Dataset1-Regression_output_6

November 2, 2021

1 Dataset 1 - Regression

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 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 = 1
```

1.4 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 $\sigma = 0.1$

[5]:
```
X,Y = regressionDataset.regression_data(n_samples,n_features)
```

```
         X1        X2        X3        X4        X5        X6        X7  \
0  2.101310 -1.159082 -0.859004 -0.482396  1.723772 -0.380154 -0.479376
1 -0.618225 -0.231130 -1.504373  0.364974 -0.493353 -1.273273  0.284176
2  0.742590  0.406798 -0.407174 -0.707827  1.242789  1.339120 -2.178362
3  1.395862  0.217220 -2.349368  0.087988  1.167551 -0.210160  0.924579
4 -0.288377  0.677366 -0.032879  1.113577 -1.921581  0.092525 -1.241477

         X8        X9       X10          Y
```

```
0  0.159169 -0.020145 -1.219130 -57.644694
1  0.470256  0.647875  2.904470   4.925825
2  0.497151  0.751699  1.262086  89.823537
3 -0.995281  0.126181  0.040308  55.879887
4 -1.352766  1.470113  1.545339 -91.497793
```

1.5 Stats Model

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

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

OLS Regression Results

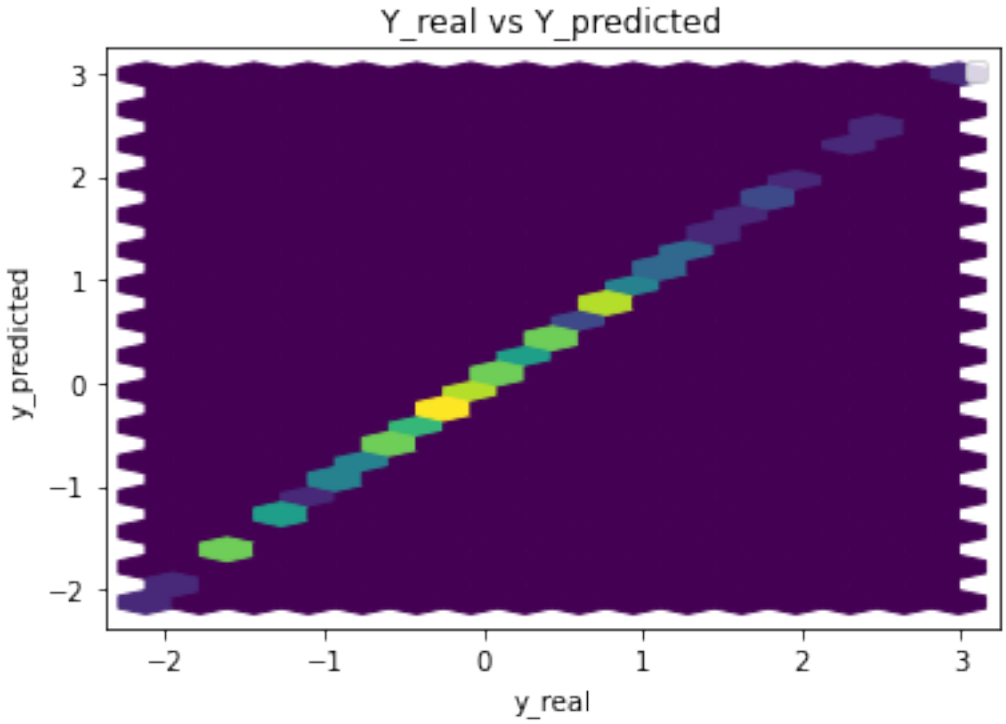
Dep. Variable:	Y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	3.670e+07
Date:	Tue, 02 Nov 2021	Prob (F-statistic):	8.48e-290
Time:	18:24:03	Log-Likelihood:	619.72
No. Observations:	100	AIC:	-1217.
Df Residuals:	89	BIC:	-1189.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.082e-17	5.22e-05	3.99e-13	1.000	-0.000	0.000
x1	0.1293	5.55e-05	2331.090	0.000	0.129	0.129
x2	0.2878	5.39e-05	5342.747	0.000	0.288	0.288
x3	0.2671	5.3e-05	5038.036	0.000	0.267	0.267
x4	0.3598	5.58e-05	6452.725	0.000	0.360	0.360
x5	0.4250	5.44e-05	7805.449	0.000	0.425	0.425
x6	0.3613	5.34e-05	6762.615	0.000	0.361	0.361
x7	0.4668	5.42e-05	8620.589	0.000	0.467	0.467
x8	0.2688	5.42e-05	4959.802	0.000	0.269	0.269
x9	0.2215	5.46e-05	4054.772	0.000	0.221	0.222
x10	0.2182	5.56e-05	3924.062	0.000	0.218	0.218

Omnibus:	1.580	Durbin-Watson:	1.933
Prob(Omnibus):	0.454	Jarque-Bera (JB):	1.475
Skew:	0.181	Prob(JB):	0.478
Kurtosis:	2.528	Cond. No.	1.58

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
Parameters:  const    2.081668e-17
x1       1.293112e-01
```

```
x2        2.877830e-01
x3        2.671076e-01
x4        3.598273e-01
x5        4.250048e-01
x6        3.612810e-01
x7        4.668141e-01
x8        2.688408e-01
x9        2.215336e-01
x10       2.181567e-01
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 2.425125088393323e-07
Mean Absolute Error: 0.00040118341075792946
Manhattan distance: 0.040118341075792936
Euclidean distance: 0.004924555907280699
```

1.6 Common Training Parameters (GAN & ABC_GAN)

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

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

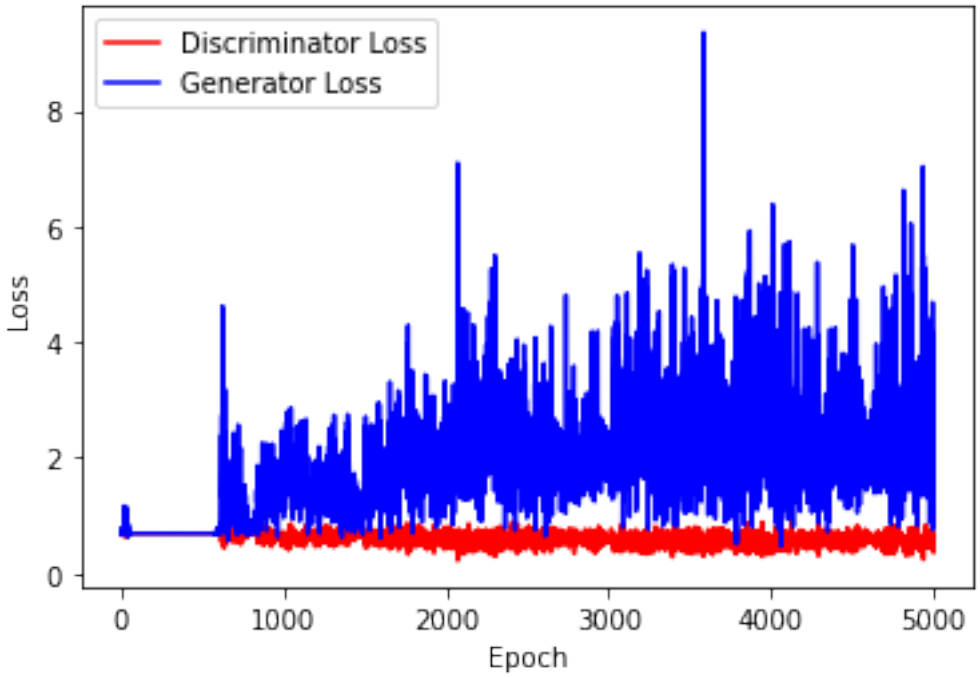
[9]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)

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)
```

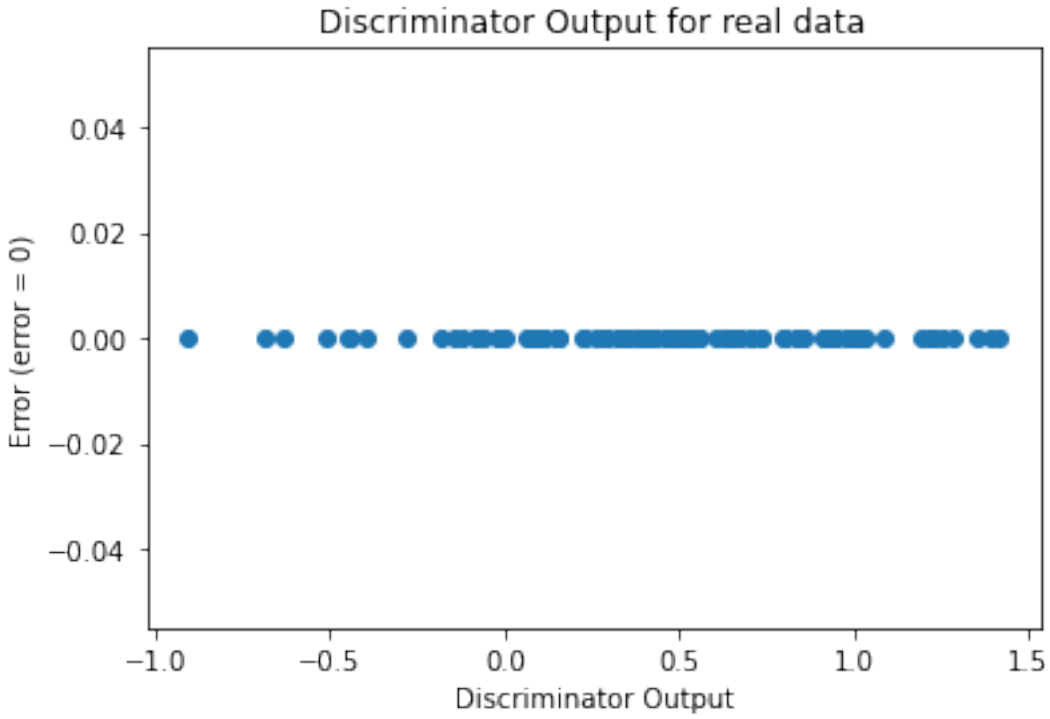
```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)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

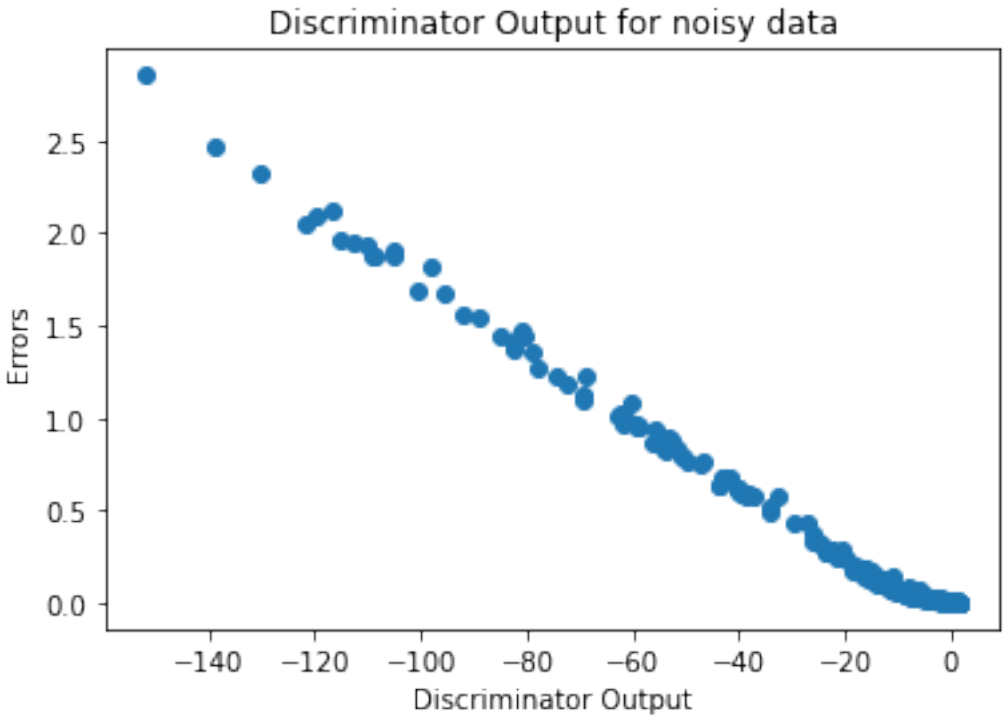
[11]:
```
train_test.
↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
↪n_epochs,criterion,device)
```



[12]:
```
GAN1_metrics = train_test.test_generator(generator,real_dataset,device)
```

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



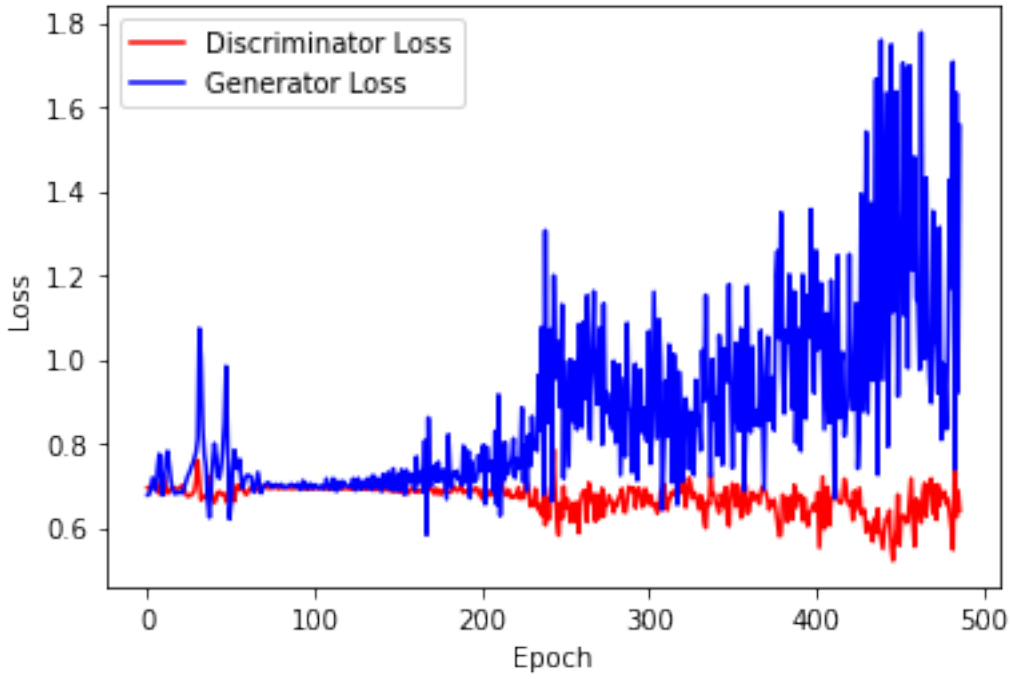


Training GAN until mse of y_pred is $>$ 0.1 or n_epochs $<$ 30000

[14]:
```
generator2 = network.Generator(n_features+2)
discriminator2 = network.Discriminator(n_features+2)
criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator2.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator2.parameters(), lr=0.01, betas=(0.5, 0.
↪999))
```

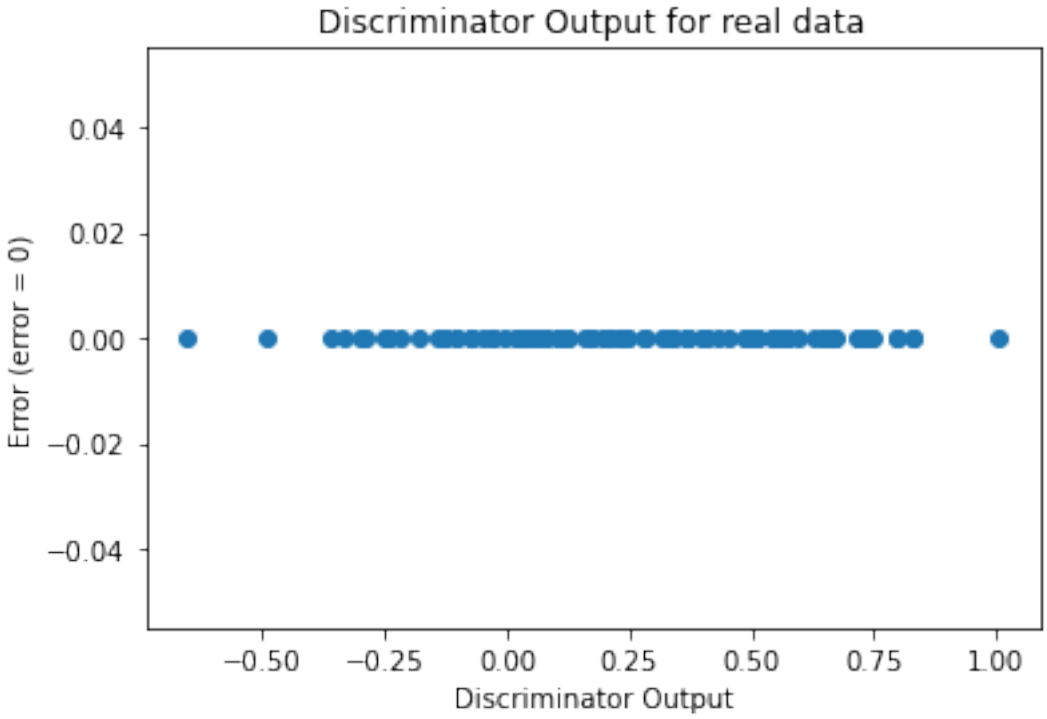
[15]:
```
train_test.
↪training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,real_dataset,batch_size,error,cri
```

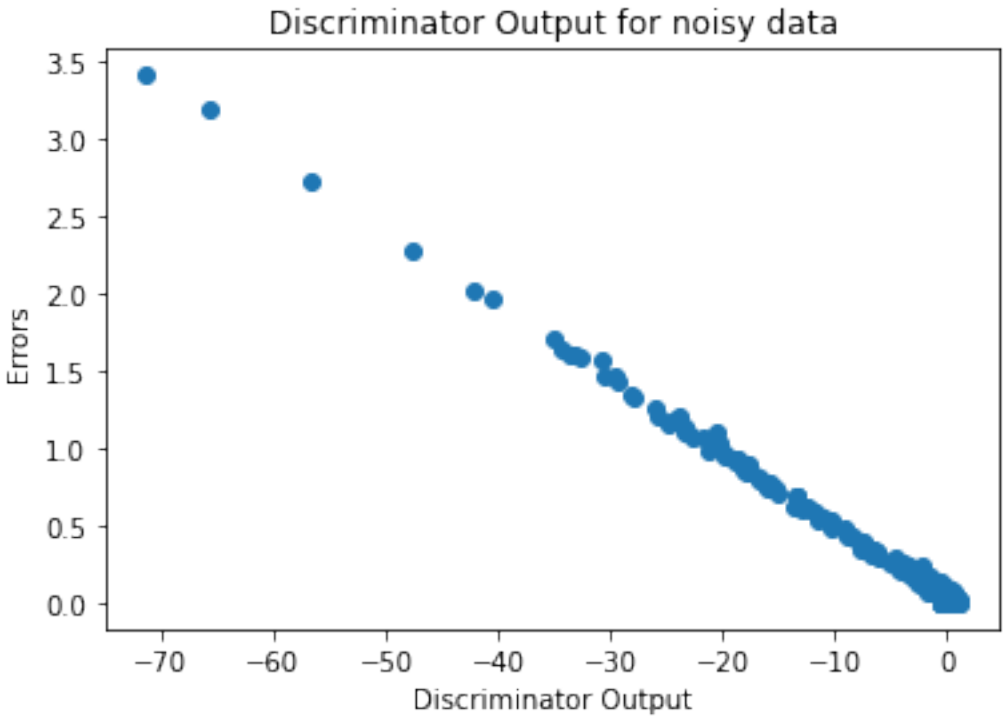
```
Number of epochs needed 487
```



[16]:
```
GAN2_metrics=train_test.test_generator_2(generator2,real_dataset,device)
```

[17]:
```
sanityChecks.discProbVsError(real_dataset,discriminator2,device)
```





2 ABC GAN Model

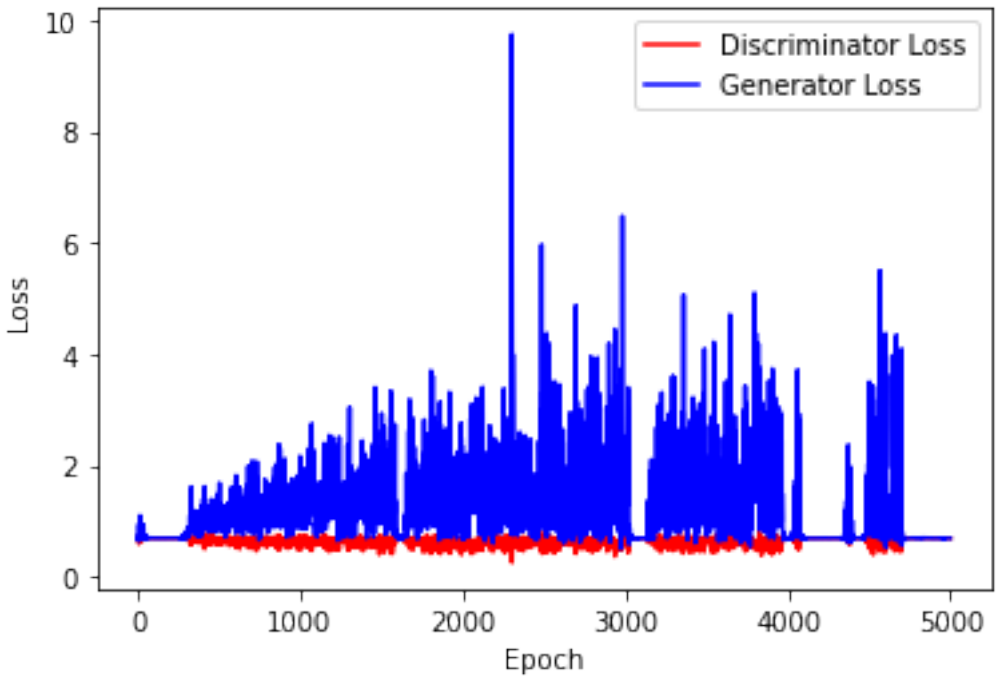
2.0.1 Training the network

Training ABC-GAN for n_epochs number of epochs

[18]:
```
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))
```

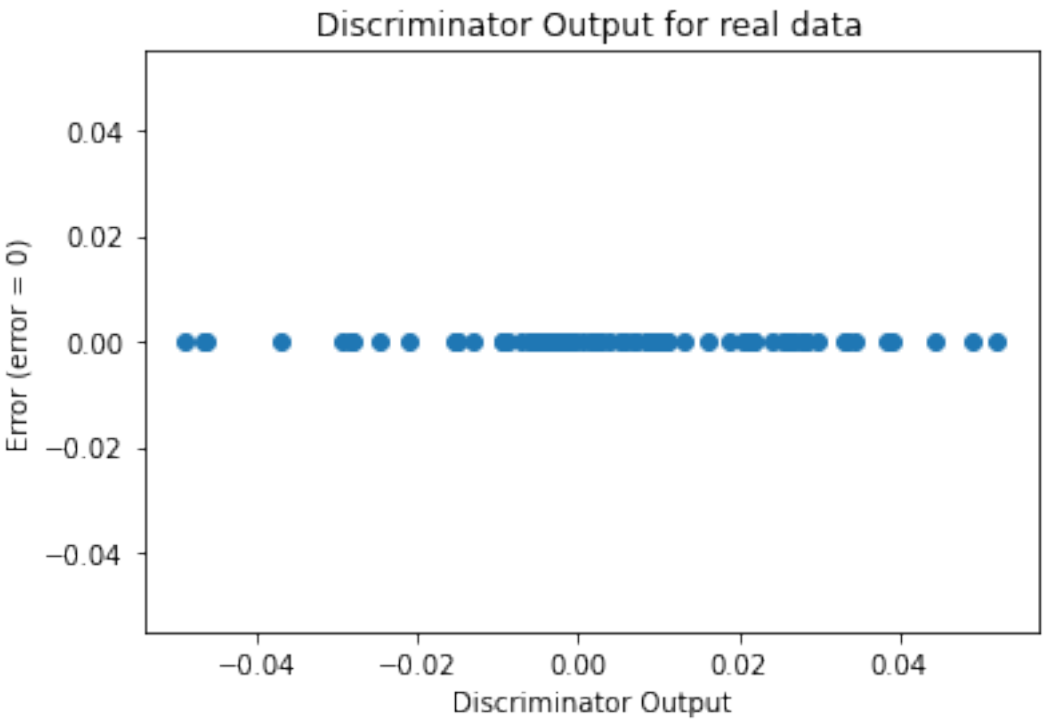
[19]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```

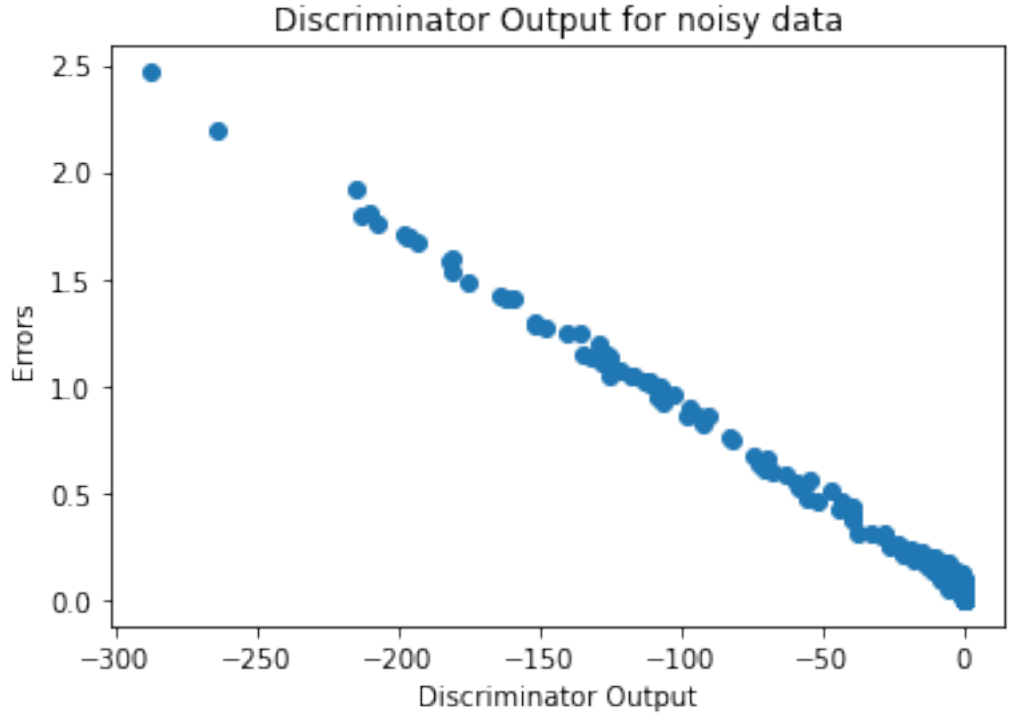


[20]:
```
ABC_GAN1_metrics=ABC_train_test.
↪test_generator(gen,real_dataset,coeff,mean,variance,device)
```

Sanity Checks

[21]:
```
sanityChecks.discProbVsError(real_dataset,disc,device)
```


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


Discriminator Output for noisy data
Errors
2.5
2.0
1.5
1.0
0.5
0.0
−300
−250
−200
−150
−100
−50
0
Discriminator Output

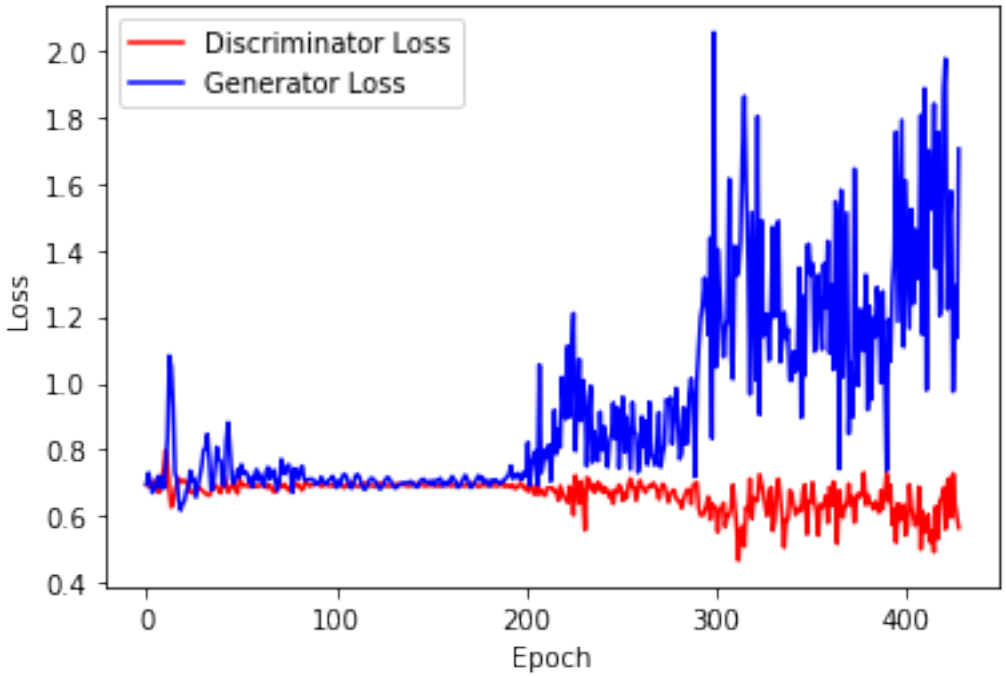
Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

[22]:
```
gen2 = network.Generator(n_features+2)
disc2 = network.Discriminator(n_features+2)

criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(gen2.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(disc2.parameters(), lr=0.01, betas=(0.5, 0.999))
```

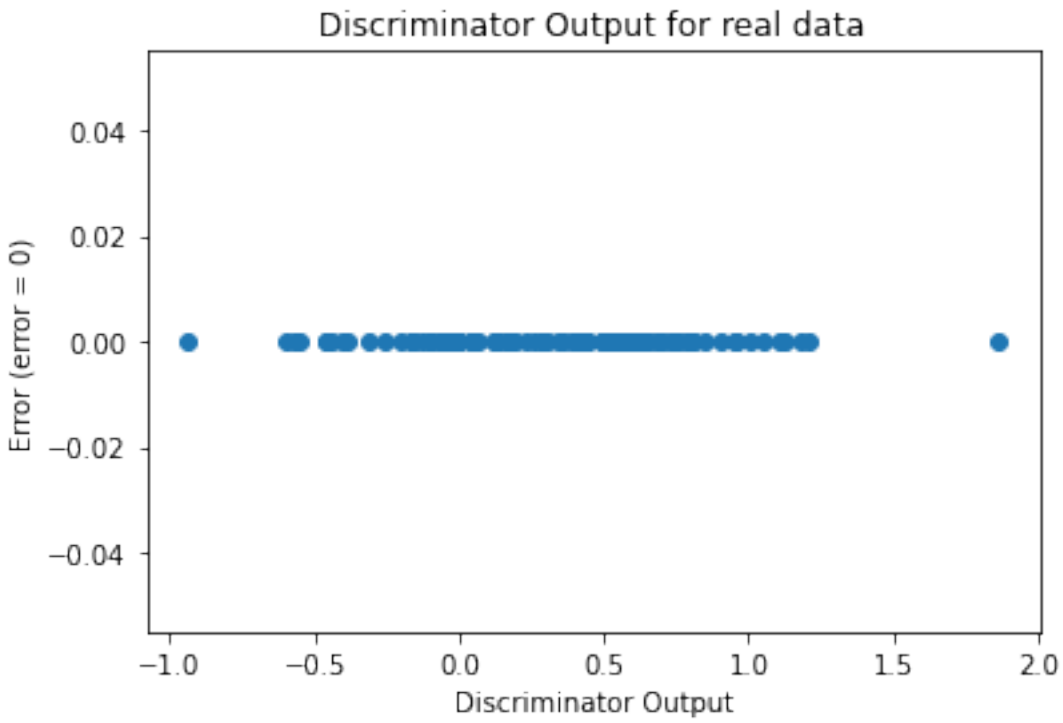
[23]:
```
ABC_train_test.
↪training_GAN_2(disc2,gen2,disc_opt,gen_opt,real_dataset,batch_size,␣
↪error,criterion,coeff,mean,variance,device)
```

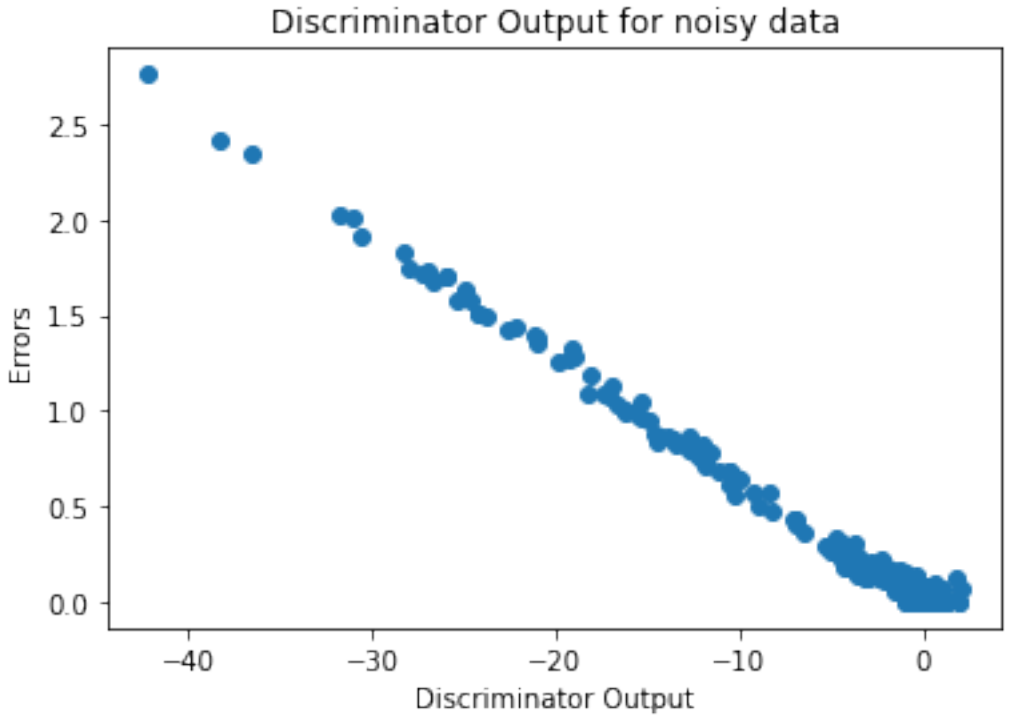
```
Number of epochs 429
```



[24]:
```
ABC_GAN2_metrics=ABC_train_test.
↪test_generator_2(gen2,real_dataset,coeff,mean,variance,device)
```

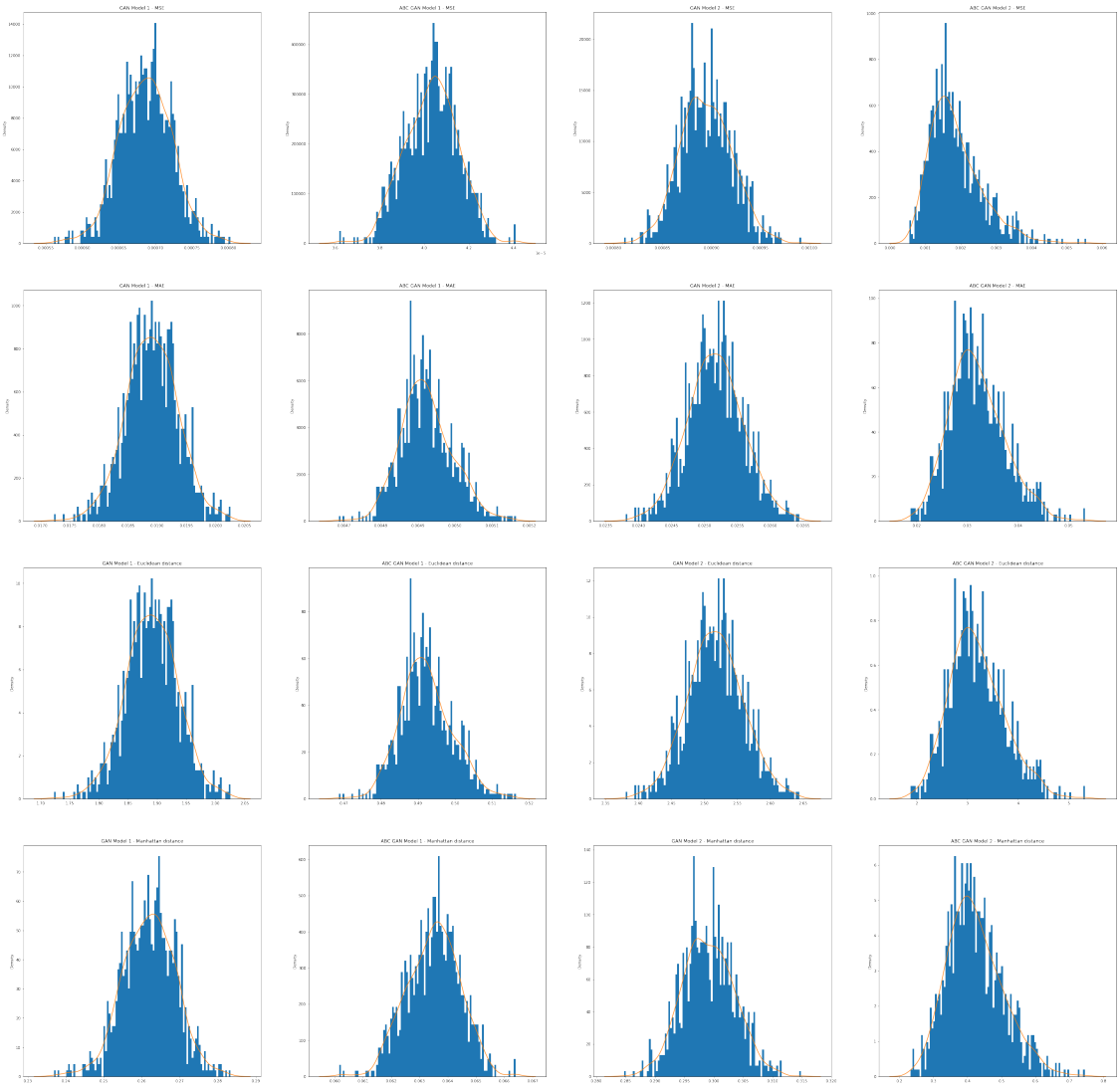
[25]:
```
sanityChecks.discProbVsError(real_dataset,disc2,device)
```


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


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

3 Model Analysis

[26]:
```
performanceMetrics.
↪modelAnalysis(GAN1_metrics,ABC_GAN1_metrics,GAN2_metrics,ABC_GAN2_metrics)
```



[ ]: