

Лабораторная работа №3
студента группы ИТ – 32
Курбатовой Софьи Андреевны

Выполнение: _____ Защита _____

КЛАСТЕРИЗАЦИЯ ТЕКСТА ПО ТЕМАТИКАМ.

Цель работы: научиться реализовывать на выбранном языке программирования алгоритмы кластеризации текстов по тематикам на основе латентно-семантического анализа.

Содержание работы

Написать программу на выбранном языке программирования, реализующую описанный выше алгоритм для кластеризации заголовков по темам. Количество заголовков задать не более 10 (оптимально от 7 до 10), количество кластеров не более 3. Программа должна запрашивать заголовки (они могут храниться в файлах). После получения двумерного сингулярного разложения осуществить прямую кластеризацию самостоятельно выбранным и изученным методом (иерархические алгоритмы, поиск k-средних и т.д.). Результатом работы программы должен быть файл, содержащий заголовки, разбитые по кластерам..

Ход работы

1. При латентно-семантическом анализе не учитываются формы слова. В качестве исходных данных метод использует частоту использования слов в отрывках текста. И данные собираются о множестве использования слов в большом массиве отрывков.

Поэтому для реализации кластеризации заголовков по темам, можно предположить следующий алгоритм:

1. Удалить из корпуса заголовков все стоп-слова и реализовать стемминг оставшихся слов.

2. Составить частотную матрицу, при этом необходимо понизить ранг матрицы, так как возможно попадание сразу всех слов из текста.

3. Провести сингулярное разложение полученной матрицы.

4. Так как столбцы и строки соответствующие меньшим сингулярным значениям дают меньший вклад в итоговое произведение, то от получившейся матрицы оптимально оставить первые две строки.

5. С использованием алгоритма кластеризации выявить группы заголовков по темам.

6. Вывести результат.

Листинг 1.1. Класс для стемминга слов в заголовках.

[illegible]


```

        "| которого | перед | уже | всегда | которые | по | хорошо | всего | кто | под | хоть | всех  

| куда | после | чего | всю | ли " +
        "| потом | человек | вы | лучше | потому | чем | г | между | почти | через | где | меня | при | что  

| говорил | мне | про " +
        "| чтоб | да | много | раз | чтобы | даже | может | разве | чуть | два | можно | с | эти | для | мой  

| сам | этого | до | моя " +
        "| свое | этой | другой | мы | свою | этом | его | на | себе | этот | ее | над | себя | эту | ей  

| надо | сегодня | я | ему " +
        "| наконец | сейчас | если | нас | сказал | есть | не | сказала )", RegexOptions.IgnoreCase);
    static Regex stopSymbolsMiddle = new Regex("( еще | него | сказать | а | ж | нее | со |  

без | же | ней " +
        "| совсем | более | жизнь | нельзя | так | больше | за | нет | такой | будет | зачем  

| ни | там | будто " +
        "| здесь | нибудь | тебя | бы | и | никогда | тем | был | из | ним | теперь | была |  

изза | них | то " +
        "| были | или | ничего | тогда | было | им | но | того | быть | иногда | ну | тоже |  

в | их | о | только " +
        "| вам | к | об | том | вас | кажется | один | тот | вдруг | как | он | три | ведь |  

какая | она | тут" +
        "| во | какой | они | ты | вот | когда | опять | у | впрочем | конечно | от | уж |  

все | которого | перед " +
        "| уже | всегда | которые | по | хорошо | всего | кто | под | хоть | всех | куда |  

после | чего | всю | ли " +
        "| потом | человек | вы | лучше | потому | чем | г | между | почти | через | где |  

меня | при | что " +
        "| говорил | мне | про | чтоб | да | много | раз | чтобы | даже | может | разве |  

чуть | два | можно " +
        "| с | эти | для | мой | сам | этого | до | моя | свое | этой | другой | мы | свою |  

этом | его | на | себе" +
        "| | этот | ее | над | себя | эту | ей | надо | сегодня | я | ему | наконец | сейчас  

| если | нас | сказал " +
        "| | есть | не | сказала )", RegexOptions.IgnoreCase);

    static Regex figures = new Regex("(\\d)");

    static Regex punctuationSymbols = new Regex("(\\W)");

    static char[] vowels = { 'a', 'o', 'y', 'ы', 'э', 'я', 'е', 'ю', 'и', 'А', 'О', 'У', 'Ы',  

'Э', 'Я', 'Е', 'Ю', 'И' };
    static char[] consonants = { 'б', 'в', 'г', 'д', 'ж', 'з', 'й', 'к', 'л', 'м', 'н', 'п',  

'р', 'с', 'т', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ь', 'ъ' };
#endregion

```

Листинг 1.3. Метод реализующий удаление стоп-символов

```

/*Remove stop Symbols. Result of it write in output file */
/*Удаляем стоп-символы. Результат запишем в выходной файл.
Если этот файл уже существовал, то он перезапишется. */
static void RemovingStopSymbols()
{
    using (StreamReader InputHeaders = new StreamReader(inputFileName))// all line from
input file
    {
        // using (StreamWriter newHeadings = new StreamWriter("D:\\newHeadings.txt"))
        using (StreamWriter NewHeaders = new StreamWriter("C:\\test\\output.txt"))
        {
            string line;
            while ((line = InputHeaders.ReadLine()) != null)
            {
                HEADERS.Add(line);
                line = stopSymbolsStart.Replace(line, "");
                line = figures.Replace(line, "");
                line = punctuationSymbols.Replace(line, " ");
                line = stopSymbolsMiddle.Replace(line, " ");
                NewHeaders.WriteLine(line);//here write new line in new file
            }
        }
    }
}

```

```
}
```

Листинг 1.4. Методы реализующие стемминг

```
/*Check that word is word, not single character*/
static string RVFinding(string word)
{
    int position = -1;
    bool check = false;
    for (int i = 0; i < word.Length; i++)
    {
        int j = 0;
        foreach (char ch in vowels)
        {
            if (word[i] == vowels[j])
            {
                position = i;
                check = true;
                break;
            }
            j++;
        }
        if (check)
            break;
    }
    if ((position == -1) || (position == word.Length - 1))
        return "";
    else
    {
        string rv = "";
        for (int i = position + 1; i < word.Length; i++)
            rv += word[i];
        return rv;
    }
}

static string R1Finding(string word)
{
    string r1 = "";
    bool check = false;
    int pos = -1;
    if (word.Length < 2)
        return "";
    for (int i = 0; i < word.Length - 1; i++)
    {
        int f = 0;
        foreach (char c in vowels)
        {
            if (word[i] == vowels[f])
            {
                check = true;
                break;
            }
            f++;
        }
        if (check)
        {
            f = 0;
            foreach (char c in consonants)
            {
                if (word[i + 1] == consonants[f])
                {
                    pos = i + 1;
                    break;
                }
                f++;
            }
        }
    }
}
```

```

        if (pos != -1)
            break;
    }
    if (pos == -1)
        return "";
    for (int i = pos + 1; i < word.Length; i++)
        r1 += word[i];
    return r1;
}

/*Stemmer*/
static void Stemming()
{
    Porter p = new Porter();
    using (StreamReader newHeadings = new StreamReader("C:\\test\\output.txt"))
    {
        using (StreamWriter newHeadingsAfterStemming = new
StreamWriter("C:\\test\\output2.txt"))
        {
            string line;
            while ((line = newHeadings.ReadLine()) != null)
            {
                int i = 0;
                string word = "";
                string rv, r1, r2;
                while (i < line.Length)
                {
                    if (line[i] != ' ')
                    {
                        word += line[i];
                    }
                    else
                    {
                        rv = RVFinding(word);
                        if (rv != "")
                        {
                            {
                                r1 = R1Finding(word);
                                r2 = R1Finding(r1);
                                p.Step1(rv, ref word);
                                p.Step2(rv, ref word);
                                p.Step3(ref word, r2);
                                p.Step4(rv, ref word);
                            }

                            newHeadingsAfterStemming.Write(word + " ");
                            word = "";
                        }
                        i++;
                    }
                }
                newHeadingsAfterStemming.WriteLine("");
            }
        }
    }
}

```

Листинг 1.5. Метод для создания частотной матрицы

```

static void CreatingFrequencyMatrix()
{
    ArrayList multiplyWords = new ArrayList();
    ArrayList lines = new ArrayList();
    string line, tempWord;
    using (StreamReader WithoutStopWordHeaders = new
StreamReader("C:\\test\\output2.txt"))
    {
        while ((line = WithoutStopWordHeaders.ReadLine()) != null)
        {

```

```

        lines.Add(line);
    }
}
for (int i = 0; i < lines.Count; i++)
{
    string temp = (string)lines[i];
    int j = 0;
    string word = "";
    while (j < temp.Length)
    {
        if (temp[j] != ' ')
        {
            word += temp[j];
        }
        else
        {
            if ((word != " ") && (word != ""))
            {
                word += " ";
                tempWord = word;
                word = word.ToLower();
                string savingLine = (string)lines[i];
                temp = temp.ToLower();
                temp = temp.Replace(word, "");
                lines[i] = temp;
                for (int k = 0; k < lines.Count; k++)
                {
                    string searchString = (string)lines[k];
                    searchString = searchString.ToLower();
                    int pos = searchString.IndexOf(word);
                    if (pos != -1)
                    {
                        bool check = false;
                        for (int f = 0; f < multiplyWords.Count; f++)
                        {
                            string multiplyWordsToLower = (string)multiplyWords[f];
                            multiplyWordsToLower = multiplyWordsToLower.ToLower();
                            if (word == multiplyWordsToLower)
                            {
                                check = true;
                                break;
                            }
                        }
                        if (!check)
                        {
                            multiplyWords.Add(tempWord);
                        }
                    }
                }
                lines[i] = savingLine;
                temp = savingLine;
                word = "";
            }
        }
        j++;
    }
}
multiplyWords.Sort();
frequencyMatrix = new double[multiplyWords.Count, lines.Count];
m = multiplyWords.Count;
n = lines.Count;
for (int i = 0; i < multiplyWords.Count; i++)
{
    for (int j = 0; j < lines.Count; j++)
    {
        frequencyMatrix[i, j] = 0;
    }
}

```

```

    }
    for (int i = 0; i < multiplyWords.Count; i++)
    {
        for (int j = 0; j < lines.Count; j++)
        {
            tempWord = (string)multiplyWords[i];
            tempWord = tempWord.ToLower();
            line = (string)lines[j];
            line = line.ToLower();
            int pos = line.IndexOf(tempWord);
            if (pos != -1)
            {
                frequencyMatrix[i, j]++;
            }
        }
    }
}

```

Листинг 1.6. Реализация метода сингулярного разложения частотной матрицы

```

/*Use alglib.dll*/
static void SVD()
{
    alglib.rmatrixsvd(frequencyMatrix, m, n, 2, 2, 2, out w, out u, out vt);
    VT = new double[rowsCount, n];
    for (int i = 0; i < rowsCount; i++)
    {
        for (int j = 0; j < n; j++)
        {
            VT[i, j] = vt[i, j]; // 2dimension matrix, rowsCount eneter when app start
        }
    }
}

```

Листинг 1.7. Реализация метода кластеризации

```

/*Clusters*/
/*Zero step*/
static void InitializeCentroid(double[,] centroid)
{
    for (int i = 0; i < rowsCount; i++)
    {
        for (int j = 0; j < clustersCount; j++)
        {
            centroid[i, j] = VT[i, j];
        }
    }
}

/*First step*/
static void FindClusters(double[,] centroid1)
{
    for (int i = 0; i < clustersCount; i++)
        clusters[i].Clear();
    for (int i = 0; i < n; i++)
    {
        double[] distances = new double[clustersCount];
        for (int j = 0; j < clustersCount; j++)
        {
            distances[j] = 0;
            for (int k = 0; k < rowsCount; k++)
            {
                distances[j] += Math.Pow(centroid1[k, j] - VT[k, i], 2);
            }
            distances[j] = Math.Sqrt(distances[j]);
        }
        double min = distances.Min();
        int minIndex = Array.IndexOf(distances, min);
        clusters[minIndex].Add(i);
    }
}

```



```

    }
}
/*Second step*/
static void CopyCentroids(double[,] centroid1, double[,] centroid2)
{
    for (int i = 0; i < rowCount; i++)
    {
        for (int j = 0; j < clustersCount; j++)
        {
            centroid2[i, j] = centroid1[i, j];
        }
    }
}

/*Third step*/
static void CreateNewCentroid(double[,] centroid1)
{
    for (int i = 0; i < clustersCount; i++)
    {
        foreach (int c in clusters[i])
        {
            for (var j = 0; j < rowCount; j++)
            {
                centroid1[j, i] += VT[j, c];
            }
        }
        for (var j = 0; j < rowCount; j++)
        {
            if (clusters[i].Count != 0)
            {
                centroid1[j, i] /= clusters[i].Count;
            }
        }
    }
}

/*If >eps*/
static double MaxChanging(double[,] centroid1, double[,] centroid2)
{
    double[] changing = new double[clustersCount];
    for (int i = 0; i < clustersCount; i++)
    {
        for (int j = 0; j < rowCount; j++)
        {
            changing[i] += Math.Pow(centroid1[j, i] - centroid2[j, i], 2);
        }

        changing[i] = Math.Sqrt(changing[i]);
    }
    return changing.Max();
}

static void Clustering()
{
    double[,] centroid1 = new double[rowCount, clustersCount];
    double[,] centroid2 = new double[rowCount, clustersCount];
    InitializeCentroid(centroid1);
    clusters = new ArrayList[clustersCount];
    for (int i = 0; i < clustersCount; i++)
    {
        clusters[i] = new ArrayList();
    }
    FindClusters(centroid1);
    CopyCentroids(centroid1, centroid2);
    CreateNewCentroid(centroid1);
    while (MaxChanging(centroid1, centroid2) > eps)
    {
        FindClusters(centroid1);
    }
}

```

```

        CopyCentroids(centroid1, centroid2);
        CreateNewCentroid(centroid1);
    }
}

```

Листинг 1.8. Реализация метода вывода

```

static void Output()
{
    using (StreamWriter result = new StreamWriter("C:\\test\\result.txt"))
    {
        for (int i = 0; i < clustersCount; i++)
        {
            result.WriteLine("Область {0}", i + 1);
            result.WriteLine("-----", i + 1);
            foreach (int doc in clusters[i])
            {
                result.WriteLine(HEADERS[doc]);
            }
            result.WriteLine("-----");
            result.WriteLine();
        }
    }
}

```

2. Тестирование:

Британская полиция знает о местонахождении основателя

WikiLeaks

**В суде США начинается процесс против россиянина,
рассылавшего спам**

**Церемонию вручения Нобелевской премии мира
бойкотируют 19 стран**

В Великобритании арестован основатель сайта Wikileaks

Джулиан Ассандж

**Украина игнорирует церемонию вручения Нобелевской
премии**

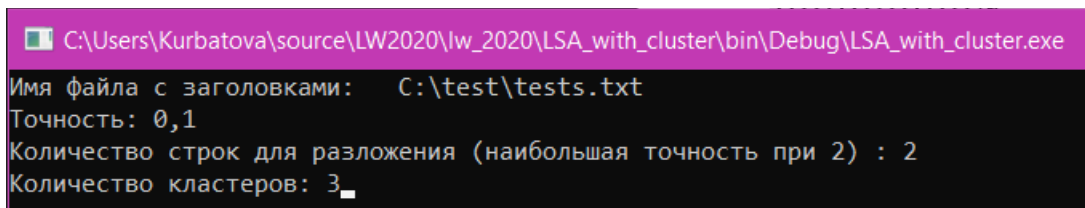
**Шведский суд отказался рассматривать апелляцию
основателя Wikileaks**

**НАТО и США разработали планы обороны стран Балтии
против России**

**Полиция Великобритании нашла основателя WikiLeaks, но,
не арестовала**

**В Стокгольме и Осло сегодня состоится вручение
Нобелевских премий**

Рис. 3.1. Файлы с заголовками



```

C:\Users\Kurbatova\source\LW2020\lw_2020\LSA_with_cluster\bin\Debug\LSA_with_cluster.exe
Имя файла с заголовками: C:\test\tests.txt
Точность: 0,1
Количество строк для разложения (наибольшая точность при 2) : 2
Количество кластеров: 3_

```

Рис. 3.2. Ввод данных

Британск полиц знает местонахожден основател
 суд США начинает процесс прот россиянин рассыл
 Церемон вручен Нобелевск прем мир бойкотир
 Великобритан арестов основател сайт Wikileaks Джул
 Украин игнорир церемон вручен Нобелевск
 Шведск суд отказал рассматрив апелляц основател
 НАТО США разработ план оборон стран Балт прот
 Полиц Великобритан наш основател WikiLeaks
 Стокгольм Осл состоит вручен Нобелевск

Рис. 3.3. Файл после удаления стоп-слов и стемминга

Область 1

Британская полиция знает о местонахождении основателя WikiLeaks
 В Великобритании арестован основатель сайта Wikileaks Джулиан Ассандж
 Шведский суд отказался рассматривать апелляцию основателя Wikileaks
 Полиция Великобритании нашла основателя WikiLeaks, но, не арестовала

Область 2

В суде США начинается процесс против россиянина, рассылавшего спам
 НАТО и США разработали планы обороны стран Балтии против России

Область 3

Церемонию вручения Нобелевской премии мира бойкотируют 19 стран
 Украина игнорирует церемонию вручения Нобелевской премии
 В Стокгольме и Осло сегодня состоится вручение Нобелевских премий

Рис. 3.4. Файл с результатом

Вывод: Таким образом, в ходе выполнения лабораторной работы на языке C# были реализованы алгоритм латентно-семантического поиска.