Imię Nazwisko Grupa Michał Słowikowski Gr 4	Temat Scenariusz 4	Data 15.12.2017r.

Celem ćwiczenia było napisanie zapoznanie się z działaniem sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowania istotnych cech kwiatów.

Syntetyczny opis algorytmu uczenia

Do wykonania ćwiczenia zbudowałem SOM (self organizing map) z użyciem algorymtmu WTA. Sieć składała się ze zmiennej ilości neuronów, które początkowo przybierały losowe wagi. Przygotowałem również zestaw danych wejściowych, na który składało się 150 rekordów i 3 gatunków kwiatów. Do uczenia wykorzystałem 105 (po 35 z każdego gatunku) natomiast do testowania wyników 45 (po 15 z każdego gatunku). Mapa składała się z siatki o różnej ilości neuronów, jednak do przedstawiania danych zawsze wypisywana była jako kwadrat.

Sieć Kohonena pomaga reprezentować wielowymiarowe dane w przestrzeni o mniejszym wymiarze. Algorytm polega na tym, że najpierw z zestawu danych losujemy losowy rekord. Następnie szukamy neuronu, który znajduje się “najbliżej” jego, będzie on tzw. “Zwycięzcą” . Obliczanie odległości odbywało się wg. Wzoru:

$$Dist = \sqrt{\sum_{i=0}^{i=n} (V_i - W_i)^2}$$

Equation 1

Następnie należało wyznaczyć nowe wagi tego neuronu wg wzoru.

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \eta_i(k)[\mathbf{x} - \mathbf{w}_i(k)]$$

Oprócz zmiany ilości neuronów w sieci, zmiany współczynnika uczenia oraz spotkałem się z modyfikacją polegającą na zmniejszaniu w każdej epoce współczynnika uczenia. Wykonałem testy z i bez tej modyfikacji (z współczynnik uczenia dzieliłem przez 2).

Dane do nauki

Dane do nauki znalazłem na stronie:
https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

Przykładowe dane:

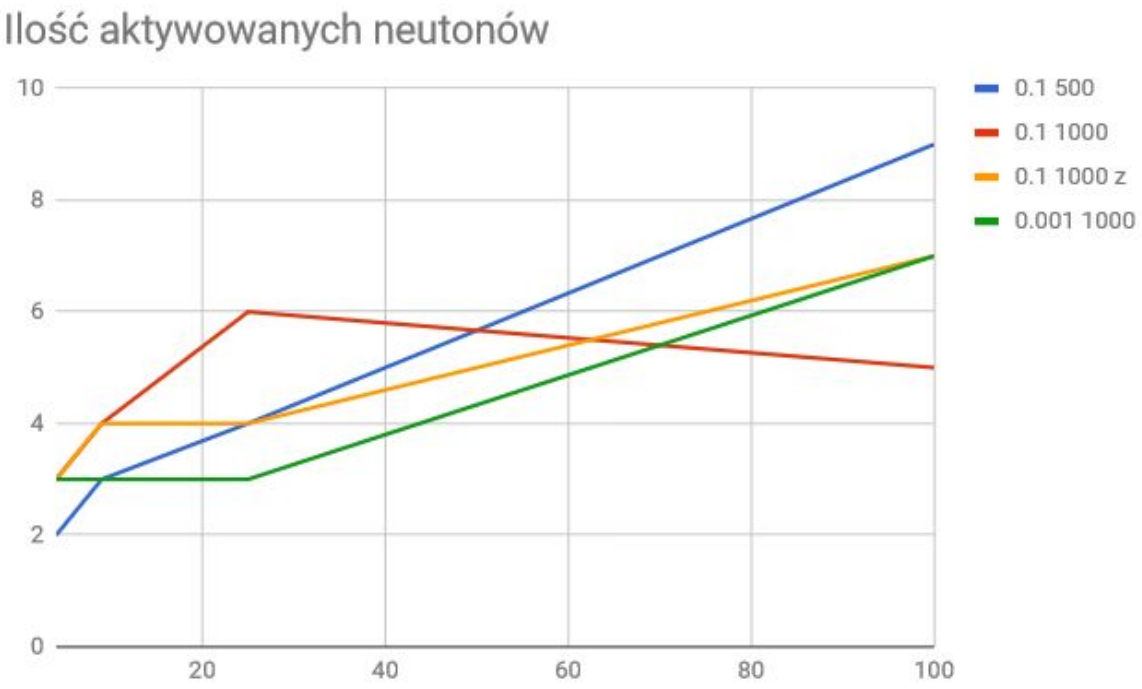
4.4,3.2,1.3,0.2,Iris-setosa
5.0,3.5,1.6,0.6,Iris-setosa
5.1,3.8,1.9,0.4,Iris-setosa
6.2,2.9,4.3,1.3,Iris-versicolor
5.1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica

Dane zostały znormalizowane

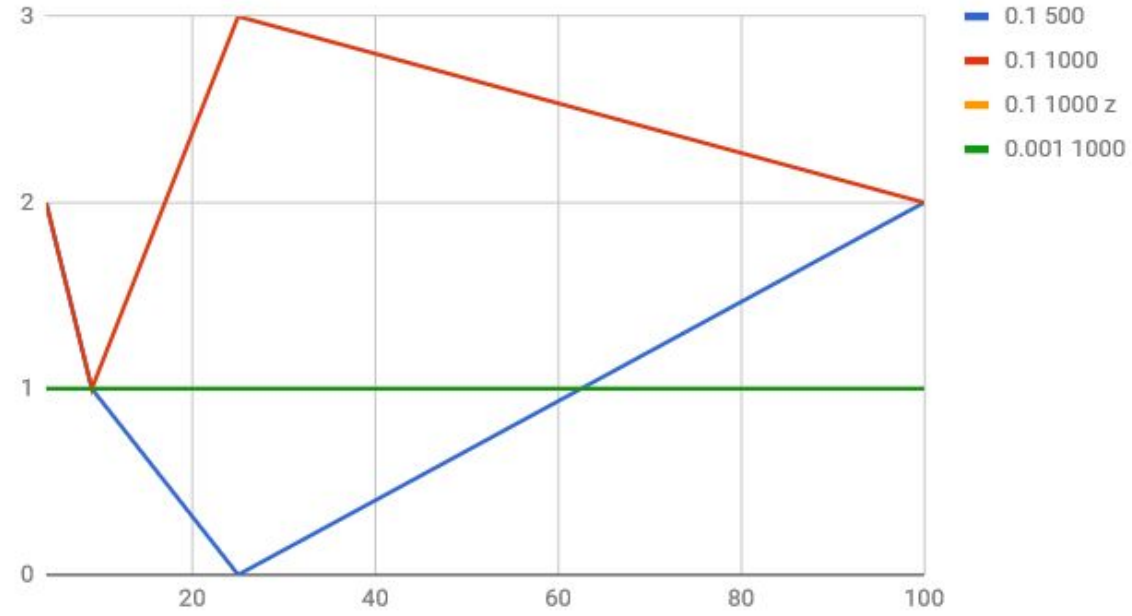
Przykładowy output:

Lr = 0.1 Liczba neuronów = 9, Ilość epok uczenia = 500
A-Iris-setosa
B-Iris-versicolor
C-Iris-virginica

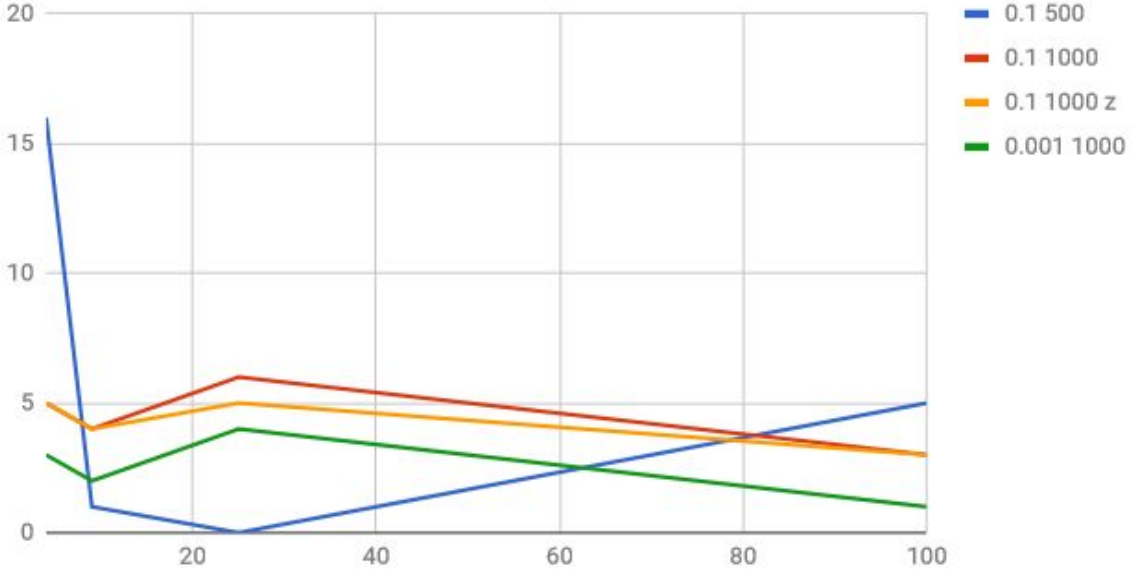
Wyniki Testów



Ilość neuronów zawierających 2 gatunki



Ilość kwiatów które różnych gatunków, które aktywowały jeden neuron



Analiza wyników

Wyniki wskazują na to, że wielkość siatki ma bardzo duży wpływ na “rozdzielczość wyników”. Im większa siatka tym więcej neuronów było aktywowanych, jednak można zauważyć, że tworzą się neurony skupiska, w których znajduje się większość kwiatów danego gatunku, oraz neurony przejściowe, mniejsze i zawierające czasami po 2 różne garunki.

Wpływy ustawień sieci na naukę:

- Mniejszy współczynnik uczenia lepiej sprawdził się od większego. Miał on również największy wpływ na jakość nauki (kolor zielony na wykresie) oraz powodował najmniejszą ilość neuronów przejściowych.
- Sposób korygowania wartości współczynnika uczenia w kolejnych epokach nie miał aż tak dużego wpływu. Jednak można zaobserwować, że powodował poprawę uczenia.
- Większa ilość cykli nauczania pozwalała osiągnąć lepsze wyniki dla większych siatek. Mniejsze zaś dobrze radzą sobie w małych siatkach

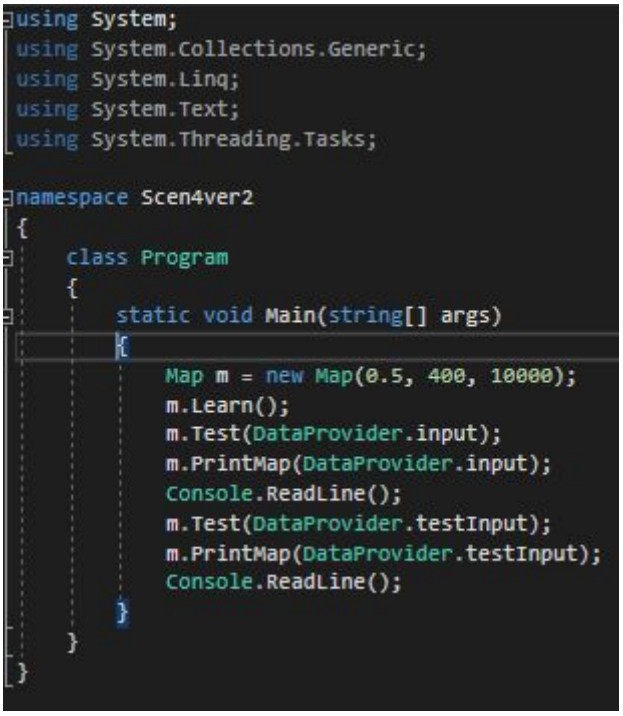
Na podstawie wyników można również stwierdzić, że nie bez znaczenia były początkowe wagi.

Porównując dane ze względu na ilość cykli uczenia łatwo zauważyć również, że dla dużych siatek o ile ilość aktywowanych neuronów była większa to ilość kwiatów z różnych gatunków, które aktywowały ten sam neuron była mniejsza.

Wnioski

- Większe sieci uzyskiwały lepsze wyniki
- Zmiana współczynnika uczenia podczas nauki zwiększała jakość uczenia, jednak w niewielkim stopniu
- Ilość aktywowanych neuronów nie świadczy o jakości uczenia
- Jeżeli 2 gatunki aktywowały ten sam neuron oznacza to, że gatunki są bardzo do siebie podobne (a przynajmniej dane wejściowe na to wskazują), mamy taką sytuację z gatunkiem B i C
- Mapa sprowadziła wielowymiarowy problem do mapy 2d
- Ważne dla jakości nauczania było losowe dobieranie danych wejściowych z setu

Kod



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scen4ver2
{
    class Program
    {
        static void Main(string[] args)
        {
            Map m = new Map(0.5, 400, 10000);
            m.Learn();
            m.Test(DataProvider.input);
            m.PrintMap(DataProvider.input);
            Console.ReadLine();
            m.Test(DataProvider.testInput);
            m.PrintMap(DataProvider.testInput);
            Console.ReadLine();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scen4ver2
{
    class Neuron
    {
        double[] weights;
        public Neuron(double[] weights)
        {
            this.weights = new double[weights.Length];
            Array.Copy(weights, this.weights, weights.Length);
        }

        public double CalculateEuclideanDistance(double[] input)
        {
            double sum = 0.0;
            for (int i = 0; i < input.Length; i++)
            {
                sum += Math.Pow((input[i] - weights[i]), 2);
            }
            sum += Math.Pow((1 - weights[input.Length]), 2);
            return Math.Sqrt(sum);
        }

        public void PrintWeights()
        {
            foreach (double w in weights)
            {
                Console.WriteLine(w);
            }
        }

        public void CalculateNewWeights(double[] input, double lr)
        {
            for (int i = 0; i < input.Length; i++)
            {
                weights[i] += lr * (input[i] - weights[i]);
            }

            weights[input.Length] += lr * (1 - weights[input.Length]);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scen4ver1
{
    class Map
    {
        private Neuron[] neuronMap;
        private int numberOfNeurons;
        private double lr;
        private double Max;
        public Map(double learningRate, int numberOfNeurons, int iter)      //tworzy tablicę neuronów i losuje początkowe wagi
        {
            this.Max = iter;
            this.numberOfNeurons = numberOfNeurons;
            lr = learningRate;
            neuronMap = new Neuron[this.numberOfNeurons];
            Random r = new Random();
            double[] weights = new double[DataProvider.input.GetLength(1) + 1];
            for (int j = 0; j < neuronMap.Length; j++)
            {
                for (int i = 0; i < weights.Length; i++)
                {
                    weights[i] = r.NextDouble();
                }

                neuronMap[j] = new Neuron(weights);
            }
        }

        public void PrintMap()
        {
            for (int i = 0; i < neuronMap.Length; i++)
            {
                Console.WriteLine("Neuron : " + i);
                neuronMap[i].PrintWeights();
            }
        }

        public void Learn()
        {

            double lenght;
            double min;
            int counter = 0;
            int neuronNumber = 0;
            do
            {
                DataProvider.ShuffleInputData();                                //miesza dane wejściowe
                for (int i = 0; i < DataProvider.input.GetLength(0); i++)       //dla każdego rekordu
                {
                    min = 0.0;
                    for (int j = 0; j < neuronMap.Length; j++)                  //szuka najbliższego neuronu
                    {
                        double[] input = DataProvider.GetInput(i, DataProvider.input);
                        lenght = neuronMap[j].CalculateEuclideanDistance(input);
                        if (lenght < min || j == 0)
                        {
                            min = lenght;
                            neuronNumber = j;
                        }
                        neuronMap[neuronNumber].CalculateNewWeights(input, lr); //oblicza nowe wagi dla zwycięzcy
                    }
                }
                // CalculateLearningRate();
                counter++;
            } while (Max > counter);
            DataProvider.RestoreInputToCorrectOrder();
        }
```

```
public void Test(double[,] inputArray)                          //testowanie, wypisuje który neuron ile razy został aktywowany
{
    int neuronNumber;
    int[] responseCounter = new int[neuronMap.Length];
    responseCounter.Select(i => 0).ToArray();
    for (int i = 0; i < inputArray.GetLength(0); i++)
    {
        double[] input = DataProvider.GetInput(i, inputArray);
        neuronNumber = ClassifyInput(input);
        responseCounter[neuronNumber]++;
    }
    int n;
    for (int i = 0; i < responseCounter.Length; i++)
    {
        n = responseCounter[i];
        if (n != 0)
            Console.WriteLine("Neuron: " + i + " Count: " + n);
    }
}

public void PrintMap(double[,] inputArray)
{
    int size = inputArray.GetLength(0) / 3;
    int mapS = (int)Math.Sqrt(numberOfNeurons);
    int beg = 0;
    int end = size;
    String[] map = new String[numberOfNeurons];
    for (int i = 0; i < map.Length; i++)
    {
        map[i] = "";
    }

    map = CalculateMap(map, beg, end, "A", inputArray);
    beg += size;
    end += size;
    map = CalculateMap(map, beg, end, "B", inputArray);
    beg += size;
    end += size;
    map = CalculateMap(map, beg, end, "C", inputArray);

    for (int i = 0; i < mapS; i++)
    {
        for (int j = 0; j < mapS; j++)
        {
            int k = i * mapS + j;
            if (map[k] == "")
            {
                Console.Write(".\t\t");
            }
            else
            {
                Console.Write(map[k] + "\t\t");
            }
        }
        Console.WriteLine();
    }
}

private String[] CalculateMap(String[] map, int beg, int end, String type, double[,] inputArray) //metoda do jakie dane aktrywuje neuron
{
    int nn;
    for (int i = beg; i < end; i++)
    {
        double[] input = DataProvider.GetInput(i, inputArray);
        nn = ClassifyInput(input);
        map[nn] += type;
    }
    return map;
}

private void CalculateLearningRate()    //oblicza nowy lr
{
    lr /= 2;
}
```

```
    private int ClassifyInput(double[] input)       //który neuron jest aktywowany dla danego inputu
    {
        int classification = 0;
        double lenght;
        double min = 0.0;
        for (int j = 0; j < neuronMap.Length; j++)
        {
            lenght = neuronMap[j].CalculateEuclideanDistance(input);
            if (lenght < min || j == 0)
            {
                min = lenght;
                classification = j;
            }
        }
        return classification;
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scen4ver1
{
    class DataProvider
    {
        //   public static int iSize = input.GetLength(1);

        public static String[] type = new String[] { "Iris-setosa", "Iris-versicolor", "Iris-virginica" };
        public static double[,] input = new double[,] {
            {0.6375,0.4375,0.175,0.025},
             {0.6125,0.375,0.175,0.025},
             {0.5875,0.4,0.1625,0.025},
             {0.575,0.3875,0.1875,0.025},
             {0.625,0.45,0.175,0.025},
             {0.675,0.4875,0.2125,0.05},
             {0.575,0.425,0.175,0.0375},
             {0.625,0.425,0.1875,0.025},
             {0.55,0.3625,0.175,0.025},
             {0.6125,0.3875,0.1875,0.0125},
             {0.675,0.4625,0.1875,0.025},
             {0.6,0.425,0.2,0.025},
             {0.6,0.375,0.175,0.0125},
             {0.5375,0.375,0.1375,0.0125},
             {0.725,0.5,0.15,0.025},
             {0.7125,0.55,0.1875,0.05},
             {0.675,0.4875,0.1625,0.05},
             {0.6375,0.4375,0.175,0.0375},
             {0.7125,0.475,0.2125,0.0375},
             {0.6375,0.475,0.1875,0.0375},
             {0.675,0.425,0.2125,0.025},
             {0.6375,0.4625,0.1875,0.05},
             {0.575,0.45,0.125,0.025},
             {0.6375,0.4125,0.2125,0.0625},
             {0.6,0.425,0.2375,0.025},
             {0.625,0.375,0.2,0.025},
             {0.625,0.425,0.2,0.05},
             {0.65,0.4375,0.1875,0.025},
             {0.65,0.425,0.175,0.025},
             {0.5875,0.4,0.2,0.025},
             {0.6,0.3875,0.2,0.025},
             {0.675,0.425,0.1875,0.05},
             {0.65,0.5125,0.1875,0.0125},
             {0.6875,0.525,0.175,0.025},
             {0.6125,0.3875,0.1875,0.0125},
             {0.875,0.4,0.5875,0.175},
             {0.8,0.4,0.5625,0.1875},
             {0.8625,0.3875,0.6125,0.1875},
             {0.6875,0.2875,0.5,0.1625},
             {0.8125,0.35,0.575,0.1875},
             {0.7125,0.35,0.5625,0.1625},
             {0.7875,0.4125,0.5875,0.2},
             {0.6125,0.3,0.4125,0.125},
             {0.825,0.3625,0.575,0.1625},
             {0.65,0.3375,0.4875,0.175},
             {0.625,0.25,0.4375,0.125},
             {0.7375,0.375,0.525,0.1875},
             {0.75,0.275,0.5,0.125},
             {0.7625,0.3625,0.5875,0.175},
             {0.7,0.3625,0.45,0.1625},
             {0.8375,0.3875,0.55,0.175},
             {0.7,0.375,0.5625,0.1875},
             {0.725,0.3375,0.5125,0.125},
             {0.775,0.275,0.5625,0.1875},
             {0.7,0.3125,0.4875,0.1375},
             {0.7375,0.4,0.6,0.225},
             {0.7625,0.35,0.5,0.1625},
             {0.7875,0.3125,0.6125,0.1875},
             {0.7625,0.35,0.5875,0.15},
             {0.8,0.3625,0.5375,0.1625},
             {0.825,0.375,0.55,0.175},
             {0.85,0.35,0.6,0.175},
             {0.8375,0.375,0.625,0.2125},
```

```
public static double[,] testInput = new double[,] {
    {0.625,0.4,0.15,0.025},
    {0.6875,0.4375,0.1625,0.025},
    {0.6125,0.3875,0.1875,0.0125},
    {0.55,0.375,0.1625,0.025},
    {0.6375,0.425,0.1875,0.025},
    {0.625,0.4375,0.1625,0.0375},
    {0.5625,0.2875,0.1625,0.0375},
    {0.55,0.4,0.1625,0.025},
    {0.625,0.4375,0.2,0.075},
    {0.6375,0.475,0.2375,0.05},
    {0.6,0.375,0.175,0.0375},
    {0.6375,0.475,0.2,0.025},
    {0.575,0.4,0.175,0.025},
    {0.6625,0.4625,0.1875,0.025},
    {0.625,0.4125,0.175,0.025},
    {0.75,0.425,0.5625,0.2},
    {0.8375,0.3875,0.5875,0.1875},
    {0.7875,0.2875,0.55,0.1625},
    {0.7,0.375,0.5125,0.1625},
    {0.6875,0.3125,0.5,0.1625},
    {0.6875,0.325,0.55,0.15},
    {0.7625,0.375,0.575,0.175},
    {0.725,0.325,0.5,0.15},
    {0.625,0.2875,0.4125,0.125},
    {0.7,0.3375,0.525,0.1625},
    {0.7125,0.375,0.525,0.15},
    {0.7125,0.3625,0.525,0.1625},
    {0.775,0.3625,0.5375,0.1625},
    {0.6375,0.3125,0.375,0.1375},
    {0.7125,0.35,0.5125,0.1625},
    {0.9625,0.375,0.7625,0.2875},
    {0.7875,0.425,0.7,0.3},
    {0.8,0.3875,0.6875,0.225},
    {0.75,0.375,0.6,0.225},
    {0.8625,0.3875,0.675,0.2625},
    {0.8375,0.3875,0.7,0.3},
    {0.8625,0.3875,0.6375,0.2875},
    {0.725,0.3375,0.6375,0.2375},
    {0.85,0.4,0.7375,0.2875},
    {0.8375,0.4125,0.7125,0.3125},
    {0.8375,0.375,0.65,0.2875},
    {0.7875,0.3125,0.625,0.2375},
    {0.8125,0.375,0.65,0.25},
    {0.775,0.425,0.675,0.2875},
    {0.7375,0.375,0.6375,0.225}
};

public static double[] GetInput(int number, double[,] data)
{
    double[] result = new double[input.GetLength(1)];

    for (int i = 0; i < result.Length; i++)
    {
        result[i] = data[number, i];
    }
    return result;
}
```

```
public static double[] GetInput(int number, double[,] data)
{
    double[] result = new double[input.GetLength(1)];

    for (int i = 0; i < result.Length; i++)
    {
        result[i] = data[number, i];
    }
    return result;
}
public static void RestoreInputToCorrectOrder()
{
    double[,] correct = new double[,]...;
    Array.Copy(correct, DataProvider.input, DataProvider.input.Length);
}
public static void ShuffleInputData()
{
    int inputDataCount = input.GetLength(0);
    int inputDataAttributionsCount = input.GetLength(1);
    Random r = new Random();
    int place;

    double[] tmp = new double[inputDataAttributionsCount];
    for (int i = 0; i < inputDataCount; i++)
    {

        for (int j = 0; j < inputDataAttributionsCount; j++)
        {
            tmp[j] = input[i, j];
        }

        place = r.Next(i, inputDataCount);

        for (int j = 0; j < inputDataAttributionsCount; j++)
        {
            input[i, j] = input[place, j];
        }

        for (int j = 0; j < inputDataAttributionsCount; j++)
        {
            input[place, j] = tmp[j];
        }
    }
}
```