Dataset1-Regression_output_8

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)
```

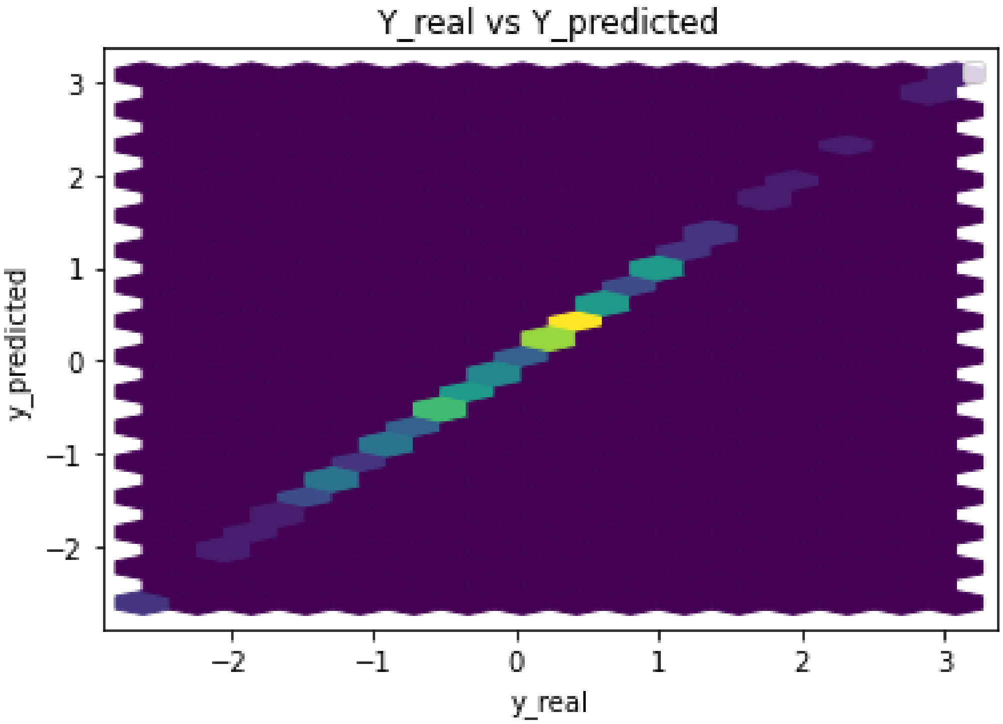
```
         X1        X2        X3        X4        X5        X6        X7  \
0 -1.656416 -0.949564 -0.692383 -1.808891  0.093037 -0.361687  0.472164
1 -0.431644 -2.265805 -0.452761  1.082639  1.864547  1.881561 -0.275738
2  1.811440 -0.872331  0.939273 -1.317775 -1.100259  0.122042  1.326330
3  0.718427  0.679431  0.008781 -0.232915 -0.134580 -1.697107  1.339020
4 -0.968756 -0.355413  0.115665 -0.046795  0.376074  0.768493  0.378615

         X8        X9       X10           Y
0  0.159213  0.265640  1.609512 -140.008215
1  0.067216 -1.162052 -0.240740  -55.745264
2 -0.403082  1.317332  0.764115  165.895638
3  0.755385 -0.907641 -0.636307  -31.211088
4  0.361567  1.125186 -0.599842   75.616759
```

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:                 4.027e+07
Date:                Thu, 07 Oct 2021   Prob (F-statistic):          1.36e-291
Time:                        19:10:27   Log-Likelihood:                 624.36
No. Observations:                 100   AIC:                            -1227.
Df Residuals:                      89   BIC:                            -1198.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const               0   4.98e-05          0      1.000    -9.9e-05     9.9e-05
x1             0.3836   5.19e-05   7387.395      0.000       0.384       0.384
x2             0.3968   5.15e-05   7701.383      0.000       0.397       0.397
x3             0.0226   5.09e-05    443.393      0.000       0.022       0.023
x4             0.1288   5.16e-05   2496.339      0.000       0.129       0.129
x5             0.4132   5.11e-05   8091.321      0.000       0.413       0.413
```

```
x6             0.2778   5.12e-05   5422.621      0.000       0.278       0.278
x7             0.2764   5.09e-05   5424.815      0.000       0.276       0.276
x8             0.0700   5.25e-05   1332.832      0.000       0.070       0.070
x9             0.4521   5.16e-05   8760.317      0.000       0.452       0.452
x10            0.1776   5.18e-05   3426.706      0.000       0.177       0.178
==============================================================================
Omnibus:                        0.619   Durbin-Watson:                   1.897
Prob(Omnibus):                  0.734   Jarque-Bera (JB):                0.271
Skew:                           0.096   Prob(JB):                        0.873
Kurtosis:                       3.167   Cond. No.                         1.54
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    0.000000
x1      0.383612
x2      0.396801
x3      0.022557
x4      0.128829
x5      0.413214
x6      0.277812
x7      0.276374
x8      0.069962
x9      0.452108
x10     0.177586
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 2.2099613483510607e-07
Mean Absolute Error: 0.00036621881978991355
Manhattan distance: 0.036621881978991355
Euclidean distance: 0.004701022599765992
```

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]:

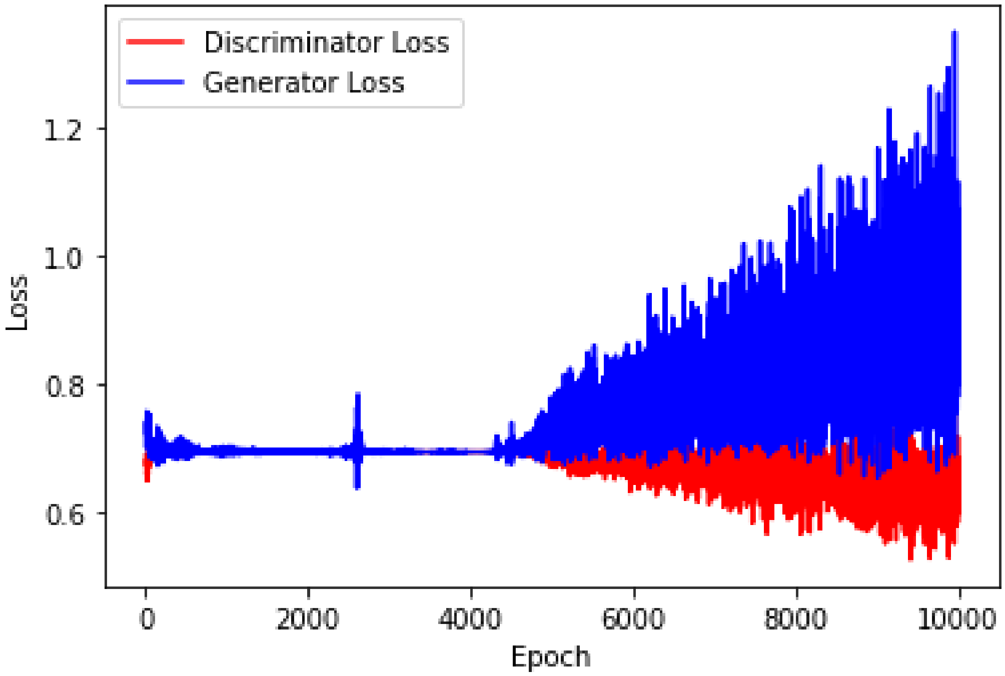
```
n_epochs = 5000
batch_size = sample_size//2
```

[12]:

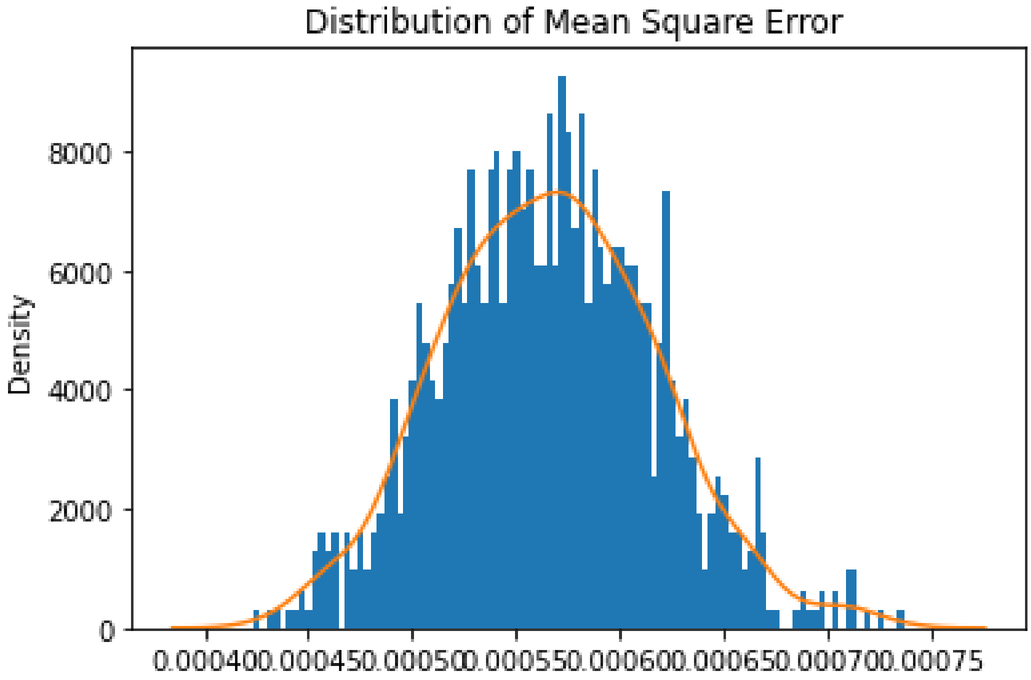
```
# Parameters
sample_size = 10000
mean = 1
std = 0.01
```

[13]:

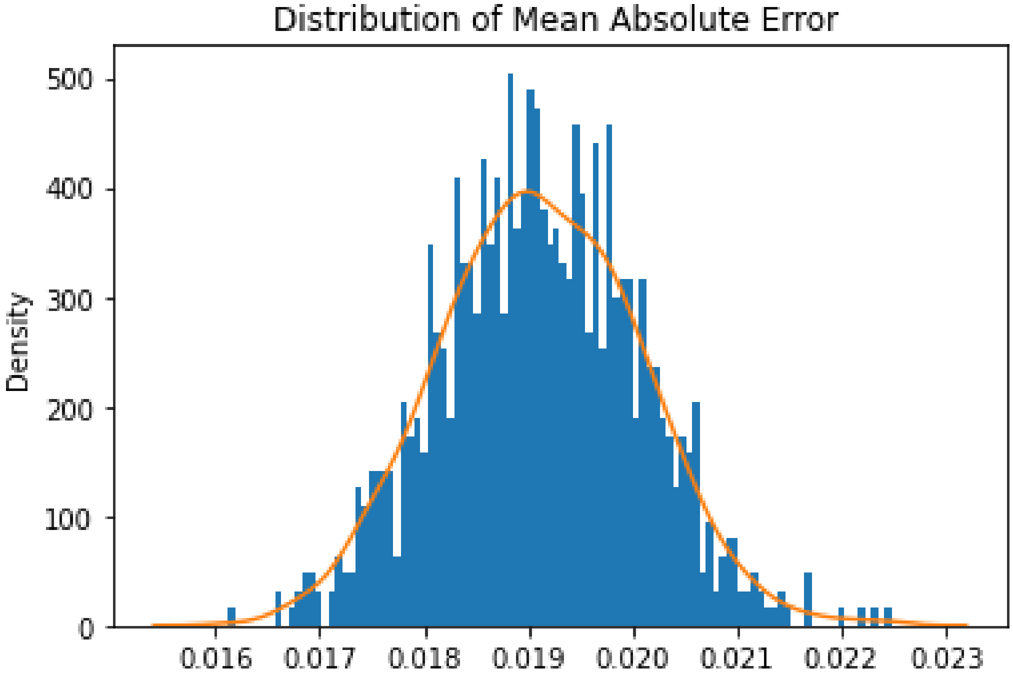
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



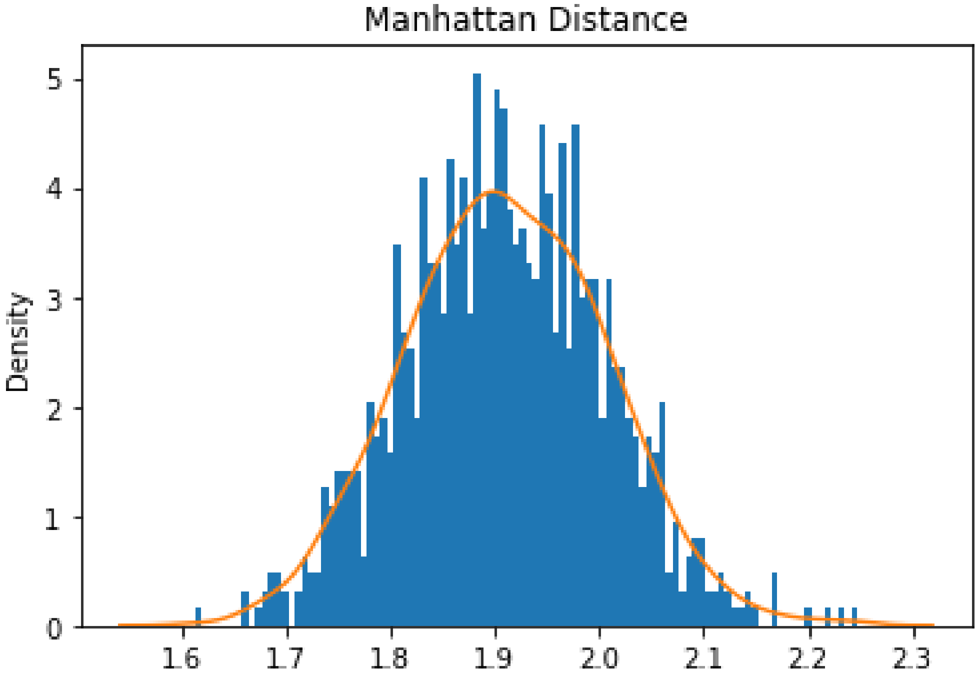
```
[14]: train_test.test_generator(generator,real_dataset,device)
```



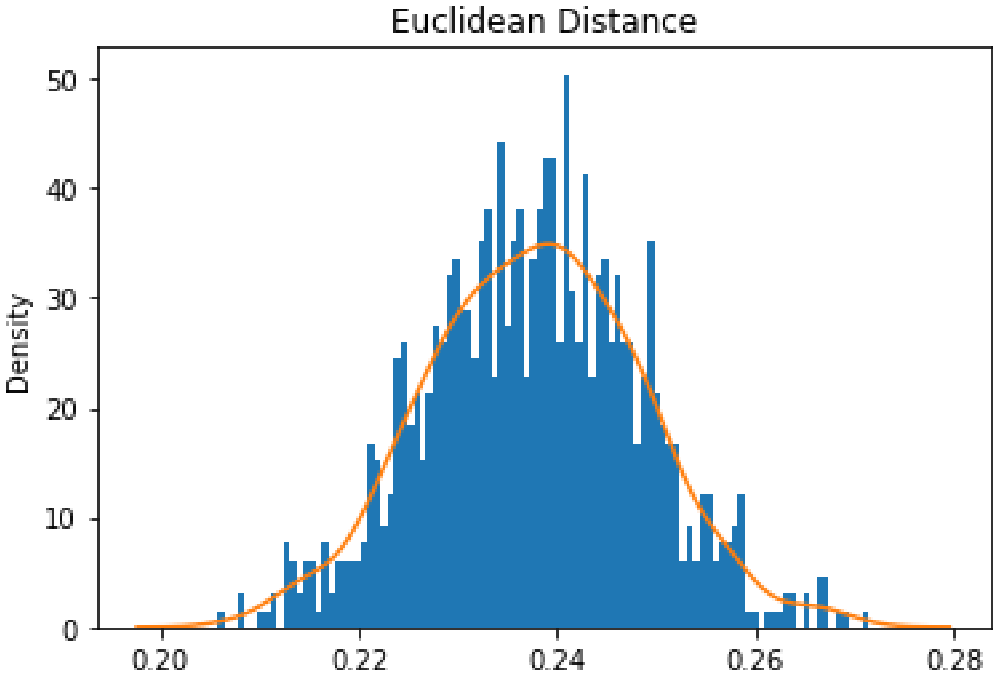
```
Mean Square Error: 0.0005659950556325695
```



```
Mean Absolute Error: 0.01910314335592091
```



```
Mean Manhattan Distance: 1.910314335592091
```



```
Mean Euclidean Distance: 1.910314335592091
```

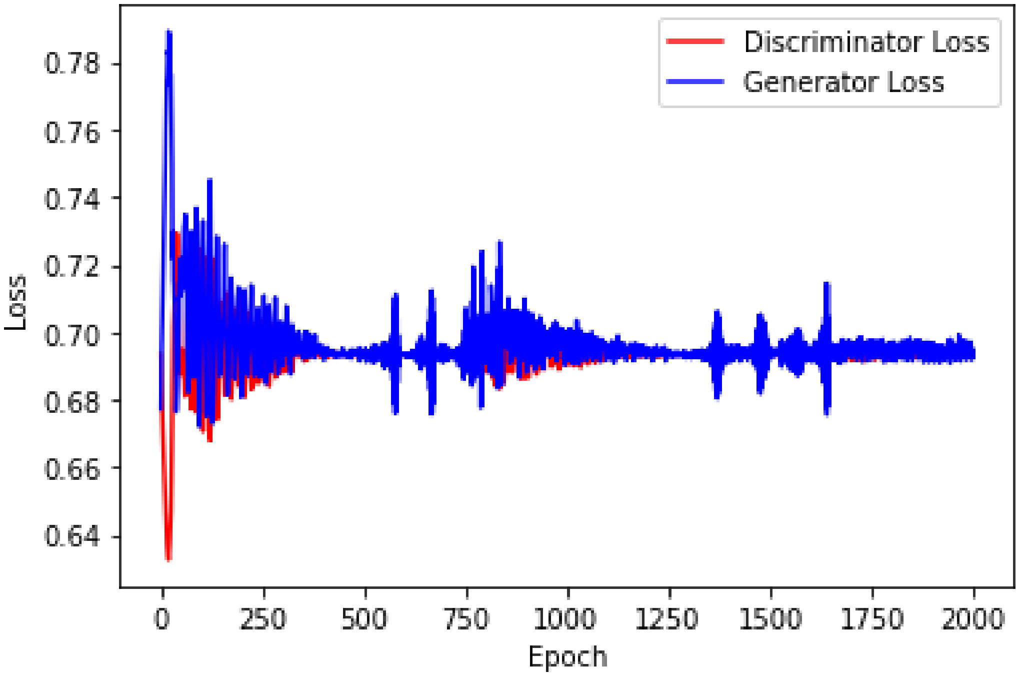
4 ABC GAN Model

Training the network

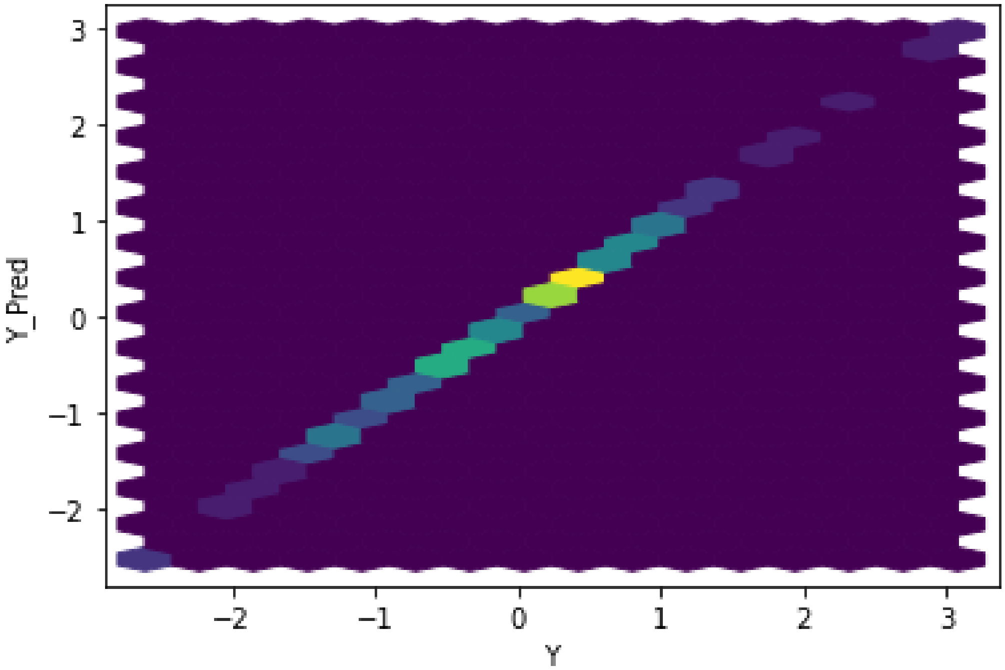
```
[15]: gen = Generator(n_features+2)
      disc = Discriminator(n_features+2,hidden_nodes)

      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))
```

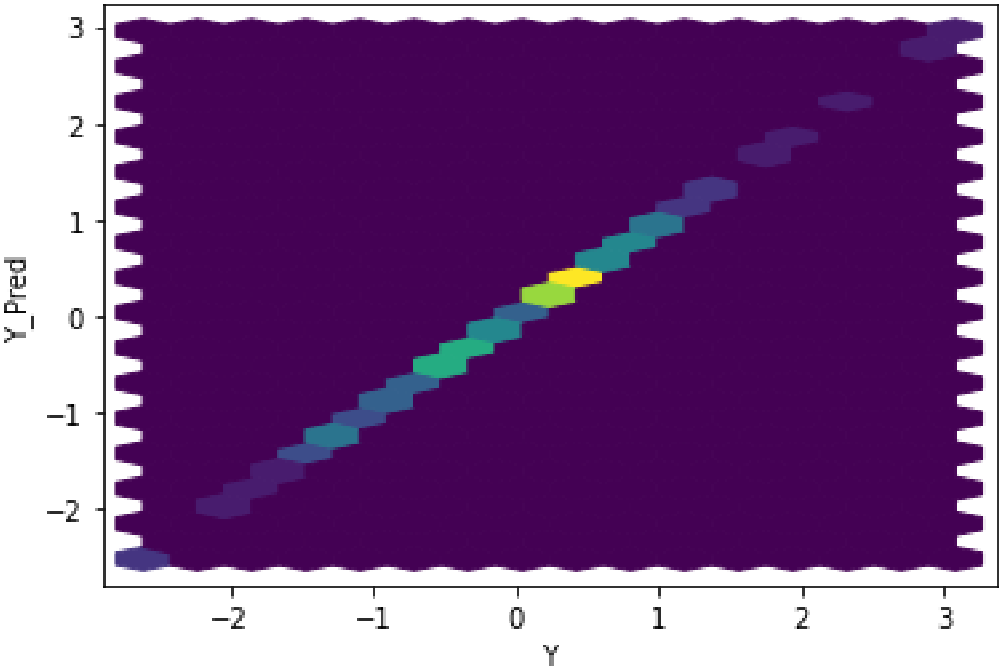
```
[16]: n_epoch_abc = 2000
      batch_size = sample_size//2
```

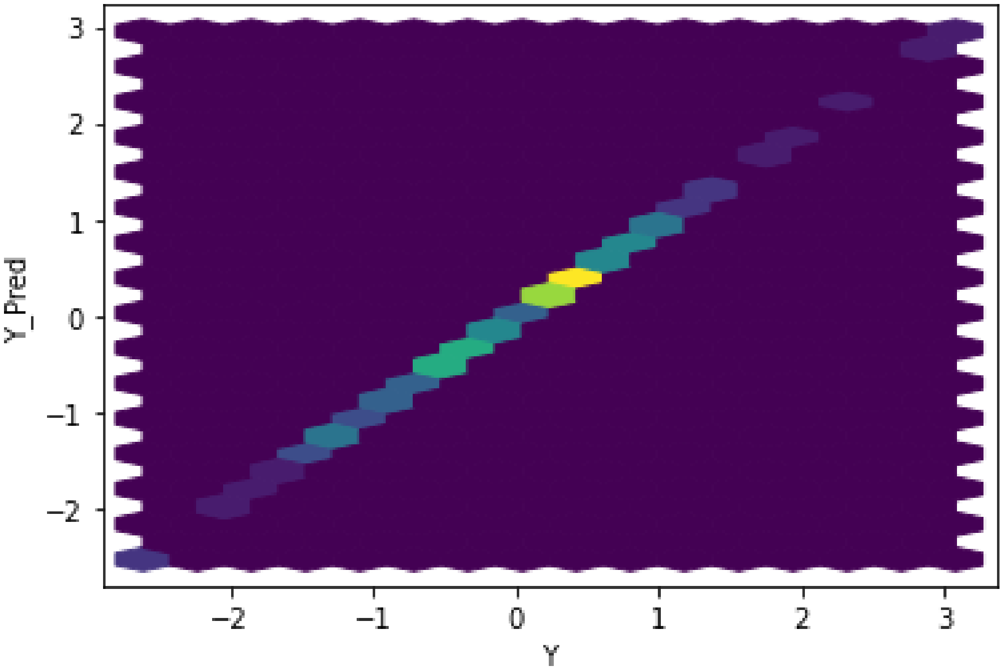
```
[17]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
       ↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

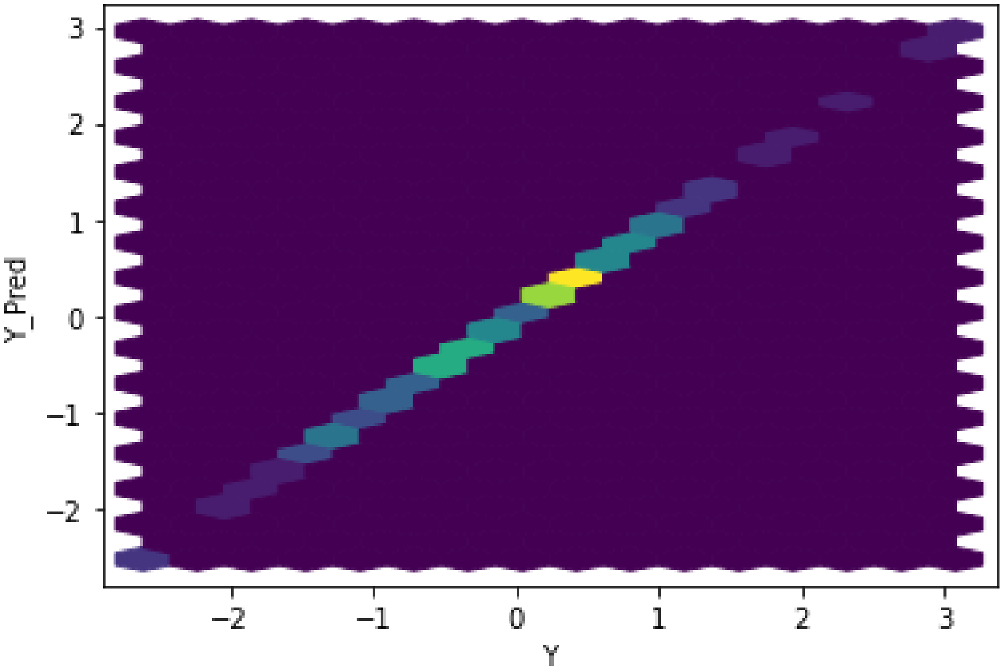


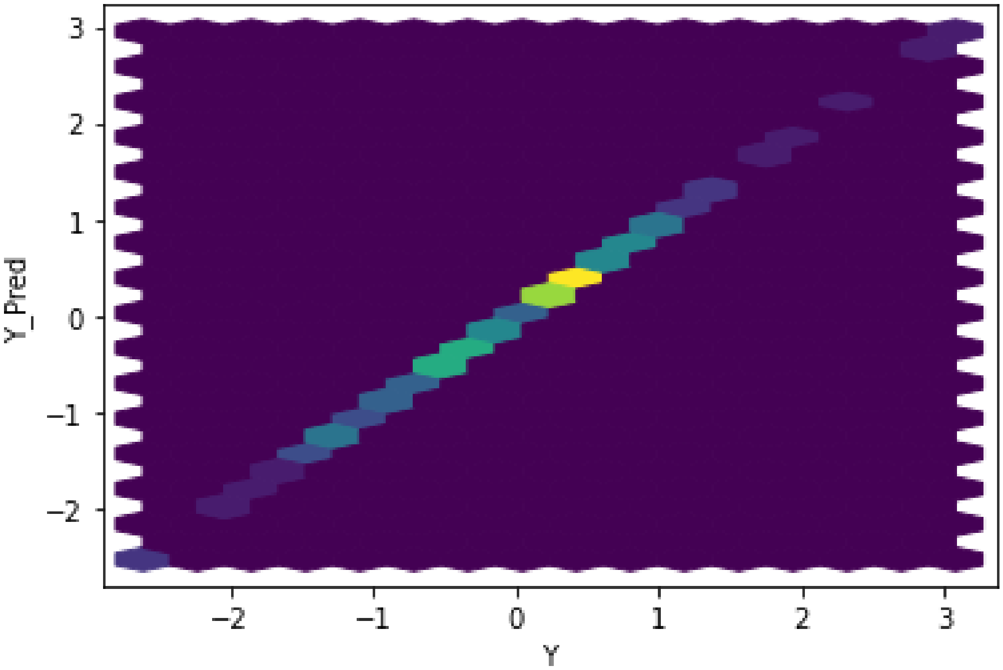
[18]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

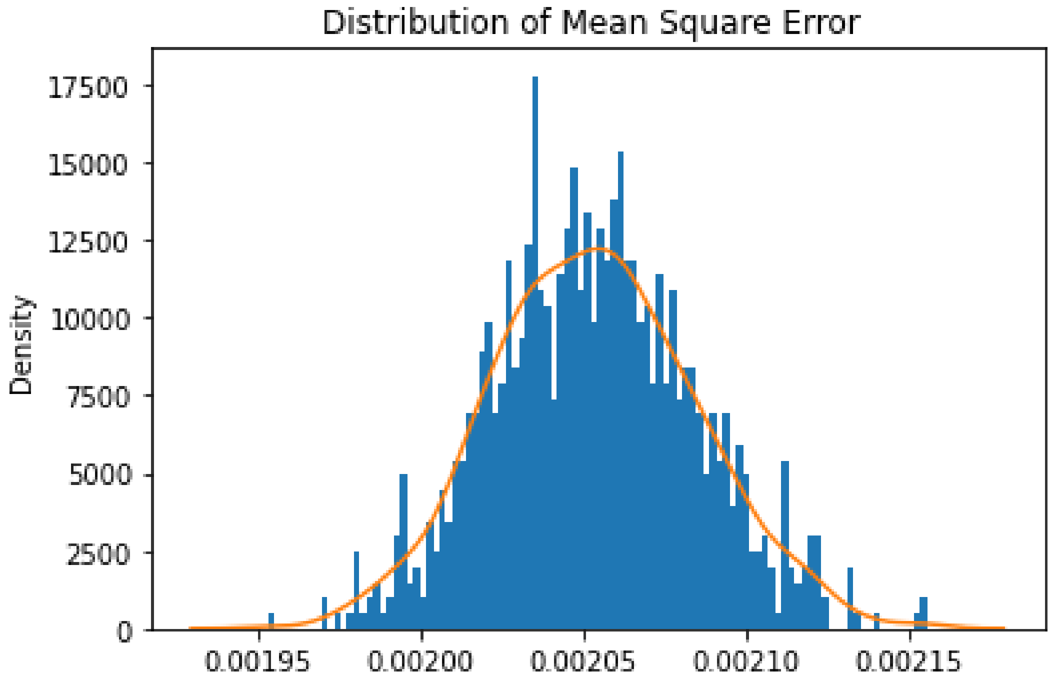



3
2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
3
Y

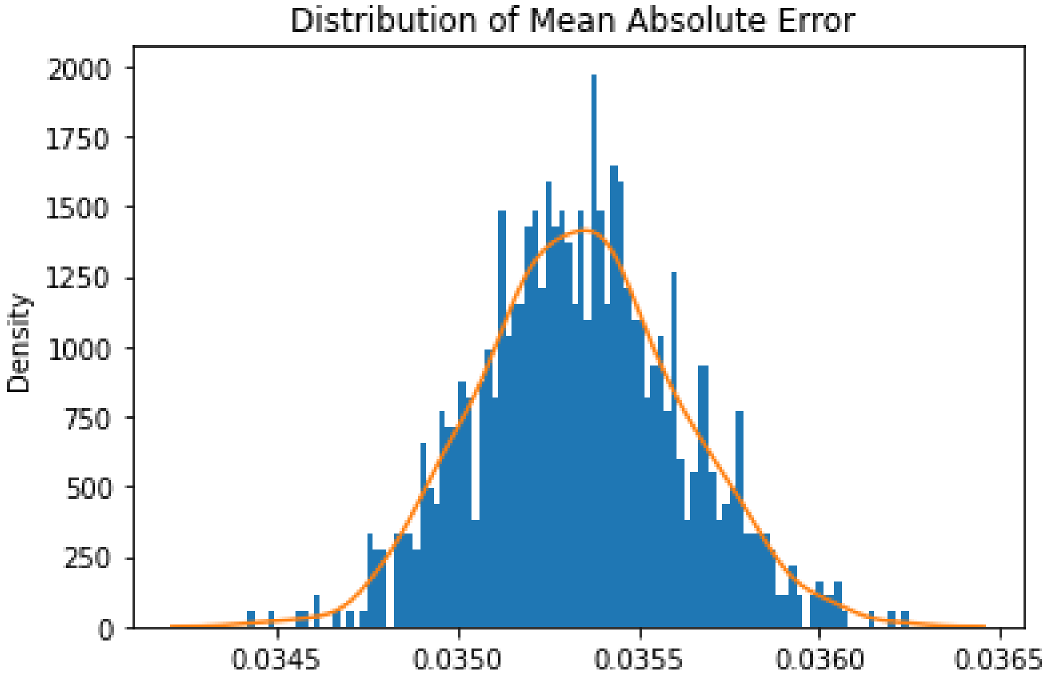

3
2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
3
Y


3
2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
3
Y

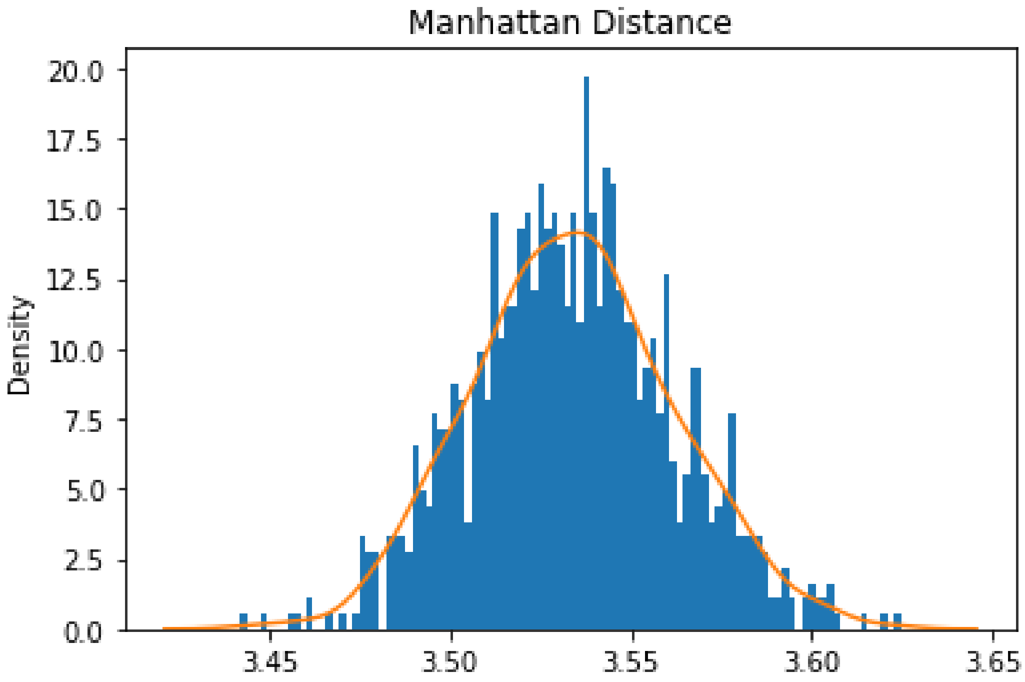

3
2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
3
Y



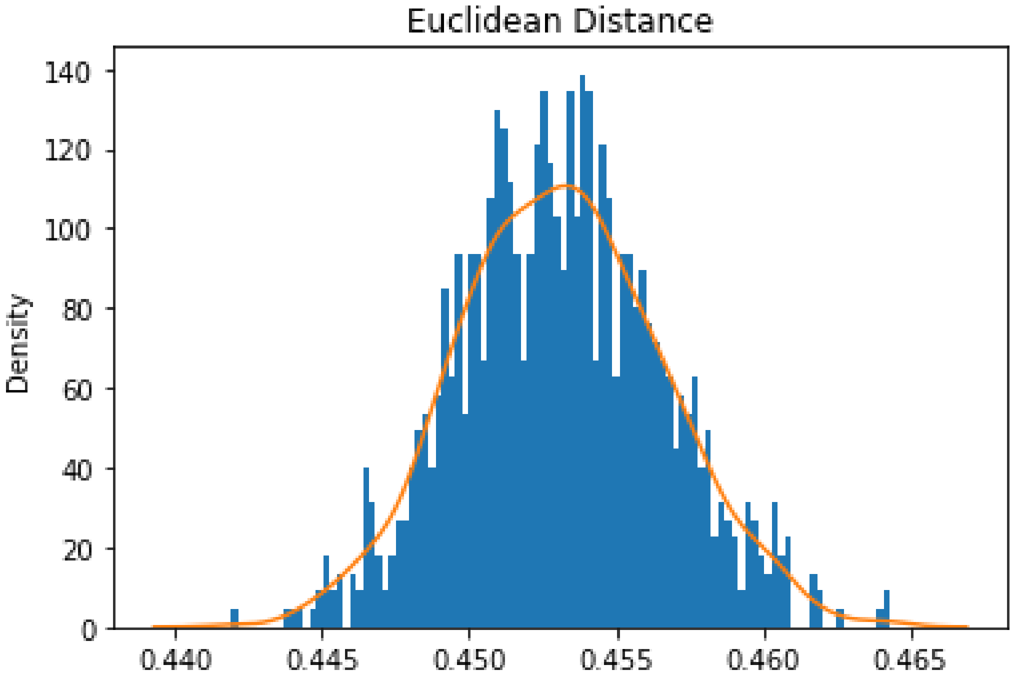
```
Mean Square Error: 0.002053233686088733
```



```
Mean Absolute Error: 0.035339297489747441
Mean Manhattan Distance: 3.5339297489747405
```



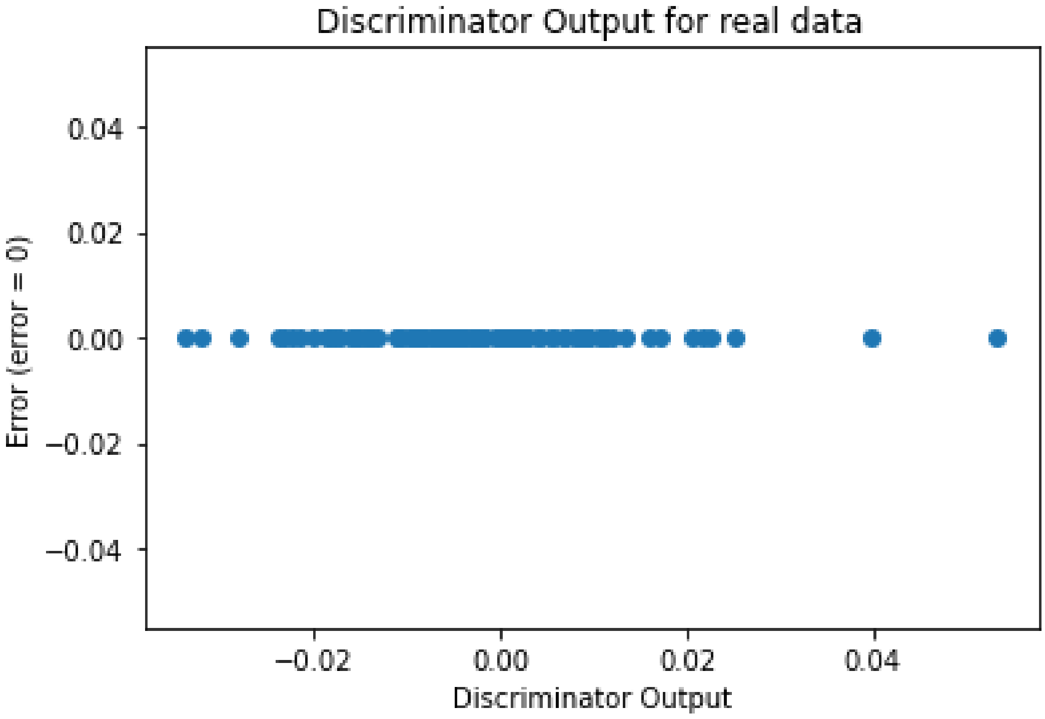
```
Mean Euclidean Distance: 0.45311302742140525
```

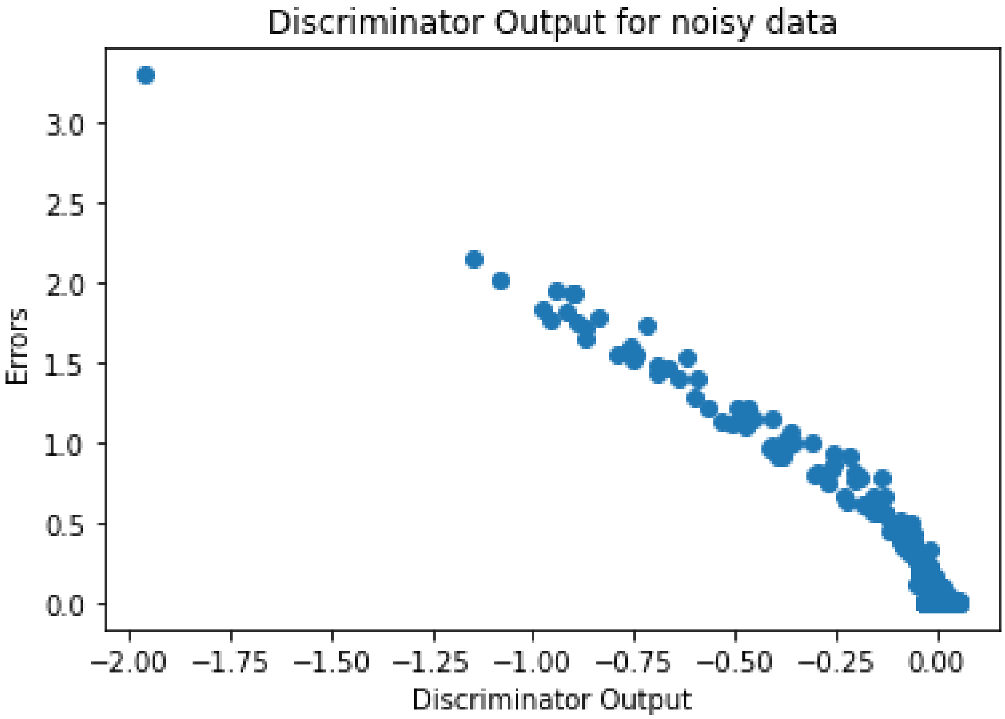


Sanity Checks

[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([[-0.2073,  0.2463,  0.2492,  0.0192,  0.0818,  0.2695,  0.1802,  0.1812,
          0.0339,  0.2886,  0.1044,  0.3221]], requires_grad=True)
output.bias Parameter containing:
tensor([0.1916], requires_grad=True)
```