Dataset2_Friedman1_output_7

October 20, 2021

1 Dataset 2 - Friedman 1

1.1 Experiment Details

The aim of the experiment is to verify if the: 1. ABC_GAN model corrects model misspecification 2. ABC_GAN model performs better and converges faster than a simple C-GAN model

In the experiment we predict the distribution that represents the real data and simulate realistic fake data points using statistical mode, C-GAN and ABC-GAN model with 3 priors. We analyze and compare their performance using metrics like mean squared error, mean absolute error , manhattan distance and euclidean distance between y_{real} and y_{pred}

The models are as follows:

1. The statistical model assumes the distribution $Y = \beta X + \mu$ where $\mu \sim N(0, 1)$
2. The Conditional GAN consists of
   1. Generator with 2 hidden layers with 100 nodes each and ReLu activation.
   2. Discriminator with 2 hidden layers with 25 and 50 nodes and ReLu activation. We use Adam's optimser and BCE Logit Loss to train the model. The input to the Generator of the GAN is (x,e) where x are the features and $e \sim N(0, 1)$. The discriminator output is linear.
3. The ABC GAN Model consists of
   1. ABC generator is defined as follows:
      1. $Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_2 x_3 + ... + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$
      2. $\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else $\beta_i \sim N(\beta_i^*, \sigma^*)$ where $\beta_i^* s$ are coefficients obtained from statistical model
      3. σ^* takes the values 0.01,0.1 and 1

   2. C-GAN network is as defined above. However the input to the Generator of the GAN is (x, y_{abc}) where y_{abc} is the output of the ABC Generator.

1.2 Import Libraries

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import train_test
import ABC_train_test
import regressionDataset
import network
```

```
import statsModel
import performanceMetrics
import friedman1Dataset
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
```

1.3 Parameters

General Parameters

1. Number of Samples
2. Number of features

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)

[3]:
```
n_features = 10
n_samples= 100

#ABC Generator Parameters
mean = 1
variance = 0.001
```

[4]:
```
# Parameters
n_samples = 100
n_features = 10
mean = 1
variance = 0.1
```

1.4 Dataset

Friedman 1 Dataset

- $y(X) = 10 * sin(pi * X_0 * X_1) + 20 * (X_2 - 0.5) * *2 + 10 * X_3 + 5 * X_4 + noise * N(0, 1)$.
- Only 5 features used to calculate y
- Noise is Gaussian
- 1000 datapoints and 10 features used in the following experiment

[5]:
```
X, Y = friedman1Dataset.friedman1_data(n_samples,n_features)
```

```
         X0        X1        X2        X3        X4        X5        X6  \
0  0.741709  0.836357  0.277396  0.238550  0.891213  0.597750  0.818617
1  0.974559  0.170014  0.586984  0.813366  0.568679  0.300919  0.891190
2  0.733821  0.503313  0.375745  0.825007  0.645372  0.953380  0.895422
```

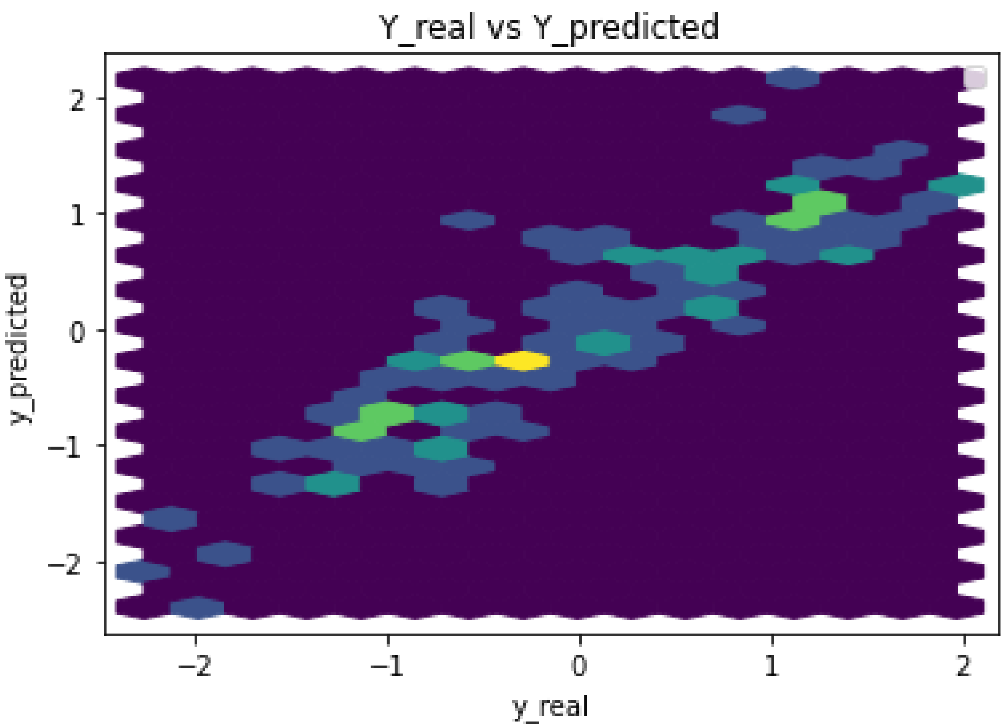
```
3  0.749737  0.322811  0.504907  0.142554  0.035325  0.978275  0.327237
4  0.510832  0.380953  0.331715  0.186408  0.771047  0.240731  0.582816

         X7        X8        X9          Y
0  0.711037  0.635157  0.053874  17.192892
1  0.620058  0.181676  0.421385  16.066280
2  0.769878  0.127839  0.557625  20.817676
3  0.436605  0.653130  0.988263   8.538705
4  0.064173  0.038953  0.442003  11.876300
```

1.5 Stats Model

[6]:
```
[coeff,y_pred] = statsModel.statsModel(X,Y)
```

```
No handles with labels found to put in legend.

                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.814
Model:                            OLS   Adj. R-squared:                  0.793
Method:                 Least Squares   F-statistic:                     38.85
Date:                Wed, 20 Oct 2021   Prob (F-statistic):           3.10e-28
Time:                        20:12:33   Log-Likelihood:                -57.898
No. Observations:                 100   AIC:                             137.8
Df Residuals:                      89   BIC:                             166.5
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.166e-16      0.046   6.92e-15      1.000      -0.091       0.091
x1             0.3624      0.047      7.716      0.000       0.269       0.456
x2             0.3879      0.047      8.239      0.000       0.294       0.482
x3             0.0067      0.048      0.140      0.889      -0.088       0.101
x4             0.5738      0.047     12.114      0.000       0.480       0.668
x5             0.2585      0.049      5.253      0.000       0.161       0.356
x6             0.0643      0.046      1.385      0.169      -0.028       0.157
x7            -0.0082      0.049     -0.169      0.867      -0.105       0.088
x8             0.0244      0.048      0.510      0.611      -0.070       0.119
x9             0.0916      0.047      1.940      0.056      -0.002       0.185
x10            0.0257      0.047      0.547      0.586      -0.068       0.119
==============================================================================
Omnibus:                        6.853   Durbin-Watson:                   2.293
Prob(Omnibus):                  0.033   Jarque-Bera (JB):                6.330
Skew:                          -0.535   Prob(JB):                       0.0422
Kurtosis:                       3.612   Cond. No.                         1.59
==============================================================================

Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    3.165870e-16
x1       3.624440e-01
x2       3.879415e-01
x3       6.652854e-03
x4       5.737678e-01
x5       2.584521e-01
x6       6.432013e-02
x7      -8.194011e-03
x8       2.435514e-02
x9       9.160908e-02
x10      2.571214e-02
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 0.18638785961512636
Mean Absolute Error: 0.3358884099513702
Manhattan distance: 33.58884099513702
Euclidean distance: 4.317266028577881
```

1.6 Common Training Parameters (GAN & ABC_GAN)

[7]:
```
n_epochs = 5000
error = 0.001
batch_size = n_samples//2
```

1.7 GAN Model

[8]:
```
real_dataset = dataset.CustomDataset(X,Y)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

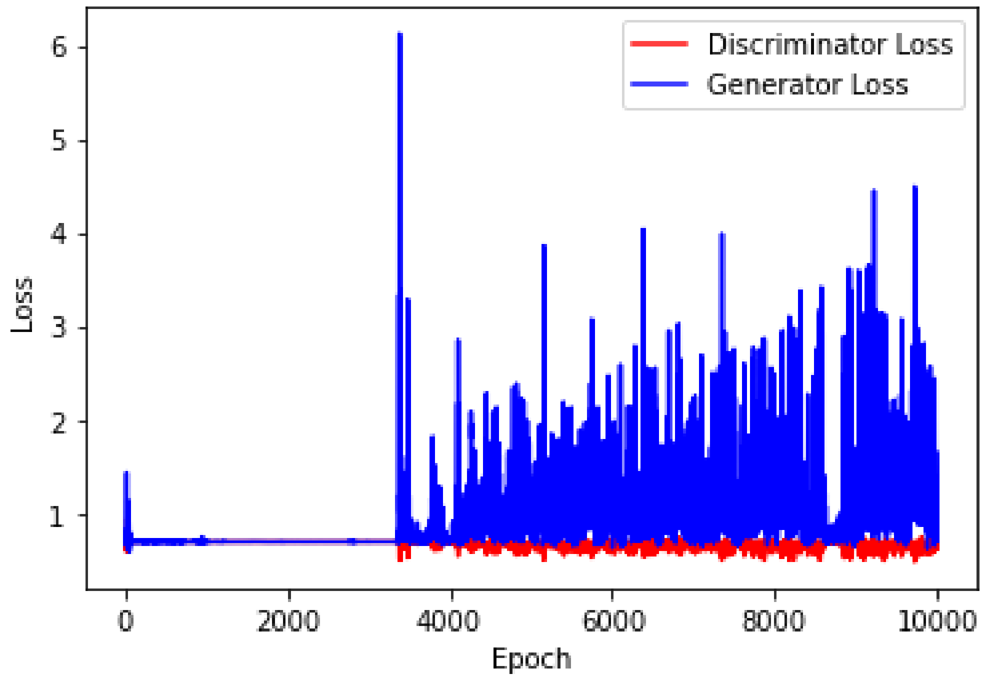
Training GAN for n_epochs number of epochs

[9]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)

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))
```

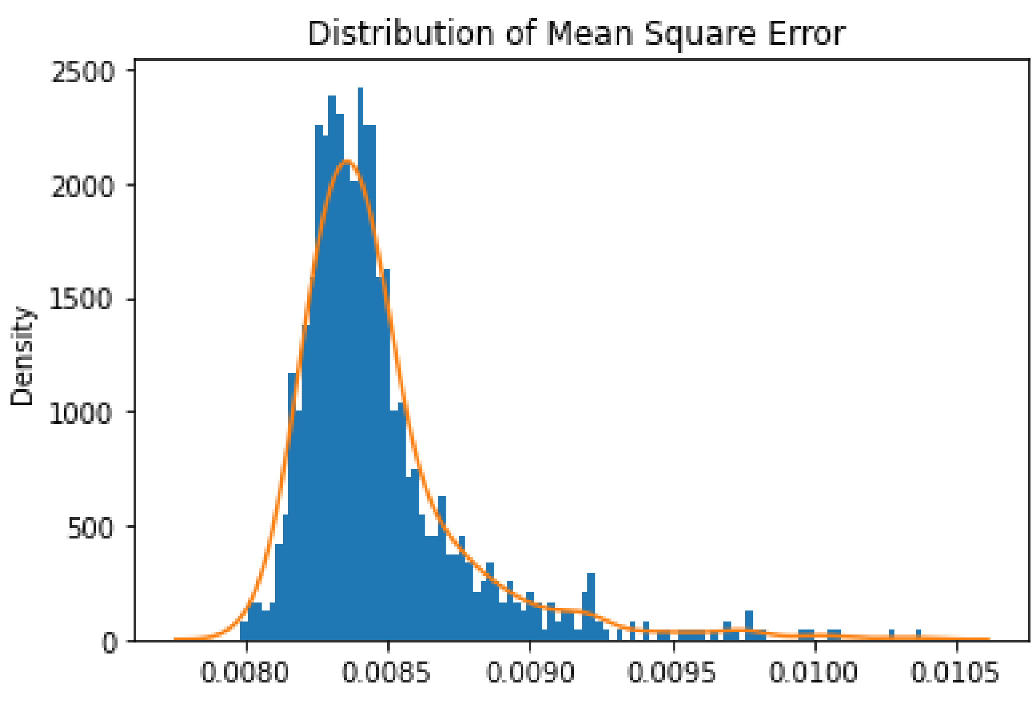
[10]:
```
print(generator)
print(discriminator)
```

```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)
  (hidden2): Linear(in_features=25, out_features=50, bias=True)
  (output): Linear(in_features=50, out_features=1, bias=True)
  (relu): ReLU()
)
```

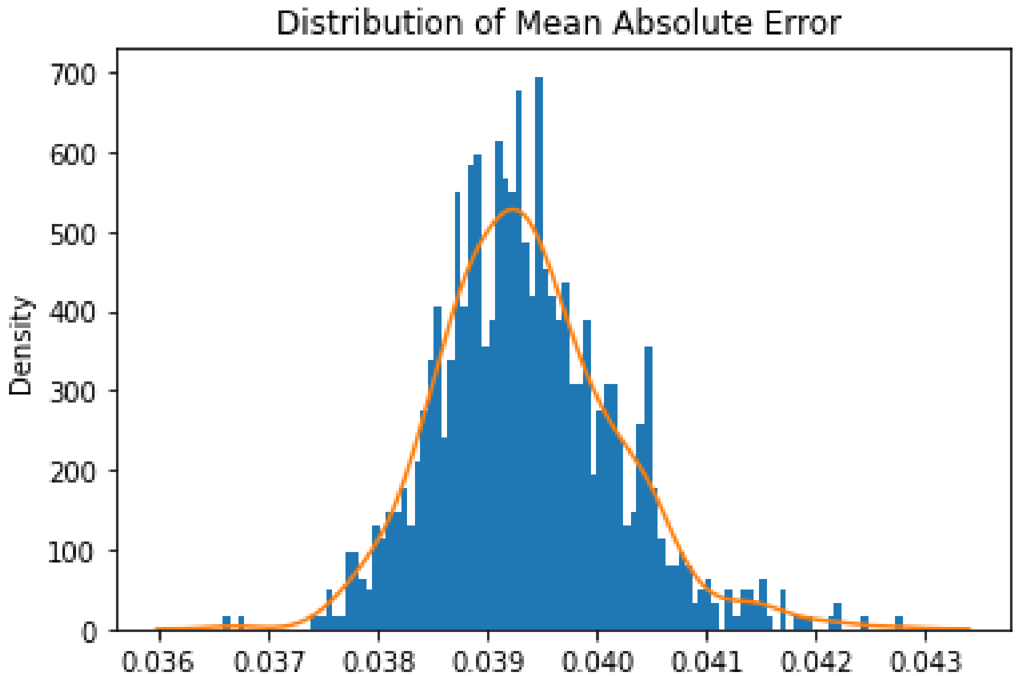
[11]:
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



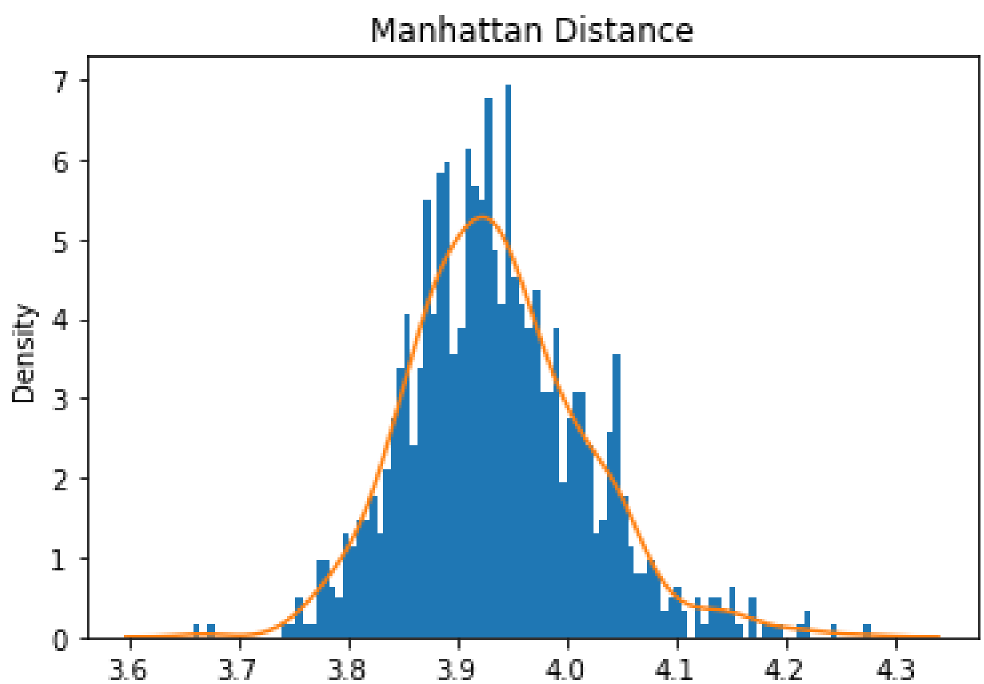
[12]:
```
train_test.test_generator(generator,real_dataset,device)
```



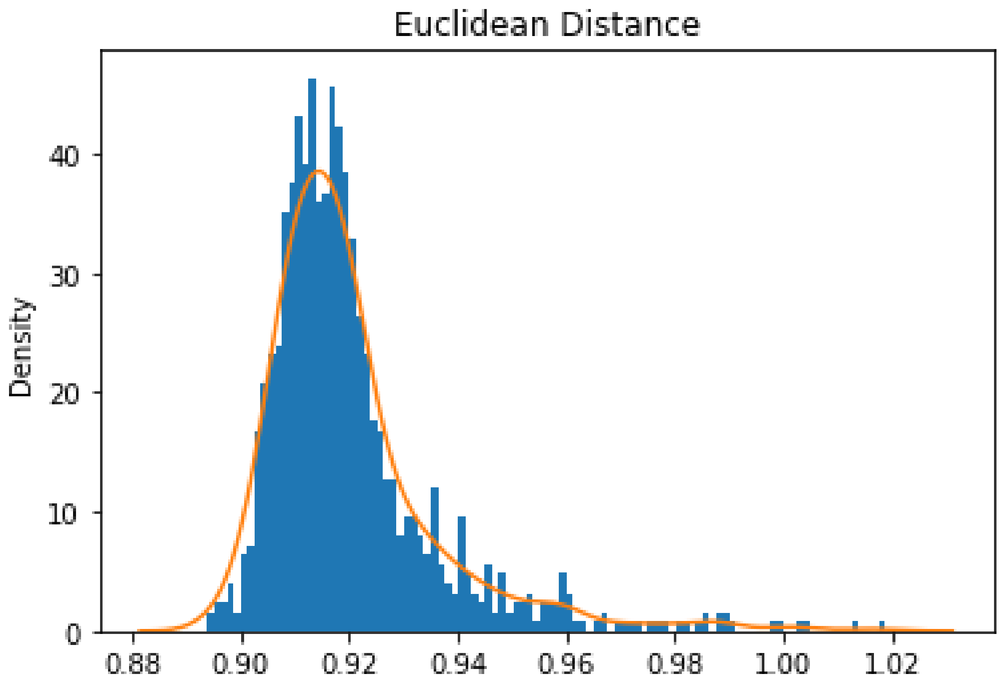
Mean Square Error: 0.008472729746303026



Mean Absolute Error: 0.03935846896100789

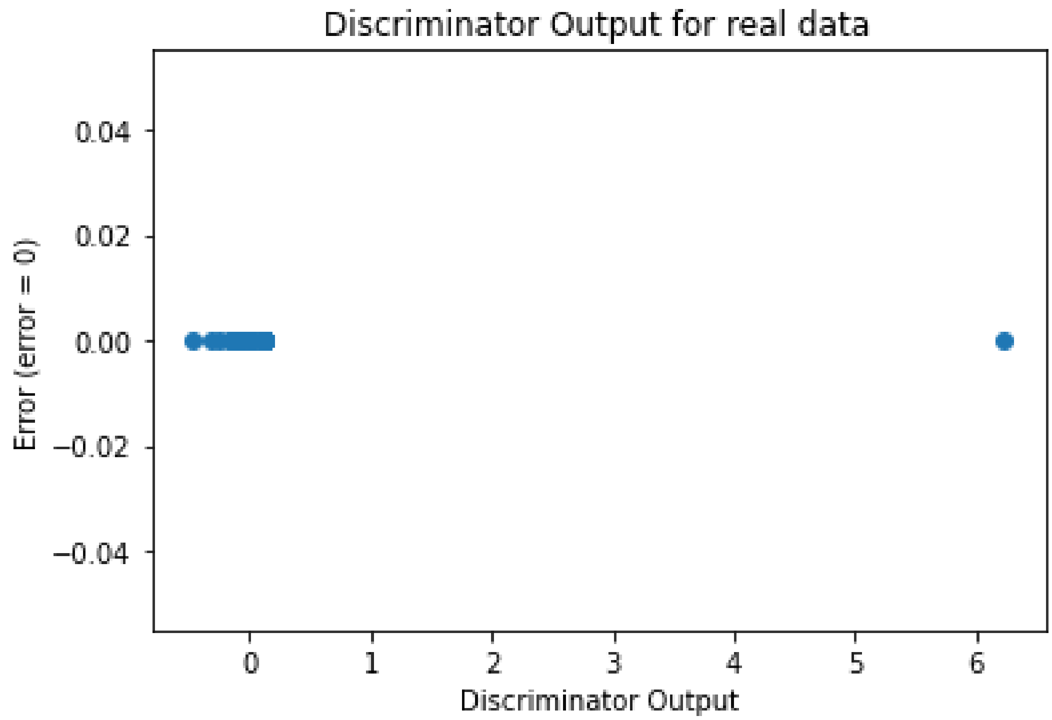


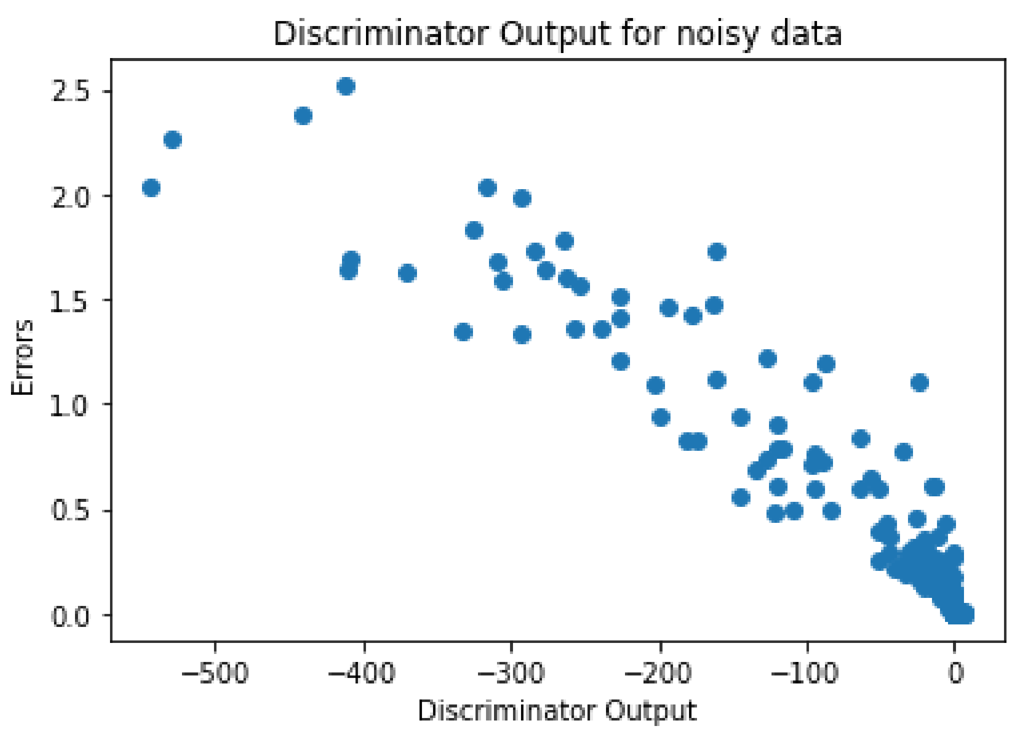
```
Mean Manhattan Distance: 3.9358468961007893
```




```
Mean Euclidean Distance: 0.9203292367295844
```

[13]:
```
sanityChecks.discProbVsError(real_dataset,discriminator,device)
```


Discriminator Output for real data
Error (error = 0)
0.04
0.02
0.00
−0.02
−0.04
0
1
2
3
4
5
6
Discriminator Output

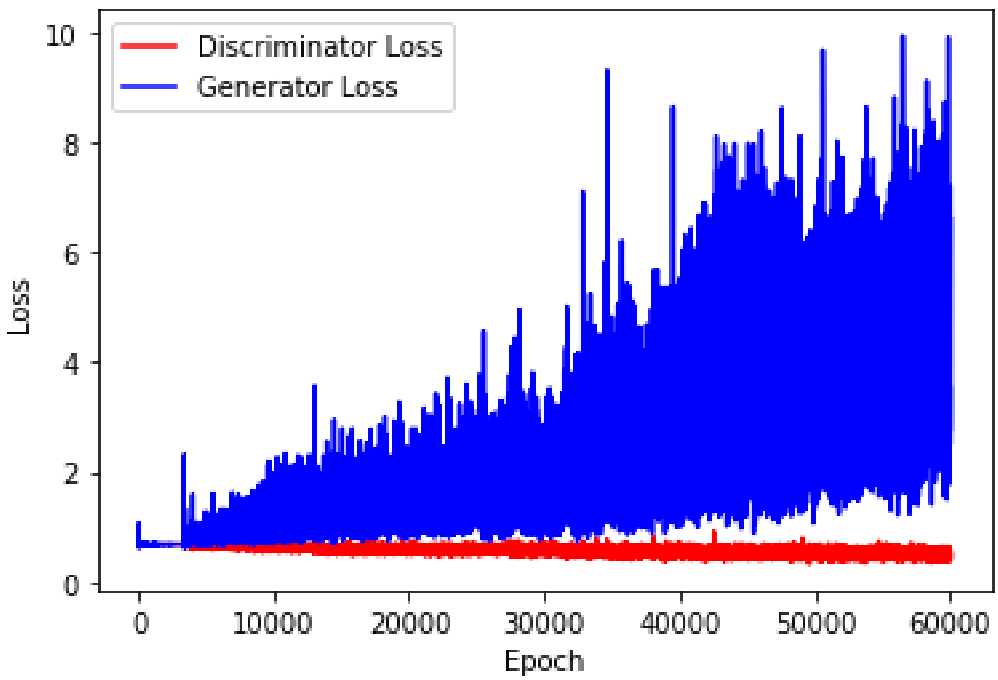

Discriminator Output for noisy data
Errors
2.5
2.0
1.5
1.0
0.5
0.0
−500
−400
−300
−200
−100
0
Discriminator Output

Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

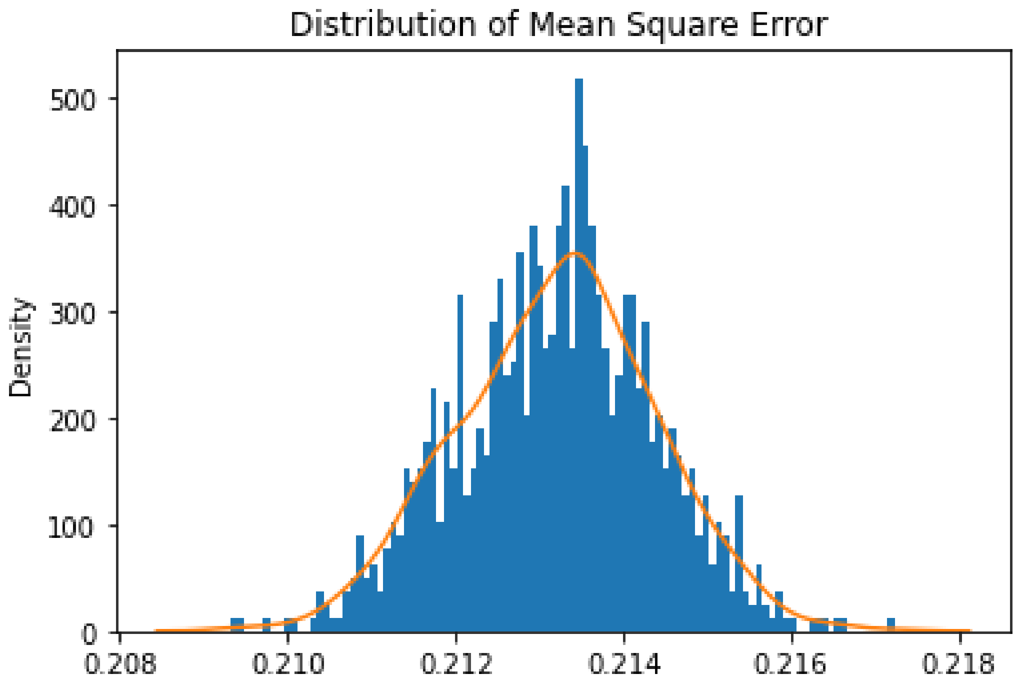
[14]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)
criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5, 0.
 ↪999))
```

[15]:
```
train_test.
 ↪training_GAN_2(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,error,crite
```

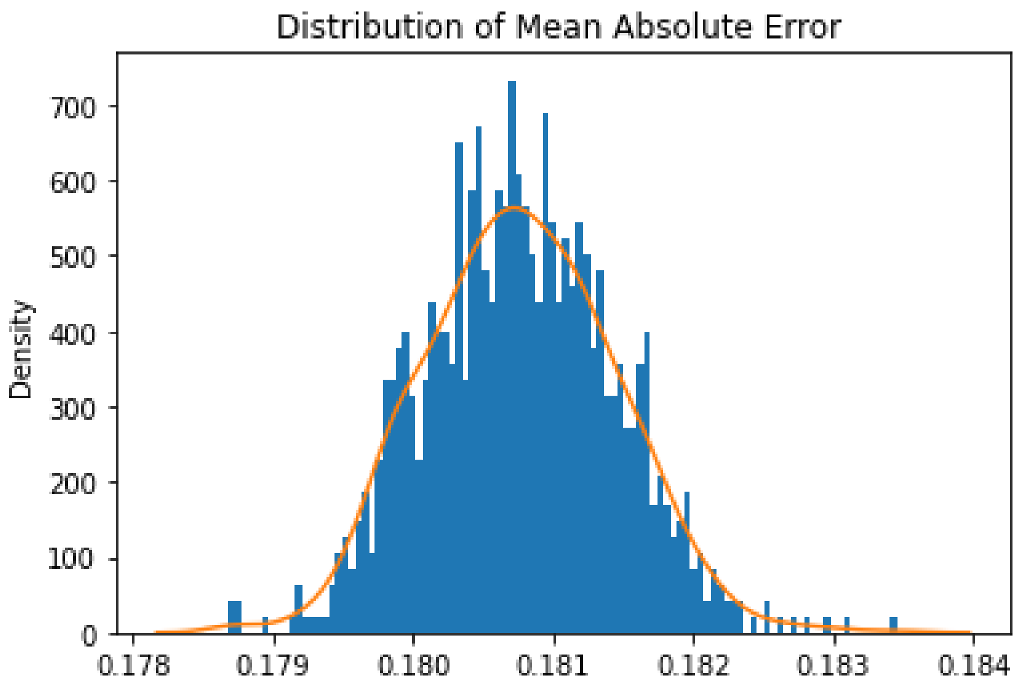
```
Number of epochs needed 30000
```



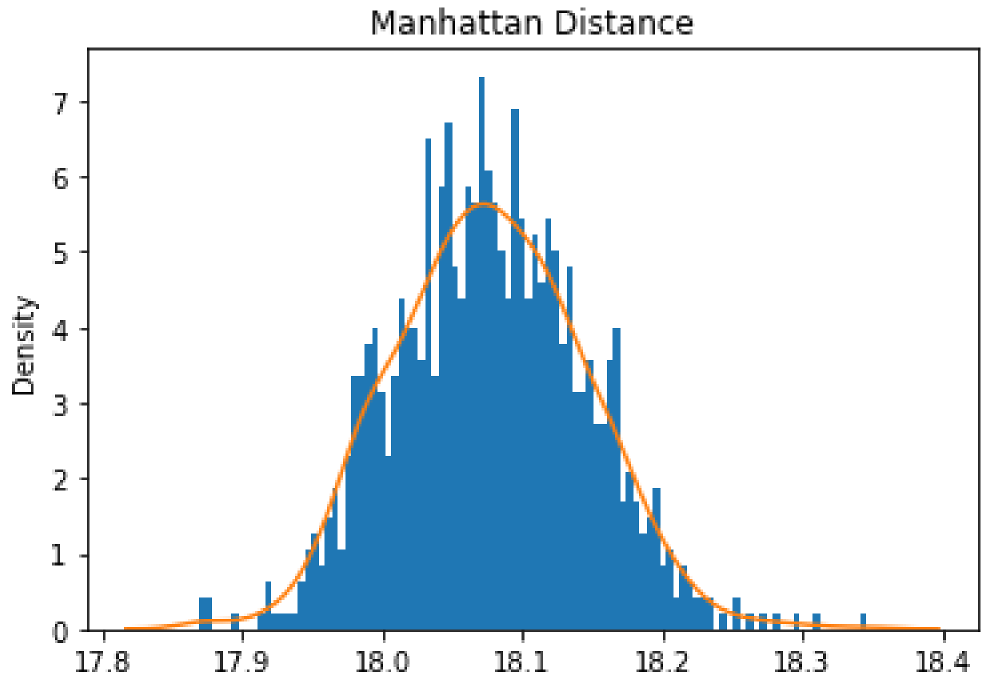
[16]:
```
train_test.test_generator(generator,real_dataset,device)
```



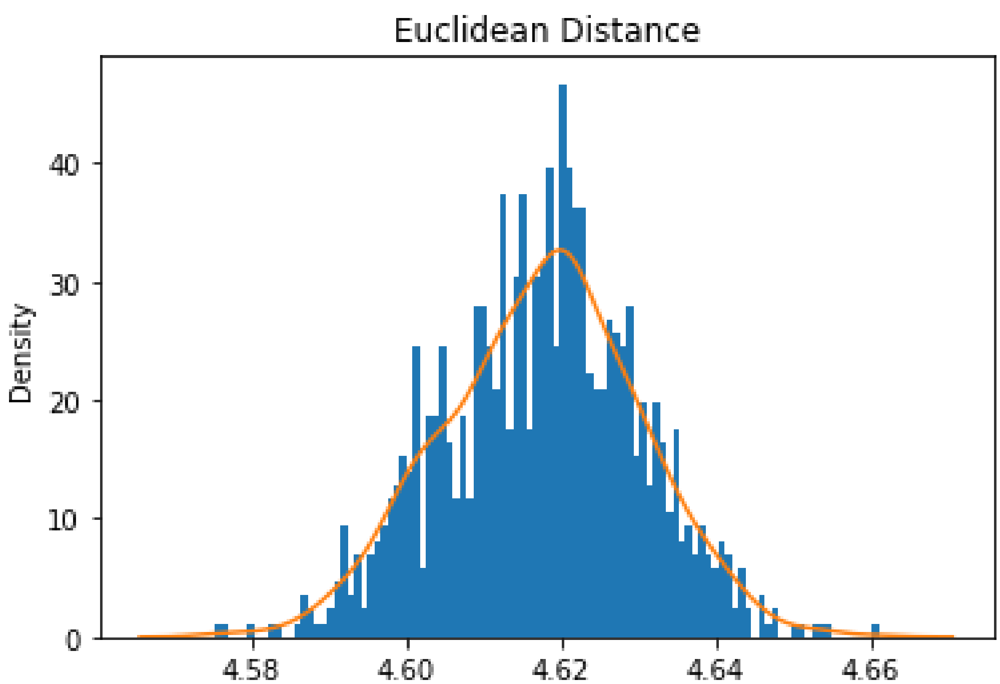
Mean Square Error: 0.2132086230648661



```
Mean Absolute Error: 0.1807658507529646
```




```
Mean Manhattan Distance: 18.07658507529646
```



```
Mean Euclidean Distance: 4.6174345184261085
```

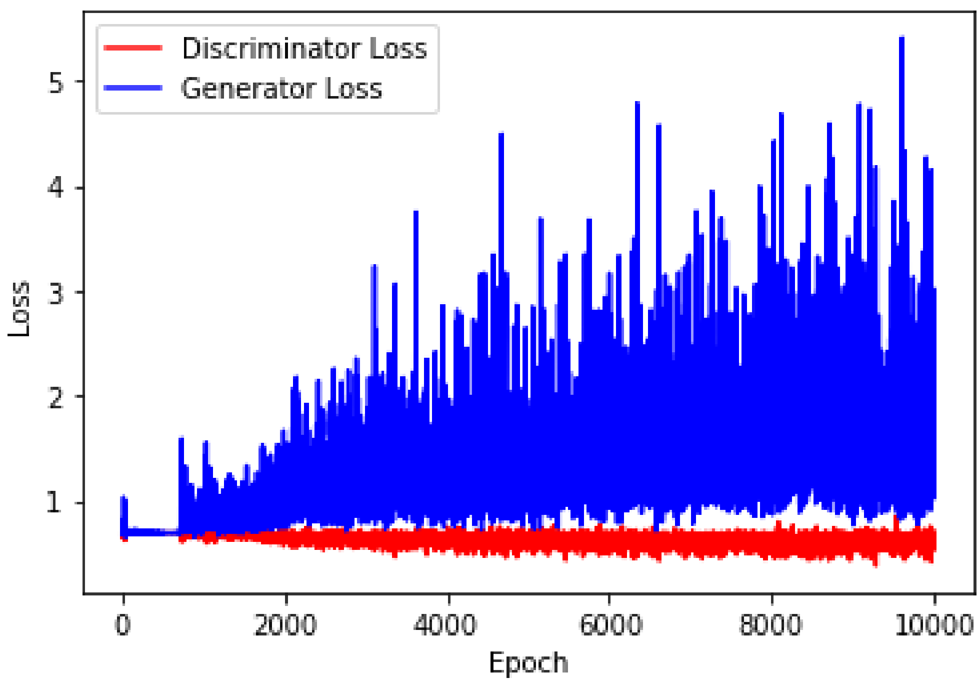
2 ABC GAN Model

2.0.1 Training the network

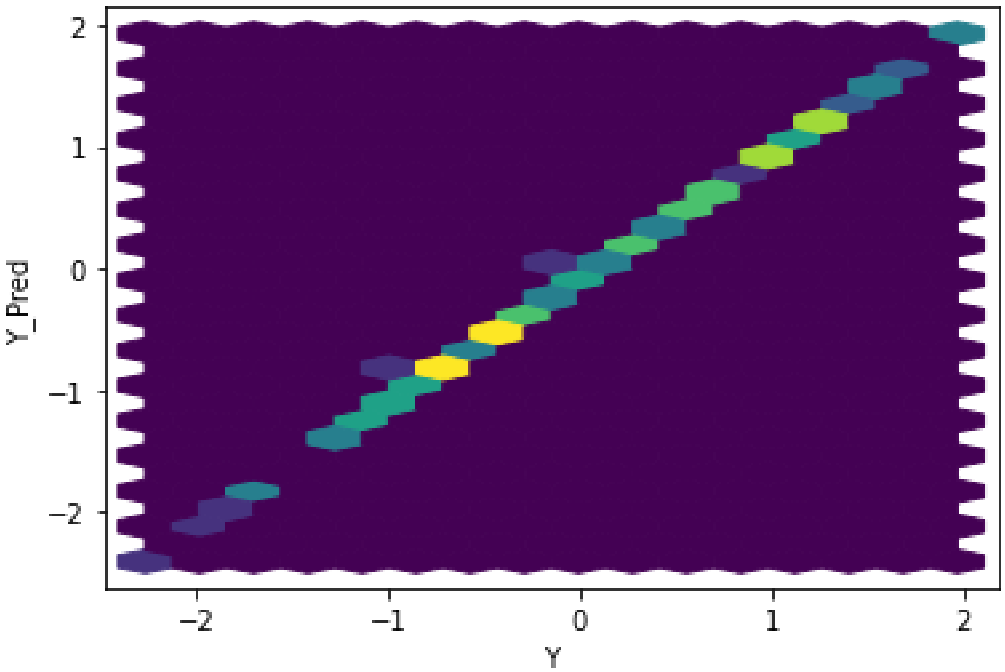
Training ABC-GAN for n_epochs number of epochs

[17]:
```
gen = network.Generator(n_features+2)
disc = network.Discriminator(n_features+2)

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))
```

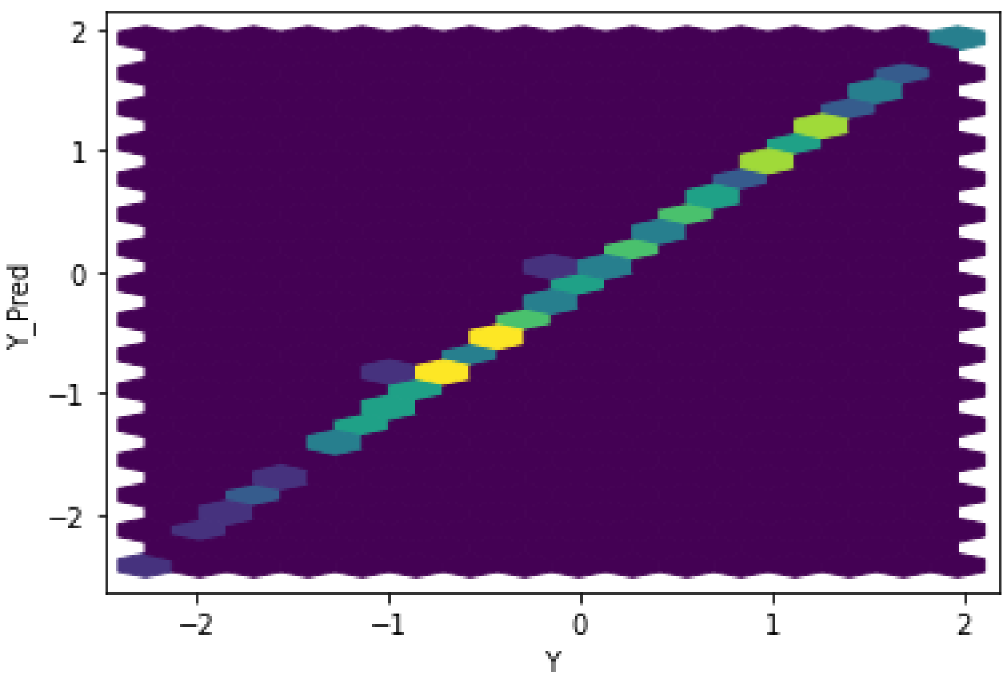
[18]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
 ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```



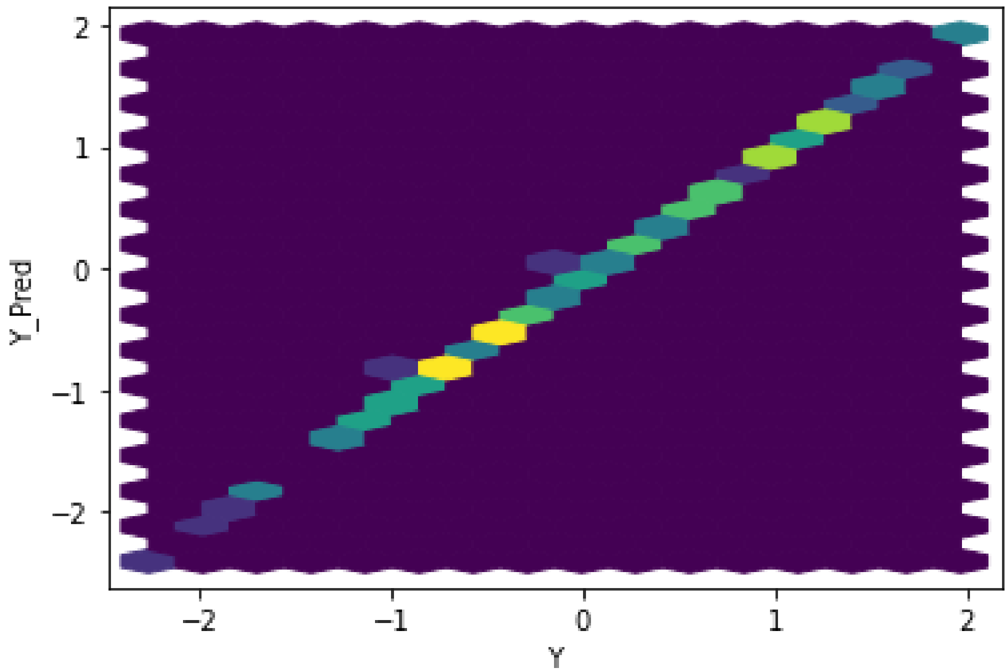
[19]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```



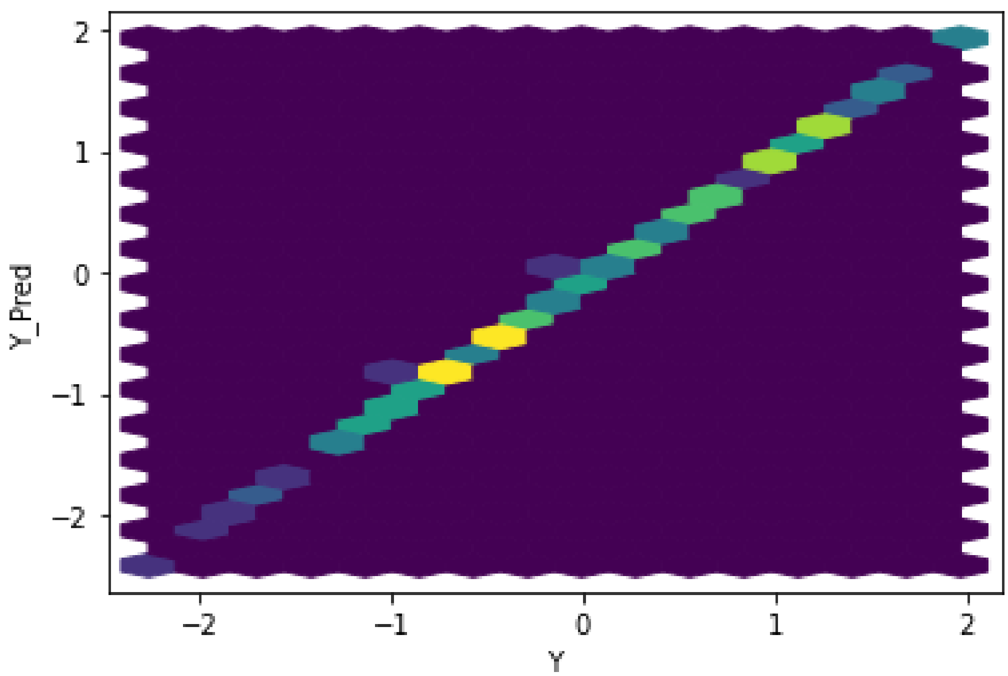
2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
Y



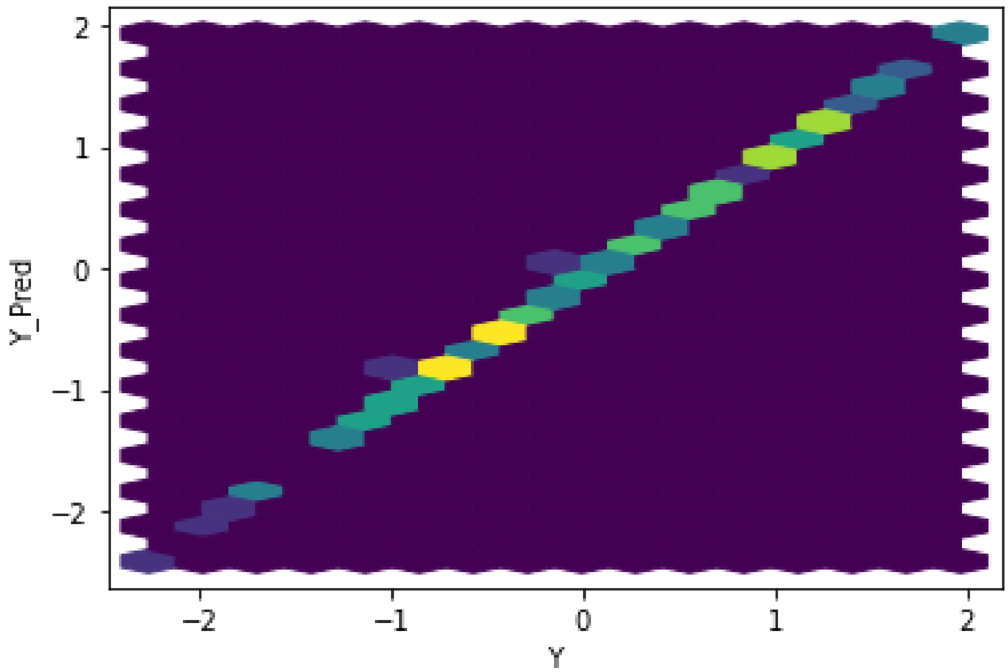
2
1
0
−1
−2
Y_Pred
−2
−1
0
1
2
Y

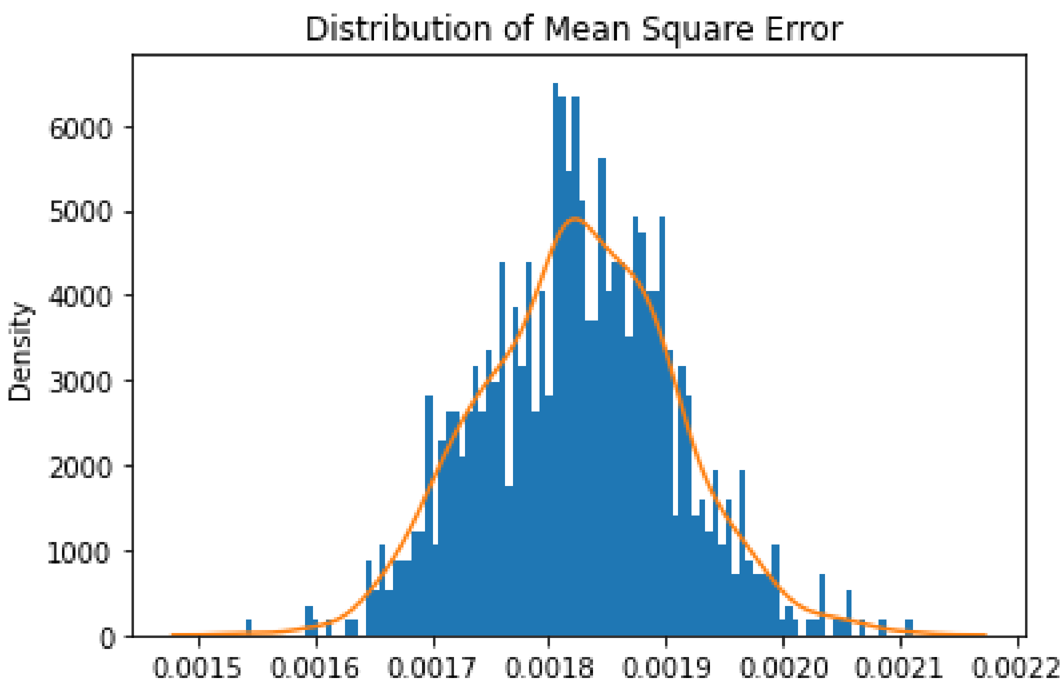


Y_Pred
Y
2
1
0
−1
−2
−2
−1
0
1
2

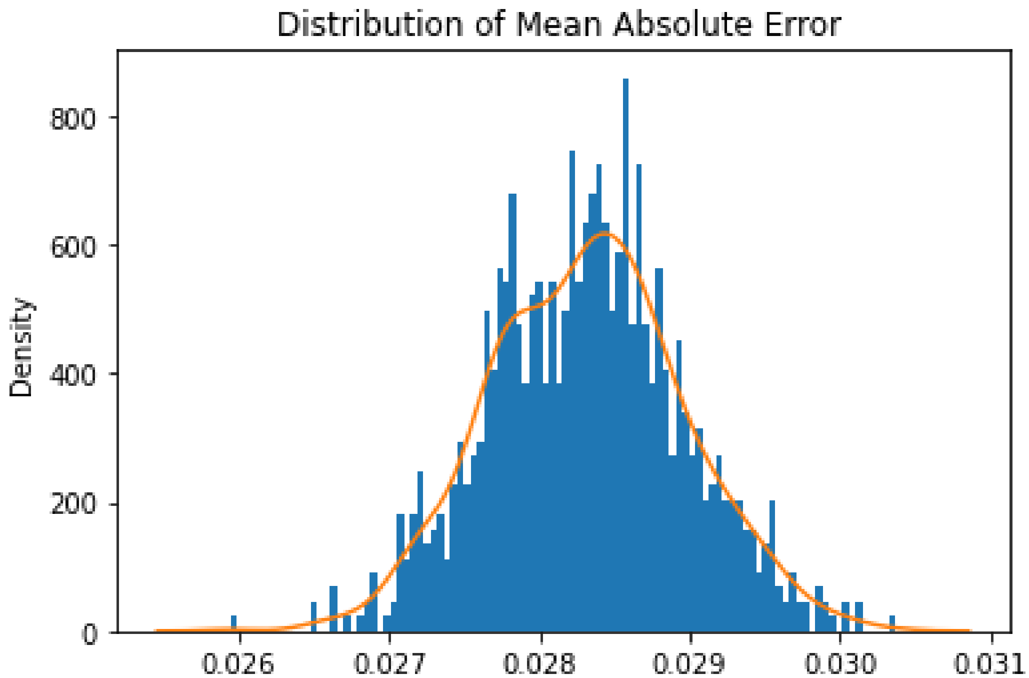


Y_Pred
Y
2
1
0
−1
−2
−2
−1
0
1
2

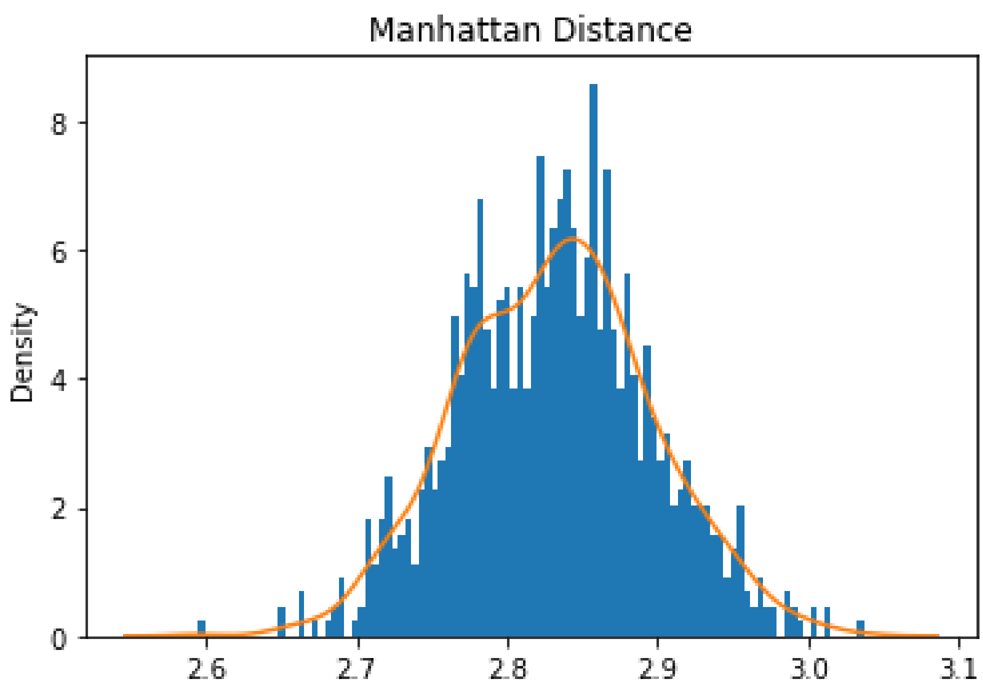




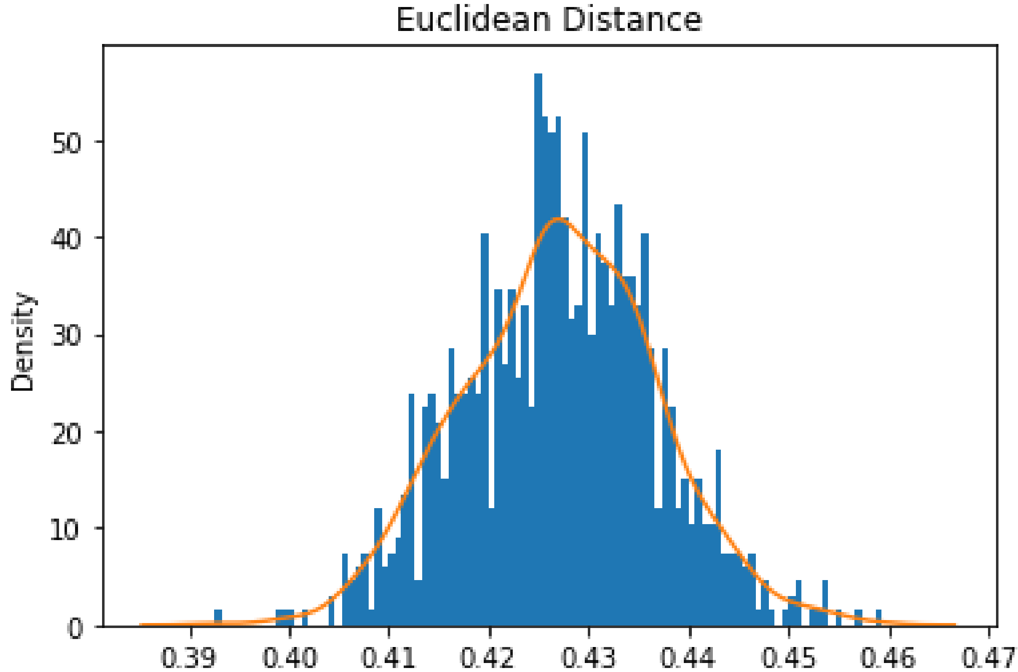

```
Mean Square Error: 0.0018251475964612166
```




```
Mean Absolute Error: 0.02831745514757931
Mean Manhattan Distance: 2.831745514757931
```

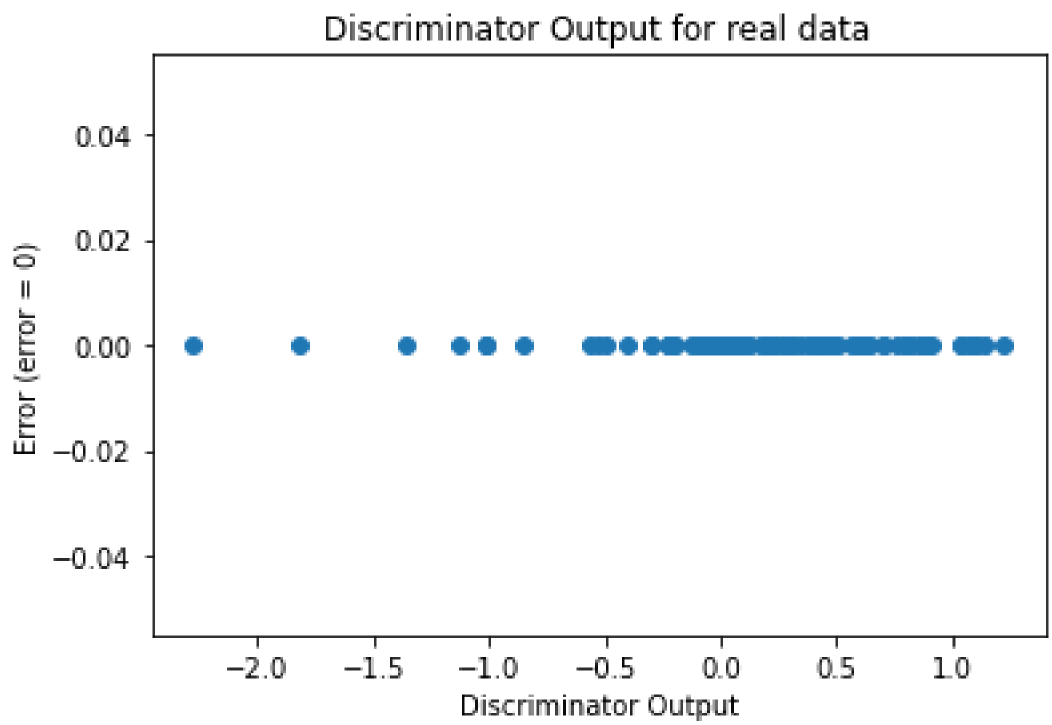


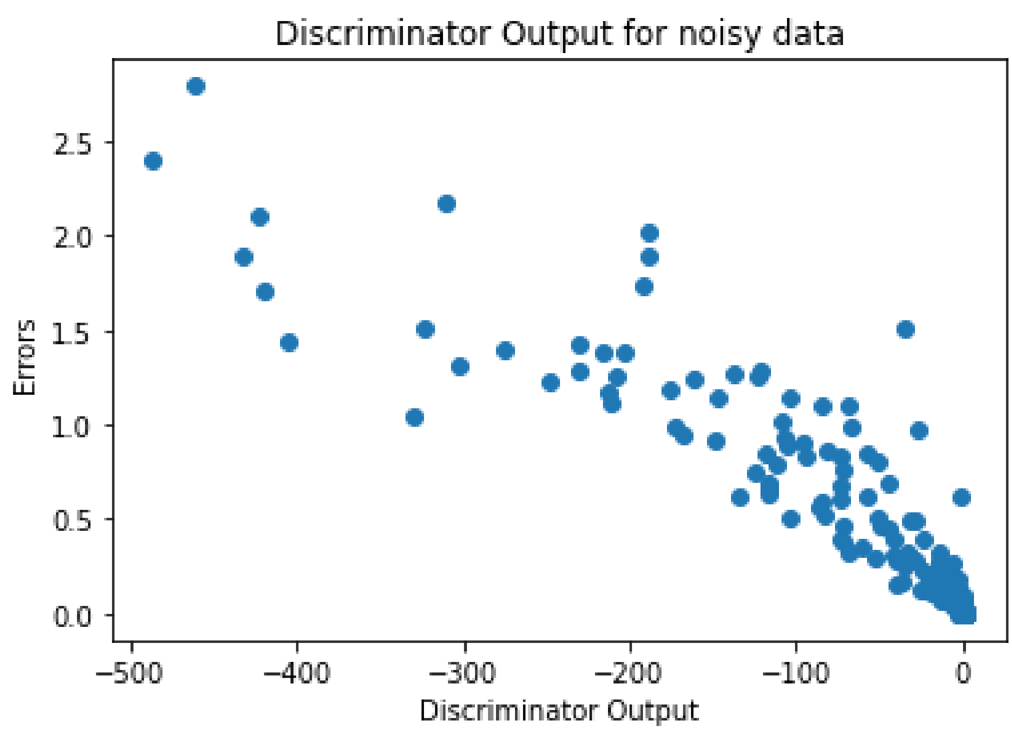
```
Mean Euclidean Distance: 0.4271082790935983
```



Sanity Checks

[20]:
```
sanityChecks.discProbVsError(real_dataset,disc,device)
```


Discriminator Output for real data
Error (error = 0)
0.04
0.02
0.00
−0.02
−0.04
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Discriminator Output

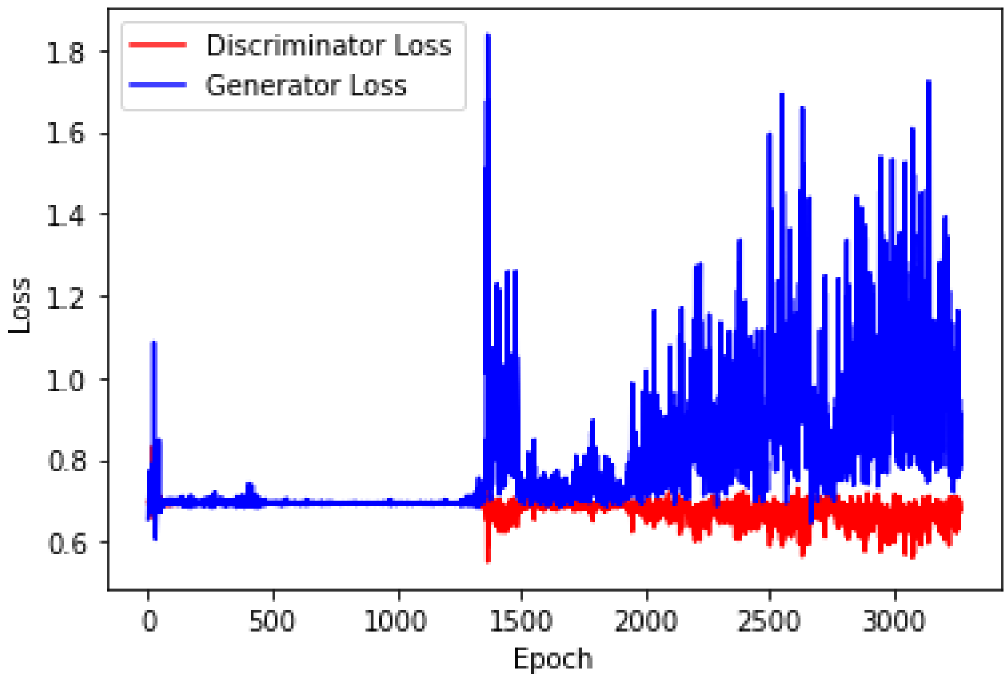

Discriminator Output for noisy data
Errors
2.5
2.0
1.5
1.0
0.5
0.0
−500
−400
−300
−200
−100
0
Discriminator Output

Training GAN until mse of y__pred is > 0.1 or n__epochs < 30000

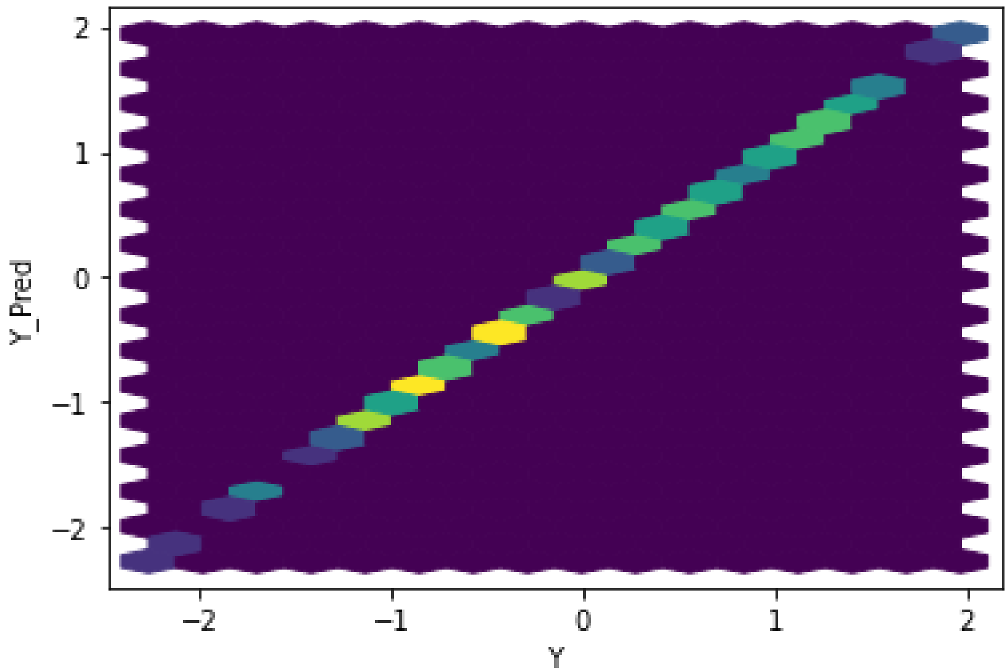
[21]:
```
gen = network.Generator(n_features+2)
disc = network.Discriminator(n_features+2)

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))
```

[22]:
```
ABC_train_test.
↪training_GAN_2(disc,gen,disc_opt,gen_opt,real_dataset,batch_size,␣
↪error,criterion,coeff,mean,variance,device)
```

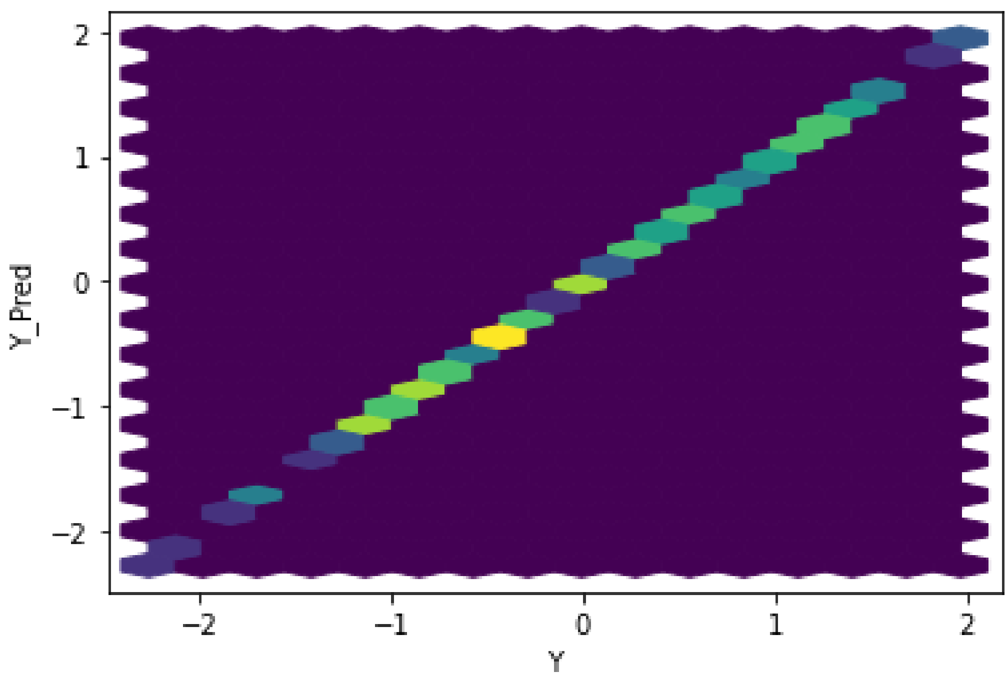
```
Number of epochs 1635
```



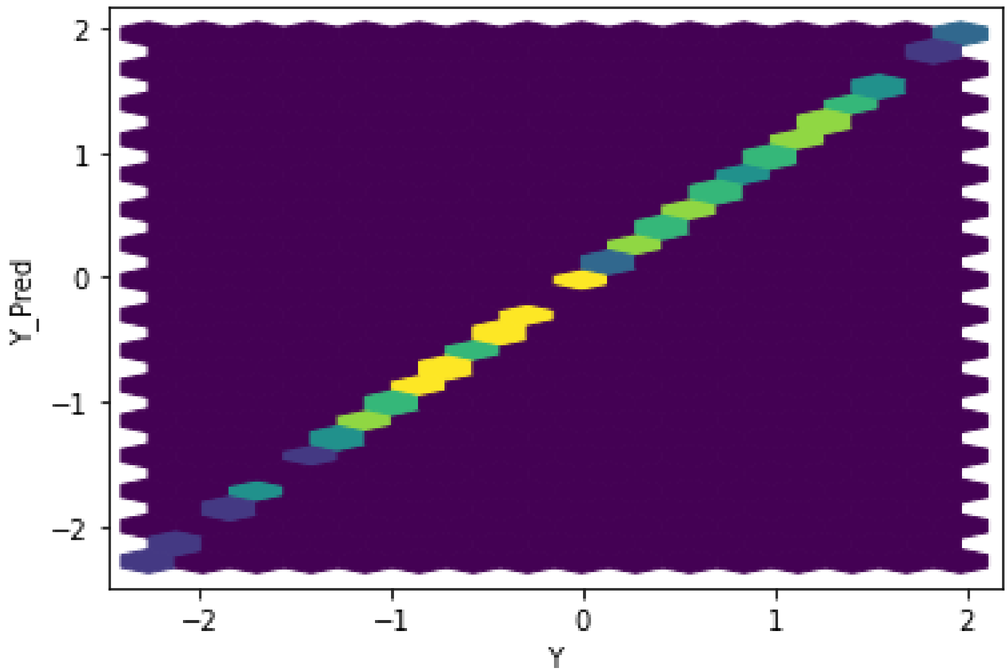
[23]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

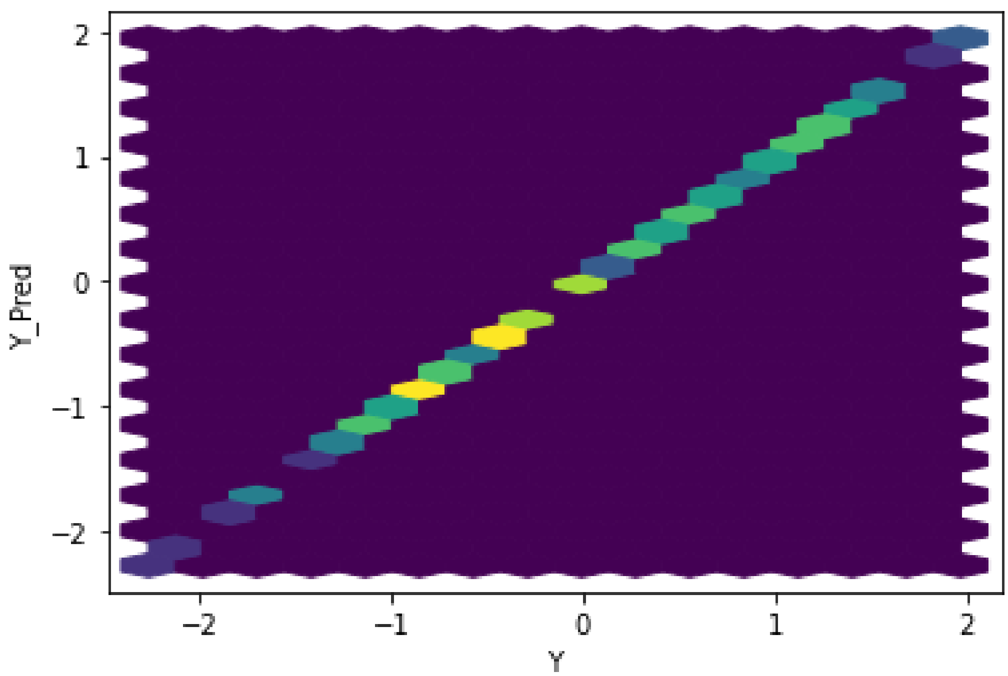


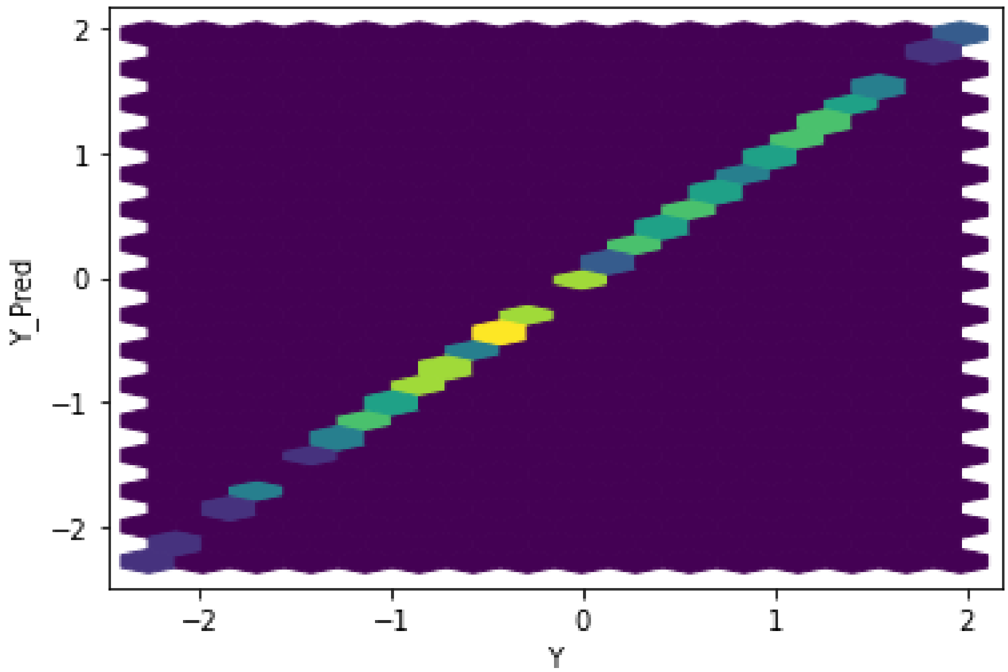
Y_Pred
Y

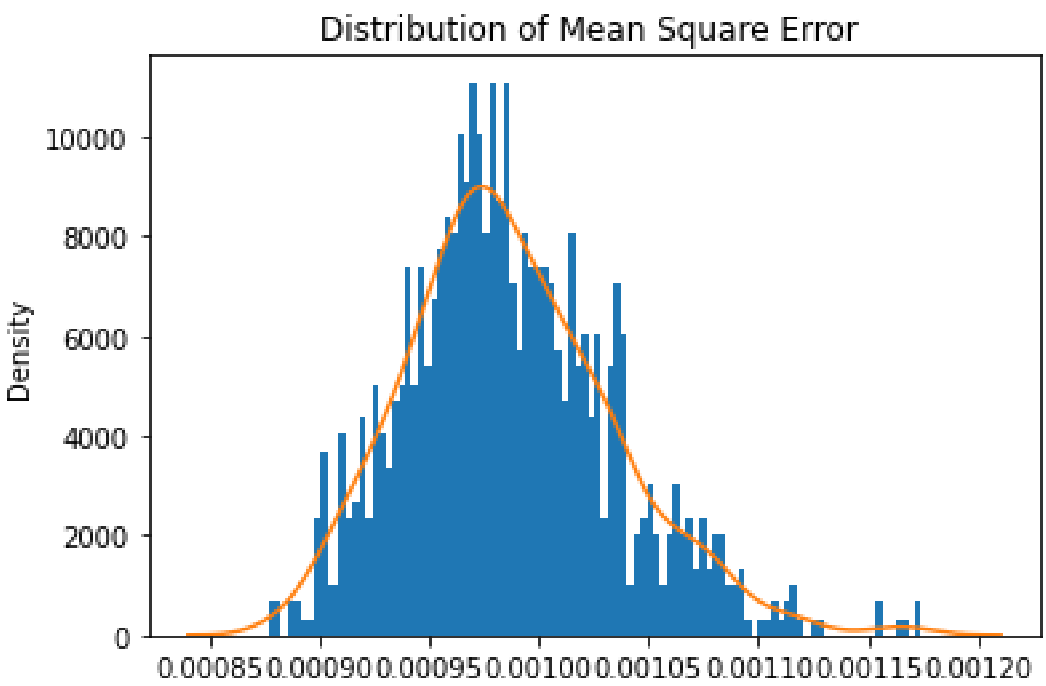


Y_Pred
Y

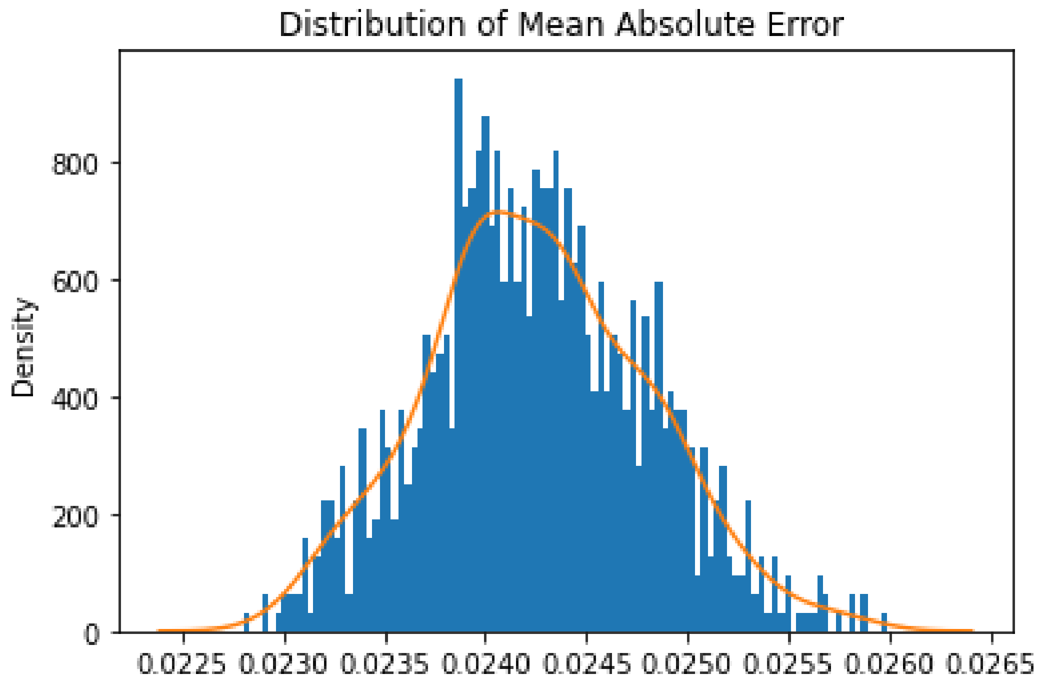

Y_Pred
Y
2
1
0
−1
−2
−2
−1
0
1
2


Y_Pred
Y
2
1
0
−1
−2
−2
−1
0
1
2

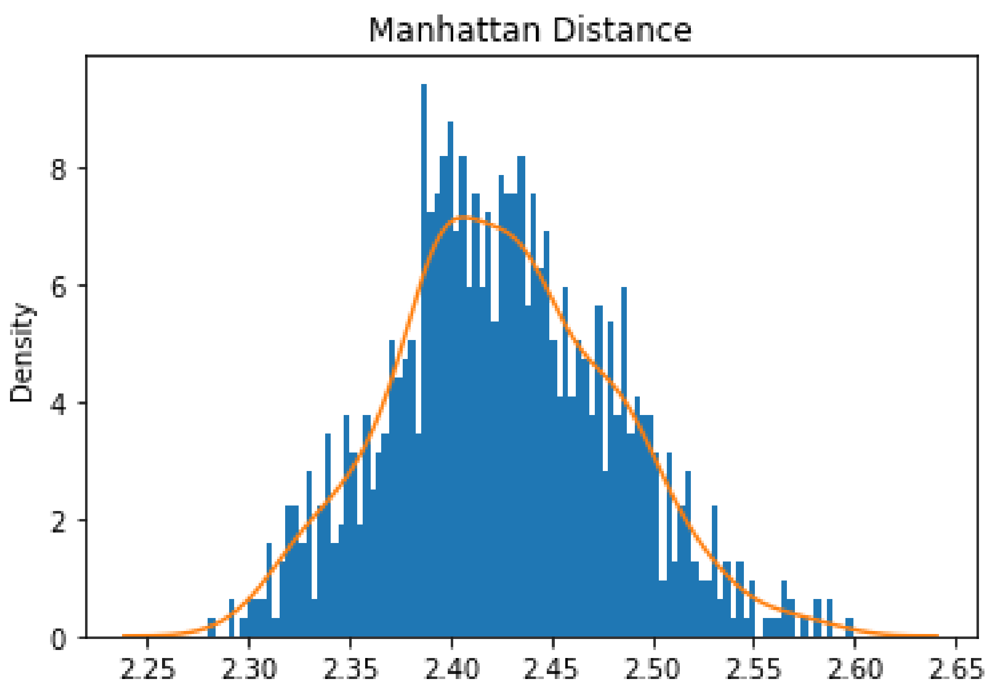




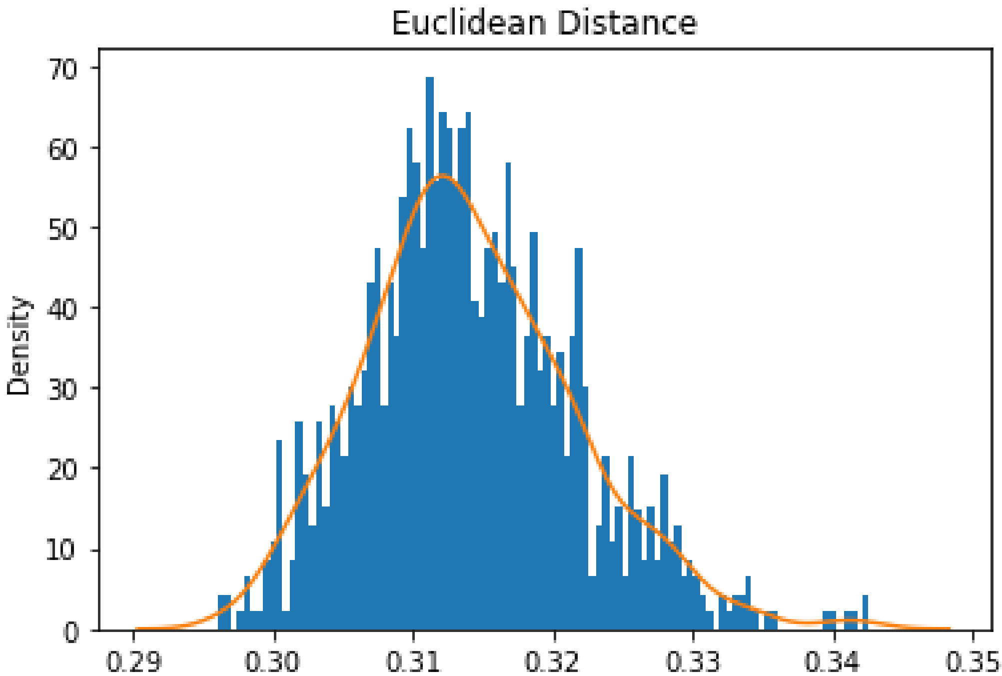

```
Mean Square Error: 0.0009868834737008357
```




```
Mean Absolute Error: 0.024253760970644654
Mean Manhattan Distance: 2.4253760970644653
```



```
Mean Euclidean Distance: 0.3140553868548683
```




```
[ ]:
```