Глава 2. Кластеризация профилей пользователей

1.1 Метод К-Средних

Метод К-Средних – это метод кластерного анализа, цель которого разделение конечного множества объектов, на заранее определенное количество кластеров (неявных классов), по определенным характеристикам объектов. Каждый объект относится к тому кластеру, к центру которого ближе. Для понимания возьмем профили двух пользователей.

	A	B	C
1	Name	Music	Games
2	Arstan	10	200
3	Andrey	20	150

Вычислим для них расстояние между друг от друга различными методами вычисления расстояний.

Расстояние Евклида: $S = \sqrt[2]{((10-200)^2 + (20-150)^2)} \sim 230.2$

Квадрат Евклидового расстояния: $S = (10-200)^2(20-150)^2 = 53\,000$

Расстояние Чебышева: $S = MAX(|10-200|, |20-150|) = 190$

Манхэттенское расстояние: $S = |10-200| + |20-150| = 320$

Как видно из примеров, полученные расстояния, из-за вычислений сумм и квадратов, выходят очень громоздкими. Арифметические выражения над большими числами затрачивают очень много ресурсов и времени работы ЭВМ.

Рассмотрим для примера вычисление расстояний после нормализации входных данных

	A	B	C
1	Name	Music	Games
2	Arstan	-0,05263	0,263158
3	Andrey	0,052632	-0,26316
4			

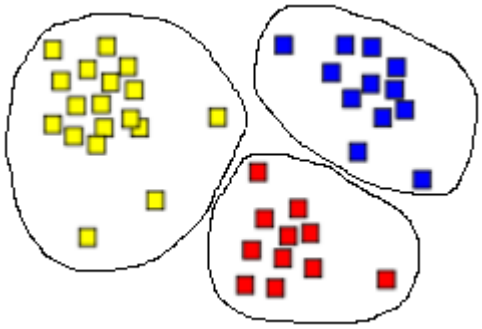
Расстояние Евклида: S =0,536741

Квадрат Евклидового расстояния: S = 0,288091

Расстояние Чебышева: S = 0,526318

Манхэттенское расстояние: S = 0,105262

Как видно



На этом примере показан пример разделения объектов на 3 кластера.

2.2 Иерархический метод кластеризации

Глава 3. Разработка проекта

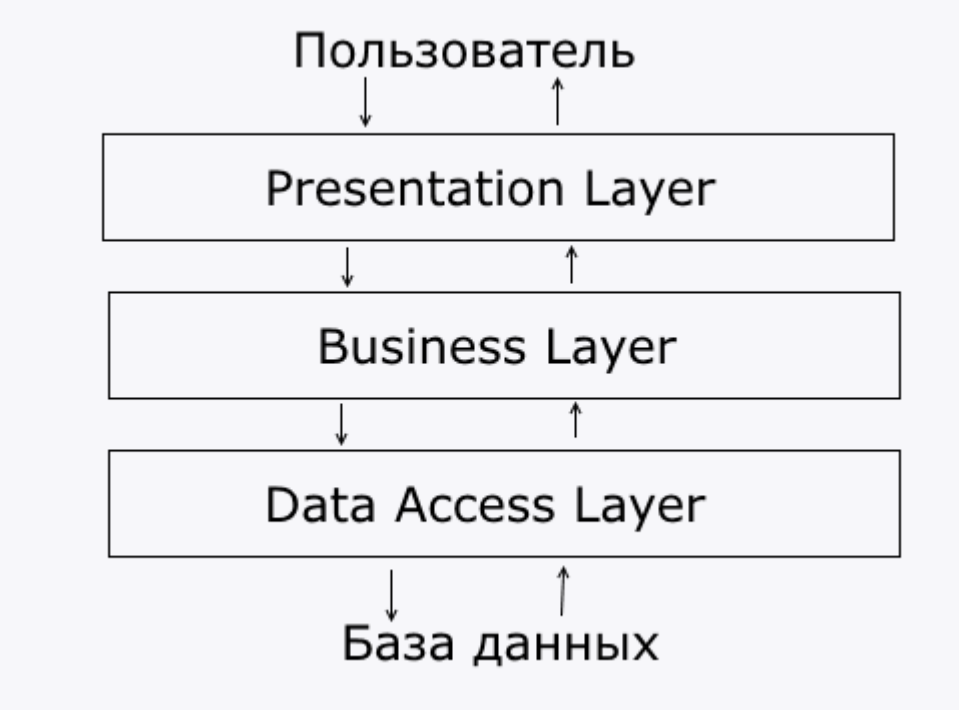
3.1 Структура программы

В программной инженерии **многоуровневая архитектура** или **многослойная архитектура** — клиент-серверная архитектура, в которой разделяются функции представления, обработки и хранения данных. Наиболее распространённой разновидностью многоуровневой архитектуры является **трёхуровневая архитектура**.

N-уровневая архитектура приложения предоставляет модель, по которой разработчики могут создавать гибкие и повторно-используемые приложения. Разделяя приложение на уровни абстракции, разработчики приобретают возможность внесения изменений в какой-то определённый слой, вместо того, чтобы перерабатывать всё приложение целиком. Трёхуровневая архитектура обычно состоит из уровня *представления*, уровня *бизнес логики* и уровня *хранения данных*.

Хотя понятия слоя и уровня зачастую используются как взаимозаменяемые, многие сходятся во мнении, что между ними всё-таки есть различие. Различие заключается в том, что *слой* — это механизм логического структурирования компонентов, из которых состоит программное решение, в то время как *уровень* — это механизм физического структурирования инфраструктуры системы. Трёхуровневое решение легко может быть развёрнуто на единственном уровне, таком как персональная рабочая станция.

В система реализована трехуровневая архитектура. Это архитектурная модель программного обеспечения, разделяющая программу на три компоненты: клиент, сервер приложения и сервер базы данных.



Presentation Layer - уровень клиентского интерфейса, интерфейс пользователя, в виде веб-сайта. В этой части отсутствует доступ к БД. На этом уровне реализована общение пользователя с программой, такие как ввод данных, выбор алгоритма и сопутствующих к нему параметров, а также вывод в удобном для пользователя формате. Клиентская часть имеет связь только с уровнем бизнес логики. Работа клиентской части и серверной части осуществляется при помощи контроллеров, которые получают запросы от пользователя и отправляют на уровень бизнес логики.

Business Layer - уровень бизнес-логики, основная рабочая часть программы. На этом уровне происходят все основные действия программы такие как:

1. Обработка выходных данных
2. Считывание файлов
3. Реализация алгоритмов вычисления расстояний между объектами
4. Реализация алгоритмов вычисления расстояний между кластерами
5. Реализация алгоритмов кластерного анализа
6. Построение объектов для последующей передачи их клиентской части, для отображения пользователю
7. Генерация данных для работы программы

Серверная часть реализована в виде сервисов, которые исполняют работу программы, получая запросы пользователя из клиентского интерфейса. Серверная часть имеет связь с клиентской частью и базой данных. Связь с уровнем базы данных реализована с использованием паттерна “Repository”. Repository pattern – это слой абстракции, в котором инкапсулируются все что относится к работе с БД.

Data Access Layer - уровень базы данных, на этом уровне хранятся данные о БД, класс обеспечивающий связь с СУБД, данные о миграциях, и реализации репозиториев. Работа с СУБД, реализована при помощи “Entity Framework”. Entity Framework – это объектно-ориентированная технология доступа к данным, является ORM (object-relational mapping) решением для .NET Framework.

Исполнение логической части реализована в виде сервисов. Для сервисов создается интерфейс и его реализация. Для ослабления связей между классами применен паттерн проектирования **Dependency Injection(DI).**

Dependency Injection (DI) - это паттерн проектирования ПО, который позволяет разрабатывать слабосвязанный код. При помощи DI, уменьшается жесткость связи между компонентами программы. На языке C# существует три способа внедрения зависимостей.

1. Через конструктор.
2. Через параметр метода, к которому применяется атрибут FromServices
3. Через свойство HttpContext.RequestServices

В этой системе в основном используется внедрение зависимости через конструктор. Когда приходит запрос к контроллеру, инфраструктура MVC вызывает провайдер сервисов для создания объекта какого-либо класса. Провайдер сервисов проверяет конструктор этого класса на наличие зависимостей. Затем создает объекты для всех используемых зависимостей и передает их в конструктор класса для создания объекта.

Работа фреймворка, обеспечивающая внедрение зависимости, описывается следующим образом. Приложение, независимо от оформления, исполняется внутри контейнера IoC, предоставляемого фреймворком. Часть объектов в программе по-прежнему создается обычным способом языка программирования, часть создается контейнером на основе предоставленной ему конфигурации.

Условно, если объекту нужно получить доступ к определенному сервису, объект берет на себя ответственность за доступ к этому сервису: он или получает прямую ссылку на местонахождение сервиса, или обращается к известному «сервис-локатору» и запрашивает ссылку на реализацию

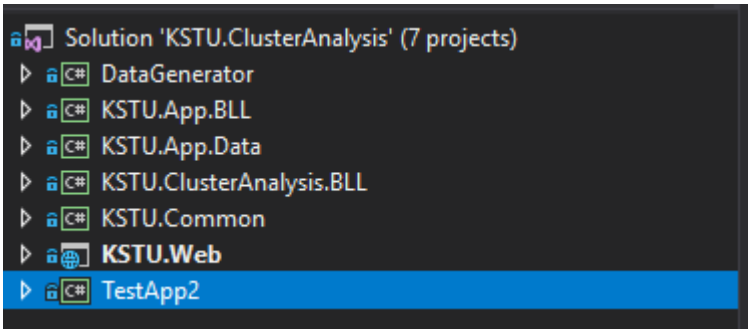
определенного типа сервиса. Используя же внедрение зависимости, объект просто предоставляет свойство, которое в состоянии хранить ссылку на нужный тип сервиса; и когда объект создается, ссылка на реализацию нужного типа сервиса автоматически вставляется в это свойство (поле), используя средства среды.

Внедрение зависимости более гибко, потому что становится легче создавать альтернативные реализации данного типа сервиса, а потом указывать, какая именно реализация должна быть использована в, например, конфигурационном файле, без изменений в объектах, которые этот сервис используют. Это особенно полезно в юнит-тестировании, потому что вставить реализацию «заглушки» сервиса в тестируемый объект очень просто.

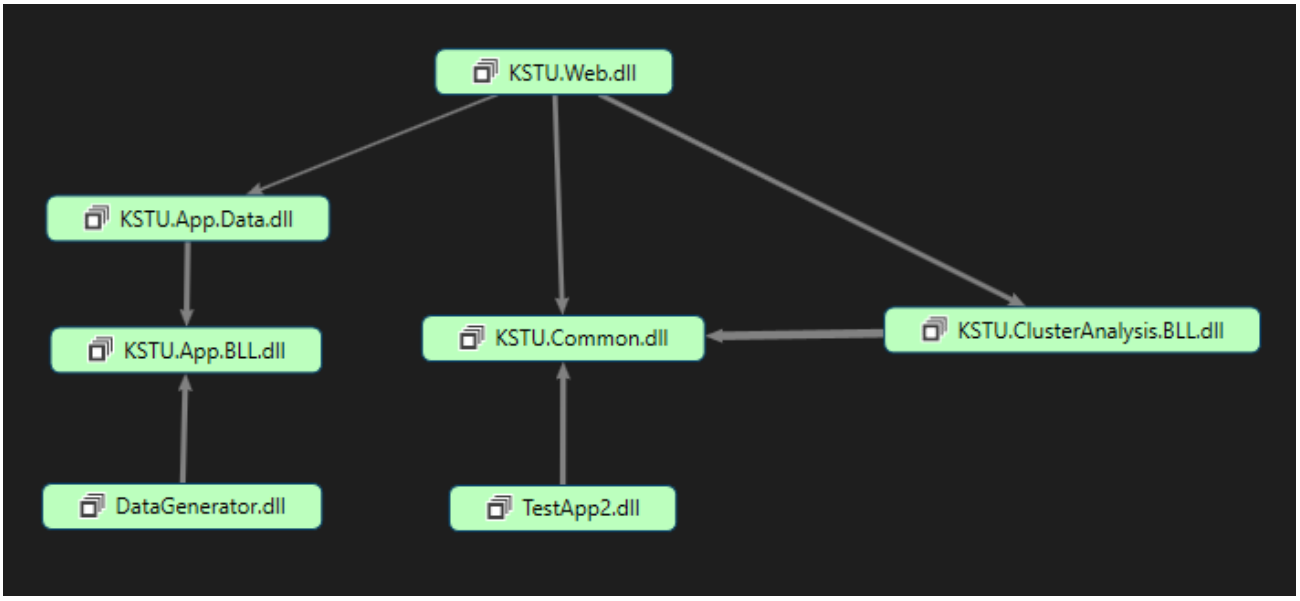
С другой стороны, излишнее использование внедрения зависимостей может сделать приложения более сложными и трудными в сопровождении: так как для понимания поведения программы программисту необходимо смотреть не только в исходный код, а еще и в конфигурацию, а конфигурация, как правило, невидима для IDE, которые поддерживают анализ ссылок и рефакторинг, если явно не указана поддержка фреймворков с внедрениями зависимостей.

3.2 Описание модулей программы

Программа состоит из 7 модулей:



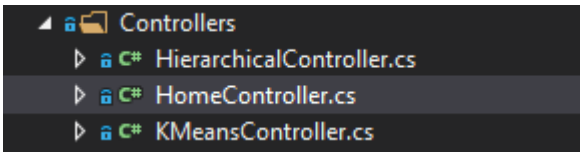
Связи между модулями.



1. Модуль KSTU.Web – модуль веб-приложения. Основной модуль сайта.
2. Модуль KSTU.Common – библиотека классов. Где находятся основные компоненты программы, например, обработчик файлов.
3. Модуль KSTU.ClusterAnalysis.BLL – модуль где реализованы методы кластерного анализа.
4. Модуль KSTU.App.Data – модуль где реализована связь с БД, репозитории и данные о миграциях.
5. Модуль KSTU.App.BLL – модуль где находятся основные сущности БД, классы пользователь и интересы.
6. Модуль DataGenerator – модуль для генерации данных для кластерного анализа.
7. Модуль TestApp – модуль для тестирования алгоритмов.

Модуль KSTU.Web – модуль веб приложения. Веб-приложение создано с использованием паттерна проектирования MVC(Model-View-Controller).

Контроллер – управляет запросами пользователя, в виде HttpGet или HttpPost, когда пользователь переходит по ссылкам или же заполняет форму и отправляет данные. Контроллер, в зависимости от действий пользователя вызывает и координирует действия необходимых ресурсов и объектов.



HomeController – это контроллер по умолчанию. Когда пользователь переходит на сайт по ссылке, его запрос приходит в этот контроллер, и он выдает главную страницу.

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}
```

Index – выдает пользователю главную страницу.

Error – при какой либо ошибке выдает страницу ошибки.

HomeController – является базовым контроллером, необходимым для работы веб-приложения.

KMeansController – контроллер который отвечает за страницы относящиеся к реализации метода К-Средних.

```
public class KMeansController : Controller
{
    private readonly IKMeans _kMeans;
    private readonly IFileService _fileService;

    public KMeansController(IKMeans kMeans, IFileService fileService)
    {
        _kMeans = kMeans;
        _fileService = fileService;
    }

    public IActionResult Index()
    {
        return View();
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public IActionResult ShowResult(KMeansCreateVM model)
    {
        var data = _fileService.GetData(model.UploadFile, model.DataType);
        IDistance distance = DistanceFactory.GetDistance(model.DistanceType);
        var result = _kMeans.Clustering(data, distance, model.ClustersCount);
        return View(result);
    }
}
```

Index – возвращает начальную страницу.

Create – возвращает страницу с формой для заполнения данных.

При отправке данных с формы, поступает пост запрос на метод **ShowResult**. Этот метод сначала вызывает метод обработки данных сервиса “FileService”, который считывает данные с Excel или текстового документа и возвращает в виде списка объектов, для последующей кластеризации. Затем создается экземпляр интерфейса IDistance, при помощи статического класса “DistanceFactory” и его метода “GetDistance”, который в зависимости от выбранного пользователем меры расстояния выдает реализацию интерфейса IDistance. После чего вызывается метод интерфейса IKmeans, отвечающий за кластеризацию. Затем результат кластеризации отправляется на страницу и выстраивается диаграмма и таблица обработанных данных.

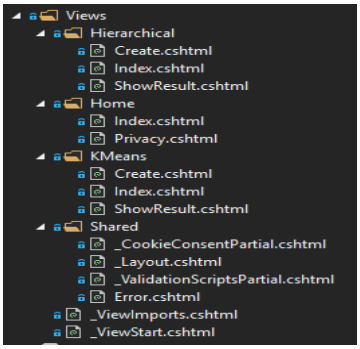
HierarchicalController – контроллер который отвечает за реализацию иерархического метода кластеризации. В этом контроллере те же методы что и в предыдущем, различаются лишь сервис, который используется для кластеризации и метод “ShowResult”

```
public IActionResult ShowResult(HierarchicalCreateVM model)
{
    var data = _fileService.GetData(model.UploadFile, model.DataType);
    IDistance distance = DistanceFactory.GetDistance(model.DistanceType);
    IClusterDistance clusterDistance = ClusterDistanceFactory.GetClusterDistance(model.ClusterUnionType);
    var result = _service.Clustering(data, distance, clusterDistance, model.CountOfUnionsInStep);
    return View(result);
}
```

Для иерархической кластеризации добавляется метод определения расстояний между кластерами. ClusterDistanceFactory – в зависимости от выбранного пользователем метода меры расстояния между кластерами выдает реализацию интерфейса “IClusterDistance”.

Представления (View) – это файлы с расширением “cshtml (Razor-page)”, в которых содержится код пользовательского интерфейса, в основном на языке html. Представления содержат, главным образом, код html, но по сути не является html-страницей. При компиляции приложения на основе представления генерируется класс на языке C#, затем этот класс компилируется. Файл с расширением cshtml имеет свой синтаксис, похожий на обычный синтаксис html-страниц, с некоторыми отличиями. В представлениях Razor, можно производить вставки кода на языке C#. Вес код на языке Razor компилируется в C# класс, затем при запросе пользователя генерируется html-страница.

Папка Views, которой хранятся все представления:



Рассмотрим основные представления:

Представление _Layout.cshtml – мастер страница для всего проекта. Мастер-страницы используются для создания единобразного, унифицированного вида сайта. Тут задается основной вид всех страниц сайта, подключаются все необходимые библиотеки, которые нужны на всех страницах сайта (Bootstrap, Jquery, Datatable и т.д.). Мастер страница можно сказать своего рода каркас или оболочка, для остальных страниц сайта. Это дает удобство при написании других страниц сайта, т.к. нет необходимости писать шаблонный код множество раз, ведь мастер страница уже включает в себя эти шаблоны.

Основная структура мастер страницы:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Главная", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
</body>
</html>
```

Подключение необходимых библиотек:

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - KSTU.Web</title>
    <link rel="stylesheet" href="~/css/site.css" />
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="~/lib/datatables/media/css/dataTables.bootstrap4.min.css" />
    <link rel="stylesheet" href="~/lib/datatables/media/css/dataTables.checkboxes.css" />
    <link rel="stylesheet" href="~/lib/select2/dist/css/select2.min.css" />
    <link rel="stylesheet" href="~/lib/select2/dist/css/select2-bootstrap.min.css" />
    <link href="~/lib/Font-Awesome/css/all.css" rel="stylesheet" />
    <link rel="stylesheet" href="~/css/site.css" />

    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/jquery/dist/jquery.mask.min.js"></script>

    <script src="~/lib/datatables/media/js/jquery.dataTables.min.js"></script>
    <script src="~/lib/datatables/media/js/dataTables.bootstrap4.min.js"></script>
    <script src="~/lib/datatables/media/js/dataTables.checkboxes.min.js"></script>
    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
    <script src="~/lib/jquery.unobtrusive-ajax/src/jquery.unobtrusive-ajax.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.min.js"></script>
    <script src="~/lib/select2/dist/js/select2.full.min.js"></script>
    <script src="~/lib/select2/dist/js/i18n/ru.js"></script>
</head>
```

Основное меню страницы(сверху):

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-controller="KMeans" asp-action="Index">K-Means</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-controller="Hierarchical" asp-action="Index">Hierarchical method</a>
        </li>
    </ul>
</div>
```

Представление ShowResult метода К-Средних – это страница вывода результата кластеризации методом К-Средних. Результат выводится в виде 3D диаграммы и таблицы отношений объектов к кластерам, количество кластеров указывается при вводе данных. Заполнение таблицы происходит при помощи вставки кода на языке C#, данными полученными от контроллера.

```
<table class="table table-striped table-bordered" id="myTable" style="width:100%">
    @{
        <thead>
            <tr>
                <th>Номер кластера</th>
                <th>Название</th>
                @foreach (var item in Model[0].Interests)
                {
                    <th>@item.Name</th>
                }
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@(item.CentroidId + 1)</td>
                    <td>@item.Name</td>
                    @{
                        foreach (var interes in item.Interests)
                        {
                            <td>
                                @interes.Weight2
                            </td>
                        }
                    }
                </tr>
            }
        </tbody>
    }
</table>
```

Затем изменяется внешний вид таблицы с использованием библиотеки “DataTable”, а так же добавляется функции сортировки по столбцам, поиск элементов, и возможность вывода определенного количества данных в таблице.

```
<script>
    $(document).ready(function () {
        $('#myTable').DataTable({
            "language": {
                "url": '/lib/datatables/Russian.json'
            }
        });
    });
</script>
```

Для построения диаграммы используется библиотека “Google Chart Tools API”. Это многофункциональный набор инструментов для визуализации данных.

```
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">

   google.charts.load('current', { 'packages': ['corechart'] });

   google.charts.setOnLoadCallback(drawChart);

   function drawChart() {

      var data = new google.visualization.DataTable();
      data.addColumn('string', 'Topping');
      data.addColumn('number', 'Slices');

      var model = @Html.Raw(Json.Serialize(Model.GroupBy(g => g.CentroidId).Select(s => new { Key = s.Key + 1, Count = s.Count() })));
      for (var i = 0; i < model.length; i++) {
         var obj = model[i];
         data.addRows([['Кластер '+obj.key,obj.count]])
         console.log(obj);
      }

      var options = {
         'title': 'Диаграмма результата кластеризации',
         'width': 400,
         'height': 300,
         is3D: true
      };

      var chart = new google.visualization.PieChart(document.getElementById('chart_div'));
      chart.draw(data, options);
   }
</script>
```

Данные для диаграммы конвертируем в формат Json, с помощью Html-Helper'а, затем из нее выстраиваем диаграмму с помощью инструментов библиотеки. После выполнения всех необходимых операции по построению диаграммы, результат записывается в определенную на странице DIV'ку.

```
<div id="chart_div"></div>
```

Представление ShowResult иерархического метода – это страница вывода результата кластеризации иерархического метода. Результат выводится в виде дендрограммы(дерева).

Для построения дендрограммы используется библиотека "D3". D3 это набор инструментов для визуализации данных. Он состоит из нескольких десятков небольших модулей, каждый из которых решает свою задачу. Кроме модулей для построения различных фигур, внутри D3 есть модули для работы с элементами на странице (простой аналог jQuery), загрузкой данных (аналог fetch/$.ajax, заточенный под форматы csv, json, xml и другие), форматированием и масштабированием данных, математическими функциями и другим.

Код javascript, для построения дендрограммы:

```
<script>
    var root =@Html.Raw(Json.Serialize(Model));
    var width = 1000,
        height = 700;

    var cluster = d3.layout.cluster()
        .size([height, width - 160]);

    var diagonal = d3.svg.diagonal()
        .projection(function (d) { return [d.y, d.x]; });

    var svg = d3.select("#MyDiv").append("svg")
        .attr("width", width)
        .attr("height", height)
        .append("g")
        .attr("transform", "translate(40,0)");

    var nodes = cluster.nodes(root),
        links = cluster.links(nodes);

    var link = svg.selectAll(".link")
        .data(links)
        .enter().append("path")
        .attr("class", "link")
        .attr("d", diagonal);

    var node = svg.selectAll(".node")
        .data(nodes)
        .enter().append("g")
        .attr("class", "node")
        .attr("transform", function (d) { return "translate(" + d.y + "," + d.x + ")"; });

    node.append("circle")
        .attr("r", 4.5);

    node.append("text")
        .attr("dx", function (d) { return d.children ? -8 : 8; })
        .attr("dy", 3)
        .style("text-anchor", function (d) { return d.children ? "end" : "start"; })
        .text(function (d) { return d.name; });

    d3.select(self.frameElement).style("height", height + "px");
</script>
```

Стили дендрограммы задаются отдельно от библиотеки:

```
.node circle {
    fill: #fff;
    stroke: steelblue;
    stroke-width: 1.5px;
}

.node {
    font: 10px sans-serif;
}

.link {
    fill: none;
    stroke: #ccc;
    stroke-width: 1.5px;
}
```

Модуль KSTU.Common – это библиотека основных классов проекта. Тут хранятся основные классы, используемые в проекте:

1. Классы, отвечающие за обработку текстовых файлов и файлов Excel.
2. Классы, отвечающие за вычисление расстояний между кластерами и между объектами.
3. Классы описывающие основные сущности проекта, используемые при кластеризации.
4. Перечисления (Enum), методов вычисления расстояний между объектами и кластерами, так же типов вводимых данных для кластеризации.

Интерфейсы в модуле Common:

1. **IFileService** – метод считывания данных с файлов.
2. **IDistance** – метод вычисления расстояния между объектами.
3. **IClusterDistance** – метод вычисления расстояний между кластерами.

FileService – класс отвечающий за обработку файлов. Реализует интерфейс **IFileService**, в котором определен один метод GetData. Метод в зависимости от входных параметров вызывает методы, отвечающие за обработку текстового файла или Excel файла.

```
public List<ClusterEntityDTO> GetData(IFormFile file, byte dataType)
{
    switch (dataType)
    {
        case (byte)EnumDataType.Text:
            return GetDataFromTxt(file);
        case (byte)EnumDataType.Excel:
            return GetDataFromExcel(file);
    }
    return new List<ClusterEntityDTO>();
}
```

Метод GetData принимает два аргумента, это файл, присылаемый пользователем и байтовое значение типа файла, байтовое значение берется из Енумератора “EnumDataType”. Затем метод, в зависимости от значения datatype, вызывает методы
GetDataFromTxt (обработка текстового файла) или GetDataFromExcel(обработка Excel файла).

Метод GetDataFromTxt – метод считывания и обработки данных с текстового файла. Этот метод принимает один аргумент, файл введенный пользователем с формы. Для считывания данных используется класс StreamReader. Считанные данные записываются в массив строк. Затем обрабатываются и создаются объекты класса ClusterEntityDTO. Затем обработанные данные возвращаются в виде списка объектов.

Метод GetDataFromExcel – метод идентичен предыдущему, с некоторыми различиями. Этот метод считывает данные с Excel файла, при помощи библиотеки ClosedXml.

IDistance – интерфейс имеющий один метод GetDistance. Принимает два объекта и возвращает численную величину расстояния между этими объектами. Имеет четыре реализации:

1. ChebyshevDistance – вычисляет расстояние между объектами по формуле Чебышева.
2. EuclideanDistance – вычисляет расстояние между объектами по формуле Евклида
3. EuclideanSquareDistance – квадрат Евклидового расстояния
4. ManhattanDistance – манхэттенское расстояние (расстояние городских кварталов)

Пример (класс реализации Евклидового расстояния):

```
public class EuclideanDistance : IDistance
{
    public double GetDinstance(ClusterEntityDTO first, ClusterEntityDTO second)
    {
        double distance = 0.0;

        for (int i = 0; i < first.Interests.Count; i++)
        {
            double diff = first.Interests[i].Weight - second.Interests[i].Weight;
            distance += (diff * diff);
        }

        return Math.Sqrt(distance);
    }
}
```

Класс **DistanceFactory –** имеет один метод, который в зависимости от выбранного пользователем формулы вычисления расстояний между объектами выдает определенную реализацию интерфейса IDistance. Имеет один метод GetDistance, который принимает байтовую переменную значения типа расстояний.

```
public static IDistance GetDistance(byte type)
{
    switch (type)
    {
        case (byte)EnumDistanceTypes.Chebyshev:
            return new ChebyshevDistance();
        case (byte)EnumDistanceTypes.EuclideanSquare:
            return new EuclideanDistance();
        case (byte)EnumDistanceTypes.Manhattan:
            return new ManhattanDistance();
        default:
            return new EuclideanDistance();
    }
}
```

IClusterDistance – интерфейс для вычисления расстояний между кластерами. Реализация схожа с IDistance, только входящие аргументы отличаются. Метод принимает полный кластер, список объектов кластера, затем различными методами вычисляет расстояние между кластерами. Интерфейс имеет четыре реализации:

1. Класс FullConnection – метод полной связи, расстояние определяется наибольшим расстоянием между любыми двумя объектами в различных кластерах.
2. Класс SingleConnection – метод одиночной связи, расстояние определяется попарно наименьшим расстоянием между всеми объектами двух кластеров.
3. Класс UnweightedCentroid – невзвешенный центроидный метод, расстояние между кластерами определятся как расстояние между центрами тяжести двух кластеров.
4. Класс UnweightedPairwiseMean – невзвешенное попарное среднее, расстояние между двумя кластерами вычисляется как среднее расстояние между всеми парами объектов каждого кластера.

Класс ClusterDistanceFactor – статический класс, который выдает реализации интерфейса IClusterDistance, в зависимости от выбранного метода вычисления расстояния между кластерами.

Модуль KSTU.ClusterAnalysis.BLL – модуль где реализованы методы кластерного анализа. Методы реализованы в виде интерфейсов и их реализации, для использования технологии Dependency Injection (Внедрение Зависимостей). Зависимости внедряются в главном классе веб-приложения (Startup.cs).

Метод К-Средних – для этого метода используется интерфейс IKmeans, который реализован в классе KMeans. Интерфейс имеет один метод “Clustering”, который имеет три входных параметра.

```
public interface IKMeans
{
    List<ClusterEntityDTO> Clustering(List<ClusterEntityDTO> clusters, IDistance distance, int clustersCount);
}
```

Где “clusters” – список сформированных объектов для кластеризации, “distance” – реализация интерфейса IDistance выбранный пользователем, “clustersCount” – введенное пользователем число кластеров, на которые надо разделить входные данные. Метод возвращает список кластеризованных объектов в виде списка.

Класс **KMeans –** реализация интерфейса IKmeans, в котором реализована работа алгоритма К-Средних. Класс имеет шесть методов:

1. **Метод Clustering –** реализация метода используемого интерфейса, основной метод в котором вызываются остальные.
2. **Метод Normalize –** метод для нормализации входных данных. Нормализация данных это процесс, в котором все входные данные “выравниваются”, приводятся к определенному интервалу, чтобы избежать большой разницы между различными входными данными.
3. **Метод SetCentroids –** метод который задает в случайном порядке *K*, случайных центров для последующей кластеризации.
4. **Метод UpdateClusters –** в этом методе каждый объект сопоставляется с определенным кластером в зависимости от расстояния к центру кластера.
5. **Метод UpdateCentroids –** метод в котором обновляются центры тяжести всех кластеров. Центры тяжести вычисляются как средние значения весов всех объектов, принадлежащих этому кластеру.
6. **Метод GetIndexOfMin –** метод который возвращает индекс минимального элемента массива, используется в методе UpdateClusters.

Основной метод реализации метода К-Средних:

```
public List<ClusterEntityDTO> Clustering(List<ClusterEntityDTO> data, IDistance distance, int clustersCount)
{
    _distance = distance;
    List<ClusterEntityDTO> clusters = Normlize(data);

    bool changed = true;
    bool success = true;

    List<ClusterEntityDTO> centroids = SetCentroids(ref data, clustersCount);
    int maxIter = clusters.Count * 10;
    int iterCount = 0;
    while (changed && iterCount < maxIter)
    {
        changed = UpdateClusters(ref clusters, ref centroids);
        success = UpdateCentroids(ref clusters, ref centroids);
        iterCount++;
    }

    return clusters.AsEnumerable().OrderBy(g => g.CentroidId).ToList();
}
```

Метод Clustering

Алгоритм:

1. Сначала входные данные нормализуются при помощи метода Normalize.
2. Задаются начальные центральные точки K - кластеров.
3. Объекты сопоставляются с кластерами, в зависимости от расстояния между центром кластера и самого объекта методом UpdateClusters.
4. Обновляются центры тяжести каждого кластера методом UpdateCentroids.
5. Возвращаться к п.3 пока при обновлении центров тяжести и сопоставлении кластеров центрам не будет изменении или итерация достигнет заданного количества (после определенного количества итерации изменения станут минимальными).

Метод Normalize

Алгоритм:

1. Вычисление среднего значения, среди численных значений определенного интереса. $mean = \frac{\sum_i^n a_i}{n}$
2. Вычисление суммы квадратов разностей веса текущего интереса и среднего значения. $sd = \frac{\sum_i^n (a_i - mean)^2}{n}$
3. Переназначить веса по текущему интересу для всех объектов. $a_i = \frac{a_i}{sd}$
4. Выполнить п.1-3 для всех имеющихся интересов в входных данных.

Метод SetCentroids

Алгоритм:

1. Взять случайный объект из входных данных (не включая уже имеющихся) и добавить в массив центроидов.

2. Повторять п. 1 пока количество центроидов не станет равным указанному.

Метод UpdateClusers

Алгоритм:

1. Найти для текущего объекта ближайший кластер и добавить к нему этот объект.
2. Повторять п.1 пока все объекты не будут присвоены к какому-либо кластеру.
3. Высчитать количество объектов в каждом кластере.

Метод UpdateCentroids

Алгоритм:

1. Очистить веса интересов центров масс.
2. Для каждого интереса текущего кластера вычислить среднее значение между всеми объектами этого кластера $mean_i = \frac{\sum_j^n a_j}{n}$

Иерархический метод – для этого метода используется интерфейс IHierarchical, который реализован в классе Hierarchical. Интерфейс имеет один метод “Clustering”, который имеет четыре входных параметра.

```
public interface IHierarchical
{
    ClusterEntityDTO Clustering(List<ClusterEntityDTO> data, IDistance distance, IClusterDistance clusterDistance, int maxUnionInStep);
}
```

Где “data” – список сформированных объектов для кластеризации, “distance” – реализация интерфейса IDistance выбранный пользователем, “clusterDistance” – реализация интерфейса IClusterDistance выбранный пользователем, “maxUnionInStep” – введенное пользователем количество объединений на каждом итерации метода. Метод возвращает список кластеризованных объектов в виде списка.

Класс **Hierarchical –** реализация интерфейса **IHierarchical**, в котором реализована работа иерархического алгоритма. Класс имеет три метода:

1. **Clustering –** реализация метода используемого интерфейса, основной метод в котором вызываются остальные.
2. **Метод Normalize –** метод для нормализации входных данных. Нормализация данных это процесс, в котором все входные данные

“выравниваются”, приводятся к определенному интервалу, чтобы избежать большой разницы между различными входными данными.

3. **Метод Union** – метод который производит, определенное пользователем, количество объединений кластеров с минимальными расстояниями между собой.
4. **Метод GetName** – метод который выдает название для кластера, полученным путем объединения двух кластеров.

Основной метод реализации иерархической кластеризации.

```
public ClusterEntityDTO Clustering(List<ClusterEntityDTO> clusteringData, IDistance distance, IClusterDistance clusterDistance, int maxUnionInStep)
{
    _distance = distance;
    _clusterDistance = clusterDistance;
    var data = Normlize(clusteringData);
    int step = 0;
    while (true)
    {
        data = Union(data, maxUnionInStep, ++step);
        if (data.Count == 1)
            break;
    }
    data[0].Name = "Root";
    return data[0];
}
```

Метод Clustering

Алгоритм:

1. Сначала входные данные нормализуются при помощи метода Normalize.
2. Выполняется *K* объединений, при помощи метода Union
3. Выполнять п. 2 пока количество кластеров не станет равным одному.

Метод Normalize

Алгоритм:

5. Вычисление среднего значения, среди численных значений определенного интереса. $mean = \frac{\sum_i^n a_i}{n}$
6. Вычисление суммы квадратов разностей веса текущего интереса и среднего значения. $sd = \frac{\sum_i^n (a_i - mean)^2}{n}$
7. Переназначить веса по текущему интересу для всех объектов. $a_i = \frac{a_i}{sd}$
8. Выполнить п.1-3 для всех имеющихся интересов в входных данных.

Метод Union

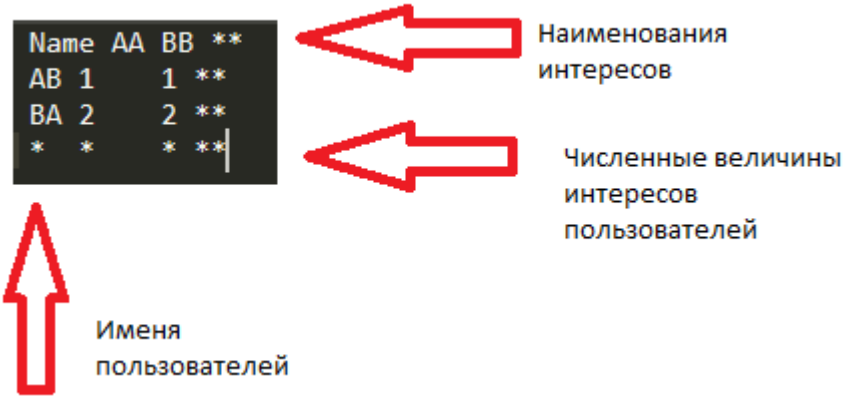
Алгоритм:

1. Создать пустой список кластеров, который будет возвращен.
2. Вычислить попарное расстояние между всеми кластерами и записать в массив с указанием индексов пар кластеров.
3. Отсортировать расстояния в возрастающем порядке.
4. Объединить кластеры с минимальным расстоянием, не включая уже объединенных.
5. Задать имя новому кластеру.
6. Добавить кластер в список кластеров для возврата.
7. Возвращаться к п. 4 пока количество объединений не достигнет определенной величины или не останется кластеров для объединения.
8. Добавить оставшиеся кластеры в список кластеров для возврата.
9. Вычислить среднюю точку всех кластеров в списке кластеров для возврата.
10. Вернуть объединенные данные.

3.3 Руководство пользователя

Программа предназначена для кластеризации профилей пользователей на основе их интересов. Входные данные задаются в текстовом(.txt) или в Excel формате (.xls, .xlsx). Для произведения кластеризации сначала необходимо сформировать файл Excel или txt. Входной формат данных должен иметь определенную структуру.

Текстовый файл должен иметь структуру:



Текстовый файл должен быть сформирован в виде таблицы, отступ между значениями должен быть ровно в один пробел. Наименования интересов и имен пользователей не должны иметь пробелов, если необходимо записать словосочетание или имя составное, то необходимо записать через какой-либо разделительный символ (например, “_”).

Структура Excel файла:

	A	B	C	D
1	Name	Интерес1	Интерес2	**
2	Имя1	1	2	*
3	Имя2	3	4	*
4	**	*	*	*

Структура Excel должна быть в виде таблицы. Где в первой строке, после первого столбца, идет перечисление интересов, по которым данные кластеризуются. В первом столбце идет перечисление имен пользователей, профили которых надо кластеризовать. Начиная со второго столбца второй строки заполняются численные коэффициенты пользователей.

Литература

1. Кластерный анализ [Электронный ресурс]: Материал из Национальной библиотеки им. Н. Э. Баумана. Режим доступа:URL - https://ru.bmstu.wiki/Кластерный_Анализ .
2. Обзор алгоритмов кластеризации данных [Электронный ресурс]. Режим доступа: URL - https://habr.com/ru/post/37185/
3. Кластерный анализ [Электронный ресурс]. Режим доступа: URL - https://ru.wikipedia.org/wiki/Кластерный_Анализ
4. С. А. Филиппов, В. Н. Захаров, С. А. Ступников, Д. Ю. Ковалев. Кластеризация профилей пользователей в рекомендательных системах поддержки жизнеобеспечения на основе реальных неявных данных. Режим доступа:URL - http://ceur-ws.org/Vol-1752/paper16.pdf