Dataset1-Regression_output_5

October 7, 2021

1 Dataset 1 - Regression

1.1 Import Libraries

[1]:

```python
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]:

```python
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 -1.379235  0.164042  0.252170  1.191248  0.439649  0.534402  0.153015
1 -0.034874  0.942978 -0.882196  0.526697  0.288316 -0.713429  0.430059
2  0.833126  0.302916 -0.691061 -0.057101  0.746433  1.672962  0.153606
3  1.198740 -0.031483 -0.573357 -0.038852  1.928704  0.628222  0.245201
4 -0.944018 -0.027398 -0.411795 -0.153953 -0.365626 -0.382030 -1.388453

         X8        X9       X10           Y
0  0.005361 -2.624366 -0.067882 -211.197534
1 -0.488881  1.277887  0.483764  128.362442
2  1.857432  0.438774 -0.165465  246.860787
3  0.687796 -0.002494  0.886734  147.985163
4 -0.481801 -0.836031  1.709714 -202.592183
```

1.4 Stats Model

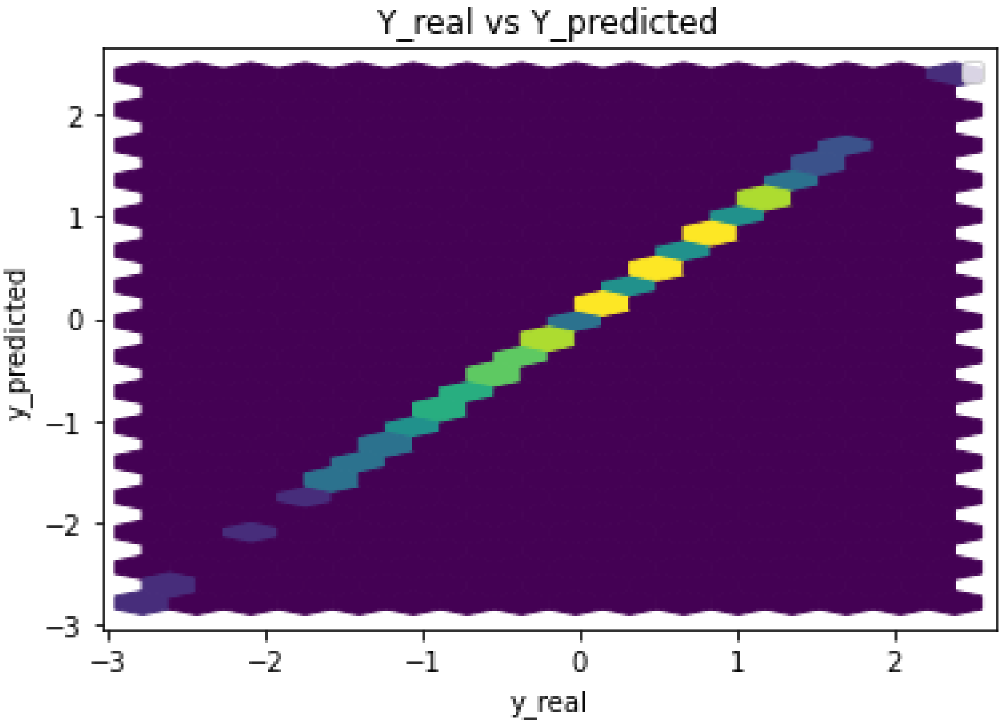
```
[4]: [coeff,y_pred] = statsModel.statsModel(X,Y)
```

```
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:                 2.021e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          2.88e-278
Time:                        19:03:12   Log-Likelihood:                 589.88
No. Observations:                 100   AIC:                            -1158.
Df Residuals:                      89   BIC:                            -1129.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.388e-17   7.03e-05   1.97e-13      1.000      -0.000       0.000
x1             0.1601   7.44e-05   2150.616      0.000       0.160       0.160
x2             0.1120   7.25e-05   1545.544      0.000       0.112       0.112
x3             0.0515   7.26e-05    708.618      0.000       0.051       0.052
x4             0.1519   7.55e-05   2011.113      0.000       0.152       0.152
x5             0.1151   7.47e-05   1541.746      0.000       0.115       0.115
```

```
x6             0.1326   7.32e-05   1813.260      0.000       0.133       0.133
x7             0.4114   7.29e-05   5641.056      0.000       0.411       0.412
x8             0.5671   7.71e-05   7359.982      0.000       0.567       0.567
x9             0.6200   7.29e-05   8508.223      0.000       0.620       0.620
x10            0.1398   7.29e-05   1917.717      0.000       0.140       0.140
==============================================================================
Omnibus:                        7.700   Durbin-Watson:                   2.295
Prob(Omnibus):                  0.021   Jarque-Bera (JB):                3.132
Skew:                           0.053   Prob(JB):                        0.209
Kurtosis:                       2.139   Cond. No.                         1.66
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    1.387779e-17
x1       1.601122e-01
x2       1.120444e-01
x3       5.147151e-02
x4       1.519071e-01
x5       1.151072e-01
x6       1.326500e-01
x7       4.113806e-01
x8       5.671167e-01
x9       6.199766e-01
x10      1.398351e-01
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 4.404163188438002e-07
Mean Absolute Error: 0.000556317642282813
Manhattan distance: 0.05563176422828129
Euclidean distance: 0.006636386960114669
```

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]:

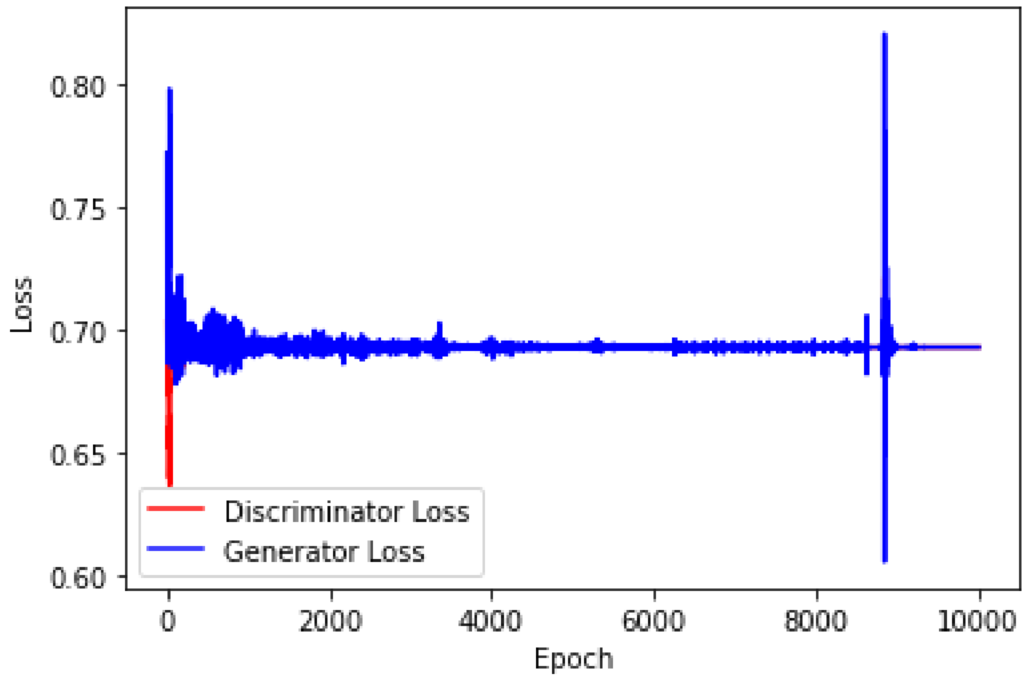
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:

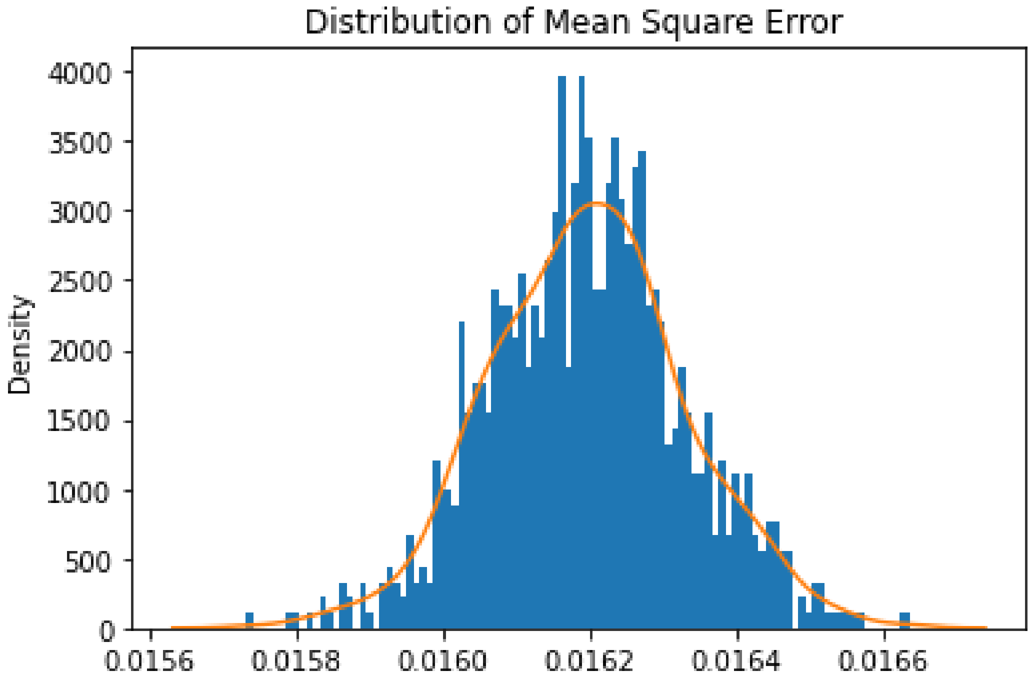
```
# Parameters
sample_size = 10000
mean = 1
std = 0.1
```

[13]:

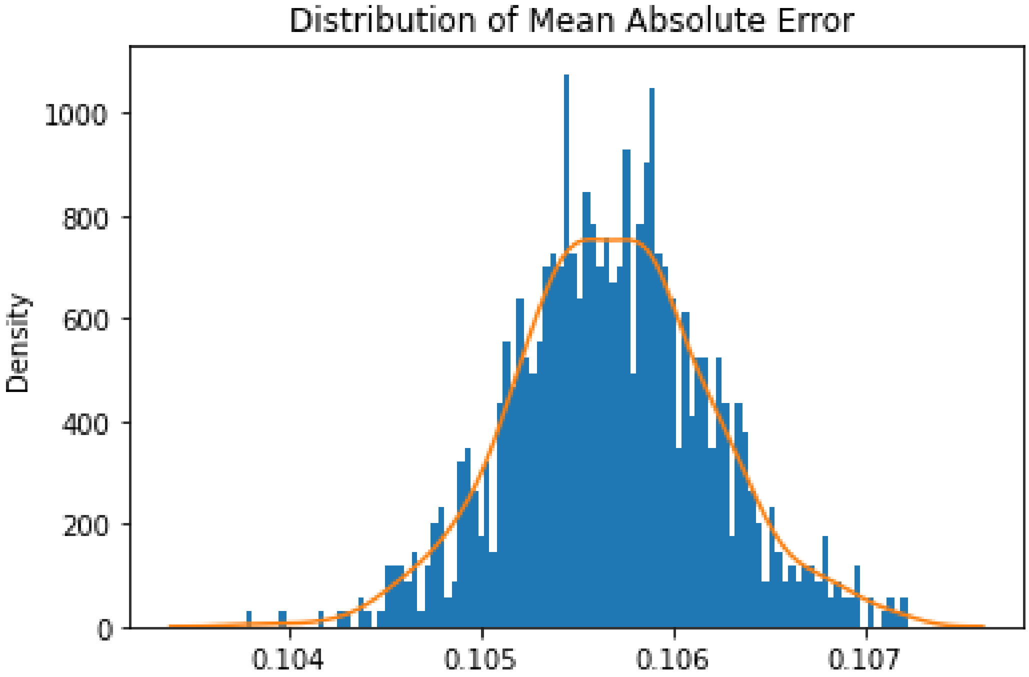
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



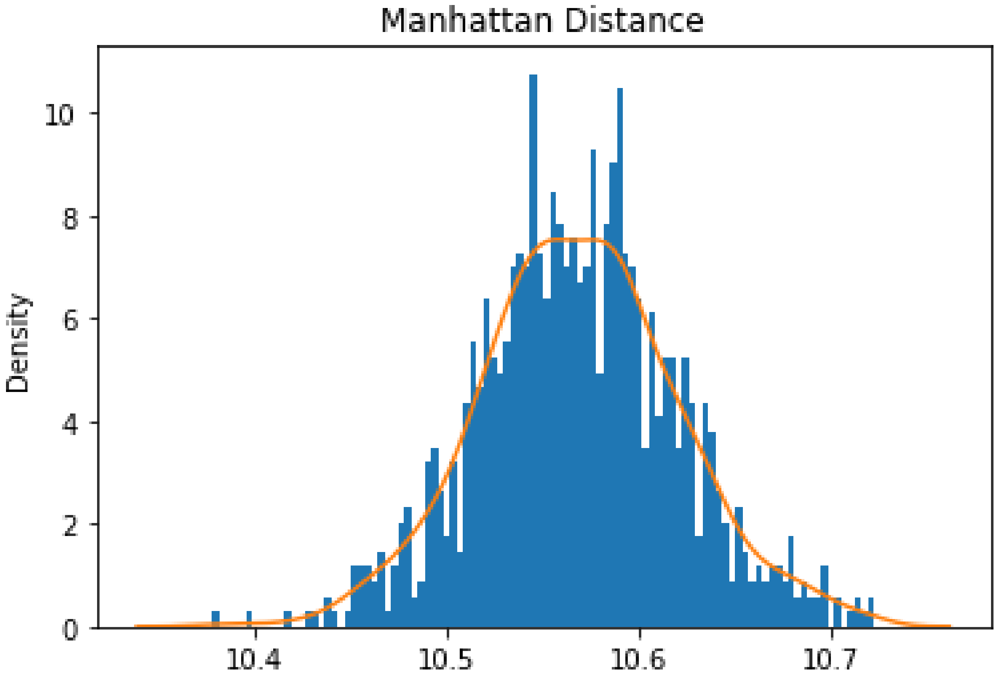
[14]:
```
train_test.test_generator(generator,real_dataset,device)
```



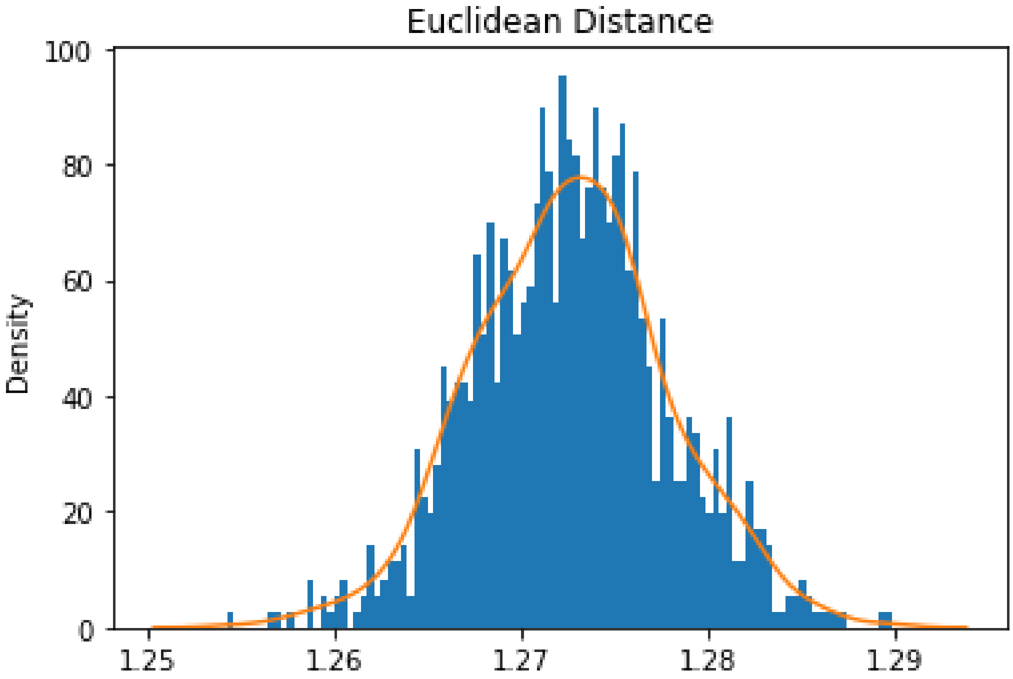
```
Mean Square Error: 0.016197146311950582
```



```
Mean Absolute Error: 0.10568452871777118
```



```
Mean Manhattan Distance: 10.568452871777117
```



```
Mean Euclidean Distance: 10.568452871777117
```

4 ABC GAN Model

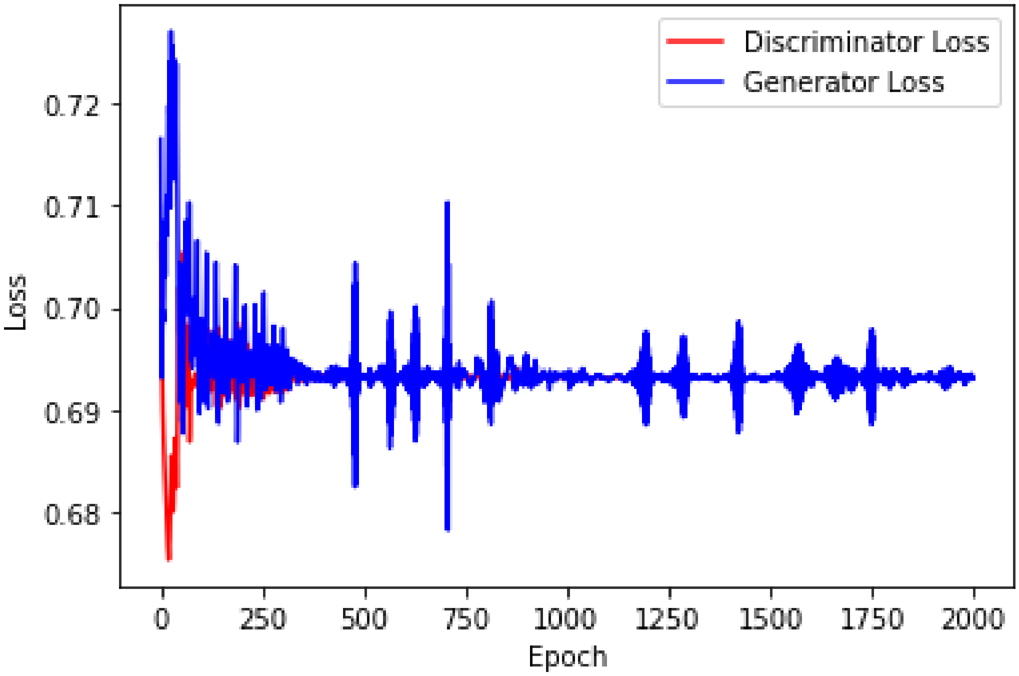
Training the network

```
[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))
```

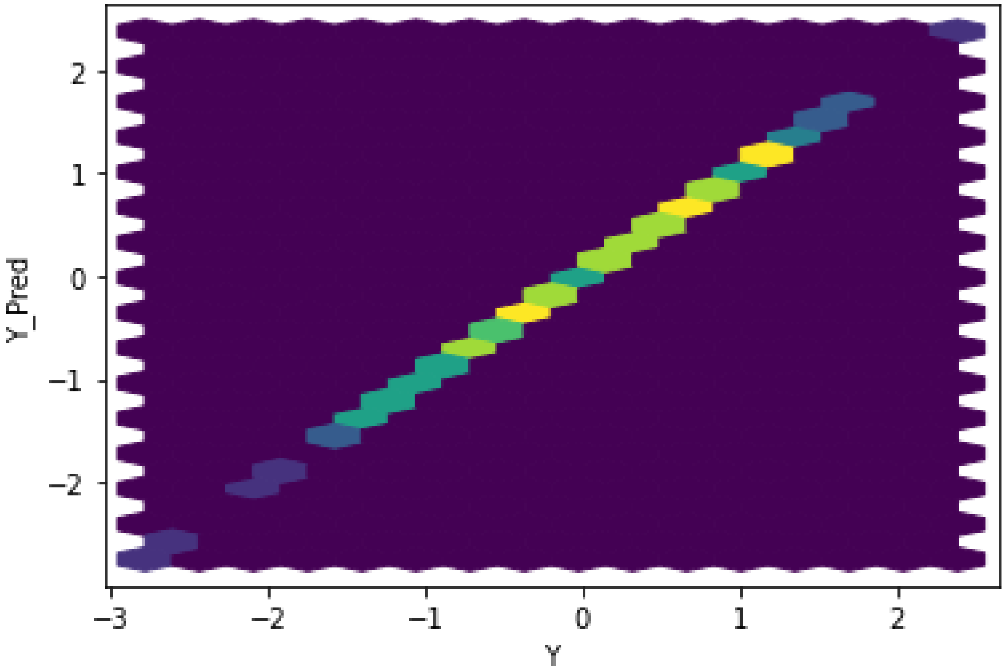
```
[16]: n_epoch_abc = 2000
      batch_size = sample_size//2
```

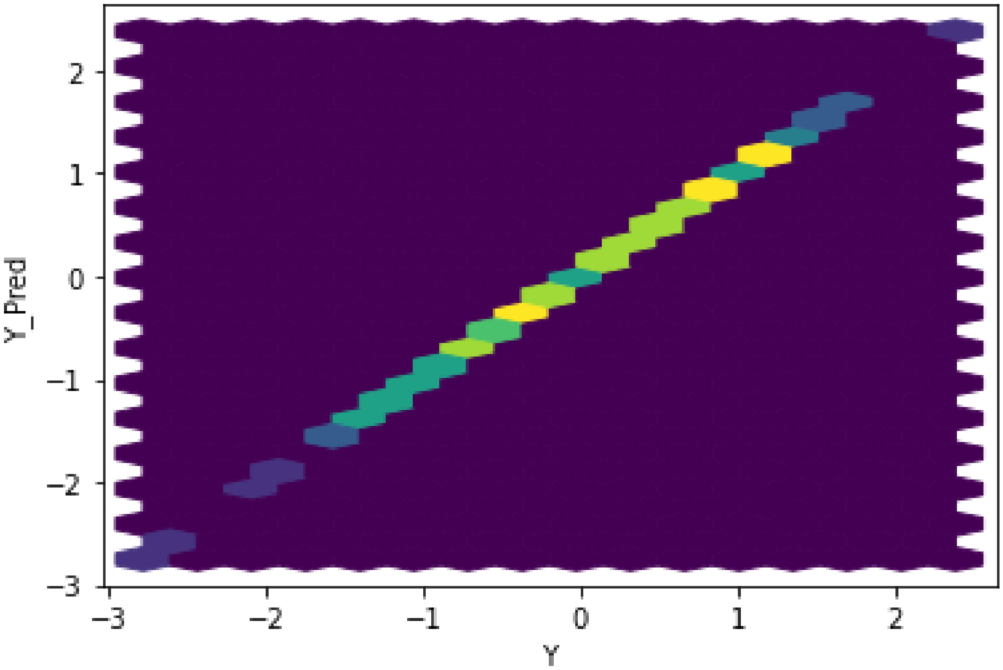
```
[17]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
       ↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

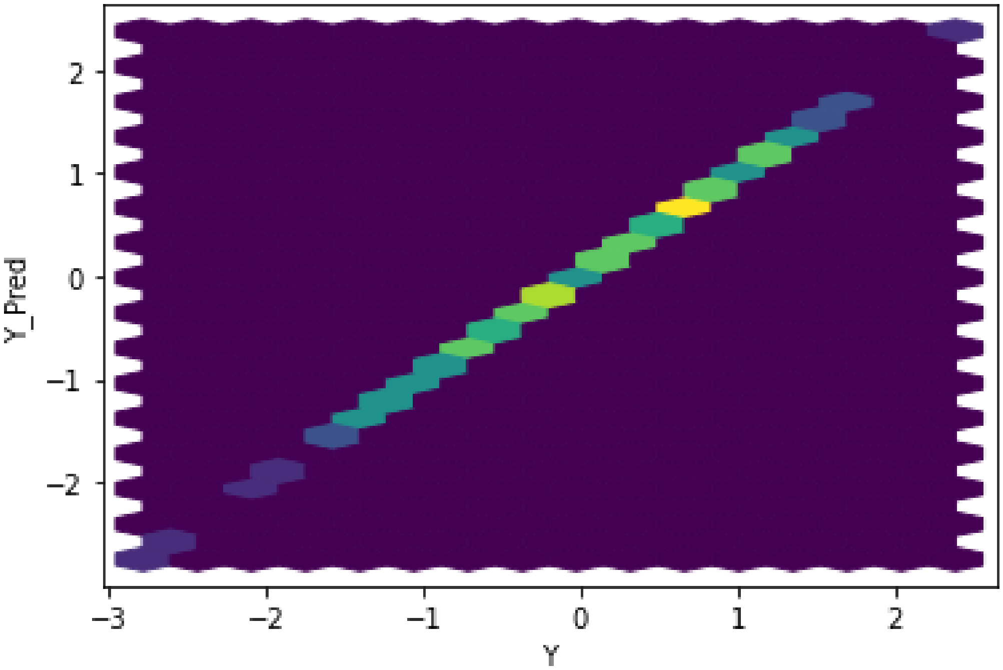


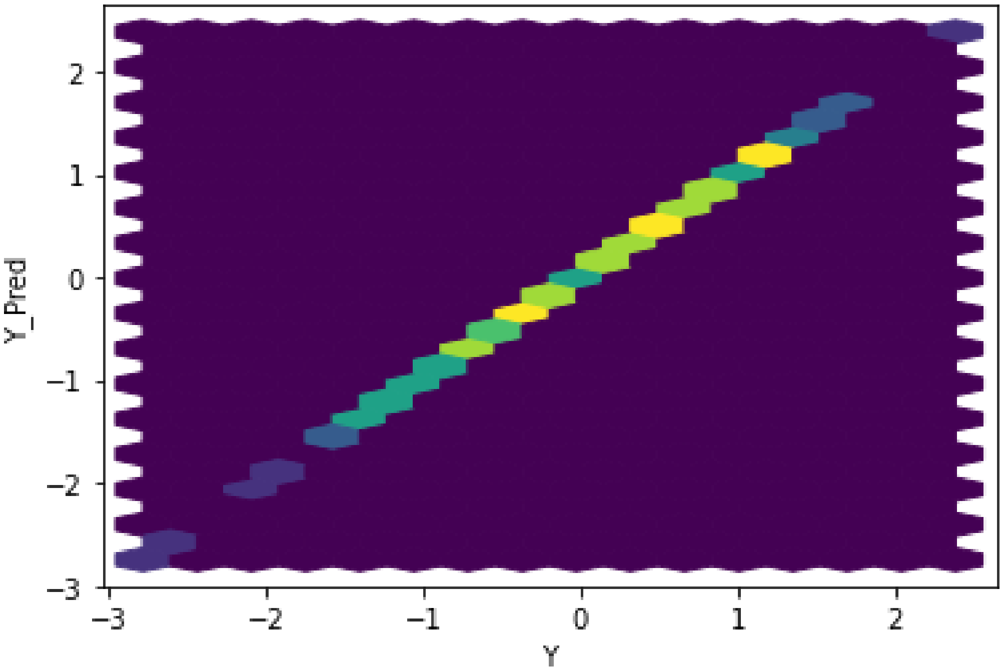
[18]:

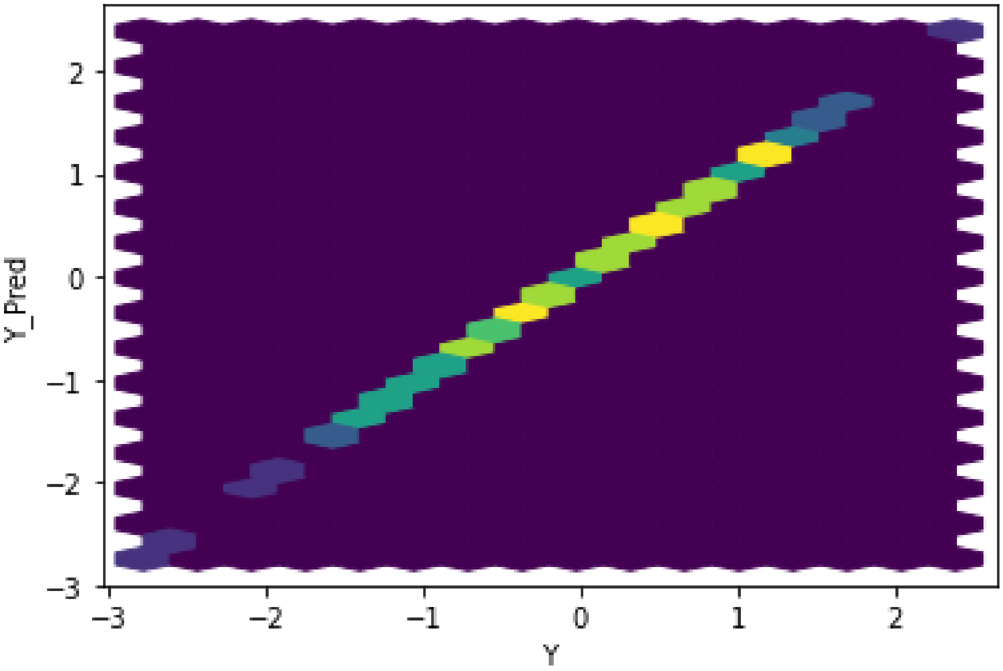
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

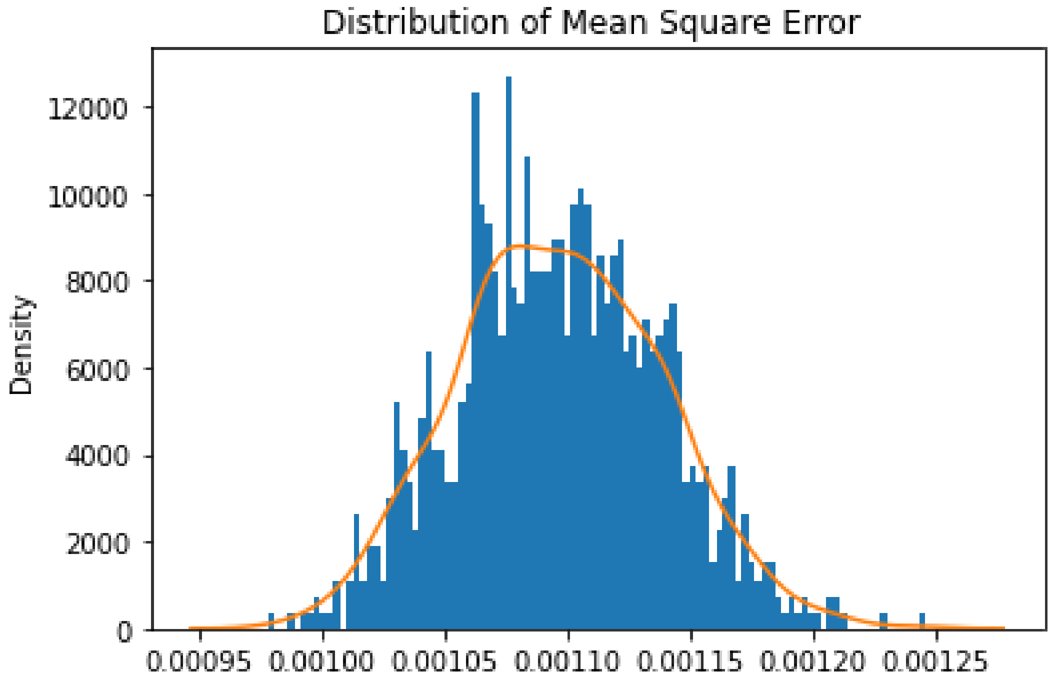



Y_Pred
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
Y

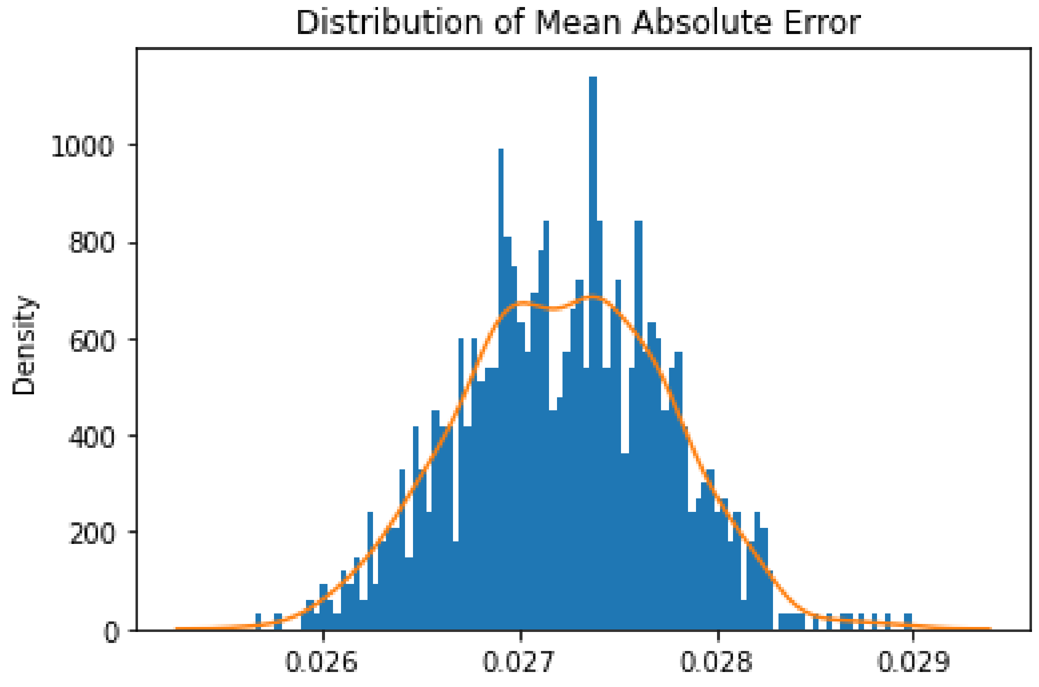

Y_Pred
2
1
0
−1
−2
−3
−2
−1
0
1
2
Y


2
1
0
−1
−2
−3
Y_Pred
−3
−2
−1
0
1
2
Y

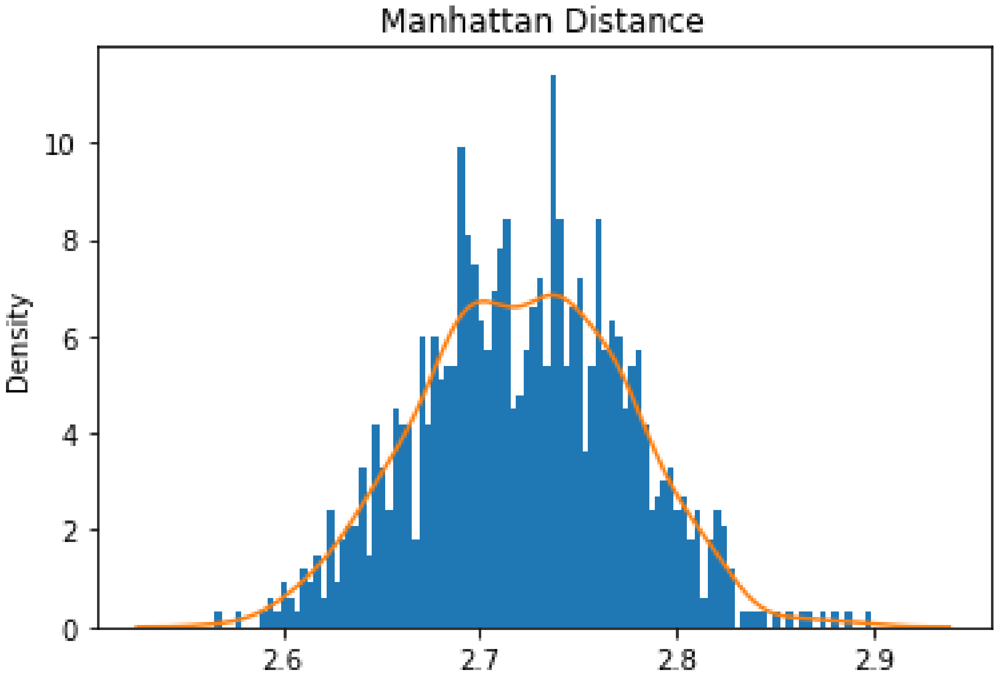

2
1
0
−1
−2
−3
Y_Pred
−3
−2
−1
0
1
2
Y



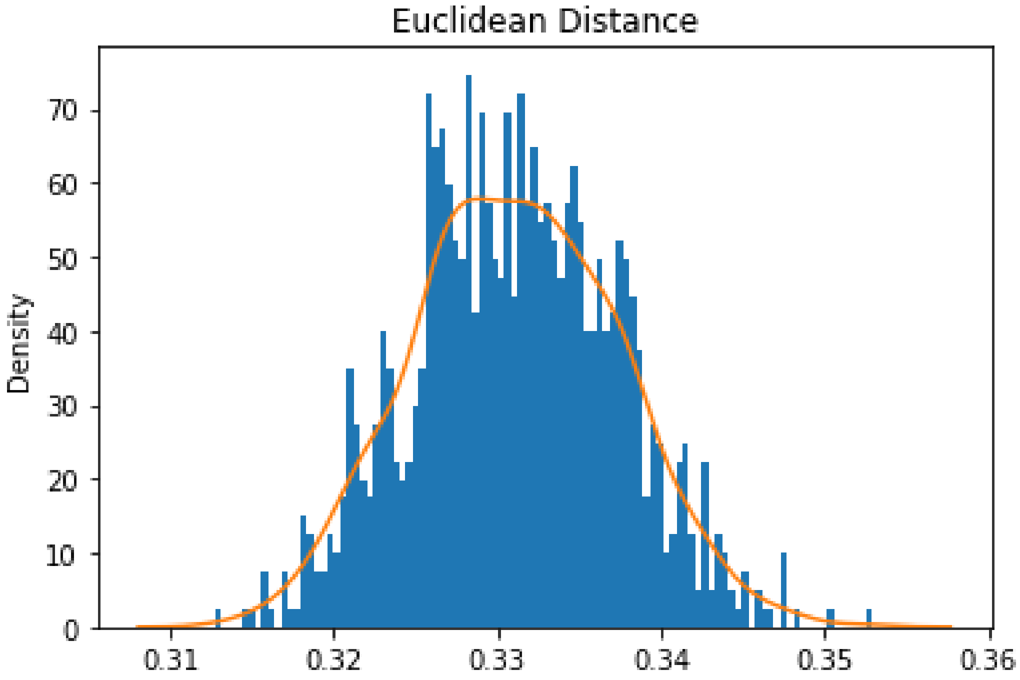
```
Mean Square Error: 0.0010967139673679445
```



```
Mean Absolute Error: 0.027223766277730464
Mean Manhattan Distance: 2.7223766277730466
```

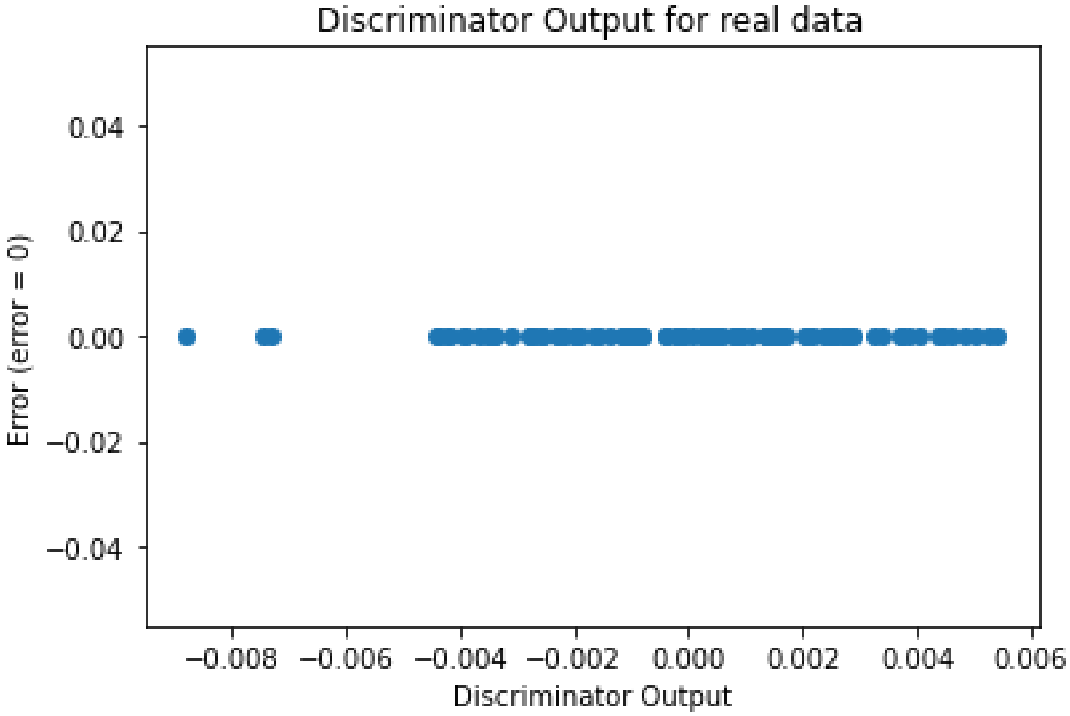


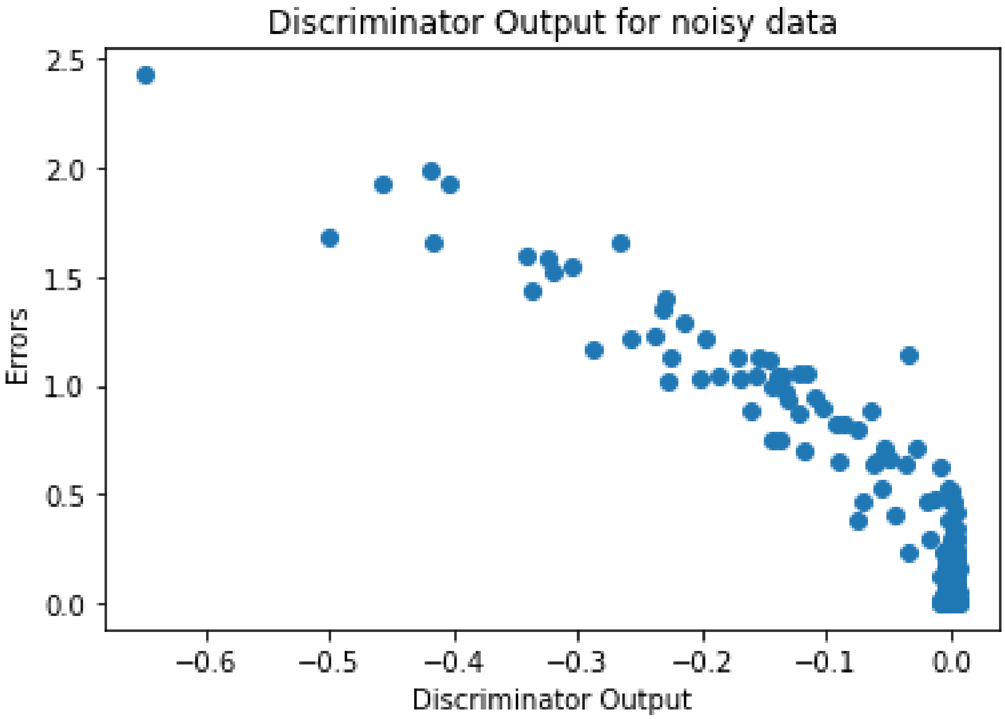
```
Mean Euclidean Distance: 0.33110759115752275
```



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([[0.1161, 0.0508, 0.0495, 0.0062, 0.0509, 0.0575, 0.0336, 0.1415, 0.2096,
         0.2183, 0.0700, 0.6454]], requires_grad=True)
output.bias Parameter containing:
tensor([-0.1166], requires_grad=True)
```