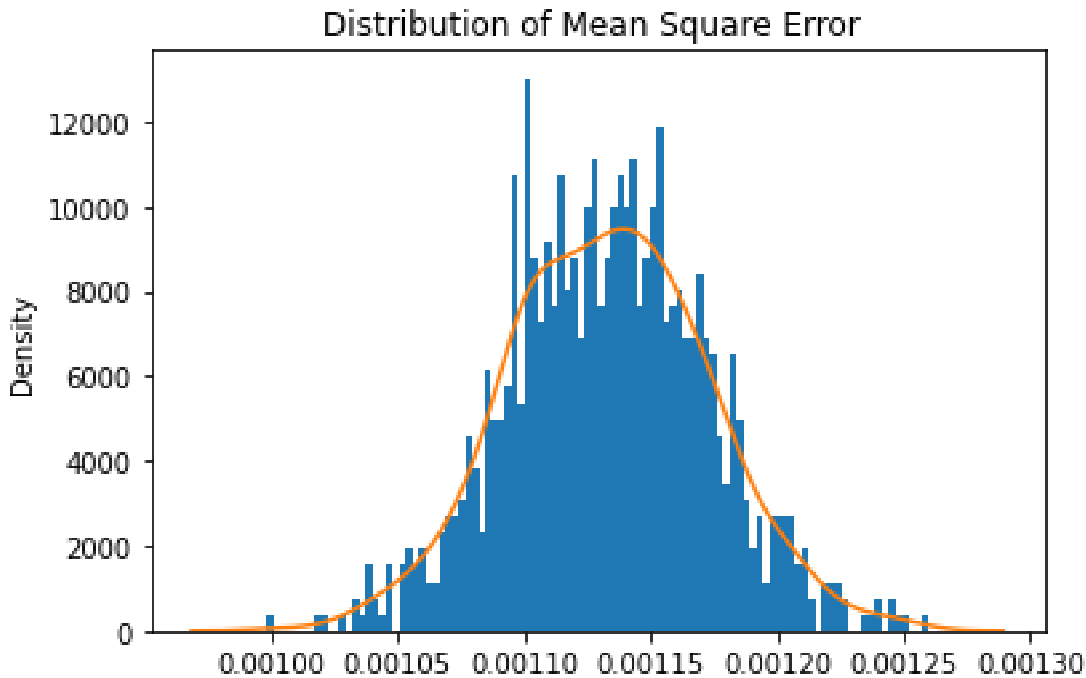
Y_Pred
Y



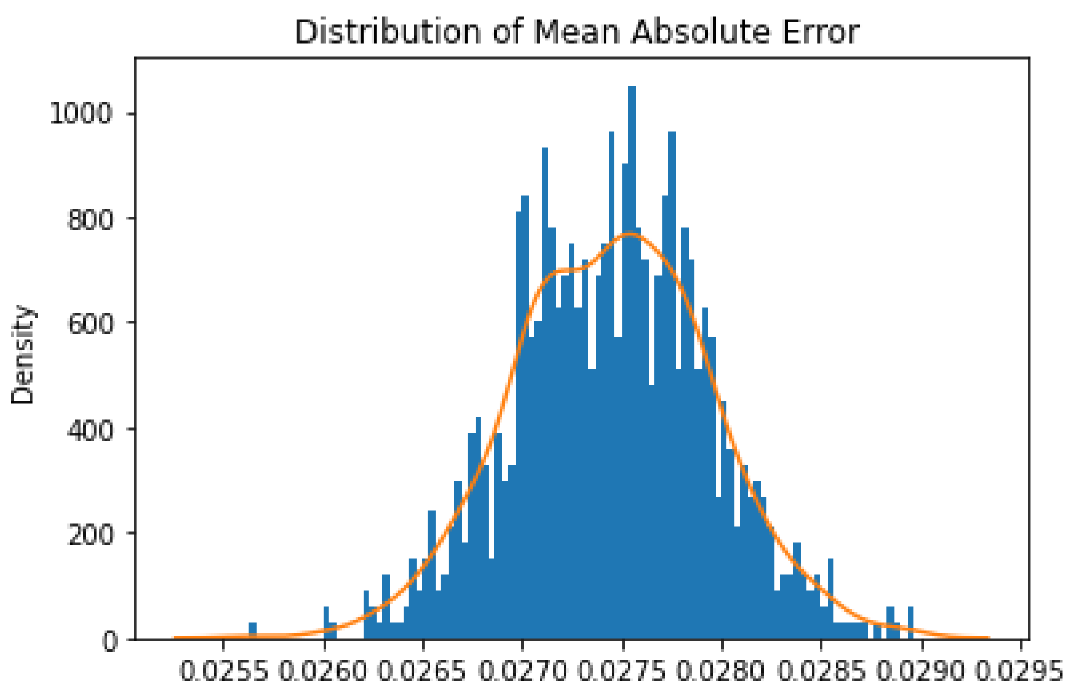
Y_Pred
Y



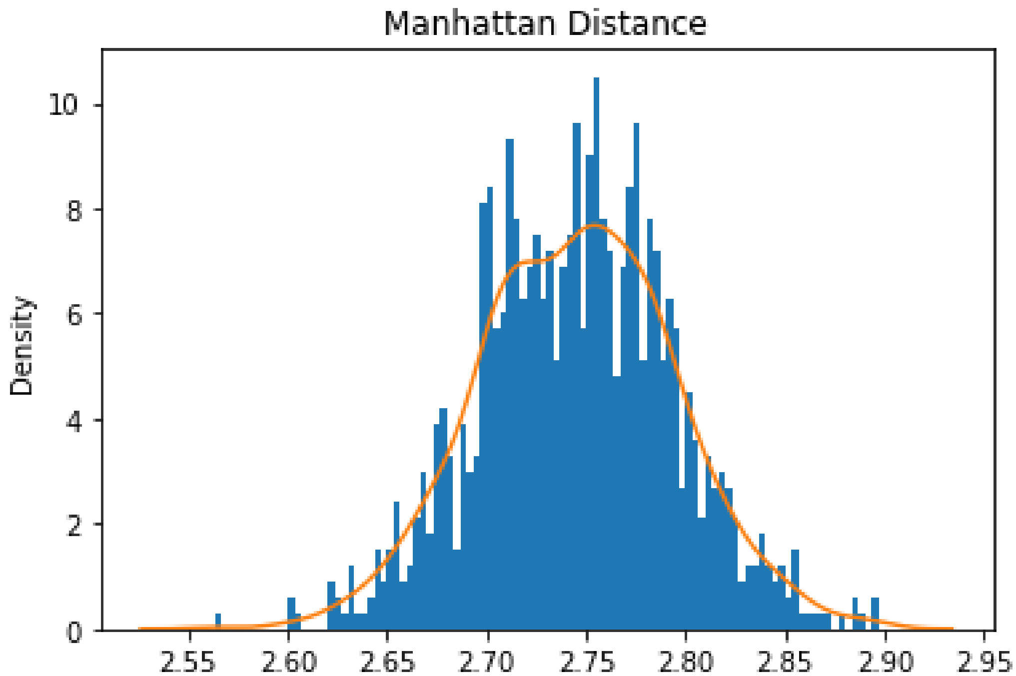
Y_Pred
Y



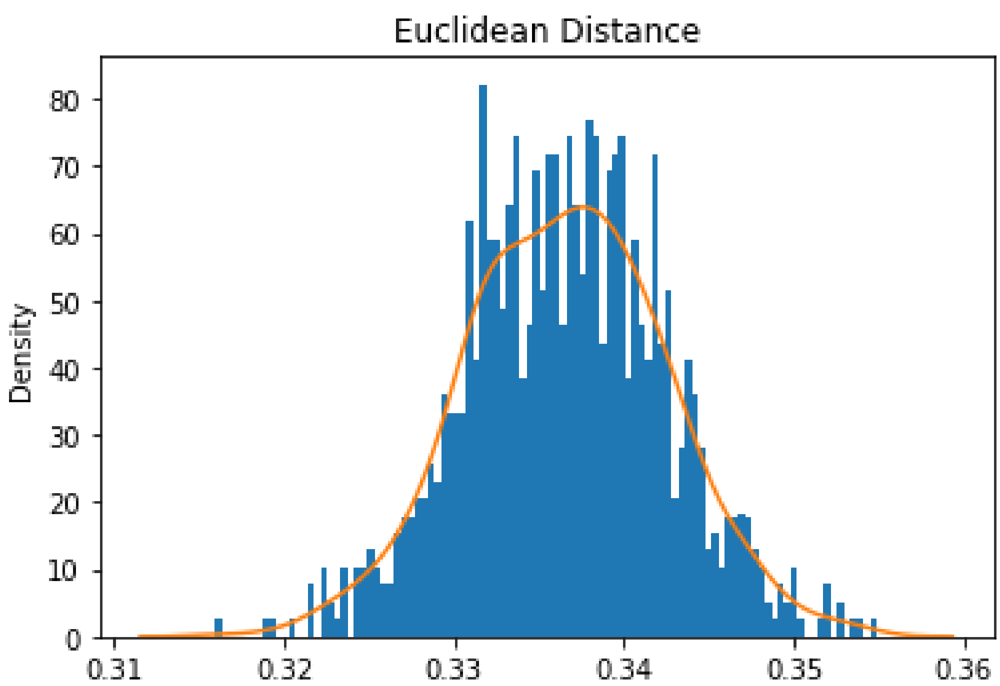

```
Mean Square Error: 0.001133103013615103
```



```
Mean Absolute Error: 0.027451700706649574
Mean Manhattan Distance: 2.7451700706649573
```

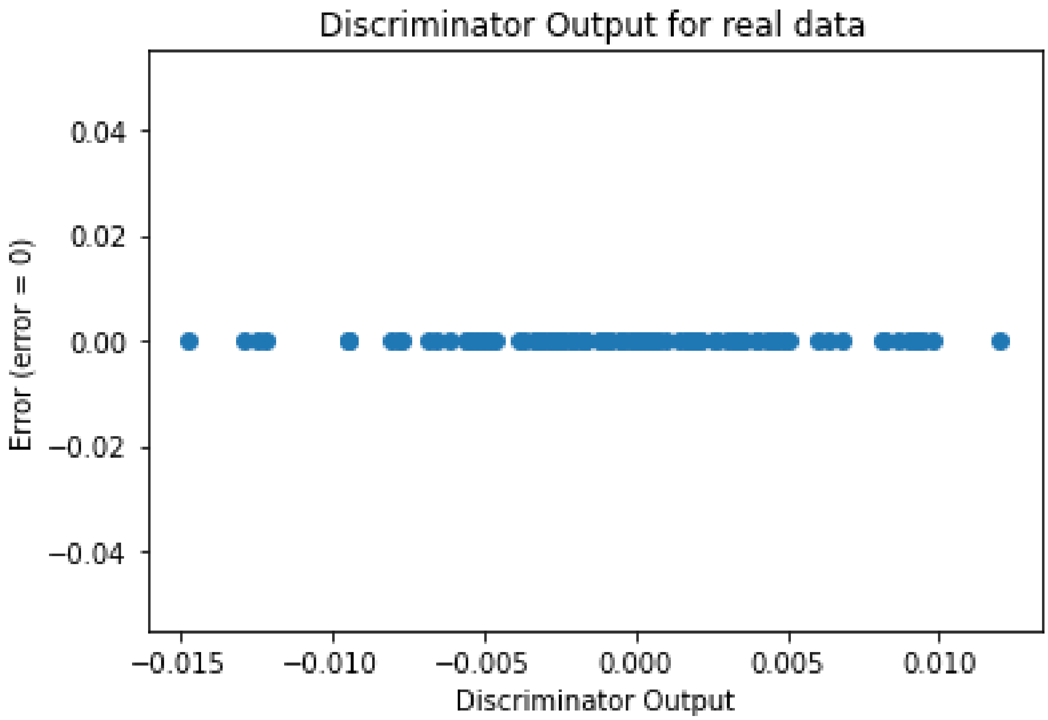


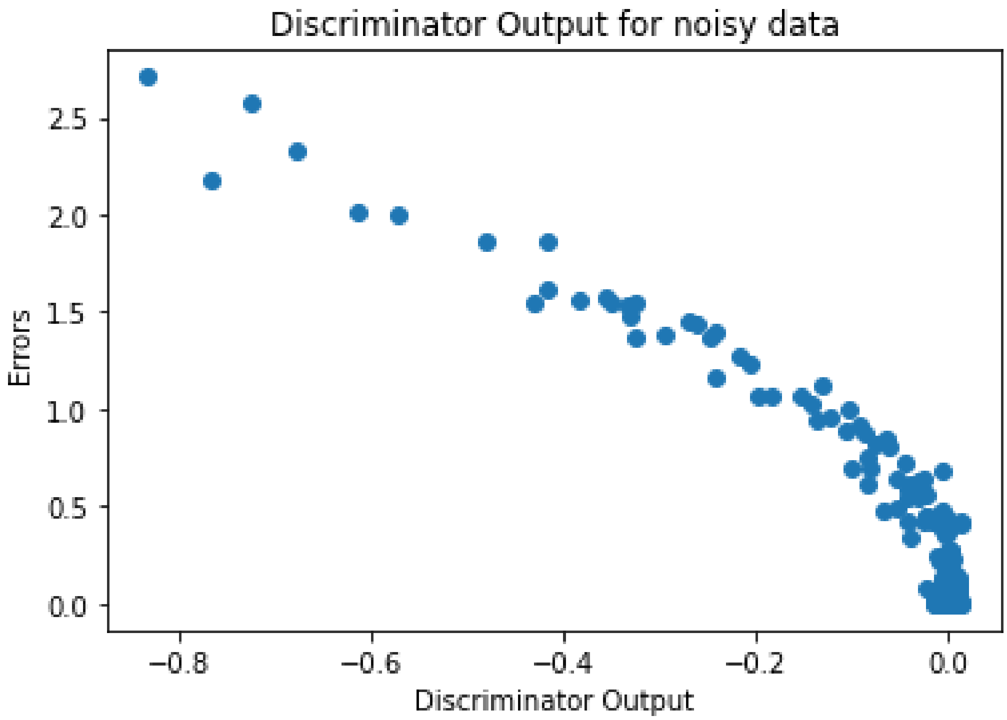

```
Mean Euclidean Distance: 0.3365645969012302
```



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([[0.0635, 0.0066, 0.1357, 0.0296, 0.1790, 0.1926, 0.1678, 0.0720, 0.1906,
         0.2278, 0.1439, 0.5368]], requires_grad=True)
output.bias Parameter containing:
tensor([-0.0686], requires_grad=True)
```