ASSIGNMENT 1: PERCEPTRON

Teacher: HOOGENDOORN, Mark

Team members:

- ORTEGA, Julian
- STABLUM, Francesco

GENERATED LINEARLY SEPARABLE DATA

The code for this task is inside the `Assignment_1_Generated_data.m` file.

The data for the first part of the assignment was generated with the `generateData` function, which allows creating a certain number of clusters using a determined amount of random numbers. The function was used to generate 2 clusters using 1000 data points in the following manner:

```
[data cp classes] = generateData(1, 0.5, 2, 15, 15, 5, 1, 2, 1000);
```

The `classes` variable states to which cluster number a data point belongs to, and the numbers start with 1 and rise in increments of one. The assigned clusters will be the data points' classes. We can check this with the tabulate command:

```
tabulate(classes)
```

Value	Count	Percent
1	439	43.90%
2	561	56.10%

Table 1. Distribution of cluster assignment

In our case there are 439 points in ***class 1*** and 561 points in ***class 2***. Now, the classes have to be adjusted to be in line with the assignment. ***Class 1*** will become ***class -1*** and ***class 2*** will become ***class -1***. To do so, the following commands were used:

```
classes(classes == 1) = -1;
classes(classes == 2) = 1;
```

For convenience and repeatability purposes the data we generated and modified was stored into the `generated_data.mat` file and it's loaded through the utility function `importfile.m`

```
importfile('generated_data.mat')
```

The `generated_data` variable is then split into `data` and `classes`:

```
data = generated_data(:,1:2);
```

```
classes = generated_data(:,3);
```

The `data` variable contains the x, y coordinates

```
x = data(:,1);
y = data(:,2);
plot_data(x, y, classes);
```

Afterwards, we can look at the visually look at the data to determine if it's linearly separable. The convenience function `plot_data` was created to ease the plotting.

Figure 1 shows how the data looks in our particular case. The data is clearly linearly separable.

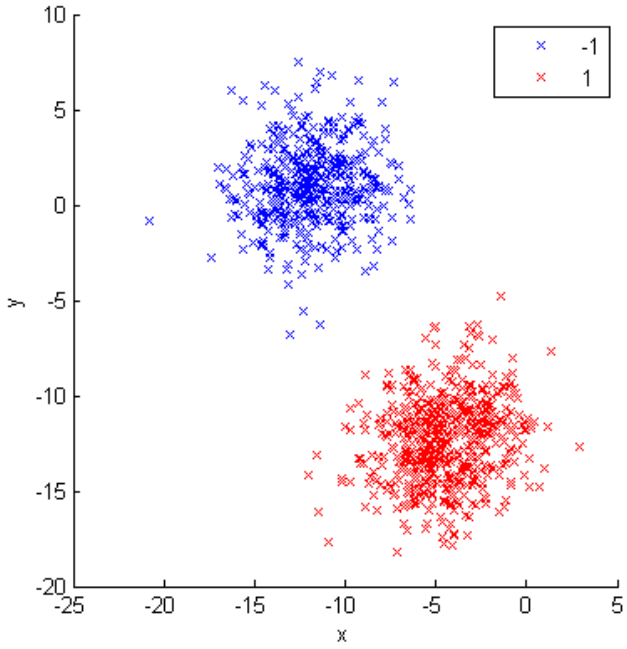

Figure 1. Visual representation of the generated data

The perceptron implementation function definition is as follows:

```
function [weights, stopping_iteration, predicted_classes] = perceptron(data, class,
learning_rate, iterations, plot_error)
```

The parameters are:

- `data` - the data points
- `class` - the assigned class to every data point
- `learning_rate` - the perceptron learning rate
- `iterations` - the maximum numbers of iterations to perform

- `plot_error` (optional) - If set to 1, the function will plot a graph of the prediction error over the iterations. Default is 0.

The output variables are:

- `weigths` are the calculated weights by the perceptron.
- `stopping_iteration` is the iteration at which the code finished. It either iterates until it reached the maximum or until a convergence point is reached
- `predicted_classes` are the predicted class for every input

The perceptron implementation was called as follows:

```
[weights iterations predicted_classes] = perceptron(data, classes, 0.1, 100, 1);
```

After execution has finished we can explore the output variables for information.

```
weights =   -0.3000    2.5769   -3.9192

iterations = 4
```

We can see now the resulting weight values after a convergence in 4 iterations by the perceptron. ***Figure 2*** shows a plot of the prediction error over the iterations.

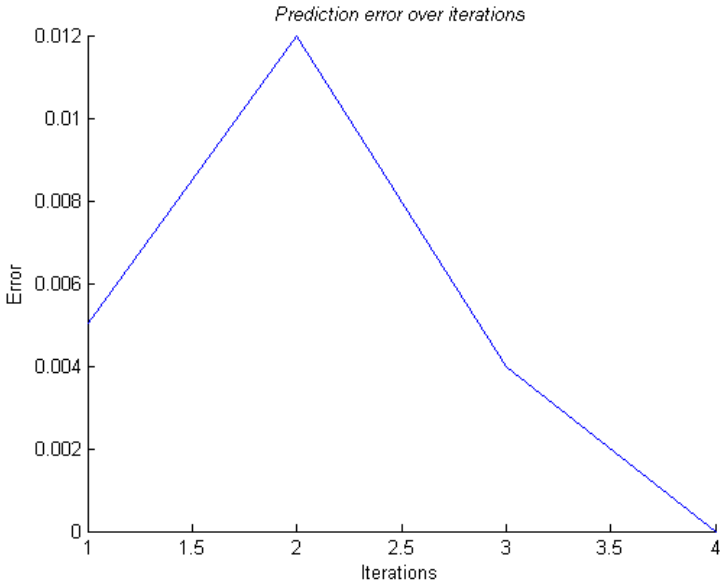

Figure 2. Prediction error over iterations

Tabulating the `predicted_classes` will show a distribution of the classes assigned by the neural network.

```
tabulate(predicted_classes)
```

Value	Count	Percent
-1	439	43.90%
1	561	56.10%

Table 2. Distribution of predicted classes

The tabulation on ***Table 2*** looks very much like the one in ***Table 1***. Additionally, ***Figure 3*** shows the data plotted against the real classes next to the data plotted against the calculated classes

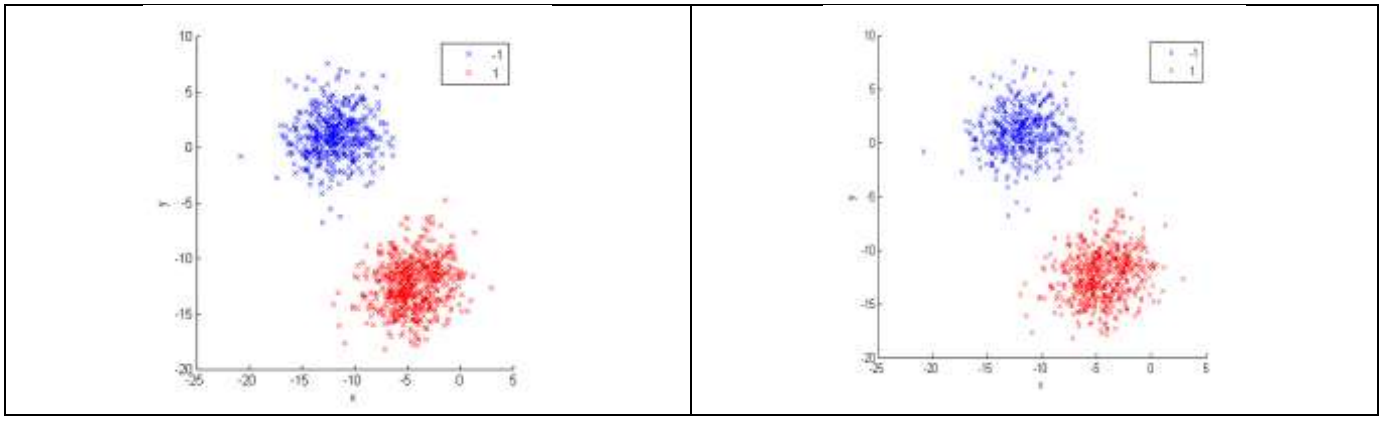

Figure 3. On the left side is the data plotted with the real classes. On the right the data is plotted with the calculated classes by the perceptron

To observe the effect that the learning rate would have over the error of the perceptron, the data was ran over and over through the neural network using different learning rate values:

```
learning_rates = [0.01:0.01:1];
```

The resulting graph in ***Figure 4*** shows that for our data, the perceptron always reaches a solution with zero error.

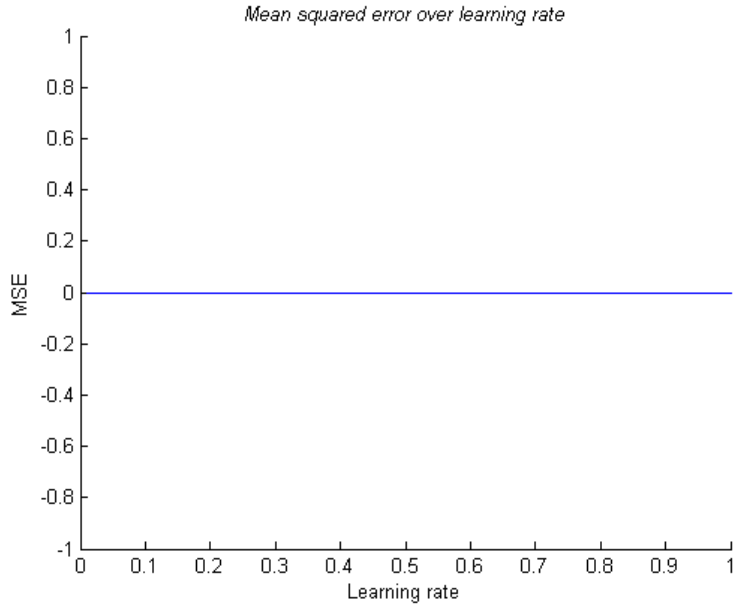

Figure 4. Effect of learning rate on the MSE

Figure 5 shows the decision boundary produced by the perceptron. The plot was generated with the utility function `plot_data_and_decision_boundary, in the following manner:`

```
plot_data_and_decision_boundary(data, classes, weights);
```

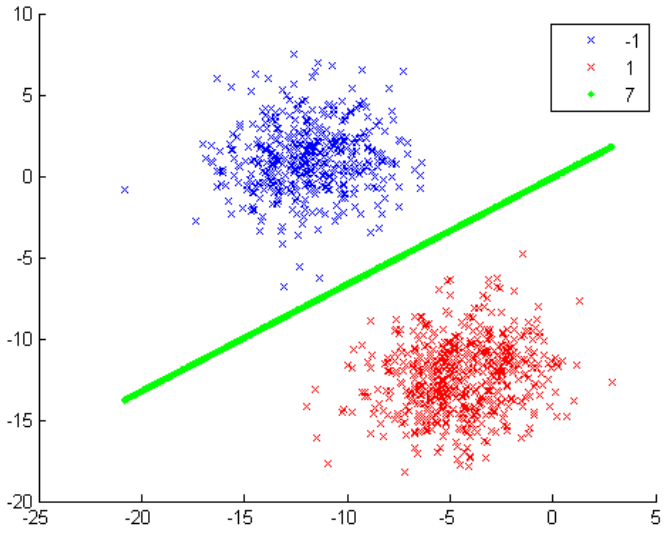

Figure 5. Data plotted with decision boundary

NON-LINEARLY SEPARABLE DATA

The code for this task is inside the `Assignment_1_Non_linearly_separable_data.m` file.

For the second part of the assignment we have a given data set, which has two classes of data that are mixed and are not linearly separable. The expectation is that the perceptron will not be able to fully and accurately find a line that will separate the 2 classes perfectly.

For convenience and repeatability purposes we read the given non-linearly separable data from the `two_class_example_not_separable.dat` file and it's loaded through the utility function `importfile.m`

```
importfile('two_class_example_not_separable.dat')
```

Figure 6 shows how the data looks like. It is also to be noted that the classes of the given data are ***class 0*** and ***class 1***.

```
x = two_class_example_not_separable(:,1);
y = two_class_example_not_separable(:,2);
classes = two_class_example_not_separable(:,3);
plot_data(x,y,classes);
```

The classes will be changed to match our expectation in the algorithm of ***class -1*** and ***class 1***, which means ***class 0*** will now be ***class -1***

```
classes(classes == 0) = -1;
```

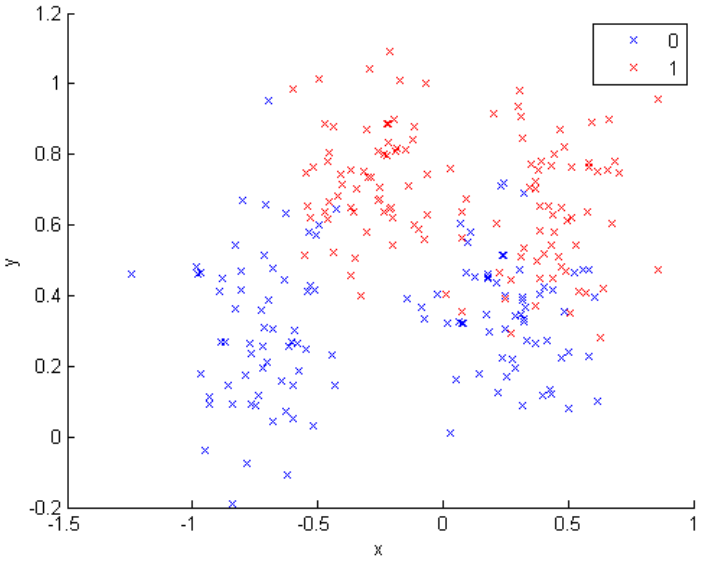

Figure 6. Plot of non-linearly separable data

Tabulating the classes, as seen in ***Table 3***, the data will reveal that there are 250 data points that are distributed equally into 2 classes.

Value	Count	Percent
-1	125	50.00%
1	125	50.00%

Table 3. Distribution of classes of non-linearly separable data

We aggregate from the `x` and `y` variable into `non_sep_data` variable and then perceptron was run in a similar fashion as it was for the generated data.

```
non_sep_data = [x y];
[weights iterations predicted_classes] = perceptron(non_sep_data, classes, 0.1,
100, 1);
```

The resulting information is the output of the perceptron:

```
weights =   -0.3000    0.3097    1.3636

iterations = 100
```

We can see now the resulting weight values did not converge in the established 100 iterations. ***Figure 7*** shows a plot of the prediction error over the iterations.

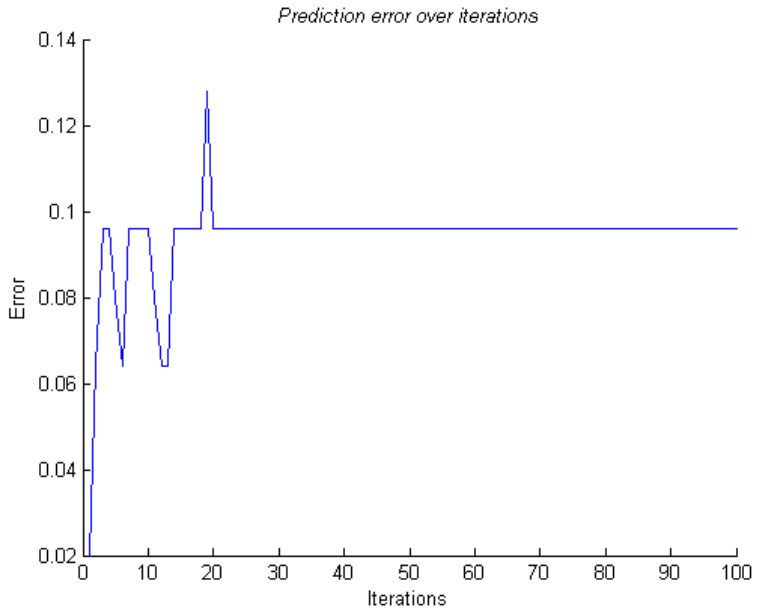

Figure 7. Prediction error over iterations for non-linearly separable data

To observe the effect that the learning rate would have over the error of the perceptron, the data was ran over and over through the neural network using different learning rate values, over 100 iterations:

```
learning_rates = [0.01:0.01:1];
```

The resulting graph in ***Figure 8*** shows that for the learning rate did not have an impact on the resulting error.

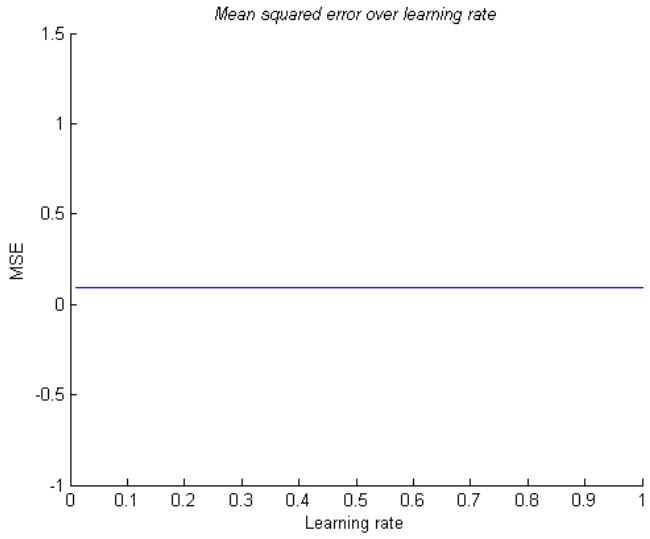

Figure 8. Effect of learning rate on the MSE on the non-linearly separable data

Figure 9 shows the decision boundary produced by the perceptron for the non-linearly separable data. The plot was generated with the utility function `plot_data_and_decision_boundary`, in the following manner:

```
plot_data_and_decision_boundary(data, classes, weights);
```

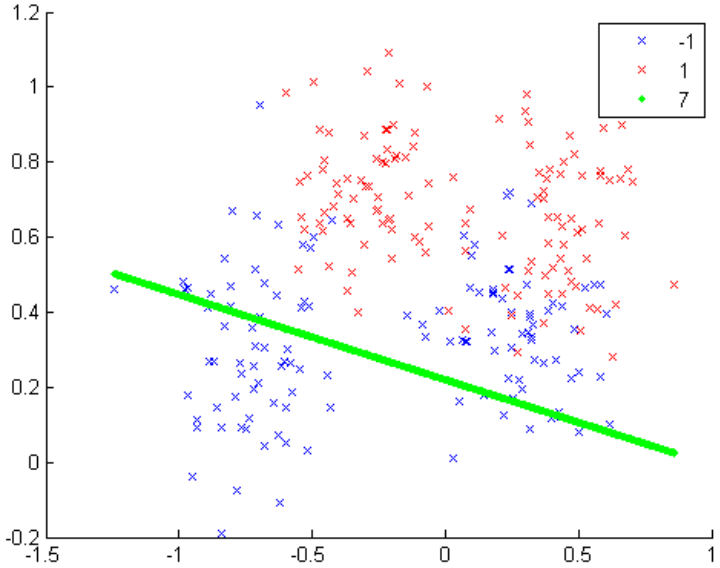

Figure 9. Data plotted with decision boundary

It is clear the perceptron is in fact, as expected, not capable of finding a boundary that perfectly separates both classes.