Dataset1-Regression_output_17

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  1.530492  0.485005 -0.047876  0.254231  0.789905 -1.757978  2.662890
1  0.559194  1.774084 -0.278990  2.022411 -0.920090  0.691142  1.276717
2  0.949423  0.533189 -1.001236  0.703650 -0.535060 -1.372876  0.503623
3 -0.293432 -0.429210 -1.184404 -0.430436  1.514147 -0.900779 -0.304132
4  0.901494 -0.093795  0.397311 -0.785109 -0.168663  1.353072 -0.164013

         X8        X9       X10           Y
0 -0.409881  0.640806  0.202630  393.919561
1  0.536149 -1.568208 -2.002866  285.995177
2  1.258819  1.140049 -0.726583   69.682112
3  0.657710  0.301500  0.242153  -91.525283
4  0.454494  1.076354 -0.757233   73.665988
```

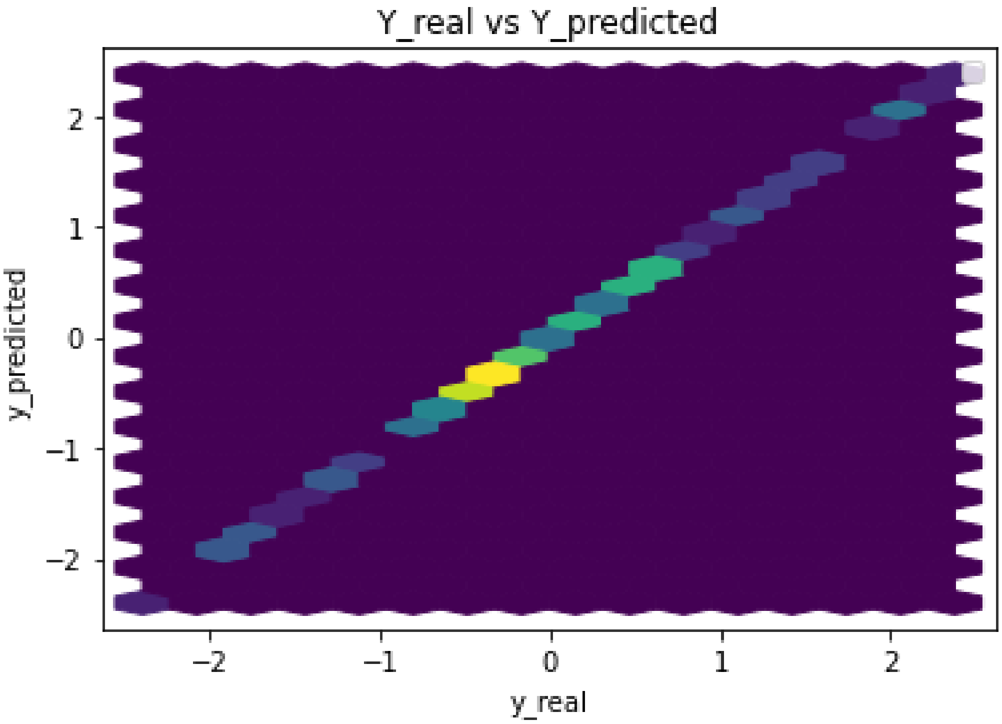
1.4 Stats Model

[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:                 5.957e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          3.68e-299
Time:                        07:48:33   Log-Likelihood:                 643.94
No. Observations:                 100   AIC:                            -1266.
Df Residuals:                      89   BIC:                            -1237.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.776e-17    4.1e-05   6.77e-13      1.000   -8.14e-05    8.14e-05
x1             0.2045    4.2e-05   4867.917      0.000       0.204       0.205
x2             0.4964   4.45e-05   1.12e+04      0.000       0.496       0.496
x3             0.4619   4.38e-05   1.06e+04      0.000       0.462       0.462
x4             0.2417   4.34e-05   5571.342      0.000       0.242       0.242
x5             0.3234   4.59e-05   7050.079      0.000       0.323       0.323
```

```
x6            0.0862   4.18e-05   2063.031      0.000       0.086       0.086
x7            0.4408   4.37e-05   1.01e+04      0.000       0.441       0.441
x8            0.1096   4.36e-05   2510.538      0.000       0.109       0.110
x9            0.1419   4.48e-05   3168.374      0.000       0.142       0.142
x10           0.0077   4.36e-05    177.564      0.000       0.008       0.008
==============================================================================
Omnibus:                        0.331   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.847   Jarque-Bera (JB):                0.080
Skew:                           0.047   Prob(JB):                        0.961
Kurtosis:                       3.103   Cond. No.                         1.79
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    2.775558e-17
x1       2.044628e-01
x2       4.963970e-01
x3       4.619462e-01
x4       2.417275e-01
x5       3.233609e-01
x6       8.622523e-02
x7       4.407733e-01
x8       1.095635e-01
x9       1.419295e-01
x10      7.746125e-03
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 1.4939766110410547e-07
Mean Absolute Error: 0.00030610289506025784
Manhattan distance: 0.030610289506025783
Euclidean distance: 0.003865199362311153
```

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]:

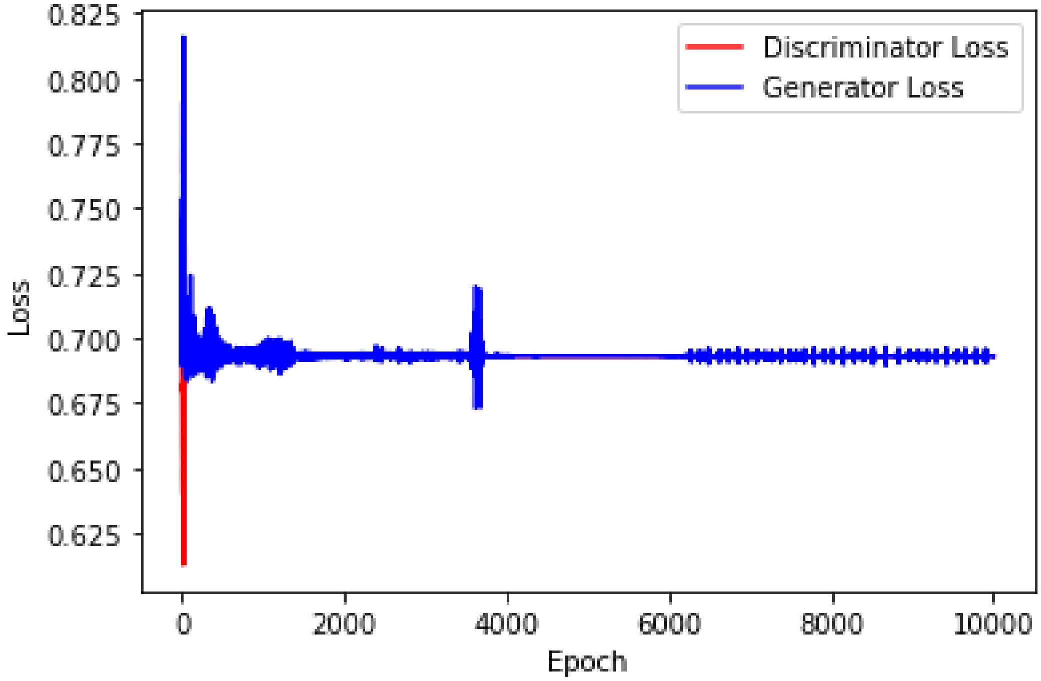
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:

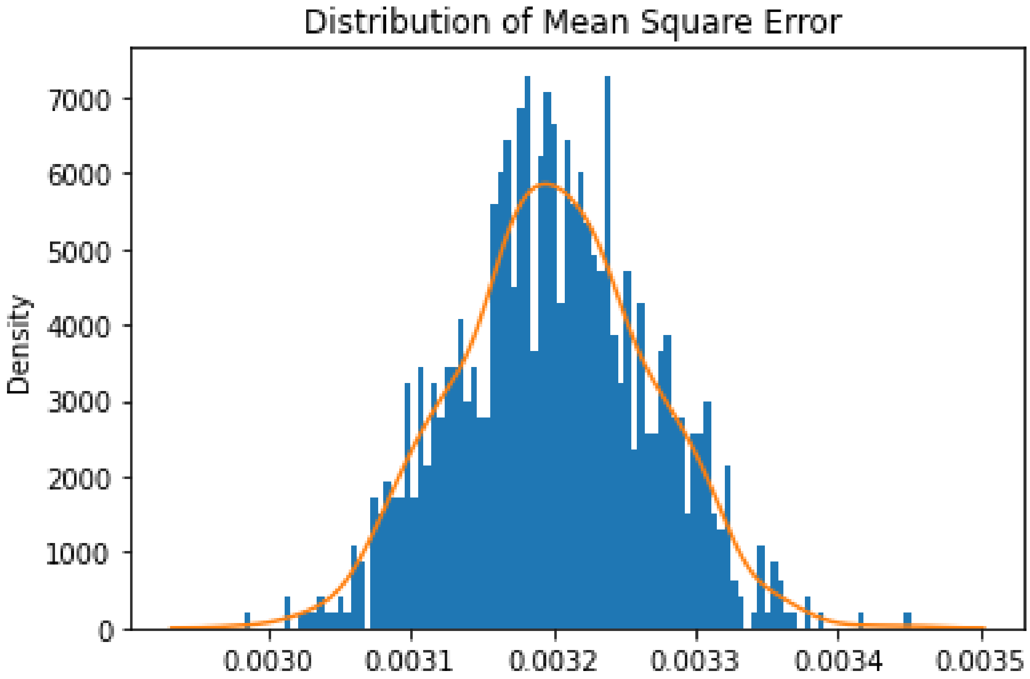
```
# Parameters
sample_size = 10000
std = 1
mean = 0.01
```

[13]:

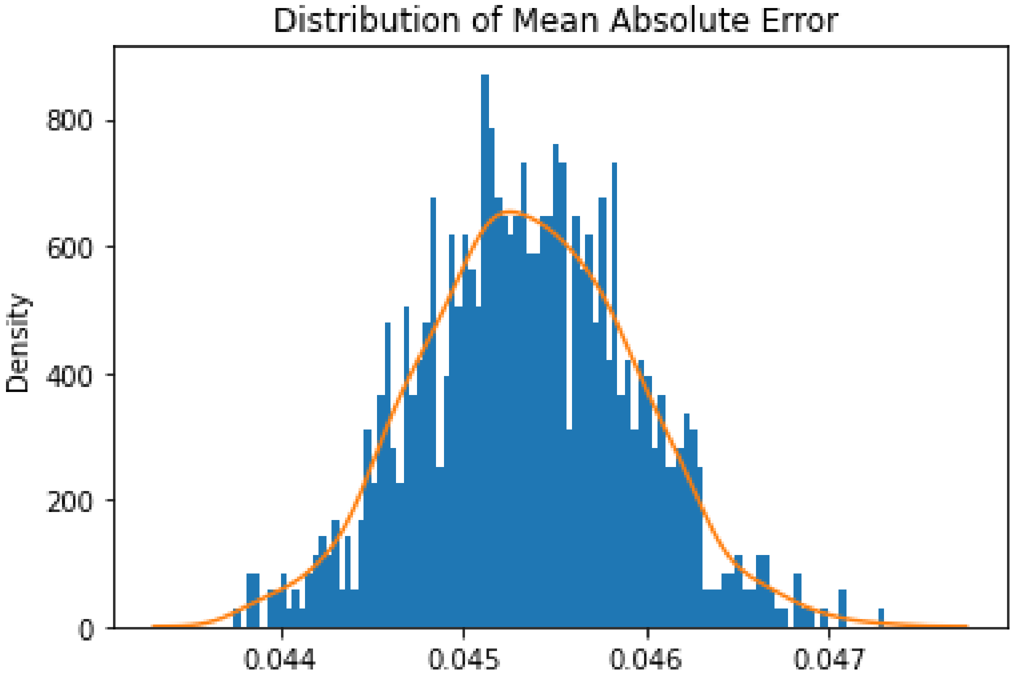
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



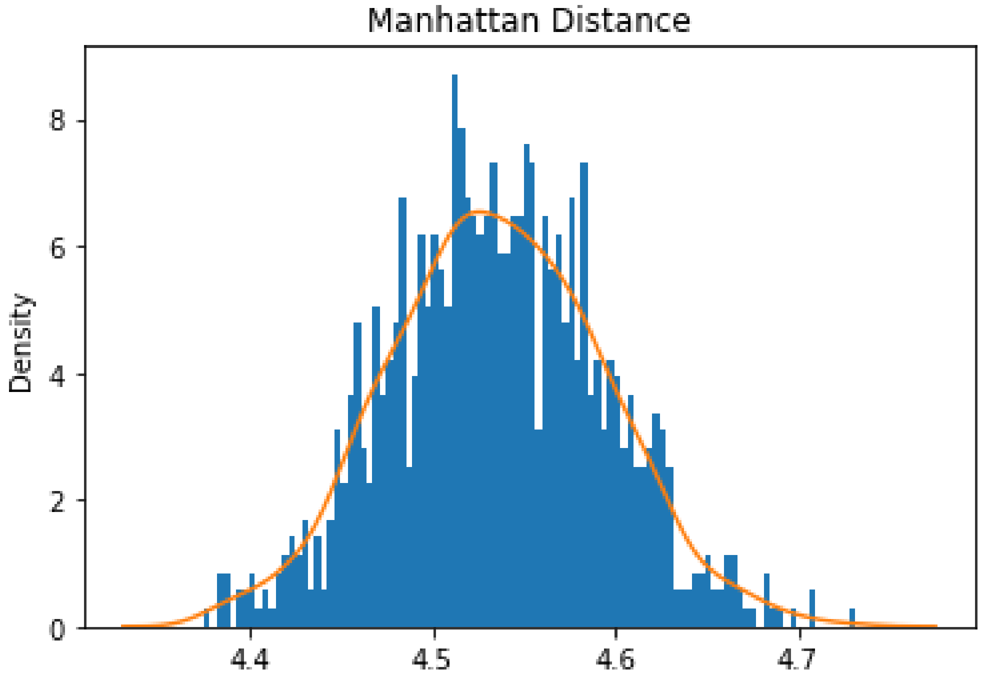
```
[14]: train_test.test_generator(generator,real_dataset,device)
```



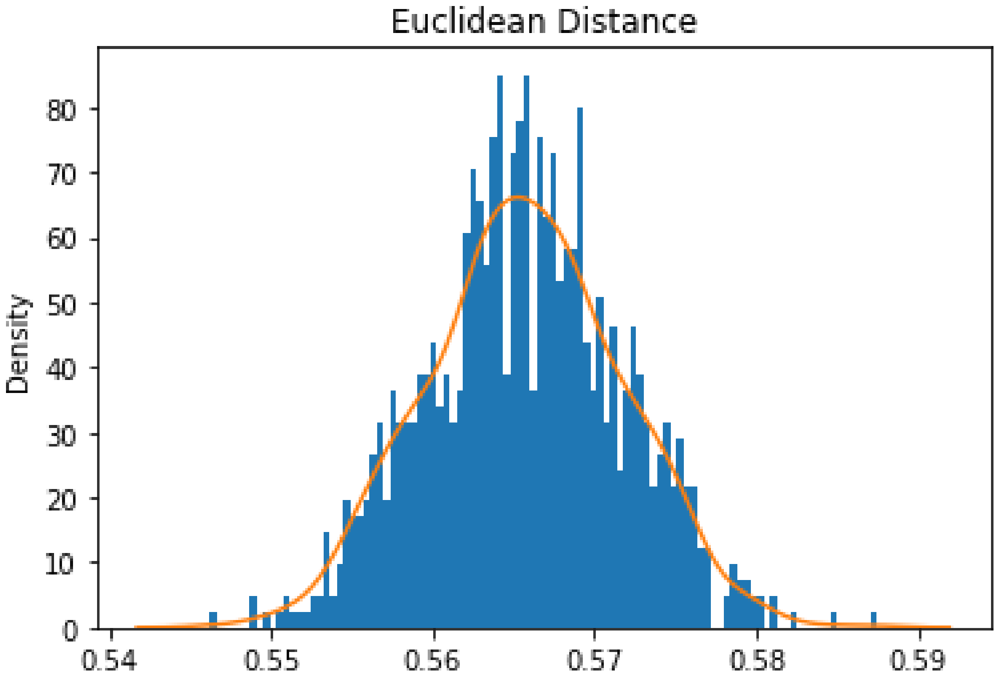
```
Mean Square Error: 0.003200260519351771
```


Distribution of Mean Absolute Error

```
Mean Absolute Error: 0.04535196477457881
```


Manhattan Distance

```
Mean Manhattan Distance: 4.535196477457881
```



```
Mean Euclidean Distance: 4.535196477457881
```

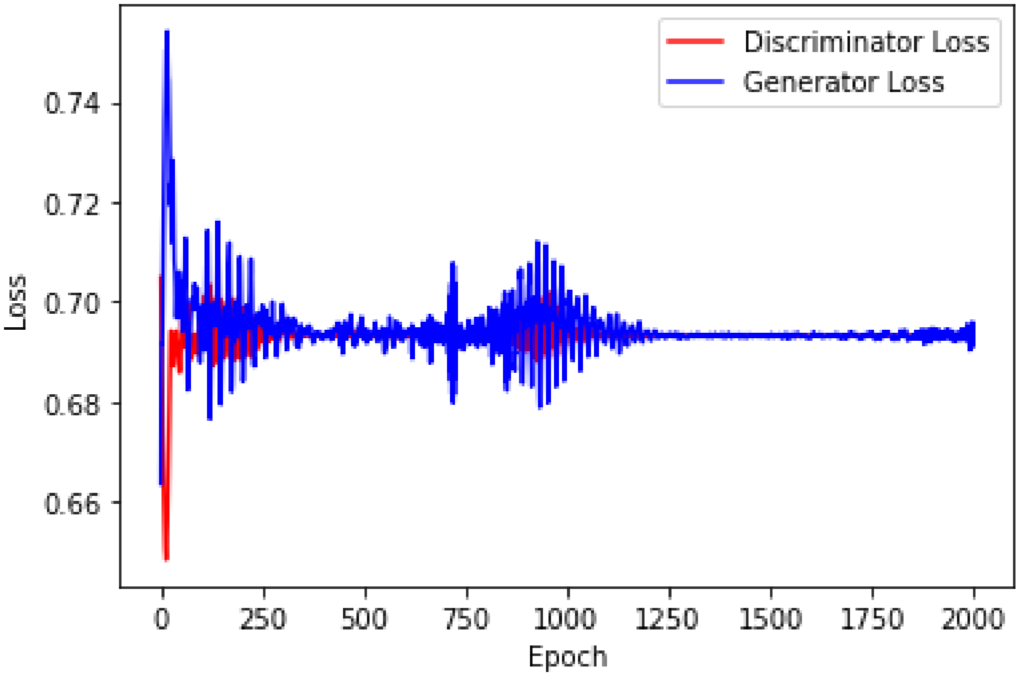
4 ABC GAN Model

Training the network

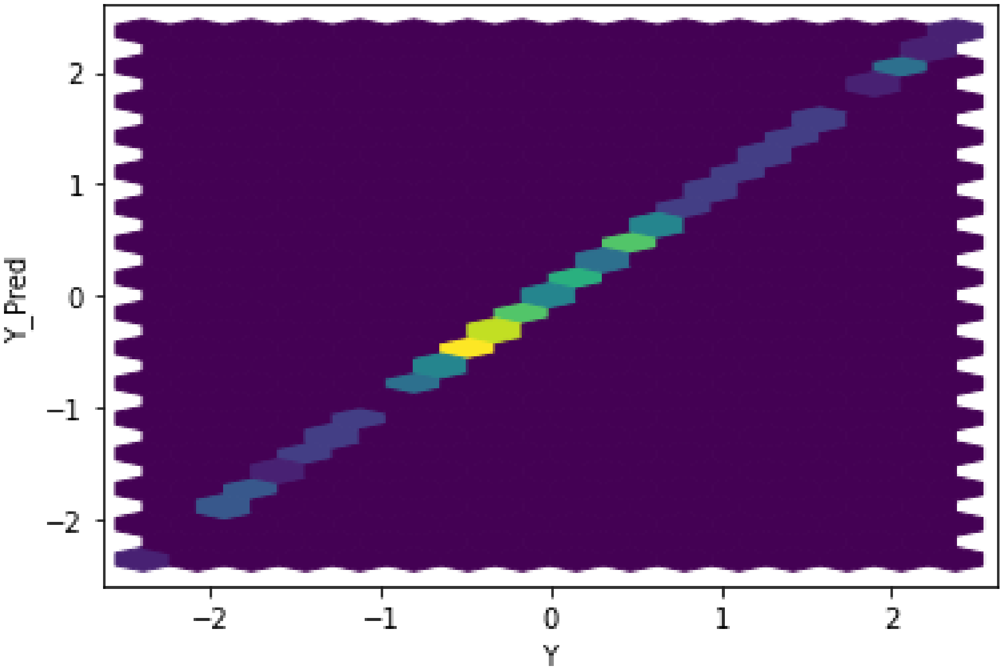
[15]:
```
gen = Generator(n_features+2)
disc = Discriminator(n_features+2,hidden_nodes)

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))
```

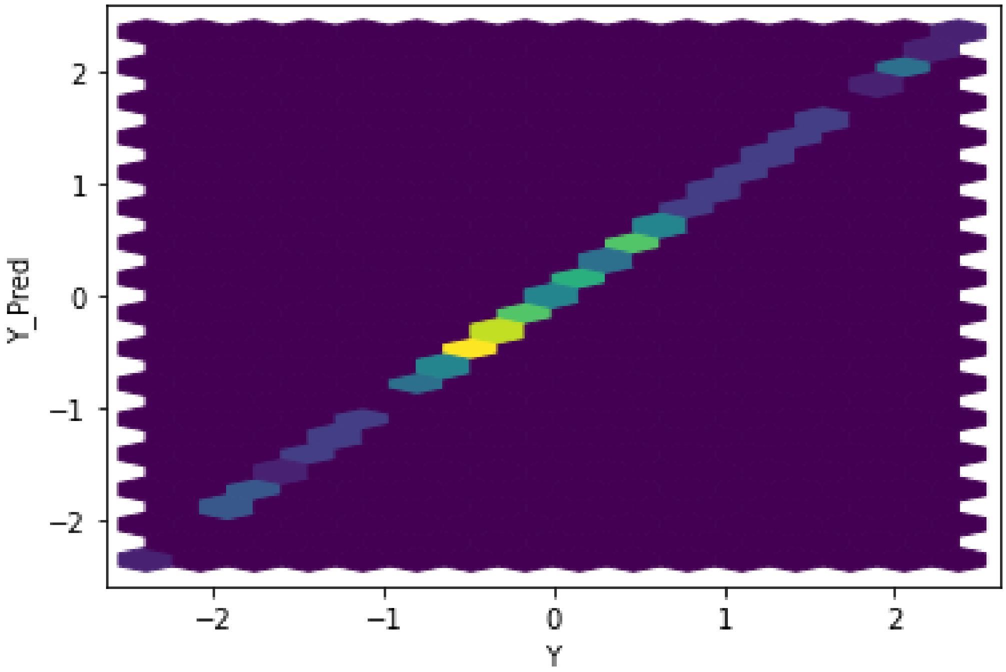
[16]:
```
n_epoch_abc = 2000
batch_size = sample_size//2
```

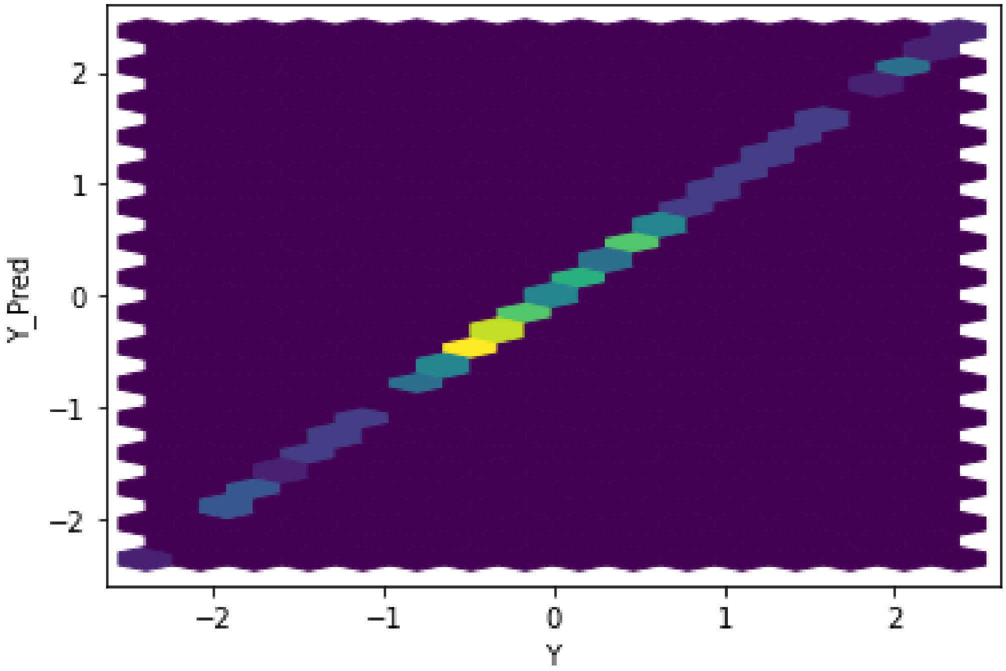
[17]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
 ↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

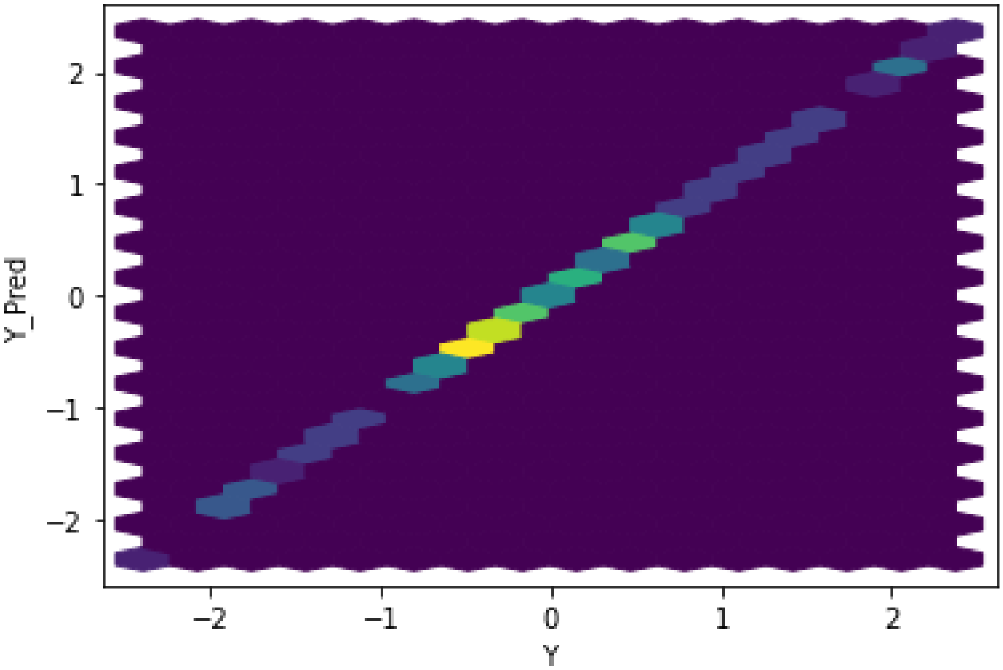


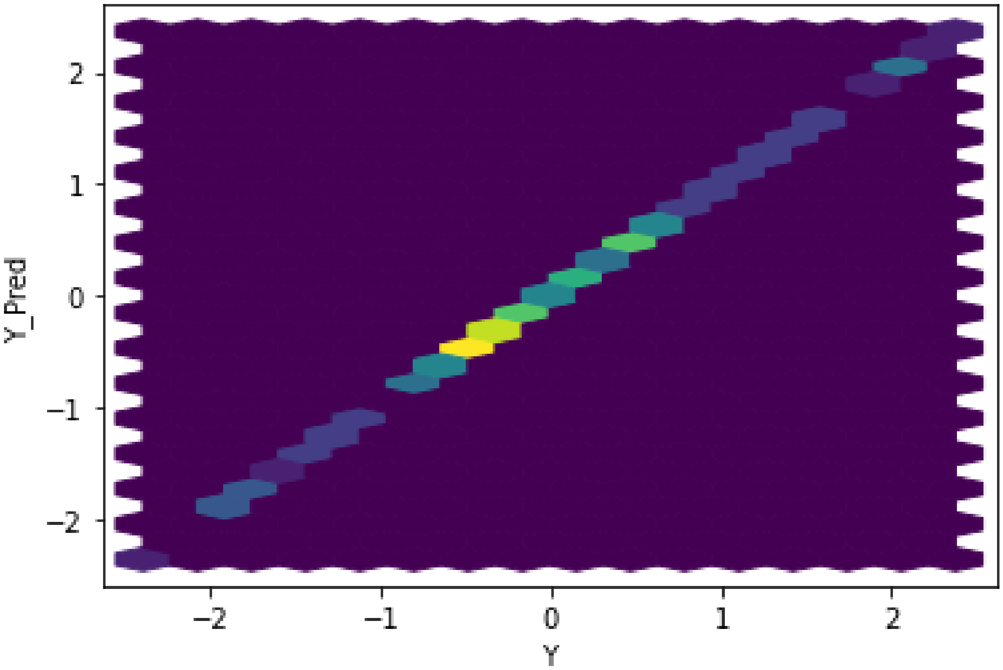
```
[18]: ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

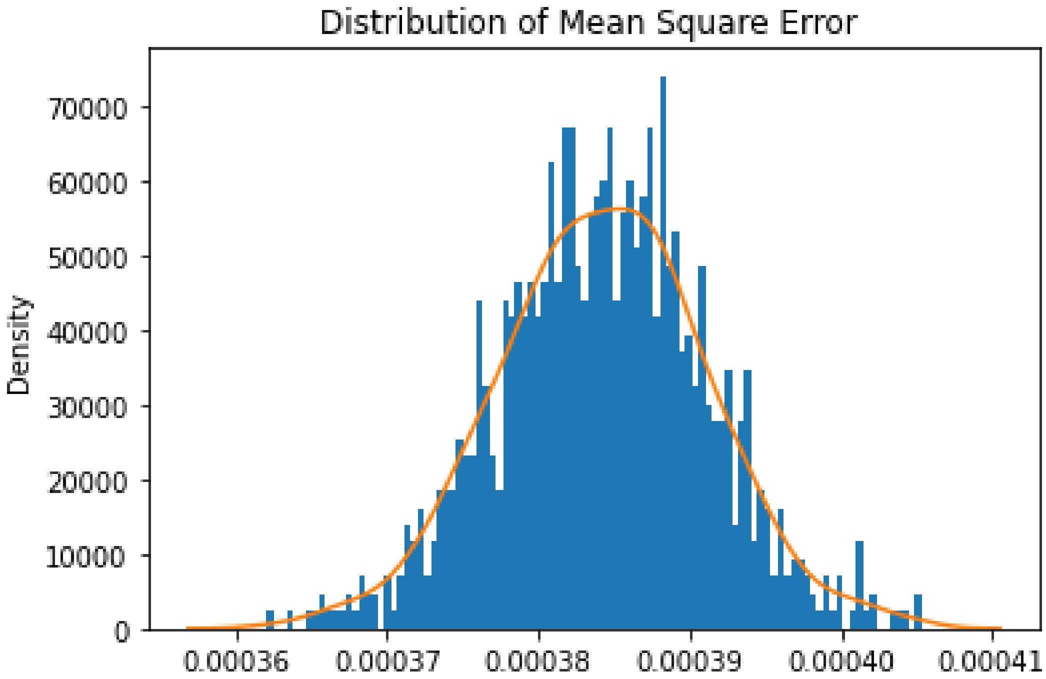



2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
Y

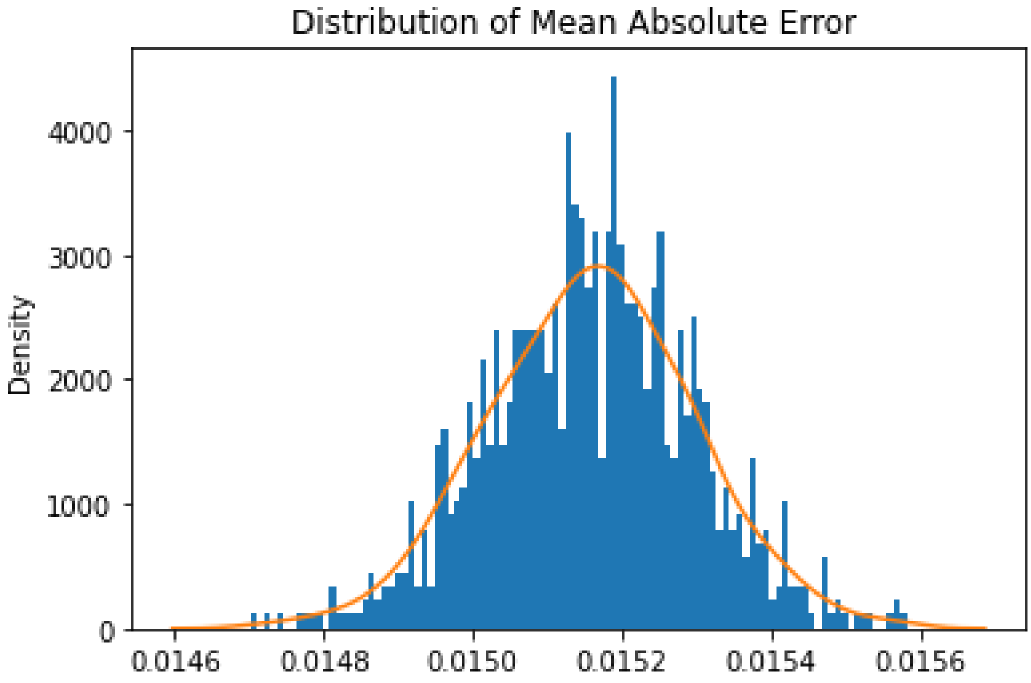

2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
Y


Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y

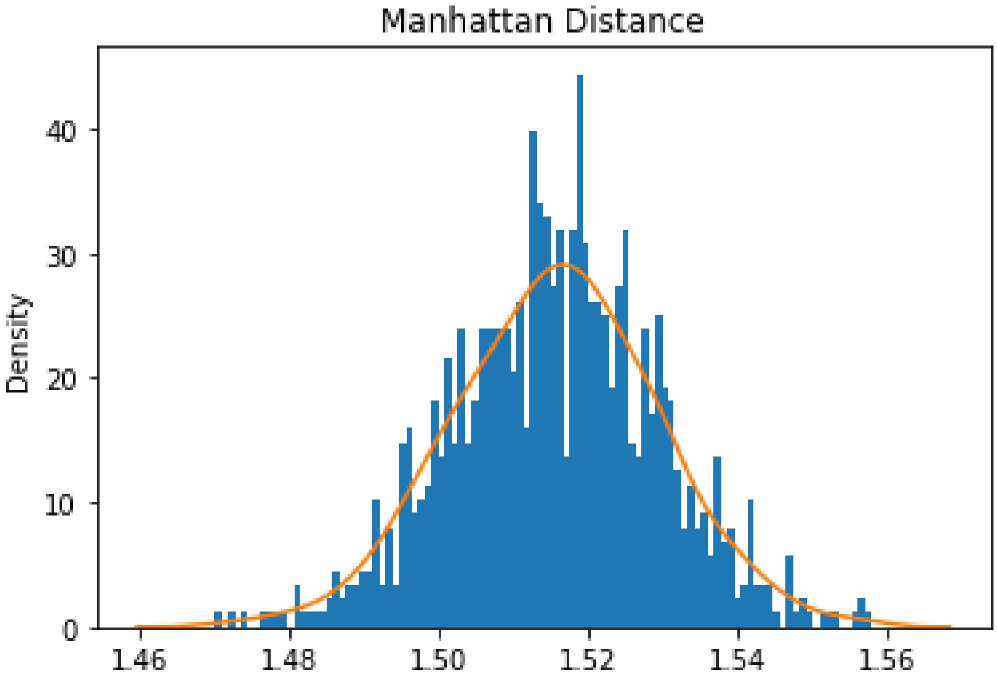

Y_Pred
2
1
0
−1
−2
−2
−1
0
1
2
Y



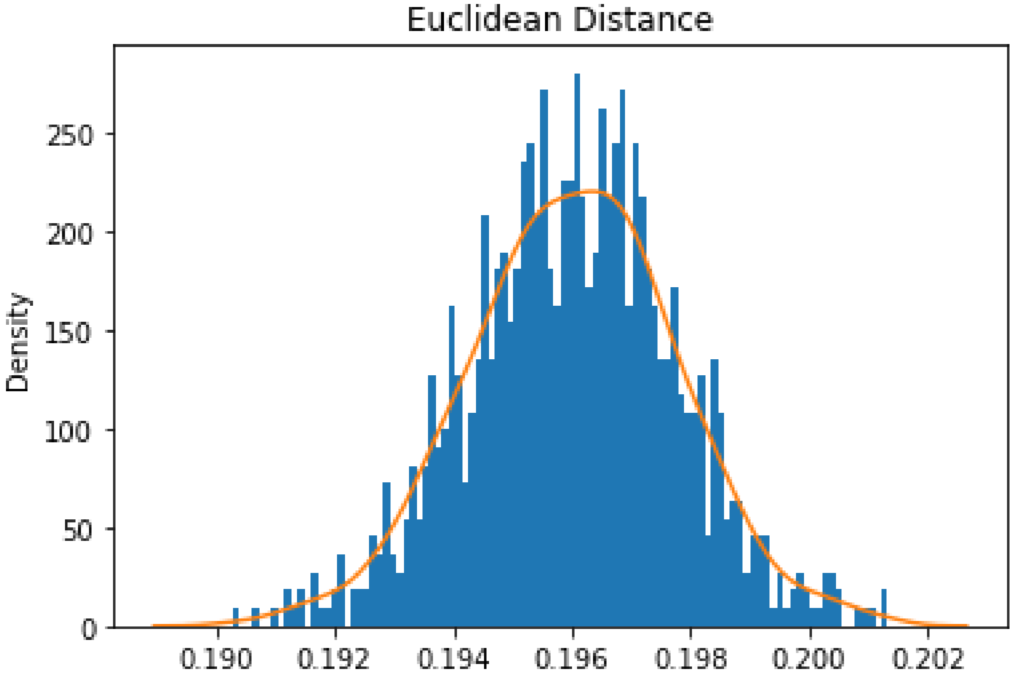
```
Mean Square Error: 0.0003842822180457364
```



```
Mean Absolute Error: 0.015157858716845513
Mean Manhattan Distance: 1.5157858716845511
```

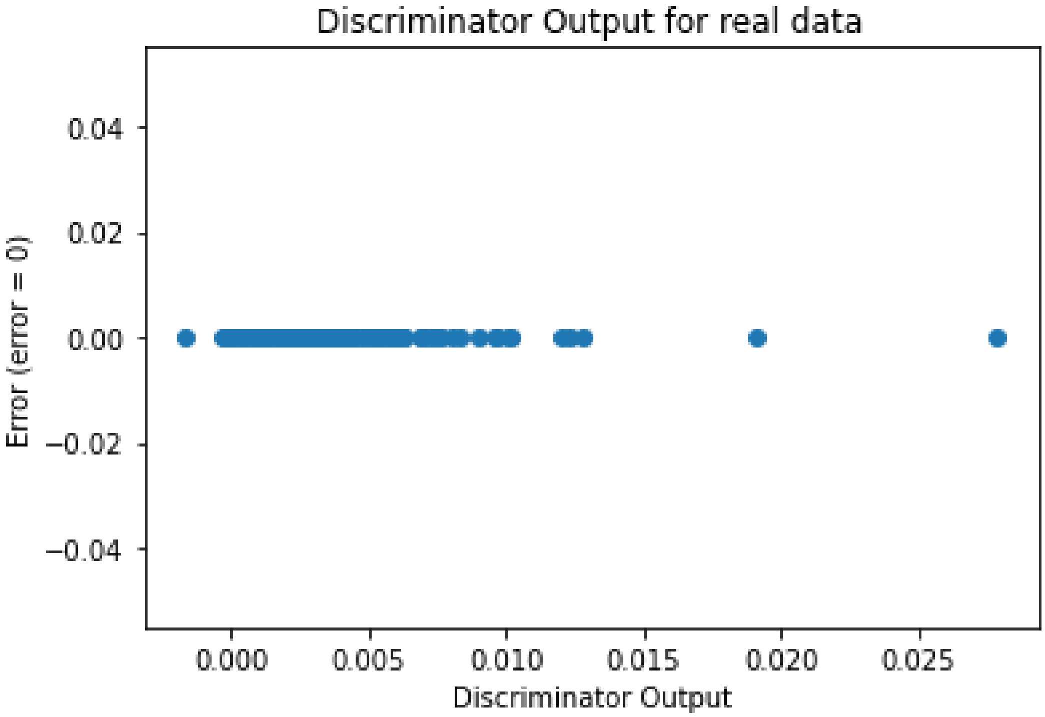


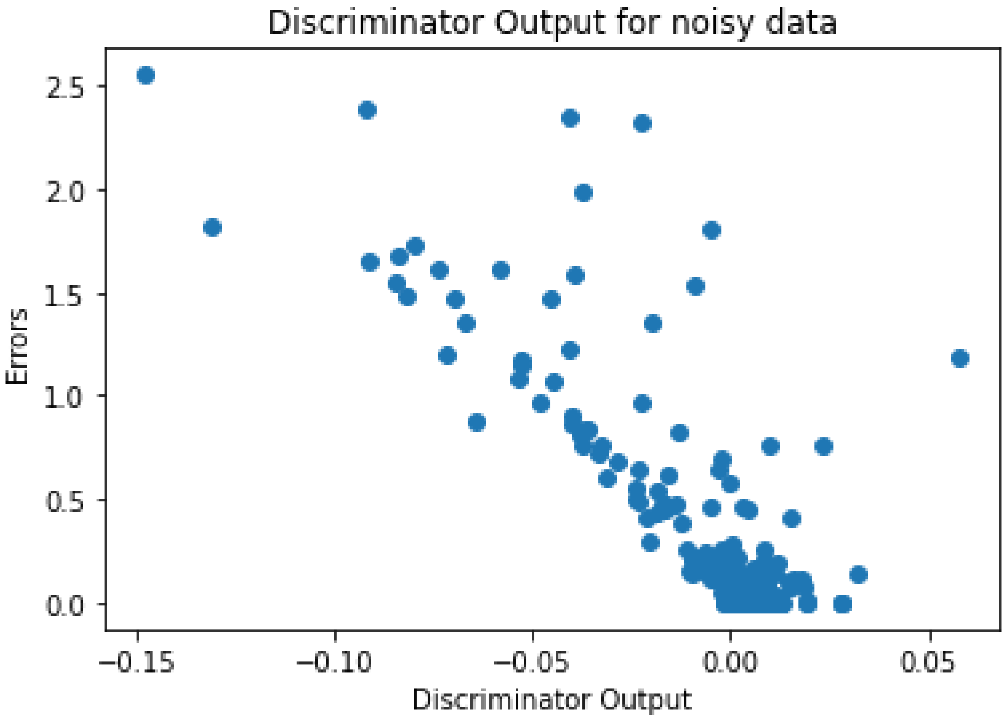
```
Mean Euclidean Distance: 0.19602336174787408
```



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([[0.1007, 0.1602, 0.3927, 0.3609, 0.1923, 0.2634, 0.0759, 0.3649, 0.1004,
         0.1134, 0.0075, 0.1926]], requires_grad=True)
output.bias Parameter containing:
tensor([-0.1011], requires_grad=True)
```