313326985 Shahar Shcheranski
206172686 Sarit Hollander

Assignment 4

Preprocessing

Diabetes Dataset

- Normalize the numeric features to range between zero and one using MinMaxScaler.
- Transform the class values to zero or one:
  - $tested\_positive \rightarrow 1$
  - $tested\_negative \rightarrow 0$

German Credit Dataset

- Transform the non-numerical features to numerical features using LabelEncoder.
- Normalize the numeric features to range between zero and one using MinMaxScaler.
- Transform the class values to zero or one:
  - $'1' \rightarrow 1$
  - $'2' \rightarrow 0$

313326985 Shahar Shcheranski
206172686 Sarit Hollander

Part 1 – Generative Adversarial Networks

Architecture

Generator

We used layers of Dense and Batch Normalization with sigmoid output.

```
def define_generator(latent_dim, n_outputs):

  input_noise = Input(shape=(latent_dim,))

  X = Dense(256, activation=LeakyReLU(alpha=0.2))(input_noise)
  X = BatchNormalization()(X)
  X = Dense(256, activation=LeakyReLU(alpha=0.2))(X)
  X = BatchNormalization()(X)

  output = Dense(n_outputs, activation='sigmoid')(X)

  generator = Model(inputs=input_noise, outputs=output)

  return generator
```

Discriminator

We used layers of Dense and Batch Normalization with sigmoid output. We also added Dropout layers to help the Generator.

```
def define_discriminator(n_inputs):

  input = Input(shape=(n_inputs,))

  X = Dense(256, activation=LeakyReLU(alpha=0.2))(input)
  X = BatchNormalization()(X)
  X = Dropout(0.2)(X)
  X = Dense(256, activation=LeakyReLU(alpha=0.2))(X)
  X = BatchNormalization()(X)
  X = Dropout(0.2)(X)

  output = Dense(1, activation='sigmoid')(X)

  discriminator = Model(input, output)

  # compile model
  discriminator.compile(loss='binary_crossentropy', optimizer=Adam(1e-4), metrics=['accuracy'])

  return discriminator
```

313326985 Shahar Shcheranski
206172686 Sarit Hollander

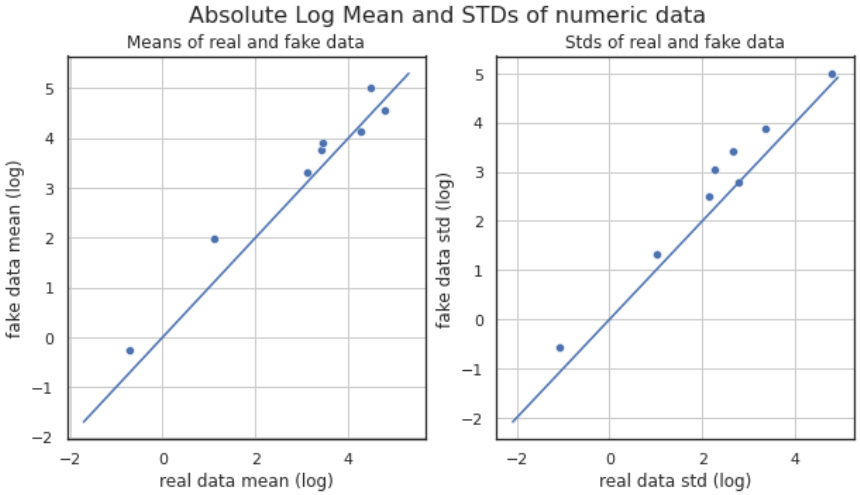
Analyze the products of your model

Diabetes Dataset

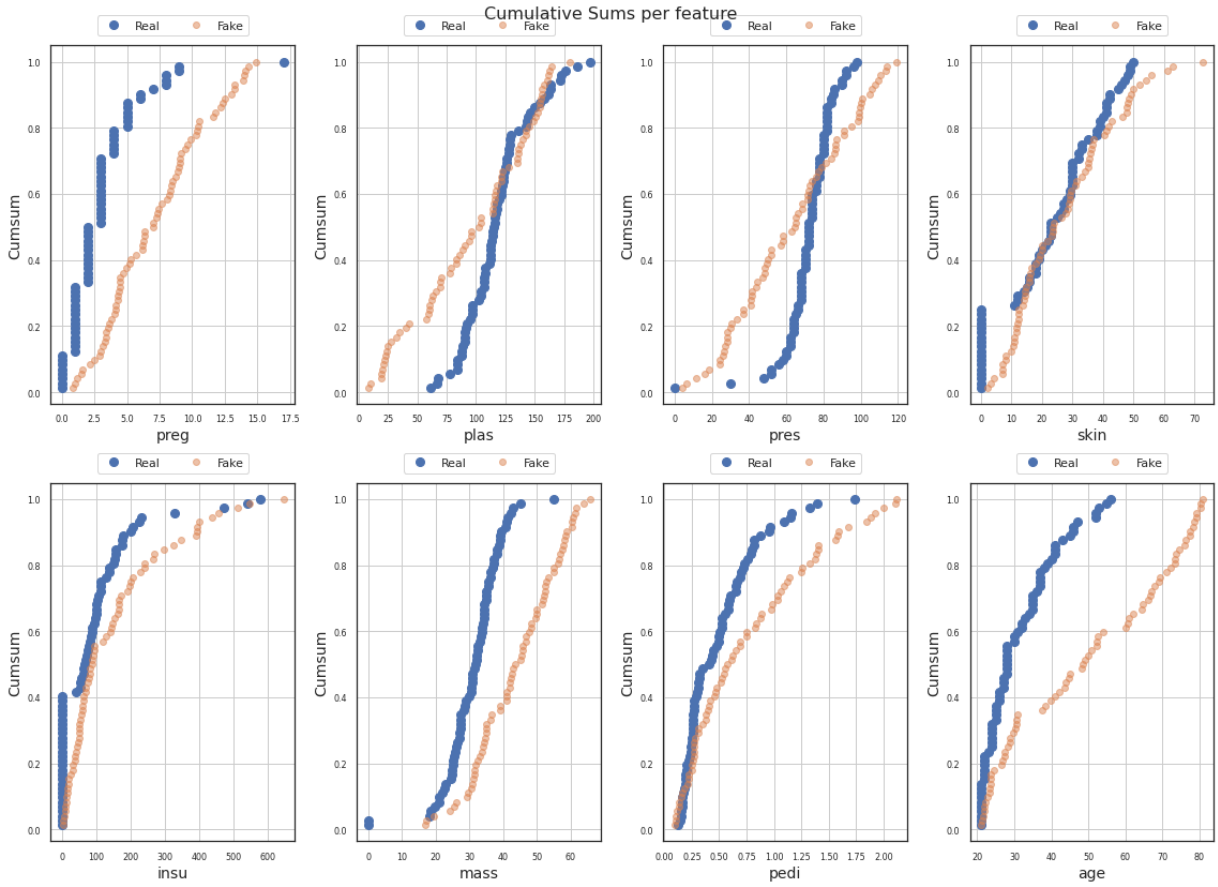
Several samples that "fooled" the detector:

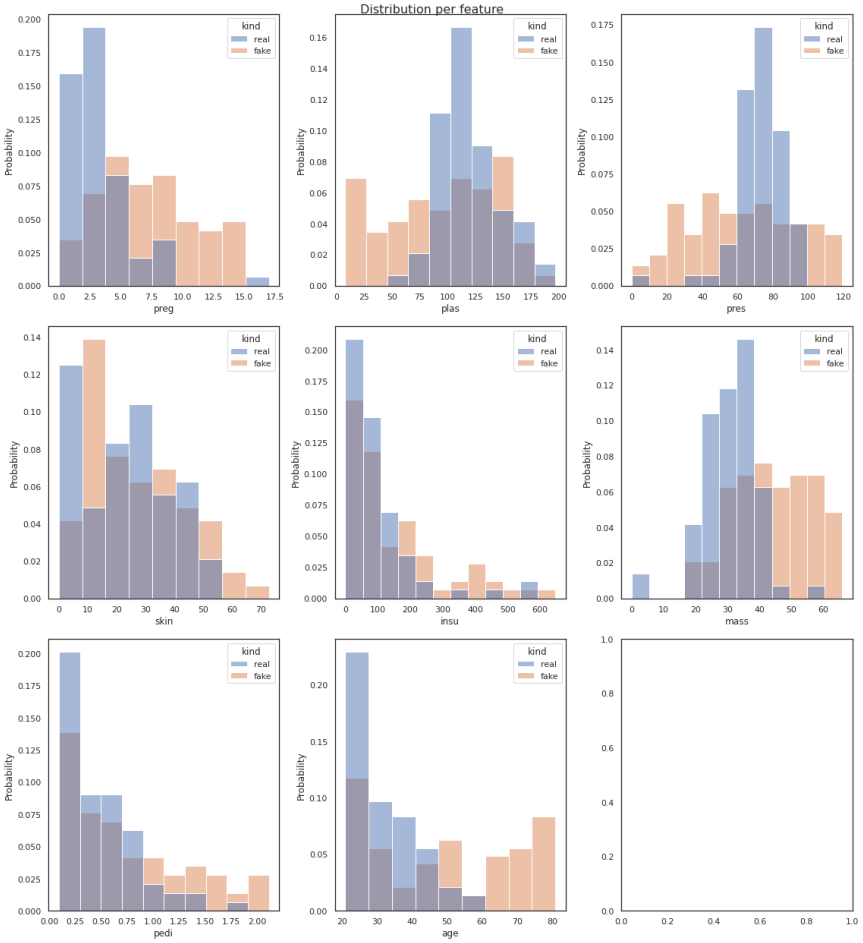
	preg	plas	pres	skin	insu	mass	pedi	age
1	9.442034	78.098351	98.246086	8.477028	192.585037	50.904148	0.227754	80.044655
2	9.882927	83.435226	36.806713	29.056288	72.707130	40.157364	0.440857	26.797838
4	9.862481	60.107925	83.449249	41.497044	166.573029	57.343826	1.227755	71.372681
6	7.956937	17.932354	1.761106	30.031279	3.793034	62.372276	1.003890	23.061424
10	7.311961	119.310211	62.467304	25.068405	138.603714	41.527843	0.641523	56.869781
...	...	...	...	...	...	...	...	...
92	4.905341	85.431511	18.710117	16.605730	4.125288	55.427589	0.174986	21.565638
93	3.998969	117.403900	29.186636	32.383846	50.014587	40.564018	1.053639	29.022583
95	3.347734	108.464409	21.432499	12.703938	10.996850	42.958126	0.252160	21.109182
97	7.719701	125.099167	69.020096	10.128746	40.865253	48.011642	0.335521	21.414043
98	3.467640	180.170822	119.394180	9.641064	296.289459	22.849209	0.117336	63.556438

For evaluate the quality of the samples that "fooled" the detector we used TableEvaluator (is a library to evaluate how similar a synthesized dataset is to a real data)

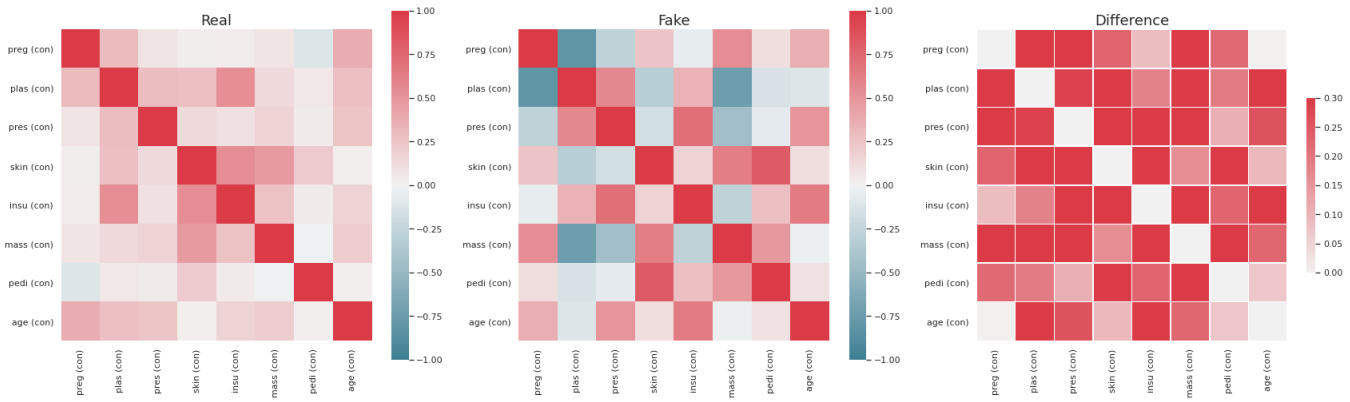


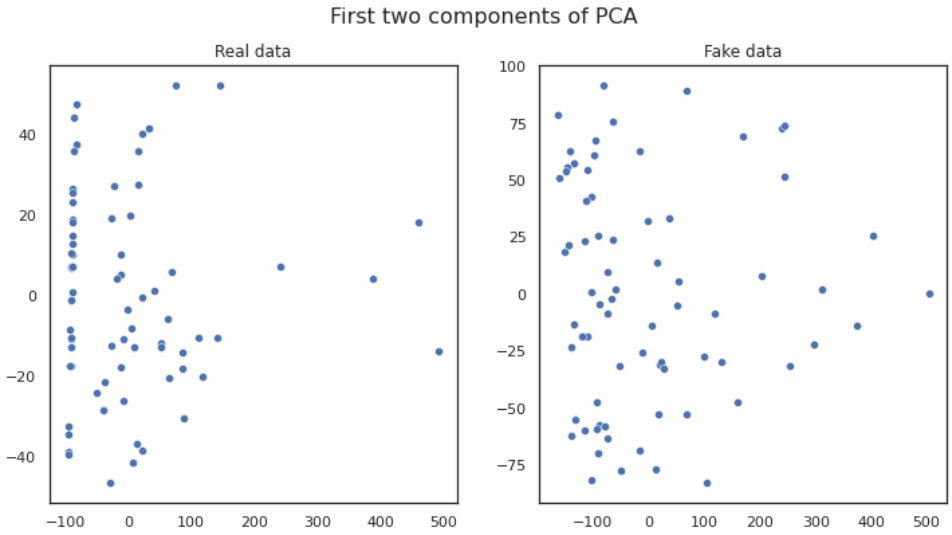
313326985 Shahar Shcheranski
206172686 Sarit Hollander





313326985 Shahar Shcheranski
206172686 Sarit Hollander





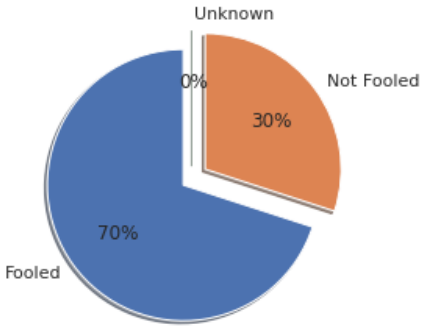
We can see from the cumulative sums for each attribute and the distribution for each attribute that some of the attributes closely match those of real data. From the PCA visualization we can see that the fake samples are as close to the left side as the real samples, but the fake samples are more diverse.

Several samples that did not "fooled" the detector:

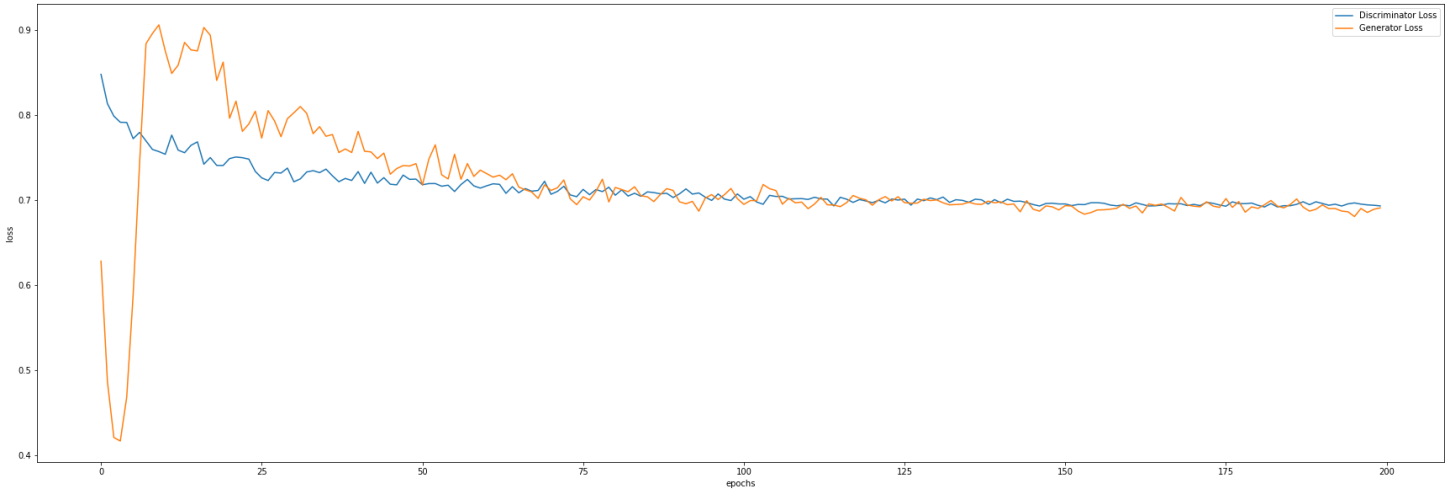
	preg	plas	pres	skin	insu	mass	pedi	age
0	1.694592	160.474915	104.341476	40.742161	127.759933	52.639233	0.601475	53.049149
3	7.391677	108.713997	80.559097	24.563471	106.057770	53.560806	0.631154	29.856232
5	1.831642	126.164871	73.107201	9.711237	15.168024	45.423538	0.110558	24.590118
7	5.472180	102.814270	42.403992	10.839455	21.665077	36.026962	0.377317	28.598471
8	3.871952	159.178879	54.256973	5.249526	42.166409	29.304565	0.162663	30.062574
9	9.652917	103.564781	67.415565	18.960186	55.245956	29.130213	0.306628	25.139990
12	3.950675	78.842972	66.643723	29.068222	89.693802	49.094040	0.980786	29.712505
23	1.443954	189.376221	62.793945	6.327091	14.988787	36.632286	0.184391	21.119137
26	10.129294	73.256203	42.719482	38.092323	470.656067	46.115063	1.449435	54.459251
28	4.347391	128.377258	69.082642	29.534672	41.789677	51.153507	0.715926	22.975267
29	7.255510	34.201679	24.814558	31.539043	49.814423	57.155655	1.361347	26.177217
30	4.176864	118.807472	82.781296	46.550068	263.817963	57.108200	0.801914	48.432194
35	3.557608	129.675919	28.017742	27.183384	11.436233	46.583271	0.819137	21.159389
38	5.008736	98.040703	71.598656	42.845997	373.091370	40.741505	1.346295	44.630188
39	3.462734	83.633430	52.183632	21.577877	118.515579	50.723423	0.861790	49.443390

313326985 Shahar Shcheranski
206172686 Sarit Hollander

Distribution of 100 samples at random after our models converged:



A graph describing the loss of the generator and the discriminator:



As we can see, the models are going "back and forth" with their losses. We also can see that the Discriminator and the Generator are "fighting" to take the lead and there is no consistent leader.

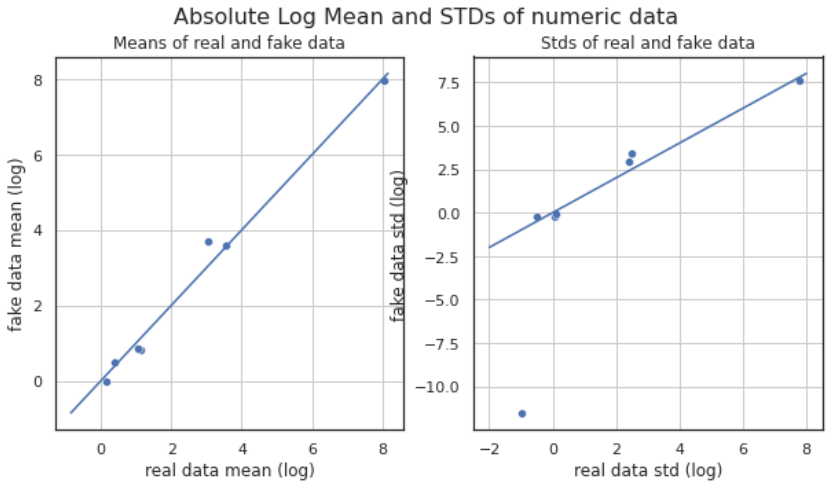
313326985 Shahar Shcheranski
206172686 Sarit Hollander

German Credit Dataset

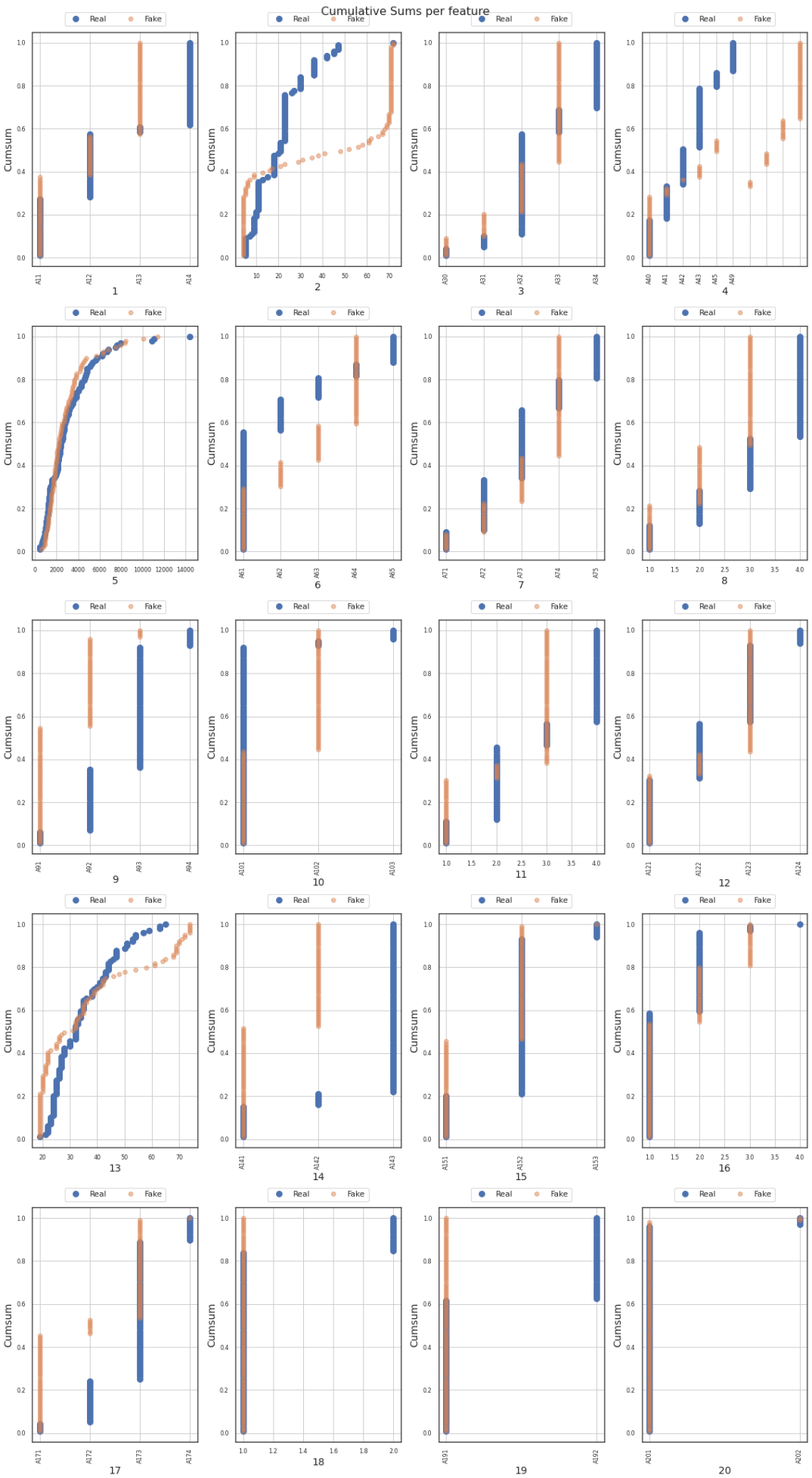
Several samples that "fooled" the detector:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	A11	4	A33	A40	6820	A61	A71	2	A91	A101	1	A123	74	A142	A151	1	A173	1	A191	A201
1	A11	4	A32	A40	1704	A62	A71	1	A91	A101	1	A123	61	A142	A151	1	A173	1	A191	A201
2	A12	35	A33	A410	901	A62	A74	2	A91	A102	3	A123	26	A142	A151	2	A171	1	A191	A201
3	A11	4	A32	A40	1514	A61	A72	1	A92	A101	1	A123	39	A142	A151	2	A173	1	A191	A201
4	A13	72	A33	A48	11419	A64	A74	3	A93	A102	3	A121	19	A141	A152	3	A171	1	A191	A202
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	A13	71	A33	A43	2842	A63	A74	3	A92	A102	3	A121	20	A141	A152	3	A171	1	A191	A201
96	A13	61	A33	A48	1493	A64	A74	3	A91	A102	3	A123	28	A141	A152	1	A173	1	A191	A201
97	A11	67	A33	A41	3317	A61	A74	3	A91	A102	1	A123	69	A142	A152	3	A171	1	A191	A201
98	A13	71	A31	A48	2055	A64	A74	2	A92	A102	3	A121	19	A141	A152	2	A171	1	A191	A201
99	A13	56	A32	A48	1865	A64	A74	1	A91	A101	3	A123	21	A141	A152	1	A172	1	A191	A201

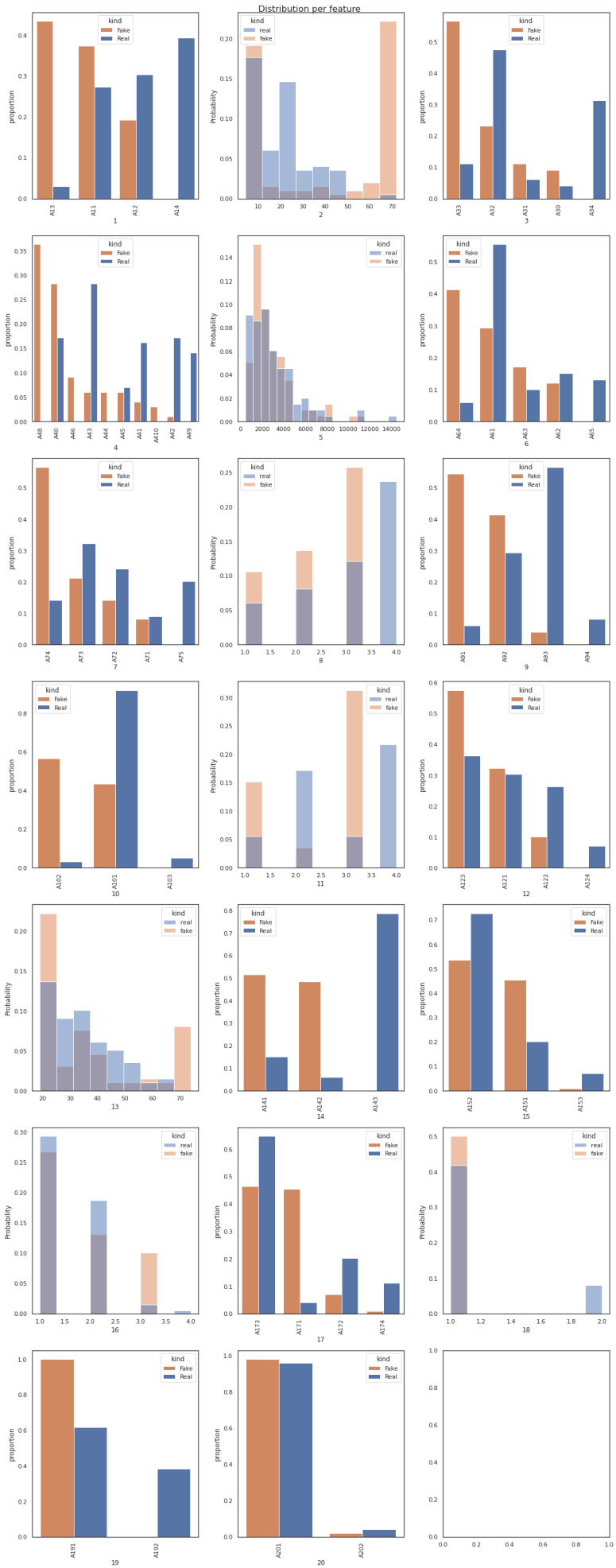
For evaluate the quality of the samples that "fooled" the detector we used TableEvaluator (is a library to evaluate how similar a synthesized dataset is to a real data)



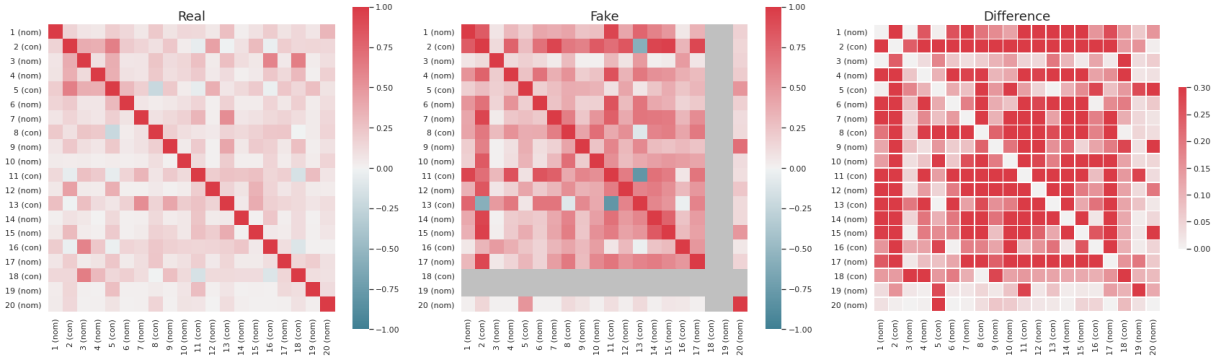
313326985 Shahar Shcheranski
206172686 Sarit Hollander

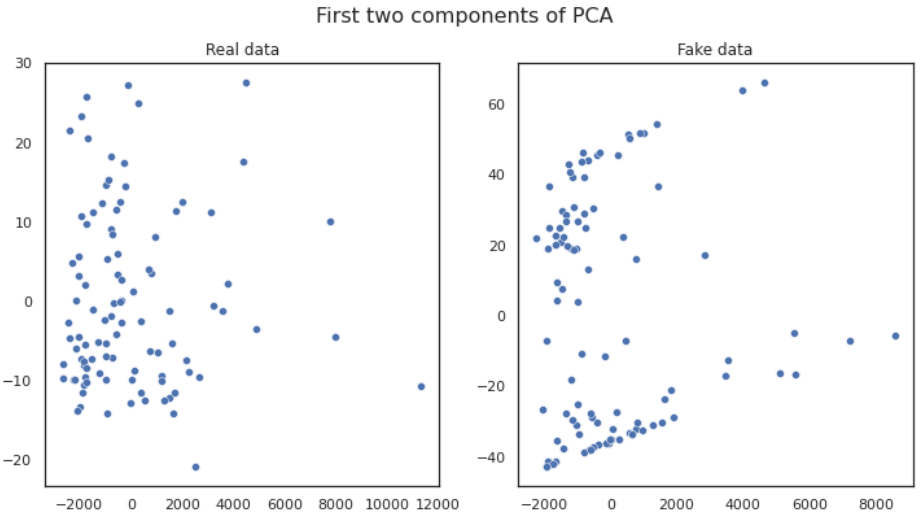


313326985 Shahar Shcheranski
206172686 Sarit Hollander



313326985 Shahar Shcheranski
206172686 Sarit Hollander



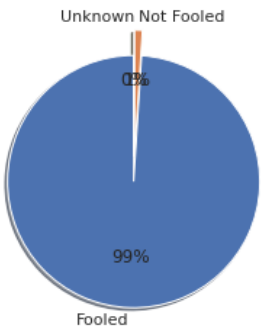


We can see from the cumulative sums for each attribute and the distribution for each attribute that some of the attributes closely match those of real data. From the PCA visualization we can see that the fake samples are as close to the left side as the real samples, but the fake samples are more scattered at the top or bottom.

Several samples that did not "fooled" the detector:

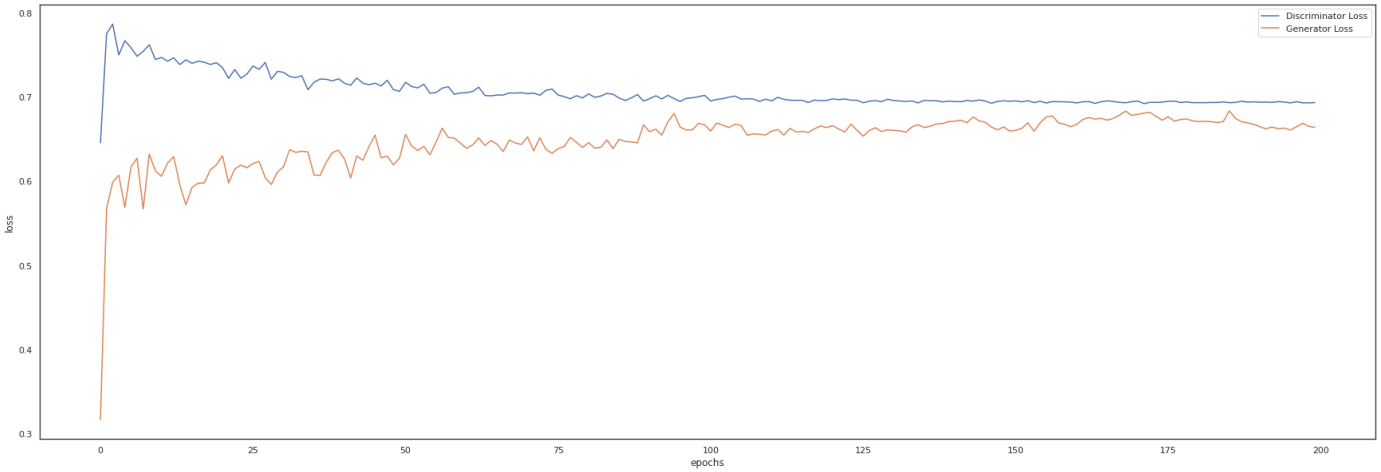
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
15	A11	9	A33	A41	1179	A62	A74	3	A91	A102	2	A123	38	A142	A151	2	A173	1	A191	A201

Distribution of 100 samples at random after our models converged:



313326985 Shahar Shcheranski
206172686 Sarit Hollander

A graph describing the loss of the generator and the discriminator



As we can see, the models are not going "back and forth" with their losses. We also can see that the Discriminator is consistent leader.

313326985 Shahar Shcheranski
206172686 Sarit Hollander

Part 2 - Generative model for sample generation

Architecture

Generator

We first concatenate the input noise ‘z’ with input ‘c’. We used layers of Dense and Batch Normalization with sigmoid output.

```
def define_twist_generator(noise_dim, n_outputs):
  input_z = Input(shape=noise_dim)
  input_c = Input(shape=(1,))

  input = Concatenate()([input_z, input_c])

  X = Dense(256, activation = LeakyReLU(alpha=0.2))(input)
  X = BatchNormalization()(X)
  X = Dense(256, activation= LeakyReLU(alpha=0.2))(X)
  X = BatchNormalization()(X)

  output = Dense(n_outputs, activation='sigmoid')(X)

  model = Model(inputs=[input_z, input_c], outputs=output)


  return model
```

Discriminator

We first concatenate the input noise ‘z’ with input ‘c’ and input ‘y’. We used layers of Dense and Batch Normalization with sigmoid output. We also added Dropout layers to help the Generator.

```
def define_twist_discriminator(n_inputs):
  input_s = Input(shape=n_inputs)
  input_c = Input(shape=(1,))
  input_y = Input(shape=(1,))

  input = Concatenate()([input_s, input_c, input_y])
  X = Dense(256, activation = LeakyReLU(alpha=0.2))(input)
  X = BatchNormalization()(X)
  X = Dropout(0.2)(X)
  X = Dense(256, activation = LeakyReLU(alpha=0.2))(X)
  X = BatchNormalization()(X)
  X = Dropout(0.2)(X)
  output = Dense(1, activation='sigmoid')(X)

  model = Model(inputs=[input_s, input_c, input_y], outputs=output)

  # compile model
  model.compile(loss='binary_crossentropy', optimizer=Adam(1e-4), metrics=['accuracy'])

  return model
```

313326985 Shahar Shcheranski
206172686 Sarit Hollander

Results and Statistics

Diabetes dataset

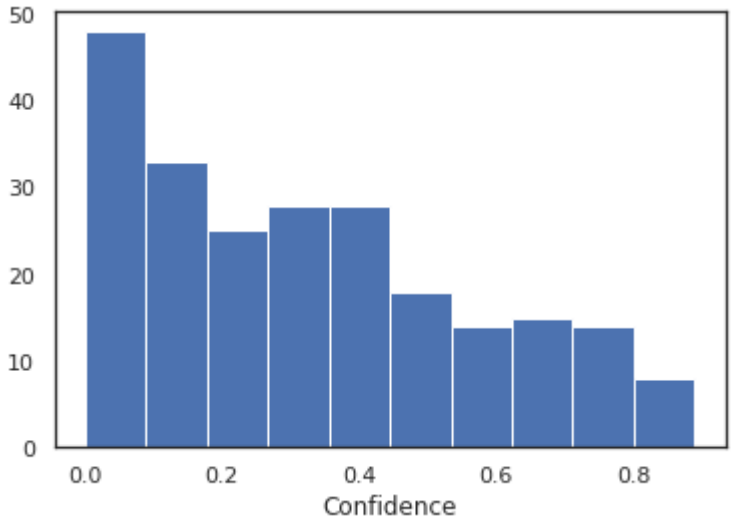
Random forest performance:

Accuracy: 75.75757575757575%

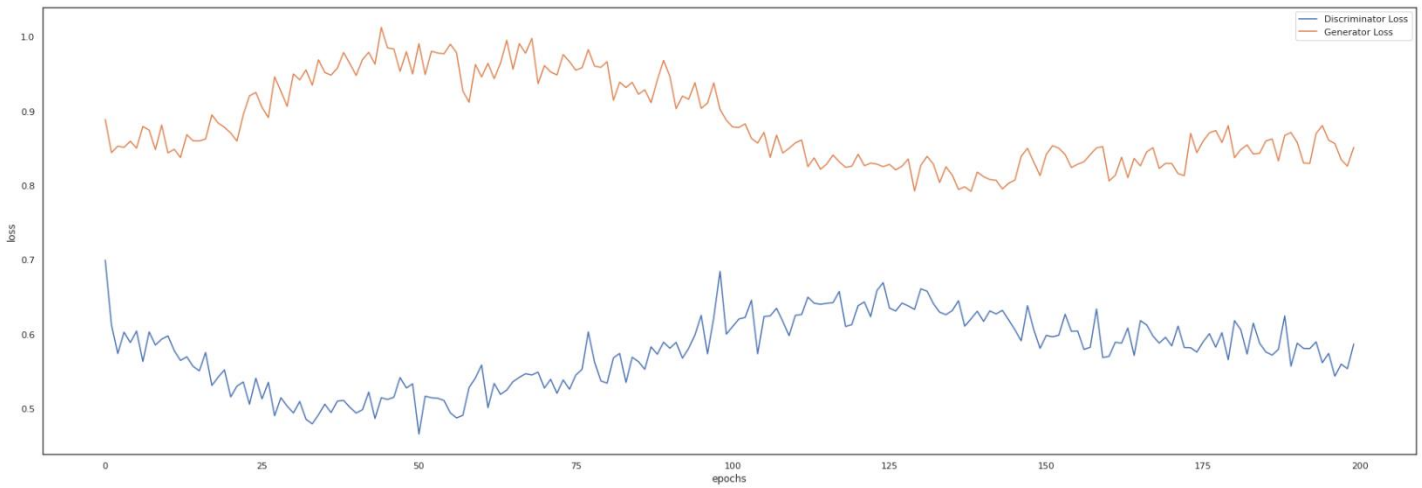
Max confidence: 0.89

Min confidence: 0.0

Average confidence: 0.3295670995670996



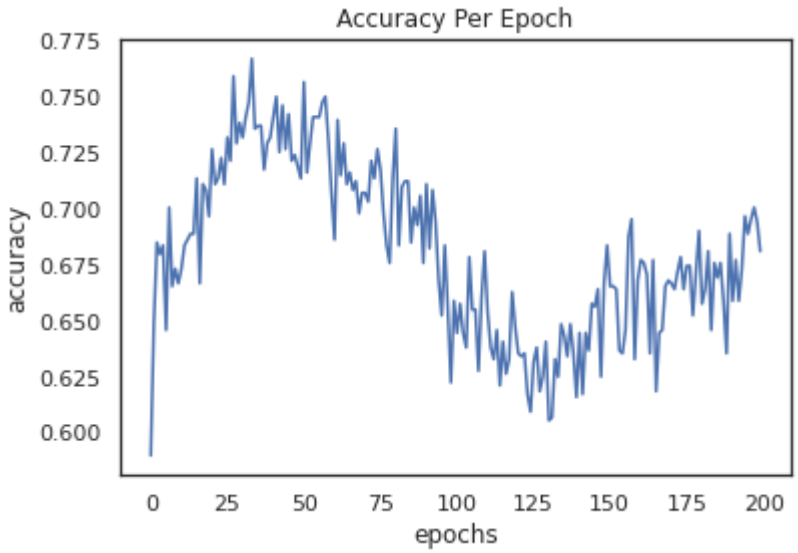
Generator and Discriminator loss per epoch:



At first we can see the generator loss is increasing and the discriminator's loss is decreasing since the discriminator task is easier compared to the generator task, so

first the discriminator gets meaningful feedback and then it improves. The generator loss is increasing because the discriminator is improving in classifying. Later, the generator loss is improving (~epoch 100) and the discriminator loss is increasing.

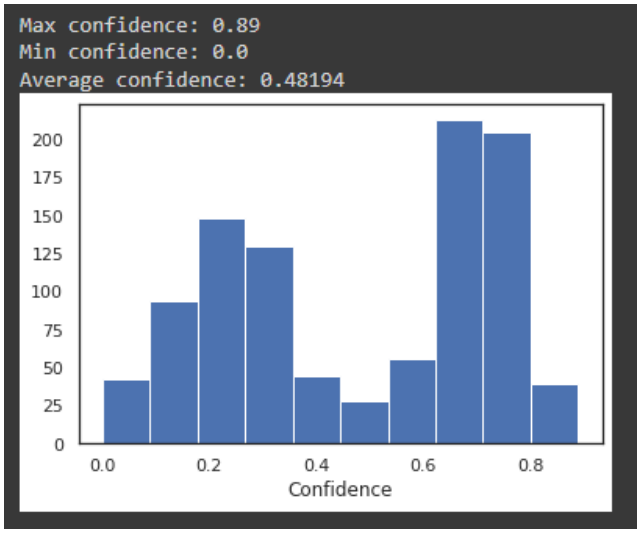
Accuracy per epoch:



We can see from this plot that the discriminator at the beginning is able to distinguish between c & y, and as the generator start to improve the accuracy is decreasing.

- We use 200 epochs

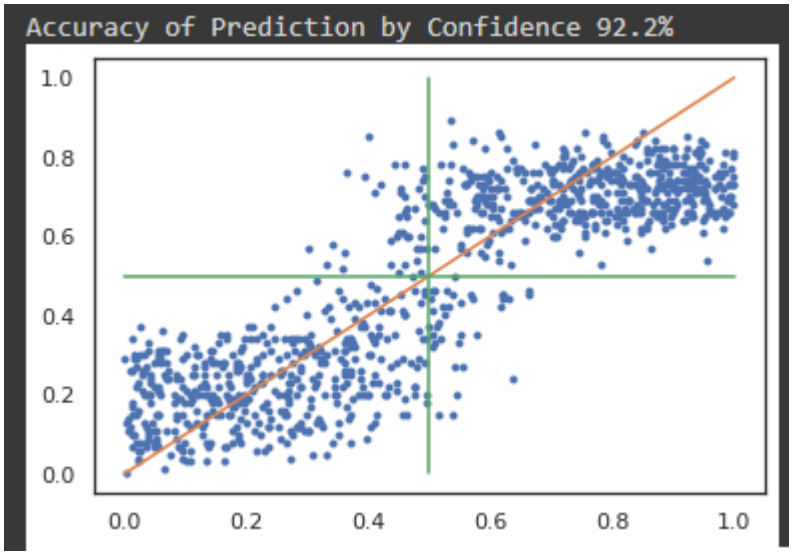
Distribution of the random forest on the generated 1000 random uniform samples:



We can see that the model is failed to generate samples with the confidence of 0-0.2, 0.4-0.6 and 0.8

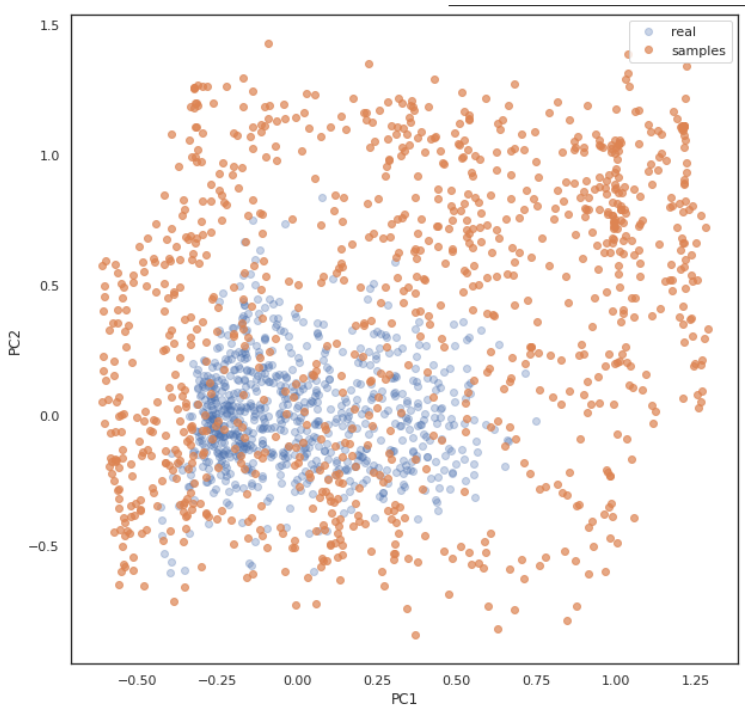
313326985 Shahar Shcheranski
206172686 Sarit Hollander

Predication of the random forest as a function of the confidence:



It would be perfect if the samples were on the orange line. In our case we can see that the samples are around this line but still varied. As for the accuracy, we calculated how much the confidence of the generator is agrees with the values from the random forest. We achieved accuracy of 92.2%

Distribution of the samples using PCA:



We can see that the samples are much varied compared to the real data.

313326985 Shahar Shcheranski
206172686 Sarit Hollander

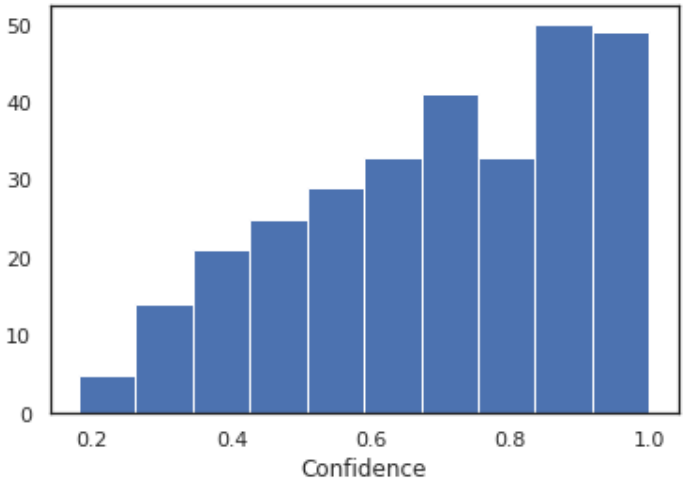
German Credit:

Accuracy: 76.66666666666667%

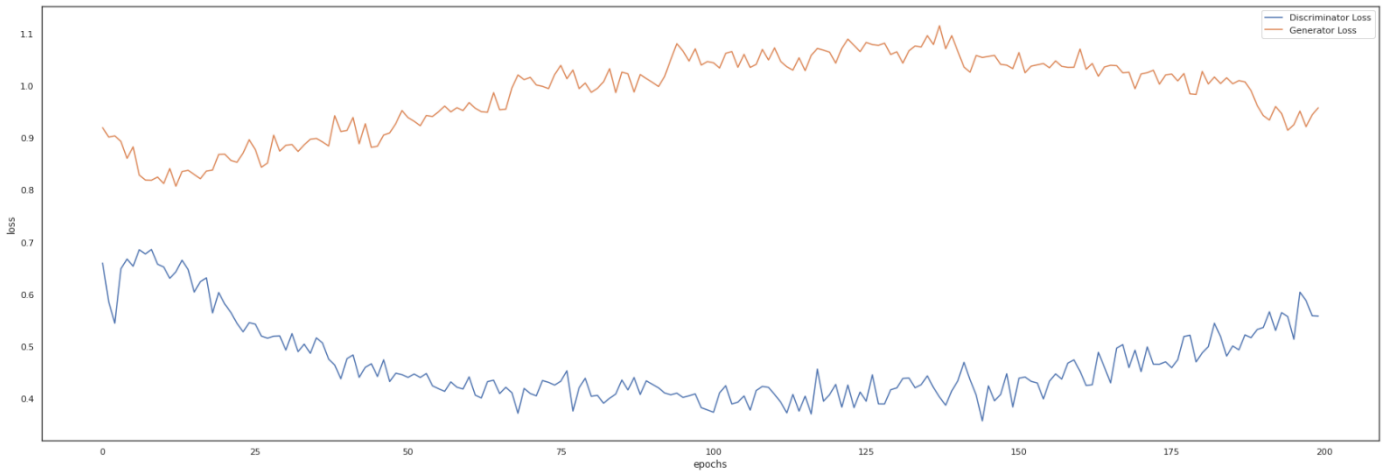
Max confidence: 1.0

Min confidence: 0.18

Average confidence: 0.6954666666666667



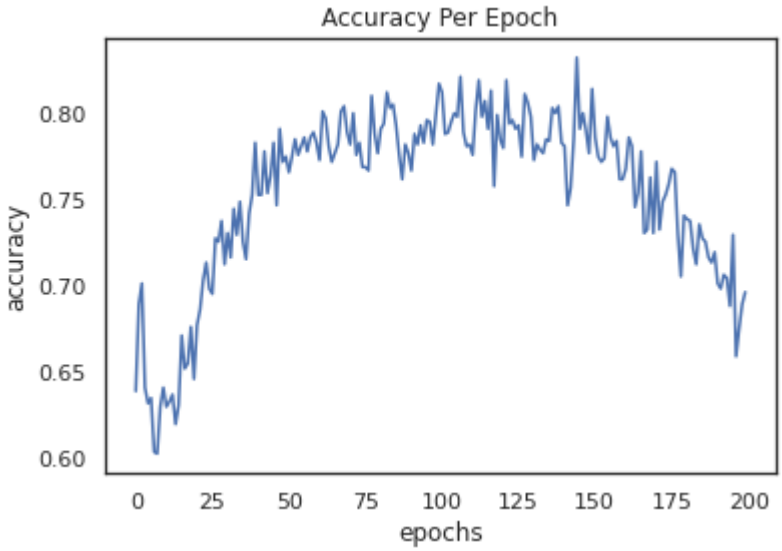
Generator and Discriminator loss per epoch:



At first we can see the generator loss is increasing and the discriminator's loss is decreasing since the discriminator task is easier compared to the generator task, so first the discriminator gets meaningful feedback and then it improves. The generator loss is increasing because the discriminator is improving in classifying. At the end it seems like the generator loss decreasing a bit while the discriminator loss is increasing.

313326985 Shahar Shcheranski
206172686 Sarit Hollander

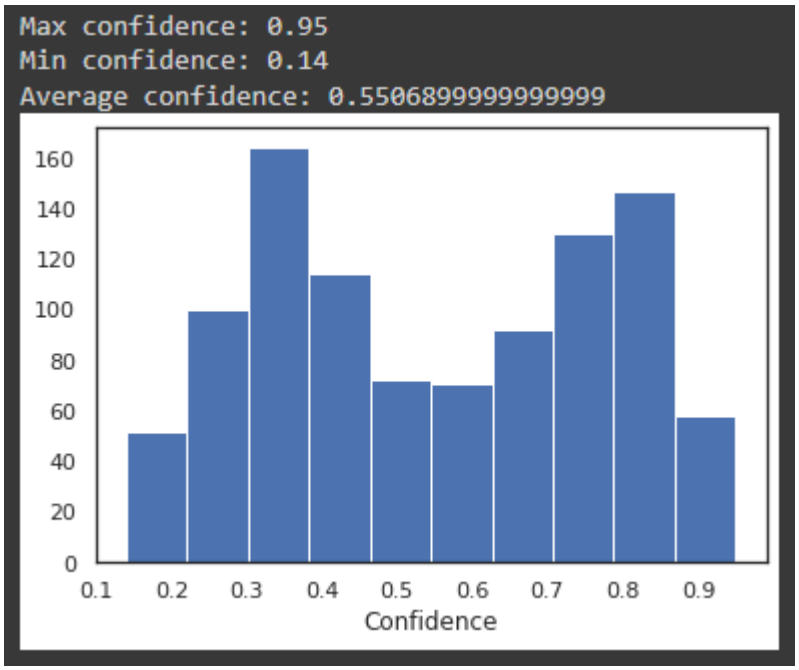
Accuracy per epoch:



We can see from this plot that the discriminator at the beginning is able to distinguish between c & y, and around epoch 150 the accuracy is decreasing as the discriminator loss start to increase.

- We use 200 epochs

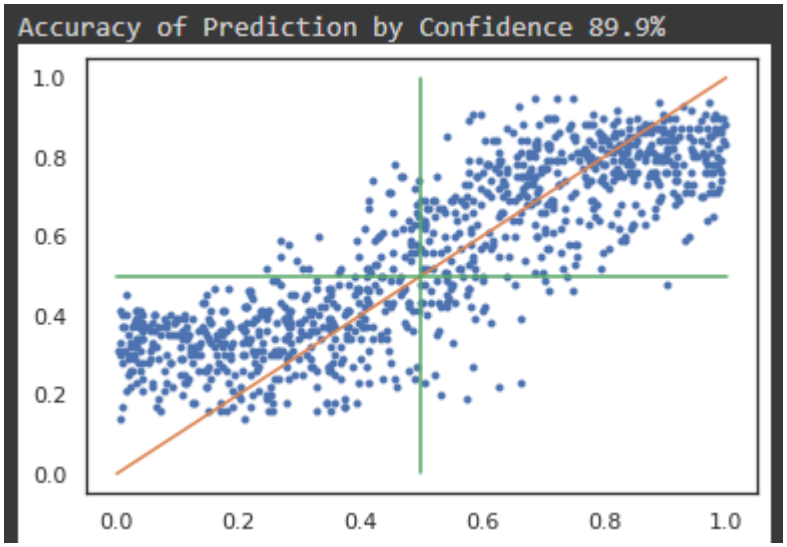
Distribution of the random forest on the generated 1000 random uniform samples:



We can see that the model is mostly failed to generate samples with the confidence of 0.1-0.2 and 0.9.

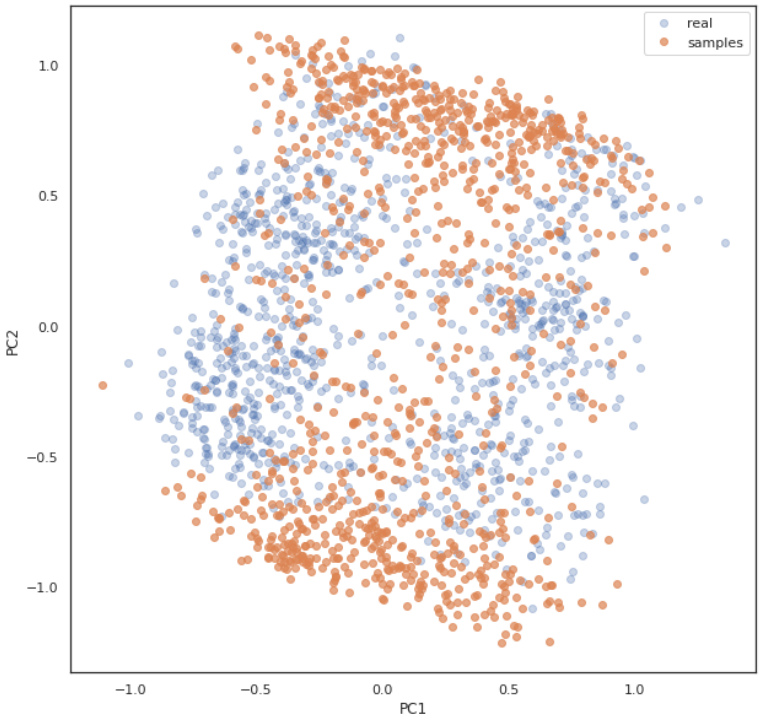
313326985 Shahar Shcheranski
206172686 Sarit Hollander

Predication of the random forest as a function of the confidence:



It would be perfect if the samples were on the orange line. In our case we can see that the samples are around this line but still varied. As for the accuracy, we calculated how much the confidence of the generator is agrees with the values from the random forest. We achieved accuracy of 89.9%

Distribution of the samples using PCA:



We can see that the samples are varied and cover the real data but are also most of them are focus on the upper and bottom parts of the real data.