Assignment 4

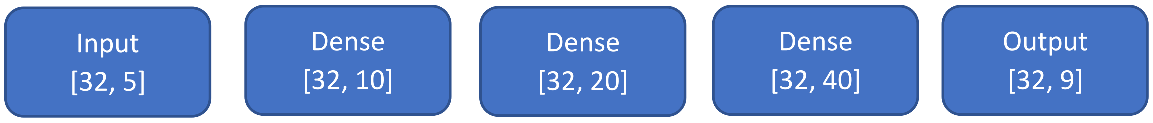
Submitted by:

Amit Sultan – 205975444

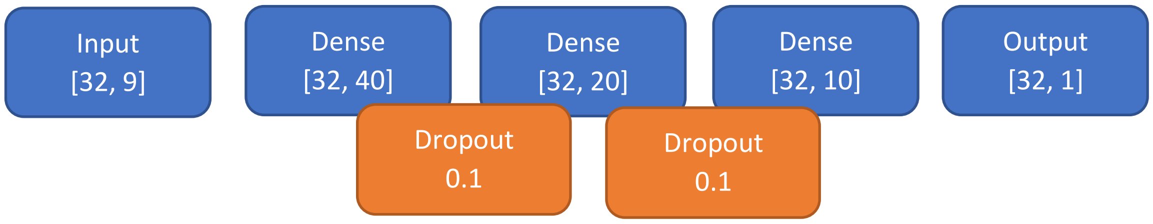
Ariel Blobstein - 206197113

First we describe the general GAN model that we created for both dataset, we note that the only change between the models is the input (noise dim) and output which is the desired sample size.

Generator:



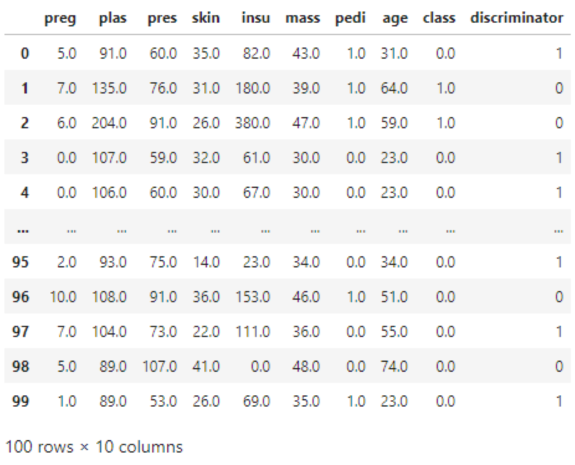
Discriminator:



The table below shows the generator and discriminator input and output for each data set.

Dataset	**Generator Input**	**Generator output**	**Discriminator Input**	**Discriminator output**
Diabetes	5	9	9	1
German Credit	5	21	21	1

After the training process we give the generator random noise and receive an input data that is similar to the normalized data that we train the model on, that is not the intended output and thus we use the reverse_transform method and receive a new dataset within the correct range of values. Following that, we let the discriminator guess if these samples are real or not and we label the discriminator guess for real samples as 1 and 0 otherwise. The example below shows the results for generating diabetes samples.



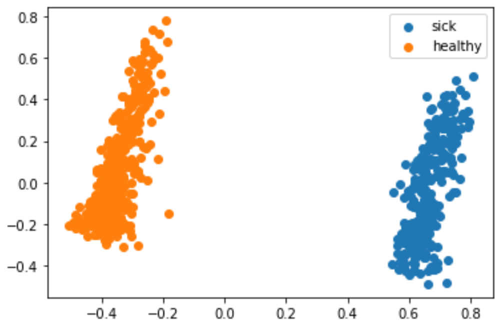
	preg	plas	pres	skin	insu	mass	pedi	age	class	discriminator
0	5.0	91.0	60.0	35.0	82.0	43.0	1.0	31.0	0.0	1
1	7.0	135.0	76.0	31.0	180.0	39.0	1.0	64.0	1.0	0
2	6.0	204.0	91.0	26.0	380.0	47.0	1.0	59.0	1.0	0
3	0.0	107.0	59.0	32.0	61.0	30.0	0.0	23.0	0.0	1
4	0.0	106.0	60.0	30.0	67.0	30.0	0.0	23.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...
95	2.0	93.0	75.0	14.0	23.0	34.0	0.0	34.0	0.0	1
96	10.0	108.0	91.0	36.0	153.0	46.0	1.0	51.0	0.0	0
97	7.0	104.0	73.0	22.0	111.0	36.0	0.0	55.0	0.0	1
98	5.0	89.0	107.0	41.0	0.0	48.0	0.0	74.0	0.0	0
99	1.0	89.0	53.0	26.0	69.0	35.0	1.0	23.0	0.0	1

100 rows × 10 columns

In the following segments we will show further results and analysis on each data-set separately.

Diabetes

First, we look at the diabetes dataset and preprocess it before continuing to the training process. We decided to use min-max normalization and we normalize the entire dataset with SKLearn function MinMaxScaler.
The use of the MinMaxScaler with PCA resulted in a more easy-to-read plot where each group of sick/not sick points are divided clearly.



Original data

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	b'tested_positive'
2	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	b'tested_positive'
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	b'tested_positive'
6	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	b'tested_positive'
8	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	b'tested_positive'

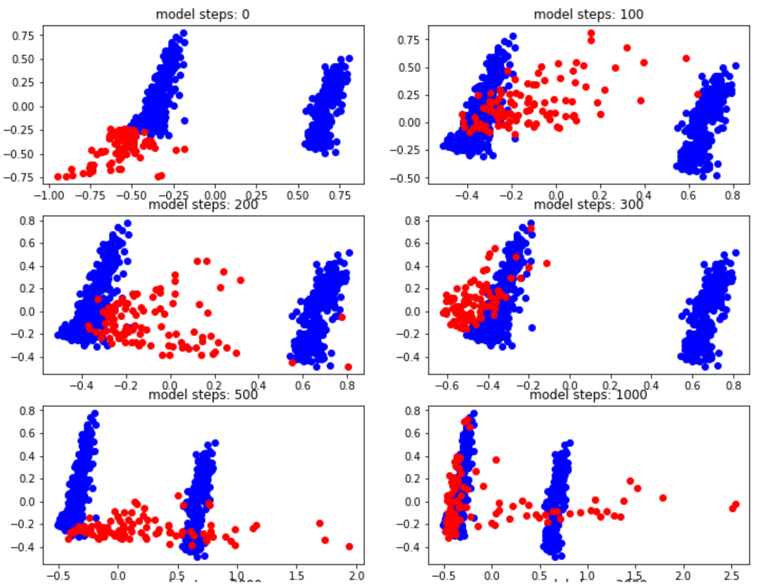
Generated data

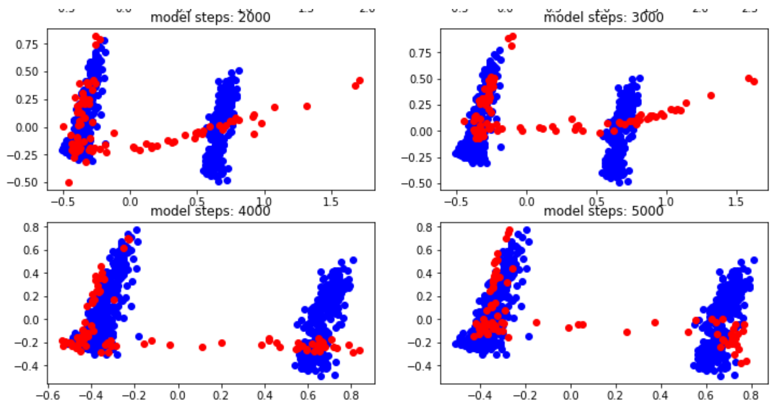
	preg	plas	pres	skin	insu	mass	pedi	age	class
1	7.0	135.0	76.0	31.0	180.0	39.0	1.0	64.0	1.0
2	6.0	204.0	91.0	26.0	380.0	47.0	1.0	59.0	1.0
6	6.0	159.0	82.0	20.0	202.0	37.0	1.0	49.0	1.0
8	5.0	191.0	69.0	54.0	197.0	54.0	1.0	15.0	1.0
16	5.0	163.0	72.0	33.0	182.0	43.0	1.0	35.0	1.0

Learning process

To capture and evaluate our GAN model we capture the weights of the network throughout a few milestones and then we generate some samples with those weights, comparing how close they are to the real data by closeness with PCA.

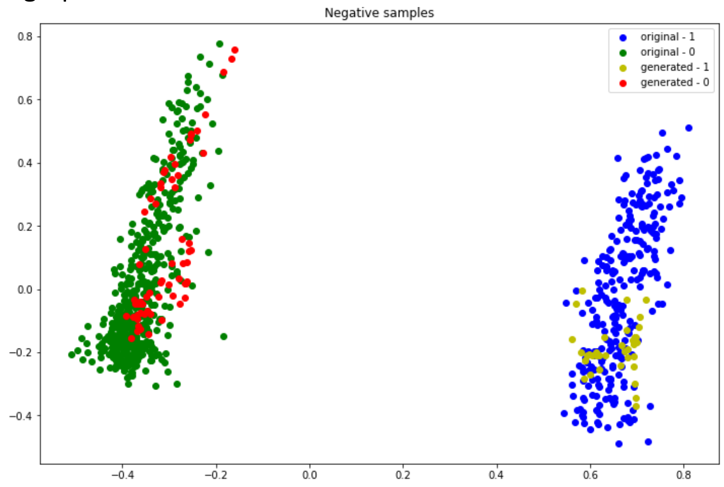
The results and progress of our model are clear and we see a great improvement as the epochs continues, the end result looks very close to the similar data except a few samples that sits somewhere between the two classes.





When analyzing the actual samples after transforming them back to their original range of values we see similar behavior for most features, for example, plas and insu are quite close in values range for the original data, we can plot both data points at a 2D graph with the use of PCA and see how both groups distribute.

The graph below shows the normalized data and the normalized generated data and we can see



Overall, the samples are close to the original, but we do see an odd behavior at the left side of the plots with the line of dots, unfortunately we are not sure why this behavior exists but that is the original data plotted with PCA so the dimension reduction might be the cause.

As of our discriminator, we want to check how many samples out of the 100 samples did fool the discriminator. We find out the 58 samples has been classified as true out of the 100 generated samples!

Euclidean distance:

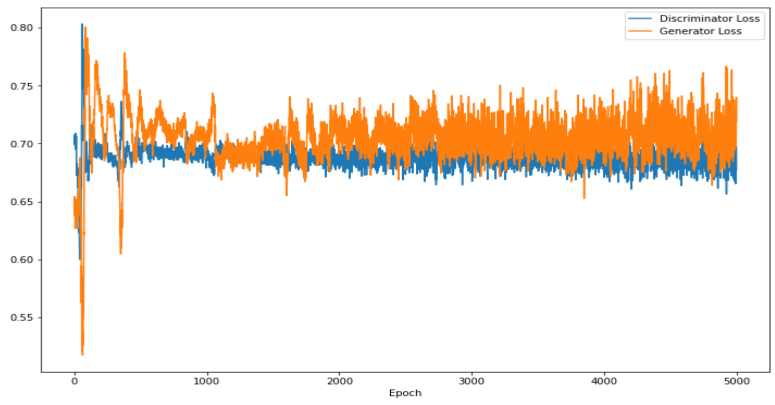
We calculated the distance between positive examples on the real data and between the generated data to the real data and found out the following [Positive samples, Negative samples]:

Distance between real data points – Mean [160.30, 114.76]
Distance between generated to real data points – Mean [185.30, 88.83]

The results are very similar in terms of distance between points for the real data compared to the generated data.

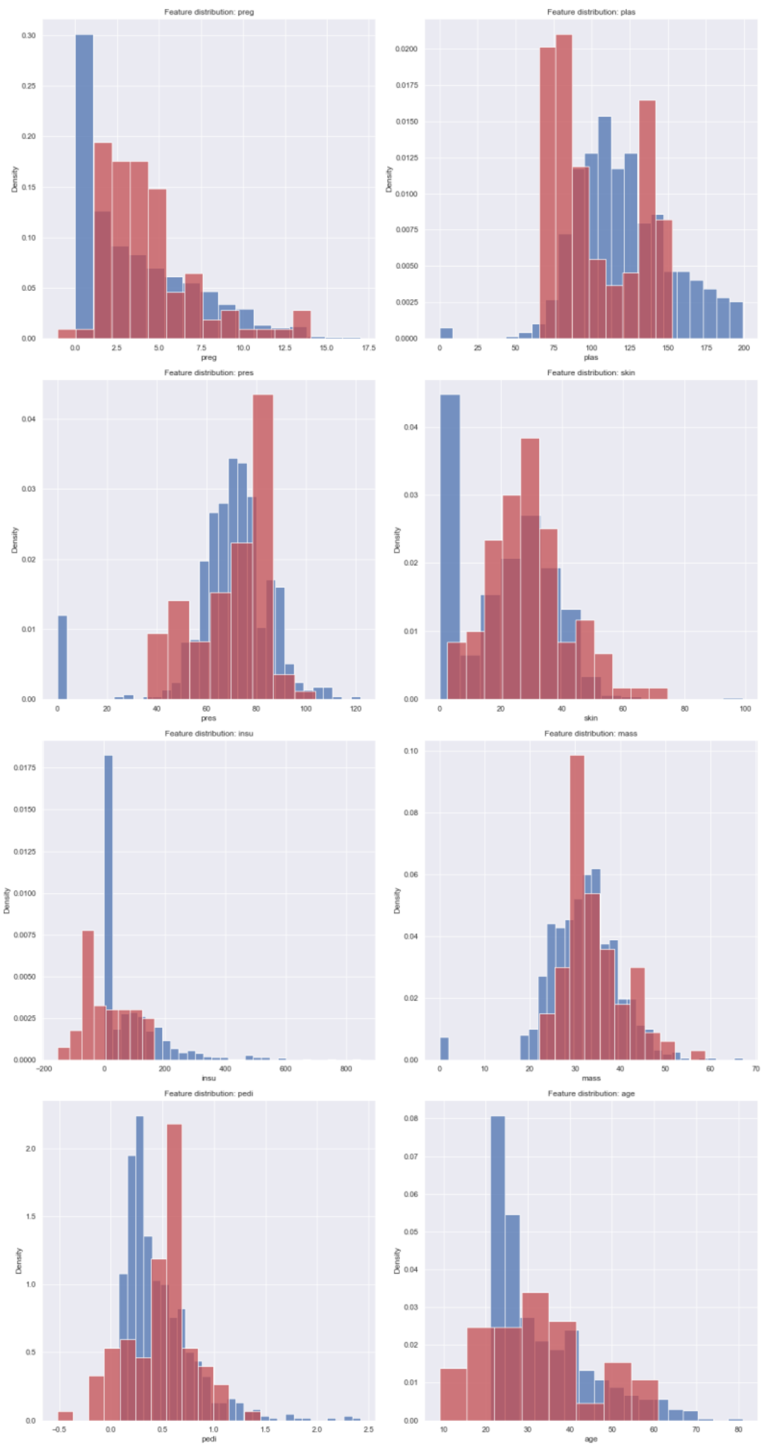
Loss graph:

We do see a “back and forth” motion in the losses graph but it is not clear whether they switched places during this period, most of the times the clear “winner” was the discriminator and we think it is related to the ease of classification task in that particular dataset.



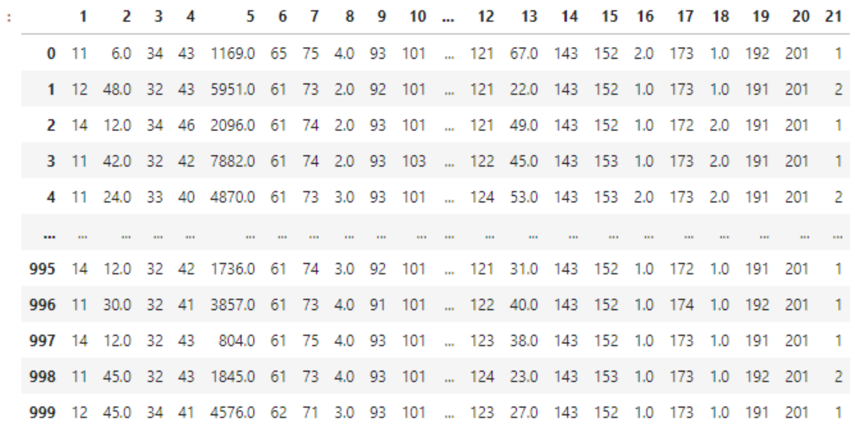
Feature analysis:

We wanted to compare the generated data per feature to try and capture how to model has learned to generate each one of them. We have noticed that the generator is indeed trying to learn some trends among the different distributions, we can see that it tries to mimic the distribution by assigning higher values in most of the cases where there are indeed high values in the data. Since our purpose in creating generative models is to learn to distribution rather than mimic the data itself, we did notice that the generator is indeed learning in that direction but as we see he is not able to capture the original distribution rather than some function of it.



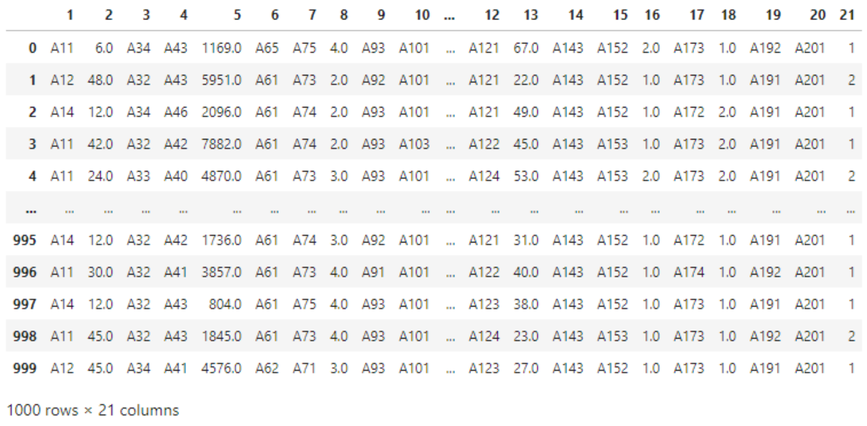
German credit

As for the German dataset we used the StandardScaler method for normalization and the results were:



:	1	2	3	4	5	6	7	8	9	10	...	12	13	14	15	16	17	18	19	20	21
0	11	6.0	34	43	1169.0	65	75	4.0	93	101	...	121	67.0	143	152	2.0	173	1.0	192	201	1
1	12	48.0	32	43	5951.0	61	73	2.0	92	101	...	121	22.0	143	152	1.0	173	1.0	191	201	2
2	14	12.0	34	46	2096.0	61	74	2.0	93	101	...	121	49.0	143	152	1.0	172	2.0	191	201	1
3	11	42.0	32	42	7882.0	61	74	2.0	93	103	...	122	45.0	143	153	1.0	173	2.0	191	201	1
4	11	24.0	33	40	4870.0	61	73	3.0	93	101	...	124	53.0	143	153	2.0	173	2.0	191	201	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	14	12.0	32	42	1736.0	61	74	3.0	92	101	...	121	31.0	143	152	1.0	172	1.0	191	201	1
996	11	30.0	32	41	3857.0	61	73	4.0	91	101	...	122	40.0	143	152	1.0	174	1.0	192	201	1
997	14	12.0	32	43	804.0	61	75	4.0	93	101	...	123	38.0	143	152	1.0	173	1.0	191	201	1
998	11	45.0	32	43	1845.0	61	73	4.0	93	101	...	124	23.0	143	153	1.0	173	1.0	192	201	2
999	12	45.0	34	41	4576.0	62	71	3.0	93	101	...	123	27.0	143	152	1.0	173	1.0	191	201	1

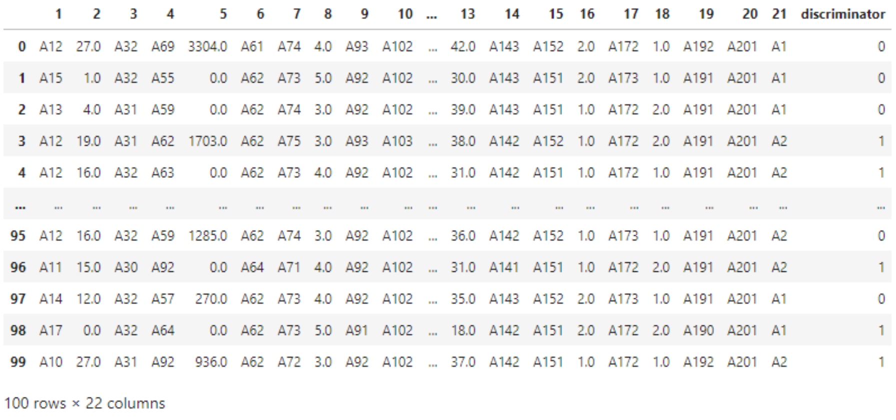
Original data



	1	2	3	4	5	6	7	8	9	10	...	12	13	14	15	16	17	18	19	20	21
0	A11	6.0	A34	A43	1169.0	A65	A75	4.0	A93	A101	...	A121	67.0	A143	A152	2.0	A173	1.0	A192	A201	1
1	A12	48.0	A32	A43	5951.0	A61	A73	2.0	A92	A101	...	A121	22.0	A143	A152	1.0	A173	1.0	A191	A201	2
2	A14	12.0	A34	A46	2096.0	A61	A74	2.0	A93	A101	...	A121	49.0	A143	A152	1.0	A172	2.0	A191	A201	1
3	A11	42.0	A32	A42	7882.0	A61	A74	2.0	A93	A103	...	A122	45.0	A143	A153	1.0	A173	2.0	A191	A201	1
4	A11	24.0	A33	A40	4870.0	A61	A73	3.0	A93	A101	...	A124	53.0	A143	A153	2.0	A173	2.0	A191	A201	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	A14	12.0	A32	A42	1736.0	A61	A74	3.0	A92	A101	...	A121	31.0	A143	A152	1.0	A172	1.0	A191	A201	1
996	A11	30.0	A32	A41	3857.0	A61	A73	4.0	A91	A101	...	A122	40.0	A143	A152	1.0	A174	1.0	A192	A201	1
997	A14	12.0	A32	A43	804.0	A61	A75	4.0	A93	A101	...	A123	38.0	A143	A152	1.0	A173	1.0	A191	A201	1
998	A11	45.0	A32	A43	1845.0	A61	A73	4.0	A93	A101	...	A124	23.0	A143	A153	1.0	A173	1.0	A192	A201	2
999	A12	45.0	A34	A41	4576.0	A62	A71	3.0	A93	A101	...	A123	27.0	A143	A152	1.0	A173	1.0	A191	A201	1

1000 rows × 21 columns

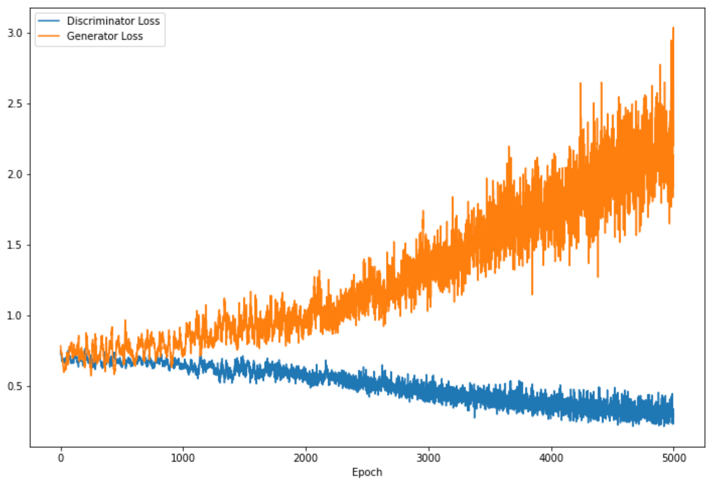
Generated data



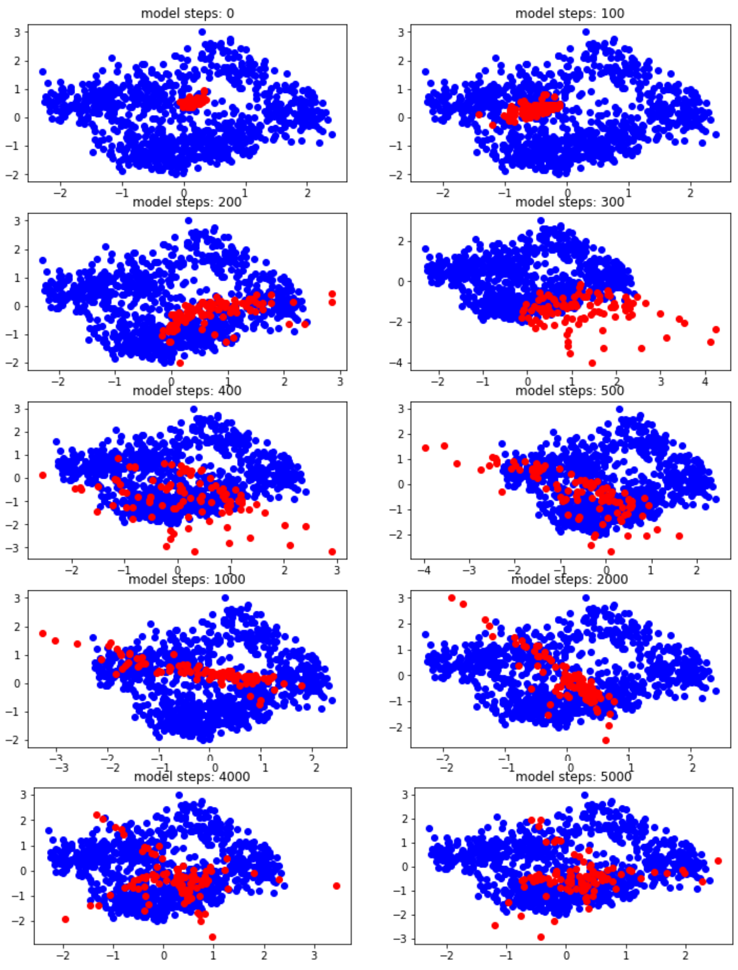
	1	2	3	4	5	6	7	8	9	10	...	13	14	15	16	17	18	19	20	21	discriminator
0	A12	27.0	A32	A69	3304.0	A61	A74	4.0	A93	A102	...	42.0	A143	A152	2.0	A172	1.0	A192	A201	A1	0
1	A15	1.0	A32	A55	0.0	A62	A73	5.0	A92	A102	...	30.0	A143	A151	2.0	A173	1.0	A191	A201	A1	0
2	A13	4.0	A31	A59	0.0	A62	A74	3.0	A92	A102	...	39.0	A143	A151	1.0	A172	2.0	A191	A201	A1	0
3	A12	19.0	A31	A62	1703.0	A62	A75	3.0	A93	A103	...	38.0	A142	A152	1.0	A172	2.0	A191	A201	A2	1
4	A12	16.0	A32	A63	0.0	A62	A73	4.0	A92	A102	...	31.0	A142	A151	1.0	A172	1.0	A191	A201	A2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	A12	16.0	A32	A59	1285.0	A62	A74	3.0	A92	A102	...	36.0	A142	A152	1.0	A173	1.0	A191	A201	A2	0
96	A11	15.0	A30	A92	0.0	A64	A71	4.0	A92	A102	...	31.0	A141	A151	1.0	A172	2.0	A191	A201	A2	1
97	A14	12.0	A32	A57	270.0	A62	A73	4.0	A92	A102	...	35.0	A143	A152	2.0	A173	1.0	A191	A201	A1	0
98	A17	0.0	A32	A64	0.0	A62	A73	5.0	A91	A102	...	18.0	A142	A151	2.0	A172	2.0	A190	A201	A1	1
99	A10	27.0	A31	A92	936.0	A62	A72	3.0	A92	A102	...	37.0	A142	A151	1.0	A172	1.0	A192	A201	A2	1

100 rows × 22 columns

Learning process



The process of generating samples in the German credit was considerably harder we assume because of facts that the dataset contains a lot more features that needs to be generated and correlated to each other and the unbalance nature of the data-set.



Like the diabetes data we attempt to create a PCA representation of the generated and original data to conduct a direct comparison between the two and see if the samples are close to each other.

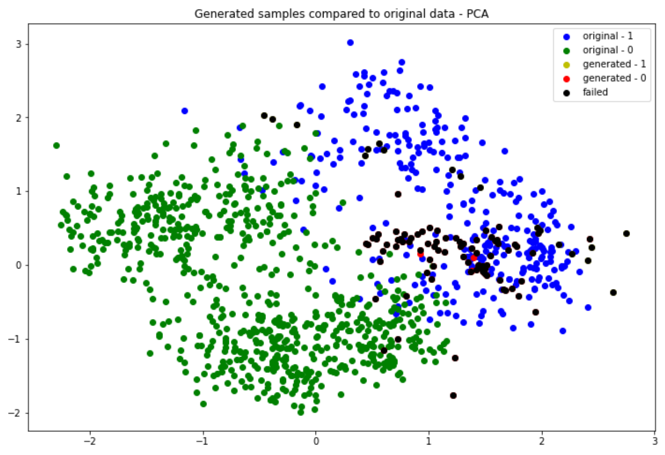
But as we can see below, unlike the diabetes data, this data set is a bit more complicated and the clear distinction between both classes is not well defined. We do see that generated samples that tends to be “clearer” and far from the middle which is the vague part of samples are easier to classify but most data that lays in the center is harder for the discriminator to classify.

In addition to the plots above which implies that the data is a bit harder to be learnt than the diabetes, we further plotted the data using PCA process as we showed before. We can see that in compared with the diabetes the classes are perfectly distinct-able which is turning the problem to be harder even for a simple classification and not only for generation of data.

We can see that there are outliers that hide in the margin that separates from the two classes, which is a known problem in machine learning world that hurt the bias variance trade-off. We can notice that the generator has focused on the bottom left corner of the PCA visualization, which is a place where there is a density of the two classes together, rather than one class apart. This caused us to believe that the data is hard to be learn since the model is getting different signals from close examples.

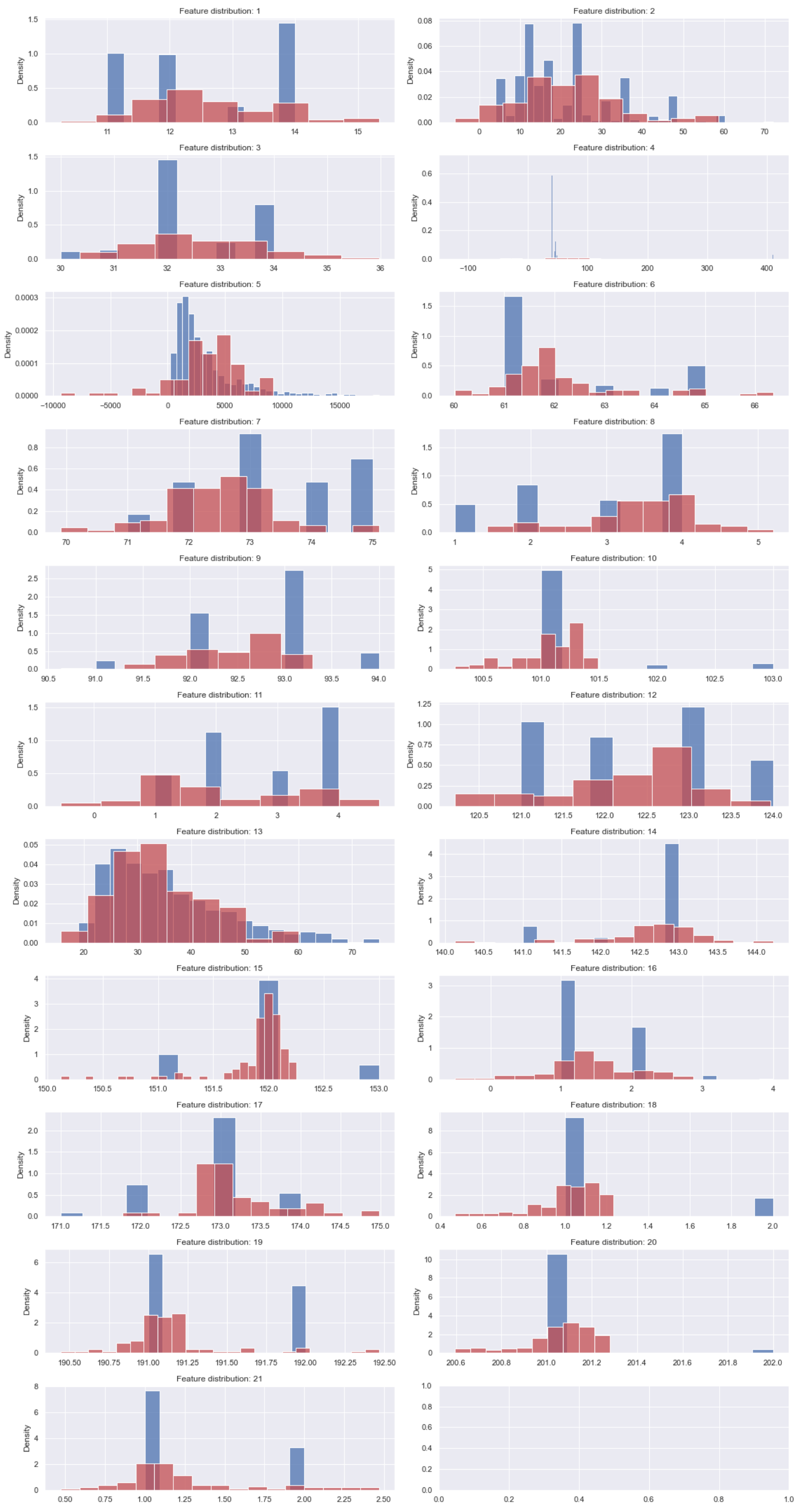
We believe that to fix the problem we have two options to future work:

1) Perhaps try to eliminate those outliers so the generator will have easier time trying to learn the distribution of the two classes, the data isn't well separated but we believe that those outliers is what caused the generator to fail completely.
2) Perhaps try to generate noise from different distribution, we noticed that the model is starting by focusing on a specific area of the data and tries to improve himself in some direction, but as we can see the model failed to improve in that way since it got himself into a vague place, perhaps starting from a different place will lead to better direction of improvement.



<u>Feature analysis</u>:

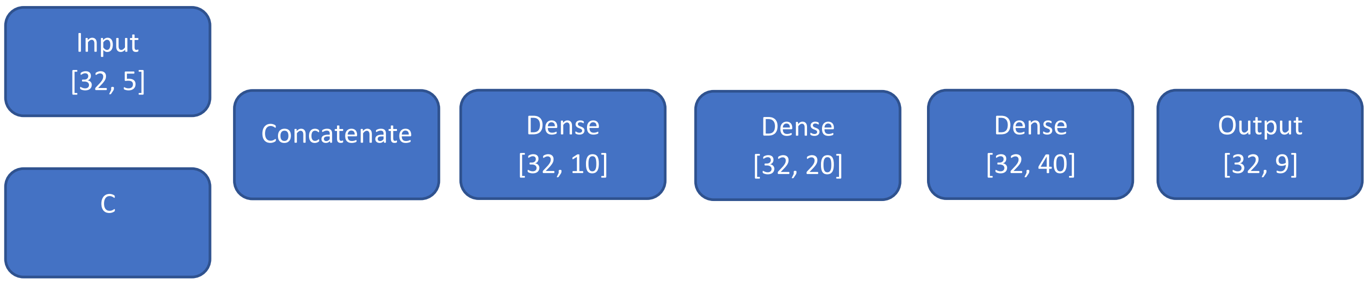
As we can see in the feature analysis, we can notice some similarities between the diabetes data set and this one in the trend finding pattern that the generator is trying to seek while training. The big difference between the two dataset is that here the density from the real data is a bit far from the generated one, we suspect that this phenomenon is caused by the fact that the generator is getting vague signals from the discriminator which leading it to learn wrong patterns of the distribution of the labels. Since it perhaps moving to the right direction, but the signals are telling him his wrong (because of the outliers) causing him to take a step in a wrong direction than intended.



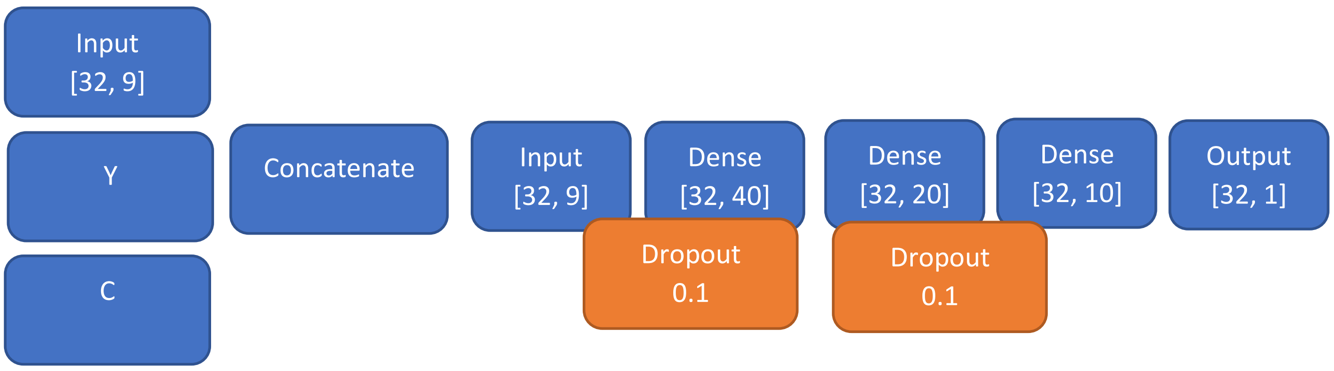
PART 2

Like the architecture stated above, we added the blackbox into the pipeline in the following matter:

Generator

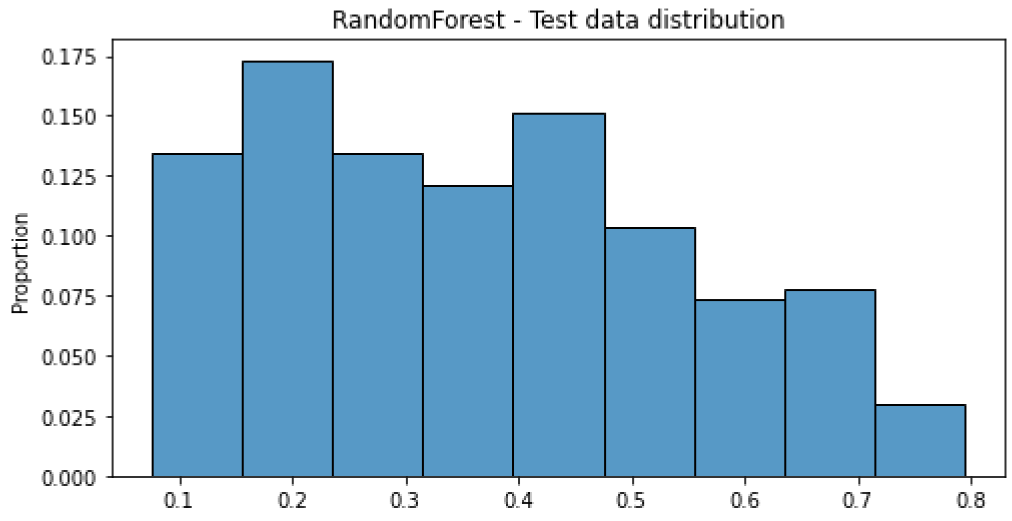


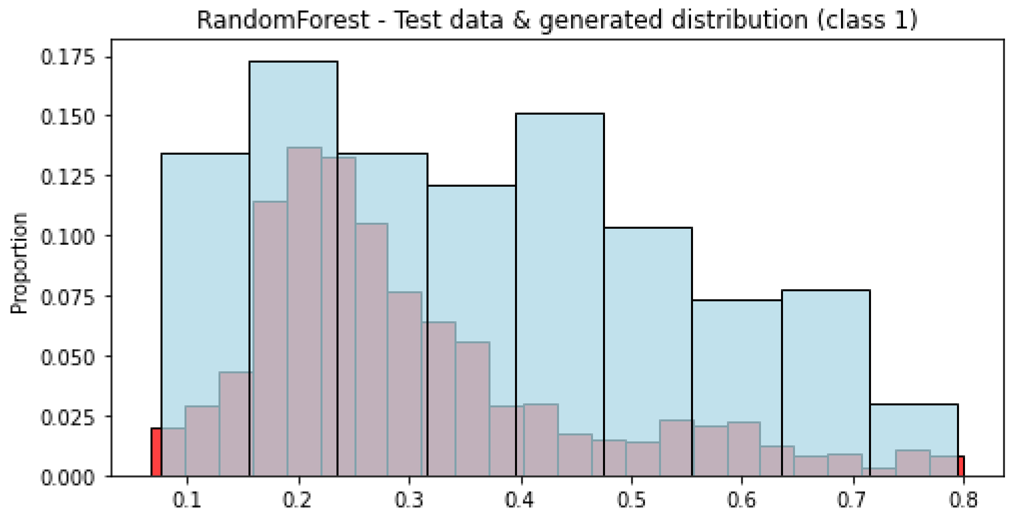
Discriminator

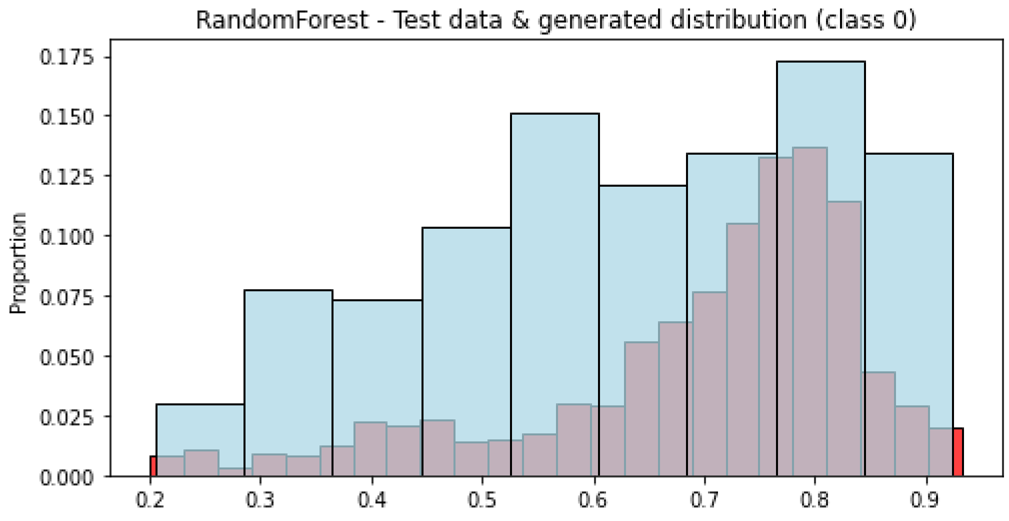


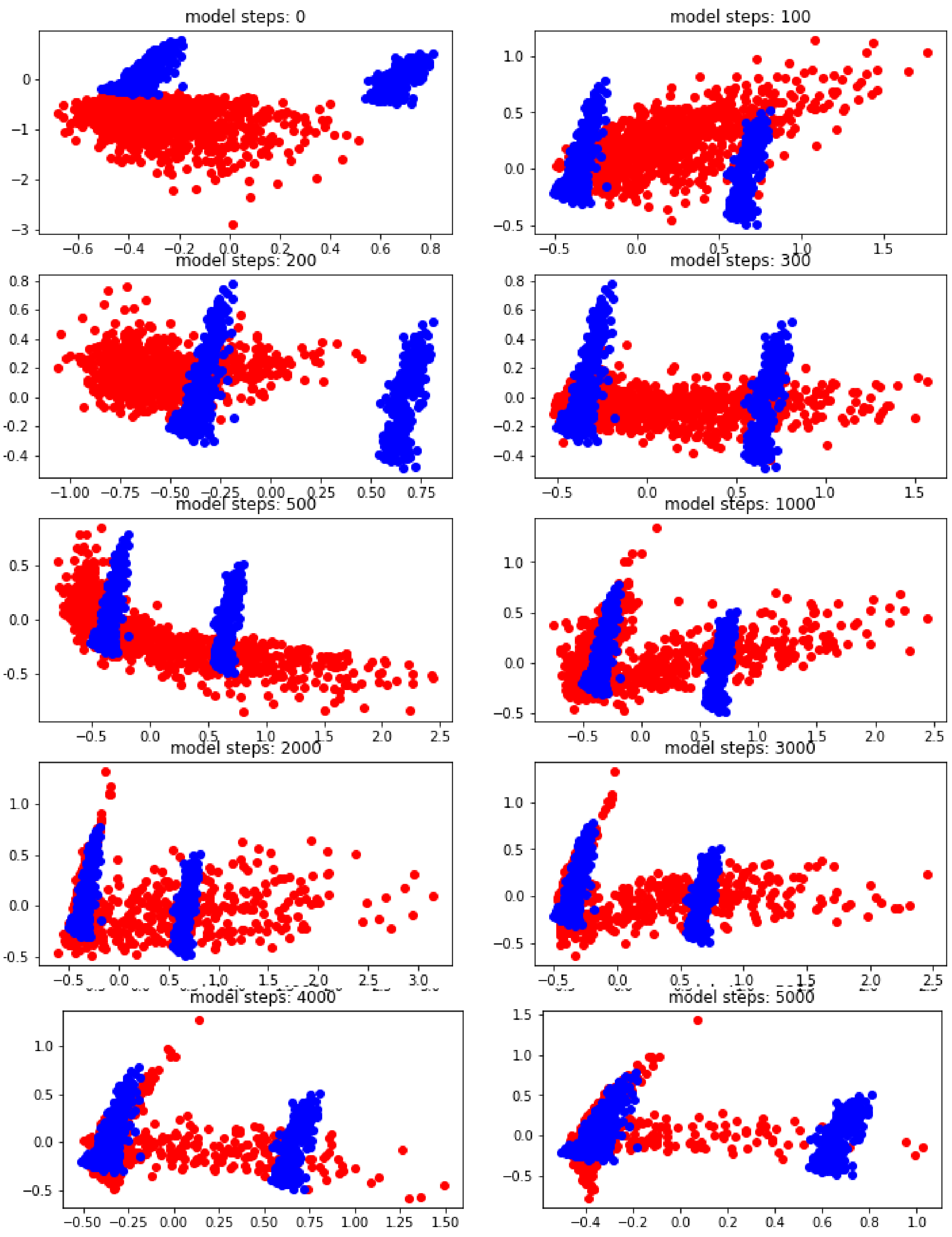
For each batch we generated C values from a Uniform distribution and matched them to the noise batch input from the generator, we then took the generated samples from the Generator and ran them through the Blackbox to get the Y values. Following that, each batch contained generated samples with both the C value use to generate them and the Y guess of our blackbox.

Diabetes


RandomForest - Test data distribution
Proportion
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


RandomForest - Test data & generated distribution (class 1)
Proportion
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


RandomForest - Test data & generated distribution (class 0)
Proportion
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9



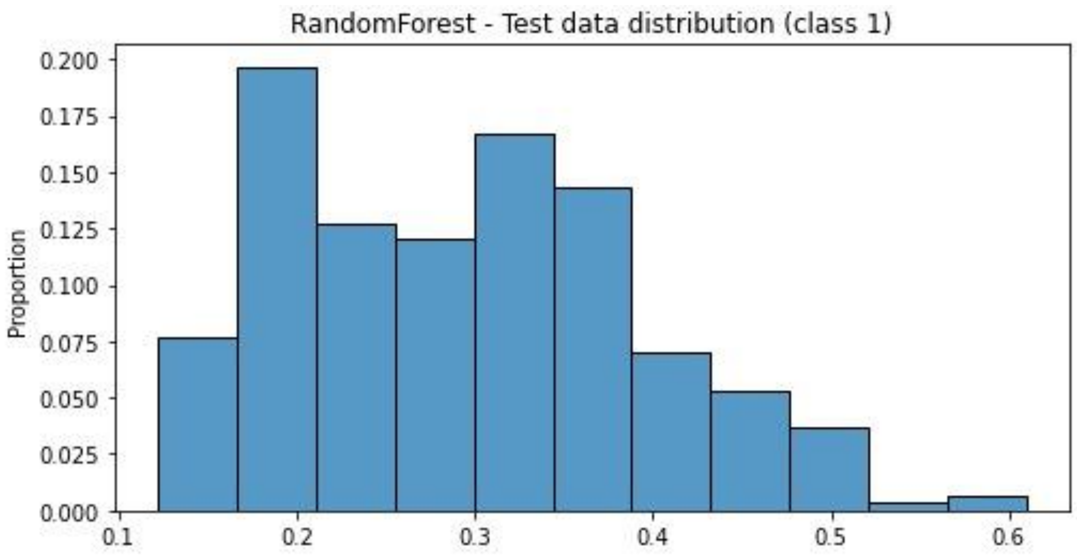
By looking at the above three plots we can analyze the statistics on the score distributions. We can notice that the generator has being able to learn the class 0 (which is also the majority one) better than class 1. We can notice that in the proportions graph which indicates the confidence of the model against the proportions among the test set, we can see that the generator is indeed showing similar patterns but limited bins in compared to random forest model. This might indicate that the confidence score has an impact which limits the confidence score of the generator to learn (each sample receives a unique parameter C which is drown from a uniform distributions). We can notice that the generator is learning to mimic the confidence score of the random forest but in such a manner that is derivate from the parameter C which has a narrower range of values.

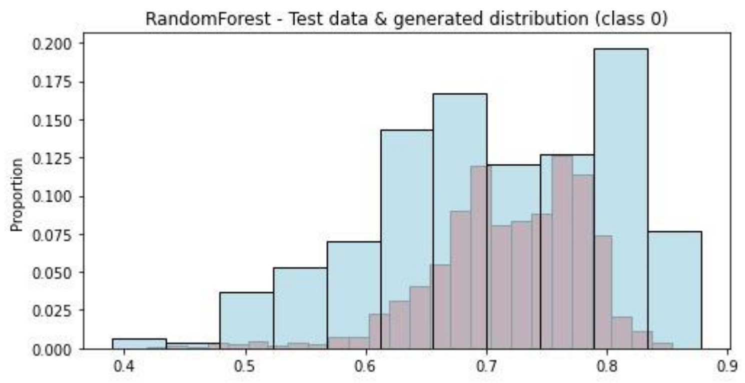
We can notice from the PCA over epochs plots that the generator is indeed learning both classes but is clearly learning more from the majority (0 class) which we sense that the model is trying to best fit that sample. We estimate by the process that the model is trying to converge to the minority class which will cause mode collapse which we will elaborate in the next paragraph.

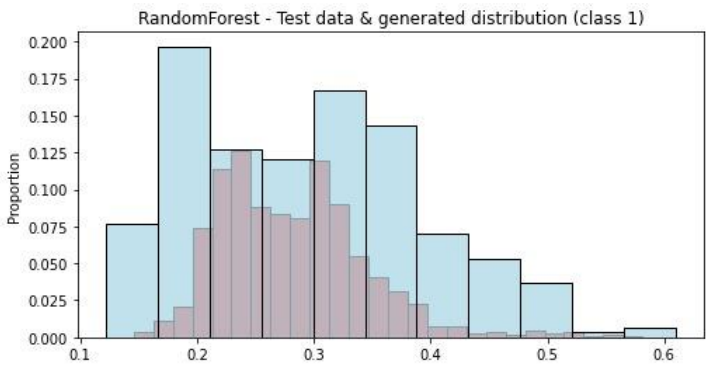
Mode collapse

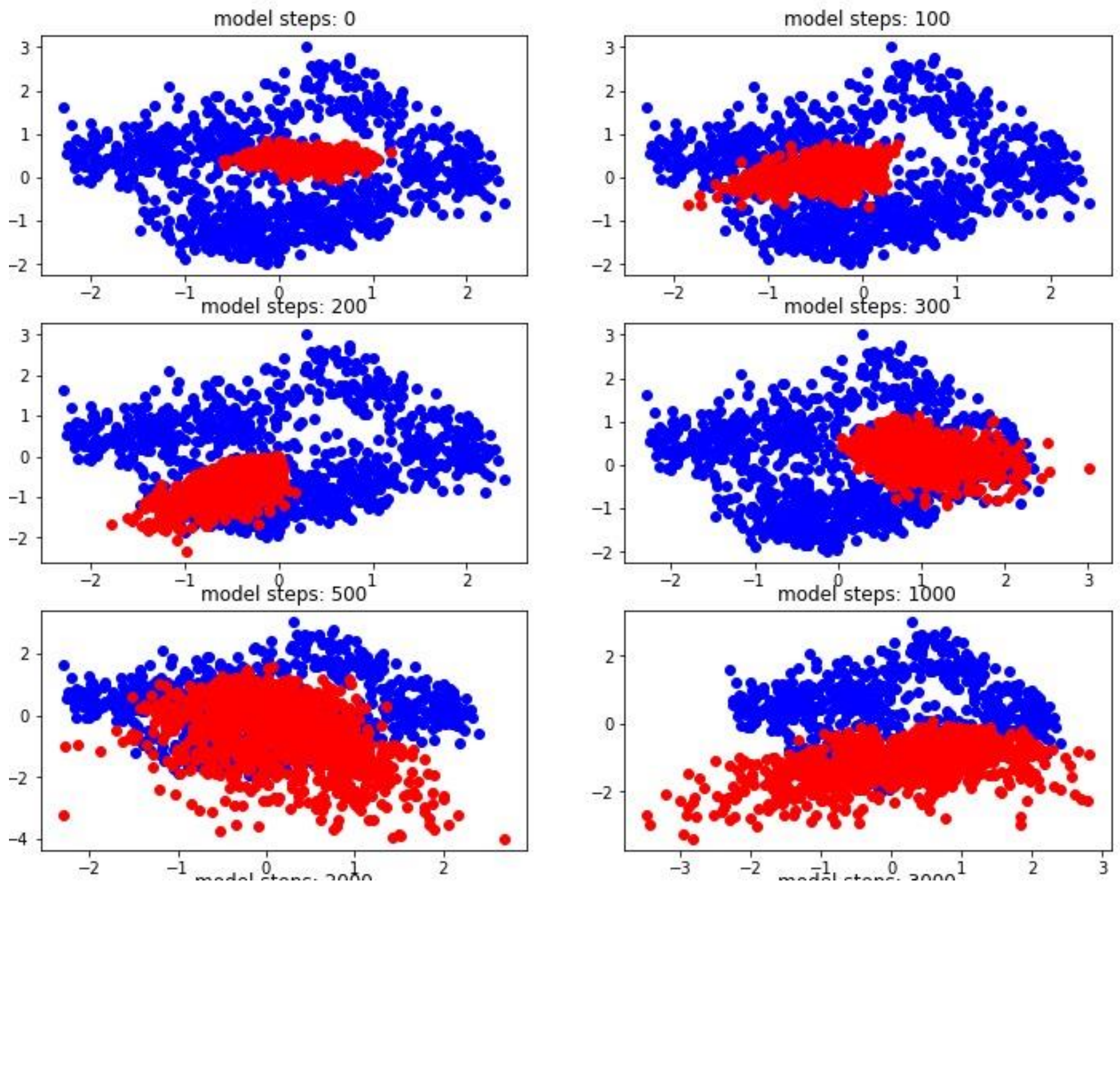
We noticed that the model has suffered from mode collapse, by plotting the PCA over the epochs we can see that while the mode has started to disperse across the sample space, since its random initiation was clearly biased towards the majority class. At the later epochs we can sense that our model is collapsing towards the majority class, we can see less density among non-classes at all, but they are all collapsing into the majority. We weren't surprised since the majority class is almost double the minority, which will cause the generator to focus on the majority more, although that we did see that the model was able to learn distributions from the minority, so if we stop the learning at a certain point we will be able to achieve the scores we wanted.

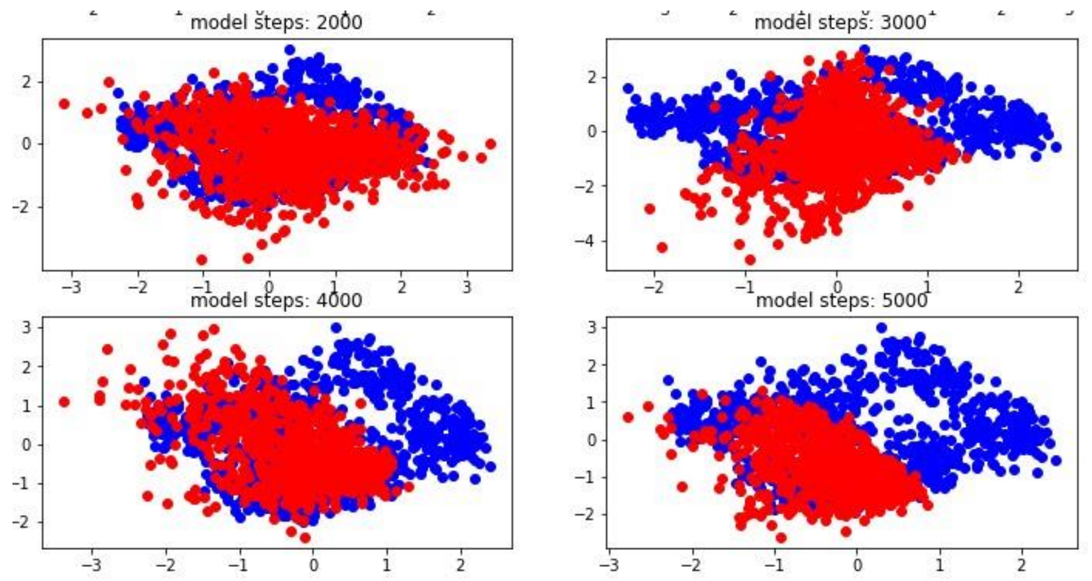
German Credit




RandomForest - Test data & generated distribution (class 0)
Proportion


RandomForest - Test data & generated distribution (class 1)
Proportion


model steps: 0
model steps: 100
model steps: 200
model steps: 300
model steps: 500
model steps: 1000
model steps: 2000
model steps: 3000



By looking at the above three plots we can analyze the statistics on the score distributions.

We can notice that the behavior of the confidence score among frequency looks same as the previous data set. This led us to believe that perhaps what we said regarding the confidence might be correct. The generator tries to mimic the distribution and perform better when learning the distribution of the majority class, in a derivate of the parameter C.

A stronger conclusion could be derived from the PCA over epochs plots, we can sense the mode collapse phenomena much stronger in this dataset. We will further elaborate this conclusion, but we do like to conclude that the model showed similar characteristics between the two dataset which are: Mimicking the distribution but with narrower space and collapsing into the majority.

<u>Mode collapse</u>

We noticed that the model has suffered from mode collapse, by plotting the PCA over the epochs we can see that while the mode has started to disperse across the sample space (epochs 500-3,000). Those epochs really let us believe that the model was able to capture the different classes.

However, when looking at epoch 4,000 and higher, we can clearly see that the model collapsing towards the majority class (class 0). This dataset is already known to us as a bad separated, perhaps when the model got to epoch 4,000 and the model noticed that he could fool the discriminator rather than be spread (since that are points in the margin between the two classes which will cause the model to receive bad signal) caused him to collapse into where he achieves the highest score, i.e., class 0. Since the model hasn't been really learning in the process, in contrast to the above dataset, we won't be able to stop the learning process at the middle.

We would suggest for further work to perform the same further work as mentioned before for this data set, and for further investigation:

1) Perhaps try to eliminate those outliers so the generator will have easier time trying to learn the distribution of the two classes, the data isn't well separated but we believe that those outliers is what caused the generator to fail completely.