Dataset1-Regression_output_2

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)
```

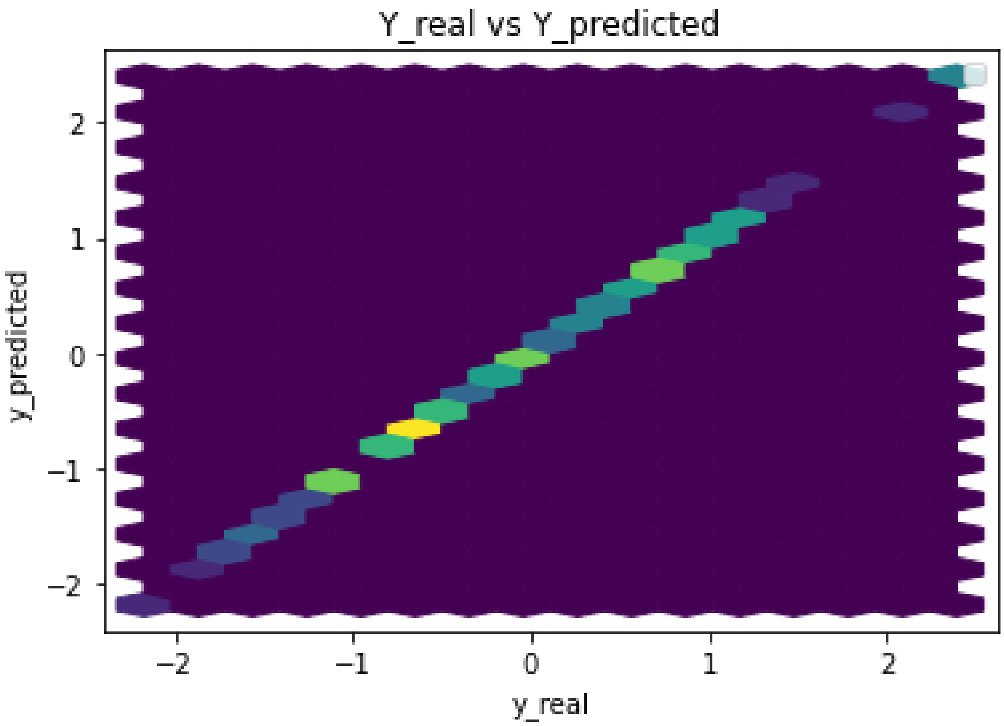
```
         X1        X2        X3        X4        X5        X6        X7  \
0  0.496606  0.754056 -1.481941 -0.322463  1.143216 -3.083906  0.388745
1 -0.087515  0.016083 -0.404803  0.606989  0.831513 -0.407532 -1.518628
2  0.763744  1.238002  0.746380 -0.685008 -1.020754  0.210082 -1.132184
3 -1.127485  0.325044 -0.077155  0.435578  2.151626 -0.626541 -0.192837
4  0.835482  0.803328 -0.571032  0.933814 -1.016780  0.830831 -0.822375

         X8        X9       X10           Y
0 -0.753045 -0.229898 -0.715254 -202.255405
1  0.755717  0.907116  0.403376  112.276105
2 -0.935086 -0.077175  1.197755   -1.965458
3 -0.642796  1.259252 -0.915190  -76.672677
4 -0.308205 -1.276515  0.655377    2.307771
```

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:                 4.304e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          7.07e-293
Time:                        18:56:04   Log-Likelihood:                 627.68
No. Observations:                 100   AIC:                            -1233.
Df Residuals:                      89   BIC:                            -1205.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const               0   4.82e-05          0      1.000   -9.58e-05    9.58e-05
x1             0.4447   4.96e-05   8964.833      0.000       0.445       0.445
x2             0.1140   5.04e-05   2259.667      0.000       0.114       0.114
x3             0.0664   4.92e-05   1349.197      0.000       0.066       0.067
x4             0.1185   4.98e-05   2378.330      0.000       0.118       0.119
x5             0.1878   5.03e-05   3736.036      0.000       0.188       0.188
```

```
x6            0.2862   4.96e-05   5765.902      0.000       0.286       0.286
x7            0.1806   4.96e-05   3641.246      0.000       0.181       0.181
x8            0.5168   5.06e-05   1.02e+04      0.000       0.517       0.517
x9            0.2946   5.02e-05   5868.065      0.000       0.294       0.295
x10           0.3465   4.93e-05   7034.750      0.000       0.346       0.347
==============================================================================
Omnibus:                        5.309   Durbin-Watson:                   2.417
Prob(Omnibus):                  0.070   Jarque-Bera (JB):                3.735
Skew:                          -0.327   Prob(JB):                        0.154
Kurtosis:                       2.315   Cond. No.                         1.52
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    0.000000
x1       0.444720
x2       0.113982
x3       0.066437
x4       0.118474
x5       0.187850
x6       0.286196
x7       0.180619
x8       0.516792
x9       0.294566
x10      0.346538
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 2.0680021043098653e-07
Mean Absolute Error: 0.0003761983379877468
Manhattan distance: 0.03761983379877468
Euclidean distance: 0.004547529114046291
```

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))
```

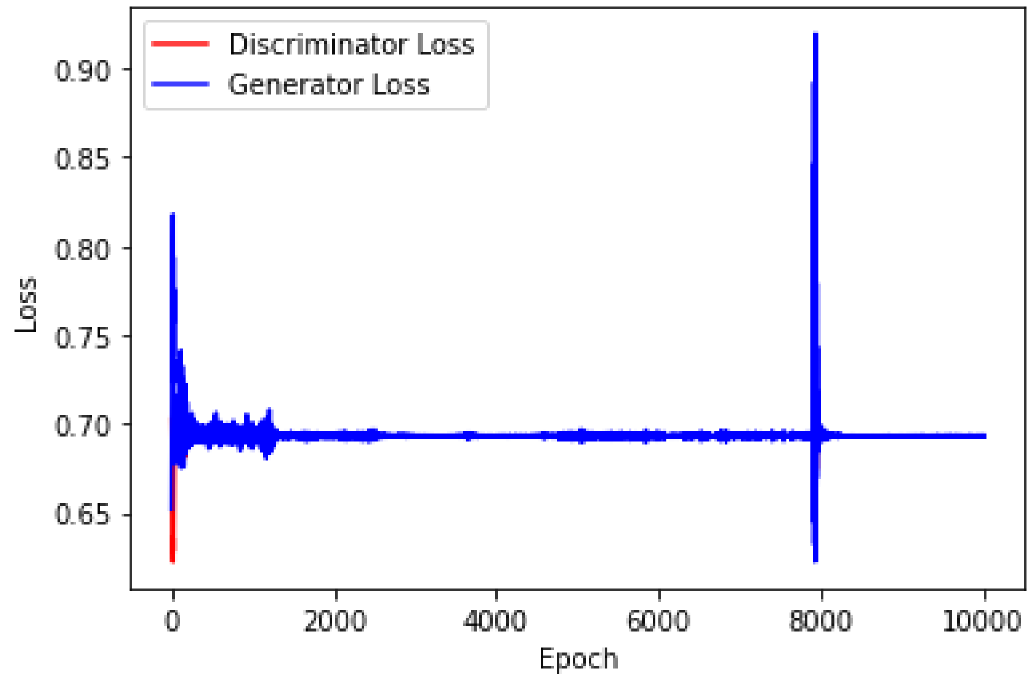
[10]:
```
print(generator)
print(discriminator)
```

```
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()
)
```

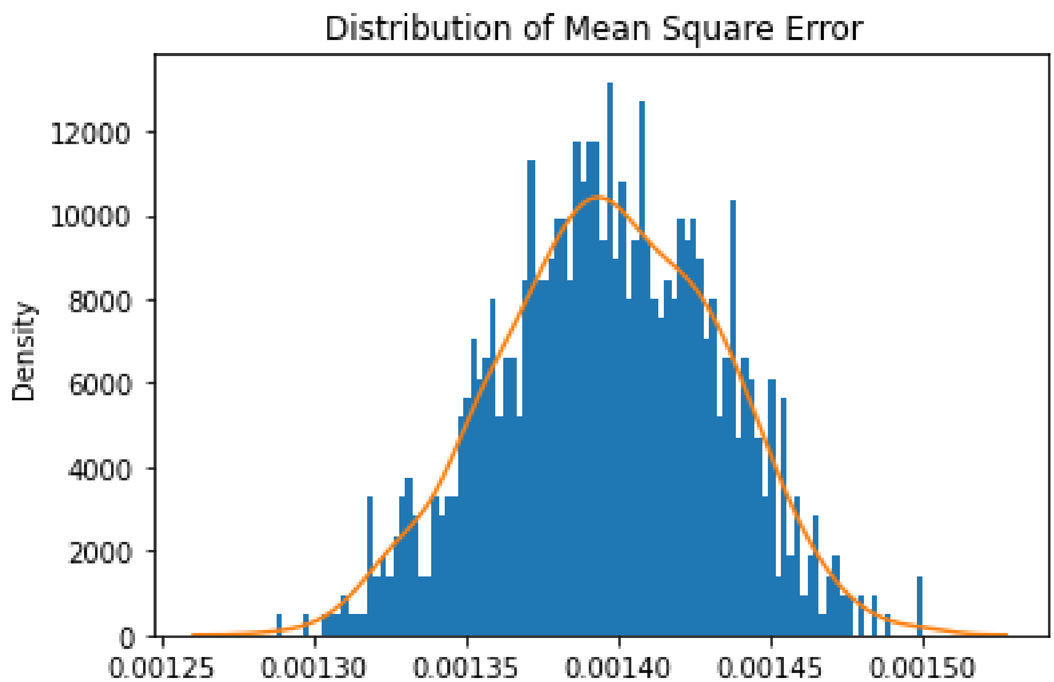
[11]:
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:
```
# Parameters
sample_size = 10000
mean = 1
std = 1
```

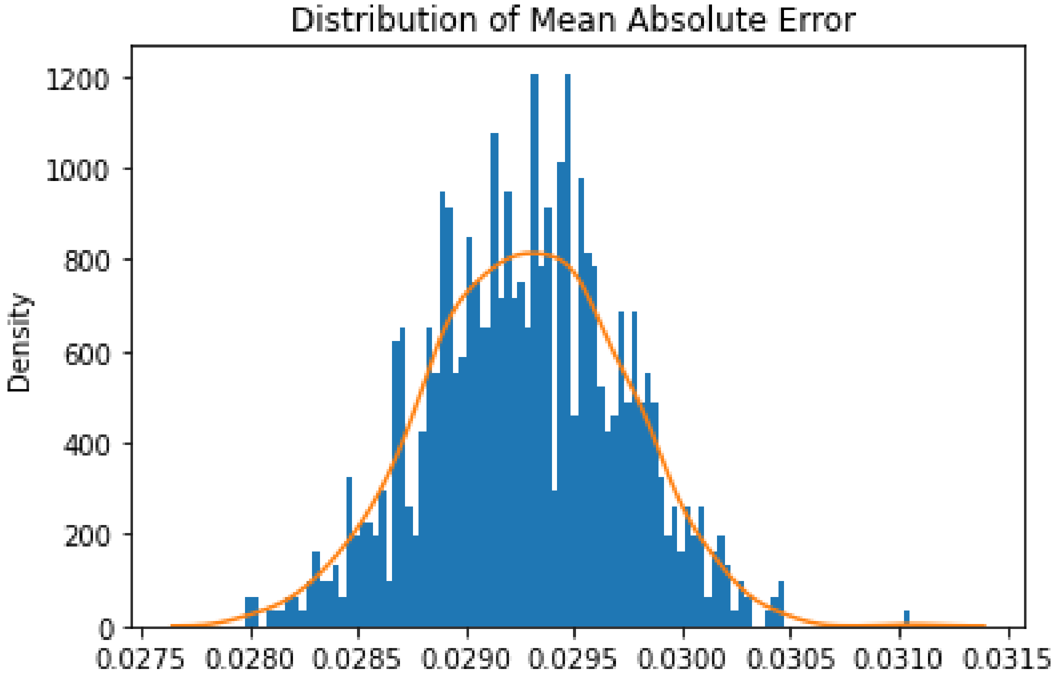
[13]:
```
train_test.
↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
↪n_epochs,criterion,device)
```



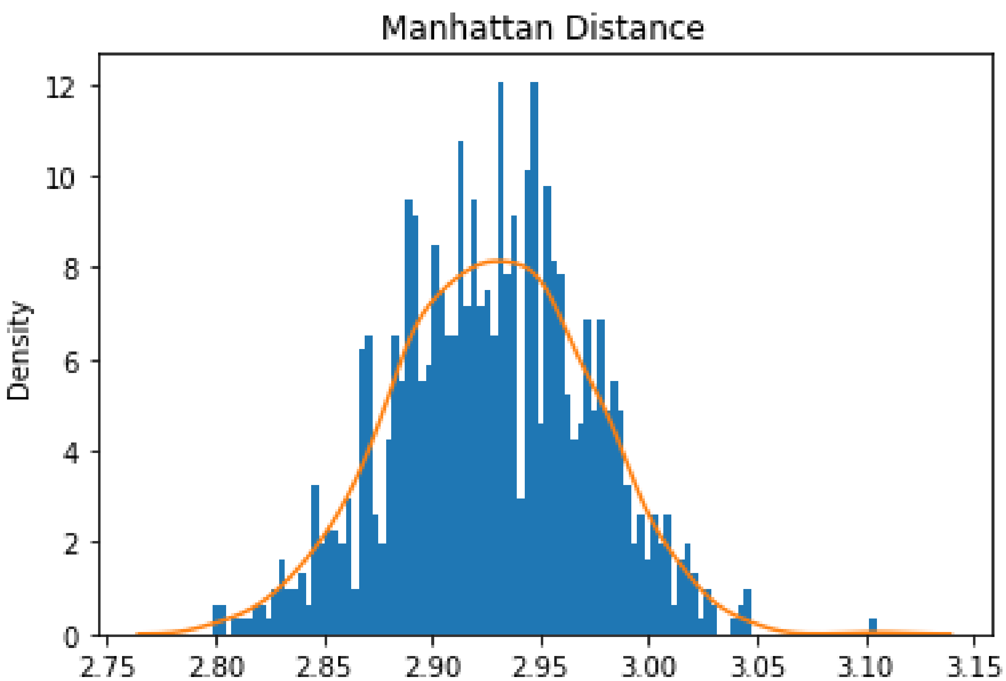
[14]:
```
train_test.test_generator(generator,real_dataset,device)
```



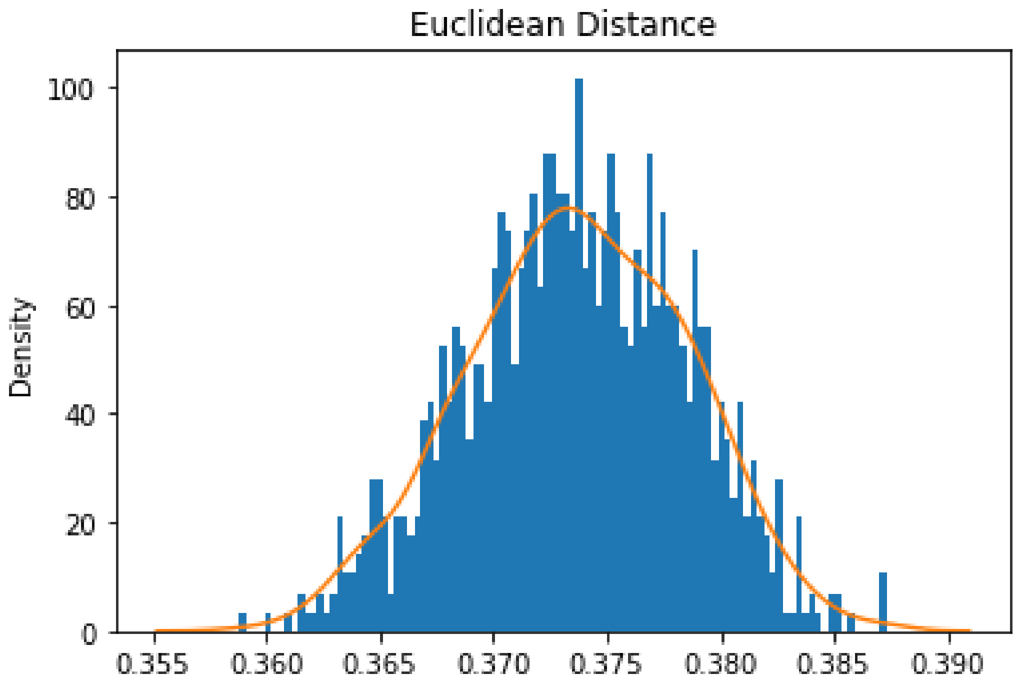
```
Mean Square Error: 0.0013963873162586966
```




```
Mean Absolute Error: 0.029275716423168778
```



```
Mean Manhattan Distance: 2.927571642316878
```




```
Mean Euclidean Distance: 2.927571642316878
```

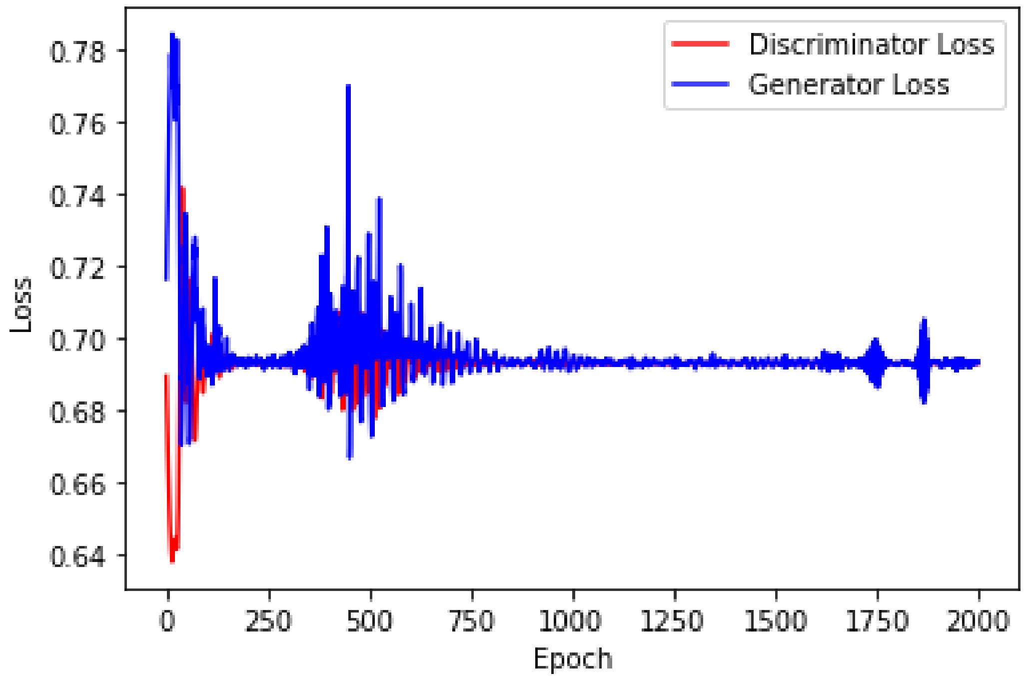
4 ABC GAN Model

Training the network

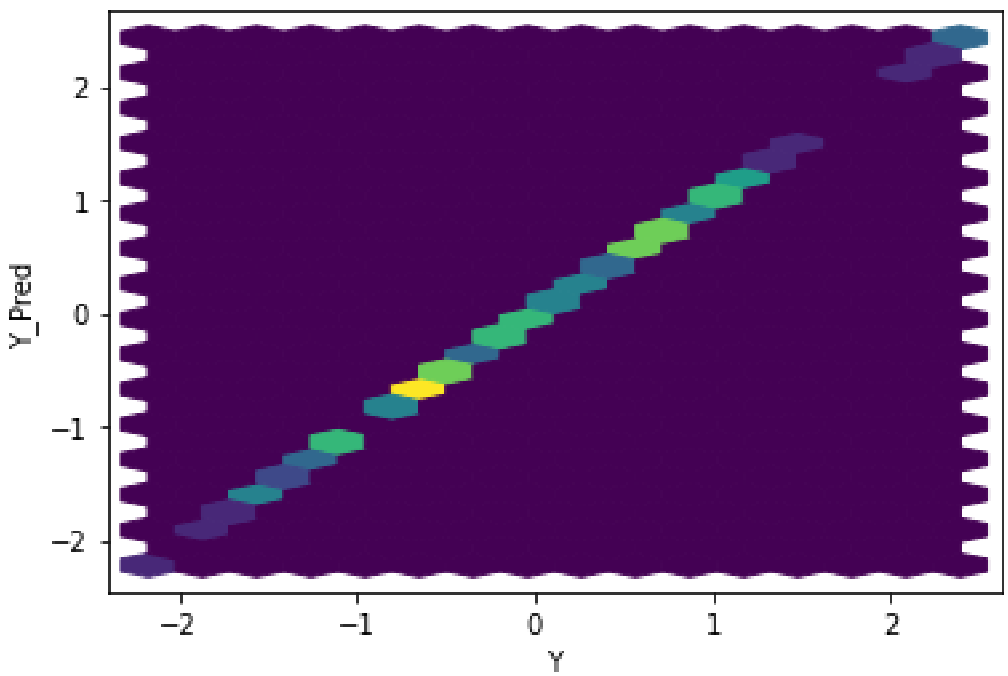
[15]:
```
gen = Generator(n_features+2)
disc = Discriminator(n_features+2,hidden_nodes)

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))
```

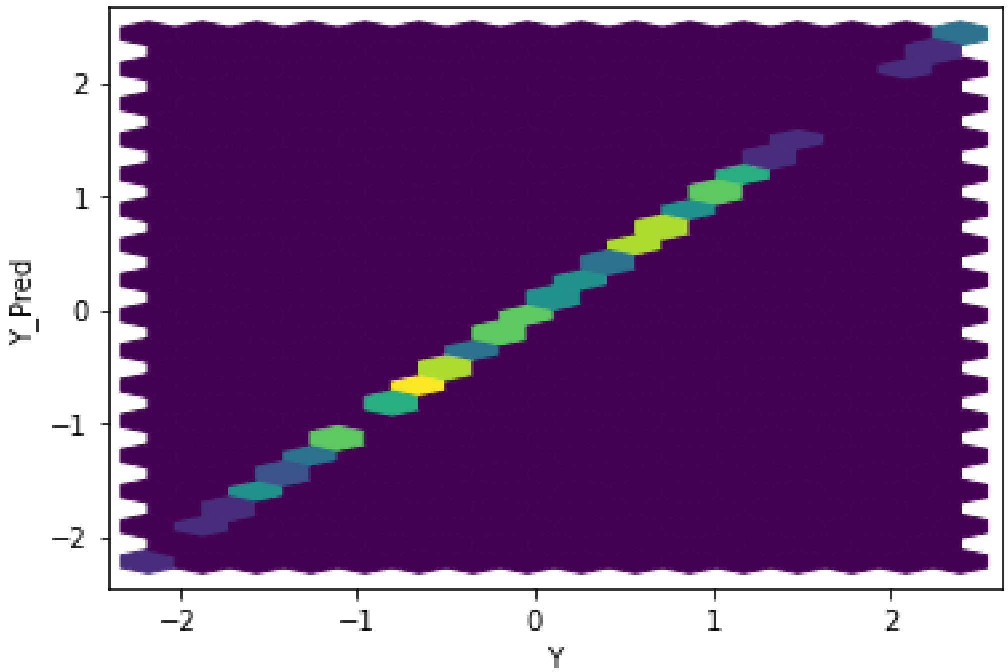
[16]:
```
n_epoch_abc = 2000
batch_size = sample_size//2
```

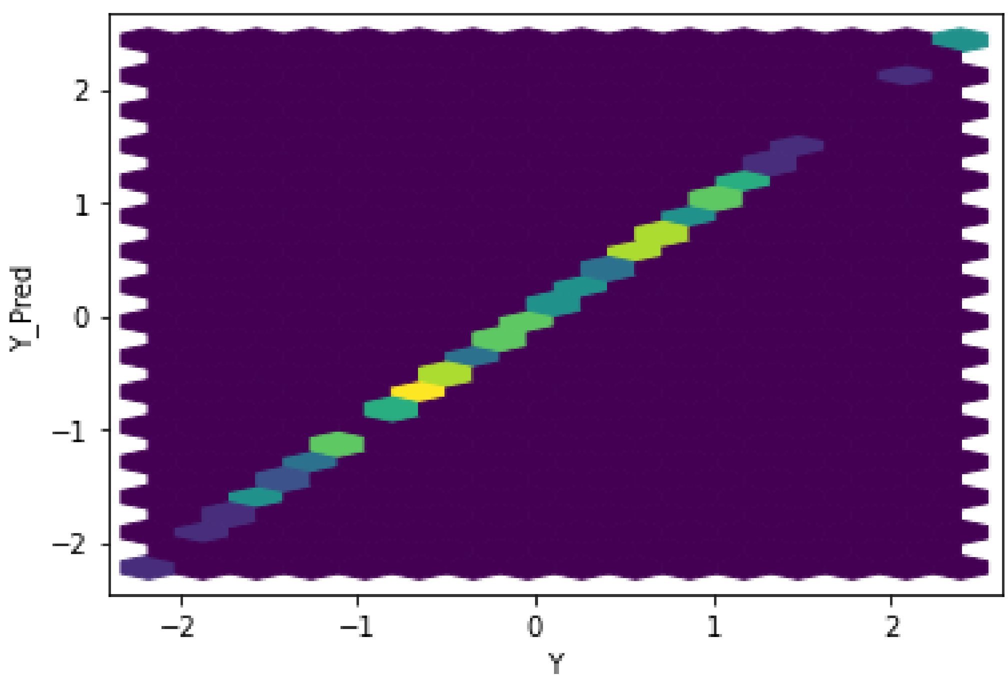
[17]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

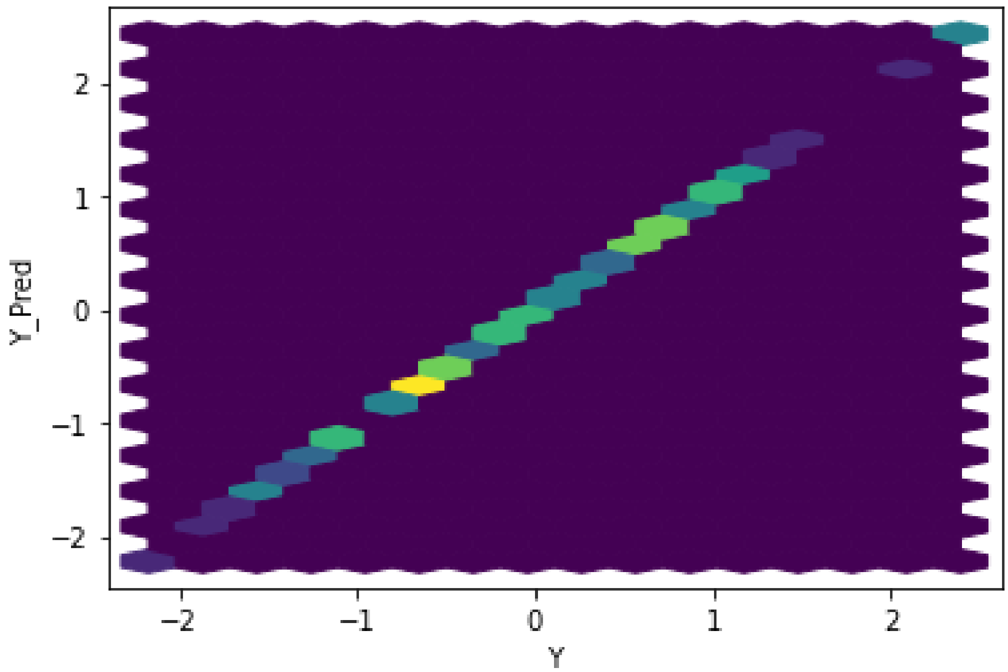


[18]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

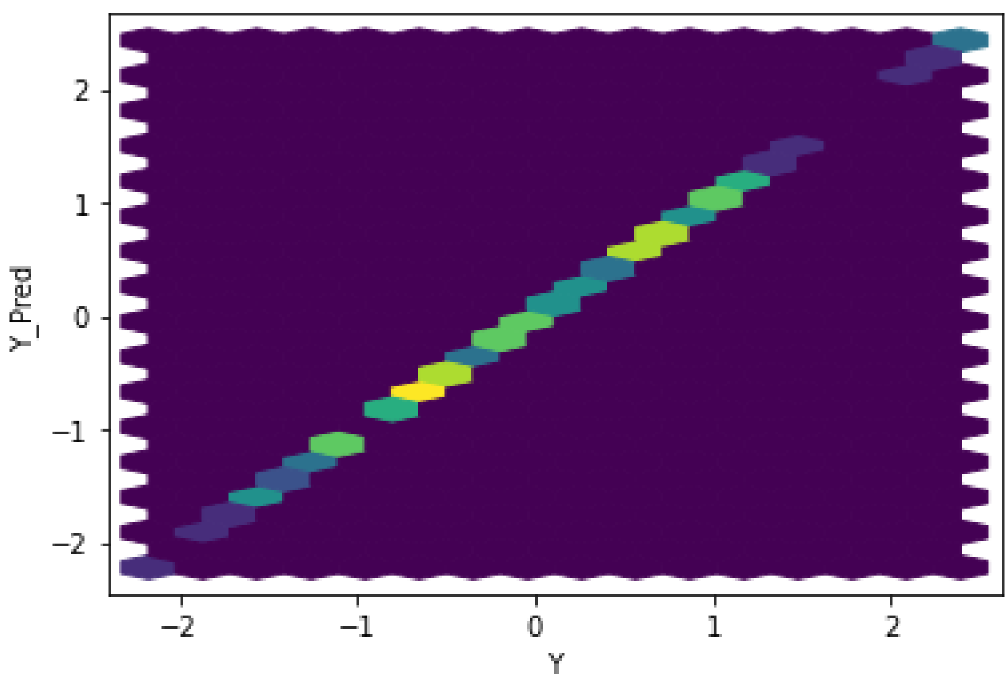



Y_Pred
Y
−2
−1
0
1
2

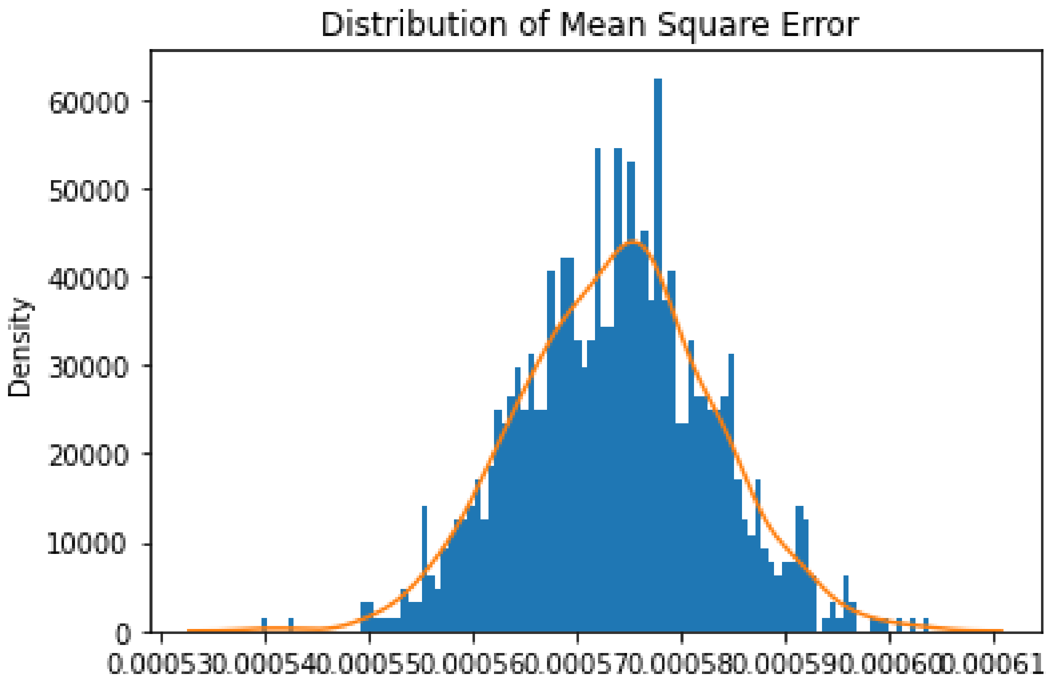

Y_Pred
Y
−2
−1
0
1
2



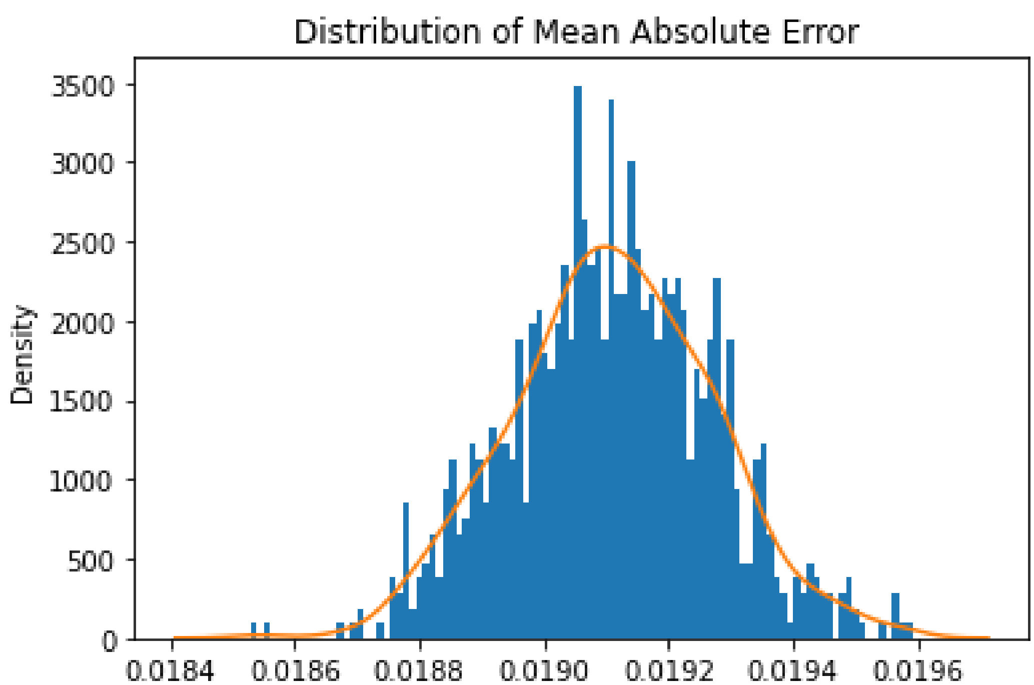
Y_Pred
Y



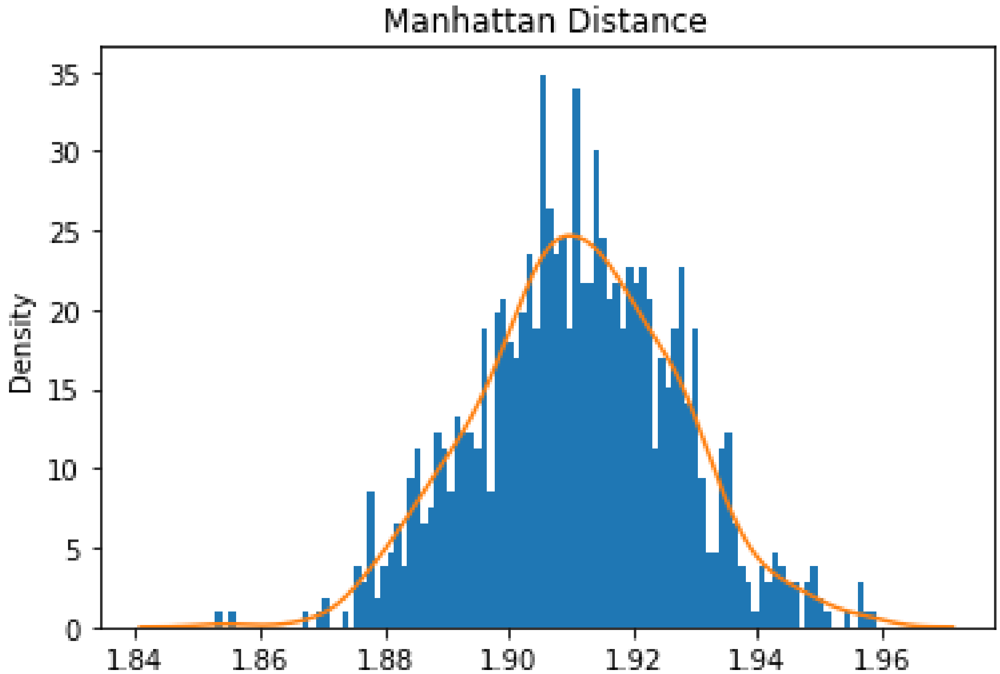
Y_Pred
Y



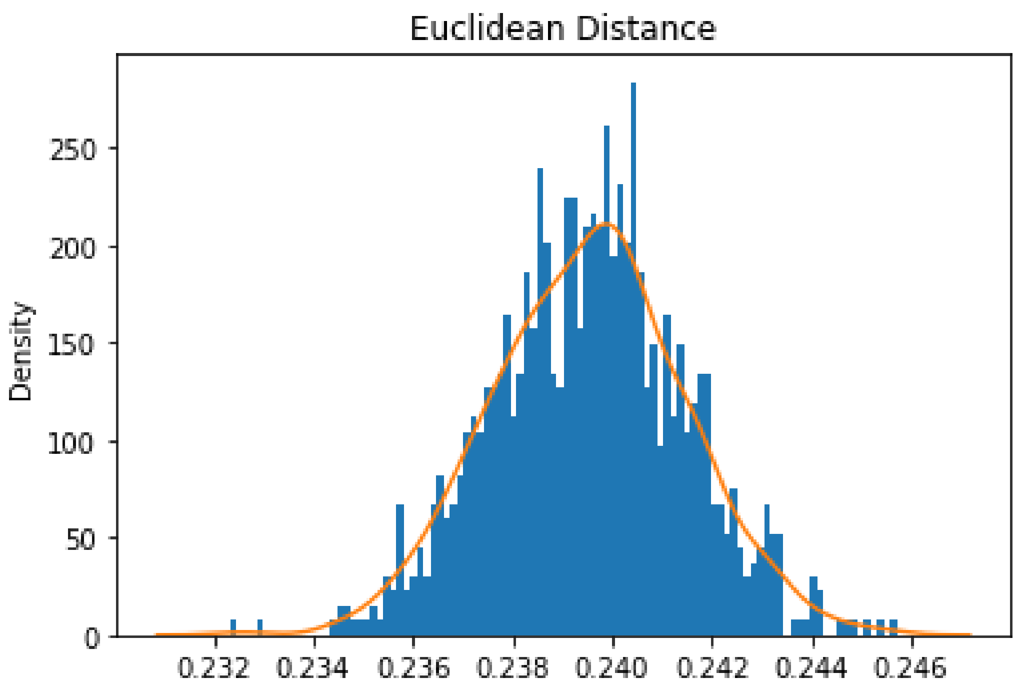

```
Mean Square Error: 0.0005737583791617501
```



```
Mean Absolute Error: 0.019109621034264564
Mean Manhattan Distance: 1.9109621034264566
```

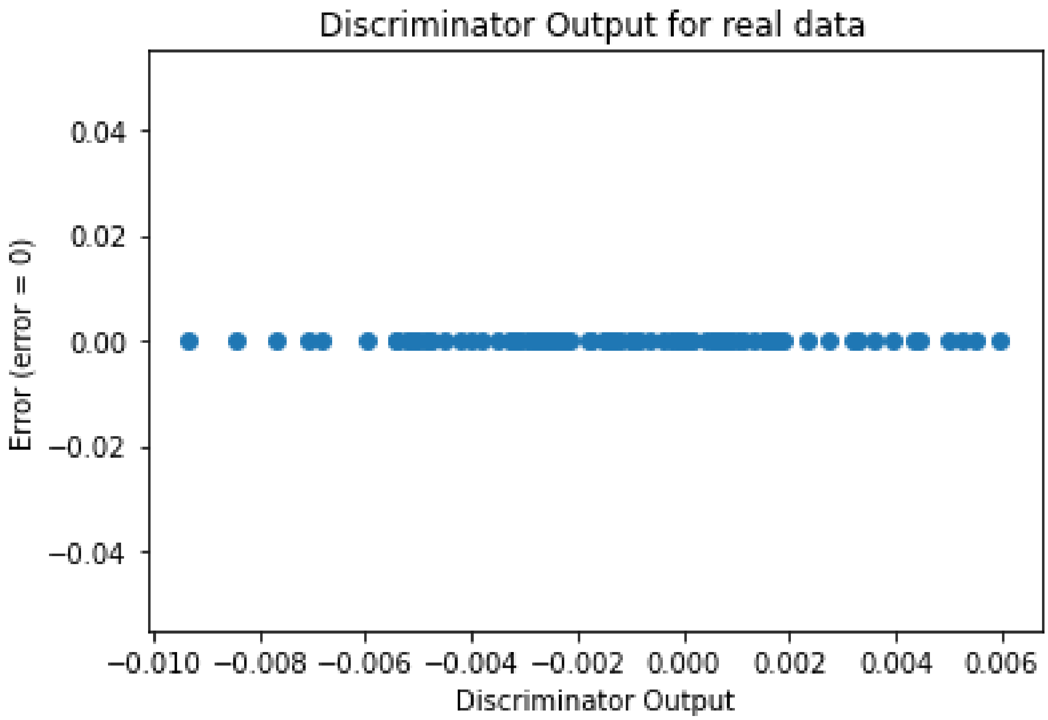


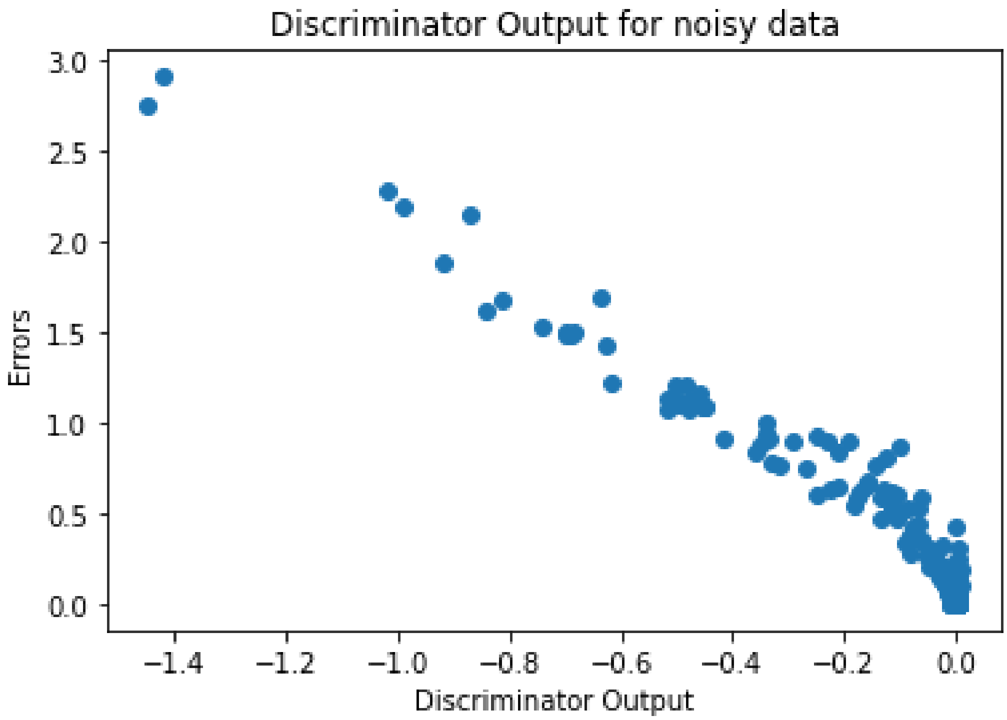

```
Mean Euclidean Distance: 0.23952480340270765
```



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([[0.0097, 0.3511, 0.0891, 0.0593, 0.1055, 0.1452, 0.2400, 0.1487, 0.4286,
         0.2421, 0.2923, 0.1884]], requires_grad=True)
output.bias Parameter containing:
tensor([-0.0044], requires_grad=True)
```