УДК 519.683.

Алексеев Г.С.,

студент

3 курс, кафедра «Теплофизика и информатика в металлургии»

факультет: «Институт новых материалов и технологий»

Уральский Федеральный университета

Россия, г. Екатеринбург

РЕАЛИЗАЦИЯ АЛГОРИТМА ТАКСОНОМИИ ВЕКТОРОВ НА ОСНОВЕ МЕТОДА КОРРЕЛЯЦИОННЫХ ПЛЕЯД НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C#

Аннотация: *статья посвящена программной реализации базового алгоритма распознавания образов – алгоритма таксономии, при котором из таксономируемого множества векторов выбираются наиболее близко расположенные между собой и объединяются в одну общую группу – таксон. Подробно описан данный алгоритм, приведена конкретная его реализация с листингами программного кода. Продемонстрирована работа полноценной программы для определения «похожих» друг на друга векторов. Приложение написано на языке программирования C# с использованием технологии Windows Forms.*

Ключевые слова: *распознавание образов, корреляционные плеяды, нормировка, таксономия.*

Abstract: the article is devoted to software realization of the basic algorithm the pattern recognition algorithm taxonomy, in which many vectors are chosen most closely located each other and unite in one common group – Tucson. Described this algorithm, given a particular implementation with listings of program code. Demonstrate operation of the full program to define "similar" to each other vectors. The application is written in the programming language C# using Windows Forms.

Key words: pattern recognition, correlation Pleiades, normalization, taxonomy.

Обозначим задачу: необходимо написать программу, которая бы могла обработать файл Excel, содержащий, например, количественные анкетные данные некоторой группы людей и предоставить информацию о том, кто из этой группы на кого похож (чьи интересы совпадают, чей характер является близким и т.д.). В качестве инструмента выберем интегрированную среду разработки (IDE) Visual Studio 2017 Enterprise, содержащей весь необходимый функционал для реализации решения поставленной задачи.

Чтение из Excel файла реализуем с использованием библиотеки Microsoft.Office.Interop.Excel, предварительно подключив ее через менеджер ссылок в Обозревателе решений. Для отображения информации из таблицы Excel на экранной форме приложения удобнее всего использовать элемент управления DataGridVeiw – сокращенно DGV. Листинг метода, реализующего импорт и отображение данных, представлен на Рис. 1.

```
///<summary>
///Метод для импорта Excel в DGV
///</summary>
///<param name="sender"></param>
///<param name="e"></param>
 ссылка: 1 | 0 изменений | 0 авторов, 0 изменений
private void openData_btn_Click(object sender, EventArgs e)
{
    SetDoubleBuffered(DGV_1);//Устанавливаем двойную буферизацию для DGV
    DGV_1.Rows.Clear(); //очищаем DGV. Для возможности повторного импорта
    OpenFileDialog dialog = new OpenFileDialog();//открываем файловый диалог
    dialog.Filter = "MS Excel | *.xlsx";//фильтруем файлы
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        Excel.Application app = new Excel.Application();//создаем экземпляр Excel
        Excel.Workbook workbook = app.Workbooks.Open(dialog.FileName);//добавляем выбранный файл в экземпляр Excel
        Excel.Worksheet worksheet = workbook.ActiveSheet;//инициализируем активный лист

        DGV_1.ColumnCount = worksheet.UsedRange.Columns.Count;//инициализируем количество колонок в DGV
        for (int index = 0; index < DGV_1.ColumnCount; index++)
        {
            DGV_1.Columns[index].Name = "Признак " + (index + 1);//заполняем наименования признаков в цикле
        }
        int rcount = worksheet.UsedRange.Rows.Count; //инициализируем кол-во строк
        DGV_1.Rows.Add(rcount);//добавляем это количество
        int columnCount = worksheet.UsedRange.Columns.Count;
        for (int i = 1; i < rcount; i++)
        {
            for (int j = 0; j < columnCount; j++)
            {
                DGV_1[j, i].Value = worksheet.Cells[i + 1, j + 1].Value;//инициализируем все ячейки значениями из Excel

            }
        }
        app.Quit();//закрываем приложение Excel
    }
}
```

Рис. 1. Листинг кода внутри метода, предназначенного для импорта данных из Excel

Несколько слов о вызываемом в самом начале методе SetDoubleBuffered(Control control), принимающем в качестве параметра класса

Control созданный DGV: установка двойной буферизации сделана для более быстрой работы DGV, поскольку в связке с библиотекой Microsoft.Office.Interop.Excel производительность DGV резко уменьшается. Листинг кода внутри метода SetDoubleBuffered(Control control) представлен на Рис. 2.

```
/// <summary>
/// Метод для добавления двойной буферизации
/// </summary>
/// <param name="control"></param>
 ссылка: 1 | 0 изменений | 0 авторов, 0 изменений
public static void SetDoubleBuffered(Control control)
{
    // set instance non-public property with name "DoubleBuffered" to true
    typeof(Control).InvokeMember("DoubleBuffered",
        BindingFlags.SetProperty | BindingFlags.Instance | BindingFlags.NonPublic,
        null, control, new object[] {true});
}
```

Рис. 2. Листинг кода внутри метода SetDoubleBuffered(Control control)

Количественные признаки перед обработкой обычно подвергаются процедуре нормирования. Это необходимо для того, чтобы привести их к одному интервалу принимаемых значений и избавиться, таким образом, от их неравнозначности. Процедура нормировки выглядит следующим образом:

$$x_i^{(j)}{}_{норм} := \frac{x_i^{(j)} - x_{\min}^{(j)}}{\Delta x^{(j)}} \forall j \in \overline{1,m},$$

где $x_{\min}^{(j)}$ - минимальное значение j-го признака, $\Delta x^{(j)}$ - разность между максимальным и минимальным значениями j-го признака, $x_i^{(j)}$ и $x_i^{(j)}{}_{норм}$ - значения j-го признака i-го вектора до и после нормирования, «:=» - знак присваивания значения. Реализуем данную задачам проходам по всем ячейкам DGV, подменяя в них значения на нормированные. Листинг соответствующего метода представлен на Рис. 3.

Вызываемые методы min(int i), max(int i) предназначены для отыскания минимального и максимального значения соответственно в столбце DGV с индексом i. Реализованы эти методы классическим подходом к решению подобных задач: установка первого элемента массива за минимальный (максимальный) и сравнение всех последующих с ним, в случае, если один из

следующих элементов оказался меньше (больше), то минимум (максимум) переназначается им. Начинать необходимо со второго столбца, поскольку внутри первого нет численных значений, в нем хранятся только наименования векторов. Также внутри метода Normalize добавлена проверка делителя на 0.

```
///<summary>
///Метод для проведения нормировки по ячейкам DGV
///</summary>
ссылка: 1 | 0 изменений | 0 авторов, 0 изменений
private void Normalize(){
    for (int i = 1; i < DGV_1.ColumnCount; i++) //необходимо начинать со второго столбца,
    {
        double minimum = min(i);
        double maximum = max(i);
        for (int j = 0; j < DGV_1.RowCount; j++){
            double num = Convert.ToDouble(DGV_1[i, j].Value) - minimum;
            double den = maximum - minimum;
            if (den == 0)
            {
                den = 1;
            }
            else
            {
                DGV_1[i, j].Value = Math.Round(num / den, 2);
            }
        }
    }
}
```

Рис. 3. Листинг кода метода нормализации данных

Далее можно приступать непосредственно к поиску неявно схожих векторов. Опишем алгоритм таксономии:

1. Задается некоторое пороговое расстояние *R* – величина, характеризующая искомую «похожесть» векторов.
2. Выбирается один из векторов таксономируемого множества, например, первый из них, и включается в таксон – в группу похожих между собой векторов.
3. В таксономируемом множестве ищутся векторы, удаленные от выбранного не больше, чем на *R*. Если таких векторов не обнаружено, формирование этого таксона заканчивается. Если такие векторы обнаружены, они включаются в таксон.
4. Затем для каждого из вновь включенных в таксон векторов таким же образом ищутся ближайшие и тоже включаются в таксон и т. д. до тех пор,

пока близкие векторы перестанут находиться. Получается некий древовидный поиск. На этом формирования таксона заканчивается.

5. Если остались векторы, не включенные в таксон, выбирается один из таких векторов, с которого начнется формирование нового таксона в соответствии с пунктами 3 и 4 и так до тех пор, пока все векторы не будут включены в таксоны.

Количество таксонов, получаемых в результате работы алгоритма, зависит от величины *R*. При очень малых значениях *R* каждый таксон будет состоять из одного объекта, при слишком больших *R* все объекты могут объединиться в один таксон. Рис. 4 иллюстрирует работу алгоритма для некоторого заданного R.

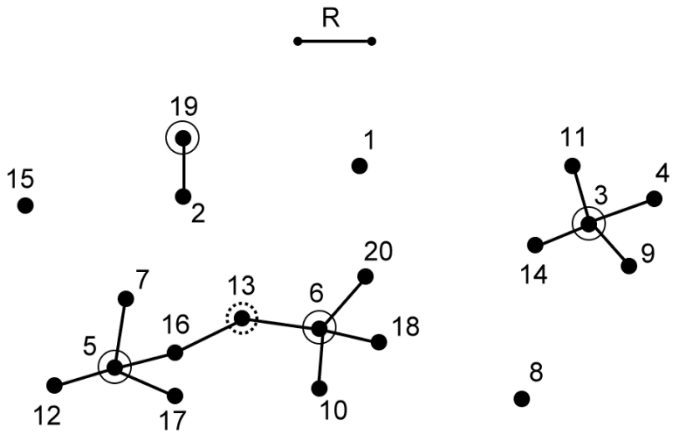


Рис. 4. Иллюстрация алгоритма таксономии

Для вычисления расстояния между векторами используют метрику Евклида. При этом, если для описания объектов используются *n* признаков, то расстояние *R* между двумя векторами $x = (x^{(1)},..., x^{(n)})$ и $y = (y^{(1)},..., y^{(n)})$ вычисляется следующим образом:

$$R = R(x, y) = \sqrt{\sum_{i=1}^{n} (x^{(i)} - y^{(i)})^2}.$$

Приведенная формула применяется для вычисления расстояния межу векторами, координатами которых являются произвольные вещественные

числа. При работе с векторами, координаты которых представлены в двоичном виде, применяется метрика Хемминга:

$$R = R(x, y) = \sum_{i=1}^{n} | x^{(i)} - y^{(i)} | .$$

Реализуем данный алгоритм на языке программирования C#: в роли векторов будут выступать строки из DGV в качестве признаков – значения его ячеек. Листинг кода поиска таксонов с соответствующими комментариями приведен на Рис. 5.

```
///<summary>
///Метод для выявления таксонов в таксономируемом множестве векторов
///</summary>
ссылка: 1 | 0 изменений | 0 авторов, 0 изменений
public void SearchTaxon()
{
    outputTxt.Clear();
    if (inputTxt.Text != "")
    {
        var unaddedVectors = GetData(); //хранилище недобавленных в таксон векторов
        double R = Convert.ToDouble(inputTxt.Text); //пороговое расстояние
        var taxon = new List<DataGridViewRow>(); //список для хранения векторов
        int counter = 1; //счетчик для определения номера таксона

        while (unaddedVectors.Count != 0) //пока в множестве недобавленных таксонов есть элементы
        {
            taxon.Clear(); //очищаем таксон, который, возможно, был заполнен на предыдущих итерациях
            taxon.Add(unaddedVectors[0]); //добавляем в таксон первый вектор множества недобавленных векторов
            unaddedVectors.RemoveAt(0); //удаляем его из этого множества.
            for (int j = taxon.Count - 1; j >= 0;j--) //начинаем обход по векторам таксона, начиная с самого последнего
            {
                for (int i = 0; i < unaddedVectors.Count; i++) //цикл для обхода всех оставшихся
                //векторов во множестве недобавленных векторов
                {
                    if (GetRange(taxon[j], unaddedVectors[i]) <= R) //если попадаем в критерий
                    {
                        taxon.Add(unaddedVectors[i]); //добавляем в таксон вектор из множества векторов
                        unaddedVectors.RemoveAt(i); //удаляем его из множества недобавленных в таксон векторов
                        i--; //корректировка индекса
                    }
                }
            }
            ShowTaxon(taxon, counter); //выводим элементы полученного таксона в текстбокс
            counter++;
        }
    }

    else
    {
        MessageBox.Show("Введите в поле корректное значение");
    }
}
```

Рис. 5. Листинг кода для поиска таксонов

Ключевой метод, стоит заметить, целиком и полностью реализован в процедурной парадигме программирования, несмотря на то, что C# является

объектно – ориентированным языком программирования. Это позволило добиться явного прироста производительности программы.

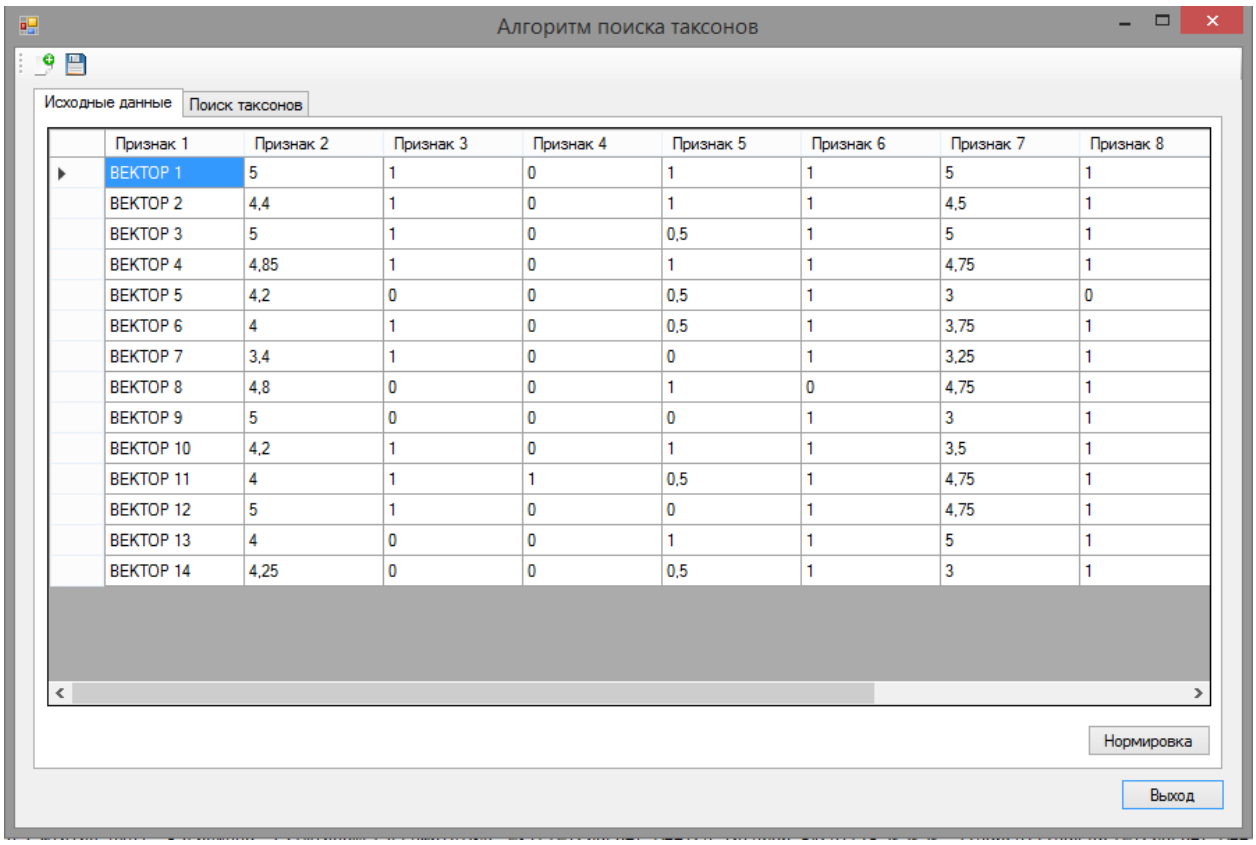
Также стоит заметить, что внутри метода реализована проверка на пустой ввод внутри поля R, что позволит избежать непредвиденного поведения в работе программы из-за некорректных вводимых данных.

Проверим работу программы на тестовых исходных данных, которые представлены в Таблице 1, в которой представлено 14 векторов и 10 признаков (предполагаемых количественных данных на основании анкетирования людей).

Таблица №1

ВЕКТОР №	1	2	3	4	5	6	7	8	9	10
ВЕКТОР 1	5	1	0	1	1	5	1	1	45	2
ВЕКТОР 2	4,4	1	0	1	1	4,5	1	1	30	3
ВЕКТОР 3	5	1	0	0,5	1	5	1	1	40	2
ВЕКТОР 4	4,85	1	0	1	1	4,75	1	1	25	3
ВЕКТОР 5	4,2	0	0	0,5	1	3	0	0	35	4
ВЕКТОР 6	4	1	0	0,5	1	3,75	1	0	30	2
ВЕКТОР 7	3,4	1	0	0	1	3,25	1	1	20	3
ВЕКТОР 8	4,8	0	0	1	0	4,75	1	0	20	2
ВЕКТОР 9	5	0	0	0	1	3	1	1	10	30
ВЕКТОР 10	4,2	1	0	1	1	3,5	1	1	30	2
ВЕКТОР 11	4	1	1	0,5	1	4,75	1	0	60	3
ВЕКТОР 12	5	1	0	0	1	4,75	1	0	35	5
ВЕКТОР 13	4	0	0	1	1	5	1	1	50	10
ВЕКТОР 14	4,25	0	0	0,5	1	3	1	1	10	4

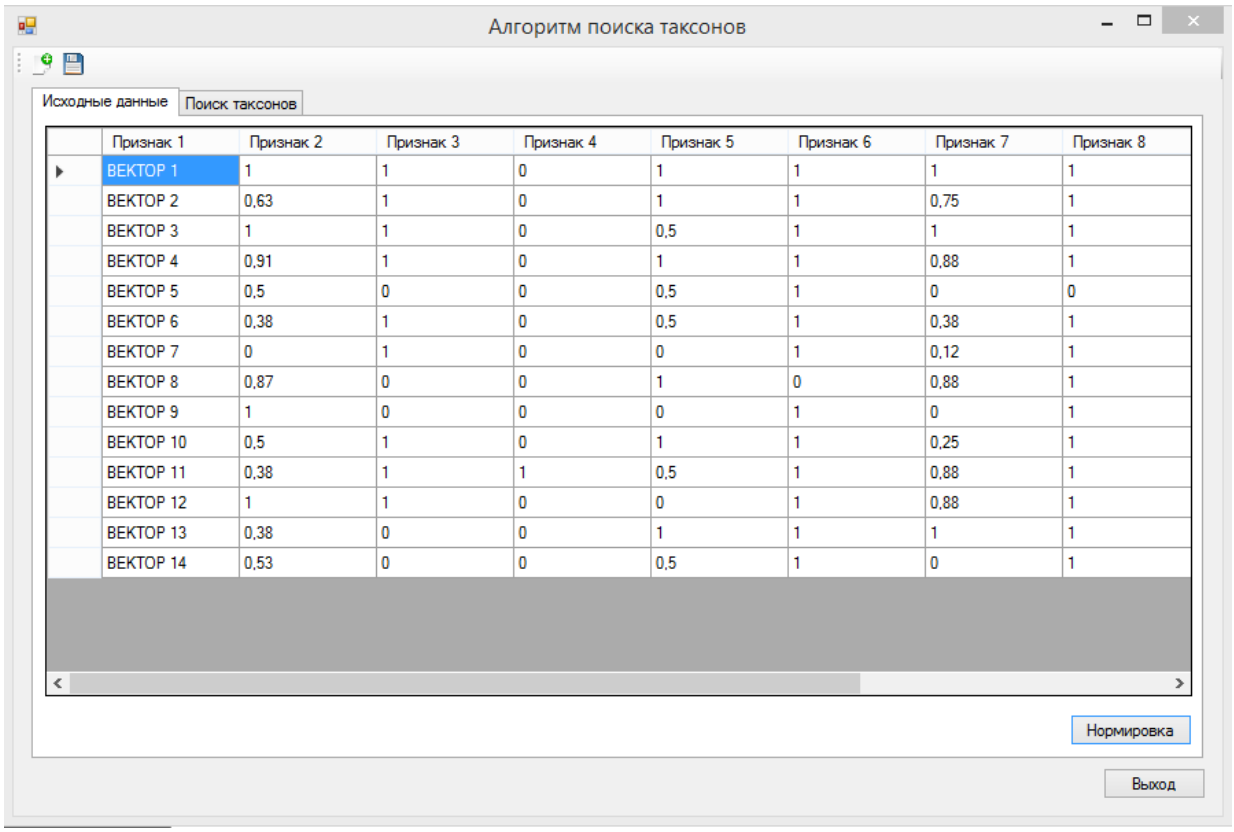
Главная экранная форма приложения после импортирования excel файла представлена на Рис. 6.



	Признак 1	Признак 2	Признак 3	Признак 4	Признак 5	Признак 6	Признак 7	Признак 8
▶	ВЕКТОР 1	5	1	0	1	1	5	1
	ВЕКТОР 2	4,4	1	0	1	1	4,5	1
	ВЕКТОР 3	5	1	0	0,5	1	5	1
	ВЕКТОР 4	4,85	1	0	1	1	4,75	1
	ВЕКТОР 5	4,2	0	0	0,5	1	3	0
	ВЕКТОР 6	4	1	0	0,5	1	3,75	1
	ВЕКТОР 7	3,4	1	0	0	1	3,25	1
	ВЕКТОР 8	4,8	0	0	1	0	4,75	1
	ВЕКТОР 9	5	0	0	0	1	3	1
	ВЕКТОР 10	4,2	1	0	1	1	3,5	1
	ВЕКТОР 11	4	1	1	0,5	1	4,75	1
	ВЕКТОР 12	5	1	0	0	1	4,75	1
	ВЕКТОР 13	4	0	0	1	1	5	1
	ВЕКТОР 14	4,25	0	0	0,5	1	3	1

Рис.6. Главная экранная форма

Проведем операцию нормировки и посмотрим, как изменятся данные (см. Рис. 7). Как можно видеть, нормированные значения не превышают единицы.



	Признак 1	Признак 2	Признак 3	Признак 4	Признак 5	Признак 6	Признак 7	Признак 8
▶	ВЕКТОР 1	1	1	0	1	1	1	1
	ВЕКТОР 2	0,63	1	0	1	1	0,75	1
	ВЕКТОР 3	1	1	0	0,5	1	1	1
	ВЕКТОР 4	0,91	1	0	1	1	0,88	1
	ВЕКТОР 5	0,5	0	0	0,5	1	0	0
	ВЕКТОР 6	0,38	1	0	0,5	1	0,38	1
	ВЕКТОР 7	0	1	0	0	1	0,12	1
	ВЕКТОР 8	0,87	0	0	1	0	0,88	1
	ВЕКТОР 9	1	0	0	0	1	0	1
	ВЕКТОР 10	0,5	1	0	1	1	0,25	1
	ВЕКТОР 11	0,38	1	1	0,5	1	0,88	1
	ВЕКТОР 12	1	1	0	0	1	0,88	1
	ВЕКТОР 13	0,38	0	0	1	1	1	1
	ВЕКТОР 14	0,53	0	0	0,5	1	0	1

Рис. 7. Данные после нормирования

Перейдем на вкладку «Поиск таксонов» и попробуем найти похожие вектора (см. Рис. 8).

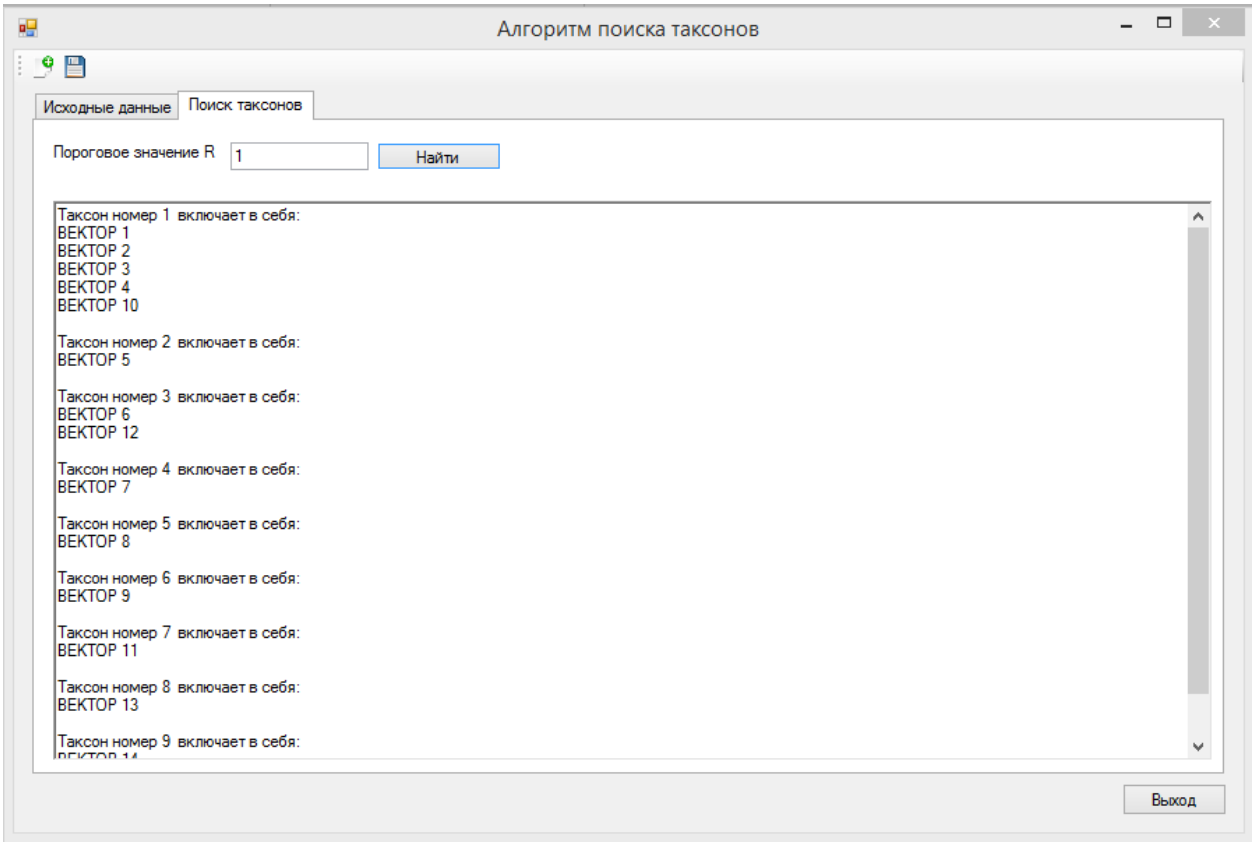


Рис. 8. Таксоны с пороговым значением 1

При R = 1 большинство векторов обособлены, что означает, что они не похожи на них. Попробуем увеличить пороговое значение R до 3 (см. Рис. 9).

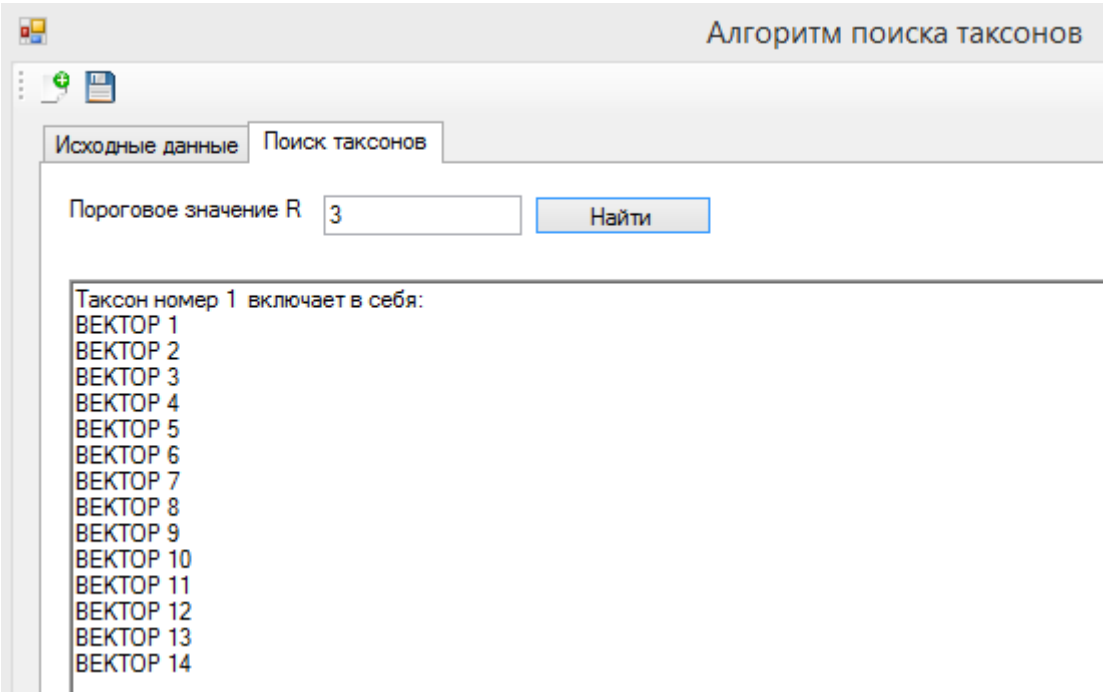


Рис. 9. Таксоны при большом пороговом значении

Таким образом, можно заметить, что все вектора были объединены в одну общую группу, поскольку при таком большом расстоянии все векторы оказываются схожими друг с другом. Данная программа, конечно, является весьма простой реализацией данного подхода, однако его можно использовать в очень широком спектре задач: от поиска похожих характеров людей до выявления наиболее оптимальных технологий на производстве.

Использованные источники:

1. Шаталкин А.И. Таксономия. Основания, принципы и правила. / А.И. Шаталкин. — Москва: Товарищество научных изданий КМК, 2012. — 600 с.;
2. Распознавание образов. Состояние и перспективы. / Верхаген К., Дёйн Р., Грун Ф. и др. Москва: Радио и связь, 1985. — 104 с.;
3. Иен Г. Программирование на C# 5.0/ Г. Иен. — Москва: Эксмо. — 2014. — 1136 с.