**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени И. Раззакова**

**Кафедра «ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»**

**ВЫПУСКНАЯ РАБОТА**

**«»**

**на квалификацию бакалавра**

**по направлению: 552800 Информатика и вычислительная техника**

**Специальность: 552801.04 Прикладная математика и информатика**

**Выполнил Ташиев Арстанбек Таалайбекович**

**Руководитель Кыштобаева Гулбара Кадыровна**

**БИШКЕК 2019**

[Введение 3](#_Toc10746813)

[Глава 1. Теоретические основы кластерного анализа 4](#_Toc10746814)

[1.1 Математическая модель кластерного анализа 5](#_Toc10746815)

[1.2 Практическое применение кластерного анализа 9](#_Toc10746816)

[1.3 Методы кластерного анализа 11](#_Toc10746817)

[1.4 Методы вычисления расстояния между объектами 15](#_Toc10746818)

[1.5 Методы объединения кластеров 16](#_Toc10746819)

[1.6 Кластерный анализ в сегментации потребителей на основе их интересов 17](#_Toc10746820)

[1.7 Описание программных и технических средств 19](#_Toc10746821)

[Глава 2. Кластеризация профилей пользователей 24](#_Toc10746822)

[2.1 Метод К-Средних 24](#_Toc10746823)

[2.2 Иерархический метод кластеризации 25](#_Toc10746824)

[Глава 3. Разработка проекта 26](#_Toc10746825)

[3.1 Структура программы 26](#_Toc10746826)

[3.2 Описание модулей программы 29](#_Toc10746827)

[3.3 Руководство пользователя 42](#_Toc10746828)

[Литература 43](#_Toc10746829)

# **Введение**

В современном мире, где количество данных очень велико, распространена проблема быстрой и эффективной обработки данных с большим объемом. В их числе проблема сегментации потребителей различных ресурсов, на группы, в котором отличие пользователей одной группы сводились бы к минимуму. Для решения таких задач, как правило, используются различные алгоритмы кластеризации. Кластеризация (или кластерный анализ) – задача разбиения множества объектов на непересекающиеся подмножества, называемых кластерами.

Алгоритмы кластеризации дают возможность разделения большого количества входных данных на неявные классы, посредством распределения каждого объекта в группы в котором, различия между объектами в одной группе сводится к минимуму.

Цель данной дипломной работы – исследование алгоритмов кластеризации различных объектов, по их характеристикам, для выделения схожих объектов в кластеры, на примере кластеризации профилей пользователей на основе их интересов. Основная цель работы реализация различных алгоритмов кластеризации данных, с использованием нескольких методов вычисления расстоянии между объектами. Основной задачей работы является построение автоматизированной информационной системы, которая обрабатывает входные, в различных форматах, и на выходе дает распределение по кластерам исходных данных, в удобном для изучения формате.

Объект исследования – различные алгоритмы кластеризации, инструменты необходимые для реализации и визуализации, такие как: c# - объектно-ориентированный язык программирования, ASP.NET Core 2 – платформа разработки веб-приложений.

Задачи для достижения поставленной цели:

1. Системный анализ и изучение предметной области
2. Формирование требований к автоматизированной системе кластеризации входных данных;
3. Создание структуры программы
   1. Реализация трехуровневой архитектуры программы
   2. Реализация алгоритмов кластеризации
   3. Разделение функционала вычисления расстояний между объектами от работы алгоритма
   4. Реализация обработки входных данных
   5. Создание интерфейса пользователя
   6. Создание БД, с использованием метода “Code First”
   7. Генерация тестовых данных
   8. Тестирование проекта
4. Анализ работы алгоритмов

# **Глава 1. Теоретические основы кластерного анализа**

Кластерный анализ - многомерная статистическая процедура, выполняющая сбор данных, содержащих информацию о выборке объектов, и затем упорядочивающая объекты в сравнительно однородные группы. Задача кластеризации относится к статистической обработке, а также к широкому классу задач обучения без учителя. Первое применение кластерного анализа был в области социологии. Впервые в 1939 году был определен предмет кластерного анализа и сделано его описание исследователем Трионом (Tryon). Главное его назначение – разбиение данных на взаимно не пересекающиеся множества, с наименьшей попарной разницей между объектами каждого множества (кластера). Тем самым кластерный анализ решает задачу классификации и выявления структуры данных.

Достоинством кластерного анализа является, возможность разбиения объектов не по одному параметру, а по целому набору признаков. Кроме того, кластерный анализ в отличии от большинства математическо-статистических методов не накладывает никаких ограничений на вид рассматриваемых объектов, и позволяет рассматривать множество исходных данных практически произвольной природы. Это имеет большое значение, например, для прогнозирования конъюнктуры, когда показатели имеют разнообразный вид, затрудняющий применение традиционных экономических подходов. Кластерный анализ позволяет рассматривать достаточно большой объем информации и резко сокращать, сжимать большие массивы разреженных данных, путём разделения их по схожести на группы, с наименьшей отличием объектов одной группы, делая их компактными и наглядными.

# **Математическая модель кластерного анализа**

**Задача кластерного анализа**

Задача заключается в том, чтобы на основании данных, содержащихся во множестве , разбить множество объектов кластеров (подмножеств) , так, чтобы каждый объект принадлежал только одному подмножеству разбиения. А объекты, принадлежащие одному и тому же кластеру, были сходными, в то время как объекты, принадлежащие разным кластерам, были однородными.

Кластерный анализ – это метод классификационного анализа; его основное назначение – разбиение множества исследуемых объектов и признаков на однородные в некотором смысле группы, или кластеры. Это многомерный статистический метод, поэтому предполагается, что исходные данные могут быть значительного объема, т.е. существенно большим может быть, как количество объектов исследования (наблюдений), так и признаков, характеризующих эти объекты. В прикладной статистике многомерными статистическими методами долго не могли пользоваться из-за отсутствия вычислительной техники для обработки больших массивов данных. Активно эти методы стали развиваться со второй половины ХХ в. при появлении быстродействующих компьютеров, выполняющих за доли секунды необходимые вычисления, на которые до этого уходили дни, недели, месяцы. В настоящее время препятствием к широкому использованию многомерных статистических методов, в том числе и кластерного анализа, является отсутствие у исследователей навыков и умения работать со статистическими пакетами прикладных программ. Техника кластеризации может применяться в самых различных прикладных областях, в том числе и в медицине. Например, кластеризация заболеваний, симптомов, признаков заболеваний, методов лечения может привести к более полному и глубокому пониманию медицинских проблем, связанных с лечением больных. Большое достоинство кластерного анализа в том, что он дает возможность производить разбиение объектов не по одному признаку, а по ряду признаков. Кроме того, кластерный анализ в отличие от большинства математико-статистических методов не накладывает никаких ограничений на вид рассматриваемых объектов и позволяет исследовать множество исходных данных практически произвольной природы. Так как кластеры – это группы однородности, то задача кластерного анализа заключается в том, чтобы на основании признаков объектов разбить их множество на m (m – целое) кластеров так, чтобы каждый объект принадлежал только одной группе разбиения. При этом объекты, принадлежащие одному кластеру, должны быть однородными (сходными), а объекты, принадлежащие разным кластерам, – разнородными. Если объекты кластеризации представить, как точки в n-мерном пространстве признаков (n – количество признаков, характеризующих объекты), то сходство между объектами определяется через понятие расстояния между точками, так как интуитивно понятно, что чем меньше расстояние между объектами, тем они более схожи.

**Математические характеристики кластера**

Кластер имеет следующие математические характеристики: центр, радиус, среднеквадратическое отклонение, размер кластера. Центр кластера – это среднее геометрическое место точек в пространстве переменных. Радиус кластера – максимальное расстояние точек от центра кластера. Кластеры могут быть перекрывающимися. Такая ситуация возникает, когда обнаруживается перекрытие кластеров. В этом случае невозможно при помощи математических процедур однозначно отнести объект к одному из двух кластеров. Такие объекты называют спорными. Спорный объект – это объект, который по мере сходства может быть отнесен к нескольким кластерам. Размер кластера может быть определен либо по радиусу кластера, либо по среднеквадратичному отклонению объектов для этого кластера. Объект относится к кластеру, если расстояние от объекта до центра кластера меньше радиуса кластера. Если это условие выполняется для двух и более кластеров, объект является спорным.

**Неоднозначность задачи кластерного анализа**

Неоднозначность данной задачи может быть устранена экспертом или аналитиком. Работа кластерного анализа опирается на два предположения. Первое предположение - рассматриваемые признаки объекта в принципе допускают желательное разбиение пула (совокупности) объектов на кластеры. Второе предположение - правильность выбора масштаба или единиц измерения признаков. Выбор масштаба в кластерном анализе имеет большое значение. Рассмотрим пример. Представим себе, что данные признака х в наборе данных А на два порядка больше данных признака у: значения переменной х находятся в диапазоне от 100 до 700, а значения переменной у - в диапазоне от 0 до 1. Тогда, при расчете величины расстояния между точками, отражающими положение объектов в пространстве их свойств, переменная, имеющая большие значения, т.е. переменная х, будет практически полностью доминировать над переменной с малыми значениями, т.е. переменной у. Таким образом из-за неоднородности единиц измерения признаков становится невозможно корректно рассчитать расстояния между точками. Эта проблема решается при помощи предварительной стандартизации переменных.

**Стандартизация**

Стандартизация или нормирование приводит значения все преобразованных переменных к единому диапазону значений путем выражения через отношение этих значений к некой величине, отражающей определенные свойства конкретного признака. Существуют различные способы нормирования исходных данных:

, , где – соответственно среднее и среднеквадратическое отклонение – наибольшее и наименьшее значение . Наряду со стандартизацией переменных, существует вариант придания каждой из них определенного коэффициента важности, или веса, который бы отражал значимость соответствующей переменной. В качестве весов могут выступать экспертные оценки, полученные в ходе опроса экспертов - специалистов предметной области. Полученные произведения нормированных переменных на соответствующие веса позволяют получать расстояния между точками в многомерном пространстве с учетом неодинакового веса переменных. В ходе экспериментов возможно сравнение результатов, полученных с учетом экспертных оценок и без них, и выбор лучшего из них.

Полученные при сборе данных начальные (первичные) оценки выполнения экспериментальных заданий далеко не всегда удобно использовать в дальнейшей работе. Их тем или иным способом преобразуют. Наиболее частыми преобразованиями являются**центрирование и нормирование**среднеквадратическими отклонениями. Под центрированием понимается линейная трансформация величин признака, при которой средняя величина распределения определенного признака становится равной нулю. Направление шкалы и ее единицы остаются при этом неизменными.

Суть нормирования состоит в переходе к другому масштабу — стандартизированным единицам измерения. При стандартизировании результатов тестовых испытаний нормирование чаще всего осуществляется с помощью среднеквадратических отклонений. Стандартизирование производится при нормальном распределении тестовых оценок или близком к нему по виду.

В психологии существует целый ряд шкал, основанных на нормальном распределении и имеющих разные значения М и s. Например, в шкале отклонений интеллекта IQ: М=100, s =15; в шкале Векслера М=10, s = 3. Распределения различных измеренных в эксперименте признаков имеют разные величины М и s . Переводя полученные первичные оценки разных признаков к распределению с одними и теми же М и s, мы получаем больше возможностей для оценки и сопоставления их варьирования.

**Формальное описание задачи**

Дано множество объектов данных , каждый из которых представлен набором атрибутов. Требуется построить множество кластеров , и отображение множества на множество , т.е. Отображение задает модель данных, являющуюся решением задачи. Множество определяется следующим образом:

Где – исследуемый объект. Каждый из объектов характеризуется набором параметров:

Каждая переменная может принимать значения из некоторого множества:

Задача кластеризации состоит в построении множества:

Здесь – кластер, содержащий похожие друг н друга объекты из множества :

Где – величина, определяющая меру близости для включения объектов в один кластер, – мера близости между объектами, называемая расстоянием.

**Расстояние**

Неотрицательное значение называется расстоянием между элементами , если выполняются следующие условия:

1. , для всех
2. , тогда и только тогда, когда
3. . Если расстояние меньше некоторого значения , то говорят, что элементы близки и помещаются в один кластер. В противном случае говорят, что элементы отличны друг от друга и их помещают в разные кластеры.

Алгоритмы(общее)

Большинство популярных алгоритмов, решающих задачу кластеризации, используют в качестве формата входных данных матрицу отличия . Строки и столбцы матрицы соответствуют элементам множества . Элементами матрицы являются значения в строке . Очевидно, что на главной диагонали значения будут равны нулю:

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин:

1. Не существует однозначно наилучшего критерия качества кластеризации. Известен целый ряд эвристических, а также ряд алгоритмов, не имеющих чётко выраженного критерия, но осуществляющих достаточно разумную кластеризацию “По построению”. Все они могут давать разные результаты.
2. Число кластеров, как правило, неизвестно заранее и устанавливается в соответствии с некоторым субъективным критерием.
3. Результат кластеризации существенно зависит от метрики, выбор которой, как правило, также субъективен и определяется экспертом.

# **Практическое применение кластерного анализа**

**В биологии**

В биологии кластеризация имеет множество приложений в самых разны областях. Например, в биоинформатике с помощью неё анализируются сложные сети взаимодействующих генов, состоящие порой из сотен или даже тысяч элементов. Кластерный анализ позволяет выделить подсети, узкие места, концентраторы и другие скрытые свойства изучаемой системы, что позволяет в конечном счете узнать вклад каждого гена в формирование изучаемого феномена.

В области экологии широко применяется для выделения пространственно однородных групп организмов, сообществ и т. п. Реже методы кластерного анализа применяются для исследования сообществ во времени. Гетерогенность структуры сообществ приводит к возникновению нетривиальных методов кластерного анализа (например, метод Чекановского).

В общем стоит отметить, что исторически сложилось так, что в качестве мер близости в биологии чаще используются меры сходства, а не меры различия.

**В социологии**

При анализе результатов социологических исследований рекомендуется осуществлять анализ методами иерархического агломеративного семейства, а именно методом Уорда, при котором внутри кластеров оптимизируется минимальная дисперсия, в итоге создаются кластеры приблизительно равных размеров. Метод Уорда наиболее удачен для анализа социологических данных. В качестве меры различия лучше квадратичное евклидово расстояние, которое способствует увеличению контрастности кластеров. Главным итогом иерархического кластерного анализа является дендрограмма или «сосульчатая диаграмма». При её интерпретации исследователи сталкиваются с проблемой того же рода, что и толкование результатов факторного анализа — отсутствием однозначных критериев выделения кластеров. В качестве главных рекомендуется использовать два способа — визуальный анализ дендрограммы и сравнение результатов кластеризации, выполненной различными методами.

Визуальный анализ дендрограммы предполагает «обрезание» дерева на оптимальном уровне сходства элементов выборки. «Виноградную ветвь» (терминология Олдендерфера М. С. и Блэшфилда Р. К.) целесообразно «обрезать» на отметке 5 шкалы Rescaled Distance Cluster Combine, таким образом будет достигнут 80 % уровень сходства. Если выделение кластеров по этой метке затруднено (на ней происходит слияние нескольких мелких кластеров в один крупный), то можно выбрать другую метку. Такая методика предлагается Олдендерфером и Блэшфилдом.

Теперь возникает вопрос устойчивости принятого кластерного решения. По сути, проверка устойчивости кластеризации сводится к проверке её достоверности. Здесь существует эмпирическое правило — устойчивая типология сохраняется при изменении методов кластеризации. Результаты иерархического кластерного анализа можно проверять итеративным кластерным анализом по методу k-средних. Если сравниваемые классификации групп респондентов имеют долю совпадений более 70 % (более 2/3 совпадений), то кластерное решение принимается.

Проверить адекватность решения, не прибегая к помощи другого вида анализа, нельзя. По крайней мере, в теоретическом плане эта проблема не решена. В классической работе Олдендерфера и Блэшфилда «Кластерный анализ» подробно рассматриваются и в итоге отвергаются дополнительные пять методов проверки устойчивости:

1. кофенетическая корреляция — не рекомендуется и ограничена в использовании;
2. тесты значимости (дисперсионный анализ) — всегда дают значимый результат;
3. методика повторных (случайных) выборок, что, тем не менее, не доказывает обоснованность решения;
4. тесты значимости для внешних признаков пригодны только для повторных измерений;
5. методы Монте-Карло очень сложны и доступны только опытным математикам.

В информатике

Кластеризация результатов поиска – используется для “Интеллектуальной” группировки результатов при поиске файлов, веб-сайтов, других объектов, предоставляя пользователю возможность быстрой навигации, выбора заведомо более релевантного подмножества и исключения заведомо менее релевантного – что может повысить простоту и удобность пользовательского интерфейса по сравнению с выводом в виде просто сортированного списка.

Сегментация изображений – кластеризация может быть использована для разбиения цифрового изображения на отдельные области с целью обнаружения границ или распознавания объектов.

Интеллектуальный анализ данных (англ. *data mining)* — кластеризация в Data Mining приобретает ценность тогда, когда она выступает одним из этапов анализа данных, построения законченного аналитического решения. Аналитику часто легче выделить группы схожих объектов, изучить их особенности и построить для каждой группы отдельную модель, чем создавать одну общую модель для всех данных. Таким приемом постоянно пользуются в маркетинге, выделяя группы клиентов, покупателей, товаров и разрабатывая для каждой из них отдельную стратегию.

# **Методы кластерного анализа**

Методы кластерного анализа можно разделить на две классификации:

1. **Иерархические и плоские**

Иерархические алгоритмы (алгоритмы таксономии) строят систему вложенных разбиений. Т.е. на выходе получается дерево кластеров, корнем которого является вся выборка, а листьями – наиболее мелкие кластеры.

Плоские алгоритмы строят одно разбиение объектов на кластеры.

1. **Четкие и нечеткие**

Четкие (или непересекающиеся) алгоритмы каждому объекту выборки ставят в соответствие номер кластера, т.е. каждый объект принадлежит одному и только одному кластеру.

Нечеткие (или пересекающиеся) алгоритмы каждому объекту ставят соответствие набор вещественных значений, показывающих степень отношения объекта к кластерам. Т.е. каждый объект относится к каждому кластеру с некоторой вероятностью.

**Алгоритмы иерархической кластеризации**

Среди алгоритмов иерархической кластеризации выделяются два основных типа. Восходящие и нисходящие алгоритмы.

Нисходящие алгоритмы работают по принципу “Сверху-вниз”: в начале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры.

Восходящие алгоритмы: в начале каждый объект помещается в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержатся в одном кластере. Таким образом строится система вложенных разбиений. Результаты таких алгоритмов обычно представляют в виде дерева- дендрограммы. Классический пример такого дерева – классификация животных и растений. Иерархические методы кластеризации, работающая по принципу снизу-вверх (от листьев к корню), называются агломеративными алгоритмами кластеризации.

Принцип работы иерархических агломеративных процедур состоит в последовательном объединении групп элементов сначала самых близких, а затем всё более отдалённых друг от друга. Принцип работы иерархических дивизивных процедур, наоборот, состоит в последовательном разделении групп элементов сначала самых далёких, а затем всё более близких друг от друга. Большинство этих алгоритмов исходит из матрицы расстояний (сходства). К недостаткам иерархических процедур следует отнести громоздкость их вычислительной реализации. На каждом шаге алгоритмы требуют вычисления матрицы расстояний, а следовательно, ёмкой машинной памяти и большого количества времени. В этой связи реализация таких алгоритмов при числе наблюдений, большем нескольких сотен, нецелесообразна, a в ряде случаев и невозможна. Общий принцип работы агломеративного алгоритма следующий. На первом шаге каждое наблюдение рассматривается как отдельный кластер. В дальнейшем на каждом шаге работы алгоритма происходит объединение двух самых близких кластеров, и с учётом принятого расстояния по формуле пересчитывается матрица расстояний, размерность которой, очевидно, снижается на единицу. Работа алгоритма заканчивается, когда все наблюдения объединены в один класс. Большинство программ, реализующих алгоритм иерархической классификации, предусматривает графическое представление классификации в виде дендрограммы.

**Алгоритмы квадратичной ошибки**

Задачу кластеризации можно рассматривать как построение оптимального разбиения объектов на группы. При этом оптимальность может быть определена как требование минимизации среднеквадратической ошибки разбиения:

Где – “Центр масс” кластера (точка со средними значения характеристик для данного кластера).

Алгоритмы квадратичной ошибки относятся к типу плоских алгоритмов. Самым распространенным алгоритмом этой категории является метод . Этот алгоритм строит заданное число кластеров, расположенных как можно дальше друг от друга. Работа алгоритма делится на несколько этапов:

1. Случайно выбрать точек, являющихся начальными центрами масс кластеров.
2. Отнести каждый объект к кластеру с ближайшим центром масс.
3. Пересчитать центры масс кластеров в зависимости от объектов внутри кластеров.
4. Если критерий остановки алгоритма не удовлетворен, вернутся к п. 2.

В качестве критерия остановки работы алгоритма обычно выбирают минимальное изменение среднеквадратической ошибки. Так же возможно останавливать работу алгоритма, если на шаге 2 не было объектов, переместившихся из кластера в кластер.

**Нечеткие алгоритмы**

Наиболее популярным алгоритмом нечеткой кластеризации является алгоритм . Он представляет собой модификатор метода .

Шаги работы алгоритма:

1. Выбрать начальное нечеткое разбиение объектов на кластеров путем выбора матрицы принадлежности размера .
2. Используя матрицу , найти значение критерия нечеткой ошибки: ,

Где – “центр масс” нечеткого кластера , .

4. Перегруппировать объекты с целью уменьшения этого значения критерия нечеткой ошибки.

3. Возвращаться в п.2 до тех пор, пока изменения матрицы не станут незначительными.

Этот алгоритм может не подойти, если заранее неизвестно число кластеров, либо необходимо однозначно отнести каждый объект к одному кластеру.

**Алгоритмы, основанные на теории графов**

Суть таких алгоритмов заключается в том, что выборка объектов представляется в виде графа , вершинами которого соответствуют объекты, а ребра имеют вес, равный расстоянию между объектами. Достоинством графовых алгоритмов кластеризации являются наглядность, относительная простота реализации и возможность внесения различных усовершенствований, основанные на геометрических соображениях. Основными алгоритмам являются алгоритм выделения связных компонент, алгоритм построения минимального покрывающего (остовного) дерева и алгоритм послойной кластеризации.

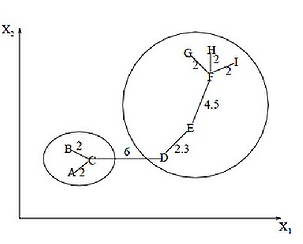
**Алгоритмы выделения связных компонент**

В алгоритме выделения связных компонент задается входной параметр и в графе удаляются все ребра, для которых расстояние больше . Соединенными остаются только наиболее близкие пары объектов. Смысл алгоритма заключается в том, чтобы подобрать такое значение , лежащее в диапазоне всех расстоянии, при котором граф разваливается на несколько связных компонент. Полученные компоненты и есть кластеры.

Для подбора параметра обычно строится гистограмма распределений попарных расстояний. В задачах с хорошо выраженной кластерной структурой данных на гистограмме будет два пика – один соответствует внутрикластерным расстояниям, второй – межкластерным расстояниям. Параметр подбирается из зоны минимума между этими пиками. При этом управлять количеством кластеров при помощи порога расстояния довольно затруднительно.

Алгоритм минимального покрывающего дерева

Алгоритм минимального покрывающего дерева сначала строит на графе минимальное покрывающее дерево, а затем последовательно удаляет ребра с наибольшим весом. На рисунке изображено минимальное покрывающее дерево, полученное для девяти объектов.



Путём удаления связи, помеченной CD, с длиной равной 6 единицам (ребро с максимальным расстоянием), получаем два кластера: {A, B, C} и {D, E, F, G, H, I}. Второй кластер в дальнейшем может быть разделён ещё на два кластера путём удаления ребра EF, которое имеет длину, равную 4,5 единицам.

# **Методы вычисления расстояния между объектами**

Для сравнения двух объектов необходим критерий, по которому будет происходить сравнение. Как правило, таким критерием является расстояние между объектами.

Евклидово расстояние – наиболее распространенная функция меры расстояния. Представляет собой геометрическое расстояние объектов в многомерно пространстве.

Квадрат Евклидового расстояния – применяется для увеличения расстояния для более отдаленных друг от друга объектов.

Расстояние городских кварталов (Манхэттенское расстояние) – Это расстояние суммы разностей по координатам. В большинстве случаев приводит к таким же результатам, как и для обычного расстояния Евклида. Однако, для этой меры влияние больших разностей уменьшается, т.к. не возводится в квадрат.

Расстояние Чебышева – Это расстояние может оказаться полезным, когда нужно определить два объекта разными, если они различаются по какой-либо одной координате. Расстояние Чебышева вычисляется по формуле:

Степенное расстояние – применяется в случае, когда необходимо увеличить или уменьшить вес, относящийся к размерности, для которой соответствующие объекты сильно отличаются. Степенное расстояние вычисляется по формуле:

Где – параметры, определяемы пользователем. Параметр ответственен за постепенное взвешивание разностей по отдельным координатам, параметр ответственен за прогрессивное взвешивание больших расстояний между объектами. Если оба параметра равны двум, то это расстояние совпадает с расстоянием Евклида.

# **Методы объединения кластеров**

В случае использования иерархических алгоритмов встает вопрос, как объединять между собой кластеры, как вычислять расстояния между ними. Существует несколько методов.

1. Одиночная связь (расстояния ближайшего соседа) – в этом методы расстояние между двумя кластерами определяется расстоянием между двумя наиболее близкими объектами (ближайшими соседями). Этот метод обычно работает очень хорошо, когда объекты происходят из отдельных групп. Если же кластеры имеют удлиненную форму или их естественный тип является “Цепочным”, то этот метод непригоден.
2. Полная связь (расстояние наиболее удаленных соседей) – В этом методе расстояния между кластерами определяются наибольшим расстоянием между любыми двумя объектами в различных кластерах (т.е. наиболее удаленными соседями). Этот метод обычно работает очень хорошо, когда объекты происходят из отдельных групп. Если же кластеры имеют удлиненную форму или их естественный тип является «цепочечным», то этот метод непригоден.
3. Невзвешенное попарное среднее - В этом методе расстояние между двумя различными кластерами вычисляется как среднее расстояние между всеми парами объектов в них. Метод эффективен, когда объекты формируют различные группы, однако он работает одинаково хорошо и в случаях протяженных, “цепочечного» типа, кластеров.
4. Взвешенное попарное среднее - Метод идентичен методу невзвешенного попарного среднего, за исключением того, что при вычислениях размер соответствующих кластеров (т.е. число объектов, содержащихся в них) используется в качестве весового коэффициента. Поэтому данный метод должен быть использован, когда предполагаются неравные размеры кластеров.
5. Невзвешенный центроидный метод – в этом методе расстояние между двумя кластерами определяется как расстояние между их центрами тяжести.
6. Взвешенный центроидный метод (медиана) – этот метод идентичен предыдущему, за исключением того, что при вычислениях используются веса для учета разницы между размерами кластеров. Поэтому, если имеются или подозреваются значительные отличия в размерах кластеров, этот метод оказывается предпочтительнее предыдущего.

# **Кластерный анализ в сегментации потребителей на основе их интересов**

Современная электронная коммерция, сопровождающая и поддерживающая процессы жизнеобеспечения, активно использует рекомендательные системы для решения задач адресного продвижения товаров и услуг с учетом конкретных пользовательских предпочтений. Основным источником информации о пользовательских предпочтениях являются данные об активности пользователей при посещении конкретного интернет ресурса. Эти данные собираются в основном неявным образом (протоколирование действий пользователя) и обладают следующими основными свойствами: значительный объем и быстрое изменение данных во времени. При этом адаптация под конкретного пользователя весьма сложная задача, поскольку для ее решения необходимо принимать во внимание как присущие человеку неопределенность и спонтанность в рамках конкретного Интернет-ресурса, так и множество неопределенностей, связанных с особенностями функционирования Интернет.

Одним из простейших подходов к выработке рекомендаций является использование статистических метрик выявления, например, наиболее популярных, дешевых(дорогих), близких по заданным характеристикам объектов и предложение их пользователям без учета их персональных предпочтений. Более сложные алгоритмы выявляют предпочтения пользователей посредством формирования, поведенческого профиля пользователя, который, в свою очередь, определяется на основании анализа его активности при выборе товаров и услуг. Также в настоящее время развиваются подходы, основанные на использовании нечеткой логики, позволяющей учитывать различные типы неопределенностей и кластеризовать пользовательские профили.

Практически все современные Интернет-ресурсы, ориентированы на работу с большим количеством пользователей, собирают информацию об их активности и анализируют(обрабатывают) ее с целью персонализации своего контента для каждого конкретного пользователя. В качестве наиболее характерных примеров можно выделить:

1. Поисковые машины, которые собирают и систематизируют информацию о страницах в сети Интернет заинтересовавших конкретных пользователей.
2. Интернет-магазины, которые собирают и систематизируют сведения о предпочтениях своих пользователей в части товаров и услуг.
3. Форумы, интернет-дневники и социальные сети, которые собирают информацию о том, в каких тематических разделах и группах принимает участие каждый пользователь и насколько активно.

Другими достаточно распространенными примерами протоколирования действий пользователей являются счетчики посещений страниц.

В контексте проблемы персонализации контента встает задача обработки этих данных и выявления определенных закономерностей, позволяющих сделать выводы о конкретных предпочтениях пользователей. Таким образом, основной целью обработки данных о пользовательской активности является извлечение полезной информации, которая может, в свою очередь, использоваться для решения следующих задач:

1. Кластеризация ресурсов. Группирование схожих по множеству посетителей ресурсов в несколько кластеров (групп) ресурсов. Кластеризация позволяет строить каталоги ресурсов, а также выявлять недостатки существующих тематических каталогов.
2. Кластеризация пользователей. Группирование схожих пользователей в кластеры аналогично кластеризации ресурсов, а также выявлять недостатки тематических каталогов.
3. Построение устойчивых поведенческих профилей пользователей в виде перечня групп ресурсов, посещаемых как данным пользователей, так и схожими с ним пользователями.
4. Построение расширенных профилей пользователей, включающих социально демографические данные (анкеты), описательные статистики и поведенческие профили. Расширенные профили позволяют классифицировать новых пользователей, выявлять зависимости между пользовательским.
5. Сегментация клиентской базы на основе расширенных профилей позволяет выделять сегменты, как по анкетным данным клиентов, так и по их поведению. Эта информация используется при маркетинговых исследованиях.
6. Прямой маркетинг. Предоставление рекламы и маркетинговых предложений конкретному пользователю на основе его поведенческого профиля.
7. Персонализация контента. Представление каждому пользователю сайта наиболее интересной для него информации в наиболее удобном для него виде. Знание информационных предпочтений пользователя позволяет динамически перестраивать контент сайта.
8. Построение карт сходства ресурсов и пользователей. Позволяет отображать множества наиболее посещаемых ресурсов и наиболее активных пользователей в виде точечного графика. Схожим ресурсам (пользователям) соответствует близкие точки на карте. Карту сходства можно использовать как графическое средство навигации.

Существуют различные методы и подходы, используемые на практике при решении перечисленных выше задач.

# **Описание программных и технических средств**

C# - язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998—2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде Visual Studio.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков C++, Delphi, Modula и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++) или вывода типов (в отличие от Haskell).

**ASP.NET** (Active Server Pages для .NET) — платформа разработки веб-приложений, в состав которой входит: веб-сервисы, программная инфраструктура, модель программирования, от компании Майкрософт. ASP.NET входит в состав платформы **.NET Framework** и является развитием более старой технологии Microsoft ASP.

**ASP.NET MVC Framework** — Фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.

В апреле 2009 года исходный код ASP.NET MVC был опубликован под лицензией Microsoft Public License (MS-PL). 27 марта 2012 года лицензия была изменена на Apache License 2.0

В настоящее время разрабатывается ASP.NET MVC 6, как часть ASP.NET Core.

**Основные компоненты ASP.NET MVC**

Платформа ASP.NET MVC базируется на взаимодействии трех компонентов: контроллера, модели и представления. Контроллер принимает запросы, обрабатывает пользовательский ввод, взаимодействует с моделью и представлением и возвращает пользователю результат обработки запроса.

Модель представляет слой, описывающий логику организации данных в приложении. Представление получает данные из контроллера и генерирует элементы пользовательского интерфейса для отображения информации.

Движок представлений

Для управления разметкой и вставками кода в представлении используется движок представлений. До версии MVC 5 использовались два движка: Web Forms и Razor.

Начиная с MVC 5 единственным движком, встроенным по умолчанию, является Razor. Движок WebForms использует файлы .aspx, а Razor — файлы .cshtml и .vbhtml для хранения кода представлений. Основой синтаксиса Razor является знак @, после которого осуществляется переход к коду на языках C#/VB.NET[[26]](https://ru.wikipedia.org/wiki/ASP.NET_MVC_Framework#cite_note-26). Также возможно и использование сторонних движков. Файлы представлений не являются стандартными статическими страницами с кодом html, а в процессе генерации контроллером ответа с использованием представлений компилируются в классы, из которых затем генерируется страница html.

**Маршрутизация**

При обработке запросов Фреймворк ASP.NET MVC опирается на систему маршрутизации, которая сопоставляет все входящие запросы с определенными в системе маршрутами, которые указывают какой контроллер и метод должен обработать данный запрос. Встроенный маршрут по умолчанию предполагает трехзвенную структуру: контроллер/действие/параметр.

**.NET Core** — это открытая универсальная платформа разработки, которая поддерживается корпорацией Майкрософт и сообществом .NET на сайте GitHub. Она является кроссплатформенной, поддерживает **Windows**, **Mac OS** и **Linux** и может использоваться на устройствах, в облаке, во внедренных системах и в сценариях **IoT**(Интернета вещей). В её основе лежат технологии .NET Framework и Silverlight. Она оптимизирована для мобильных и серверных рабочих нагрузок, поскольку обеспечивает поддержку самодостаточных развёртываний XCOPY.

Одной из определяющих особенностей .NET Core является гибкое развертывание: вы можете установить платформу либо как часть своего приложения, либо отдельно. С FDD ваш пакет развертывания будет меньше. Кроме того, использование дискового пространства и использование памяти на устройствах минимизируются, и вы можете запустить приложение .NET Core с учетом любой ОС.

Self-contained развертывание (SCD) включает в себя все компоненты (также библиотеки DotNet Core и среду выполнения). Этот тип развертывания дает вам полный контроль над версией .NET Core, используемой вашим приложением. Уникальные характеристики каждого типа развертывания гарантируют, что вы сможете развернуть приложение оптимальным способом в зависимости от конкретных потребностей.

В рамках кросс-совместимости платформа для разработки приложений включает в себя модульную инфраструктуру. Она выдается через NuGet, и вы можете получить доступ к пакетным функциям, а не к одной большой сборке. Как разработчик вы можете создавать легкие приложения, содержащие только необходимые пакеты NuGet, что сделает вашу программу безопаснее и производительнее.

Модульная инфраструктура также позволяет быстрее обновлять платформу .NET Core, поскольку затронутые модули могут обновляться и выпускаться по отдельности. Акцент на гибкость и быстрые релизы наряду с вышеупомянутым сотрудничеством позитивно позиционирует .NET Core в рамках движения DevOps.

Хотя .NET Core был разработан как кроссплатформенная версия .NET Framework с открытым исходным кодом, между ними существуют и другие отличия, которые уходят далеко за рамки этих двух ключевых свойств. Многие из этих сравнений являются результатом дизайна и новизны .NET Core. Модели приложений, основанные на технологиях Windows, не поддерживаются .NET Core, но консольные и ASP.NET модели приложений поддерживают как .NET Core, так и .NET Framework.

API .NET Core меньше, чем в .NET Framework, но он по мере его развития он будет увеличиваться. Кроме того, .NET Core реализует только некоторые подсистемы .NET Framework, чтобы поддерживать упрощенный и гибкий дизайн платформы. Эти различия могут в какой-то мере ограничить .NET Core, однако преимущества кроссплатформенного дизайна с открытым исходным кодом перевешивают любые ограничения по мере дальнейшего расширения платформы.

**ClosedXML** – это библиотека .NET для считывания, обработки и записи Excel 2007+ (.xls, xlsx) файлов.

**ADO.NET Entity Framework** (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как ADO.NET Data Services (*Astoria*), так и связка из Windows Communication Foundation и Windows Presentation Foundation, позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования MVC, MVP или MVVM. Подключенный и автономный уровни ADO.NET снабжают фабрикой, которая позволяет выбирать, вставлять, обновлять и удалять данные с помощью объектов соединений, команд, чтения данных, адаптеров данных и DataSet. Хотя все это замечательно, эти аспекты ADO.NET заставляют трактовать полученные данные в манере, которая тесно связана с физической схемой данных. Вдобавок способ конструирования физической базы данных (администратором баз данных) полностью сосредоточен на таких конструкциях базы, как внешние ключи, представления и хранимые процедуры. Сложность баз данных, спроектированных администратором, может еще более возрастать, если администратор при этом заботится о безопасности и масштабируемости. Это также усложняет код C#, который приходится писать для взаимодействия с хранилищем данных.

Платформа ADO.NET Entity Framework (EF) — это программная модель, которая пытается заполнить пробел между конструкциями базы данных и объектно-ориентированными конструкциями. Используя EF, можно взаимодействовать с реляционными базами данных, не имея дело с кодом SQL (при желании). Исполняющая среда EF генерирует подходящие операторы SQL, когда вы применяете запросы LINQ к строго типизированным классам.

LINQ to Entities — это термин, описывающий применение запросов LINQ к сущностным объектам ADO.NET.

Другой возможный подход состоит в том, чтобы вместо обновления базы данных посредством нахождения строки, обновления строки и отправки строки обратно на обработку в пакете запросов SQL, просто изменять свойства объекта и сохранять его состояние. И в этом случае исполняющая среда EF обновляет базу данных автоматически.

В Microsoft считают ADO.NET Entity Framework новым членом семейства технологий доступа к данным, и не намерены заменять им подключенный и автономный уровни. Однако после недолгого использования EF часто отдается предпочтение этой развитой объектной модели перед относительно примитивным миром SQL-запросов и коллекций строк/столбцов.

Тем не менее, иногда в проектах .NET используются все три подхода, поскольку одна только модель EF чрезмерно усложняет код. Например, при построении внутреннего приложения, которому нужно взаимодействовать с единственной таблицей базы данных, подключенный уровень может применяться для запуска пакета хранимых процедур. Существенно выиграть от использования EF могут более крупные приложения, особенно если команда разработчиков уверенно работает с LINQ. Как с любой новой технологией, следует знать, как (и когда) имеет смысл применять ADO.NET EF.

**Подходы в Entity Framework**

Изначально с самой первой версии Entity Framework поддерживал подход Database First, который позволял по готовой базе данных сгенерировать модель edmx. Затем эта модель использовалась для подключения к базе данных. Позже был добавлен подход Model First. Он позволял создать вручную с помощью визуального редактора модель edmx, и по ней создать базу данных. Начиная с 5.0 предпочтительным подходом становится Code First. Его суть - сначала пишется код модели на C#, а затем по нему генерируется база данных. При этом модель edmx уже не используется.

Подход Code First - появился позже подходов Model-First и Database-First и, как вы уже поняли, больше всего подходит для разработчиков, которые хотят писать код, а не работать с дизайнером модели EDM или средствами работы с базами данных (SQL Server Management Studio и T-SQL). Вы можете создать модель для вашего приложения, используя **объекты CLR (Common Language Runtime)** и **специальные объекты POCO (Plain Old CLR Object)**.

При проектировании приложений с подходом Code-First, вы сначала создаете классы модели данных не обращая никакого внимания на Entity Framework. После того, как вам понадобилось работать с базой данных, вы используете различные инструменты, которые проецируют структуру базы данных из созданной модели классов. После этого вы можете вернуться к этой модели в коде и, например, изменить ее. Эти изменения затем можно будет отразить в базе данных используя все те же инструменты.

Важным нововведением версии Entity Framework 5 в плане подхода Code-First, является то, что созданная модель классов теперь сразу является *сущностной моделью данных EDM (Entity Data Model)*, поэтому отпала необходимость использовать файл EDMX. В более ранних версиях разработчику, использующему подход Code-First, приходилось добавлять отношения между моделью классов и файлом EDMX, т.е. отображать любые изменения модели сразу в двух местах. Очевидно, что этот подход приводил к появлению кучи ошибок, если разработчик забывал синхронизировать эти изменения в обоих файлах.

Чтобы указать среде Visual Studio, что модель классов является моделью EDM, нужно во-первых установить сборки Entity Framework в проект, а во-вторых добавить класс контекста базы данных, унаследованный от класса **DbContext**, находящегося в пространстве имен *System.Data.Entity.*

# **Глава 2. Кластеризация профилей пользователей**

# **Метод К-Средних**

# **2.2 Иерархический метод кластеризации**

# **Глава 3. Разработка проекта**

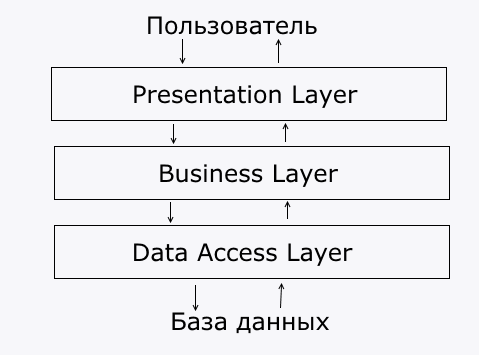
# **3.1 Структура программы**

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.

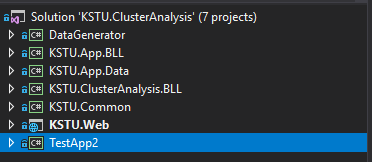
Условно, если объекту нужно получить доступ к определенному [сервису](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B5%D1%80%D0%B2%D0%B8%D1%81_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC)&action=edit&redlink=1), объект берет на себя 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.

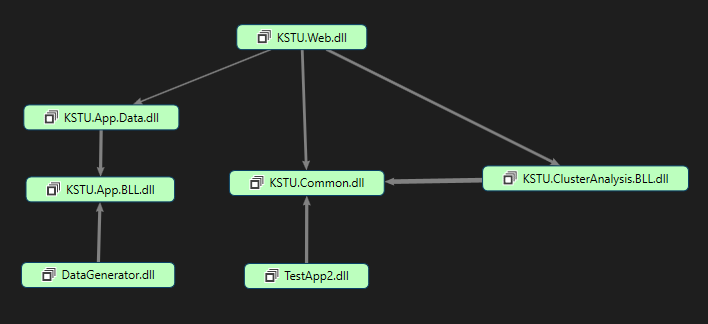
С другой стороны, излишнее использование внедрения зависимостей может сделать приложения более сложными и трудными в сопровождении: так как для понимания поведения программы программисту необходимо смотреть не только в исходный код, а еще и в конфигурацию, а конфигурация, как правило, невидима для IDE, которые поддерживают анализ ссылок и рефакторинг, если явно не указана поддержка [фреймворков](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с внедрениями зависимостей.

# **3.2 Описание модулей программы**

Программа состоит из 7 модулей:



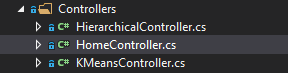
Связи между модулями.



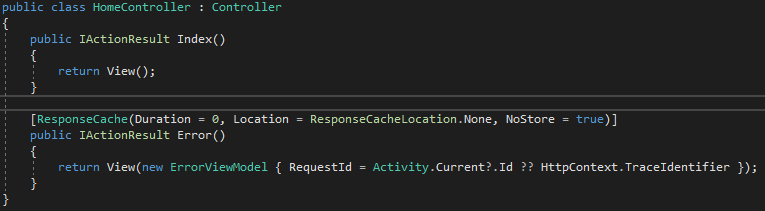
1. Модуль KSTU.Web – модуль веб-приложения. Основной модуль сайта.
2. Модуль KSTU.Common – библиотека классов. Где находятся основные компоненты программы, например, обработчик файлов.
3. Модуль KSTU.ClusterAnalysis.BLL – модуль где реализованы методы кластерного анализа.
4. Модуль KSTU.App.Data – модуль 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.

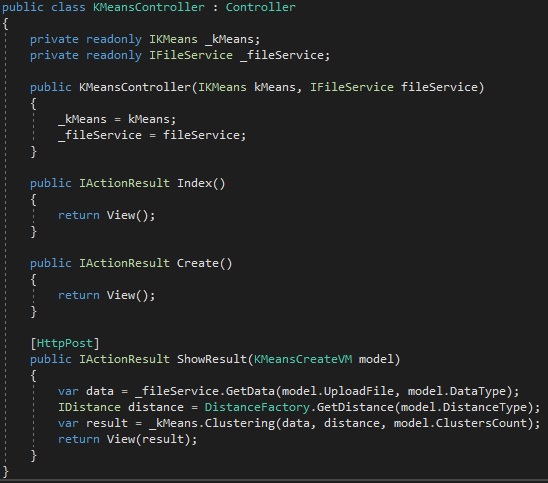
****

**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.

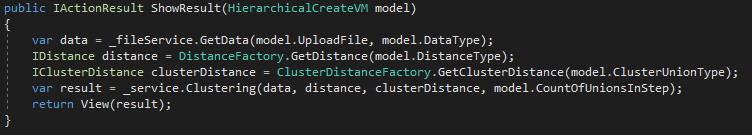
****

**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.

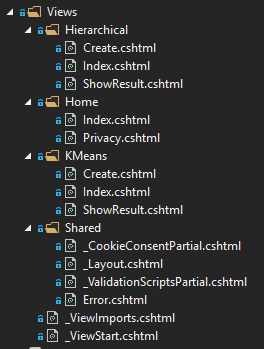
**HierarchicalController –** контроллер который отвечает за реализацию иерархического метода кластеризации. В этом контроллере те же методы что и в предыдущем, различаются лишь сервис, который используется для кластеризации и метод “ShowResult”

****

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.

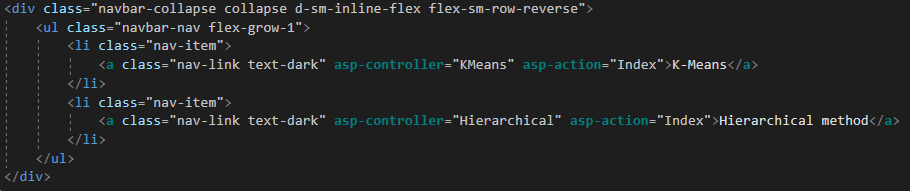
Основнаяструктура мастер страницы:

****

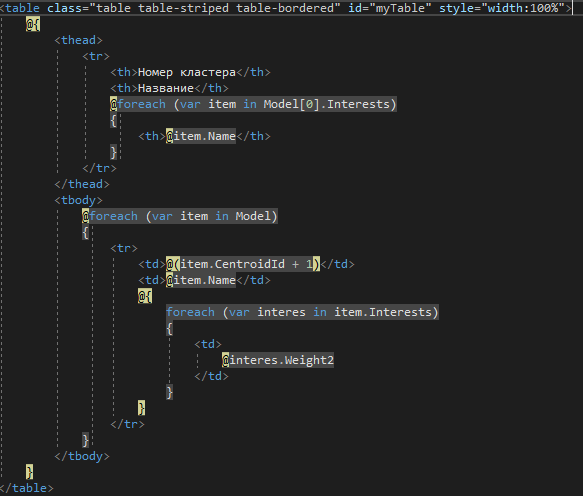
Подключение необходимых библиотек:

****

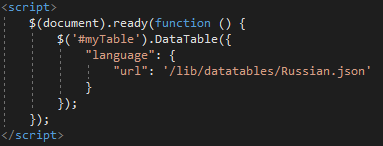
Основное меню страницы(сверху):

****

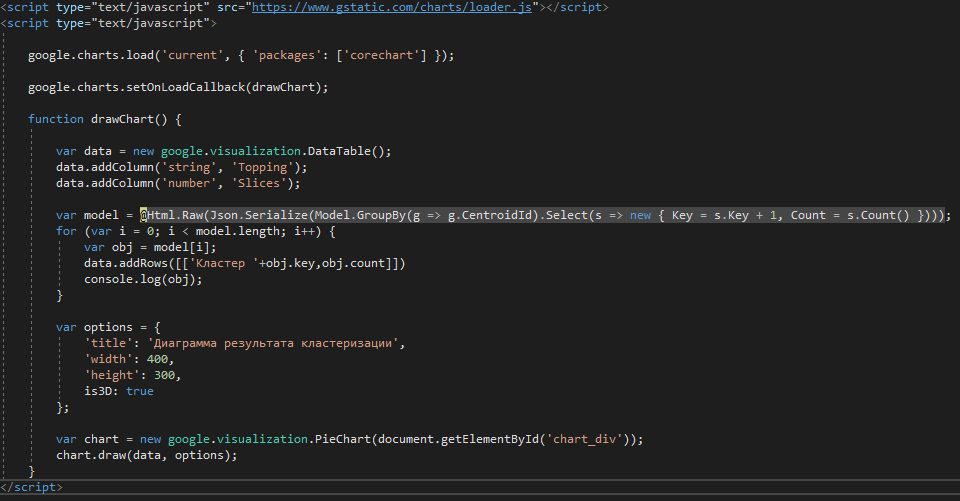
Представление ShowResult метода К-Средних – это страница вывода результата кластеризации методом К-Средних. Результат выводится в виде 3D диаграммы и таблицы отношений объектов к кластерам, количество кластеров указывается при вводе данных. Заполнение таблицы происходит при помощи вставки кода на языке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

****

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.

****

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

****

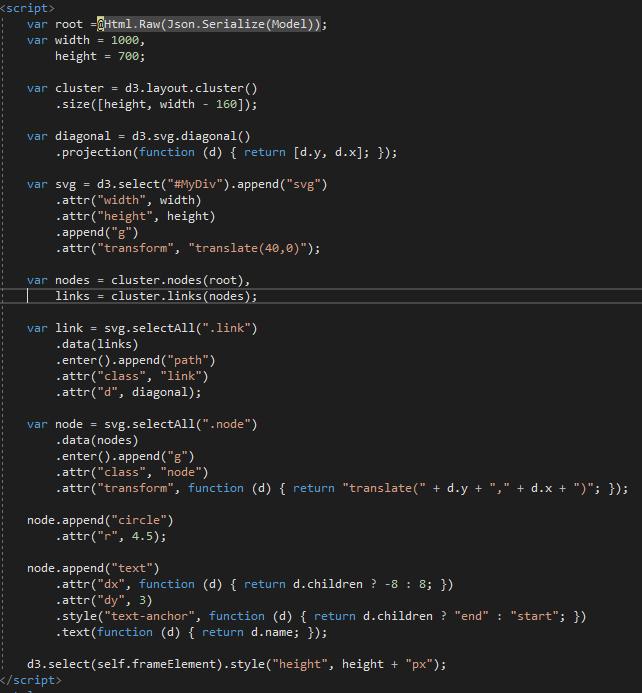
Данные для диаграммы конвертируем в формат Json, с помощью Html-Helper’а, затем из нее выстраиваем диаграмму с помощью инструментов библиотеки. После выполнения всех необходимых операции по построению диаграммы, результат записывается в определенную на странице DIV’ку.

****

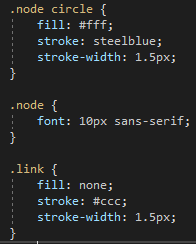
Представление ShowResult иерархического метода – это страница вывода результата кластеризации иерархического метода. Результат выводится в виде дендрограммы(дерева).

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:

****

Стили дендрограммы задаются отдельно от библиотеки:

****

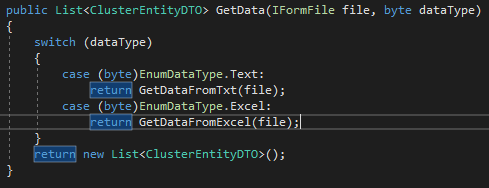
**Модуль KSTU.Common** – это библиотека основных классов проекта. Тут хранятся основные классы, используемые в проекте:

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.

****

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).

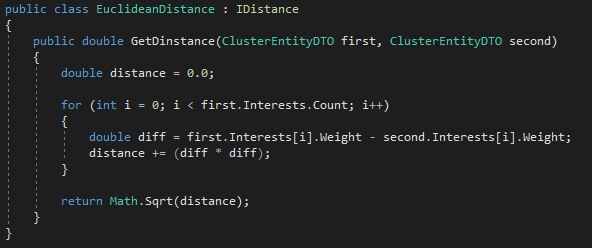
Метод GetDataFromTxt – метод считывания и обработки данных с текстового файла. Этот метод принимает один аргумент, файл введенный пользователем с формы. Для считывания данных используется класс StreamReader. Считанные данные записываются в массив строк. Затем обрабатываются и создаются объекты класса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

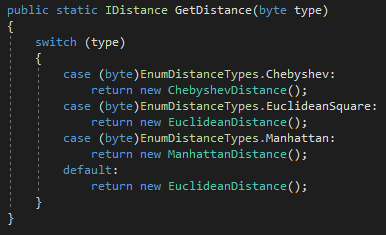
**IDistance –** интерфейс имеющий один метод GetDistance. Принимает два объекта и возвращает численную величину расстояния между этими объектами. Имеет четыре реализации:

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):



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.



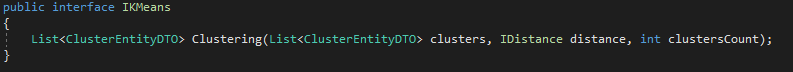
**IClusterDistance** – интерфейс для вычисления расстояний между кластерами. Реализация схожа с IDistance, только входящие аргументы отличаются. Метод принимает полный кластер, список объектов кластера, затем различными методами вычисляет расстояние между кластерами. Интерфейс имеет четыре реализации:

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.

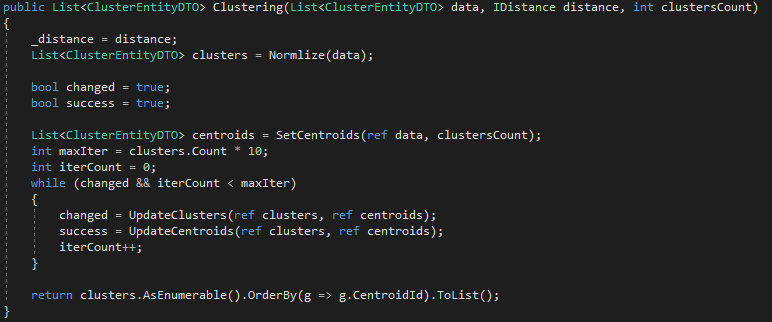
****

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:

****

**Метод Clustering**

**Алгоритм:**

1. Сначала входные данные нормализуются при помощи метода Normalize.
2. Задаются начальные центральные точки *К -* 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.
2. Вычисление суммы квадратов разностей веса текущего интереса и среднего значения.
3. Переназначить веса по текущему интересу для всех объектов.
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:**

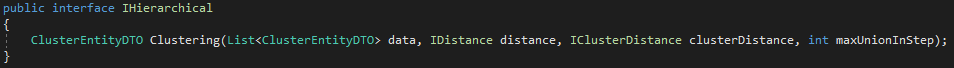
1. Найти для текущего объекта ближайший кластер и добавить к нему этот объект.
2. Повторять п.1 пока все объекты не будут присвоены к какому-либо кластеру.
3. Высчитать количество объектов в каждом кластере.

**Метод UpdateCentroids**

**Алгоритм:**

1. Очистить веса интересов центров масс.
2. Для каждого интереса текущего кластера вычислить среднее значение между всеми объектами этого кластера

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

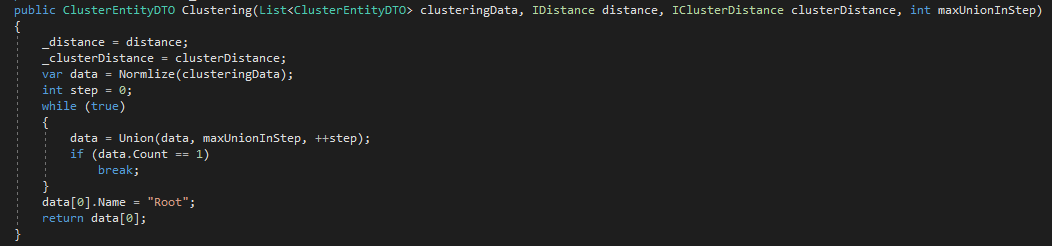
****

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.



**Метод Clustering**

**Алгоритм:**

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

**Метод Normalize**

**Алгоритм:**

1. Вычисление среднего значения, среди численных значений определенного интереса.
2. Вычисление суммы квадратов разностей веса текущего интереса и среднего значения.
3. Переназначить веса по текущему интересу для всех объектов.
4. Выполнить п.1-3 для всех имеющихся интересов в входных данных.

**Метод Union**

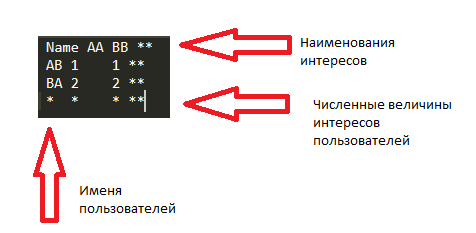
**Алгоритм:**

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.

# **Руководство пользователя**

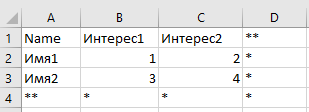
Программа предназначена для кластеризации профилей пользователей на основе их интересов. Входные данные задаются в текстовом(.txt) или в Excel формате (.xls, .xlsx). Для произведения кластеризации сначала необходимо сформировать файл Excel или txt. Входной формат данных должен иметь определенную структуру.

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:



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>