

Page AB 1:

- יהושע גורון - ת.ז 332307073
- סיון נעמה עזרי - 208479311
- אלעד וינברנד - ת.ז 203306014

Page A2-A3

The details of the Homework: $x, y \leq 100$. The data is all data points where x is of the form $m/100$ where m is an integer between -10000 and $+10000$ and y is of the form $n/100$ with n an integer between -10000 and $+10000$. Suppose that all data points with $y > 1$ have the value 1; all other points have the value -1. **Now suppose you do not know this;** but you are given a random sample of data of size 1000 together with its value (e.g. the point has value 1; while the point has the value -1. (Part A and Part B are due by April 29)

Part A. Implement the Adaline learning algorithm and show how it generalizes to develop a decision that works on all the set. That is, you randomly select a training data set which is a random sample of 1000 data points of all the data points. You use the training data set as input to the Adaline algorithm.

Now randomly select a “test” set of 1000 data points and see how well your trained Adaline performs. Pick a second random test set and see if the results change.

What can you conclude about your results?

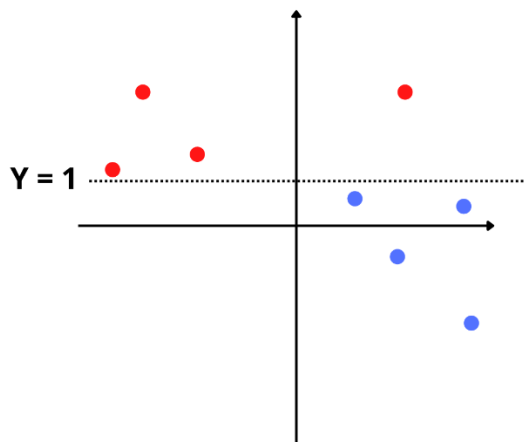
How Does the accuracy of the result depend on the training set?

Present tables and possibly a picture indicating your results. Discuss the impact on the size of training set and choice of test set.

Answer:

It is obvious that from our data set there is a linear separable line that will correctly classify all data points $\langle x, y \rangle$ of our 10,000 points and quite frankly any future $\langle x, y \rangle$.

$Y = 1$



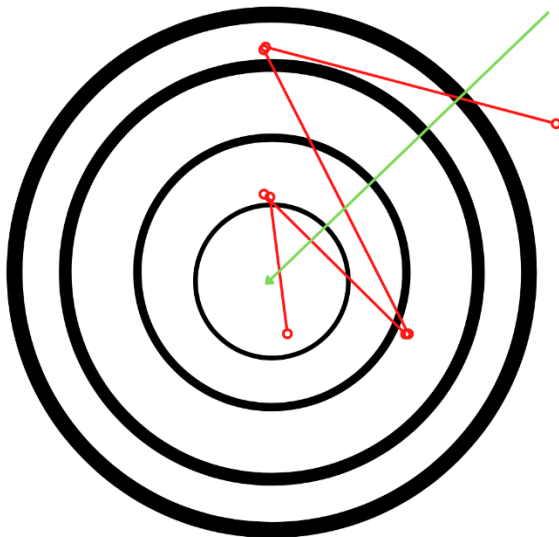
However our Adaline algorithm did not classify all points $\langle x, y \rangle$ correctly.

what we concluded from our results is the following:

the Aldine algorithm is as follows:

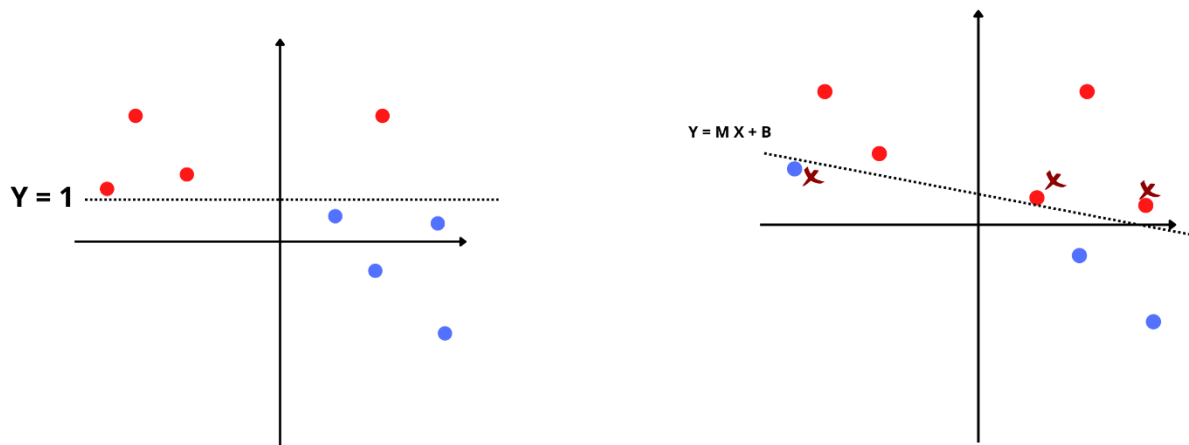
- 1) apply input to Aldine input
- 2) find the square error of current input $\text{errsq}(k) = (d(k) - Wx(k))^2$
- 3) approximate $\text{grad}(\text{error square})$ by differentiating errsq
- 4) approximating average errsq by $\text{errsq}(k)$
- 5) obtain $-2\text{error}(k)x(k)$ also called delta rule
- 6) update W , $W(\text{new}) = W(\text{old}) + 2\text{udelta}(k)$
- 7) repeat steps 1 - 6

we learnt in lesson 3 that using the least mean square (LMS) algorithm will improve our Adaline algorithm in the following manner , see image below:



And will supposedly Lim \rightarrow to the correct linear separable line.

There for it is no surprise that when the line is not a perfect $Y = 1$ and has a bit of an angle that points would be classified incorrectly, see figure below:

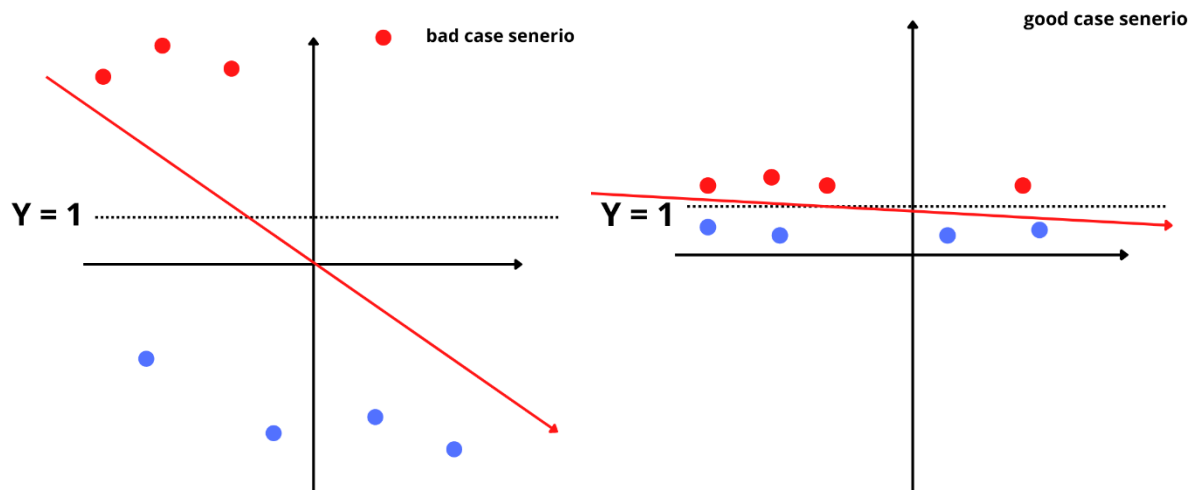


We can conclude that the **Bigger** the training set the more our Adaline algorithm will correctly classify points in the 2D plane as it will more and more resemble the line $Y = 1$ which in our case represents a perfect linear separable line between each class.

the accuracy of the result depends on the training set in the following manner:

(Present tables and possibly a picture indicating your results. Discuss the impact on the size of training set and choice of test set.)

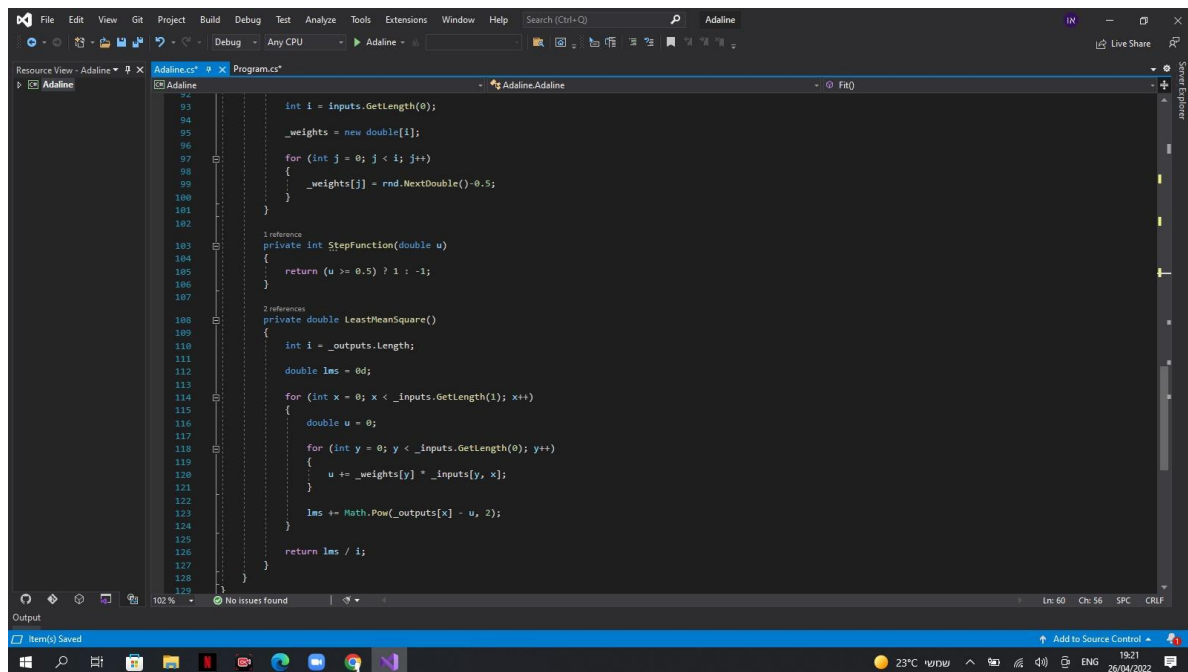
lets take two extreme examples of possible training sets and explain the spectrum of possibility's:



So one side of the spectrum we have possibility of bad case scenario in this example our training set is made up of 1000 points where all the points above $y = 1$ have y values **much higher than 1** and all the points below $y = 1$ have values **much lower than 1**. Once our Adaline model is trained it will be hard for it to differentiate between new examples as our training set was extremely poor quality for our algorithm to train on as seen above.

On the other hand in a good case scenario is the complete opposite, all values with y value greater than $y = 1$ are **extremely close to 1** and values below $y = 1$ are **extremely close to 1** as seen above in the second graph. This is as good as it gets in terms of training sets and will give us much better results for classifying new points in the 2D plane.

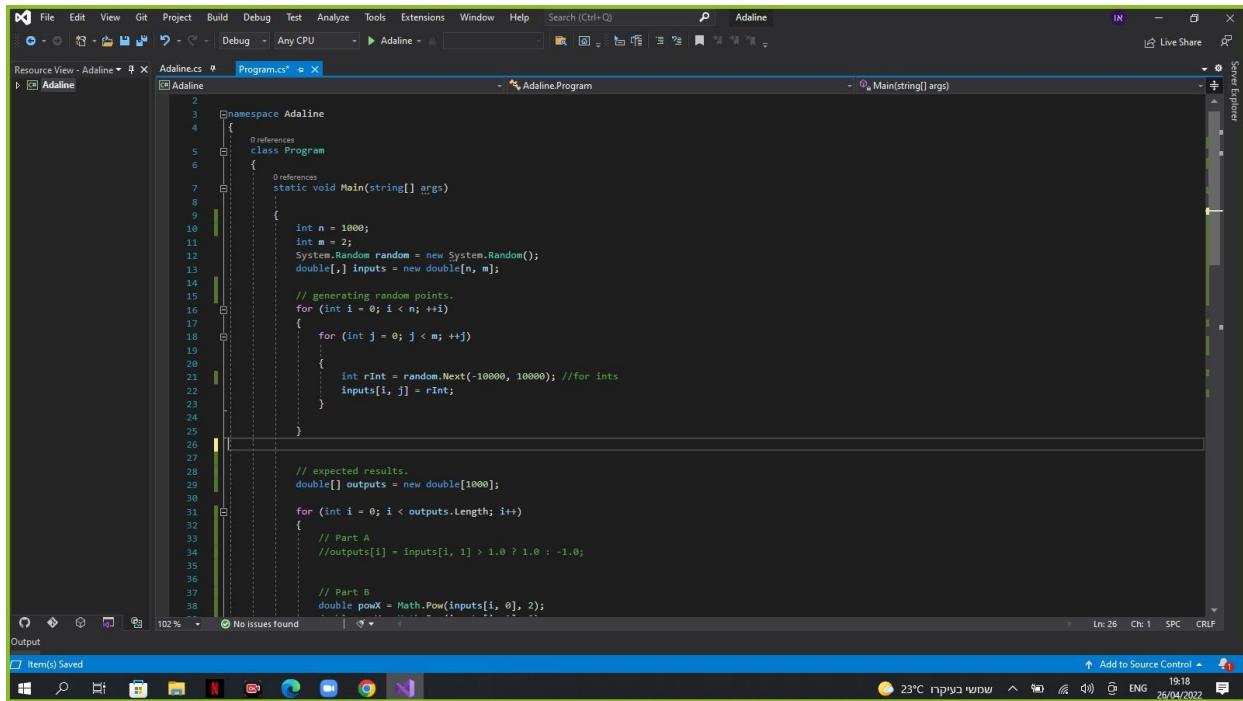
Page A4:



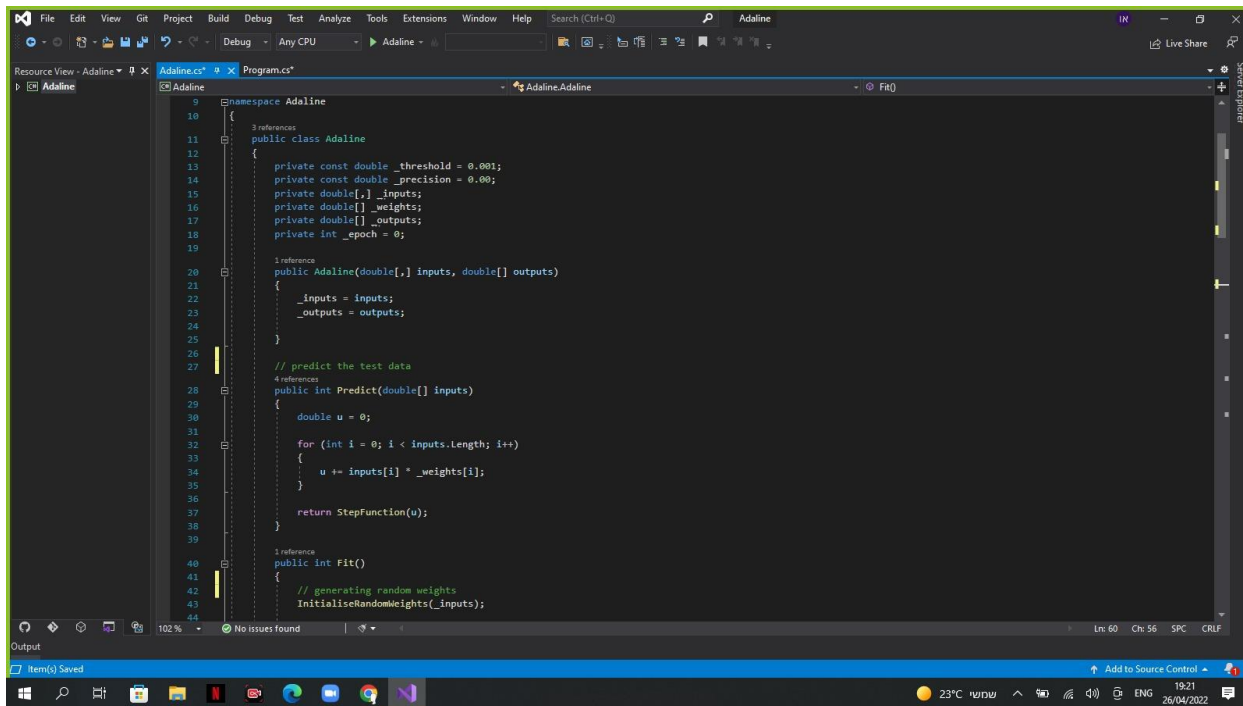
The screenshot shows the Visual Studio IDE with a C# file named `Program.cs` open. The code implements a simple neural network with the following components:

- `int i = inputs.GetLength(0);` and `_weights = new double[i];` to initialize weights.
- A `for` loop to initialize weights: `for (int j = 0; j < i; j++) { _weights[j] = rnd.NextDouble() * 0.5; }`
- A `private int StepFunction(double u)` method that returns `1` if `u >= 0.5`, otherwise `-1`.
- A `private double LeastMeanSquare()` method that calculates the loss. It iterates over inputs and weights to compute the squared error for each input, sums them, and returns the average loss: `return lms / i;`

The bottom status bar indicates "No issues found" and the output window shows "Item(s) Saved". The system tray at the bottom shows the date and time as 26/04/2022, 19:21.



```
1 namespace Adaline
2 {
3     0 references
4     class Program
5     {
6         0 references
7         static void Main(string[] args)
8         {
9             int n = 1000;
10            int m = 2;
11            System.Random random = new System.Random();
12            double[,] inputs = new double[n, m];
13
14            // generating random points.
15            for (int i = 0; i < n; ++i)
16            {
17                for (int j = 0; j < m; ++j)
18                {
19                    int rint = random.Next(-10000, 10000); //for ints
20                    inputs[i, j] = rint;
21                }
22            }
23
24            // expected results.
25            double[] outputs = new double[1000];
26
27            for (int i = 0; i < outputs.Length; i++)
28            {
29                // Part A
30                //outputs[i] = inputs[i, 1] > 1.0 ? 1.0 : -1.0;
31
32                // Part B
33                double powX = Math.Pow(inputs[i, 0], 2);
34            }
35
36            Output.WriteLine("Part A:");
37            for (int i = 0; i < outputs.Length; i++)
38            {
39                Console.WriteLine(outputs[i]);
40            }
41
42            Output.WriteLine("Part B:");
43            for (int i = 0; i < outputs.Length; i++)
44            {
45                Console.WriteLine(powX);
46            }
47        }
48    }
49 }
```



```
1 namespace Adaline
2 {
3     3 references
4     public class Adaline
5     {
6         private const double _threshold = 0.001;
7         private const double _precision = 0.001;
8         private double[,] _inputs;
9         private double[,] _weights;
10        private double[,] _outputs;
11        private int _epoch = 0;
12
13        1 reference
14        public Adaline(double[,] inputs, double[] outputs)
15        {
16            _inputs = inputs;
17            _outputs = outputs;
18        }
19
20        // predict the test data
21        4 references
22        public int Predict(double[] inputs)
23        {
24            double u = 0;
25
26            for (int i = 0; i < inputs.Length; i++)
27            {
28                u += inputs[i] * _weights[i];
29            }
30
31            return StepFunction(u);
32        }
33
34        1 reference
35        public int Fit()
36        {
37            // generating random weights
38            InitialiseRandomWeights(_inputs);
39        }
40    }
41 }
```


Page B2:

Part B: Now change the problem so that points such that has value 1 only if $4 \leq x^2 + y^2 \leq 9$ What are the best results you obtain using an Adaline? Does the quality of the results change if you use more data? Present tables and perhaps a figure

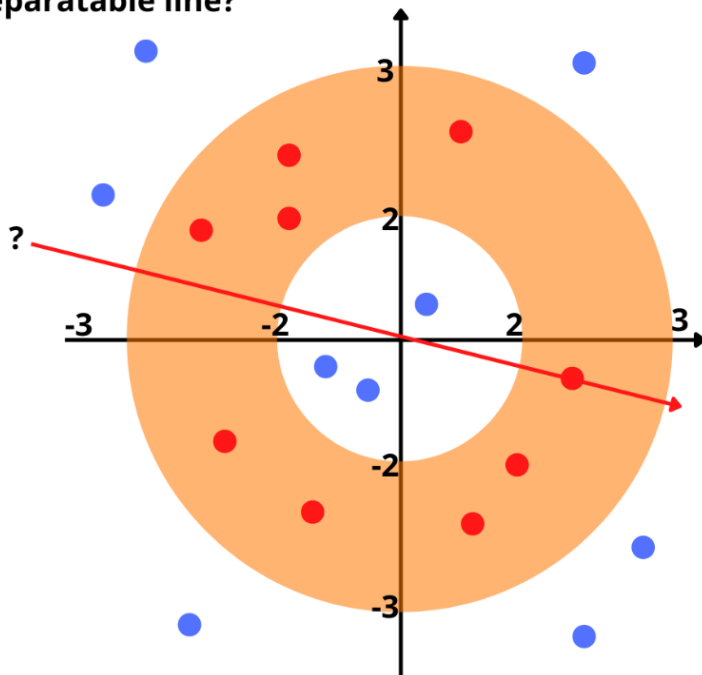
Answer:

What are the best results you obtain using an Adaline?

We observed that the algorithm updated weights much more often than the previous one.

From analyzing the $\langle x, y \rangle$ coordinates from the graph bellow we concluded that it would extremely hard and in fact near impossible to obtain good results using Adaline on the 2D plane $\langle x, y \rangle$. The reason being that there is no liner separable line that would differentiate between the two classes. As seen in the image below:

what is best linear separable line?



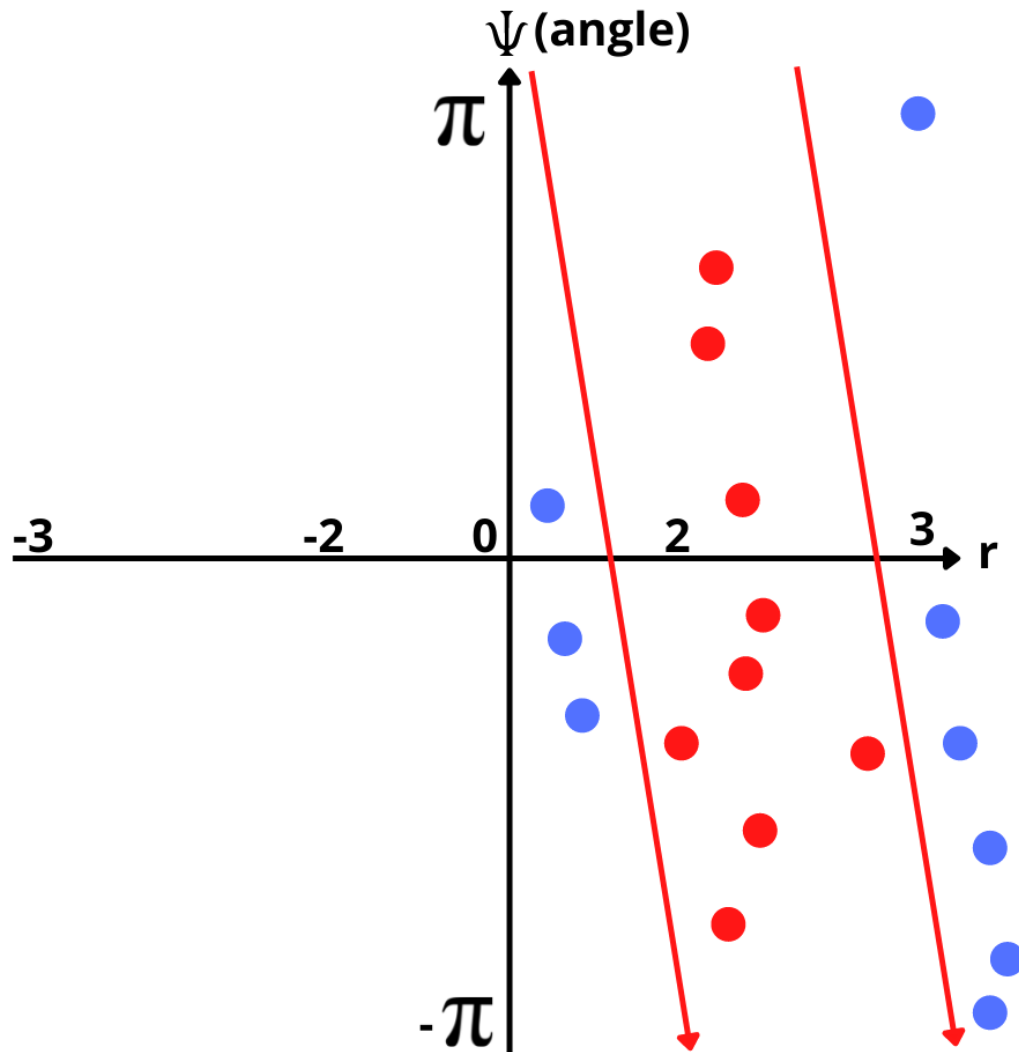
Does the quality of the results change if you use more data?

We concluded that the quality of the results will not change when using more data.

We came up with the following solution know as: **polar coordinates**.

Instead of our points sitting on the $\langle x, y \rangle$ 2D plane we shall “convert” each point in to a $\langle r, \theta \rangle$ point on a new 2D plane where r is the radius from the center of the circle and θ is the angle from the center of the circle in terms of π .

By doing the following we will receive the following data points. (as seen in image below)



Here we still can not differentiate between the two classes with one liner separatable line. In fact the image above reminded us of a **XOR gate**. We did conclude though that by dividing the problem: $4 \leq x^2 + y^2 \leq 9$

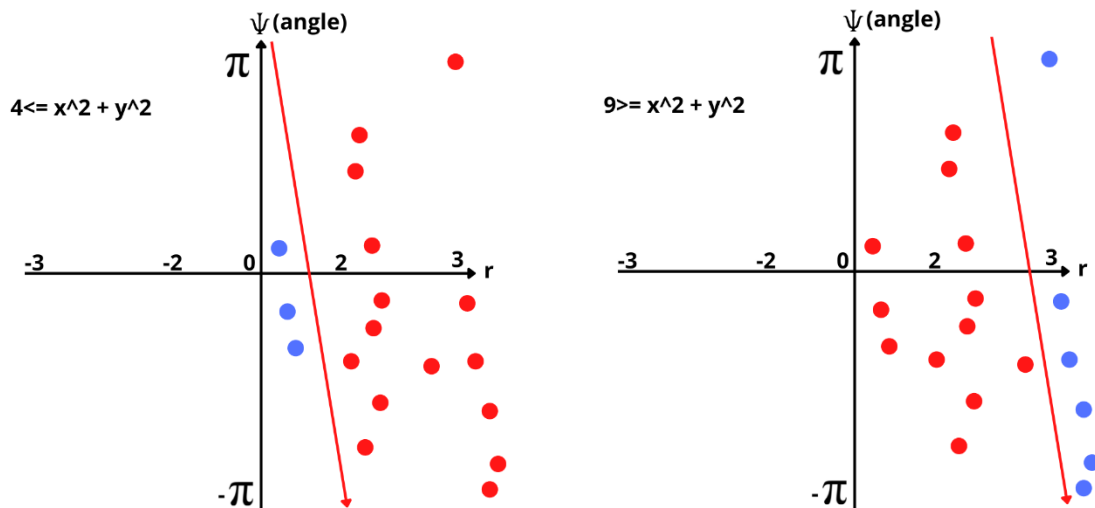
In to two problems:

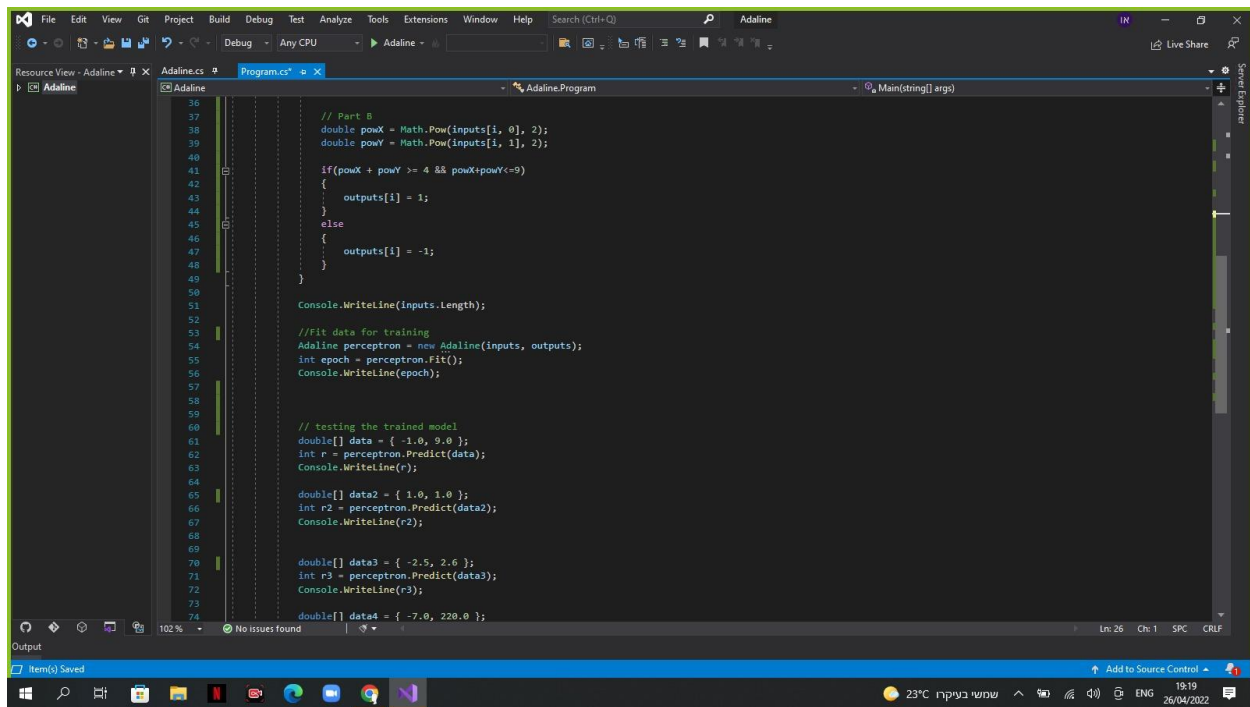
$$4 \leq x^2 + y^2$$

And

$$9 \geq x^2 + y^2$$

That there would be a linear separable line for each case and we would receive better results as shown below:

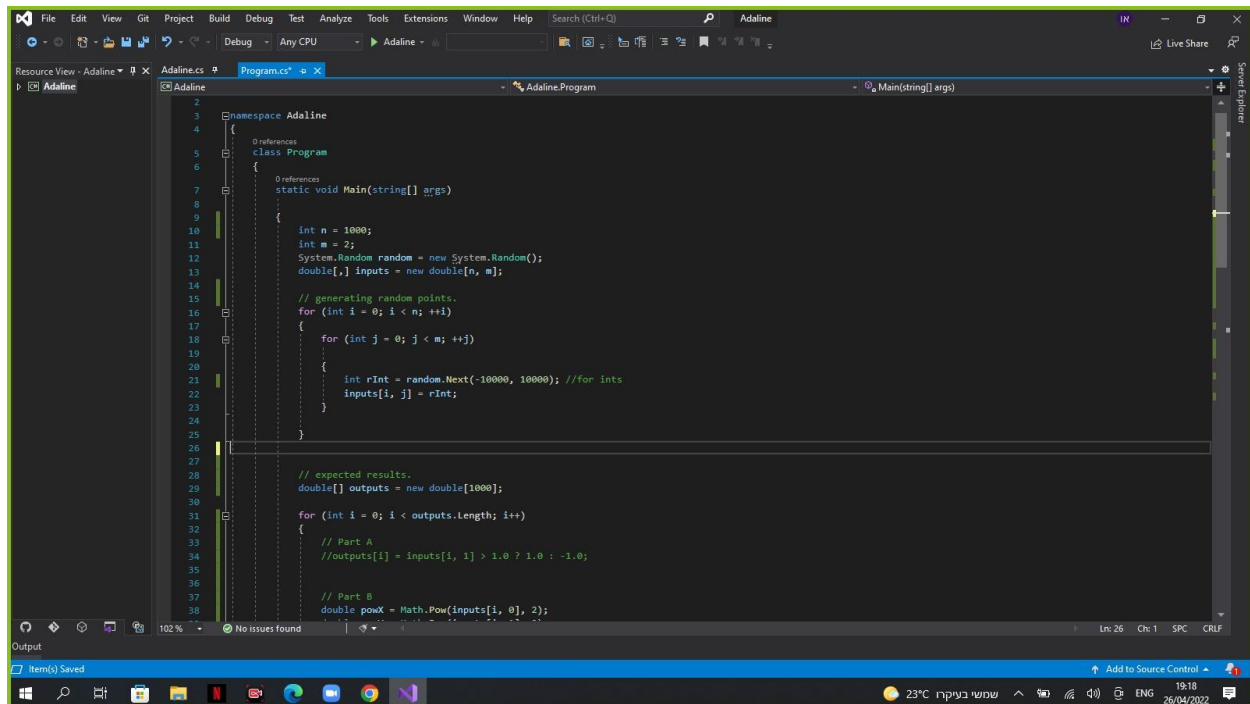




This screenshot shows the Visual Studio IDE with the `Adaline.cs` file open. The code implements a perceptron model with the following logic:

- Part B:** Calculates $\text{powX} = \text{Math.Pow}(\text{inputs}[1], 2)$ and $\text{powY} = \text{Math.Pow}(\text{inputs}[1], 2)$. It then checks if $\text{powX} + \text{powY} \geq 4$ and $\text{powX} + \text{powY} \leq 9$. If true, `outputs[i] = 1`; otherwise, `outputs[i] = -1`.
- Training:** Uses `Adaline perceptron = new Adaline(inputs, outputs);` and `int epoch = perceptron.Fit();` to train the model.
- Testing:** Tests the model with three data sets:
 - `data1 = { -1.0, 9.0 }` → `int r1 = perceptron.Predict(data1);`
 - `data2 = { 1.0, 1.0 }` → `int r2 = perceptron.Predict(data2);`
 - `data3 = { -2.5, 2.6 }` → `int r3 = perceptron.Predict(data3);`

The bottom status bar shows "102%", "No issues found", and the system tray includes a temperature of 23°C and the date 26/04/2022.



This screenshot shows the Visual Studio IDE with the `Adaline.cs` file open, displaying the `namespace Adaline` and `class Program` structure. The code includes the following sections:

- Imports:** `using System;`
- Main Method:** `static void Main(string[] args)`
- Initialization:** `int n = 1000;`, `int m = 2;`, `System.Random random = new System.Random();`, and `double[,] inputs = new double[n, m];`
- Generating random points:** A nested loop for `i` (0 to n) and `j` (0 to m) that assigns random values to `inputs[i, j]` using `random.Next(-10000, 10000)`.
- Expected results:** `double[,] outputs = new double[1000];`
- Part A:** `//outputs[i] = inputs[i, 1] > 1.0 ? 1.0 : -1.0;`
- Part B:** `double powX = Math.Pow(inputs[i, 0], 2);`

The bottom status bar shows "102%", "No issues found", and the system tray includes a temperature of 23°C and the date 26/04/2022.