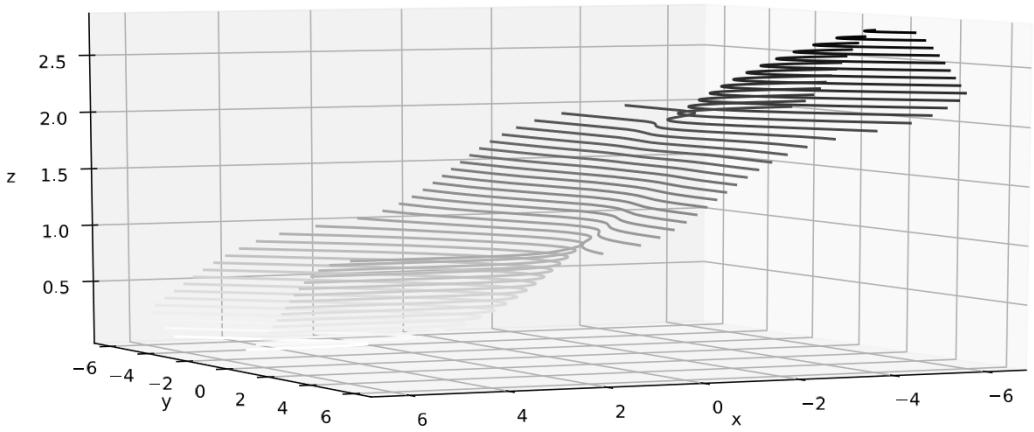
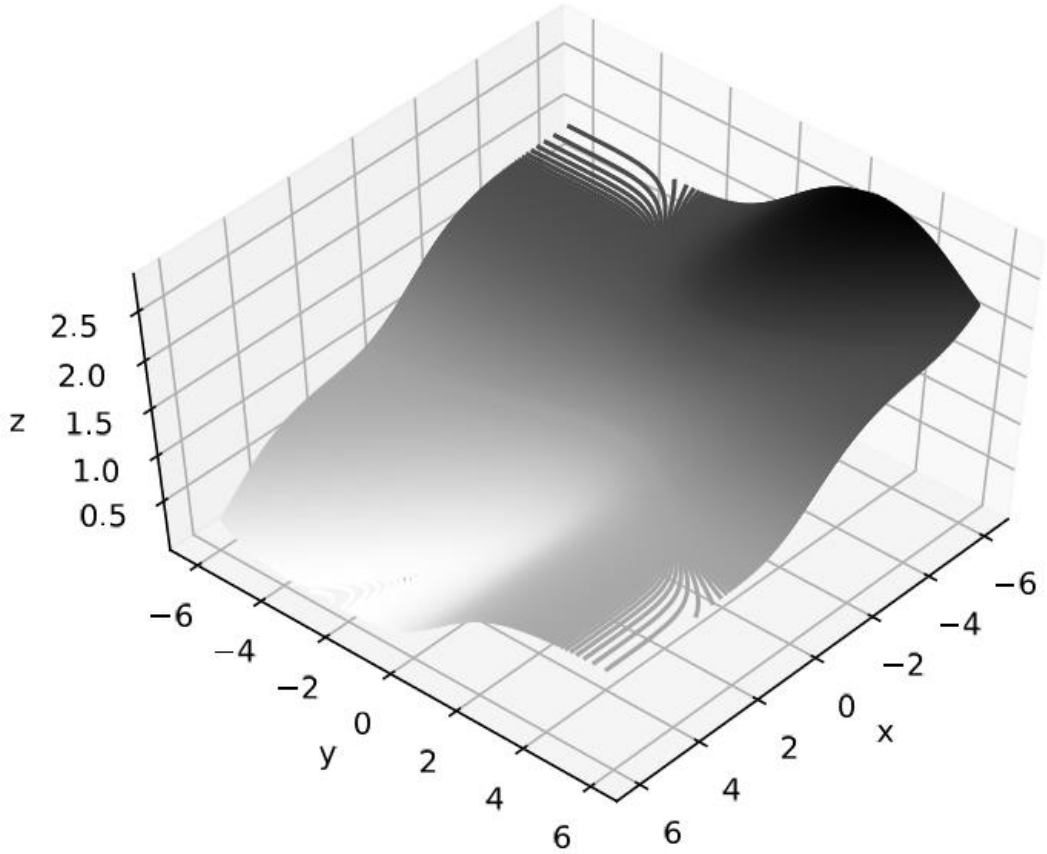
Task 1.1

Done in skeleton.py, in plot_L_simple().





Reading from the figure it seems that (w_1, w_2) = (4.5, -2.5) is a pretty good estimate for locating the minimum of L_{simple}. At first glance the minimum is between 0 and 0.5, without being completely sure.

Task 1.2

See appendix problem1b_figure.pdf

Task 1.3

This is done in skeleton_v2.py, in gradient_descent().

See appendix problem1c_figure.pdf and problem1c_loss_eta.pdf.

Note: when the learning rate is a small number, we need more iterations to come closer to the minimum. This is because the algorithm will not move too much in the correct direction due to the small learning rate. When the learning rate is increased, we see improvements/quicker convergence. But when the learning rate is too big, we see that we move away from the expected minimum, which we observed to be around (4,-2). This means that we do not converge to the expected value, because each time the update rule is used, the learning rate will overcompensate so the new estimate is “on the other side” of the minima, and we will hence never converge towards (4,-2).

Task 2.1

See appendix problem2a_figure.pdf

Task 2.2

This This is done in skeleton_v2.py, in batch_train_w and stochast_train_w.

Small_nonsep

For the default settings (learning rate = 0.1 and 10 iterations), we get the following for stochastic and batch, respectfully:

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156567097
error= 0.43
Total duration:  1.1720311642
error= 0.5
```

Since there are few iterations here/the learning rate is low, we will get a high percentage of errors. But as seen by the screenshot, with (only) 10 iterations the batch algorithm spends over 100x more time computing than the stochastic.

The number of iterations were raised to 100 and several tests were deducted. The general trend seemed to be summarized by the screenshot below:

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156261921
error= 0.275
Total duration:  11.7296845913
error= 0.37
```

Both algorithms drop their error rate when the number of iterations increase, but the batch algorithm shows again how slow it is computationally. The time for the stochastic algorithm seems to be untouched by the number of iterations while dropping its error rate!

For small_nonsep the general trend is that stochastic has fewer errors and spends less time computing. It seems that the batch algorithm is the tougher one to train, because it has more errors and spends more time computing the results.

Big_nonsep

For learning rate = 0.1 and 10 iterations the trends were as follows:

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0171926022
error= 0.502
Total duration:  4.2463667393
error= 0.494
```

The two algorithms had less difference in error now, but the batch algorithm was still the slower computationally, by far. The error was in general significantly higher than with small_nonsep.

Increasing the iterations to 100 again:

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0200514793
error= 0.285
Total duration:  42.3882074356
error= 0.47
```

The error for the batch algorithm did not seem to change too much, while the computation time increased severely. The error in stochastic dropped and the computation time did not increase significantly.

For big_nonsep the general trend is that both algorithms have the same error rate when the number of iterations are small and stochastic still spends less time computing. It seems that the batch algorithm is the tougher one to train, because the error rate persists, even after increasing the number of iterations and it spends more time computing the results.

Small_separable

Learning rate = 0.1 and iterations = 10.

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0030071735
error= 0.5
Total duration:  0.9391644001
error= 0.5
```

Stochastic was even faster now, but so was the batch. Both had roughly the same error rate.

Increasing the number of iterations to 100:

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0060153008
error= 0.005
Total duration:  8.7784719467
error= 0.11
```

Now, the error dropped by major degree on both algorithms.

For small_separable the general trend is that both algorithms have the same error rate when the number of iterations are small and stochastic still spends less time computing. Error rates for both fall when the number of iterations increase, but the computation time for the batch algorithm is still a problem. It seems that the batch algorithm is easier to train for the small separable dataset than the previous non-separable. The batch computation time is significantly lowered now compared to the non-separable tests, and the error for separable test data is also lower.

Big_separable

Learning rate = 0.1, iterations = 10.

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0149607658
error= 0.494
Total duration:  4.3254332542
error= 0.5
```

Same as before. Roughly same error rate, batch slower than stochastic

Iterations increased to 100:

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0312538147
error= 0.045
Total duration:  54.0577692986
error= 0.033
```

The error of stochastic is very low now, and the computational time is again only slightly increased. The batch time, on the other hand, is close to a minute now. But on the bright side, it is the first time that the batch algorithm has lower error rate than the stochastic

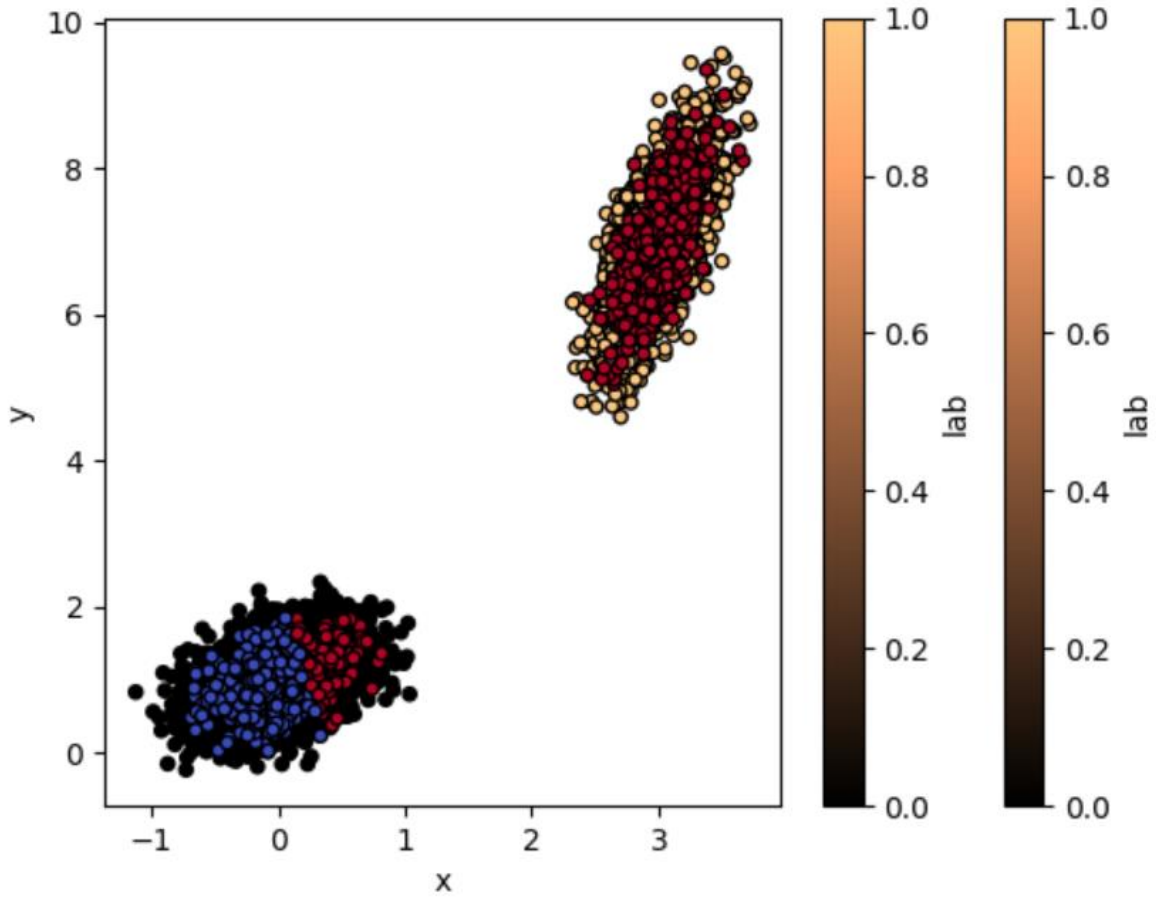
The general trend stays almost the same as before. Almost the same error rate (which is very small now when iterations are increased). I would still say that batch is tougher to train when it takes a minute only to get an error rate that is basically the same as the one you get from the stochastic in 0.03s.

Scatter plot for big_separable with stochastic gradient descent

Learning rate = 0.1, iterations = 100

Blue = classified as 0

Red = classified as 1

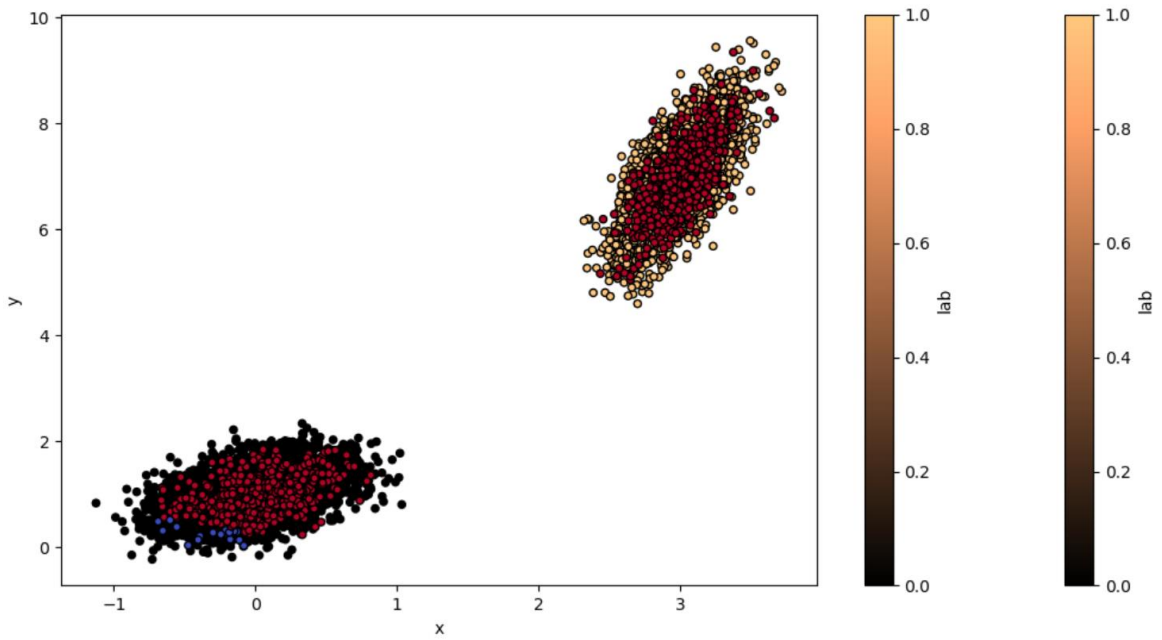


The red dots in the lower leftmost shape are (most of) the errors in the model.

```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156257153
error= 0.112
```

As seen the results here are pretty good but could be better if all the test points in the lower lefthand corner were blue, because all the training points there are classified as 0, so the test points should also be classified as 0.

The model is pretty good, but if we decrease the number of iterations to 10 (as we had in the previous discussions) we will see that the quality of the results are decreasing:



There are even more red (classified as 1) test points in the left corner, but it should be (ideally) all blue.

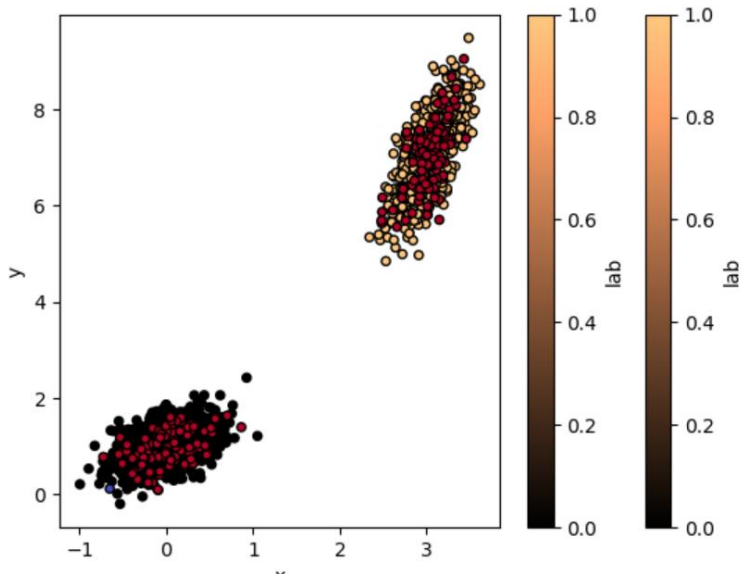
```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0155982971
error= 0.483
```

This also corresponds to the increased error rate.

One dataset + one training algorithm

Dataset: small_separable
Training algorithm: stochastic

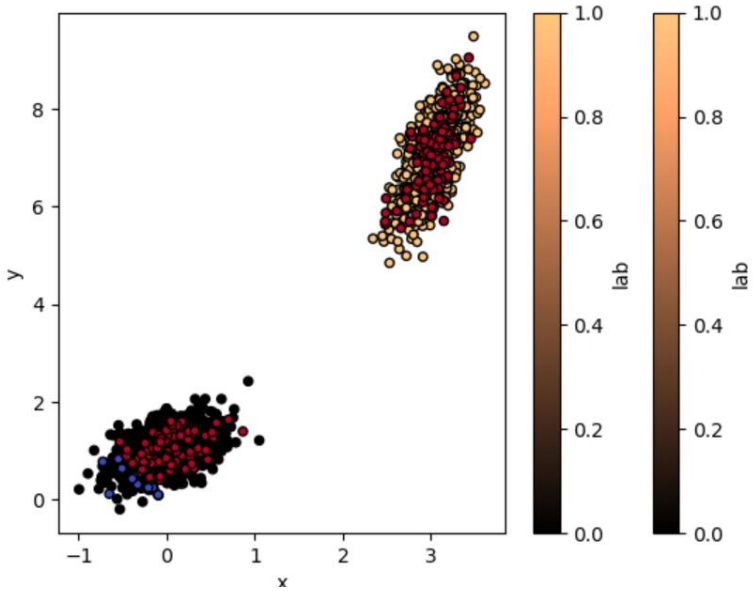
10 iterations:




```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156261921
error= 0.495
```

Almost all 0-classified training points are wrong, hence almost 50% error rate

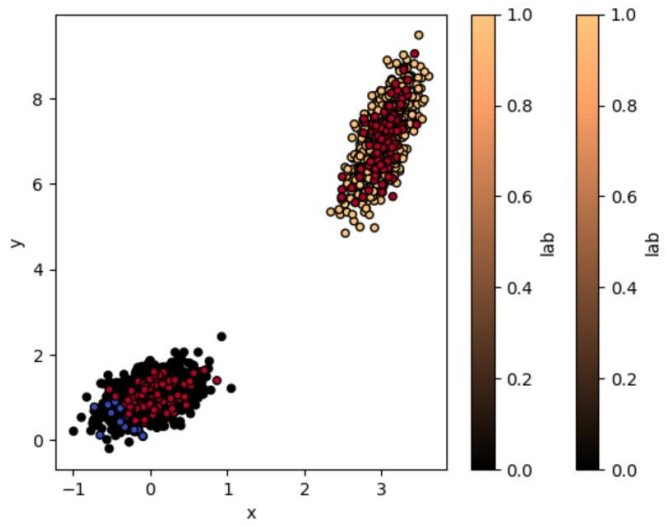
20 iterations:



```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156261921
error= 0.445
```

Some improvements, but still a high error rate

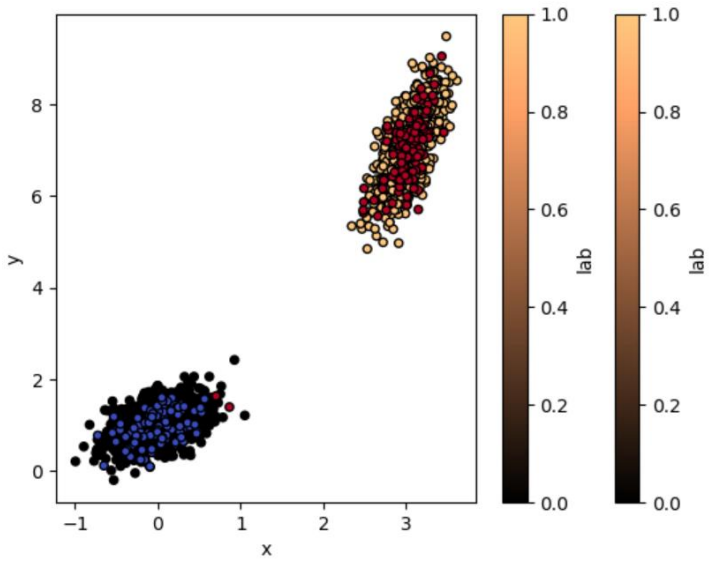
50 iterations:




```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0
error= 0.425
```

Further improvements. Still high error rate. Do not know why the duration time sometime does not show the decimals

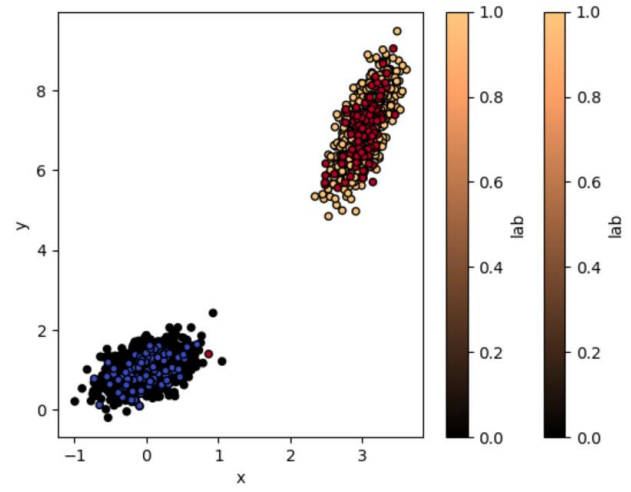
100 iterations:




```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156257153
error= 0.01
```

Great result. The time increase from 10 iterations to 100 iterations is negligible, while the error rate has dropped by an enormous amount. This goes to show that the starting weights were off by a good portion when we do not see significant improvements between 10-50 iterations but massive improvements between 50-100 iterations!

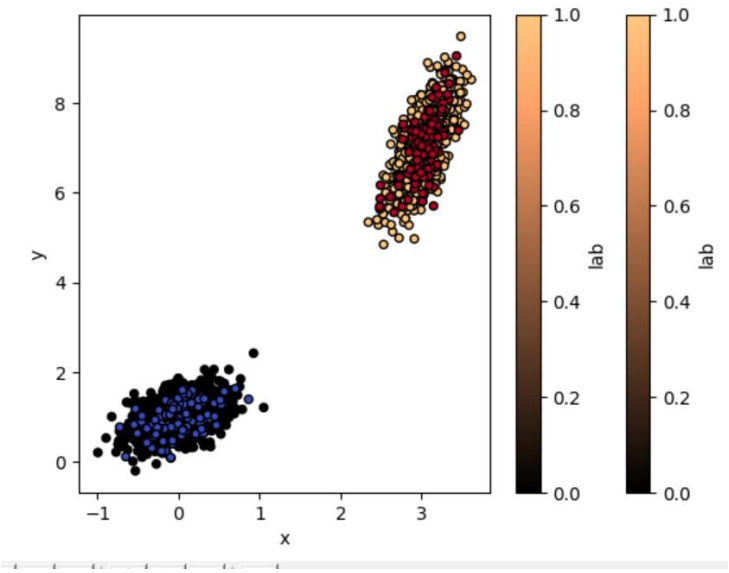
200 iterations:



```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0156297684
error= 0.005
```

Even lower error rate. Time duration pretty much untouched

500 iterations:




```
C:\Users\danie\Desktop\TDT4171\Exercise 4>python.exe skeleton_v2.py
Importing
Import successful
Total duration:  0.0312552452
error= 0.0
```

Now there are “no errors”. In reality the error rate is too small to be expressed with 10 decimals. This is due to the amount of iterations we are executing. This goes to show that the iterations play a crucial part to eliminate errors and converge towards a very good test result. The more iterations we have, the more time do we have to “correct” the initial weightings, so we can adjust them to yield no errors in our test. The duration has doubled but it is still a very respectable completion time (especially compared with the batch algorithm).