Dataset1-Regression_output_14

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 -1.116754  1.430833 -0.577537 -0.627913  2.291493 -2.088936  0.033238
1  0.406499  1.111493  0.643894  0.327376  1.325048  1.535105 -0.050613
2 -0.133592  0.924598 -1.508380  0.332899  1.353804  0.555442  1.198784
3 -0.042678 -0.384300 -0.691338 -1.031297  2.312522  1.665538  1.058641
4  0.028844 -0.335899 -0.031172  0.776656 -0.624756  0.594004  0.577952

         X8        X9       X10           Y
0  1.055504 -0.477446 -0.193944  -56.609701
1  0.715199 -1.135080  1.030566  265.772133
2  0.104877 -0.144629  0.865847  183.348799
3 -0.436929  0.277090  0.758939   21.385726
4  1.158505 -0.062871 -1.131395  106.921863
```

1.4 Stats Model

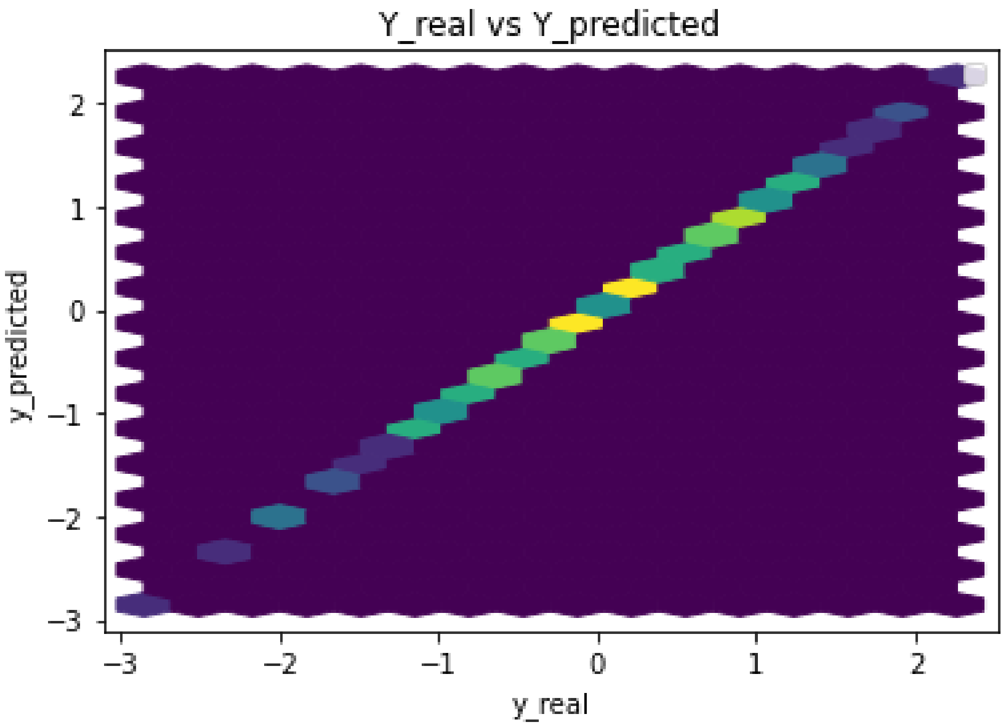
[4]:
```
[coeff,y_pred] = statsModel.statsModel(X,Y)
```

```
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:                 4.465e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          1.38e-293
Time:                        07:46:19   Log-Likelihood:                 629.52
No. Observations:                 100   AIC:                            -1237.
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   4.73e-05  -1.47e-13      1.000    -9.4e-05     9.4e-05
x1             0.4032   4.85e-05   8320.801      0.000       0.403       0.403
x2             0.4368   4.83e-05   9046.197      0.000       0.437       0.437
x3             0.1600   4.94e-05   3237.744      0.000       0.160       0.160
x4             0.4685   4.85e-05   9649.510      0.000       0.468       0.469
x5             0.1173   4.98e-05   2357.221      0.000       0.117       0.117
```

```
x6             0.1534   4.87e-05   3147.684      0.000       0.153       0.154
x7             0.1942   4.96e-05   3917.172      0.000       0.194       0.194
x8             0.3181    4.9e-05   6493.150      0.000       0.318       0.318
x9             0.3707   5.03e-05   7364.993      0.000       0.371       0.371
x10            0.2178   4.99e-05   4363.182      0.000       0.218       0.218
==============================================================================
Omnibus:                        0.557   Durbin-Watson:                   2.100
Prob(Omnibus):                  0.757   Jarque-Bera (JB):                0.672
Skew:                           0.016   Prob(JB):                        0.715
Kurtosis:                       2.600   Cond. No.                         1.59
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const   -6.938894e-18
x1       4.032268e-01
x2       4.367509e-01
x3       1.599544e-01
x4       4.684547e-01
x5       1.173011e-01
x6       1.534421e-01
x7       1.941506e-01
x8       3.180862e-01
x9       3.706857e-01
x10      2.178296e-01
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 1.9933376861820828e-07
Mean Absolute Error: 0.000355426656859984
Manhattan distance: 0.0355426656859984
Euclidean distance: 0.004464681048162437
```

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))
```

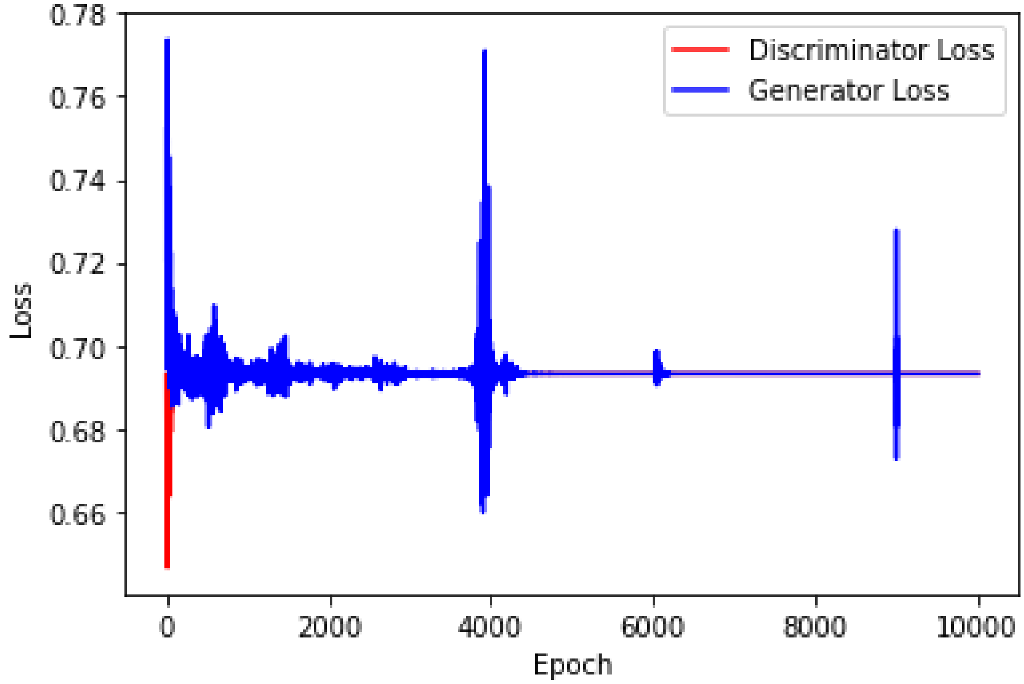
[10]:
```
print(generator)
print(discriminator)
```

```
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()
)
```

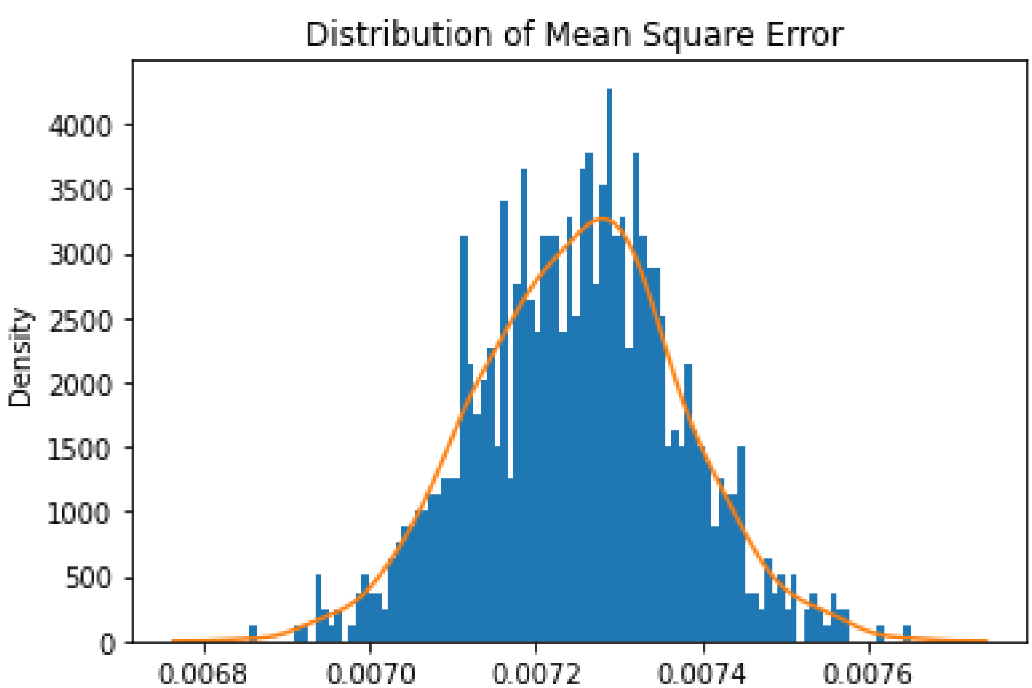
[11]:
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:
```
# Parameters
sample_size = 10000
std = 1
mean = 0.1
```

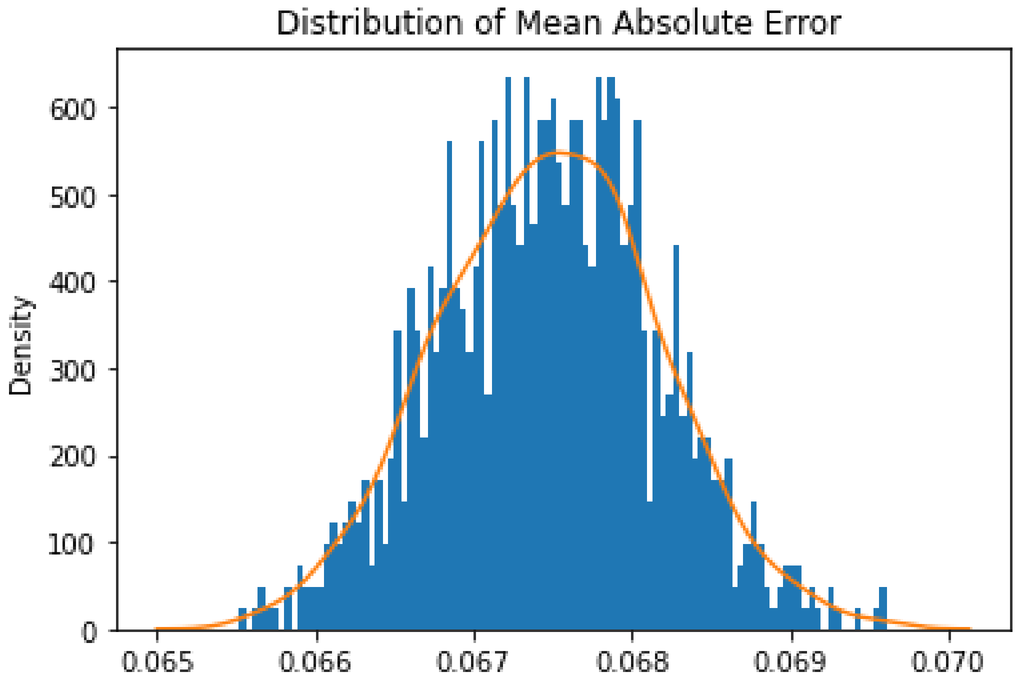
[13]:
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



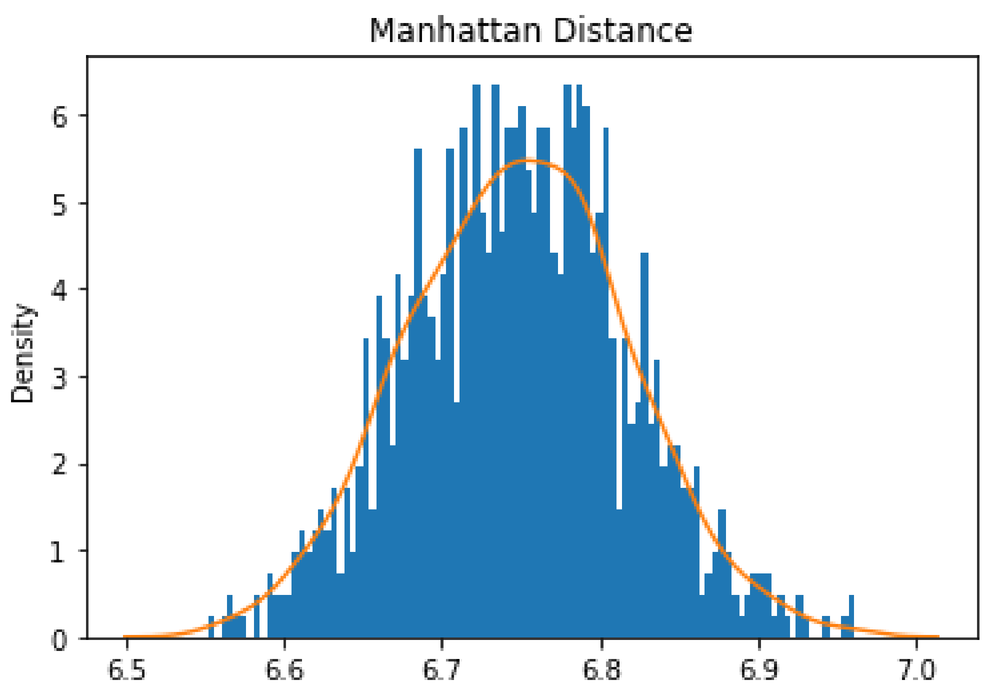
[14]:
```
train_test.test_generator(generator,real_dataset,device)
```



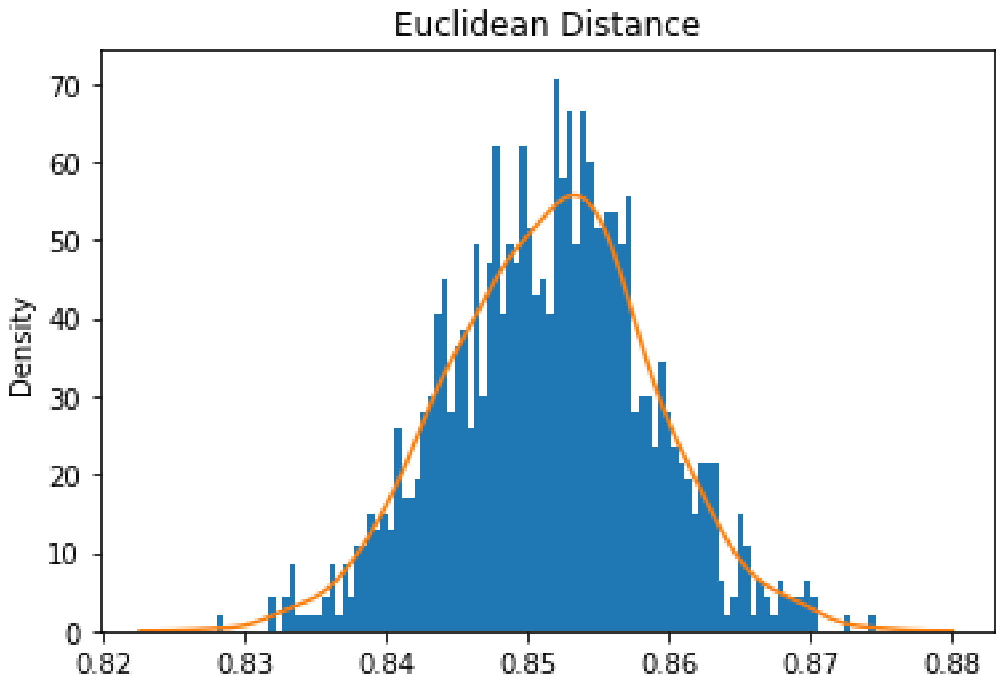
```
Mean Square Error: 0.007251511294670153
```




```
Mean Absolute Error: 0.06747856961668469
```



```
Mean Manhattan Distance: 6.747856961668469
```




```
Mean Euclidean Distance: 6.747856961668469
```

4 ABC GAN Model

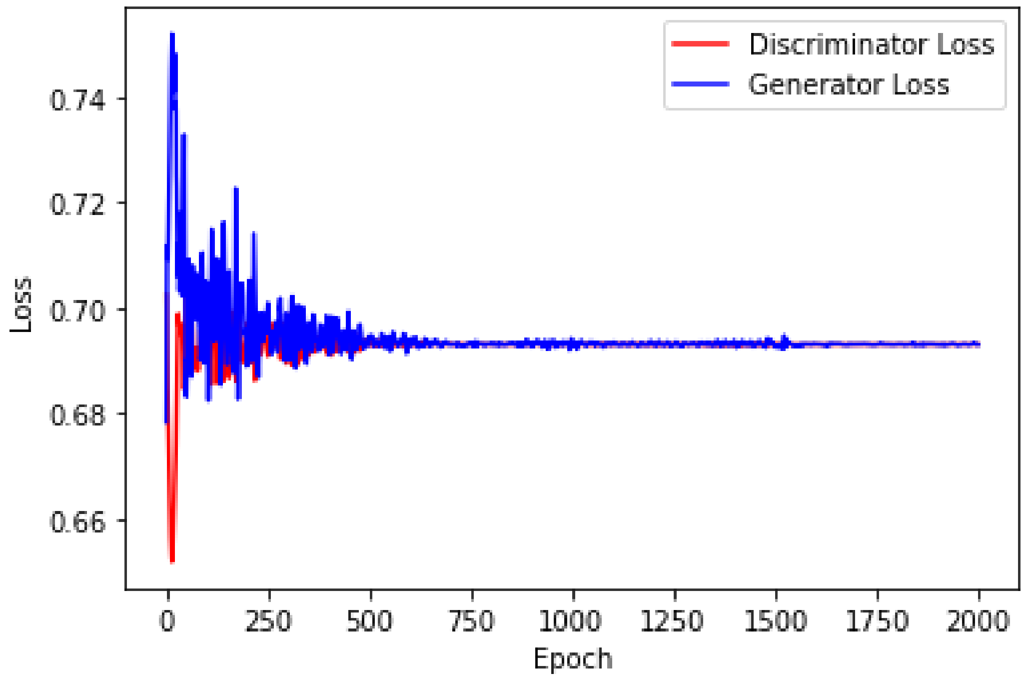
Training the network

[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))
```

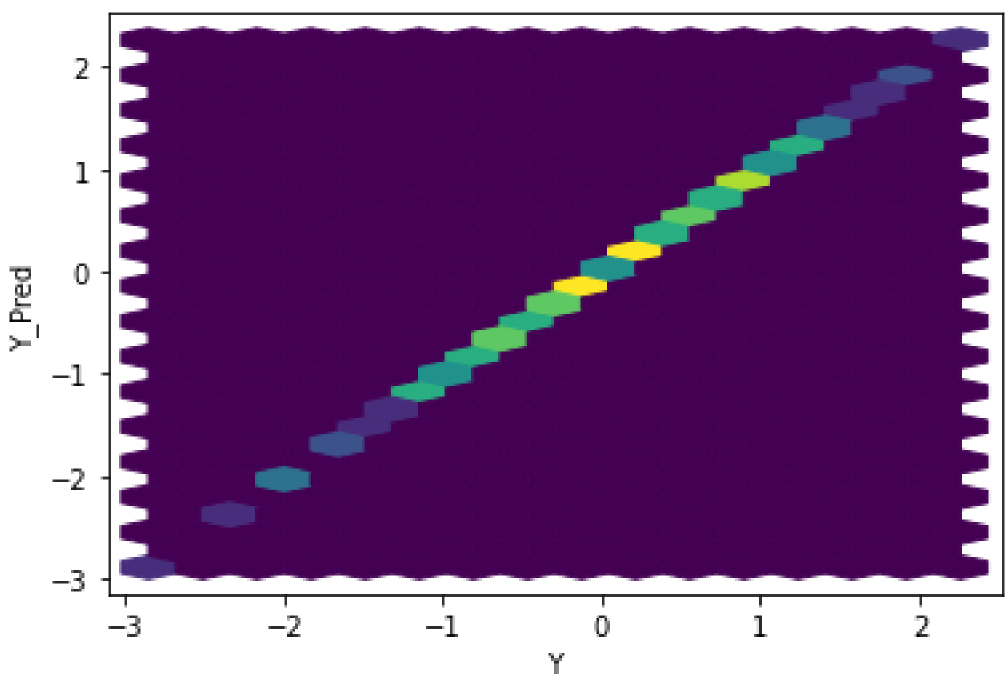
[16]:
```
n_epoch_abc = 2000
batch_size = sample_size//2
```

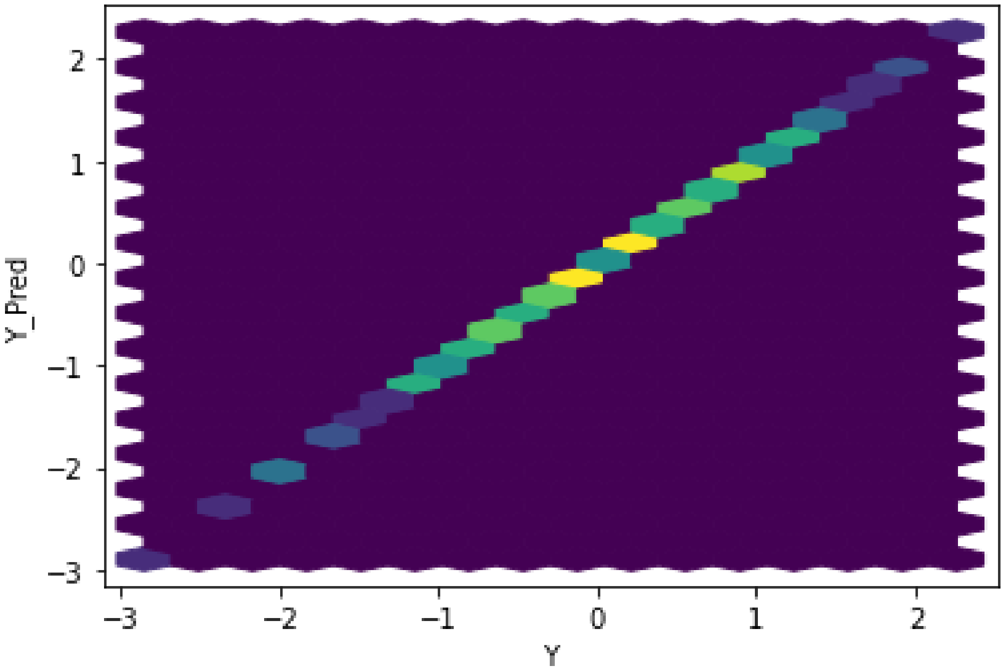
[17]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
 ↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```



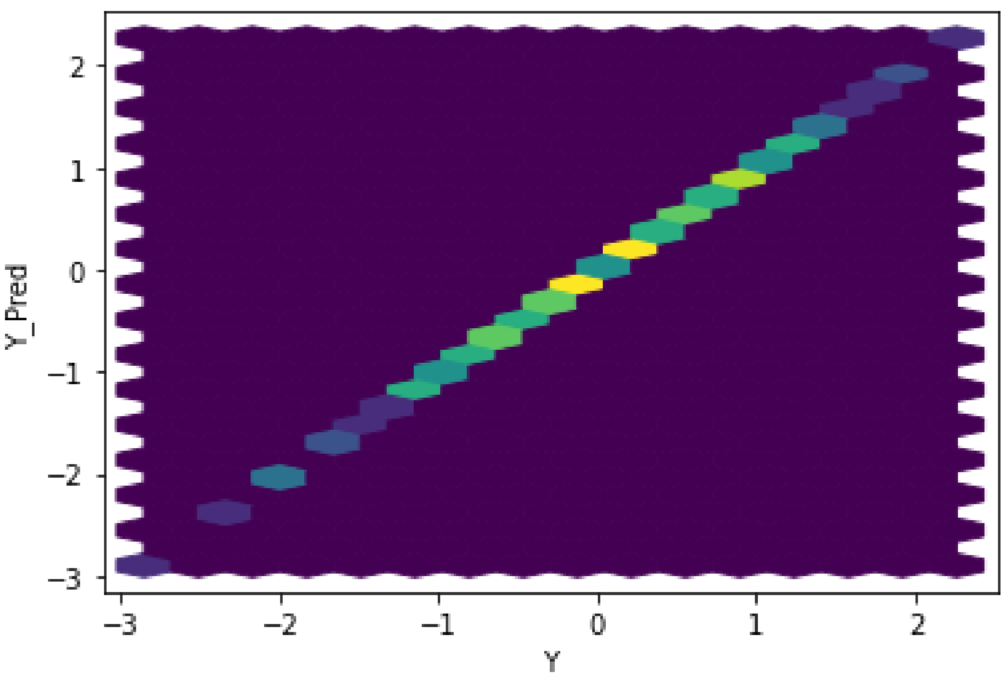
[18]:

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

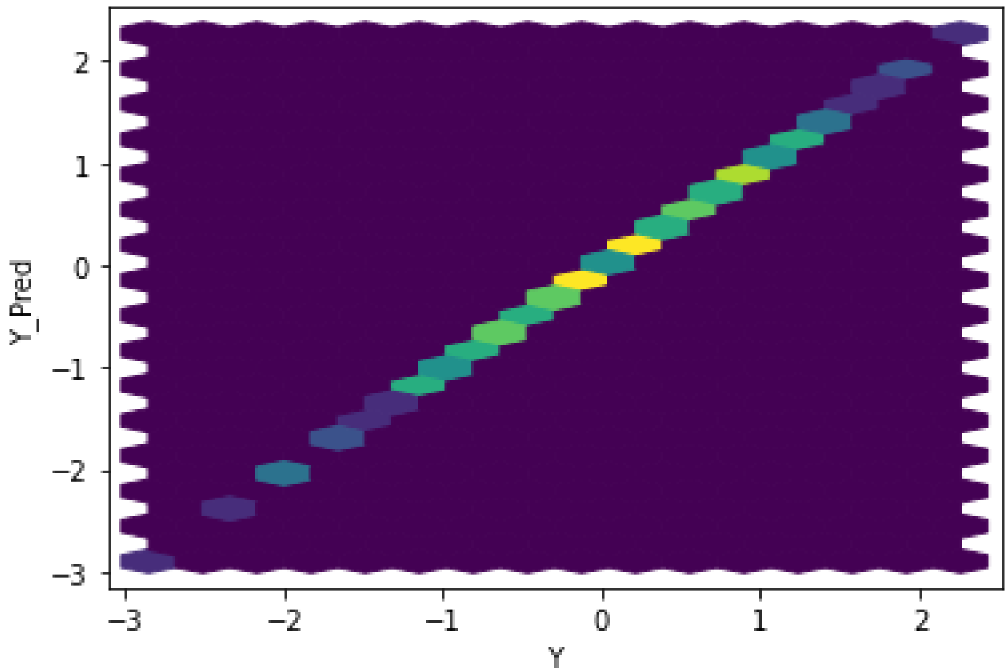


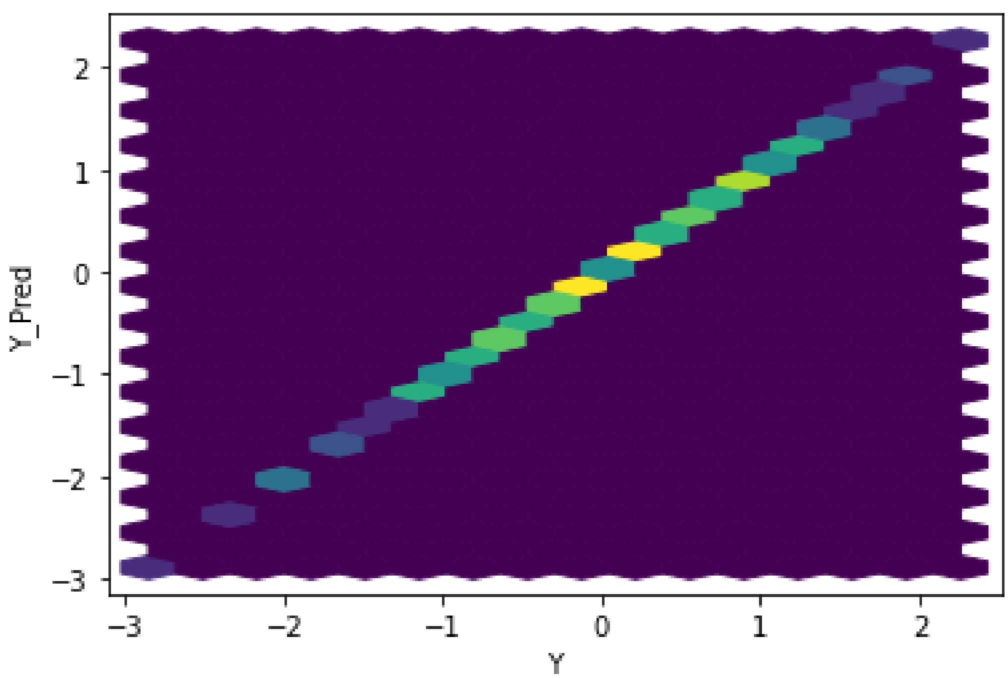


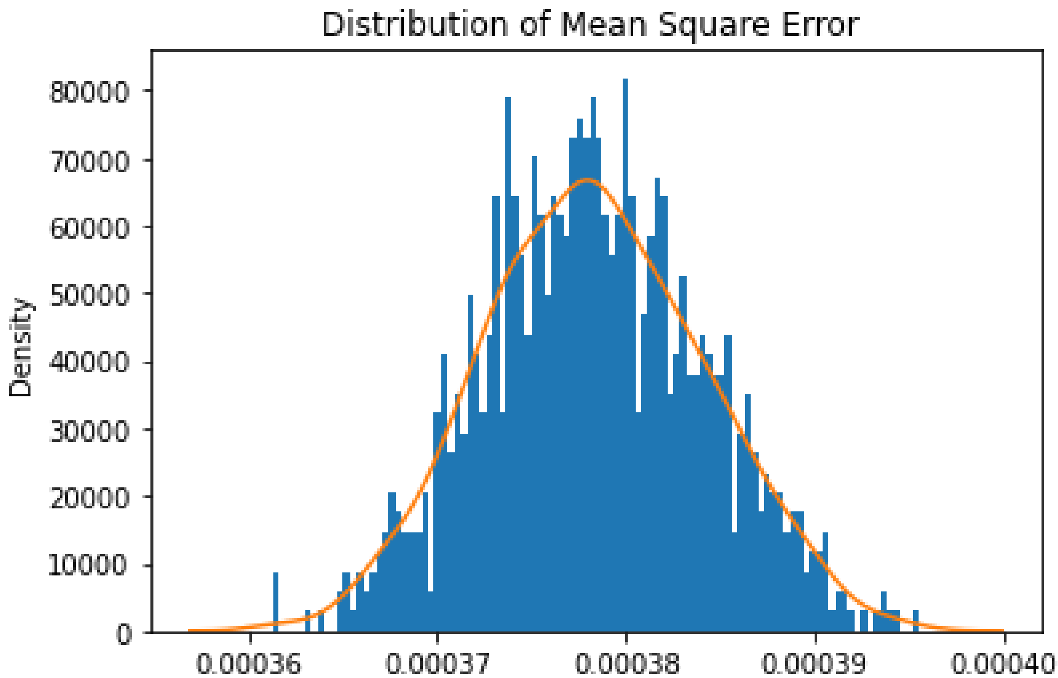
Y_Pred
Y



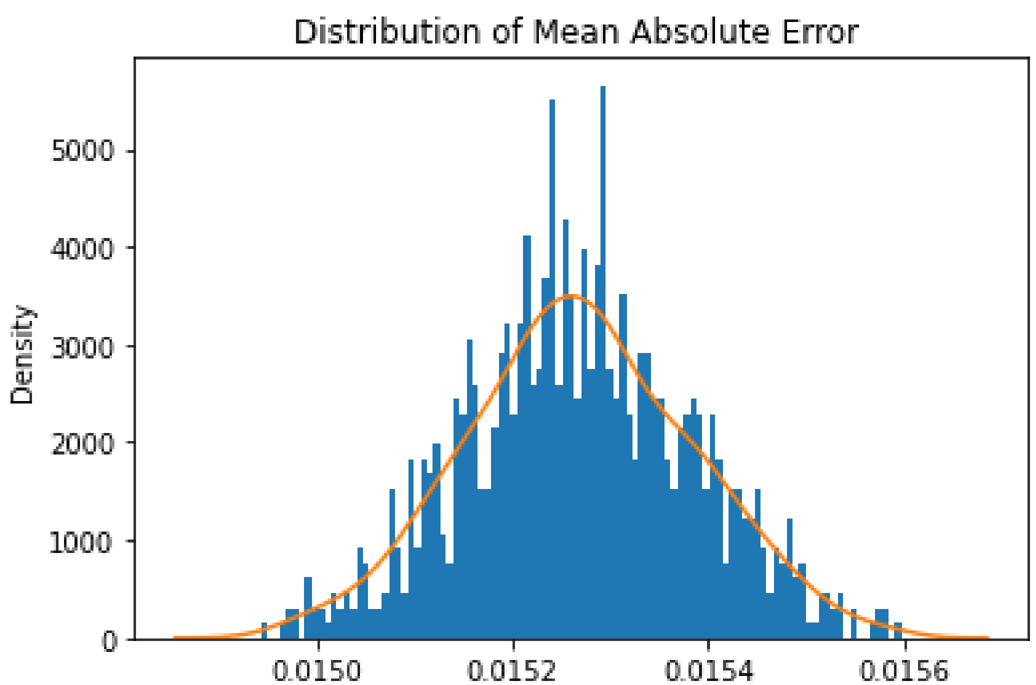
Y_Pred
Y


Y_Pred
Y

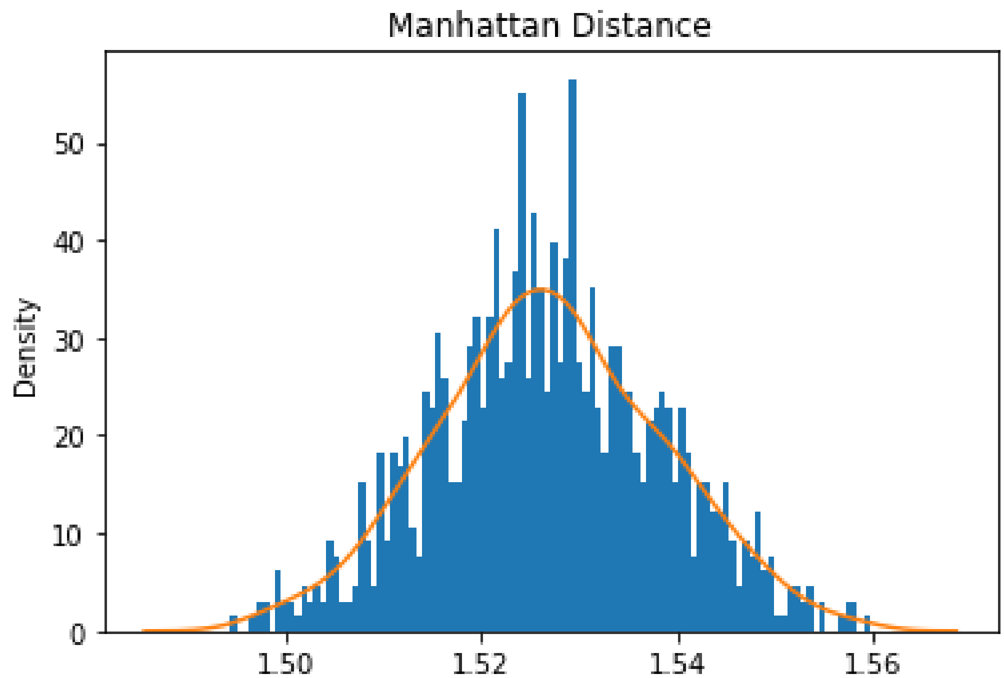

Y_Pred
Y



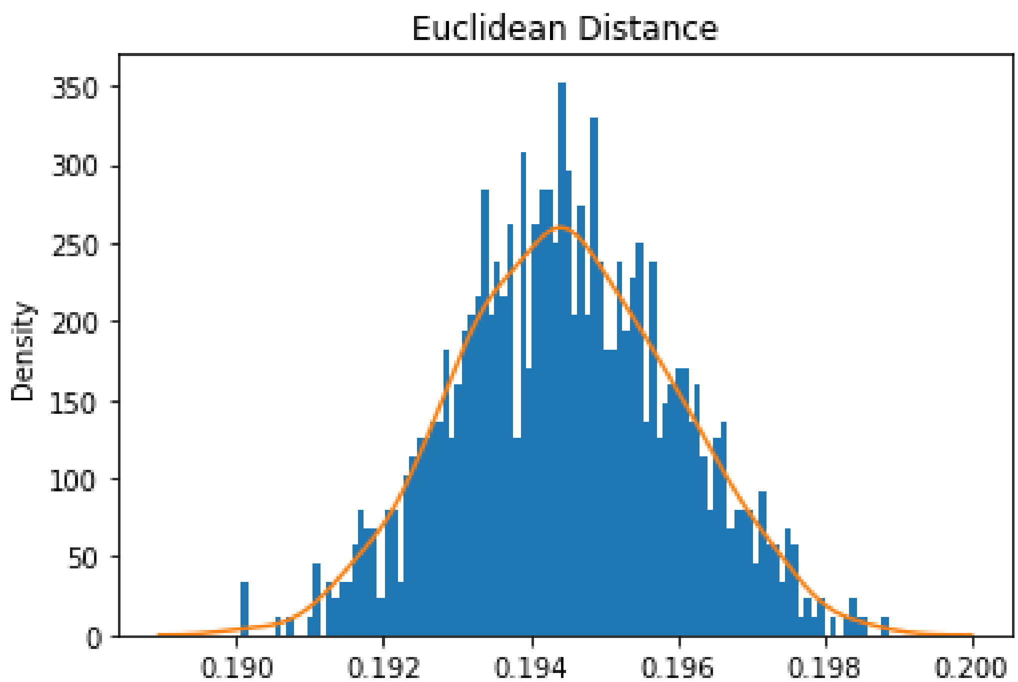

```
Mean Square Error: 0.0003782807381156427
```



```
Mean Absolute Error: 0.015268390294015407
Mean Manhattan Distance: 1.5268390294015408
```

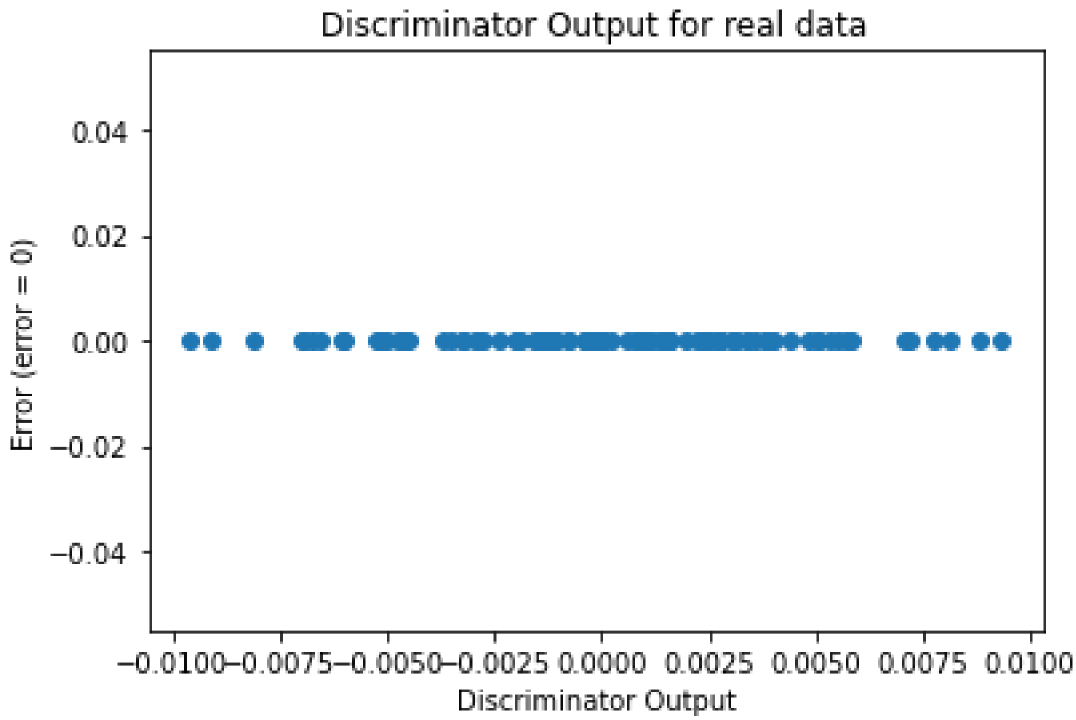


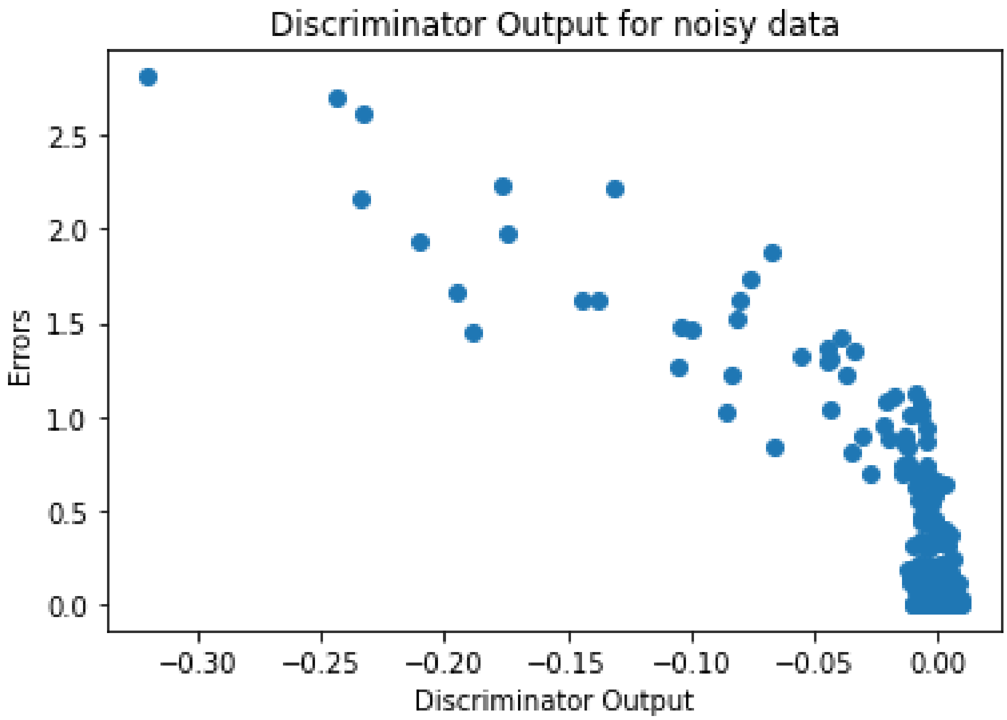

```
Mean Euclidean Distance: 0.19448867799436548
```



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([[0.1665, 0.3399, 0.3730, 0.1431, 0.4017, 0.1041, 0.1344, 0.1729, 0.2834,
         0.3194, 0.1829, 0.1448]], requires_grad=True)
output.bias Parameter containing:
tensor([-0.1759], requires_grad=True)
```