Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Методы машинного обучения

Лабораторная работа №1

Метод k ближайших соседей

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: доцент кафедры ПМиК

Ракитский Антон Андреевич

Новосибирск, 2020

**Содержание**

1. Задание.
2. Текст программы.
3. Результаты работы
4. Вывод

**Задание**

Суть лабораторной работы заключается в написании классификатора на

основе метода k ближайших соседей. Данные из файла необходимо разбить

на две выборки, обучающую и тестовую, согласно общепринятым

правилам разбиения. На основе этих данных необходимо обучить

разработанный классификатор и протестировать его на обеих выборках.

Тип классификатора: Метод парзеновского окна с фиксированным h

Функция ядра: Q – квартическое

**Текст программы**

using CsvHelper;

using Microsoft.Win32.SafeHandles;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

namespace MMO\_Lab1

{

public class CsvLine

{

public int Param\_1

{

get;

set;

}

public int Param\_2

{

get;

set;

}

public int Class\_number

{

get;

set;

}

}

class Program

{

static double EuclidDistance(CsvLine A, CsvLine B)

{

return Math.Sqrt(Math.Pow(A.Param\_1 - B.Param\_1, 2) + Math.Pow(A.Param\_2 - B.Param\_2, 2));

}

static double Weight(CsvLine U, CsvLine X)

{

return EuclidDistance(U, X);

}

static double Q\_kvartic\_function(double distance)

{

double r = Math.Min(distance / 10, 1);

if (r > 1)

{

return 0;

}

else

{

return Math.Pow((1 - (r \* r)), 2);

}

}

static List<Tuple<double, CsvLine>> Get\_neighbors(List<CsvLine> test\_mas, CsvLine train\_object, int k)

{

List<Tuple<double, CsvLine>> distances = new List<Tuple<double, CsvLine>>();

List<Tuple<double, CsvLine>> neighbors = new List<Tuple<double, CsvLine>>();

List<Tuple<double, CsvLine>> sorted\_distances = new List<Tuple<double, CsvLine>>();

foreach (CsvLine test in test\_mas)

{

distances.Add(new Tuple<double, CsvLine>(Weight(train\_object, test), test));

}

sorted\_distances = distances.OrderBy(o => o.Item1).ToList();

for (int i = 0; i < k; i++)

{

neighbors.Add(new Tuple<double, CsvLine>(Q\_kvartic\_function(sorted\_distances[i].Item1), sorted\_distances[i].Item2));

}

return neighbors;

}

private static void Main(string[] args)

{

if (args is null)

{

throw new ArgumentNullException(nameof(args));

}

Random rand = new Random();

const double split = 1 / 3f;

StreamReader reader = new StreamReader("data1.csv");

CsvReader data\_file = new CsvReader(reader, CultureInfo.InvariantCulture);

data\_file.Configuration.HasHeaderRecord = false;

CsvLine[] csv\_massive = data\_file.GetRecords<CsvLine>().ToArray();

List<int> classes = new List<int>();

List<CsvLine> train\_mas = new List<CsvLine>();

List<CsvLine> test\_mas = new List<CsvLine>();

Dictionary<int, List<CsvLine>> full\_csv\_mas = new Dictionary<int, List<CsvLine>>();

foreach (CsvLine csv\_element in csv\_massive)

{

if (!classes.Contains(csv\_element.Class\_number))

{

classes.Add(csv\_element.Class\_number);

}

}

foreach (int index in classes)

{

full\_csv\_mas.Add(index, new List<CsvLine>());

}

foreach (CsvLine csv\_element in csv\_massive)

{

full\_csv\_mas[csv\_element.Class\_number].Add(csv\_element);

}

foreach (CsvLine csv\_element in full\_csv\_mas.SelectMany(full\_csv\_mas\_key => full\_csv\_mas\_key.Value))

{

if (rand.NextDouble() < split)

{

test\_mas.Add(csv\_element);

}

else

{

train\_mas.Add(csv\_element);

}

}

Console.WriteLine($"Test\_mas count: {test\_mas.Count}");

Console.WriteLine($"Train\_mas count: {train\_mas.Count}");

Console.WriteLine($"Total count: {test\_mas.Count + train\_mas.Count}");

for (int k = 1; k < 11; k++)

{

int errors = 0;

int total\_objects = train\_mas.Count;

foreach (CsvLine train\_object in train\_mas)

{

Dictionary<int, double> class\_weight = new Dictionary<int, double>();

List<Tuple<double, CsvLine>> neighbors = Get\_neighbors(test\_mas, train\_object, k);

foreach (int index in classes)

{

class\_weight.Add(index, 0f);

}

foreach (Tuple<double, CsvLine> neighbor in neighbors)

{

class\_weight[neighbor.Item2.Class\_number] += neighbor.Item1;

}

int class\_index = -1;

double max\_weight = 0;

foreach (int index in classes)

{

if (max\_weight < class\_weight[index])

{

max\_weight = class\_weight[index];

class\_index = index;

}

}

if (class\_index == -1)

{

Dictionary<int, int> class\_count = new Dictionary<int, int>();

foreach (int classIndex in classes)

{

class\_count.Add(classIndex, 0);

}

foreach (Tuple<double, CsvLine> neighbor in neighbors)

{

class\_count[neighbor.Item2.Class\_number] += 1;

}

int count = 0;

foreach (int index in classes)

{

if (count < class\_count[index])

{

count = class\_count[index];

class\_index = index;

}

}

}

if (train\_object.Class\_number != class\_index)

{

errors++;

}

}

Console.WriteLine($"Correct precisions for {k} neighbors: {100 - errors \* 100 / (double)total\_objects:F2}%");

}

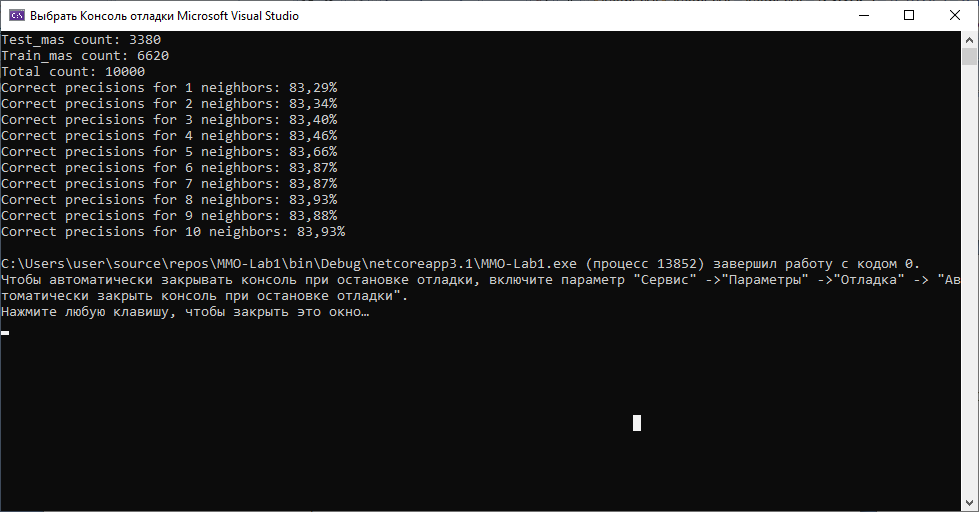
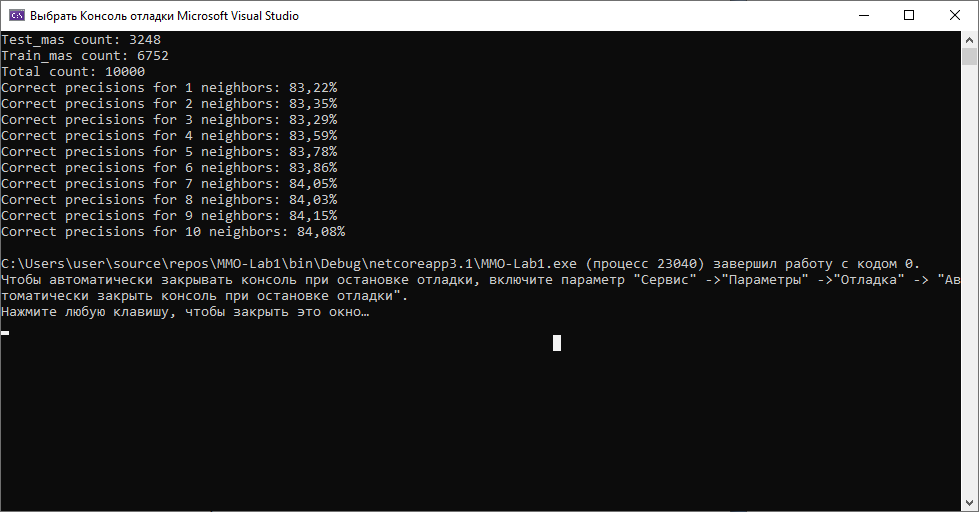
}

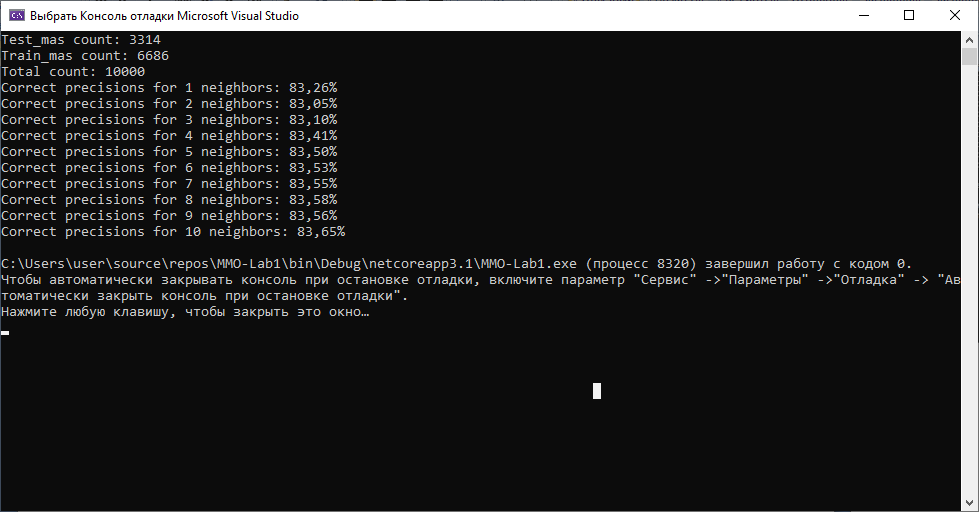
}

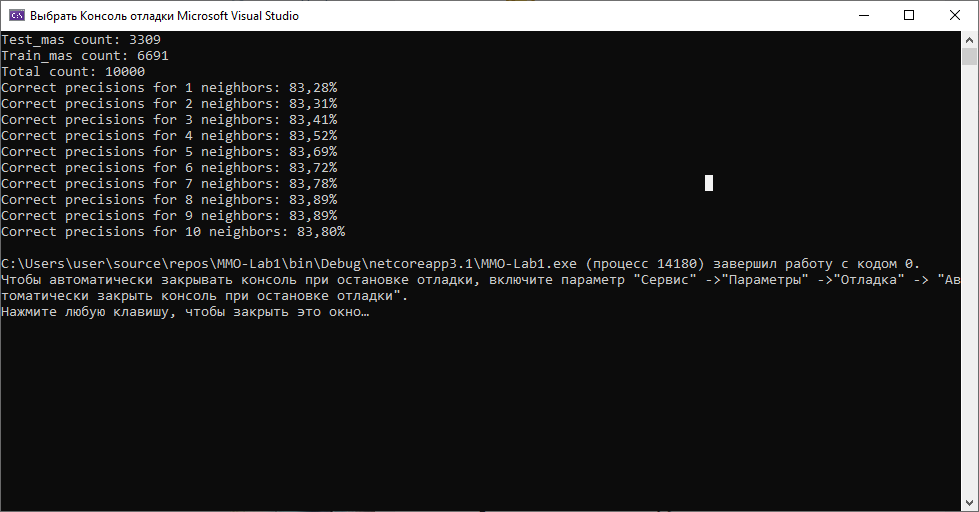
}

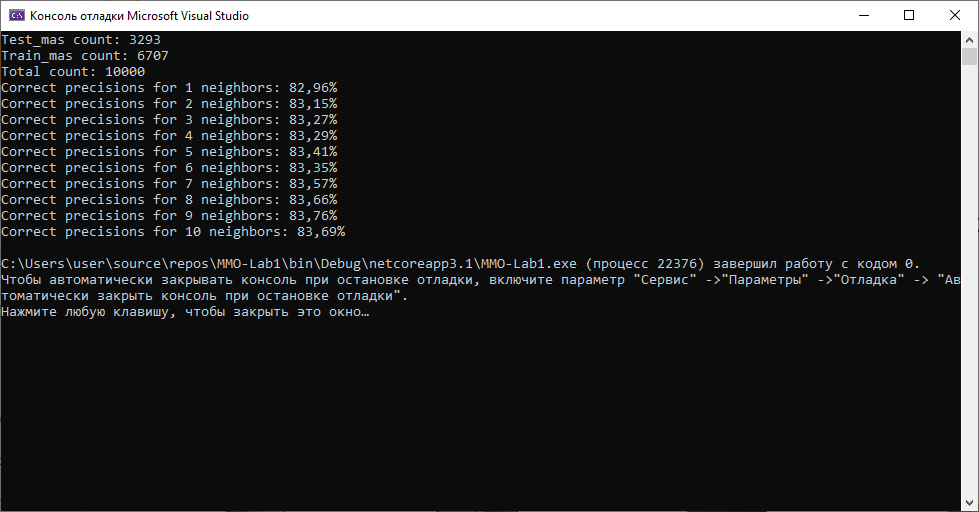
**Результаты работы**

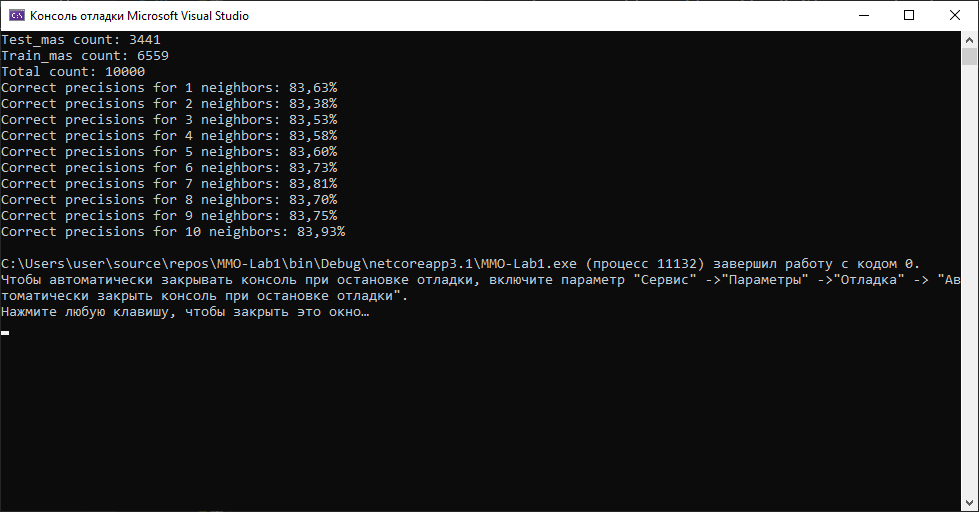
**h=10;**

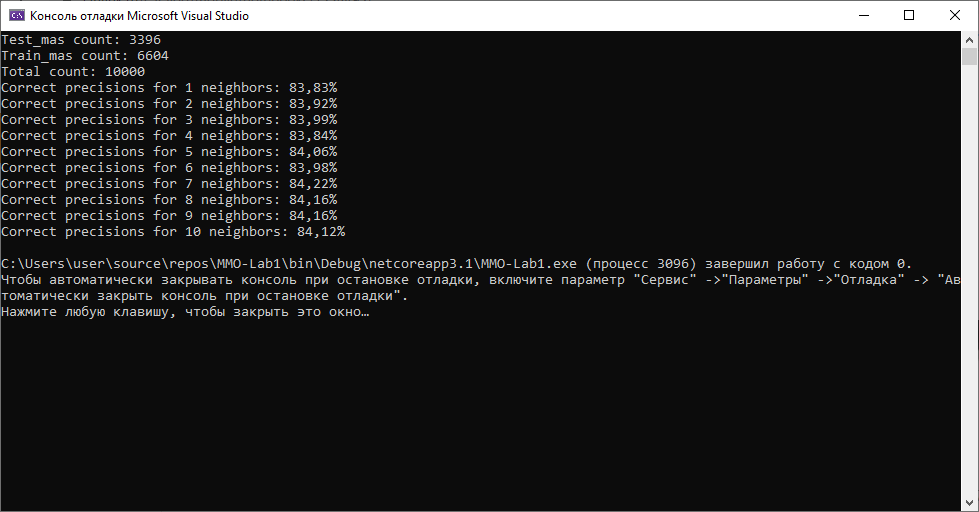
** **

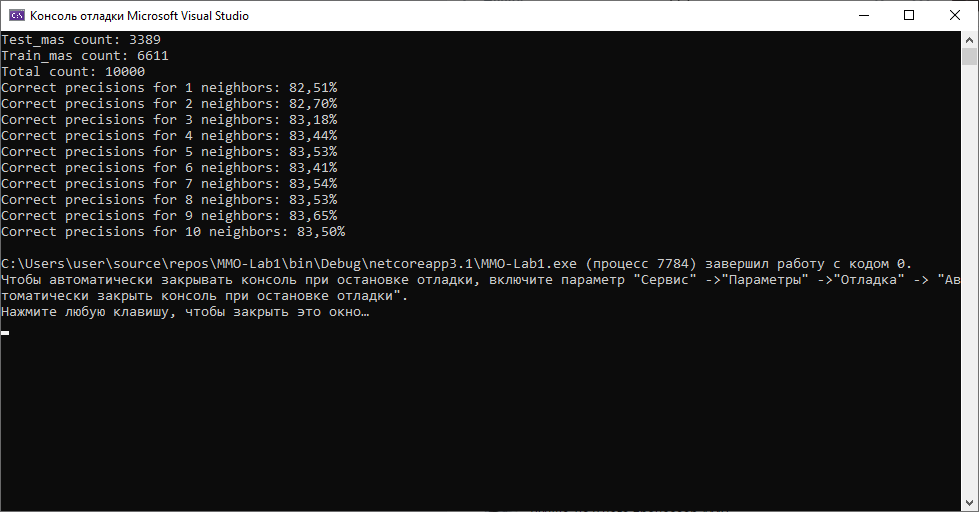


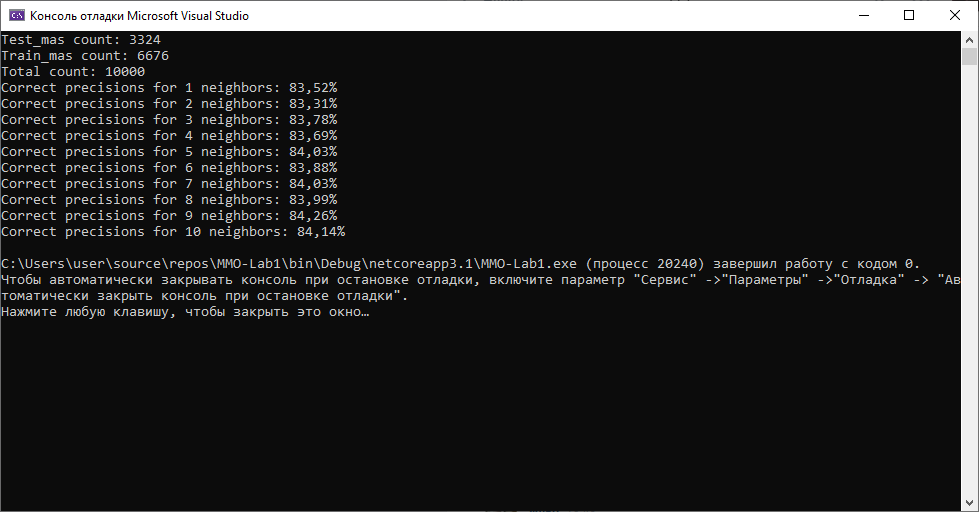


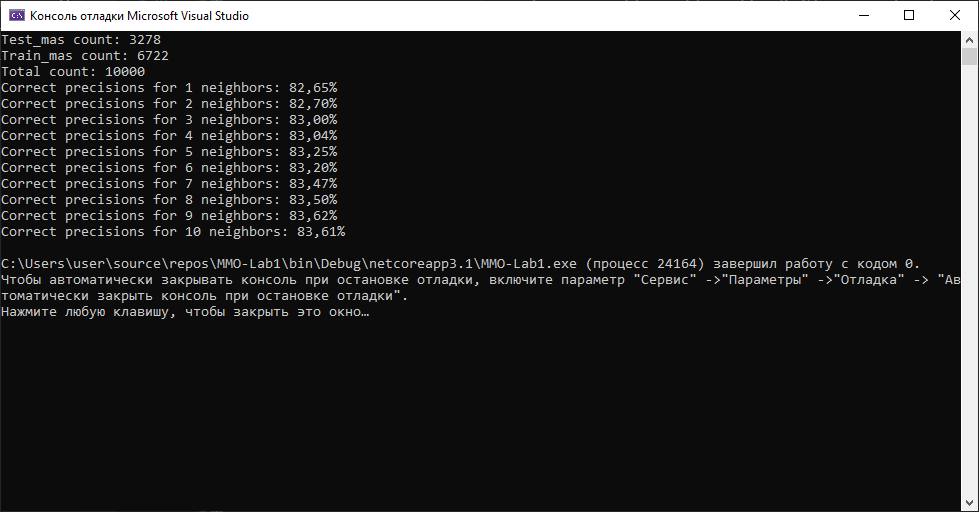












Оптимальное диапазон k – (7 – 10)

**Вывод**

В результате практической работы был создан классификатор использующий метод k ближайших соседей с использованием метода парзеновского окна с фиксированным размером окна и Q – квартической функцией ядра K(z).