Dataset2_Friedman1_output_0

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 = 10
n_features = 10
mean = 1
variance = 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  0.201652  0.271652  0.865756  0.487913  0.952374  0.140608  0.710875
1  0.017707  0.936539  0.702572  0.984386  0.653584  0.748357  0.885770
2  0.695478  0.052018  0.964602  0.092810  0.359608  0.017609  0.165994
```

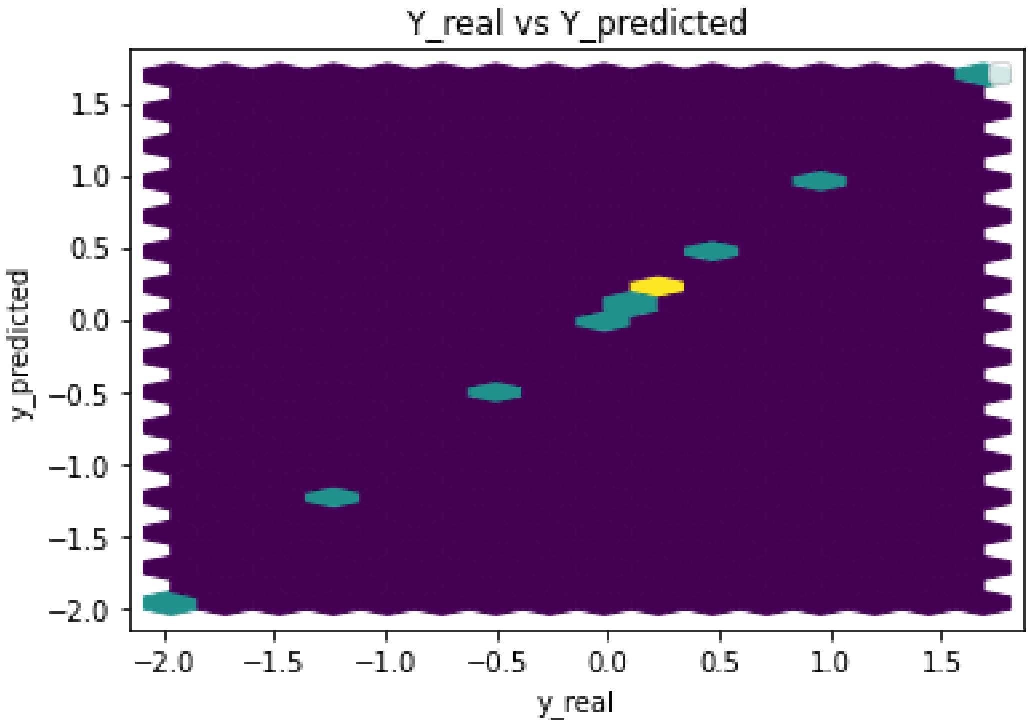
```
3  0.818823  0.633385  0.134007  0.066667  0.392892  0.650759  0.524594
4  0.635803  0.210224  0.343699  0.587949  0.406391  0.803966  0.480173

         X7        X8        X9          Y
0  0.733148  0.817787  0.854646  14.012819
1  0.402372  0.108709  0.757517  14.374033
2  0.570334  0.740412  0.477038   8.058322
3  0.748148  0.727863  0.034443  15.075098
4  0.987012  0.950208  0.185677  12.504245
```

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:                       1.000
Model:                            OLS   Adj. R-squared:                    nan
Method:                 Least Squares   F-statistic:                       nan
Date:                Wed, 20 Oct 2021   Prob (F-statistic):                nan
Time:                        19:49:19   Log-Likelihood:                 326.15
No. Observations:                  10   AIC:                            -632.3
Df Residuals:                       0   BIC:                            -629.3
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -6.106e-16        inf         -0        nan         nan         nan
x1            -0.7675        inf         -0        nan         nan         nan
x2             0.9628        inf          0        nan         nan         nan
x3            -0.7283        inf         -0        nan         nan         nan
x4             0.1929        inf          0        nan         nan         nan
x5             0.5119        inf          0        nan         nan         nan
x6            -0.4034        inf         -0        nan         nan         nan
x7            -0.9737        inf         -0        nan         nan         nan
x8            -0.0044        inf         -0        nan         nan         nan
x9             0.5524        inf          0        nan         nan         nan
x10           -0.1442        inf         -0        nan         nan         nan
==============================================================================
Omnibus:                        2.557   Durbin-Watson:                   1.203
Prob(Omnibus):                  0.278   Jarque-Bera (JB):                0.957
Skew:                          -0.205   Prob(JB):                        0.620
Kurtosis:                       1.541   Cond. No.                         12.0
==============================================================================

Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The input rank is higher than the number of observations.
Parameters:  const   -6.106227e-16
x1      -7.674823e-01
x2       9.628043e-01
x3      -7.282937e-01
x4       1.929266e-01
x5       5.118685e-01
x6      -4.033538e-01
x7      -9.736886e-01
x8      -4.398004e-03
x9       5.524257e-01
x10     -1.442205e-01
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 2.7455912842438604e-30
Mean Absolute Error: 1.4106771306643396e-15
Manhattan distance: 1.4106771306643395e-14
Euclidean distance: 5.2398390092099784e-15
```

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')
```

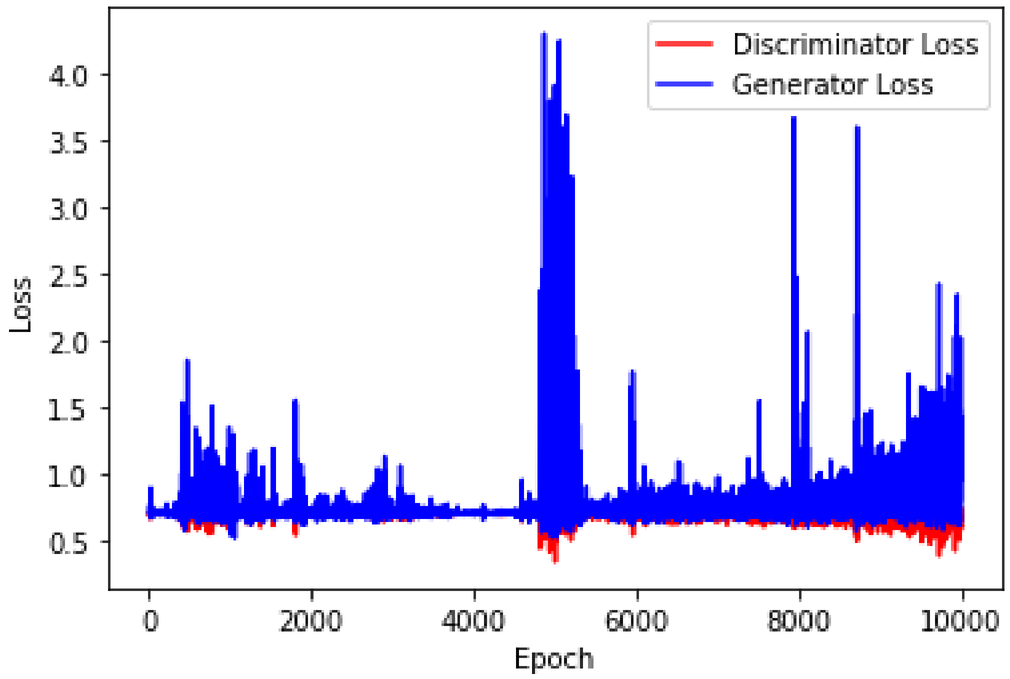
Training GAN for n_epochs number of epochs

[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))
```

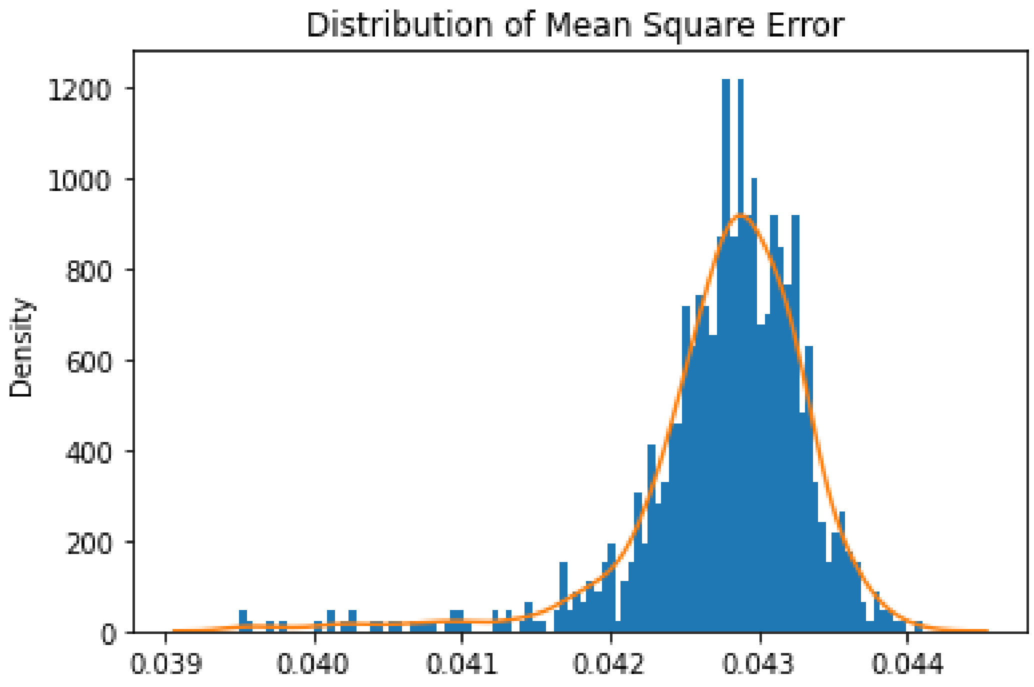
[10]:
```
print(generator)
print(discriminator)
```

```
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()
)
```

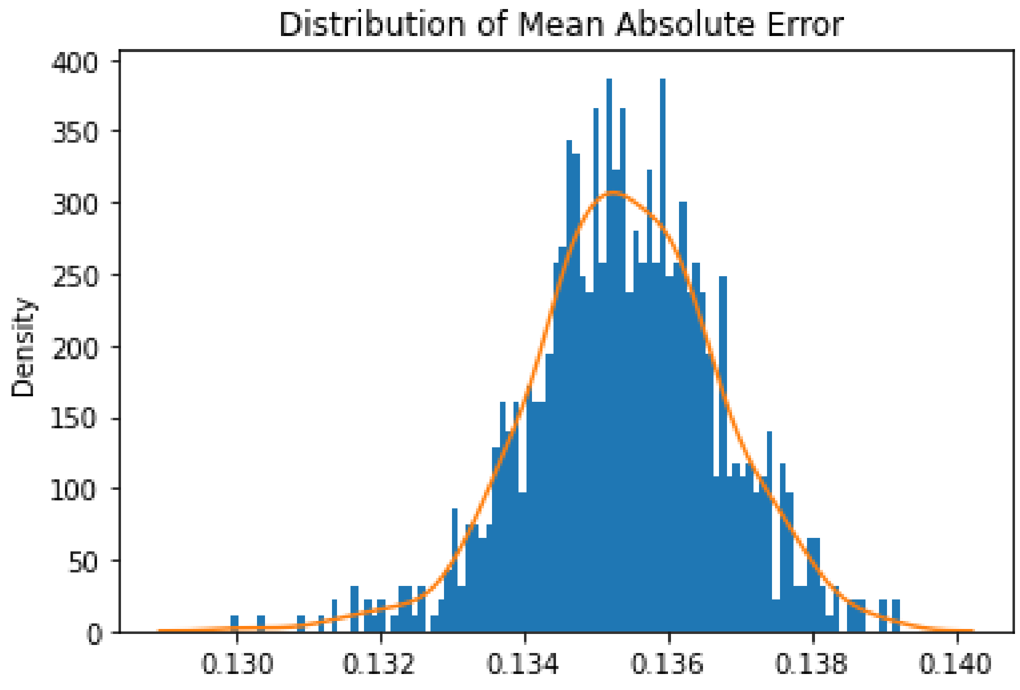
[11]:
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



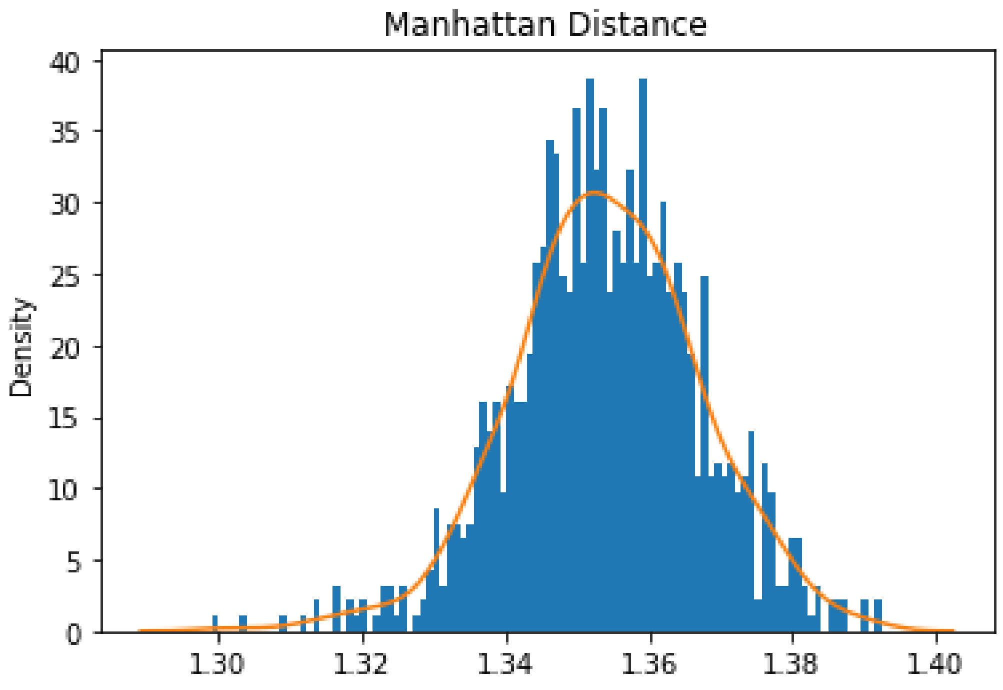
[12]:
```
train_test.test_generator(generator,real_dataset,device)
```



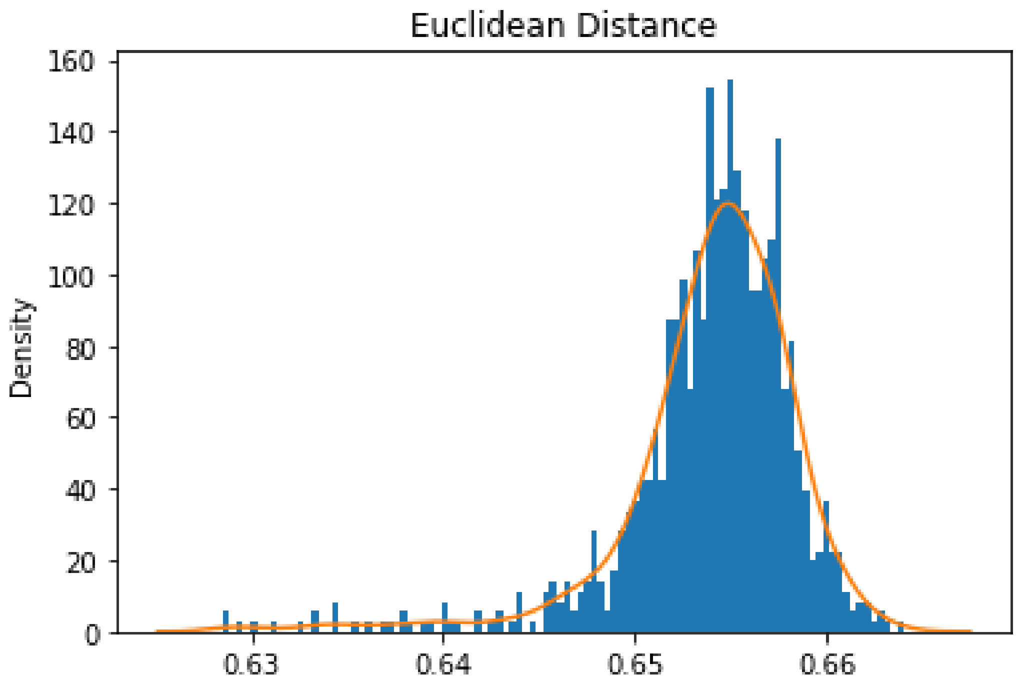
```
Mean Square Error: 0.04277566197967325
```



```
Mean Absolute Error: 0.13540636349767446
```

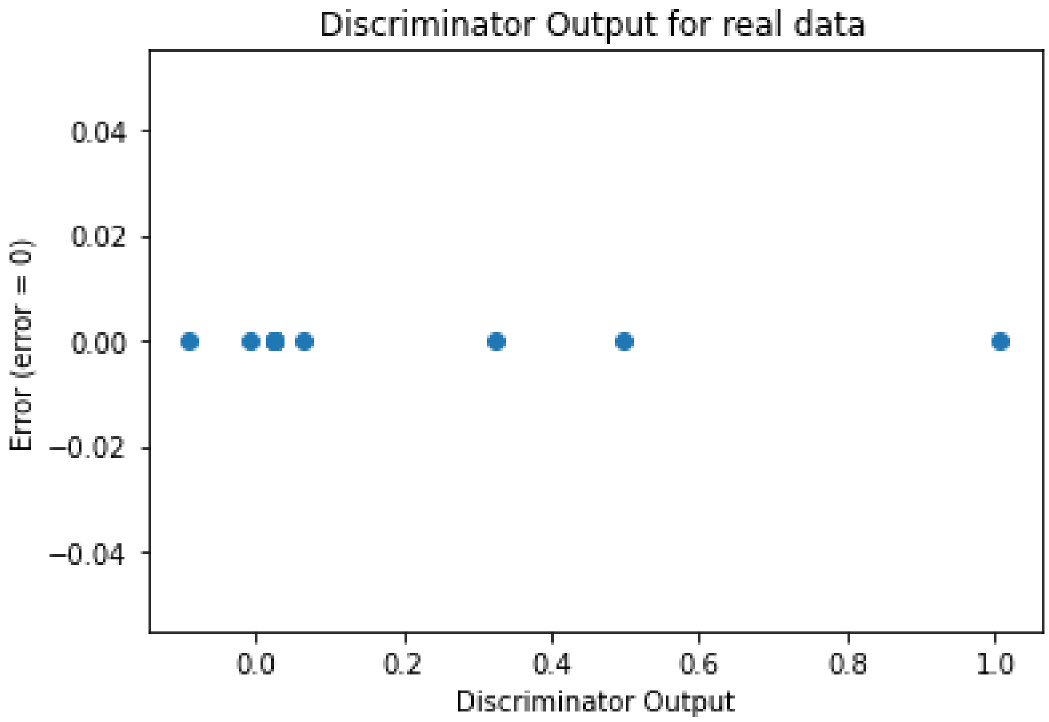


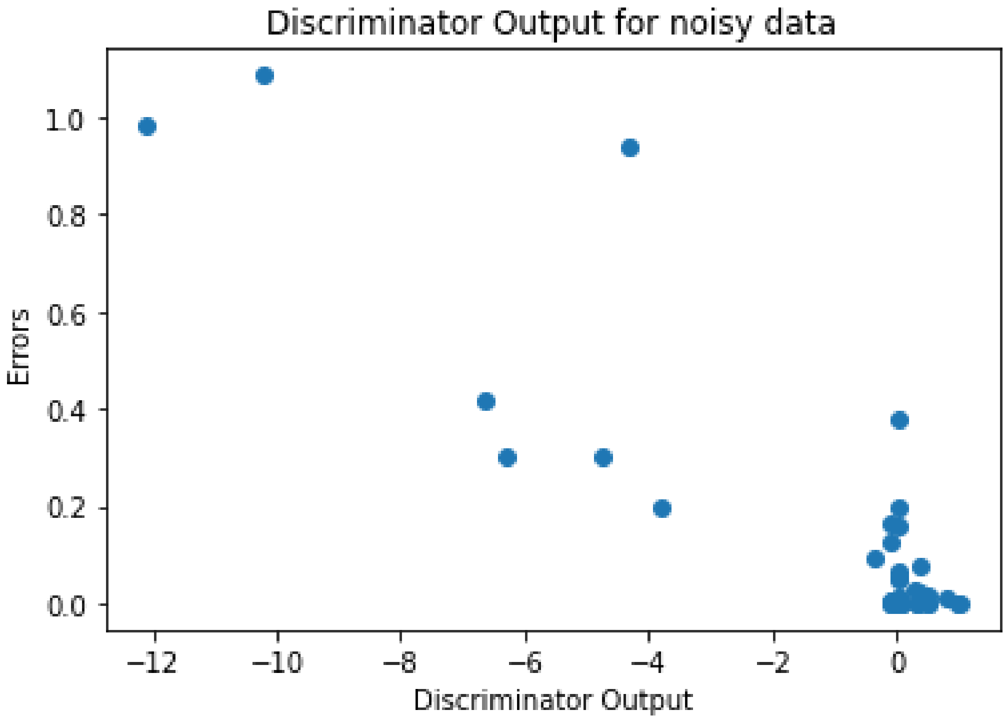
```
Mean Manhattan Distance: 1.3540636349767448
```



```
Mean Euclidean Distance: 0.654015545455313
```

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


Discriminator Output for real data
Error (error = 0)
Discriminator Output
0.04
0.02
0.00
−0.02
−0.04
0.0
0.2
0.4
0.6
0.8
1.0

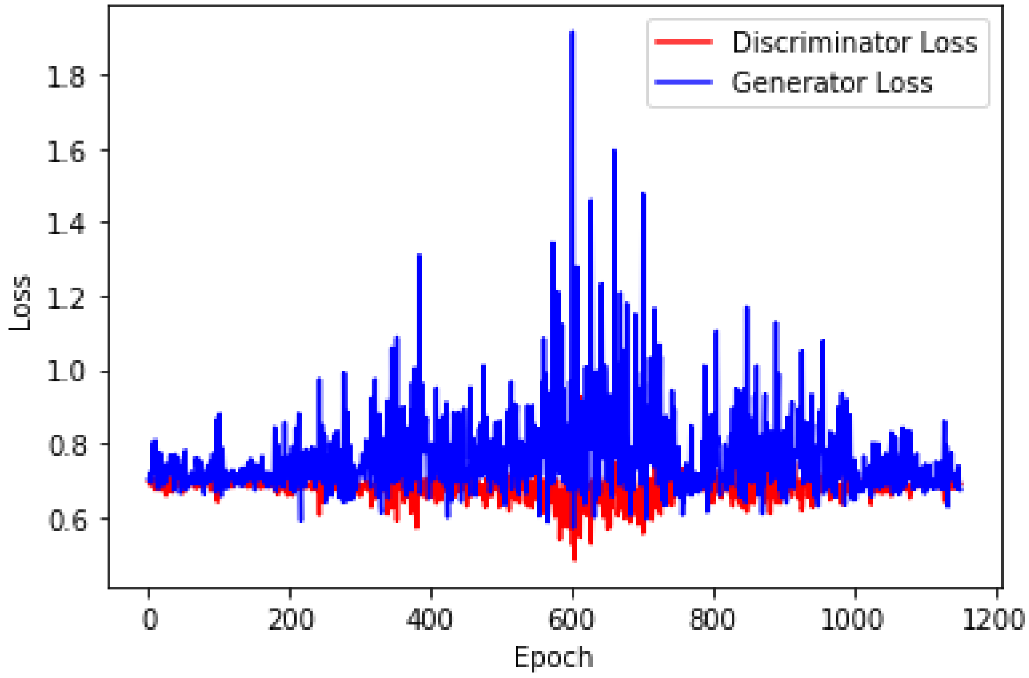

Discriminator Output for noisy data
Errors
Discriminator Output
1.0
0.8
0.6
0.4
0.2
0.0
−12
−10
−8
−6
−4
−2
0

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

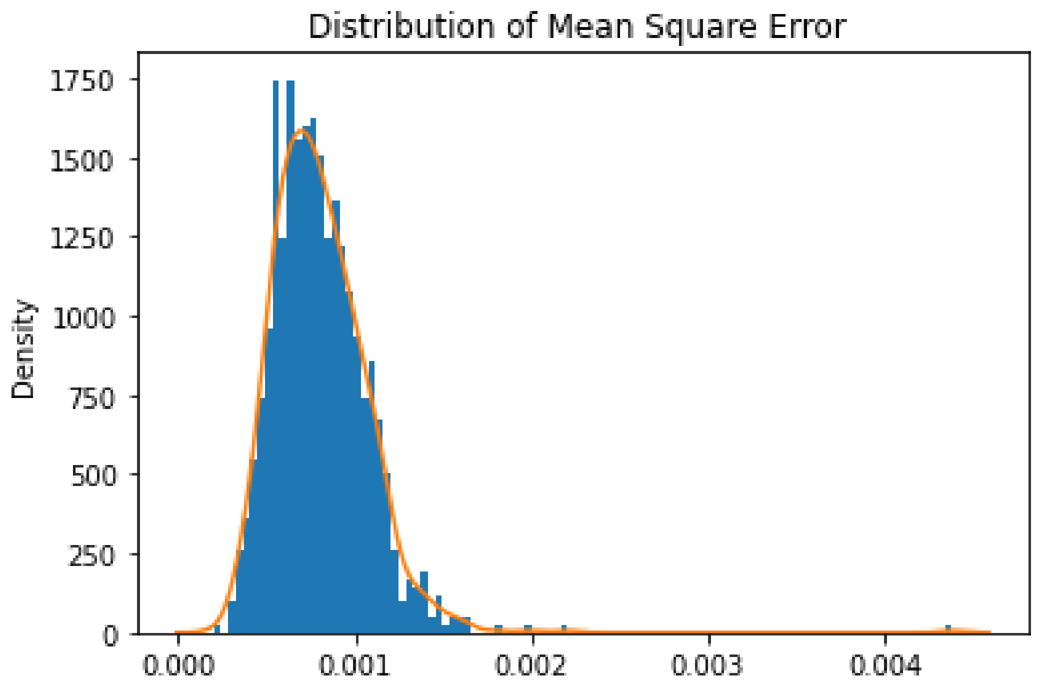
[14]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)
criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5,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

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

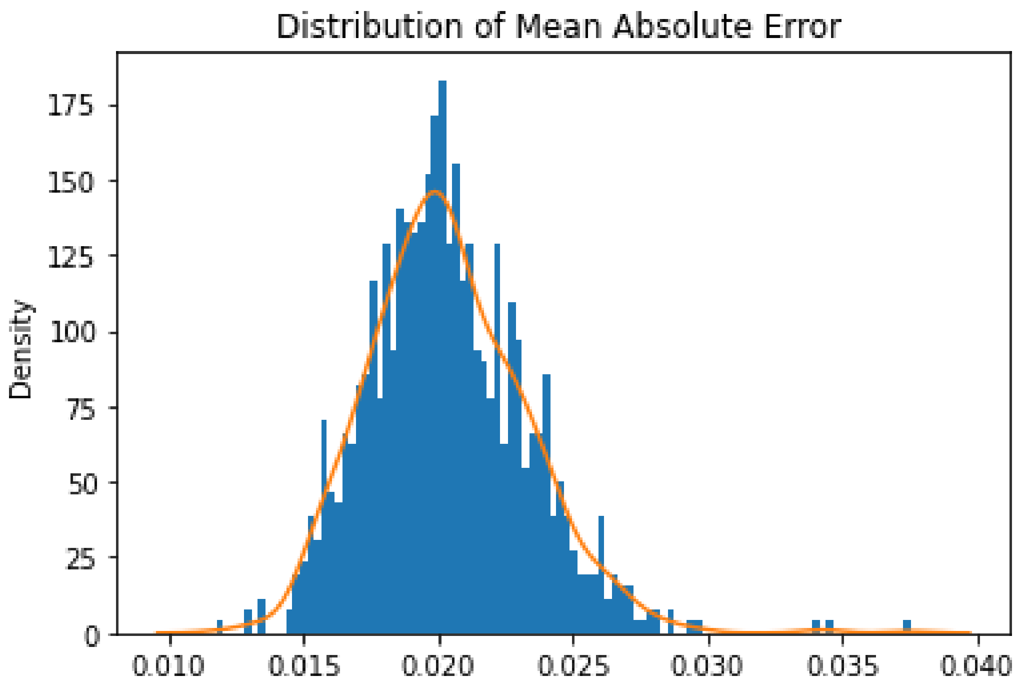
```
Number of epochs needed 576
```



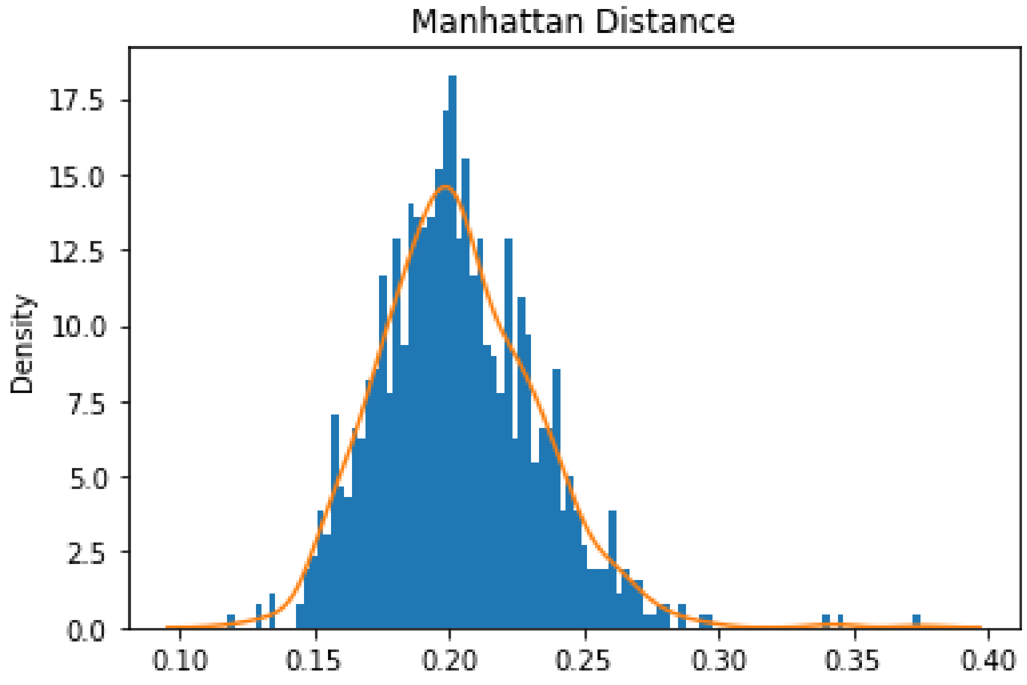
[16]:
```
train_test.test_generator(generator,real_dataset,device)
```



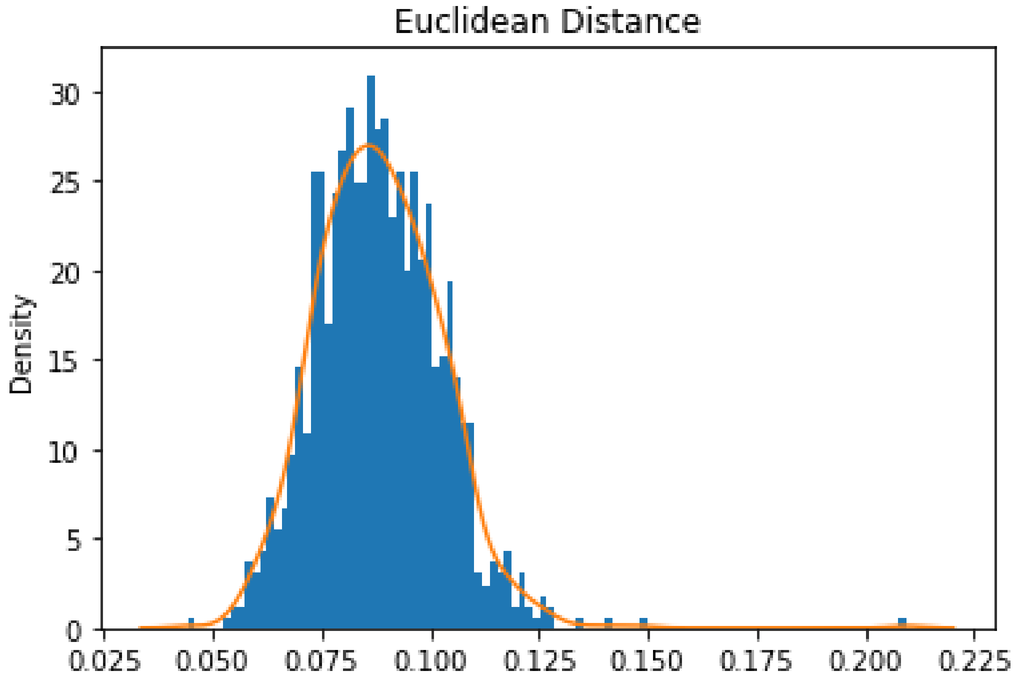
```
Mean Square Error: 0.0008008192685560117
```



```
Mean Absolute Error: 0.020340705460309998
```



```
Mean Manhattan Distance: 0.20340705460309982
```



```
Mean Euclidean Distance: 0.08833452166119447
```

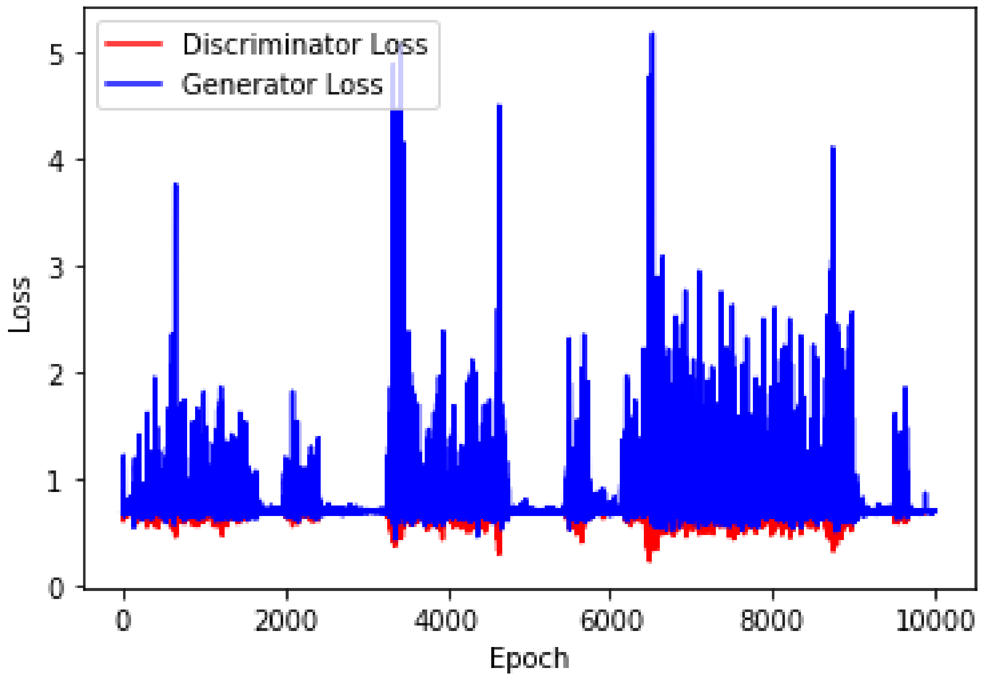
2 ABC GAN Model

2.0.1 Training the network

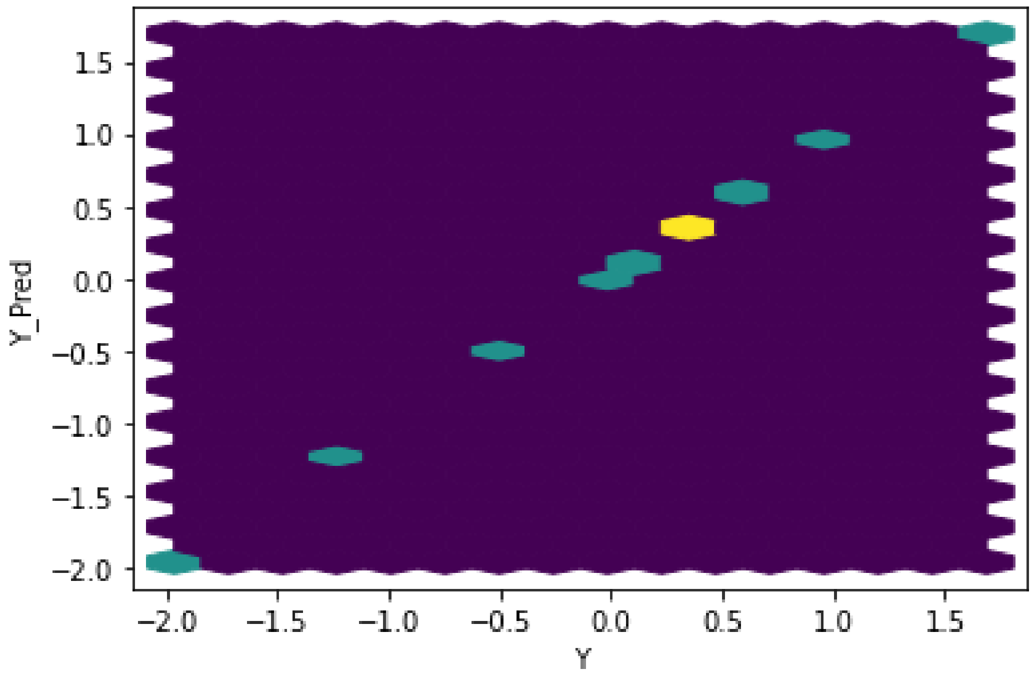
Training ABC-GAN for n_epochs number of epochs

[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))
```

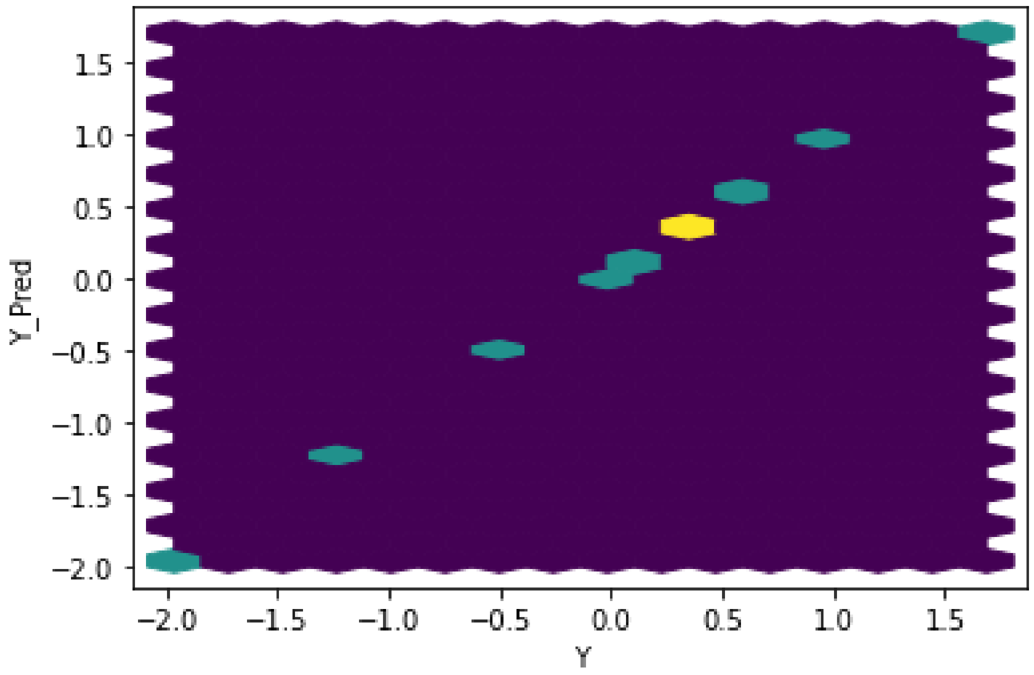
[18]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```



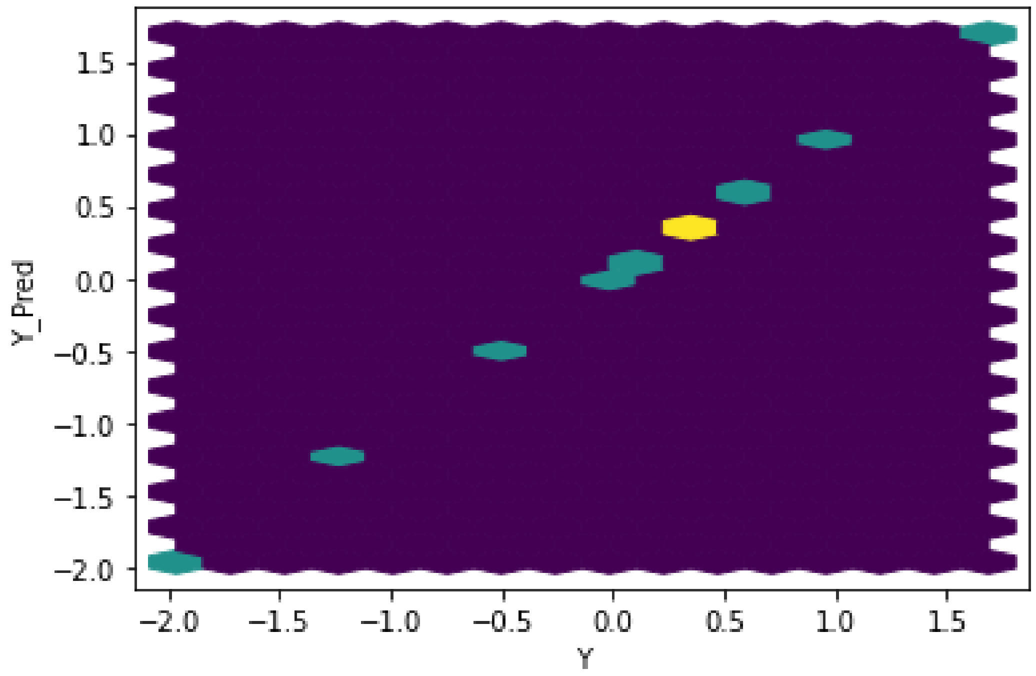
[19]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

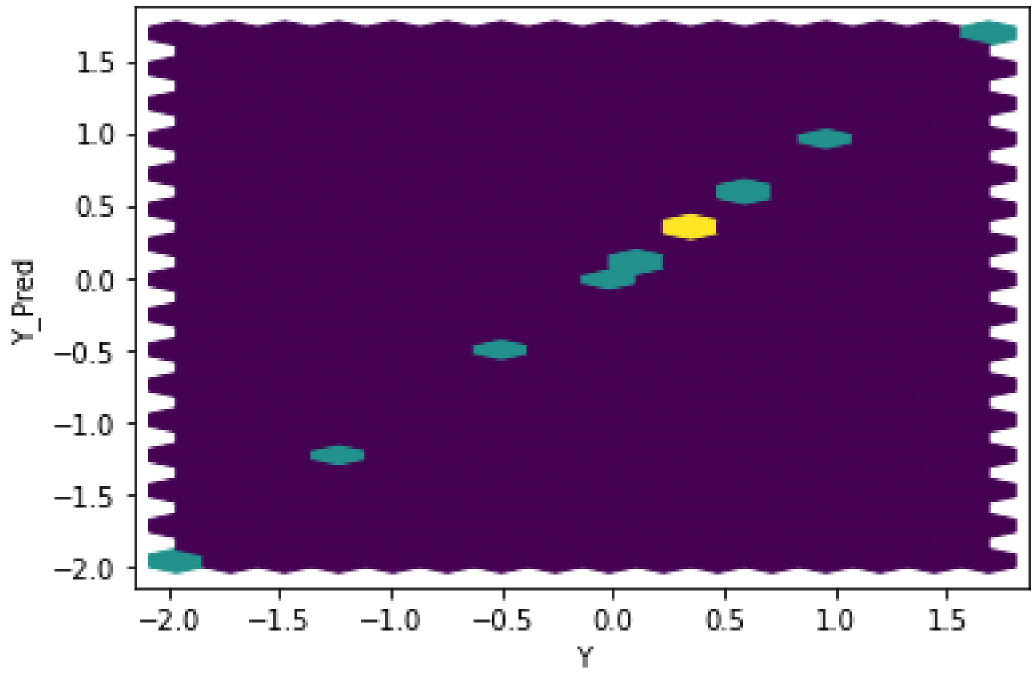


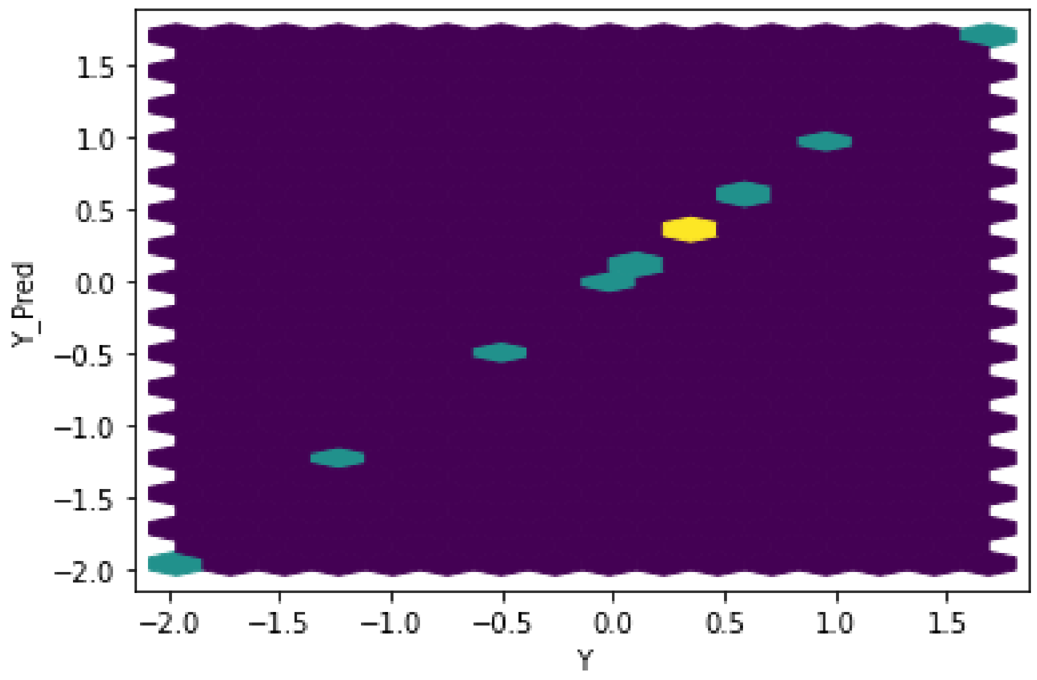
Y_Pred
Y

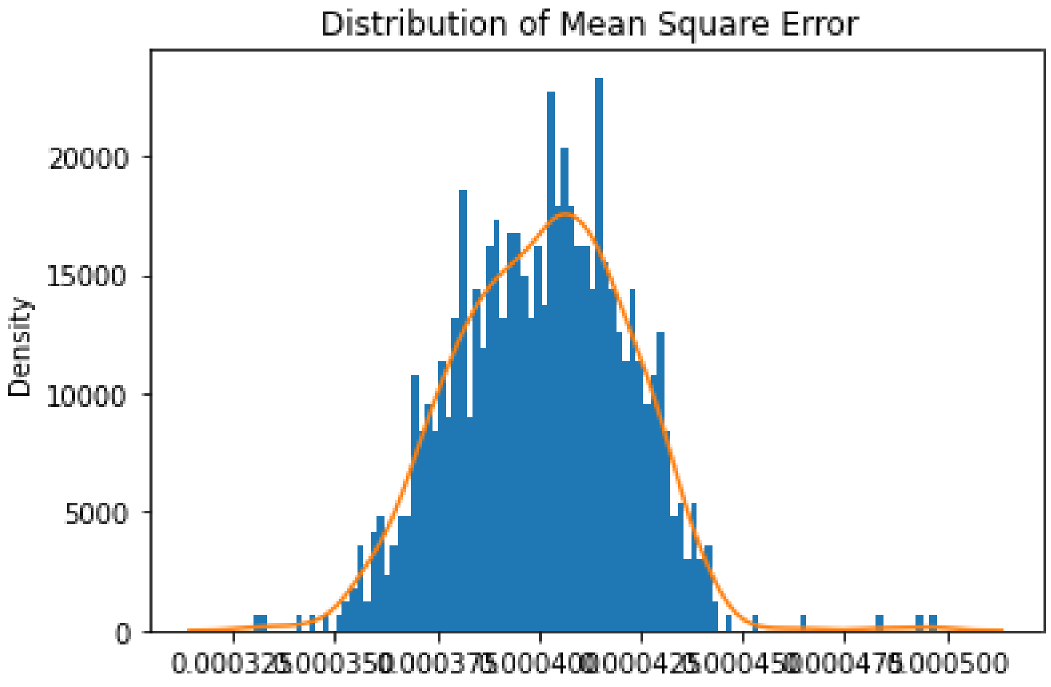


Y_Pred
Y

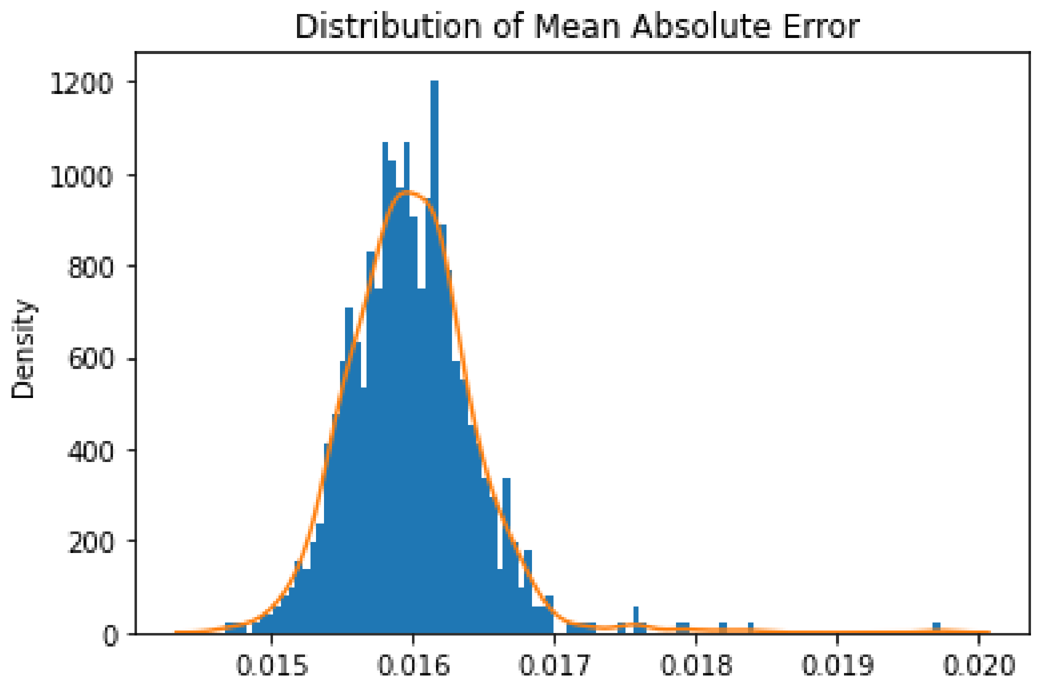

Y_Pred
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Y


Y_Pred
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Y

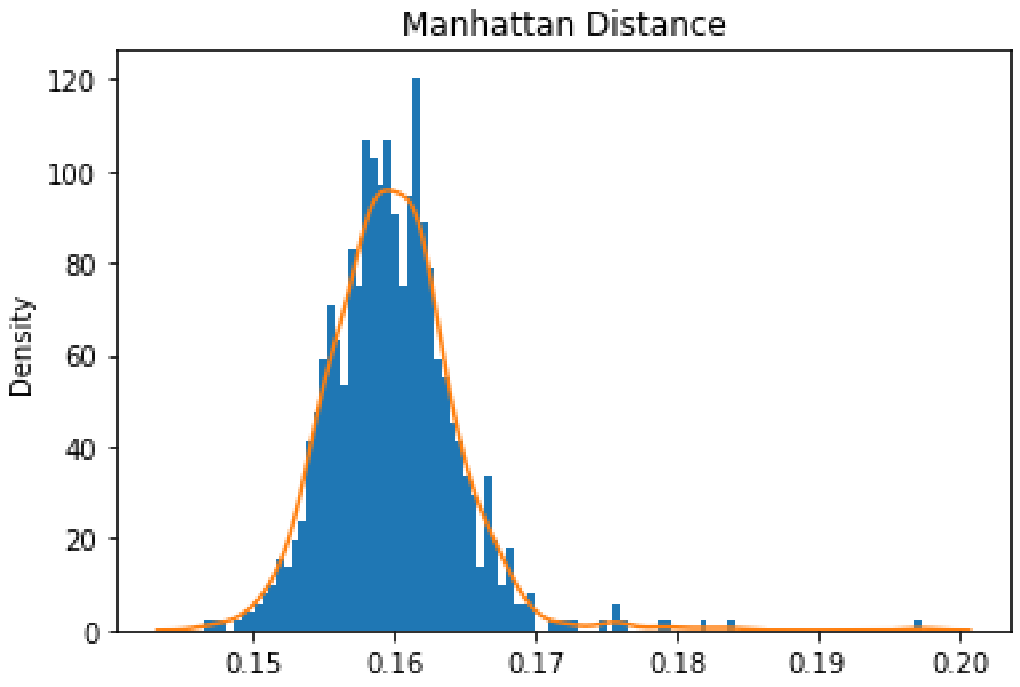




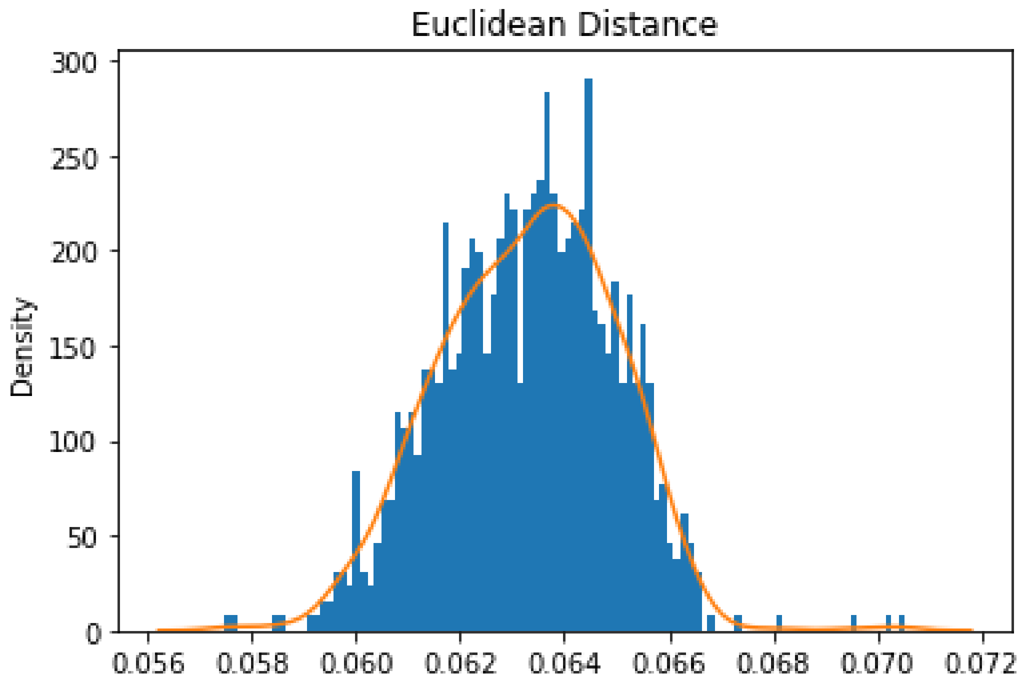
```
Mean Square Error: 0.00040088307450485947
```



```
Mean Absolute Error: 0.015990156817436218
Mean Manhattan Distance: 0.15990156817436219
```

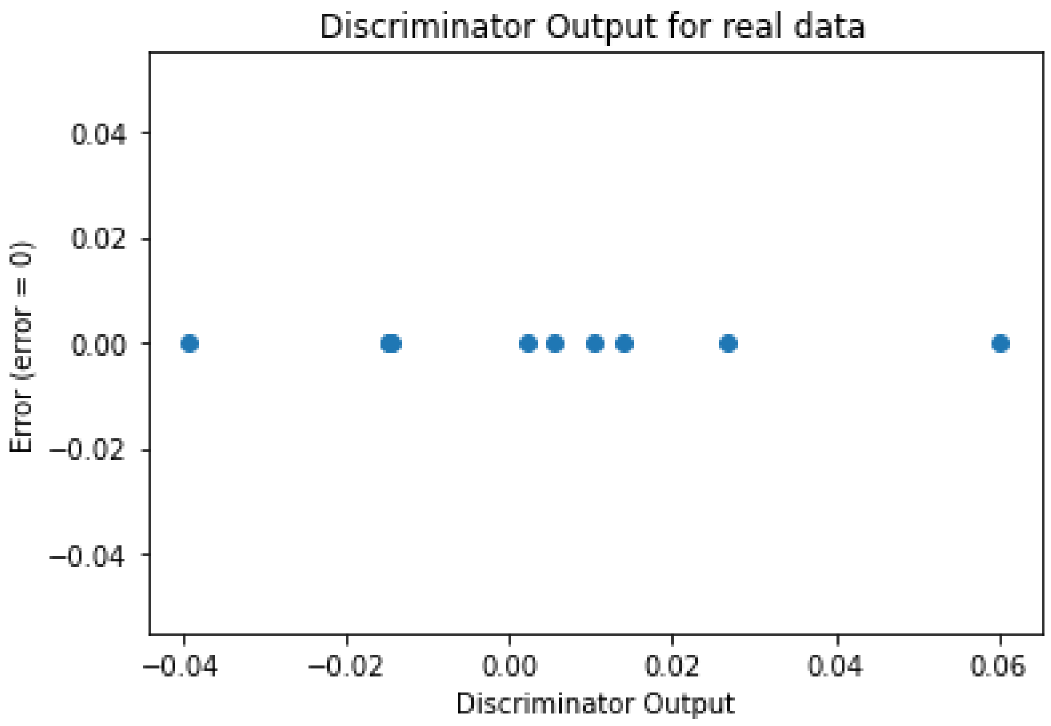


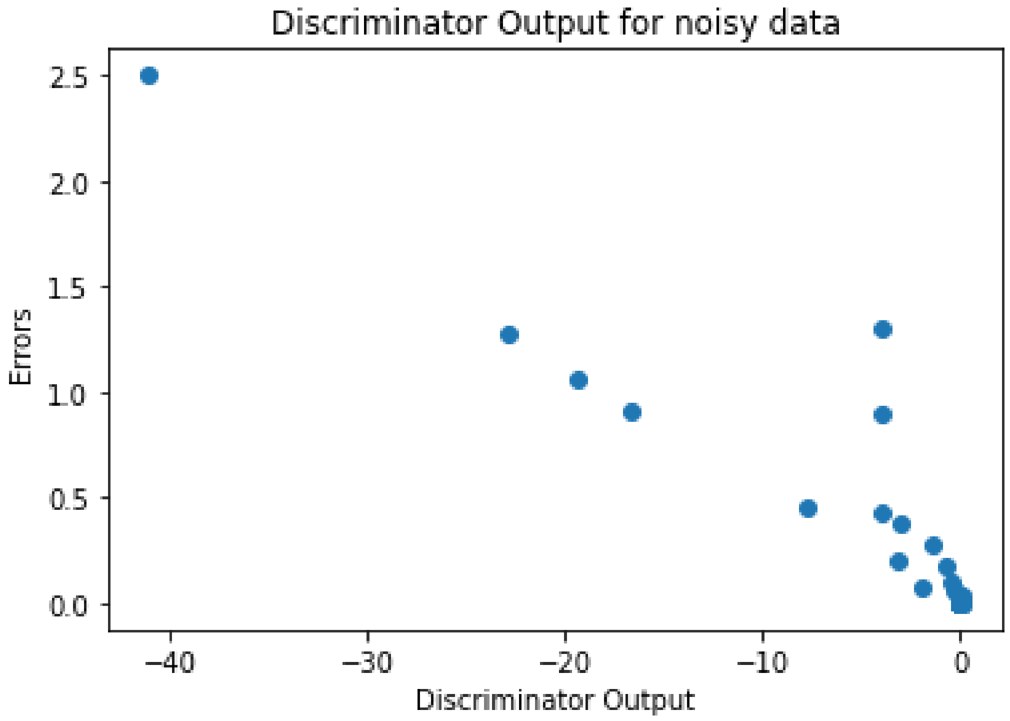
```
Mean Euclidean Distance: 0.06329338961442459
```



Sanity Checks

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


Discriminator Output for real data
Error (error = 0)
Discriminator Output
0.04
0.02
0.00
−0.02
−0.04
0.06

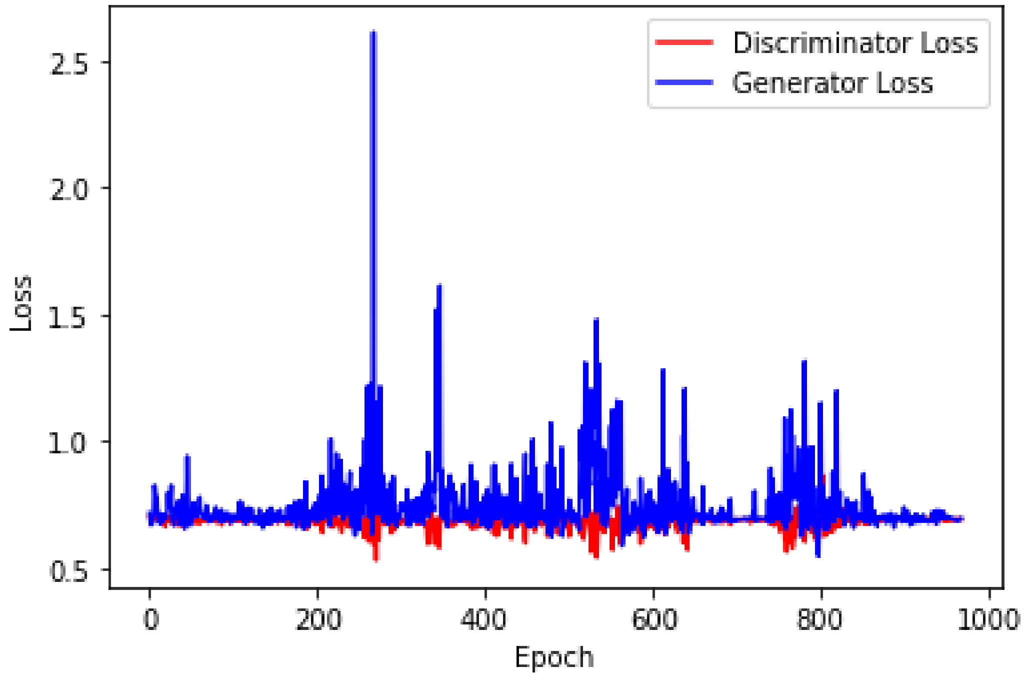

Discriminator Output for noisy data
Errors
Discriminator Output
2.5
2.0
1.5
1.0
0.5
0.0
−40
−30
−20
−10
0

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

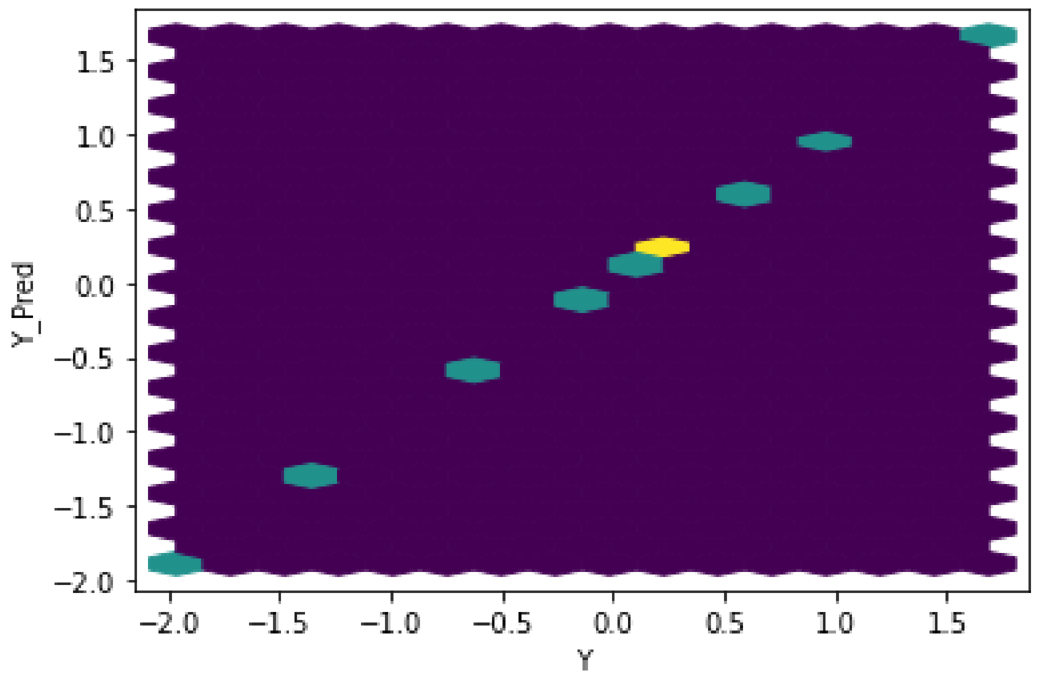
[21]:
```
gen = network.Generator(n_features+2)
disc = network.Discriminator(n_features+2)

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))
```

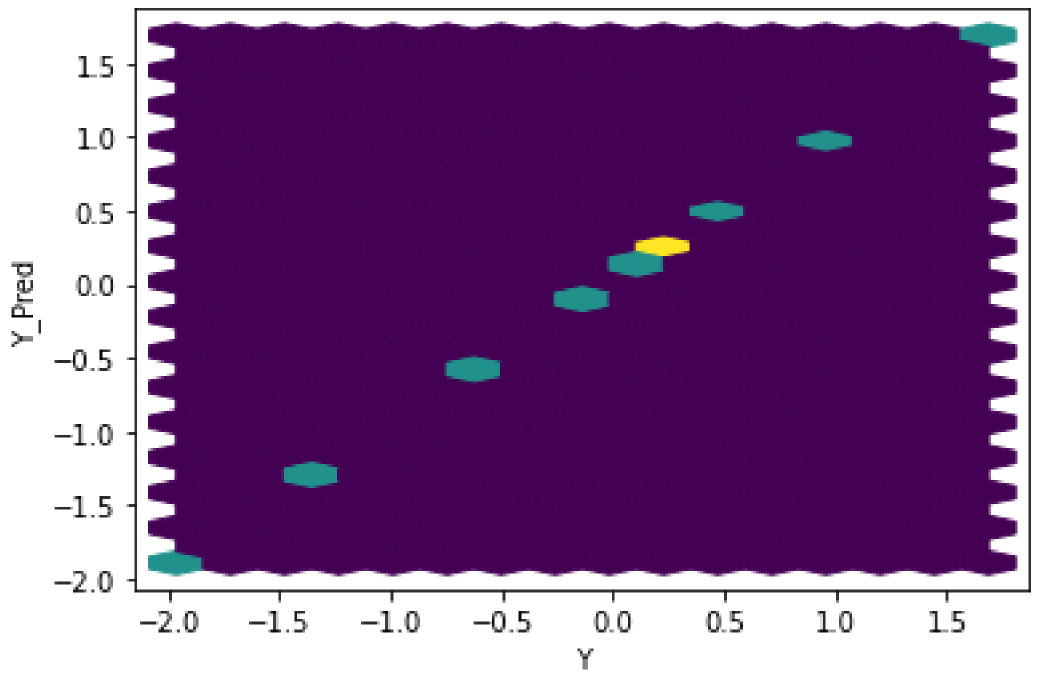
[22]:
```
ABC_train_test.
 ↪training_GAN_2(disc,gen,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪error,criterion,coeff,mean,variance,device)
```

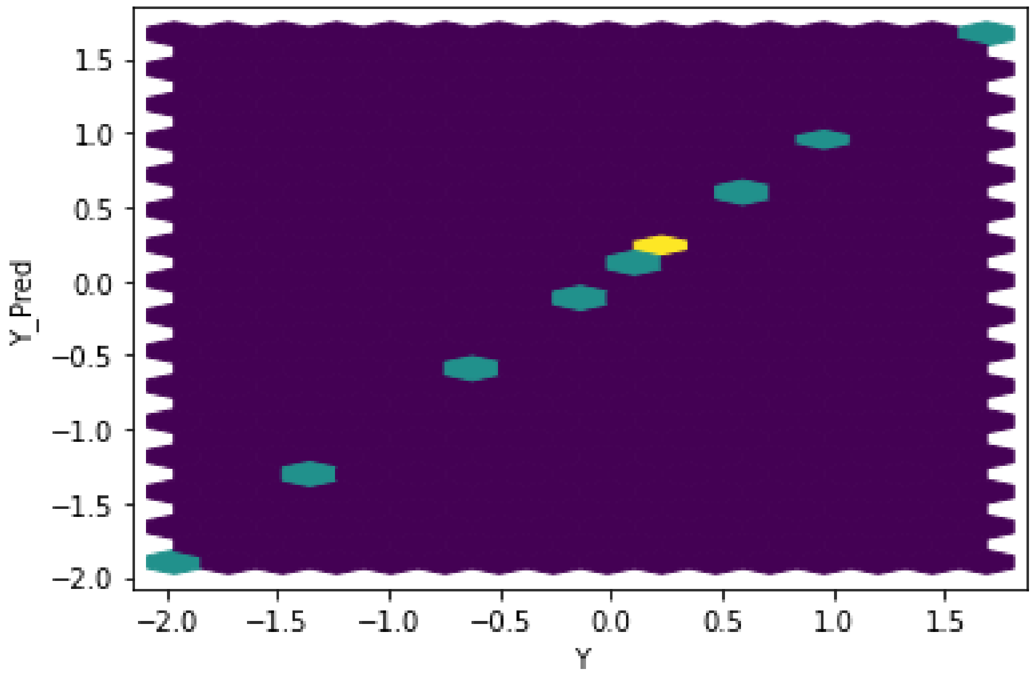
```
Number of epochs 484
```

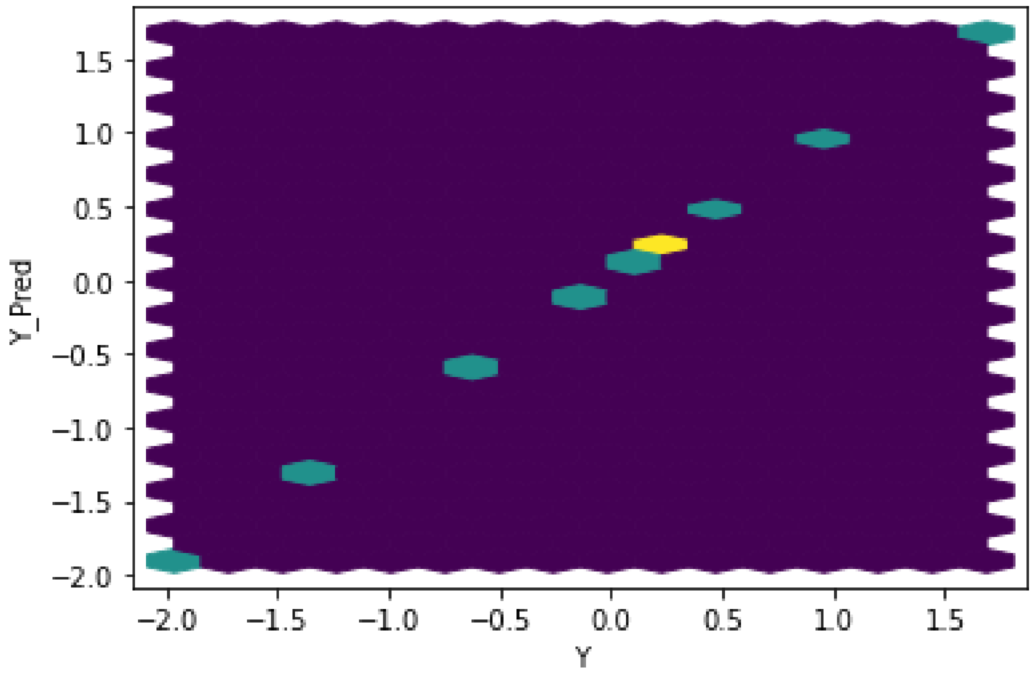


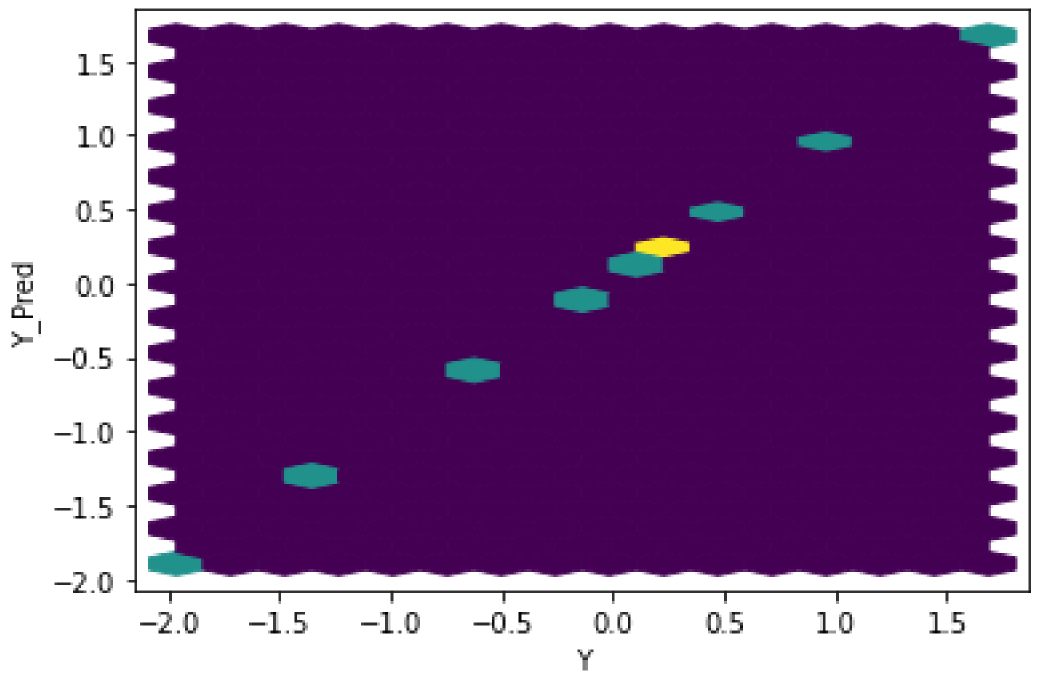
[23]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

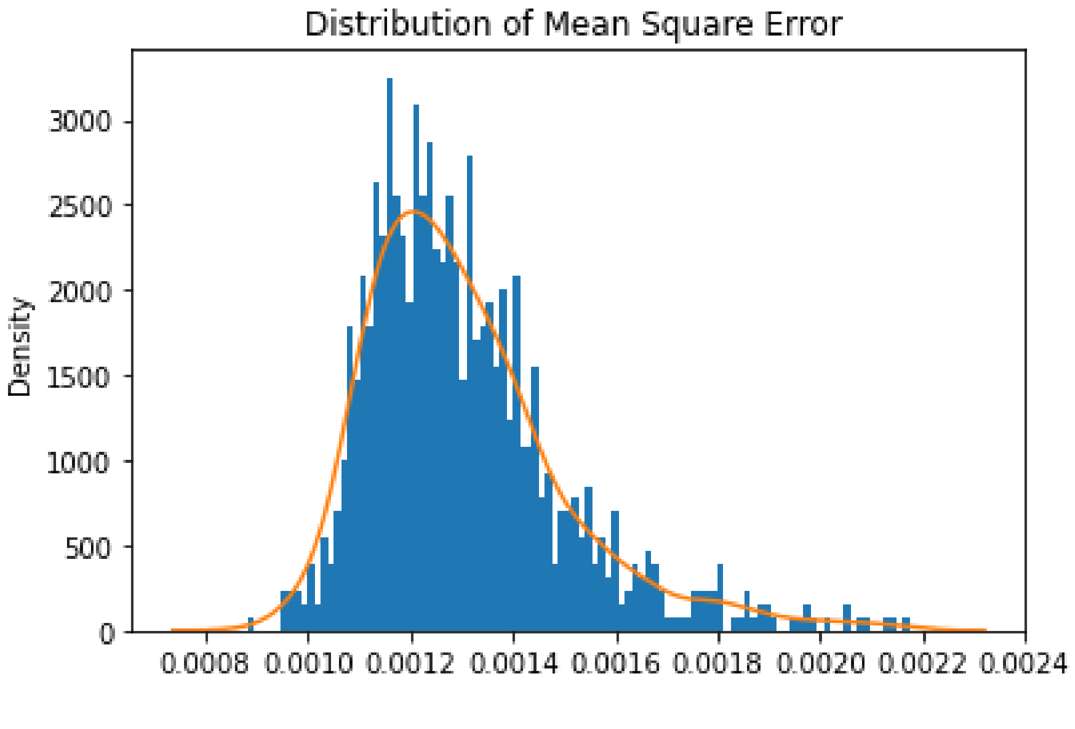

Y_Pred
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Y


Y_Pred
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Y

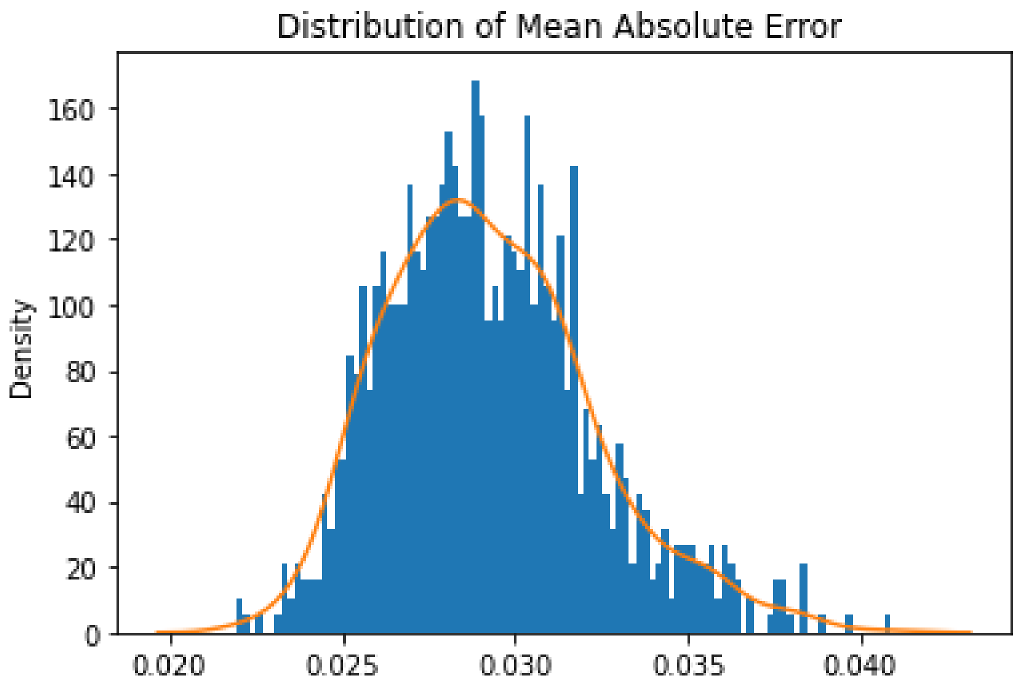

Y_Pred
Y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Y_Pred
Y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

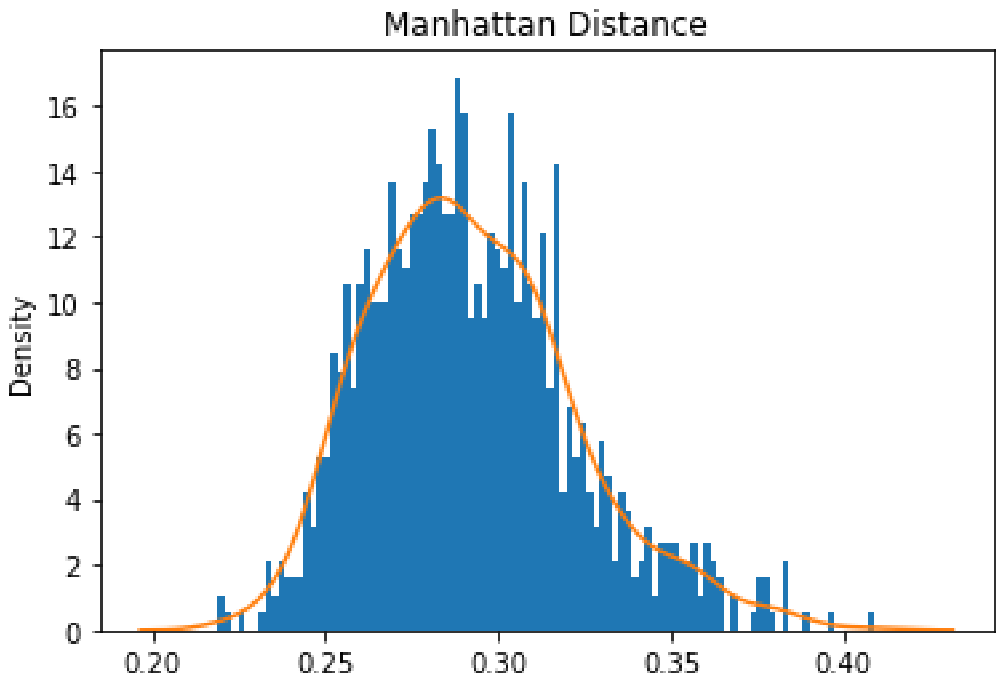




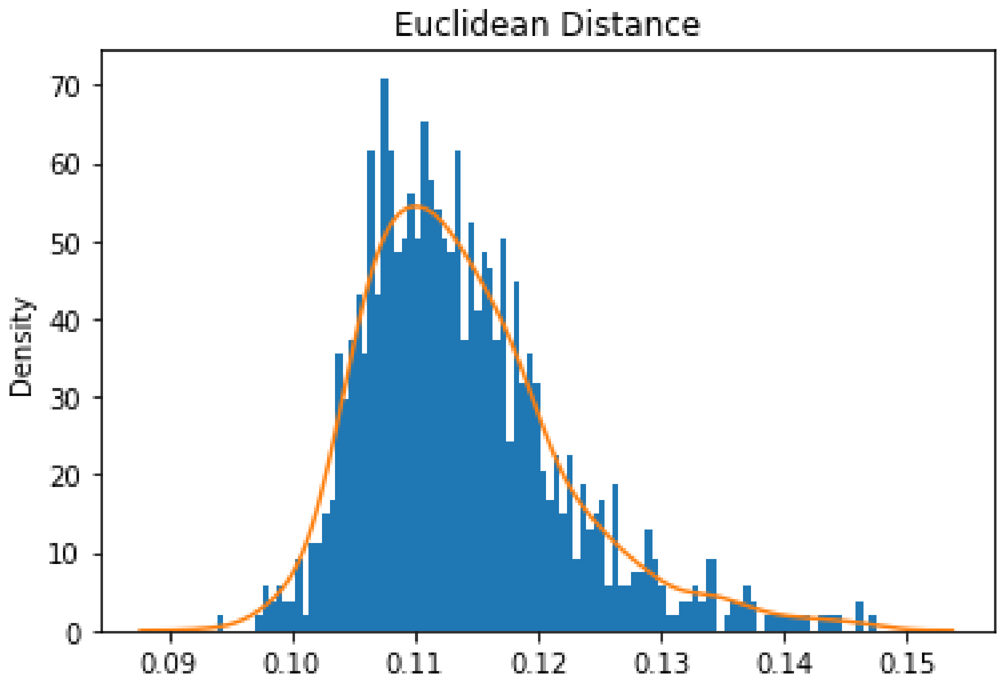
```
Mean Square Error: 0.0013041351233231923
```



```
Mean Absolute Error: 0.029243734639137983
Mean Manhattan Distance: 0.29243734639137986
```



```
Mean Euclidean Distance: 0.11390248953042062
```



```
[ ]:
```