

СОДЕРЖАНИЕ

| | |
|---|----|
| ВВЕДЕНИЕ | 2 |
| 1 ТЕХНОЛОГИИ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ ИНТЕРНЕТ-СИСТЕМ | 6 |
| 1.1 Рекомендательные системы | 6 |
| 1.2 Примеры использования РС | 8 |
| 1.3 РС для рынка недвижимости | 17 |
| 1.4 Среда программирования .NET | 22 |
| 1.5 Технология Entity Framework | 23 |
| 1.6 Библиотека Brightstar DB | 25 |
| 1.7 Библиотека Nancy для создания интернет-приложений . . | 28 |
| 1.8 Библиотека SharpTAL для создания HTML-страниц . . . | 30 |
| 1.9 Идентификация сеанса пользователя в HTTP | 31 |
| 2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ | 35 |
| 2.1 Функциональное моделирование предметной области . . | 35 |
| 2.2 Архитектура системы | 37 |
| 2.3 Проектирование структуры базы данных | 38 |
| 2.4 Импорт данных в хранилище | 41 |
| 2.5 Проектирование интернет-приложения | 45 |
| 2.6 Реализация подсистемы рекомендаций | 53 |
| 2.7 Тестирование системы | 71 |
| ЗАКЛЮЧЕНИЕ | 79 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 80 |
| ПРИЛОЖЕНИЕ 1 Листинг ClusterAnalysis.cs | 84 |
| ПРИЛОЖЕНИЕ 2 Листинг FlatClusterAnalysis.cs | 89 |
| ПРИЛОЖЕНИЕ 3 Листинг SlopeOne.cs | 95 |

ВВЕДЕНИЕ

Каждый человек рано или поздно сталкивается с проблемой выбора на рынке недвижимости. Принимаемые в это время решения значительно влияют на дальнейшую жизнь семьи, организации, фирмы и т.д. Поэтому всякая помощь на этом этапе представляется необходимым моментом. Для поддержки принятия решения существует целая индустрия – услуги риелторов, задач которых организовывать процесс поиска интересующих объектов недвижимости на рынке, обеспечение безопасности сделки, соответствующий документооборот.

С другой стороны, на рынке недвижимости Иркутской области находятся много различных уникальных объектов, которые уже долгое время не могут найти себе рачительного хозяина, способного эффективно эту собственность ввести в эксплуатацию. Как следствие, продавцы терпят убытки, платя налоги за нерентабельную собственность, а у покупателей нет возможности инвестировать свободные капиталы. Причина такой ситуации – неразвитость информационного обеспечения, т.е. несмотря на активную рекламу таких объектов риелторским конторам не удается найти покупателя.

Деятельность риелторов в некоторой степени автоматизируется. Офисные пакеты используются для подготовки документации, ведется бухгалтерия, а также существуют сетевое программное обеспечение, которое организует и регламентирует взаимоотношения риелторов друг с другом, и с сервисами услуг оценки объектов и рынка недвижимости в целом. При помощи таких CRM организуются сделки между риелторами: на некоторую плату объект передается другому риелтору на продажу, – контроль ранних договоренностей.

С другой стороны, согласно рекламе сайта Avito, каждая вторая квартира продается именно через интернет-сайты без участия риелторов. Опрос риелторов показал, что это не совсем так. Много зависит от «традиций» того или иного региона Российской Федерации. Например, в Иркутской области и Москве недвижимость предпочитают продавать через интернет-сайты, причем делается несколько предложений одной и той же квартиры на нескольких сайтах. Опыт показывает, что это

практически всегда приводит к тому, что цена на объект недвижимости падает ниже рыночной. В Екатеринбурге, наоборот, недвижимость преимущественно продается через риелторские конторы, что и стабилизирует цены на жилье, и делает процесс купли-продажи более предсказуемым для продавца.

В таблице 1 представлены количественные характеристики роста рынка недвижимости Иркутской области.. В период с 2010 по 2016 год количество сделок только по жилой недвижимости на вторичном рынке выросло примерно на 20%. Ежегодно в Иркутской области вводится в эксплуатацию не менее 30 000 новых квартир. На сайте Avito.ru выставлены на продажу более 5 000 квартир, находящихся на территории Иркутской области.

Таблица 1 — Характеристика рынка недвижимости Иркутской области

| Год | Количество сделок |
|------|-------------------|
| 2010 | 12594 |
| 2011 | 13756 |
| 2012 | 15080 |
| 2013 | 14540 |
| 2014 | 15651 |
| 2015 | 15005 |
| 2016 | 15982 |

Таким образом, для гармонизации процессов на рынке недвижимости имеет смысл разработать информационную систему, которая не только бы сообщала актуальную информацию об объектах недвижимости, но и помогала бы искать такие объекты покупателям. Такой сервис должен функционировать на основе современных методов анализа и структуризации данных: выбирать для конкретных пользователей объекты недвижимости, которые потенциально могли бы его заинтересовать. Такими системами в разных предметных областях выступают *рекомендательные системы* (РС).

Рекомендательные системы [6] – информационные системы поддержки принятия решений, предназначенные для оценки уровня интереса пользователя к определенному продукту или сервису объекту

на основе имеющейся информации о пользователе и/или объекте. Отрасль разработки РС начала активно развиваться при появлении онлайн-сервисов продаж, и в настоящее время РС – одно из активных направлений развития систем поддержки принятия решений, ориентированное, прежде всего, на коммерческое использование, а также на решение задач повышения продуктивности поиска релевантной информации.

В коммерции РС позволяют решать задачи установления, что именно представляет ценность для потребителя в виде набора конкретных объектов (например, товаров или услуг), сужение вариантов выбора и предоставление схожих вариантов других объектов, тем самым упрощая выбор. РС позволяют также выявлять новые характеристики объектов, например, при помощи ведения классификаций объектов и анализа набора известных признаков. Использование РС позволяет отделам снабжения коммерческих фирм-поставщиков предоставлять уникальный сервис каждому потребителю, увеличивая его доверие и лояльность к поставщику, увеличивая продажи и конверсию, а также получая и накапливая больше знаний о потребителях.

Рекомендательные системы появились в интернете достаточно давно, около 20 лет назад. Однако настоящий подъем в этой области случился примерно 5-10 лет назад, когда произошло соревнование Netflix Prize. Компания Netflix тогда давала в прокат не цифровые копии, а рассылала VHS-кассеты и DVD. Для них было очень важно повысить качество рекомендаций. Чем лучше Netflix рекомендует своим пользователям фильмы, тем больше фильмов они берут в прокат. Соответственно, растет и прибыль компании. В 2006 году они запустили соревнование Netflix Prize. Они выложили в открытый доступ собранные данные: около 100 миллионов оценок по пятибалльной шкале с указанием ID поставивших их пользователей. Участники соревнования должны были как можно лучше предугадывать, какую оценку поставит определенному фильму тот или иной пользователь. Качество предсказания измерялось при помощи метрики RMSE (средне-квадратичное отклонение). У Netflix уже был алгоритм, который предсказывал оценки пользователей с качеством 0.9514 по метрике RMSE (см. раздел 1.2.5). Задача была улучшить предсказание хотя бы на 10%

— до 0.8563. Победителю был обещан приз в \$1 000 000. Соревнование длилось примерно три года. За первый год качество улучшили на 7%, дальше все немного замедлилось. Но в конце две команды с разницей в 20 минут прислали свои решения, каждое из которых проходило порог в 10%, качество у них было одинаковое с точностью до четвертого знака. В задаче, над которой множество команд билось три года, все решили каких-то двадцать минут. Опоздавшая команда (как и многие другие, участвовавшие в конкурсе) остались ни с чем, однако сам конкурс очень сильно подстегнул развитие в этой области [7].

Целью данной выпускной квалификационной работы магистранта является проектирование и реализация рекомендательной системы для рынка недвижимости Иркутской области, применимой для решения перечисленных выше проблем.

Для достижения цели **решены следующие задачи:**

1. Анализ предметной области рекомендательных систем и РС рынка недвижимости;
2. Разработать методику вычисления рекомендаций для объектов рынка недвижимости;
3. Проектирование структуры базы данных и архитектуры системы;
4. Реализация РС на основе предложенных методик и структур данных;
5. Тестирование разработанной рекомендательной системы.

К создаваемому программному обеспечению выдвигались следующие дополнительные **требования:**

1. система должна вырабатывать рекомендации в режиме «холодного старта»;
2. не принуждать пользователей к регистрации и авторизации, но при этом обеспечивать накопление необходимой информации;
3. тестирование системы должно производиться на данных рынка недвижимости Иркутской области, что обеспечивает экспертную оценку рекомендаций;
4. программное обеспечение необходимо реализовать преимущественно на свободных (open-source)) технологиях.

1 ТЕХНОЛОГИИ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ ИНТЕРНЕТ-СИСТЕМ

В данном разделе рассматриваются технологии, которые используются для проектирования и реализации рекомендательной системы для рынка недвижимости Иркутской области.

1.1 Рекомендательные системы

В области рекомендательных систем используется специальная терминология. *Объектом* обозначается то, что система рекомендует *пользователям*, например, продукты, услуги, товары, новости, книги, DVD и т.п. *Профилем пользователя* или *объекта* являются данные, характеризующие пользователя или объект. Именно эти данные используются в процессе оценивания *релевантности* объекта к желаниям пользователя. Этот процесс называется *фильтрацией* (filtering). В результате фильтрации объекты *ранжируются* в соответствии с полученной оценкой, а пользователю предоставляется некоторое конечное подмножество, элементы которого имеют максимальную релевантность, т.е. оцениваются как наиболее интересные пользователю. Далее под *интересом* будем понимать именно интерес пользователя к объекту. Т.к. РС – это, прежде всего, информационные системы, то все объекты и пользователи описываются при помощи *атрибутов*. Именно атрибуты являются входной информацией во все процедуры оценивания интереса. *Качество рекомендации* – оценка точности предсказания интереса, сделанного РС, например, в сравнении с имеющимися примерами, т.е. оценками конкретных объектов конкретными пользователями.

Рекомендательные системы полезны не только для информационных ресурсов и порталов электронной коммерции, но и могут также открыть новые возможности в области безопасности, автомобильной промышленности [8], рекламе [9] и др. Существует ряд подходов к оценке интереса.

Существует ряд подходов к оценке интереса:

1. на основе *фильтрации содержания* (content-based information fil-

tering), при этом в информационной системе создаются профили пользователей и объектов, включающие социальный статус пользователя, возраст, место проживания, род деятельности, а также характеристики, выражающие интерес пользователя к объекту; профили объектов включают позицию в системе классификации, его потребительские характеристики.

2. на основе *коллаборативной фильтрации* (collaborative filtering), где используется информация о поведении пользователей в прошлом, например, перечень покупок или оценок объектов, сделанных на сайте интернет-магазина в прошлом пользователями из той же группы интересов, при этом аналитическим блоком информационной системы автоматически формируются классификации объектов, производится *ранжирование атрибутов* по степени значимости в оценке интереса.
3. *интеллектуальные* (knowledge-based), где оценка вычисляется на основе формализованных знаний.
4. *гибридные* (hybridprediction) методы, которые базируются на подходах пп. 1 и 2, включая элементы из 3, что призвано повышать эффективность 1 и/или 2.

Например, в Music Genome Project музыкальный аналитик оценивает каждую композицию по сотням различных музыкальных характеристик, при помощи которых выявляются музыкальные предпочтения пользователя. Перечень оценок формирует *профиль музыкального произведения*. Основная проблема первого типа РС (фильтрации содержания) — это работоспособность системы на начальном этапе ее эксплуатации, так называемый «*холодный старт*»: для новых пользователей в системе нет необходимой информации в профиле для принятия решения о том, какие объекты следует предлагать. В связи с этим в современных рекомендательных системах реализуется механизм сбора и анализа данных о пользователях с применением *явных* и *неявных методов*.

Явные методы сбора данных выполняют следующие действия:

- запрос у пользователя оценки объекта по некоторой шкале;

- ❑ запрос у пользователя ранжировки группы объектов от наилучшего к наихудшему;
- ❑ предъявление пользователю двух объектов с вопросом о том, какой из них лучше;
- ❑ предложение создать список объектов, характеризующих предпочтения пользователя.

Примерами неявного сбора данных выступают:

- ❑ наблюдение за тем, что просматривает пользователь в интернет-магазине или базе данных;
- ❑ ведение записей о поведении пользователя онлайн.

Сбор информации из социальных сетей, например, как в [5-7].

Второй тип РС, основанные на коллаборативной фильтрации, сравнивают однотипные данные, полученные от разных людей и вычисляют список рекомендаций для конкретного пользователя. Для вычисления рекомендаций используется, например, граф интересов. Таким образом, РС представляют собой информационные системы, дополненные алгоритмами, позволяющими обнаружить в хранилище объекты, которые не имеют непосредственного отношения к запросу пользователя. Любопытно, что рекомендательные системы часто используют как поисковые машины для индексации необычных данных.

Более подробное описание различных подходов к реализации подсистем РС рассмотрим на примерах из современной литературы.

1.2 Примеры использования РС

В обзоре [13] рассмотрены РС в области предоставления пользователям текстовых документов, в частности, научных статей. Больше половины (55%, 34 из 62) систем построены на основе фильтрации содержания. Алгоритмы коллаборативной фильтрации использованы только в 18% (11 из 62) случаев. Представлены подходы, основывающиеся на стереотипировании и гибридных методах. Авторы исследования пришли к выводу, что в 81% случаев моделирование пользователя на основе автоматического сбора информации не приносит значимых результатов по сравнению с явным указанием набора ключевых слов.

В основе характеристик объектов, научных статей, в исследованных РС используют просто ключевые слова, содержащиеся в документах, реже N -граммы, а также нетекстовые элементы, такие как ссылки на другие статьи и фамилии авторов. Самая популярная модель для хранения представления статей – модель векторного пространства. Моделирование пользователя осуществляется при помощи графов и списков тем, назначенных пользователям в результате машинного обучения. Темы объединяются в иерархические справочники, например, на основе классификаторов АСМ. В рассмотренных подходах тексты извлекаются из заголовков, аннотаций, заголовков, введения, предисловия, предоставленных автором ключевых слов, библиографии, основного текста, социальных тегов и цитирований контекста.

В РС, где применялась коллаборативная фильтрация, и ни в одном из проектов не удалось успешно использовать явные рейтинги: пользователи были слишком ленивы, чтобы самостоятельно задавать рейтинг статьям. Неявные рейтинги получены из данных по количеству страниц, прочитанных пользователем, взаимодействию пользователей с документами (загрузка, редактирование, представление) и цитирования. Главная проблема коллаборативной фильтрации для научных работ – это дефицит информации, например, для РС научных статей Mendeley по сравнению с Netflix (онлайн фильмы) дефицит составляет три порядка. Неявные рейтинги объектов получаются из анализа одновременной загрузки статьи (со-загрузка) разными пользователями одной группы, совместного просмотра (со-просмотр), совместное цитирование статьями (со-цитирование) одних и тех же источников. Оказалось, что со-цитирование, будучи эффективным в начале появления статьи на сервисе РС, через два года начинает уступать со-загрузке. Популярным подходом представления результата такого анализа являются графы. Вершины графа – это статьи, представленные наборами атрибутов, а дуги – соотношения между статьями.

Основными проблемами в области РС являются следующие:

- отсутствие общего базиса оценивания качества систем (по предметным областям), включая объективную информацию о реаль-

ных оценках реальных пользователей, нестабильность методов оценивания и высокая их зависимость от «шума»;

- ❑ неиспользованный потенциал научных исследований: новые научные результаты не внедряются в практические приложения (большинство работающих РС базируются на простых методах), данные существующих практических реализаций РС научно не исследуются, нет тесного взаимодействия со смежными областями анализа данных, а так же друг с другом, низкий научный интерес к РС;
- ❑ в оценке удовлетворенности не учитываются факторы конфиденциальность, безопасности данных, разнообразие, разметка и презентация информации; в значимом количестве РС моделирование пользователя было крайне примитивно - набор ключевых слов, собственная статья или просто фрагмент текста, представляющих научные интересы пользователя.

Среди открытых проектов выделяются MyMediaLite, LensKit, Mahout, Duine, RecLabCore, Easyrec и Recommender.

1.2.1 Методы фильтрации содержания

В статье [10] решается задача анализа профиля пользователя в социальной сети ВКонтакте для решения проблемы холодного старта в решении задачи рекомендации жанров и произведений музыки и фильмов. Авторами разработана РС «EZSurf» автоматизирующая процесс веб-сёрфинга и фильтрации контента, используя профиль пользователя в социальной сети «ВКонтакте», а также API сервисов last.fm, TheMovieDB для получения сведений о схожих объектах (музыкальных произведений). Такой подход существенно упрощает хранилище данных РС, поскольку не требует создания собственной системы классификаций и базы объектов.

В статье [14] рассматривается задача выделения N объектов с наивысшими оценками интереса, задача top- N , при применении фильтрации контента. Предлагается математическая модель контентной рекомендательной системы, основанная на нечетких множествах, кри-

терий оценки качества рекомендаций и алгоритм решения задачи. Математическая модель и алгоритм протестированы на данных сайта last.fm.

1.2.2 Методы коллаборативной фильтрации

Подходы, основанный на коллаборативной фильтрации, в настоящее время более популярны, чем подходы на основе фильтрации содержимого, вероятно из-за того, что представляет собой отражение практического опыта: большинство коммерческих РС вынуждены решать проблему недостатка информации, «холодный старт», а также адаптируемости существующих сообществ пользователей к новым объектам.

Суть идеи коллаборативной фильтрации изображена на рисунке 1.1. Пользователям предоставляются интересующие их товары, которые они оценивают «положительно» или «отрицательно» пока пользуются сайтом, например, интернет-магазином.

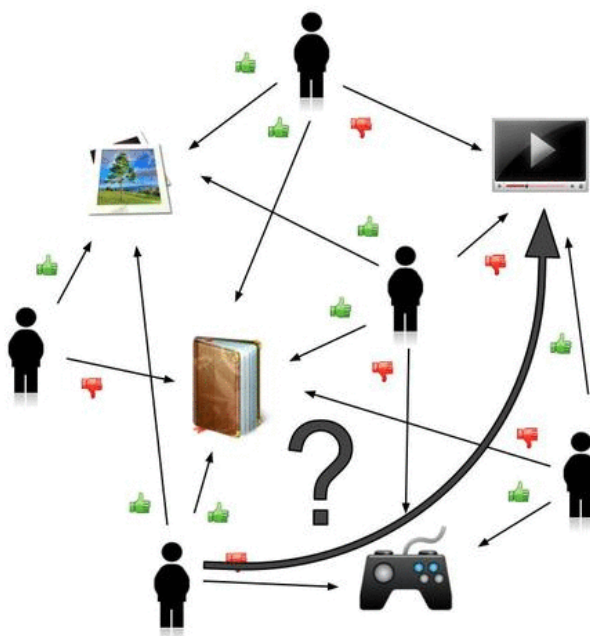


Рисунок 1.1 — Накопление рейтингов покупателей о товарах

Накопленные данные записываются в таблицу, которая представлена на рисунке 1.2, где столбцы – это объекты, т.е. товары, а строки – это оценки пользователей.


























| |  |  |  |  |
|---|---|---|---|--|
|  |  |  |  |  |
|  | |  |  |  |
|  |  |  |  | |
|  |  | |  | |
|  |  |  |  |  |

Рисунок 1.2 — Таблица собранной информации об оценках товаров пользователями

Задача рекомендательной системы, заключается в заполнение «всех» ячеек, помеченных вопросительным знаком на рисунке 1.2.

Для оценки интереса пользователя к объекту, помеченному вопросительным знаком, сравниваем пятого (последнего) пользователя со всеми остальными пользователями по критерию схожести его выбора. Пользователи под номерами «пять» и «три», схожи по первым двум товарам, т.к. оба ответили «положительно» по данным товарам. Пользователи под номерами «пять» и «два», схожи по товарам под номерами «два» и «четыре», т.к. оба ответили одинаково «положительно» на товар под номером «два» и «отрицательно» под номером «четыре». Следовательно, можно предположить, что пользователь под пятым номером скорее всего отнесется «отрицательно» к товару под номером «три», что представлено на рисунке 1.3.

Полученная оценка означает, что пользователю данный товар будет показываться в последнюю очередь.

Математические обозначения элементов модели сравнения состоит из набора пользователей U и набора объектов O . Тогда O_u — это множество элементов, оцененных пользователем u , U_o — множество пользователей, которые оценили объект o , $r_{u,o}$ — оценка пользователя

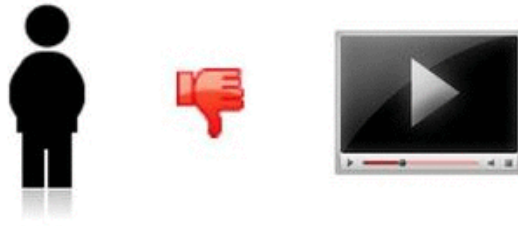


Рисунок 1.3 — Пример рекомендации

и для объекта o , \mathbf{r}_u — вектор всех оценок пользователя u , \mathbf{r}_o — вектор всех оценок объекта o , \bar{r}_u и \bar{r}_o — средние значения оценок пользователя u и объекта o , соответственно. Сравнительная оценка обозначается $\hat{r}_{u,i}$. Для задания этой оценки сначала задается мера близости объекта i к объекту j . Рассмотрим несколько популярных вариантов оценки близости.

Коэффициент Пирсона [8]:

$$s_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}$$

где $U = U_i \cup U_j$ — множество пользователей, которые оценили объекты i и j .

Косинус угла между двумя векторами \mathbf{r}_i и \mathbf{r}_j :

$$s_{i,j} = \cos(\mathbf{r}_i, \mathbf{r}_j) = \frac{\mathbf{r}_i \cdot \mathbf{r}_j}{|\mathbf{r}_i| |\mathbf{r}_j|}.$$

Затем производится формирование конечного множества объектов S наиболее близких к объекту o . Вычисление рейтинга объекта o делается по формуле:

$$\hat{r}_{u,o} = \frac{\sum_{j \in S} S_{o,j} \cdot r_{u,j}}{\sum_{j \in S} |S_{o,j}|}.$$

Популярный подход к формированию множества рекомендаций — это упорядочивавшие всех объектов по критерию схожести и выборке некоторого фиксированного количества объектов с максимальным

рейтингом [Нефедова]. В качестве меры схожести (similarity) двух объектов выступает \cos угла между N -мерными векторами.

В [3,10] так же представлен обзор способов использования вышеупомянутых методов вычисления оценок, которые разделены на два класса – *анамнестические*, т.е. основывающиеся на одновременной обработке всех имеющихся данных, и *модельные*, где производится предварительная обработка данных, выполняемая, например, раз в сутки. Второй класс позволяет быстрее вычислять оценки интереса, однако не обеспечивает актуальности данных. В классе аналитических способов, как правило, используются методы многомерного анализа данных на основе «ближайшего соседства» (Neighbourhood-based), в то время как в модельных методах используется методы анализа скрытых факторов (Latent Factors). Существуют гибридные методы, объединяющие оба предыдущих класса.

1.2.3 Метод Slope One

Одним из интересных достижений в области РС является изобретение метода коллаборативной фильтрации «Slope One» [1], разработанного в компании Amazon для формирования рейтинга товаров известного на весь мир магазина. Исследователи изучали факторы, приводящие к эффекту «переобучения», т.е. нестабильному поведению системы при подаче на вход примерно одинаковых данных. В [1] предложено три схемы с формулой оценки вида $f(x) = x + b$ и предварительно вычисленном средней разнице рейтингов (оценок пользователей) двух объектов для всех пользователей, которые оценили оба эти объекта одновременно. Здесь, x – известный рейтинг объекта, заданного (оцененного) некоторым пользователем, b – средняя разница рейтинга по всем пользователям для данного объекта, $f(x)$ – оценка объекта для нового пользователя, т.е. пользователя, не задававшего какую-либо оценку для данного объекта.

Основное отличие данного алгоритма заключается в его простоте реализации, запросы к базе исходных данных также реализуются просто, алгоритм достаточно точен, а также алгоритм поддерживает

и статический и динамический режим обновления промежуточных данных. То есть алгоритм позволяет создавать подсистемы оценки объектов для метода коллаборативной фильтрации для реально функционирующих систем.

Наглядно суть функционирования метода представлена на рисунке 1.4 (рисунок адаптирован из оригинальной статьи [1]). Средняя разница в рейтинге товаров o_1 и o_2 по двум известным оценкам пользователя u_1 составляет $1.5 - 1 = 0.5$. Следовательно, если известно, что пользователь u_2 присвоил рейтинг товару o_1 рейтинг 2, то рейтинг товара o_2 составит $2 + 0.5 = 2 + (1.5 - 1) = 2.5$.

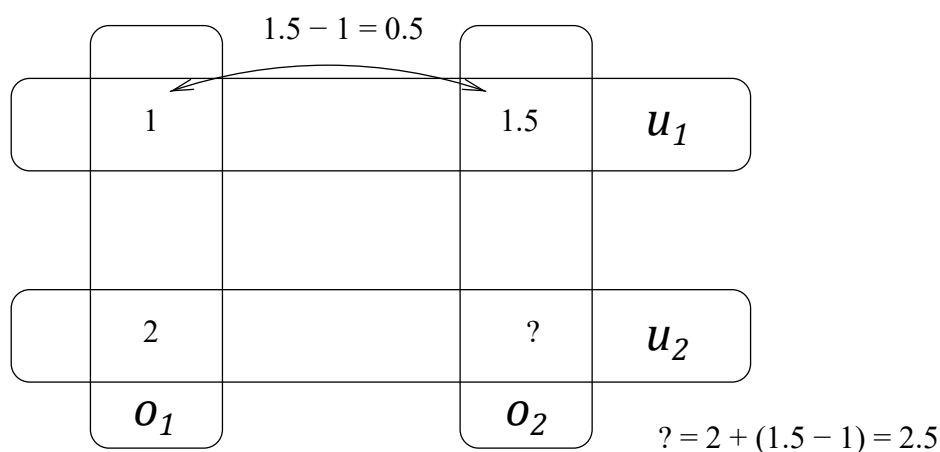


Рисунок 1.4 — К объяснению работы алгоритма Slope One

При помощи данного алгоритма рейтинги объекта o_2 , в общем случае, вычисляются из оценок нескольких пользователей и относительно разных объектов. Т.е. получается несколько рейтингов одного и того же объекта. В этом случае считается средневзвешенная оценка полученных рейтингов, где в качестве весового коэффициента выступает количество раундов оценивания, другими словами, «сколько раз пользователь u_i проголосовал за «опорный» объект o_j ?»

1.2.4 Гибридные методы

В статье [16] рассмотрена задача разработки алгоритмов оценивания лекционного материала. Авторами предложен алгоритм вычисления близости лекций (объектов), где каждая лекция характеризуется

подмножеством некоторого набора значений (например, подмножеством авторов лекций относительно множества всех авторов). Базовый алгоритм реализует подход фильтрации содержания. Для алгоритма подобраны коэффициенты, при помощи которых можно объединять оценки различных атрибутов в одну общую оценку лекции. Наиболее значимыми атрибутами оказались «категории», «авторы», «языки», «название» и «описание». Цель – синтезировать набор лекций фиксированной длины, рекомендованных для просмотра заданному пользователю, из фиксированного множества «новых», не использованных в построении профилей пользователя и объекта.

Далее алгоритм дополняется предсказателем последовательностей лекций: заданы примеры последовательностей из трех лекций, требуется для последовательностей из двух предложить третью, четвертую и т.д. Последовательности лекций приобретены системой неявно, т.е. фиксируя просмотренные пользователем лекции. Алгоритм занял первое место в соревновании, причем со значительным отрывом от второго места. Производится внедрение результатов исследований в области анализа сигналов и предсказания последовательностей событий.

В [17] создан рекомендательный сервис новостей посетителям сайта, время пересчета рекомендаций в котором на каждую тысячу новых записей в журнале WEB-сервера составляет 1.5–2 с., что авторами заявлено как ресурс, функционирующий в режиме, близком к реальному времени. Для проекта Рамблер-новости подобный результат является удовлетворительным, так как 1000 новых запросов к сайту делается за чуть большее время. В исследовании использован адаптированный алгоритм MinHash для идентификации записей журнала и неточного их сравнения. Целью работы было показать целесообразность применения NoSQL-технологий для создания сервисов указанного качества.

Важным свойством приведенной реализации является то, что задачи хранения и анализа данных удалось объединить с задачей предоставления доступа к результатам в единой системе, избежав накладных расходов на перемещение данных из одного источника в другой, что

улучшило общую производительность сервиса. Кроме того, предложенный подход упрощает решение повседневных задач сбора статистики о взаимодействии пользователя с веб-приложением путем анализа структурированных логов мощным языком запросов СУБД MongoDB. В результате продемонстрировано, что применение NoSQL к решению подобного класса задач является весьма перспективным.

1.2.5 Оценка релевантности рекомендаций

Как и любая система структуризации информации РС оцениваются с точки зрения степени корректности расчетов рекомендаций. Классическим подходом является пример из [18], где рассматривается задача сравнительной оценки различных подходов к построению РС. Эта оценка, RMSE, среднеквадратическое отклонение, выполняется при помощи формулы:

$$\text{RMSE} = \sqrt{\frac{1}{|D|} \sum_{(u,o) \in D} (\hat{r}_{u,o} - r_{u,o})^2},$$

где D – множество пар пользователей u и объектов o , $r_{u,o}$ – оценка интереса объекта o пользователем u , $\hat{r}_{u,o}$ – оценка интереса, сделанная РС. Другие оценки представлены в [15].

1.3 РС для рынка недвижимости

В [22] предложен проект системы управления недвижимым имуществом, где варианты объектов предлагаются на основе вывода на прецедентах (case-based reasoning). Задача подсистемы вывода – найти прецедент в базе имеющихся прецедентов, похожий на запрос пользователя. Система помогает покупателям найти имущество, соответствующее их запросам. При этом система выводит суждения о свойствах объектов. Полученная информация затем используется в процессе фильтрации содержания и коллаборативной фильтрации. В дополнение к полученному списку выводится также наиболее популярные (most visited) варианты.

В теоретическом исследовании [20] в системе пользователи разделены на продавцов и покупателей. Продавцы "рекламируют" свое имущество, выставленное на продажу, выделяя те свойства недвижимости, которые сами считают важными. Предложена идея того, как получать данные для фильтрации содержания в задаче разработки РС управления имуществом: продавцы выступают в виде экспертов-оценщиков недвижимости, формируя информационную базу для фильтрации содержания. Предложенную идею можно дополнить, если ввести третий класс пользователей - экспертов-риелторов и позволить им дополнять базу данных прецедентов новыми суждениями.

В [23] опытным путем показано, что использование Интернета не влияет значительно на эффективность поиска недвижимости с целью ее покупки по критериям времени поиска, его гибкости и "удовлетворенность результатом". Согласно исследованию Национальной ассоциации риелторов, проведенном в 2011 году, показано, что 88% покупателей выбрали Интернет в качестве основного источника информации, но при этом среднее время на поиск жилья составило 12 недель, оно оказалось сравнимым с измерением, проведенным в 2009 году. Пользователь просматривает больше информации (как объектов, так и их свойств), но на анализ этой информации так же тратится много времени. Для повышения скорости выдачи результата авторами разработан алгоритм поиска на основе анализа поведения пользователей в процессе поиска объекта недвижимости, а также WEB-система, основанная на прецедентном выводе и онтологической концептуальной модели предметной области, ориентированная на пользователя.

В [23] авторами выделены три ключевые характеристики объекта недвижимости, которые в значительной мере являются определяющими в процессе принятия решения – это "расположение"(location), "потребительская характеристика"(housing unit property) и "цена"(price). Большинство РС используют именно эти характеристики для фильтрации содержания, причем для критерия "потребительская характеристика" задаются формальные параметры объекта (площадь, номер этажа, количество балконов, комнат-спален и т.п.). На окончательное решение также влияет окружение объекта - расстояние до магазинов,

школ, детских садов. Для того, чтобы учесть эти характеристики в [23] построена онтологическая модель, связывающая различные характеристики недвижимости в три древовидные структуры, описывающие варианты терминов "расположение" "потребительская характеристика" и "цена". Например, как вариант, под "расположением" понимается "расстояние" до места работы "пешком выраженное в минутах. Так же "расположение это наличие в "окружении" (environment) объекта недвижимости "услуг фитнеса". При помощи онтологии получена возможность сравнивать не вполне "схожие" объекты, что повышает точность обработки информации.

Важным достижением авторов [23] является разработанный прототип РС, в котором пользователю предоставляется возможность указать на карте города область (окружность с заданным радиусом), в которой он хотел бы приобрести объект недвижимости, уточнить его потребительские характеристики и возможный диапазон цен.

Затем система выводит на карту варианты объектов недвижимости. Далее пользователь может уточнить другие характеристики, тем самым сужая количество предоставляемых вариантов. В сравнении с сервисами, подобным Avito.ru [24], пользователю предлагается меньше вариантов, т.е. система оценивает интерес пользователя более точно, и сильнее сужает количество альтернатив.

В [21] решается сложная логистическая задача организация процесса управления имуществом, в который вовлечены разнообразные группы людей в изменяющихся деловых и экономических условиях. В оценке учитываются не только экономические и бизнес-критерии, но и такие критерии, как "технологичность" "комфорт" "пространство" "административные" и "технические". Основная цель исследования - разработать модель, в которой различные группы людей будут максимально удовлетворены в "рациональной микро- и макро-среде".

Эффективность использования имущества предлагается оценивать по целой системе критериев, включающей цену объекта, цену владения этим объектом, цену ремонта, возможности его использования (capacity), количеству операций, которые необходимо выполнить по передаче собственности, надежность, комфорт, срок физической и

технической эксплуатации, и др. Авторы разрабатывают математический аппарат для оценивания каждого объекта недвижимости – цены, эргономики, стоимости ремонта, назначение и т.п. Математическое обеспечение РС предложено развивать в направлении ухода от поиска «наиболее экономически выгодного управления недвижимостью» к мультикритериальному выбору и тем самым повысить эффективность вычислительных процессов.

1.3.1 Коллаборативная фильтрация объектов недвижимости

Коллаборативная фильтрация в ее изначальной постановке эффективно применима для расчета оценок интереса к массовым товарам, которые приобретаются покупателями на интернет-сайтах. Объекты недвижимости – это штучный товар и могут быть приобретены только одним покупателем. Поэтому выработка положительной оценки затруднена: невозможно интерпретировать факт покупки квартиры как положительную оценку. Причины тому две – сайты продаж недвижимости не распространяют данную информацию, они только фиксируют факт наличие квартиры на рынке. Вторая причина (проблема) состоит в том, что если даже стало известно, что квартира куплена конкретным покупателем, эта квартира больше не участвует в оценивании другими покупателями.

Проблема в большей части решается следующим образом. Считать любое проявление некоторого интереса к объекту недвижимости со стороны покупателя положительной оценкой. Например, если покупатель выбрал квартиру из списка и занялся подробным ознакомлением с ее параметрами и фотографиями, считать, что данный объект представляет некоторый интерес.

Вторым подходом к применению метода коллаборативной фильтрации к объектам недвижимости – это проводить оценку не отдельных объектов недвижимости, а из классов. Для этого надо реализовать в РС подсистему анализа схожести объектов недвижимости (их атрибутов) и разделении их на некоторые классы. И если покупатель заинтересовался объектом из некоторого подмножества (кластера), то положительную

оценку получает этот весь класс. Внутри класса объекты недвижимости можно считать равноценными.

Таким образом, применение одной лишь коллаборативной фильтрации для выработки рекомендаций и «борьбы» с «холодным стартом» в РС рынка недвижимости невозможно, поэтому, необходимо дополнять ее другими методами и подходами, что автоматически переводит такие РС в класс гибридных.

1.3.2 Сопутствующие технические задачи

Одной из важных задач, решаемых при разработке РС, является создания пользовательского интерфейса, адекватно отображающего систему критериев, ко которым необходимо производить подбор объектов для пользователя. Например, в статье [19] представлен модуль естественно-языкового интерфейса к базе данных РС, который реализован на основе математических моделей семантических объектов. При помощи модели решаются задачи определения семантики языковой конструкции, заданной пользователем, включая синонимы, классы, отношения и ограничения. В статье приводятся сведения о программной реализации предложенного метода в среде PHP + SQL и результатах тестирования программы на задаче доступа к базе данных РС автомобильного салона.

В [20] решается проблема обеспечения ограничения доступа к личным данным пользователей в контексте построения РС встраивания рекламных сообщений в информационный поток. При этом необходимо контролируемо предоставлять в одностороннем порядке в РС информацию из БД пользователей. Предложено вместо традиционных средств VPN (Virtual private network) использовать режим функционирования сети с синхронным изменением IP-адреса сервера и переключение клиента на этот адрес.

Кроме того, на современном этапе развития РС важными вопросами, решаемыми в процессе проектирования РС объектов недвижимости являются:

- ❑ разработка концептуальной модели предметной области в виде онтологии;
- ❑ создание математических моделей предсказания значений атрибутов, описывающих объект недвижимости;
- ❑ реализация механизмов компьютерного обучения и логического вывода на основе прецедентов;
- ❑ обеспечение информационного наполнения РС для оценки качественных атрибутов (например, наличия школ и магазинов в шаговой доступности).

Решение данных задач в значительной мере влияет на качество и точность предсказания оценки значимости объекта недвижимости для пользователя.

1.4 Среда программирования .NET

Средой разработки веб-приложения выбран .NET/Mono как некоторый компромисс между производительностью реализации программного кода и производительностью его исполнения, а также богатым набором программных пакетов.

.NET Framework - программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общезыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Mono - проект по созданию полноценного воплощения системы .NET Framework на базе свободного программного обеспечения. Он обеспечивает практически полную переносимость кода и среды исполнения для стандарта .NET версии 4.65.

Особенностью платформы является высокая производительность виртуальной машины, которая работает по принципу JIT (just in-time compilation), при этом исходный код компилируется в код стековой виртуальной машины CIL (Common intermediate language), и уже на платформе исполнения CIL преобразуется в код процессора рабочей

станции. Интуитивно процесс запуска приложений .NET чуть медленнее, чем чисто скомпилированные программы, значительно быстрее, чем программы на языке Python.

В настоящее время в сообществе Open-source (Free software foundation) ведется разработка прямого компилятора языка. Если разработка окажется успешной, то получится среда программирования такая же гибкая как Java и с высокой скоростью исполнения программного кода.

Также необходимо отметить гибкость синтаксиса .NET, в сравнении, например со средой программирования Java. В .NET можно создавать классы по отдельным частям, это так называемые `partial class`-ы. Такую возможность легко использовать например при разработке систем, где часть структуры классов порождается из како-го либо описания, при этом не т необходимости создавать дополнительные наследования. Языки .NET развиты в части поддержки свойств (`properties`), а также декораторов – структур вида `[Entity]`. Все это вместе создает среду для программирования программного обеспечения, ориентированного на решение именно прикладных задач.

Основным же недостатком .NET является жесткая необходимость использования специализированных сред разработки приложений (IDE, `integrated development environments`) для управления проектом, в частности настройки `.sln`- и `.csproj`-файла проекта. Вручную такую настройку сделать практически невозможно ввиду чрезвычайной сложности этих форматов.

1.5 Технология Entity Framework

Entity Framework представляет объектно-ориентированную технологию на базе среды программирования .NET, которая обеспечивает объектно-ориентированное представление реляционных данных. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных (БД), то Entity Framework представляет собой более высокий уровень, который позволяет абстрагироваться от реляционных свойств базы данных и получать доступ к данными независимо от типа хранилища [4].

Первая версия Entity Framework – 1.0 вышла в 2008 году и представляла библиотеку с небольшим набором функций, базовую поддержку ORM (object-relational mapping – отображения данных на объекты) и один подход к взаимодействию с БД – Database First. С выходом версии 4.0 в 2010 году Entity Framework стал рекомендуемой технологией для доступа к данным, а в сам фреймворк были введены новые возможности взаимодействия с БД – Model First и Code First. Дополнительные улучшения функций среды последовали с выходом версии 5.0 в 2012 году. И наконец, в 2013 году был выпущен Entity Framework 6.0, обладающий возможностью асинхронного доступа к данным.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. Любая сущность обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа int, но и могут представлять более комплексные структуры данных. Каждая сущность, как правило, описывается несколькими свойствами, которые отличают эту сущность от других и уникально ее определять. Подобные свойства играют роль ключей. При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ не только извлекаются из БД определенные строки, хранящие объекты, но и конструируются объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД. Entity Data Model состоит из трех уровней: *концептуального, уровень хранилища и уровень сопоставления* (отображения, mapping). На концептуальном уровне происходит определение классов сущностей, ис-

пользуемых в приложении. Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных. Уровень сопоставления служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц. Таким образом, через классы, определенные в приложении, обеспечивается взаимодействие с таблицами из базы данных.

Рассмотрим пример определения новой сущности в БД, поддерживаемой Entity Framework. В качестве объекта создадим пользователя:

```
public class User  
{  
    public int Id { get; set; }  
    public string Name { get; set; }  
    public int Age { get; set; }  
}
```

Это обычный класс, который содержит некоторое количество свойств. Каждое свойство будет сопоставляться с отдельным столбцом в таблице БД.

Надо отметить, что Entity Framework в режиме Code First требует определения ключа элемента для создания первичного ключа в таблице в БД. По умолчанию при генерации БД среда в качестве первичных ключей рассматривает свойства с именами Id или [Имя_класса]Id (то есть UserId).

Для взаимодействия с БД нам нужен контекст данных, т.е. среда, при помощи которой осуществляется создание экземпляров, в данном случае, пользователей, их запись в БД, механизмы транзакций, модификация объектов и т.п.

1.6 Библиотека Brightstar DB

В качестве контекста в данном проекте выступает база данных, реализованная на основе библиотеки Brightstar DB.

BrightstarDB является уникальной технологией хранения данных для платформы .NET. Она сочетает в себе гибкость, масштабируемость и производительность, а также позволяет создавать приложения, используя знакомые инструменты разработчиков. Ассоциативная модель

BrightstarDB построен на гибкой модели ассоциативных данных – RDF. Граф RDF позволяет представлять все существующие, имеющие практический смысл, виды моделей данных. Модель основана на концепции троек. Каждая тройка – это ассоциация некоторого свойства с определенным ресурсом или значением. Модель позволяет описывать и представлять данные любой структуры высокого уровня, что создает средства разработки систем, эволюционирующих во времени, и комплексных систем на основе объединения данных различной природы.

BrightstarDB находится среди немногих NoSQL – баз данных, предлагающих механизмы, «понимающие» описания структур данных, что позволяет им автоматически управлять отношениями между сущностями. Большинство NoSQL – баз данных требуют от разработчиков, чтобы он сам заботился об обновлении связей между документами и хранении дополнительных данных в хранилищах, отображающих ключи на значения. Базы данных NoSQL не особо хорошо справляются со моделями данных реальных приложений, таких как социальные сети или обработка графов.

BrightstarDB, будучи NoSQL – базой данных, поддерживает технологии Семантического веба. Она может использоваться как встроенная библиотека или как служба на сервере. К серверу обеспечивается сетевые соединения по протоколам HTTP, TCP/IP и через поименованные каналы. BrightstarDB обеспечивает поддержку Entity Framework в режиме Code First. Инструментарий, совместимый с Entity Framework также порождает объектный контекст при помощи задаваемых разработчиком .NET-приложений интерфейсов хранимых объектов в BrightstarDB.

Используемая ассоциативная модель позволяет добавлять новые данные в базу данных BrightstarDB без традиционной необходимости определения схемы данных. Это еще больше повышает гибкость и поддерживает эволюционную разработку проектов, которая является важнейшей особенностью именно современных программных

решений. Хранилище без схемы данных, реализованное в BrightstarDB, позволяют импортировать данные любой формы, а также связывать их друг с другом. Но и типизированные строго определенные структуры объектов также хорошо поддерживаются BrightstarDB. Разработчикам приложений, таким образом, можно отображать несколько моделей в .NET и в нескольких базах данных BrightstarDB.

Результаты запросов и представления кэшируются автоматически для повышения производительности приложений с интенсивным использованием запросов к БД. Как правило, кэширование данных производится самими приложениями, т.е. реализуется разработчиком как отдельный функциональный блок, но BrightstarDB обеспечивает эту функцию как ключевую возможность.

BrightstarDB использует форматы хранения данных, которые сохраняют полную информацию об истории на каждом этапе транзакции. Это позволяет приложениям выполнять запросы данных в любой момент времени. Обеспечивается проверка состояния данных в хранилище данных, а также возврат в предыдущее состояние или снимок (snapshot), если ранее он был сделан. Такой подход конечно увеличивает объем используемого дискового пространства, но BrightstarDB предоставляет и возможность консолидировать данные, относящиеся только к текущему моменту времени.

Большинство разработчиков на .NET привыкли использовать объекты и LINQ для построения приложений. LINQ поддерживается практически в полной мере, включая типизированную модель данных. Причем, множество различных объектных моделей могут налагаться друг на друга в общей семантической модели данных. В BrightstarDB существует поддержка языка запросов SPARQL, а также экспорт/импорт данных в формате NTriples, используемые при построении семантических веб-приложений. В большинстве ORM изменения в объектной модели и в реляционной схеме должны осуществляться одновременно. RDF, в идеале, может быть использован для хранения и свойств-значений и отношений между объектами. Если объектная модель изменяется, то новое значение свойства может быть просто добавлено, так как нет фиксированной схемы. Аналогичным образом, если дополнительные

данные добавляется в хранилище в виде RDF, то объектная модель может либо их игнорировать, либо использовать эти данные.

BrightstarDB является СУБД с однократной записью, многократным чтением (WORM). Изменения в данные добавляются в конец файла хранилища, данные никогда не перезаписываются. Так как запись производится только одним клиентом, и данные не перезаписываются, то нет и необходимости в реализации блокировок. WORM-подход поддерживает откат и запросы по всей истории состояний базы данных на любом этапе транзакции. База данных может периодически консолидироваться, при этом происходит удаление ненужной истории, и это позволяет управлять ростом размера файла хранилища.

База данных РС создана на основе технологий семантического веба, реализованных в программном пакете BrightStarDB. Данные хранятся в виде графа, к графу реализован объектно-ориентированный доступ по стандарту Entity Framework. При этом обеспечивается как удобный инструмент реализации приложения, так и перспектива расширения методов анализа данных интеллектуальными методами на основе концептуальной модели предметной области.

1.7 Библиотека Nancy для создания интернет-приложений

Для реализации системы отслеживания действий пользователя на сайте в данном приложении был выбран фреймворк NancyFX. Главное преимущество данного фреймворка заключается в том, что маршрут, т.е. сам запрос определяется в конструкторе модуля. Чтобы определить маршрут в NancyFX, необходимо указать Method (метод), Pattern (шаблон) и Action (действие). Method (метод) – это метод HTTP, который используется для доступа к ресурсу. Nancy поддерживает следующие методы: DELETE, GET, HEAD, OPTIONS, POST, PUT и PATCH.

Метод GET запрашивает данные из указанного ресурса; POST отправляет данные, подлежащие обработке, на указанный ресурс; PUT заменяет все текущие представления ресурса данными запроса; DELETE удаляет указанный ресурс; HEAD запрашивает ресурс так же, как и метод GET, но без получения клиентом тела ответа; OPTIONS исполь-

зуется для описания параметров соединения с ресурсом; PATCHN используется для частичного изменения ресурса.

Pattern (Шаблон) объявляет URL-адрес приложения на который отвечает маршрут.

Action (Действие) - это программа, реализующая поведение, когда запрос сопоставляется с маршрутом.

Для примера можно привести запрос, представленный в листинге 1.1

Листинг 1.1 – Пример GET-запроса на NancyFX

```
1  public class HelloNancyFx : NancyModule
2  {
3      public HelloNancyFx()
4      {
5          Get["/"] = parameters => "Hello, NancyFX!";
6      }
7  }
```

В конструкторе данного класса есть строка 5, в которой реализуется GET-запрос, соответствующий корню сайта в браузере. В результате пользователю выдается сообщение «Hello, NancyFX!» в виде простого (plain) текста. Библиотека Nancy разрабатывалась как негромоздкий способ реализации функций интернет-приложения, более понятный и простой в отличие от ASP.NET. Кроме этого, NancyFX работает на Mono.

Приложение Nancy строится как набор таких отображений вида GET["<адрес>"], POST["<адрес>"] и т.д. При этом <адрес>, в общем случае, содержать структуры вида "<адрес>/{var}", где var – это динамическая переменная C#, куда поместится строка, стоящая в адресе после «/». Доступ к var, к данным формы, к параметрам GET- и POST-запросам и т.п. передается через переменную parameters и public-атрибут Request. Например, данные формы доступны в Request.form.<имя_поля>.

Результатом запуска методов HTTP является возвращаемый объект Response, кодирующий ответ сервера в том числе и передаваемый

клиенту гипертекст. При помощи данного объекта также осуществляется управление браузером пользователя, например, переадресация на другую страницу, хранение куки и т.п. Для генерирования гипертекста в Nancy реализован ряд механизмов заполнения шаблонов данными (шаблонизаторов). Но можно также подключать и другие технологии, в частности использованный в данной ВКР SHARP TAL.

1.8 Библиотека SharpTAL для создания HTML-страниц

В проекте использован шаблонизатор SHARP TAL, обладающий рядом важных преимуществ, в том числе, строгой изоляцией исполняемого кода от описания HTML-шаблона; все вычисляемые конструкции встраиваются внутрь XML-атрибутов, не влияя на базовую спецификацию XML (HTML), в которой представлен шаблон. Рассмотрим пример шаблона, представленного в листинге 1.2.

Листинг 1.2 – Шаблон SHARP TAL (пример)

```
1 <html>
2   <body>
3     <h1>Hello, ${"world"}!</h1>
4     <table>
5       <tr tal:repeat='row new string[] {"red","green","blue"}'>
6         <td tal:repeat='col new string[] {"rectangle","triangle"}'>
7           ${row} ${col}
8         </td>
9       </tr>
10    </table>
11  </body>
12 </html>
```

Для генерирования текста по этому шаблону нужно в параметры функции-интерпретатора передать шаблон и параметр – отображение имени на объект C#. В данном примере необходимо задать значение

переменной `world`, которое, является или строкой или должно поддерживать интерфейс преобразования к строковому значению. В строках 5 и 6 листинга 1.2 задается вложенных цикла, при помощи которых генерируется декартово произведение из элементов двух множеств объектов. Значения передаются в переменные цикла `row` и `col`. Циклы в шаблоне задаются при помощи TAL-выражений `tal:repeat`, структура этого выражения следующая:

`<переменная_цикла> <выражение C#>`,

причем `<выражение C#>` должно возвращать список значений или объект, поддерживающий интерфейс `IEnumerable`.

Подсистема веб-сервера сконструирована на основе пакетов `Nancy` и `SharpTAL`.

1.9 Идентификация сеанса пользователя в HTTP

Для того, чтобы пользователю давать наиболее адекватную рекомендацию, необходимо связать каждый сеанс взаимодействия пользователя с сайтом в один общий сеанс. Данная техническая проблема решается двумя способами. Первый способ – это разработать систему регистрации пользователей на сайте, второй – связать отдельные сеансы при помощи так называемых *записей куки*. Первый способ требует от пользователя проявить желание зарегистрироваться, и каждый раз как пользователь будет регистрироваться на сайте старые данные будут дополняться новыми. Для обеспечения функционирования сеансов с регистрацией также используются куки.

Второй способ позволяет создавать пользовательские учетные записи, привязанные только к куки. Рассмотрим использование куки на примере. Для доступа к странице `http://www.example.org/index.html`, браузер отправляет на сервер `www.example.org` следующий запрос (браузер → сервер):

```
GET /index.html HTTP/1.1
Host: www.example.org
```

Сервер отвечает, отправляя запрашиваемую страницу вместе с текстом, содержащим HTTP-ответ. Там может содержаться указание браузеру сохранить куки (браузер ← сервер):

```
1 HTTP/1.1 200 OK
2 Content-type: text/html
3 Set-Cookie: name = value
4
5 <Содержимое страницы>
```

Поле Set-cookie: (строка 3) отправляется лишь тогда, когда приложение на сервере сообщает браузеру команду на сохранение куки-значений. В этом случае, если куки поддерживаются браузером и их приём включён, браузер запоминает строку name=value и отправляет её обратно серверу с каждым последующим запросом. Например, при запросе следующей страницы `http://www.example.org/spec.html` браузер пошлёт серверу `www.example.org` следующий запрос (браузер → сервер):

```
1 GET /spec.html HTTP/1.1
2 Host: www.example.org
3 Cookie: name = value
4 Accept: */*
```

Этот запрос отличается от первого запроса тем, что содержит в строке 3 значения, которые сервер отправил браузеру ранее. Таким образом, сервер узнает, что этот запрос связан с предыдущим. Сервер отвечает, отправляя запрашиваемую страницу и, возможно, добавив новые куки. Значение куки может быть изменено сервером путём отправления новых строк Set-Cookie: name=newvalue. После этого браузер заменяет старое куки с тем же name на новую строку.

Куки также могут устанавливаться программами на языках типа JAVASCRIPT, встроенными в текст страниц, или аналогичными скриптами, работающими в браузере. В JAVASCRIPT для этого используется свойство cookie объекта document — `document.cookie`. Например, команда присвоения `document.cookie = "temperature=20"` создаст куки под именем «temperature» и значением 20.

Преимущества значений куки:

- ☐ Их очень легко использовать и реализовать;
- ☐ За отсылку данных отвечает браузер;
- ☐ Браузер автоматически сохраняет файлы куки посещаемых сайтов.

Ограничения технологии куки:

- ☐ данные хранятся в простом текстовом формате, поэтому никакая безопасность не гарантируется;
- ☐ ограничения на объем памяти данных значений – 4096 bytes (4KB);
- ☐ число хранимых значений ограничено, многие браузеры предоставляют возможность хранить 20 значений куки, и если будет отослан новое значений куки, то старое будет удалено; некоторые браузеры поддерживают до 300 значений куки.

Значения куки разделяются на два типа:

- ☐ Постоянные куки;
- ☐ Сеансовые куки или временные;

Сеансовые куки действуют на время работы пользователя с сайтом (активного сеанса) и хранятся в оперативной памяти браузера.

В отличие от сеансовых куки *постоянные куки* хранятся на клиентском жестком диске до тех пор, пока не истечет их срок хранения, который задается специальным образом, обычно это конкретная дата. Кроме того, куки могут быть удалены пользователем, для этого в браузерах есть специальные инструменты. Постоянные куки – это основной инструмент сбора определенной информации о пользователе и его операционной системе.

Выводы по разделу. Рекомендательные системы (РС) позволяют структурировать объекты в базе данных информационного ресурса (интернет-магазина) по степени релевантности к интересам того или иного пользователя (категории пользователей). Они позволяют решать задачи поиска новых закономерностей в таких данных, новые свойства и характеристики объектов.

РС активно применяются в интернет-магазинах, библиотеках текстового содержания (цифровых архивов статей научного учреждения или проекта, книг), а также на рынках недвижимости.

В области РС существует ряд фундаментальных и практических проблем, требующих решения как на этапе разработки РС, так и на этапе их эксплуатации:

- ❑ пользователи неохотно предоставляют информацию о себе и своих потребностях, либо разработчики РС уделяют мало внимания процессу информационного наполнения профиля пользователя;
- ❑ в предметных областях, связанных с большой стоимостью объекта или услуги (где принимаются серьезные решения по вложения материальных средств), информационные модели объекта и профиля пользователя сложны по своей структуре и связи компонент структуры, что требует явного представления концептуальной модели предметной области во время выполнения РС как своих основных функций, так и функций предсказания значений атрибутов объекта или профиля пользователя на основе прецедентов;
- ❑ для предыдущего пункта важным является также разработка пользовательского интерфейса, позволяющего в удобной для пользователя форме и достаточно гибко задавать запросы к РС, а также визуализировать результаты, предлагаемые РС;
- ❑ при практической реализации РС практически всегда необходимо решать проблему «холодного старта», т.е. обеспечивать функционирование информационной системы в режиме недостатка исходных данных о предпочтениях пользователей.

Таким образом РС, как системы поддержки принятия решения, являются типичным представителем систем *искусственного интеллекта*, ориентированными, прежде всего, на обработку неполной и противоречивой информации, а также использующими системы, основанные на формализованных знаниях (*knowledge-based systems*). Рекомендательные системы реализуются, как правило, в виде интернет-приложений, чтобы обеспечить максимальную доступность пользователю необходимой информации.

2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

2.1 Функциональное моделирование предметной области

Перед тем, как начать разработку программного продукта необходимо провести исследование предметной области автоматизации, на первом этапе, – это узнать какие функции должен будущий программный продукт выполнять.

На рисунке 2.1 изображена диаграмма UML «Варианты использования» (Use Case). С РС взаимодействуют пользователи, которым можно присвоить три роли: «покупатель», «риелтор» и «эксперт». Все функции, доступные для покупателя доступны и для риелтора и для эксперта. Для выявления функций произведен опрос пользователей. Задавались вопросы о том, какие действия выполняют пользователи (риэлторы и покупатели) совершают, чтобы достичь своих целей на сайтах продажи недвижимости. Роль эксперта появилась вследствие необходимости реализации функций кластерного анализа, что в свою очередь обусловлено невозможностью прямого применения метода коллаборативной фильтрации к решению данной задачи (см. раздел ??).

Покупатель, находясь на сайте РС, может просматривать объекты недвижимости. Для того, чтобы выдать информацию пользователю необходимо загрузить данные из базы данных. В это время информационная система собирает дополнительную информацию о том, интересен ли данный объект покупателю. Если пользователь находится на странице больше некоторого порогового значения времени, то система делает предположение, что покупатель положительно оценил данный объект недвижимости. Реализуется эта функция при помощи функции таймера, которая запускается в момент отображения страницы пользователю.

Если покупатель не покинул страницу в течение некоторого времени, то производится запись в базу данных о «положительной оценке» данного объекта, после чего запускается процедура пересчета оценок других объектов в данном классе объектов недвижимости. Предло-



Рисунок 2.1 — UML-Диаграмма вариантов использования

женный вариант функционирования основывается на умозрительном предположении, что покупателей интересуют только объекты недвижимости одного класса.

Роль риелтора дополняется функцией загрузки исходных данных об объектах недвижимости в базу данных РС. Процедура состоит из трех шагов. Сперва надо скачать базу данных с сайта продажи недвижимости, например, с <http://atlcom.ru/> (Атлант-недвижимость), и поместить информацию в файл. Затем, на втором шаге, надо выбрать этот же файл для загрузки в РС. Третий шаг – это запуск выполнения процесса загрузки данных.

Роль эксперта – это структуризация данных базы данных, которая, как правило, проводится после загрузки данных риелтором. Функции эксперта включают проведение кластерного анализа данных, что формирует начальное распределение объектов недвижимости по классам, и сами эти классы. Следующей функцией эксперта является «Означивание классов». При помощи этой функции эксперт просматривает элементы каждого кластера, проводить визуальный анализ (например,

сравнение объектов друг с другом), и задавать имена кластерам. Два кластера, обозначенных одним и тем же названием, объединяются в РС в один общий кластер, и дальнейшая обработка информации уже производится в рамках этого объединенного множества объектов недвижимости. Так же эксперт может редактировать накопленные значения оценок в базе данных.

В системе реализованы два программных *агента*. Агентами являются подсистемы, которые запускаются при возникновении некоторых событий, например, по факту попадания положительной оценки в базу данных, по факту просмотра пользователем объектов заданного кластера, а также вследствие действий эксперта.

Первый агент – это механизм слежения за пользователем, который запускается таймером на странице веб-браузера. Второй программный агент – это механизм пересчета оценок непросмотренных объектов, который запускается первым агентом или экспертом. Разделение этих агентов обуславливается тем, что не всякое срабатывание таймера должно обязательно запускать выработку рекомендаций, так как это процедура вычислительно емкая. Другой причиной является потенциальное разделение процессов работы сервера, слежения за пользователем и пересчета рекомендаций по разным ядрам микропроцессора сервера. Такое разделение преследует целью повысить скорость отклика системы за счет утилизации всех ресурсов процессора.

2.2 Архитектура системы

Архитектура системы (рисунок 2.2) реализует общую клиент-серверную схему взаимодействия интернет-приложений.

Предложения по недвижимости импортируются риелтором или администратором, которые хранятся в базе данных сайта типа Avito.ru. Пользователь взаимодействует с рекомендательной системой через веб-браузер (клиентская часть), а веб-сервер отвечает за хранение, обработку и преобразование информации к виду, воспринимаемому пользователем. База данных (БД) взаимодействует с блоком многомерного статистического анализа (МСА) данных, в котором находятся функции

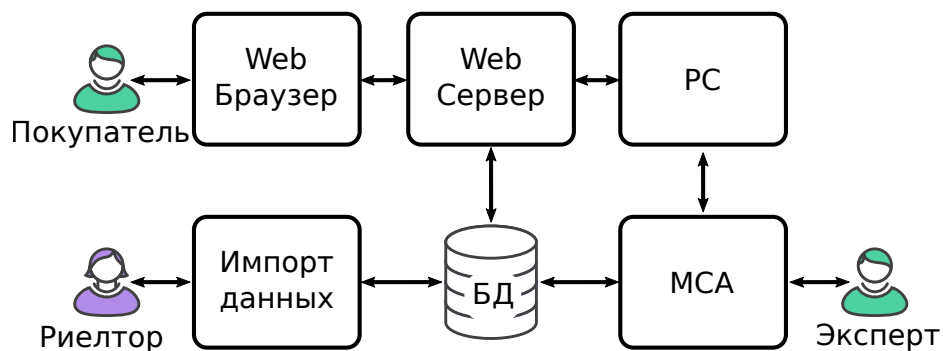


Рисунок 2.2 — Архитектура рекомендательной информационной системы

оценки схожести объектов, кластерный анализ и др. В рекомендательной системе (PC) находятся функции выработки рекомендаций для пользователя, рекомендации которой представляются в виде веб-страницы и передаются Web-сервером Web-клиенту.

Все данные, в том числе, учетные записи пользователей, сведения об объектах недвижимости, промежуточные данные расчетов, результаты расчетов и их интерпретация в виде наборов объектов и их оценок, хранятся в одной базе данных.

2.3 Проектирование структуры базы данных

Как было сказано в главе 1, база данных BrightstarDB поддерживает Entity Framework, что дает возможность решить несколько технических проблем одновременно:

- ❑ Проводить реализацию бизнес-логики приложения в терминах объектно-ориентированного проектирования без использования специального языка запросов;
- ❑ Представлять объекты, хранимые в базе данных, в виде объектов бизнес-логики, а не в виде записей базы данных;
- ❑ Обращаться к объектам целыми списками, например, проводить запросы и фильтрацию, запускать функции-агрегаты;
- ❑ Разрабатывать бизнес-логику независимо от свойств хранилища данных.

Процесс проектирования состоял из следующих шагов согласно требованиям Entity Framework:

1. Выделить объекты, представляющие предметную область;
2. Представить объекты в виде интерфейсов,
3. Представить объекты в виде набора свойств, свойствам задать типы;
4. Типы данных выбираются из стандартных (целое, строка и т.д.), перечисления и ссылки на другие интерфейсы объектов, хранимых в БД;
5. Представить отношения, интерпретируемые как один-ко многим в виде списков объектов.

В целом процедура схожа с применением метода построения ER-диаграмм Чена [5].

На рис. 2.3 представлена диаграмма классов (интерфейсов), спроектированные в рамках Entity Framework.

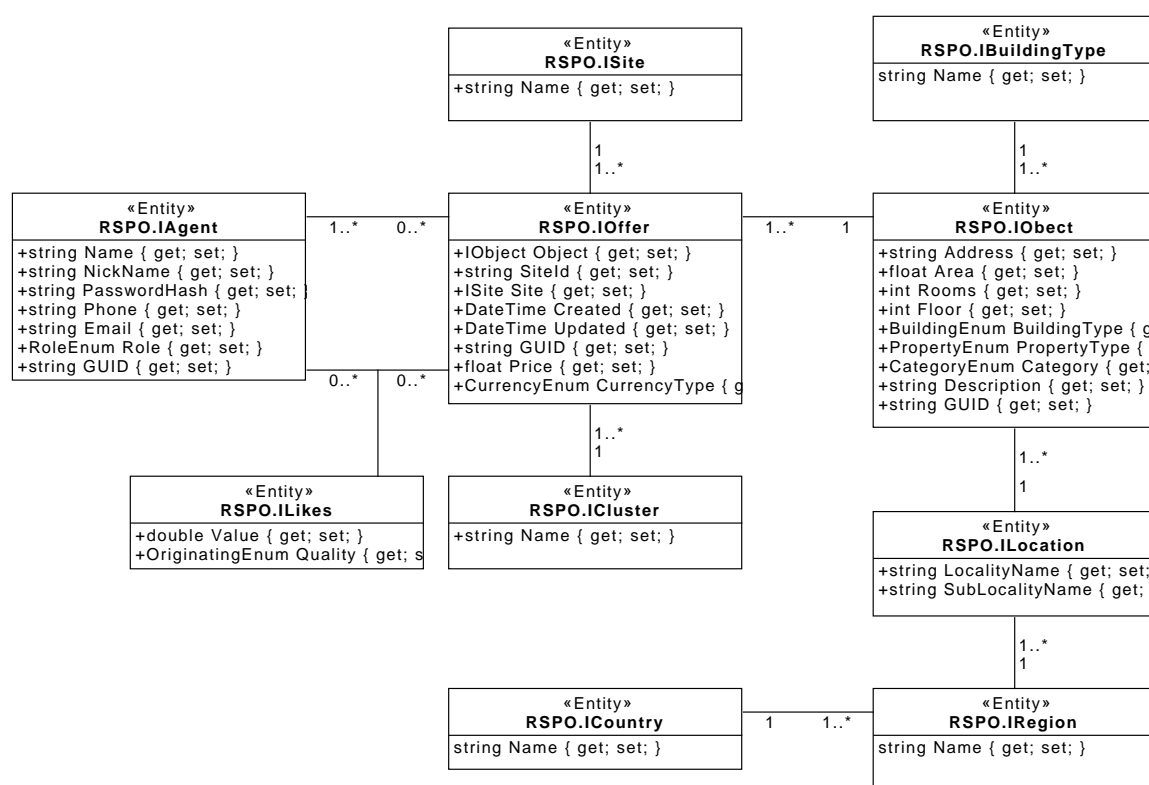


Рисунок 2.3 — Диаграмма классов рекомендательной системы (база данных)

В центре диаграммы находится интерфейс `IOffer`, представляющий предложение о продаже объекта недвижимости на рынке недвижимости. Данный класс содержит ссылку на объект недвижимости

`IObject`, его стоимость, ссылку на агента-риелтора `IAgent`. Фактически данный класс является параметром отношения многие-ко-многим между агентами и объектами недвижимости. Кроме того, предложение включает ссылку на сайт, где оно размещено и две даты — дата выставления на продажу и дата последнего изменения информации о предложении у данного риелтора.

Объект недвижимости `IObject` содержит информацию, на которую не влияет предложение, т.е. физические параметры объекта. В объекте представлена информация об адресе квартиры, который состоит из ссылки на справочник «местоположений» `ILocation` и строки адреса (улица, дом) внутри «местоположения»; ссылки на запись справочника типов зданий, перечислений фиксированного размера, характеризующих другие параметры объекта (материал, тип собственности, категория жилья), площадь жилья и т.д.

Предложения, объекты недвижимости и агенты получают дополнительно 128-битный идентификатор `GUID`, псевдослучайное число, которое с высокой степенью вероятности можно считать глобально уникальным. Идентификатор `GUID` генерируется на основе данных о рабочей станции, что несколько повышает надежность обеспечения уникальности. Использование дополнительного идентификатора помимо того, что создает `Entity Framework`, удобно для отслеживания экземпляров в процессе отладки приложения. Также `GUID` используется для передачи идентификатора объекта страницам, отображающим данный объект.

Местоположение (город, село, деревня и т.п.) `ILocation` представляет собой запись справочника и состоит из двух полей: название местности (например, города), и название части местности (например, района города). Объект ссылается на справочник регионов (областей) `IRegion`, который содержит название региона и, в свою очередь, ссылается на аналогичный справочник государств `ICountry`. Такое распределение адреса по справочникам взято из структур данных, используемых `Google` и `Yandex`, т.е. учтен опыт этих организаций в геокодировании.

Оставшиеся интерфейсы на диаграмме используются для хране-

ния данных, связанных с выработкой рекомендаций покупателям, т.е. связанными с основными функциями рекомендательной системы.

2.4 Импорт данных в хранилище

Особенностью рынка недвижимости является тот факт, что актуальные данные об объектах недвижимости опубликованы в свободном доступе: разработаны специальные форматы обмена данных и API-интерфейсы сайтов для доступа к этим данным. Ввиду достаточно высокой универсальности эти данные не содержат каких-либо характеристик релевантности к интересам клиентов.

Данные практически обо всех объектах базы данных загружаются с существующих сайтов. На сайтах есть и агенты-риелторы, и предложения и объекты недвижимости. Поэтому на 99% информационное наполнение базы данных РС осуществляется именно загрузкой данных с сайтов продаж квартир и других объектов недвижимости.

Обменный формат данных. Большинство сайтов по продаже недвижимости типа Avito.ru поддерживают экспорт данных в формате, разработанном фирмой Yandex. Формат представляет собой набор предложений, представленных в виде XML-структуры. Листинг 2.1 показывает пример представления данных об одном объекте недвижимости.

2.4.1 Подсистема импорта данных

Для того, чтобы загрузить импортируемые данные разработана соответствующая подсистема. Общая схема функционирования подсистемы состоит из следующих шагов:

1. Загрузка данных с сайта продажи недвижимости в локальный файл; как правило загружаемый сайт представляет XML, заархивированный каким-либо архиватором, например, ZIP.
2. Переход на страницу импорта данных в РС.
3. Выбор файла для импорта и загрузка его на сервер.
4. Запуск процедуры импорта.

Листинг 2.1 – Пример представления импортируемых данных об объекте недвижимости в формате Yandex.

```
<?xml version="1.0" encoding="utf-8"?>
<realty-feed xmlns="http://webmaster.yandex.ru/schemas/feed/realty/2010-06">
  <generation-date>2018-03-20T10:08:42+04:00</generation-date>
  <offer internal-id="3952-000348236">
    <type>продажа</type>
    <property-type>жилая</property-type>
    <category>Комната</category>
    <url>http://www.atlcom.ru/...</url>
    <creation-date>2018-03-20T00:00:00+04:00</creation-date>
    <last-update-date>2018-03-20T13:56:06+04:00</last-update-date>
    <manually-added>1</manually-added>
    <location>
      <country>Россия</country>
      <region>Иркутская область</region>
      <locality-name>Иркутск</locality-name>
      <sub-locality-name>Ленинский</sub-locality-name>
      <address>Ленинградская улица, ...</address>
    </location>
    <sales-agent>
      <category>владелец</category>
      <name>Атлант-Недвижимость</name>
      <phone>7499.....</phone>
    </sales-agent>
    <price>
      <value>750000</value>
      <currency>RUR</currency>
    </price>
    <area>
      <value>13</value>
      <unit>КВ.М</unit>
    </area>
    <image>http://...img.avito.st....jpg</image>
    <rooms>1</rooms>
    <rooms-offered>1</rooms-offered>
    <floor>5</floor>
    <floors-total>5</floors-total>
    <building-type>панельный</building-type>
    <building-series>секционка</building-series>
    <description>Телефоны, Фотографии, Описание от собственника
      доступны на сайте ATLCOM.RU</description>
  </offer>
  . . . . .
```

Все шаги, выполняются риелтором вручную, за исключением самой процедуры импорта.

Подсистема импорта данных реализована в виде класса `ImportFromAtlc` содержащего следующие `public`-методы:

`public string FileName` – имя импортируемого файла (временный файл на сервере);

`public Stream InputStream` – ссылка на входной поток байтов файла;

`public XDocument Document` – документ библиотеки `XDocument`, подобный структуре `DOM2`;

`public void Import(bool onlyLoad=false)` – метод, осуществляющий импорт данных, его единственный параметр указывает, надо ли делать импорт или просто попытаться загрузить данные в `Document`;

`public static Dictionary... buildingTypes` – словарь отображения строковых значений на перечисление типов строений;

`public static Dictionary... categoryTypes` – словарь отображения строковых значений на перечисление видов категорий;

`public static Dictionary... roles` – словарь отображения строковых значений на перечисление ролей агентов;

`public static string GetGUID()` – генератор идентификаторов `GUID`.

Объект-импортер после загрузки данных в атрибут `Document` делает запрос на перечисление узлов дерева, соответствующих тегу `<offer>`, корневой тег игнорируется. Запрос делается при помощи метода `Descendants`:

```
private XElement GetFirstElement(XElement e, string tagName)
{
    return e.Descendants(YName(tagName)).First();
}

protected XName YName(string name)
{
    return XName.Get(name, YandexNS);
}
```

Здесь `e` – это узел дерева, к которому производится запрос, `YName` –

вспомогательный метод, который создает объекты, отождествляемые с тегами заданного пространства имен XML YandexNS.

В цикле по каждому результату запроса выполняется процедура загрузки данных (текст сокращен):

```
protected void ProcessProposal(XElement input, ISite site)
{
    MyEntityContext ctx = Application.Context; // получение доступа к БД
    . . . . .
    // Создание объектов для предложения и объекта недвижимости.
    IOffer offer = ctx.Offers.Create(); offer.GUID = GetGUID();
    IObject obj = ctx.Objects.Create(); obj.GUID = GetGUID();

    offer.Object = obj; // Связывание объекта с предложением.
    offer.SiteId = internalId.Value;
    offer.OfferType = GetOfferType(input); // загрузка типа предложения
    offer.Site = site;

    offer.Created=GetDateTime(input, "creation-date");
    offer.Updated=GetDateTime(input, "last-update-date");

    GetSalesAgent(offer, input); // Поиск агента в БД, если есть,
    // если нет, то создать его.

    obj.PropertyType = GetPropertyType(input);
    obj.Category = GetCategoryType(input);
    obj.URL = WebUtility.UrlDecode(GetText(input, "url"));

    GetLocationData(obj, input);
    GetPrice(obj, input);
    . . . . .
    try {
        obj.Floor=int.Parse(GetText(input, "floor"));
    } catch (InvalidOperationException) {
        obj.Floor=-1000;
    }

    try {
        obj.FloorTotal=int.Parse(GetText(input, "floors-total"));
    } catch (InvalidOperationException) {
        obj.FloorTotal=-1000;
    }
}
```

```

    }

    try {
        obj.BuildingType=GetBuildingType(input);
    } catch (InvalidOperationException) {
        obj.BuildingType=BuildingEnum.Unknown;
    };
    try {
        obj.BuildingSeries=GetBuildingSeries(input);
    } catch (InvalidOperationException) {
        obj.BuildingSeries = null;
    }
    obj.Description=GetText(input, "description");

    ctx.Add(obj);
    ctx.Add(offer);
    // Затем закрывается транзакция.
}

```

Импорт данных для рынка Иркутской области на лето 2018 года составляет более 5600 предложений.

2.5 Проектирование интернет-приложения

Интернет-приложение реализуется при помощи класса `WebModule`. Каждому HTTP-запросу сопоставляется лямбда-функция с одним параметром `parameters`, в которой содержится параметр шаблона адреса URL. Реализация функции запрограммирована в соответствии с шаблоном проектирования Model–View–Controller [25].

Каждый метод класса вначале пытается восстановить сессионный объект, затем создает или загружает модели и их представления, затем генерирует HTML. Рассмотрим несколько примеров обработки запросов.

```

namespace RSP0
{
    public class WebModule : NancyModule
    {
        public WebModule() : base()
    }
}

```

```

{
    Get["/" ] = parameters => // главная страница
    {
        RestoreSession(); // восстановление сессии
        ApplicationModel appModel = new ApplicationModel(Application.APPLICATION_NAME)
        // генератор HTML на технологии SharpTAL
        return Render("index.pt", context: appModel,
            view: new ApplicationView(appModel, CurrentSession)); //предст
    };

    Get["/objs"] = parameters => // Это страница сайта с квартирами.
    {
        RestoreSession();

        ObjectList objList = new ObjectList(); // модель
        ObjectListView objView = new ObjectListView(objList); // представление
        return Render("objlist.pt", context: objList, view: objView);
    };

    Get["/offers"] = parameters => // предложения
    {
        RestoreSession();
        OfferList model = new OfferList(null);
        OfferListView view = new OfferListView(model);
        return Render("offerlist.pt", context: model, view: view);
    };

    Get["/offers/{clid}"] = parameters => // предложения в заданном кластере
    {
        int clid = int.Parse(parameters.clid); // получить идентификатор кластера
        RestoreSession();
        OfferList model = new OfferList(clid: clid); // выбрать объекты
        OfferListView view = new OfferListView(model);
        return Render("offerlist.pt", context: model, view: view);
    };

    Get["/offer/{GUID}"] = parameters => // страница отображения квартиры
    {
        RestoreSession();
        string GUID = parameters.GUID; // получить идентификатор квартиры
        // найти квартиру в БД
    }
}

```

```

IOffer model = Application.Context.Offers.Where(x => x.GUID == GUID).FirstOr

string msg = "Объект (Offer) не найден!: " + GUID;
if (model == null)
{
    Console.WriteLine(msg);
    return "msg";
}
else Console.WriteLine(model);

OfferView view = new OfferView(model);
Nancy.Response response = Render("offer.pt", context: model, view: view);

addLike(CurrentSession.Agent, view.Object);

return response;
};

Get["/sneak/{GUID}"] = parameters => // получить рекомендации
//Этот метод грузится из JQuery (AJAX)
{
    RestoreSession();
    string GUID = parameters.GUID;
    IOffer offer = Application.Context.Offers.Where(x => x.GUID == GUID).FirstOr

    if (offer == null)
    {
        return Render("nosneak.pt"); // нет рекомендаций
    }

    OfferList model = new OfferList(clid: null, like: offer, session: CurrentSes
    OfferListView view = new OfferListView(model);

    return Render("sneak.pt", context: model, view: view);
};

. . . . .

// прием данных пользователя из формы регистрации
Post["/login0"] = parameters =>
{
    RestoreSession();

```

```

LoginObject model = new LoginObject();
LoginView view = new LoginView(model, this.Request, CurrentSession);

Response response = null;
bool res = view.Process();

CurrentSession = view.Session; // Обновление сессии
if (res)
{
    response = Response.AsRedirect("/");
}
else // Неуданая идентификация
{
    response = Response.AsRedirect("/login");
}
// Перенаправить браузер на домашнюю страницу.
return InSession(response);
};
. . . . .
}

```

Лямбда-функции реализуются при помощи методов объекта-сервера, один из самых важных - это `Render`, который заполняет объект `Response` содержимым страницы. Для создания HTML-страницы надо в этот метод передать файл-шаблон и два объекта – модель и ее представление. Из этих объектов, а также из объекта `Response` берется информация для отображения. SharpTAL исходный код шаблона компилирует в код C#, который потом компилируется в CIL. Этот процесс достаточно длительный, поэтому желательно проводить кэширование шаблонов, а именно их скомпилированного кода.

```

public Nancy.Response Render(string templateFile, // Шаблон SharpTAL
                             object context = null, // модель
                             object view = null) // представление
{
    . . . . .
    if (Application.USE_TEMPLATE_CACHE) // Механизм кэширования шаблонов
    {
        gotCache = Application.templateCache.TryGetValue(templateFile,

```



```

        out template);
    }

    // В режиме отладки иногда удобно, если при каждом запросе
    // шаблон заново верстается.
    // Не надо сервер перезапускать при изменении шаблона.

    if (!gotCache)
    {
        // Компилирование шаблона
        . . . . .
    }

    var dict = new Dictionary<string, object>(); // словарь объектов,
        // доступных из шаблона по именам.

    if (this.Request != null)
    {
        Request = this.Request;
        dict.Add("request", Request);
    }
    if (context != null)
    {
        dict.Add("model", context);
    }

    if (view == null) throw new RenderException("null view");

    dict.Add("view", view);
    dict.Add("application", Application.APPLICATION); // модель приложения
    dict.Add("appview", new ApplicationView(Application.APPLICATION,
        CurrentSession));
    . . . . .
    dict.Add("message", message); // сообщение пользователю.
    dict.Add("user", CurrentSession.Agent); // Данные о пользователе.
    . . . . .
    string result = template.Render(dict); // запуск SharpTAL
    CurrentSession.Remove("message"); // убрать сообщение из сессии.
    return InSession(result); // оснастить сессию
}

```

Все методы HTTP, реализованные в серверном объекте должны

либо а) возвращать объект HTML-страницы, либо б) перенаправлять веб-браузер на другую страницу, либо в) сообщать об ошибке, например, 404 (Not found).

2.5.1 Шаблоны SharpTAL

Созданные для HTML-страниц шаблоны SharpTAL строятся на основе одного общего шаблона, что позволяет один раз настроить все технологии управления процессом отображения, например, CSS (Cascading Style Sheets), а также общим форматом страниц, размещением элементов, и затем просто использовать эту общую настройку на страницах, относящимся к конкретным адресам сайта (URL). Рассмотрим пример страницы отображения информации об объекте недвижимости.

Сначала надо оформить преамбулу шаблона.

```
1 <!DOCTYPE html>
2 <html lang="en"
3     xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:tal="http://xml.zope.org/namespaces/tal"
5     xmlns:metal="http://xml.zope.org/namespaces/metal"
6     xmlns:i18n="http://xml.zope.org/namespaces/i18n"
7     xmlns:zmetal="http://example.org/nothing"
8     metal:import="@TEMPLATEDIR@page.pt"
9     metal:use-macro='macros["main"]'
10 >
```

В преамбуле в строках 9–10 импортируется общий шаблон и выбирается макрос «страница» в этом шаблоне в качестве той, которую будем генерировать. Один шаблон может содержать много макросов, в том числе и вложенных. Замечательное свойство технологии TAL (ZPT, Zope page tanplets) состоит в том, что, если в качестве основного шаблона используется готовая страница, разработанная дизайнером, то для того, чтобы использовать плоды его труда не надо разрезать эту страницу на кусочки. Можно просто дополнительно ее разметить в рамках технологии ZPT.

```
11 <metal:tag metal:fill-slot="head"><!-- Hello --></metal:tag>
12
13 <metal:tag metal:fill-slot="content">
14     <h1 tal:content="view.Title">Информация по квартире</h1>
```

В этом куске кода указывается, что в макросе надо заполнить (заменить) два слота содержимым тега `<metal:tag>`. Содержимое же также динамически генерируется. В строке 14 команда подставляет в тег текст — заголовок страницы, который устанавливается в объектах—представлениях.

```
15      <span tal:omit-tag="default"
16          tal:define="obj view.Object; objview view.ObjView">
```

Здесь производится определение двух новых переменных, обозначающих объекты недвижимости, на которые ссылается отображаемое предложение.

```
17      <table width="100%">
18          <tr>
19              <td>
20                  <metal:tag metal:use-macro='macros["table"]'>
21                      <metal:tag metal:fill-slot="header">
22                          </metal:tag>
```

Взять заготовку (шаблон) таблицы и использовать его для отрисовки данных по квартире. Таблица состоит из двух столбцов — наименование атрибута и его значение.

```
23      <metal:tag metal:fill-slot="body">
24          <tr>
25              <th>Объект</th>
26              <th tal:content="objview.RuCategory">Квартира</th>
27          </tr>
28          <tr>
29              <th>Комнат</th>
30              <td tal:content="objview.ROR" class="text-sm-left">9к</td>
31          </tr>
32          <tr>
33              <th>Площадь</th>
34              <td tal:content="obj.Area">190</td>
35          </tr>
36          <tr>
37              <th>Район</th>
38              <td title="Завеличи (Псков)"
39                  tal:attributes="title obj.Location.SubLocalityName"
40                  ><strong tal:content="obj.Location.SubLocalityName" class="rspos-blue">Свердловский
41                  </td>
42          </tr>
43          <tr>
```

```

44         <th>Адрес</th>
45         <td tal:content="obj.Address" class="text-sm-left">Маршала Конева улица...</td>
46     </tr>
47     <tr>
48         <th>Материал</th>
49         <td tal:content="objview.RuType">Секвойя</td>
50     </tr>
51     <tr>
52         <th>Этаж</th>
53         <td tal:content="objview.FOF">Этаж/Этажи</td>
54     </tr>
55     <tr>
56         <th>Цена</th>
57         <td class="text-sm-left"><strong tal:content="objview.Price" >1 000 000</strong></td>
58     </tr>
59 </metal:tag>

```

Рисуем фотографию объекта недвижимости. Фотографии имеют разный размер, поэтому тег генерируется специальной подпрограммой в представлении.

```

60     </metal:tag>
61 </td>
62 <td>
63     <span tal:replace="structure view.ImageTag">
64     </td>
65 </tr>
66 </table>
67 </span>
68 <div id="rs-table">
69 </div>
70
71 <hr/>

```

Здесь приводится код, порождающий по таймеры запрос на сервер о том, что покупатель заинтересовался данным предложением, т.е. просматривает его больше некоторого времени. В качестве ответа браузер получает список рекомендованных квартир.

```

72 <style>
73     .ajax_loader i{
74         left:50%;
75         top:50%;
76     }
77 </style>
78 <div id="result-display" tal:attributes="data-guid model.GUID">

```

```

79         <!-- Здесь появится ответ от сервера - таблица рекомендаций -->
80         <div class="ajax_loader"><i class="fa fa-spinner fa-spin fa-2x"></i></div>
81     </div>
82 </metal:tag>
83 <metal:tag metal:fill-slot="scripts">
84     <script type="text/javascript">

```

Скрипт запроса сервера. В запросе передается GUID объекта, а GUID пользователя берется из сессии.

```

85     $(document).ready(function(){
86         var guid = $("#result-display").data("guid");
87         setTimeout(function () {
88             $.ajax({
89                 method:"GET",
90                 url: "/sneak/"+guid,
91                 success: function(resdata, textStatus, jqXHR) {
92                     $("#result-display").html(resdata);
93                 },
94                 error: function(jqXHR, textStatus, errorThrown) {
95                     alert("Error: "+textStatus+" "+errorThrown);
96                 },
97                 dataType: "html"
98             });
99         }, 2000);
100     });

```

Закрытие слота «основной текст страницы» в основном ZPT-макросе.

```

101     </script>
102 </metal:tag>
103 </html>

```

Из данного примера видно, что разделение процесса порождения HTML по функциям на объекты Model–View–Controller позволяет эффективно, по критерию выразительности, реализовать все его элементы.

2.6 Реализация подсистемы рекомендаций

Основное свойство разработанной информационной системы является ее функция выработки рекомендаций в виде списка объектов недвижимости, которые могли бы потенциально заинтересовать покупателя. Выработка рекомендаций как было сказано в разделе 1.3.1

осуществляется при помощи гибридного подхода – объединения коллаборативной фильтрации и структуризации множества объектов недвижимости. Процедура коллаборативной фильтрации реализована на основе метода Slope One, а структуризация – при помощи метода иерархического кластерного анализа.

Метод Slope One выбран по двум причинам: а) прост в реализации; б) накоплен большой опыт в его эксплуатации. В качестве метода иерархического кластерного анализа выступает агломеративный метод, т.к. в среде C#, в отличие от Python и Java, реализовано не так много библиотек многомерного статистического анализа данных. В разработке пришлось использовать то, что было найдено. Для того, чтобы выработать рекомендацию необходимо постоянно собирать информацию о желаниях покупателя и периодически обновлять промежуточные данные, на основе которых вычисляются рекомендации.

Общая стратегия сборки информации для выработки рекомендаций следующая:

- ❑ Каждый пользователь, посещающий сайт, проходит регистрацию. При первом посещении его браузер связывается с системой таким образом, чтобы при новом посещении восстанавливать предыдущую историю действий пользователя.
- ❑ Если пользователь желает, то он может зарегистрироваться полноценно, т.е. с вводом некоторой информацией о себе и получением учетной записи. В этом случае при посещении сайта из другого места (в другом браузере, на другом компьютере) восстановятся данные его прошлых сеансов.
- ❑ Все множество объектов разбивается на подмножества по схожести элементов друг с другом, например, выделяются отдельно продаваемые квартиры, комнаты, дома, участки, гаражи и т.п.
- ❑ Новому пользователю в начале представляется список из небольшого набора объектов недвижимости, состоящий из объектов, принадлежащих разным подмножествам. По реакции пользователя (просмотр конкретных экземпляров), затем, определяется, какой класс недвижимости его интересует.
- ❑ Каждый объект недвижимости, просмотренный пользователем

в течение некоторого времени, считать «понравившимся» и добавлять положительную оценку («лайк») в базу данных, связывая пользователя и предложение (интерфейс ILikes).

- При просмотре покупателем данных объекта недвижимости снизу экрана предлагать список рекомендуемых объектов.

Выработка рекомендаций на основе собранной информации делается следующим образом:

1. Если собрано достаточно информации для применения метода Slope One, то применяя этот метод построить список предложений. Если список полученных предложений включает более 20 штук, то закончить процесс выработки рекомендаций.
2. В другом случае, т.е. если количество предложений недостаточно, то дополнить список рекомендаций записями, схожими с просматриваемой (до 20 штук).

Коллаборативная фильтрация осуществляется только на тех объектах, за которые проголосовал хотя бы один покупатель. Поэтому на начальном этапе просто может не оказаться таких объектов в подмножестве интереса покупателя. Чтобы РС выполняла свои функции, то именно по этой причине необходимо использовать второй шаг.

2.6.1 Подсистема слежения за пользователем

Итак, основной функцией разрабатываемого веб-приложения являются сбор информации об интересах покупателя: покупатель, находясь на сайте и просматривая перечень объектов, сообщает системе свои намерения. РС, затем, производит анализ полученных данных. Рекомендации строятся в виде подмножеств объектов, просмотренных пользователями данного класса, т. е. на основе метода коллаборативной фильтрации.

Идентификация пользователя. При первом подключении к сайту, пользователю присваивается уникальный идентификатор GUID и куки, если пользователь уже зарегистрирован на сайте, то его GUID является значением куки. GUID используется абсолютно для всех пользователей,

зарегистрированных или анонимных. Напомним, что GUID (Globally Unique Identifier) – это уникальный, статический 128-битный идентификатор. Главная особенность GUID – это его уникальность, которая позволяет создавать расширяемые сервисы и приложения без опасения конфликтов в случае совпадения идентификаторов. Хотя уникальность каждого отдельного GUID, в принципе, не гарантируется, потому что общее количество уникальных ключей настолько велико (2^{128}), что вероятность того, что в мире будут независимо сгенерированы два совпадающих ключа, крайне мала. В момент подключения к сайту также создается и сессия для пользователя, которая при помощи метода `protected Nancy.Response InSession` добавляет текущую сессию в словарь всех сессий. При подключении к любой странице сайта запускается тот или иной обработчик GET/POST-запроса, в теле обработчика которого присутствует метод `RestoreSession()`, восстанавливающий предыдущую сессию.

Восстановление сессии. Метод `protected void RestoreSession()` отвечает за восстановление сессии по кукам. При первоначальном подключении пользователя к сайту, ему присваивается пустое строковое значение `string value = ""`; , которое далее проверяется с помощью конструкции `try-catch`. В блоке `try` в значение `value` смотрим куки и если, они существуют в данном подключении, то восстанавливаем предыдущую сессию пользователя. В случае если значение оказывается пустым, то создаем новую сессию с присвоением GUID. Если пользователь зарегистрирован на сайте, то восстановление его сессии происходит с помощью GUID. Далее производится поиск такого GUID в базе банных и если такой GUID уже имеется, то происходит восстановление предыдущей сессии со всеми запросами, которые когда-либо делал данный пользователь на данном сайте.

Следующий фрагмент программного кода демонстрирует процедуру восстановления сессии.

```
protected void RestoreSession()  
{  
    MyEntityContext ctx = Application.Context;
```



```

string GUID = "";
// попытка восстановит сессию (найти нужный куки в запросе)
try
{
    GUID = this.Request.Cookies[IN_SESSION_COOKIE_NAME];
}
catch (KeyNotFoundException) // попытка не удалась: куки не найден
{
    GUID = "";
}

try
{
    CurrentSession = activeSessions[GUID]; // найти сессию среди
    // других сессий по идентификатору GUID.
}
catch (System.Collections.Generic.KeyNotFoundException) // неудача
{
    // создание объекта-сессии.
    CurrentSession = new SessionModel();
    CurrentSession.GUID = ImportFromAtlcomru.GetGUID();
}
bool a = CurrentSession.Valid; // догрузка данных о пользователе из БД.
}

```

Создание сессии и регистрация куки. Для создания сессии необходимо присвоить какой-нибудь уникальный идентификатора пользователю сайта, так как некоторые пользователи могут быть не зарегистрированы и для сохранения их результатов запроса необходимо использовать данные методы. Для начала создадим метод, который будет присваивать куки для каждого пользователя, посетившего данный сайт:

```

protected Nancy.Response InSession(Nancy.IResponseFormatter response = null)
{
    if (response == null) response = this.Response;
    return InSession((Nancy.Response)response);
}

```

```

protected Nancy.Response InSession(Nancy.Response response = null)
{
    if (response == null) throw new Exception("null response object");

    bool a = CurrentSession.Valid; // этот запуск свойства создает
    // анонимную учетную запись
    activeSessions[CurrentSession.GUID] = CurrentSession;

    return response.WithCookie(IN_SESSION_COOKIE_NAME,
        CurrentSession.GUID, DateTime.Today.AddYears(1));
    // Установить "исчезновение" куки через год.
}

```

В нашем случае, срок хранения куки файлов составляет один год. Куки хранят информацию об аутентификации пользователя, персональные предпочтения и настройки пользователя, отслеживают состояние сеанса доступа пользователя. Далее рассмотрим, как происходит регистрация пользователя на сайте.

Регистрация пользователя. На сайте присутствует возможность регистрации пользователя. Данные, которые вводятся в поля регистрации пользователя, отвечает отдельный интерфейс `IAgent`. Рассмотрим `IAgent` подробнее.

Интерфейс `IAgent`. Интерфейс `IAgent.cs` определяет основные свойства объектов, представляющих пользователя. По структуре этого интерфейса порождаются объекты `Agent`, хранимые в Entity Framework — базах данных, в частности BrightstarDB. В листинге 2.2 представлен подробно `public interface IAgent`.

```

string Name — отвечает за имя пользователя,
string NickName — логин пользователя,
string PasswordHash — хэш пароля пользователя,
string Phone — номер сотового телефона пользователя,
string Email — электронная почта пользователя,
RoleEnum Role — роль пользователя в системе: пользователь может зарегистрироваться как «Агент продаж», «Покупатель», «Эксперт»;

```

Листинг 2.2 – Интерфейс регистрации пользователя

```
public interface IAgent
{
    string Name { get; set; }
    string NickName { get; set; }
    string PasswordHash { get; set; }
    string Phone { get; set; }
    string Email { get; set; }
    RoleEnum Role { get; set; }
    string GUID { get; set; }
    [Ignore]
    bool Valid { get; }
}
```

string GUID – уникальный идентификатор пользователя;

bool Valid – флаг анонимности пользователя.

После регистрации, пользователю выводится сообщение об успешной регистрации в системе. Для него создается новая сессия и объект пользователя в сессии `Session.Agent=agent;` и уникальный идентификатор его сессии `Session.GUID = user.GUID;`. Все зарегистрированные пользователи хранятся в базе данных, список которых можно посмотреть, как представлено на рисунке 2.4.

[ГЛАВНАЯ](#)
[ПРЕДЛОЖЕНИЯ](#)
[АГЕНТЫ](#)
[АНАЛИЗ](#)
[ВОЙТИ](#)

| Агент | Ник | Тел | E-mail | Роль | GUID |
|----------|------------------|-----------------|--------|-------------|--------------------------------------|
| Татьяна | | 8 902 511-59-95 | | агентство | 6b0c826f-0f30-4a87-9c05-5bbc449b8f30 |
| Татьяна | | 8 902 515-06-09 | | агентство | fd27ba28-53bf-4452-98d4-7b2a2176eeab |
| dfde68bd | eugeneaidfde68bd | | | покупатель | dfde68bd-2efd-4f88-94e7-eb0e0db837fa |
| | | | | неизвестный | 98ec3614-44aa-4f88-b554-4eb27741cba1 |
| | | | | неизвестный | e45bfbcb-a60c-4391-bf0b-d8f2e076c9f3 |
| | eugeneai1 | | | покупатель | ee282a60-1608-4129-8873-1928d89428d8 |
| 2 3 1 | eugeneai | 34234234 234 23 | | покупатель | 062a4739-1a5e-4bf0-a771-0458b5b126b3 |
| | | | | неизвестный | 8eb9dd01-1afb-4ae2-a39a-e1105dffbb55 |
| P-Маркет | | 8 902 512-65-83 | | агентство | b7cb6316-3747-4f0f-8782-1f933282fec8 |
| | | | | неизвестный | 3e9cebef-b5e1-42cd-8107-75093568c1ed |
| | | | | неизвестный | 2d479bda-05f4-4fab-84dc-721f28859d9f |

Рисунок 2.4 — Список пользователей, включая анонимных

Объекты сессии Объектом сессия является словарь следующего вида:

```
public class SessionModel : Dictionary<string, object> // словарь,  
    //отображающий строку на объект C#.  
{  
    public SessionModel(): base() {}  
    // Отдельно для удобства выделены некоторые значения словаря.  
    public string GUID  
    {  
        set  
        {  
            this["GUID"] = value;  
        }  
  
        get  
        {  
            return (string) this["GUID"];  
        }  
    }  
  
    public bool Valid // восстановление данных сессии.  
    {  
        get  
        {  
            IAgent a=Agent; // свойство  
            return a.Role != RoleEnum.Invalid &&  
                a.Role != RoleEnum.Unknown;  
        }  
    }  
  
    public IAgent Agent // Agent in the session  
    {  
        get  
        {  
            object o = null;  
            bool s = TryGetValue("agent", out o);  
            if (s) return (IAgent) o;  
            return createAnonymousAgent();  
        }  
        set  
        {
```

```

        this["agent"] = value;
    }
}

private IAgent createAnonymousAgent()
{
    MyEntityContext ctx = Application.Context;
    if (GUID == null)
    {
        GUID = ImportFromAtlcomru.GetGUID();
    }
    else
    {
        // попытка загрузки из БД
        IAgent agent = ctx.Agents.Where(x => x.GUID == GUID).FirstOrDefault();
        if (agent != null) Agent = agent;
    }

    // создание анонимного пользователя
    IAgent anonym = ctx.Agents.Create();
    anonym.GUID = GUID;
    anonym.Role = RoleEnum.Unknown;
    this.Agent = anonym;
    ctx.Add(this.Agent);
    ctx.SaveChanges();

    return this.Agent;
}
. . . . .
public SessionModel Invalidate() // отмена сессии
. . . . .
}

```

Авторизация пользователя. Авторизация пользователя реализована в классе `Server`, в лямбда-функции запроса `Post["/login"]`, принимаем данные пользователя из формы регистрации и в случае успешной авторизации перенаправляем пользователя на домашнюю страницу сайта (листинг 2.3).

Листинг 2.3 – Авторизация пользователя на сайте

```
Post["/login"] = parameters =>
{
    RestoreSession();

    LoginObject model = new LoginObject();
    LoginView view = new LoginView(model,
        this.Request, CurrentSession);

    Response response = null;
    bool res = view.Process();

    CurrentSession = view.Session;
    if (res)
    {
        response = Response.AsRedirect("/");
    }
    else
    {
        response = Response.AsRedirect("/login");
    }
    return InSession(response);
};
```

При выполнении регистрации пользователя используется специальная модель и представление:

```
public class LoginView : View<LoginObject>
{
    public LoginView(LoginObject context, Nancy.Request request,
                     SessionModel session) : base(context)
    {
        . . . . .
        protected bool UserBad(string message)
        {
            Session.Invalidate();
            Session["message"] = error(message + ", однако.", msg: "Неуспешная идентификация");
            return false;
        }

        public bool Process()
        {
            MyEntityContext ctx = Application.Context;
            . . . . .
            // Достать данные из формы
            . . . . .
            IAgent user = ctx.Agents.Where(x => x.NickName == nick).FirstOrDefault();
            MessageModel success = null;

            if (register != null && user != null)
            {
                return UserBad("Пользователь уже зарегистрирован");
            }
            else if (register == null && user == null)
            {
                return UserBad("Пользователь не найден");
            }
            else if (register != null && user == null)
            {
                // Проверки правильности данных и регистрация в БД.
                . . . . .
                user.NickName = nick;
                user.Email = request.Form.email;
                ctx.Add(user);
                ctx.SaveChanges();
                success = info("Теперь вы зарегистрированы в системе." +
                             "Можно начинать бояться.",
```

```

        msg: "Успешная регистрация");
    }
    else // register == null && user != null
    . . . . .
    // Уточнение данных текущей сессии
    Session.Agent = user;    // Объект пользователя в сессии
    Session.GUID = user.GUID; // Идентификатор сессии пользователя.

    Session["message"] = success;

    return true;
}

public void Logout()
. . . . .
protected static string SALT = "$2a$10$.lvjuUJj9nor/DArhPtrgu";
// BCryptHelper.GenerateSalt();
. . . . .
public new string Title = "Идентификация пользователя";

```

На рисунке 2.5 изображен пример страницы регистрации пользователя.

В РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА ПО РЫНКУ НЕДВИЖИМОСТИ ИРКУТСКОЙ ОБЛАСТИ
Идентификация Пользователя

ГЛАВНАЯ ПРЕДЛОЖЕНИЯ АГЕНТЫ АНАЛИЗ ВОЙТИ

Неуспешная идентификация! Пароли не совпадают, однако. ×

Идентификация пользователя

Пользователь Пароль

Пароль

Фамилия Имя Отчество

Адрес электронной почты Телефон

Ваш email и телефон мы будем использовать только для связи с вами.

☒ Я согласен, что за мною будут следить на этом сайте. ☐ Я - риэлтор!

Рисунок 2.5 — страница входа и регистрации пользователя

2.6.2 Реализация алгоритма Slope One

Полный текст реализации находится в приложении, здесь приведем только процедуру вычисления оценки интереса пользователя к непросмотренному им объекту недвижимости. Метод возвращает либо оценку (число с поавоющей запятой), либо `null`, если такую оценку сделать не представляется возможным.

```
public double? Estimate(IAgent agent, IObject obj)
{
    if (m == null) throw new EstimationException("matrix is not constructed");
    int ia, io;
    if (!agents.TryGetValue(agent, out ia)) // найти агента среди
        // входных данных
    {
        return null;
    }
    if (!objects.TryGetValue(obj, out io)) // ... квартиру
    {
        return null;
    }

    // найдены, значит оценка и так известна.
    if (m[ia, io] > 0)
    {
        return m[ia, io];
    }

    double value = 0;
    int num = 0;

    // Цикл по всем объектам во входных данных
    for (int row = 0; row < objects.Count; row++)
    {
        double a = d[io, row];
        if (!(a > 0)) continue;
        double theirest = m[ia, row]; // [Моя] оценка квартиры
        if (!(theirest > 0)) continue;
        theirest += a; // Добавить среднюю оценку.
        int cnt = v[row]; // количество лайков других пользователей
        num += cnt;
    }
}
```

```

        theirest *= cnt;
        value += theirest;
    }

    if (num == 0) return null; // Никто не голосовал за эту квартиру

    return value / num; // Средневзвешанная оценка квартиры.
}

```

2.6.3 Подсистема кластерного анализа

Цель кластерного анализа – это структуризация множества объектов по схожести. Как правило, такая структуризация преследует целью образование групп схожих между собой объектов, которые называются кластерами. Не зная интересов нового пользователя, либо не имея обширной базы данных о предпочтениях других пользователей, иногда невозможно применение методов коллаборативной фильтрации для выработки рекомендации. Поэтому необходимо использовать какой-либо другой способ сравнить объекты недвижимости в контексте интересов текущего покупателя. Для этого используем кластерный анализ, предлагая пользователю разные варианты недвижимости из разных классов. Далее более подробно рассмотрим принцип агломеративного метода кластерного анализа.

Агломеративный метод кластерного анализа. Агломеративный метод кластерного анализа предполагает, что каждый объект в начале исследования является отдельным кластером и производится группировка схожих объектов на основании матрицы мер сходства по кластерам. На заключительном этапе выполняется сохранение данных о кластеризации объектов. Вычисляются центры кластеров, каждому объекту присваивается свой номер кластера для дальнейшего использования кластеров при автоматизированных расчетах без повторной кластеризации данных.

Агломеративный метод кластерного анализа характеризуется последовательным объединением исходных элементов и соответствующим

щим уменьшением числа кластеров. В начале работы алгоритма все объекты являются отдельными кластерами. На первом шаге наиболее похожие объекты объединяются в кластер. На последующих шагах объединение продолжается до тех пор, пока все объекты не будут составлять один кластер, как это представлено на рисунке 2.6.

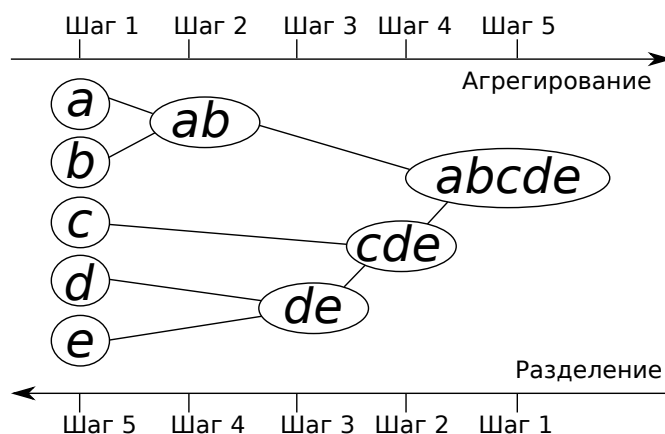


Рисунок 2.6 — Иерархический кластерный анализ

Процесс поиска представляет собой процедуру определения «ближайшего» (наилучшего) кластера, изначально пользователю предлагаются разные варианты недвижимости из разных кластеров, например, квартира, дом или участок. В свою очередь у **этих кластеров имеются свои параметры**, такие как «тип строения», кирпичное, деревянное, панельное и другие здания. При выборе такого или иного варианта недвижимости и типа строения, пользователю выводятся предложения о недвижимости с наиболее «близкими» исходными данными основанные на кластеризации.

Рассмотрим элементы исходного множества $I = (I_1, I_2, \dots, I_n)$ как множество одноэлементных кластеров $\{I_1\}, \{I_2\}, \dots, \{I_n\}$. Выберем два из них, например, I_i и I_j , которые в некотором смысле более близки друг к другу и объединим их в один кластер. Новое множество кластеров, состоящее уже из $n - 1$ кластеров, будет: $\{I_1\}, \{I_2\}, \dots, \{I_i, I_j\}, \dots, \{I_n\}$. Повторяя процесс, получим последовательные множества кластеров, состоящие из $(n - 2)$, $(n - 3)$, $(n - 4)$ и т.д. кластеров. В конце процедуры можно получить кластер, состоящий из n объектов и совпадающий с первоначальным множеством $I = (I_1, I_2, \dots, I_n)$. В качестве меры рас-

стояния возьмем квадрат евклидовой метрики и вычислим матрицу $D = \{d_{ij}^2\}$, где d_{ij}^2 – квадрат расстояния между I_i и I_j .

| | I_1 | I_2 | I_3 | ... | I_n |
|-------|-------|-------------|-------------|-----|-------------|
| I_1 | 0 | $d_{1,2}^2$ | $d_{1,3}^2$ | ... | $d_{1,n}^2$ |
| I_2 | | 0 | $d_{2,3}^2$ | ... | $d_{2,n}^2$ |
| I_3 | | | 0 | ... | $d_{3,n}^2$ |
| ... | | | | ... | ... |
| I_n | | | | | 0 |

Рисунок 2.7 — Симметричная матрица расстояний

Пусть расстояние между I_i и I_j будет минимальным: $d_{ij}^2 = \min (d_{ij}^2, i \neq j)$. Образует с помощью I_i и I_j новый кластер $\{I_i, I_j\}$.

| | $\{I_i, I_j\}$ | I_1 | I_2 | I_3 | ... | I_n |
|----------------|----------------|--------------|--------------|--------------|-----|--------------|
| $\{I_i, I_j\}$ | 0 | $d_{ij,1}^2$ | $d_{ij,2}^2$ | $d_{ij,3}^2$ | ... | $d_{ij,n}^2$ |
| I_1 | | 0 | $d_{1,2}^2$ | $d_{1,3}^2$ | ... | $d_{1,n}^2$ |
| I_2 | | | 0 | $d_{2,3}^2$ | ... | $d_{2,n}^2$ |
| I_3 | | | | 0 | ... | $d_{3,n}^2$ |
| ... | | | | | ... | ... |
| I_n | | | | | | 0 |

Рисунок 2.8 — Новая матрица расстояний

Для новой матрицы $(n - 2)$ строки взяты из предыдущей, а первая строка вычислена заново. Исходно определено расстояние лишь между одноэлементными кластерами, но надо определять расстояния и между кластерами, содержащими более чем один элемент. Это можно сделать различными способами, в зависимости от выбранного способа мы получаем алгоритмы кластерного анализа с различными свойствами. Можно, например, положить расстояние между кластером $i + j$ и некоторым другим кластером k , равным среднему арифметическому из расстояний между кластерами i и k и кластерами j и k :

$$d_{i+j,k} = (d_{i,k} + d_{j,k})/2.$$

Но можно также определить его как минимальное из этих двух расстояний:

$$d_{i+j,k} = \min \{d_{i,k}, d_{j,k}\}.$$

Подобная операция выполняется на каждом шаге агломеративного иерархического метода кластерного анализа. В результате получается древовидная структура, представленная на рисунке 2.9.

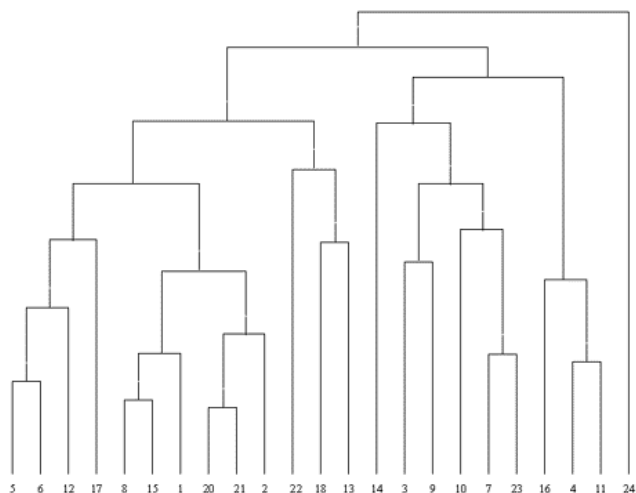


Рисунок 2.9 — Пример дерева кластерного анализа [3]

Существуют методы, которые строят подобные деревья но в обратном направлении, т.е. в отличие от агломеративного метода они разделяют все множество объектов на подмножества. Методы второго класса удобны, если множество объектов большое и не требуется строить дерево до листовых вершин. При беглом поиске библиотеки, реализующей метод второго типа, такую библиотеку обнаружить нам, к сожалению, в среде .NET не удалось, поэтому в данной работе использован *агломеративный* метод из библиотеки ALGLIB [2].

2.6.4 Функция меры схожести для объектов недвижимости

Как было сказано выше применение методов кластерного анализа неразрывно связано с построением матрицы различия между объектами исходного множества. В нашем случае необходимо задать меру схожести между объектами недвижимости. Каждый объект недвижимости задан атрибутами, поэтому необходимо построить функцию сравнения двух объектов как совокупность функции сравнения их атрибутов.

Согласно диаграмме классов, изображенной на рисунке 2.3 (стр. 39), атрибуты разнородны (описываются различными типами данных), поэтому необходимо задать для каждого типа данных, фактически для каждого атрибута, функцию сравнения $d_k(i, j) : A_k \rightarrow [0, 1]$, где k – номер атрибута $k = 1, 2, \dots, m$, m – число атрибутов в описании объекта.

Таблица 2.1 — Перечень функций сравнения атрибутов объекта недвижимости

| Атрибут | Способ вычисления | $v_k, \%$ | Формула |
|---------------------------|---|-----------|---|
| string Name | не используется | | |
| ILocation Location | совпадение значений | 10 | $d_k(i, j) = \begin{cases} 0, & \text{если } a_i = a_j, \\ 1, & \text{если } a_i \neq a_j, \end{cases} \quad (\star)$ |
| string Address | —”— | 10 | |
| float Price | относительная разница величины | 25 | $d_k(i, j) = \frac{ a_i^k - a_j^k }{a_i^k + a_j^k}, a_i^k + a_j^k > 0 \quad (\star\star)$ |
| CurrencyEnum CurrencyType | не используется | | |
| float Area | относительная разница величины | 35 | $(\star\star)$ |
| AreaUnits AreaUnit | не используется | | |
| string ImageURL | —”— | | |
| string URL | —”— | | |
| int Rooms | относительная разница величины | 100 | $(\star\star)$ |
| int RoomsOffered | —”— | 100 | |
| int Floor | —”— | 30 | $(\star\star)$ |
| int FloorTotal | —”— | 10 | |
| BuildingEnum BuildingType | совпадение значений | 30 | (\star) |
| IBuild... BuildingSeries | —”— | 30 | |
| PropertyEnum PropertyType | —”— | 30 | (\star) |
| CategoryEnum Category | —”— (вид объекта: квартира, дом и т.д.) | 100 | |
| string Description | не используется | | |
| string GUID | —”— | | |

В таблице v_k – весовой коэффициент k -го атрибута.

Далее значения по всем атрибутам необходимо свести в одно, используем формулу манхеттеновского расстояния:

$$d(i, j) = \frac{\sum_{k=1}^m |v_k \cdot d_k(i, j)|}{\sum_{k=1}^m v_k}, \quad 0 \leq d_{i,j} \leq 1.$$

где v_k – весовой коэффициент, который задает степень «важности» данного атрибута. В нашем случае использован ряд значений от 0 до

100, где числа до 15 обозначают неважный атрибут, 15-50 – атрибут «средней» важности, и число, большее 50 обозначает важный атрибут. Принятая нами система, таким образом, задает атрибуты типа жилья и «количество комнат» самыми важными в сравнении. Меняя соотношение весовых коэффициентов можно строить новые варианты кластеров.

2.7 Тестирование системы

Тестирование программной системы проводилось при помощи двух подходов:

- ❑ разработка функциональных тестов и тестов корректности;
- ❑ тестирование вручную несколькими «виртуальными» пользователями;

2.7.1 Автоматическое функциональное тестирование

xUnit - это собирательное название семейства фреймворков для модульного тестирования, структура и функциональность которых основана на SUnit, предназначавшегося для языка программирования Smalltalk.

В данном проекте автоматическое тестирование использовалось для а) тестирования функционирования внешних библиотек (работают ли они так, как заявлено) и б) для тестирования некоторых методов без запуска сервера. Приведем три примера.

Первый пример – это проверка алгоритма импорта данных. Импорт успешен, если не возникло исключительных ситуаций.

```
[Theory]
[InlineData("all.xml")]
// Загрузится ли файл с данными квартир?
public void LongImportTest(string importName)
{
    string fileName = CombineWithDataPath(importName);
    Console.WriteLine("Importing : "+fileName);
    MyEntityContext ctx = Application.Context;
```

```

ImportFromAtlcomru import = new ImportFromAtlcomru() // Сделать импорт данных
{
    FileName = fileName
};
if (DO_REAL_IMPORT)
{
    import.Import();
}
else
{
    Console.WriteLine("Doing FAKE import");
}

Assert.True(true);
}

```

Проведение кластерного анализа данных БД представлено в этом примере. Тест удачен, если не произошло исключительных ситуаций.

```

[Fact]
public void Hierarchical_Clustering()
{
    // Квартирный кластерный анализатор
    var a = FlatClusterAnalyzer.AnalyzeFlatWithCluster(50);
    // bool res = a.Process();
    a.PrepareClusters(5);
    Console.Write("-Cluster-> ");
    Console.WriteLine(a.Cidx);
    Assert.True(true);
}

```

Последний пример – тестирование библиотеки на примере из документации. Проверка результата тестирования аналогична предыдущим примерам.

```

[Fact] // Потестим, работает ли библиотека.
public void alglib_hclust_works()
{
    //
    // The very simple clusterization example
    //
    // We have a set of points in 2D space:

```



```

//      (P0,P1,P2,P3,P4) = ((1,1),(1,2),(4,1),(2,3),(4,1.5))
. . . . .
//
Console.WriteLine("— Testing cluster analyzer — ");
alglib.clusterizerstate s;
alglib.ahcreport rep;
double[,] xy = new double[,] { { 1, 1 }, { 1, 2 }, { 4, 1 },
                                { 2, 3 }, { 4, 1.5 } };

alglib.clusterizercreate(out s); // Создать кластерайзер ж-)
alglib.clusterizersetpoints(s, xy, 2); // Ввод точек
alglib.clusterizerrunahc(s, out rep); // Запуск кластеризации

//
// Now we've built our clusterization tree. Rep.z contains information which ...
//
. . . . .
// Thus, we have following dendrogram:
//
//      —8—   cluster = 2 значит... туплю... завтра продолжим.
//      |       |
//      |       --7--
//      |       |       |
//      -5-   |   -6-
//      |     |   |   |   |
//      P2    P4  P3  P0  P1
//      5     5   7   0   0
//
System.Console.WriteLine("{0}", alglib.ap.format(rep.z));
// EXPECTED: [[2,4],[0,1],[3,6],[5,7]]

//
// We've built dendrogram above by reordering our dataset.
//
. . . . .
//
System.Console.WriteLine("{0}", alglib.ap.format(rep.p));
// EXPECTED: [3,4,0,2,1]
System.Console.WriteLine("{0}", alglib.ap.format(rep.pm));
// EXPECTED: [[0,0,1,1,0,0],[3,3,4,4,0,0],[2,2,3,4,0,1],[0,1,2,4,1,2]]
// System.Console.ReadLine(); // Из оригинального примера.

```

```
Assert.True(true);  
}
```

Использование автоматического тестирования позволяет существенно сократить сроки реализации конкретных подсистем, т.к. а) разрабатываемый метод или процедура тестируется отдельно от остальных подсистем: не надо загружать лишние сервисы; б) нет необходимости делать какие-либо действия с интерфейсом пользователя, достаточно ассоциировать запуск тестирования с клавишей редактора. Но, конечно, приходится писать много дополнительного кода, т.е. программного кода теста.

2.7.2 Структуризация множества объектов недвижимости

Перед тем, как начать тестирование функций рекомендательной системы можно выполнить процесс структуризации объектов недвижимости, т.е. разбить все множество объектов на подмножества при помощи иерархического кластерного анализа. Далее экспериментальным путем подбирается количество кластеров для представления структуры рынка недвижимости региона.

Процедура выполняется экспертом при помощи интерфейса пользователя по адресу <http://127.0.0.1:8080/analysis>. Пример использования данного режима информационной системы приведен на рисунке 2.10.

Экран представляет собой перечень пронумерованных кластеров (подмножеств), начиная с 0. Страница позволяет выполнить следующие операции:

- ❑ задать название подмножеству: в поле соответствующего кластера вписывается строка символов (название кластера), например «Дома», нажимается верхняя кнопка «Обновить», при этом все кластеры с одинаковым названиями объединяются в один общий кластер, и процедуры оценивания будут выполняться только с объектами данного объединения;
- ❑ перестроить построенное дерево анализа, при этом выделив из него заданное количество кластеров, для этого надо в поле «Кол-

| Номер | Название | Кол-во объектов |
|-------|----------|-----------------|
| 3 | 3 | 10 |
| 2 | 2 | 2 |
| 5 | 5 | 324 |
| 0 | 0 | 1 |
| 6 | 6 | 140 |
| 1 | 1 | 1 |
| 4 | 4 | 22 |

Обновить

Кол-во кластеров

5

Перестроить кластер

Максимальное количество квартир в выборке

100

Обновить

Рисунок 2.10 — Проведение структуризации данных рынка недвижимости (начальный этап)

во кластеров» ввести число требуемых кластеров и нажать кнопку «Перестроить кластер»;

- перезапустить процедуру построения дерева кластерного анализа заново, нажав нижнюю кнопку «Обновить», при этом предварительно можно указать на каком количестве случайно выбранных объектов недвижимости требуется выполнять данную операцию.

На рисунке 2.12 Изображена часть множества объектов, обозначенных как «Дом». Переход на эту страницу осуществляется при помощи нажатия кнопки с числом (количеством объектов кластера) на экране стр. 2.10.

Просмотрев содержимое кластера можно задать ему название. Несколько кластеров могут называться одинаково, при этом в системе они обрабатываются как один общий кластер (см. рисунок ??).

Если не удастся по каким либо причинам означить кластер, построенный при заданных по умолчанию параметрах, то можно попробовать разбить множество объектов недвижимости на большее количество кластеров. Для этого в форму на рисунке 2.10 в поле «кол-во кластеров» вводится требуемое количество и нажимается кнопка «Перестроить кластер». Для 7 новых кластеров в нашем примере получен результат, изображенный на рисунке 2.13.

| Объект | Комн | Площ | Регион | Адрес | Мат | Этаж | Цена |
|--------|------|------|--------|------------------------------------|------|------|---------------|
| Дом | 2к | 0 | ПРИ | ий СНТ Ангарские Хутора улица, €€€ | шлак | 2 | 950 000,00 |
| Дом | 2к | 0 | ПРИ | Большая речка пос, 1 | кирп | 2 | 20 000 000,00 |
| Дом | 5к | 0 | ПРИ | Мира улица, €€€ | кирп | 2 | 4 300 000,00 |
| Дом | 4к | 0 | ПРИ | €€€ | кирп | 1/2 | 850 000,00 |
| Дом | 1к | 0 | ЛЕН | СНТ 6-я Пятилетка улица, €€€ | шлак | 2 | 3 500 000,00 |
| Дом | 1к | 0 | ПРИ | Пивовариха с | дере | 2 | 2 100 000,00 |
| Дом | 3к | 0 | ПРИ | €€€ | кирп | 2 | 1 700 000,00 |
| Дом | 2к | 0 | ОКТ | Лебедева-Кумача улица, 21 | дере | 2/2 | 12 000 000,00 |
| Дом | 2к | 0 | ПРИ | €€€, 25а | пане | 2 | 4 100 000,00 |
| Дом | 2к | 0 | ПРИ | | дере | 2 | 800 000,00 |

Рисунок 2.11 — Представление содержимого выбранного кластера

| Номер | Название | Кол-во объектов |
|-------|---|-----------------|
| 19 | <input type="text" value="Двухкомнатные"/> | 1049 |
| 16 | <input type="text" value="Однокомнатные"/> | 394 |
| 15 | <input type="text" value="Двухкомнатные"/> | 5 |
| 7 | <input type="text" value="Одно-двухкомнатные"/> | 25 |
| 17 | <input type="text" value="Двухкомнатные"/> | 62 |
| 4 | <input type="text" value="Участки"/> | 17 |
| 10 | <input type="text" value="Дома"/> | 680 |
| 8 | <input type="text" value="Комнаты"/> | 23 |
| 13 | <input type="text" value="Торговое"/> | 13 |
| 2 | <input type="text" value="Гараж"/> | 1 |
| 6 | <input type="text" value="6"/> | 24 |

Кол-во кластеров

Максимальное количество квартир в выборке

Рисунок 2.12 — Пример задания названия кластерам

Удачное завершение операции! Произведена перестройка кластера

| Номер | Название | Кол-во объектов |
|-------|----------|-----------------|
| 3 | 3 | 10 |
| 2 | 2 | 2 |
| 4 | 4 | 22 |
| 5 | 5 | 324 |
| 0 | 0 | 1 |
| 6 | 6 | 140 |
| 1 | 1 | 1 |

Обновить

Кол-во кластеров

7

Перестроить кластер


Рисунок 2.13 — Перестроенный кластер для 7 подмножеств объектов

Таким образом можно добиться необходимого разбиения исходного множества объектов на несколько подмножеств.

2.7.3 Тестирование системы пользователями

Для тестирования РС было «зарегистрировано» два пользователя. Один пользователь случайно выбирал объекты одного кластера, другой обновлял страницу с одним и тем же объектом недвижимости и наблюдал, как меняется список рекомендаций (рис. 2.14). Параллельно проводилось наблюдение за выдачей отладочной информации в консоль сервера (листинг 2.4).

Дом, Большая речка пос, 1

| | | |
|----------|----------------------|--|
| Объект | Дом |  |
| Комнат | 2к | |
| Площадь | 0 | |
| Район | Пригород | |
| Адрес | Большая речка пос, 1 | |
| Материал | кирп | |
| Этаж | 2 | |
| Цена | 20 000 000,00 | |

Рекомендуем посмотреть следующие объекты:

| Объект | Комн | Площ | Регион | Адрес | Мат | Этаж | Цена |
|--------|------|------|--------|------------------------------------|------|------|---------------|
| Дом | 2к | 0 | ПРИ | Большая речка пос, 1 | кирп | 2 | 20 000 000,00 |
| Дом | 2к | 0 | ПРИ | ий СНТ Ангарские Хутора улица, €€€ | шлак | 2 | 950 000,00 |
| Дом | 5к | 0 | ПРИ | Мира улица, €€€ | кирп | 2 | 4 300 000,00 |
| Дом | 4к | 0 | ПРИ | €€€ | кирп | 1/2 | 850 000,00 |
| Дом | 1к | 0 | ЛЕН | СНТ 6-я Пятилетка улица, €€€ | шлак | 2 | 3 500 000,00 |

Рисунок 2.14 — Пример экрана пользователя

Листинг 2.4 – Журнал отладки и тестирования рекомендательной системы в режиме выработки рекомендаций

```

Template Path:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/build/sneak.pt
RSP0.Offer
Template Path:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/build/offer.pt
Spying: Added like of agent c0f04518-3a0e-4ecd-9100-0e883dd6300d to object
f1b865fd-fa17-42b0-96f1-39a8bfe7b7ef and now it is 5
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/css/screen.min.css
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/css/custom.min.css
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/js/body.min.js
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/images/logo.jpg
RS: importing data
RS: importing data
Template Path:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/build/sneak.pt
RSP0.Offer
Template Path:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/build/offer.pt
Spying: Added like of agent c0f04518-3a0e-4ecd-9100-0e883dd6300d to object
acc19d1c-c77e-40c5-879a-ff5e449fb548 and now it is 5
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/css/screen.min.css
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/css/custom.min.css
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/js/body.min.js
READ:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/images/logo.jpg
RS: importing data
RS: importing data
Template Path:/home/eugeneai/projects/code/RSP0/design-studio_one-page-template/build/sneak.pt

```

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе магистранта предложен вариант информационной рекомендательной системы для рынка недвижимости Иркутской области. Решены следующие задачи:

1. Проанализирована предметная область рекомендательных систем, опыт их использования в различных отраслях, в том числе на рынке недвижимости;
2. Предложена методика вычисления рекомендаций для объектов рынка недвижимости, основанная на комбинации метода коллаборативной фильтрации Slope One и кластерного анализа;
3. Спроектирована объектно-ориентированная структура базы данных, предложена клиент-серверная архитектуры системы;
4. Реализованы подсистемы РС в среде программирования C# на основе предложенных методик и структур данных;
5. Проведено тестирование разработанных модулей рекомендательной системы.

В результате проектирования создан интернет-приложение, выполняющее функции рекомендательной системы, которые можно использовать в качестве подсистем существующих сайтов продажи недвижимости, таким образом, совершенствуя их.

В рамках проекта не удалось разработать вычислительно-эффективную систему: некоторые алгоритмы могут быть адаптированы к процессу изменения данных, что значительно повысит их эффективность. Вторым направлением совершенствования системы является интеграция с существующими сервисами: автоматическая загрузка данных с сайтов непосредственно в РС, сбор информации о предпочтениях с других сайтов.

Исходный код рекомендательной системы размещен на сайте по адресу <https://github.com/VapeNationGitHub/RSP0>. В приложении приводится код основных вычислительных модулей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Daniel Lemire, Anna Maclachlan. Slope One Predictors for Online Rating-Based Collaborative. [Электронный ресурс] URL: http://cogprints.org/4031/1/lemiremaclachlan_sdm05.pdf (дата доступа: 10.05.2018)
2. ALGLIB – C++/C# numerical analysis library. [Электронный ресурс] URL: <http://www.alglib.net/> (дата доступа: 10.05.2018)
3. Иерархическое группирование // Курс лекций по предмету ”Основы проектирования систем с искусственным интеллектом”. Составитель – С. Л. Сотник. [Электронный ресурс] URL: http://www.codenet.ru/progr/alg/ai/htm/gl3_11.php (дата доступа: 10.05.2018)
4. Руководство по Entity Framework [Электронный ресурс] URL: <https://metanit.com/sharp/entityframework/> (дата доступа: 10.05.2018)
5. Петер Пин-Шен Чен. Модель «сущность-связь» – шаг к единому представлению о данных // П. П.-Ш. Чен Перевод: М. Р. Кога-ловский. [Электронный ресурс] URL: <http://citforum.ru/database/classics/chen/> (дата доступа: 10.05.2018)
6. D. Jannach, M. Zanker, A. Felfernig, G. Friedrich. Recommender Systems: An Introduction. Cambridge University Press (2010).
7. Как работают рекомендательные системы. Лекция в Яндексе / Блог компании Яндекс / Хабрахабр. [Электронный ресурс] URL: <https://habrahabr.ru/company/yandex/blog/241455/> (дата обращения: 12.12.2016).
8. Е.Е. Пятикоп. Исследование метода коллаборативной фильтрации на основе сходства элементов // Наукові праці ДонНТУ Серія “Інформатика, кібернетика та обчислювальна техніка”, вып.2(18), 2013. с.109-114.

9. Е. В. Бритвина. Сегментирование рекомендательной системы с использованием метода организации соединения «клиент - сервер», основанного на программно-конфигурируемых сетях и применении протокола с быстрым перескоком IP-адреса. // Современные проблемы науки и образования. Электронный научный журнал. URL: <https://www.science-education.ru/ru/article/view?id=16875> № 6. 2015. (дата обращения: 12.12.2016)
10. Б. Р. Авхадеев, Л. И. Воронова, Е. П. Охупкина. Разработка рекомендательной системы на основе данных из профиля социальной сети «ВКонтакте» // Вестник Нижневартовского государственного университета. Выпуск № 3. 2014.
11. N. Hossain. Why the Interest Graph Is a Marketer's Best Friend. URL: http://mashable.com/2012/06/19/interest-graph-marketer/#Hr95qUR_7Eqa (дата обращения: 12.12.2016)
12. О. Жернакова. Системы рекомендаций и поиска видеоконтента // Телемультимедиа. 2012. URL: <http://www.telemultimedia.ru/art.php?id=464> (дата обращения: 12.12.2016)
13. J. Beel, B. Gripp, S. Langer, C. Breiting. Research-paper recommender systems: a literature survey // International Journal on Digital Libraries (2016) 17: 305. doi:10.1007/s00799-015-0156-0. (дата обращения: 12.12.2016)
14. С. А. Амелькин, Д. М. Понизовкин. Математическая модель задачи top-N для контентных рекомендательных систем. // Известия МГТУ «МАМИ» No 3(17), 2013, т.2. с. 26-31.
15. Ю. С. Нефедова. Архитектура гибридной рекомендательной системы GEFEST (Generation–Expansion–Filtering–Sorting–Truncation) // Системы и средства информатики. 2012, Т.22, вып.2, с.176–196.
16. А.Г. Дьяконов. Алгоритмы для рекомендательной системы: технология Lenkor // Бизнес-информатика No1(19) – 2012 г. с. 32-39.

17. П. А. Клеменков. Построение новостного рекомендательного сервиса реального времени с использованием NoSQL СУБД // Информатика и её применения. 2013, Т.7, вып.3, с.14–21.
18. С. А. Амелькин. Оценка эффективности рекомендательных систем. // Труды 14-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2012, Переславль-Залесский, Россия, 15-18 октября 2012 г.
19. А. А. Правиков, В. А. Фомичев Разработка рекомендательной системы с естественно-языковым интерфейсом на основе математических моделей семантических объектов. // Бизнес-информатика № 4(14), 2010. с.3-11.
20. Е. В. Бритвина. Сегментирование рекомендательной системы с использованием метода организации соединения «клиент - сервер», основанного на программно-конфигурируемых сетях и применении протокола с быстрым перескоком IP-адреса. // Современные проблемы науки и образования. Электронный научный журнал. URL:<https://www.science-education.ru/ru/article/view?id=16875> № 6. 2015.
21. T. Ginevičius, A. Kaklauskas, P. Kazokaitis, J. Alchimovienė. Recommender system for real estate management // Verslas: Teorija ir praktika (Business: Theory and Practice). 2011 12(3). p. 258–267 doi: 10.3846/btp.2011.26
22. E. M. Alrawhani, H. Basirona, Z. Sa'ayaa. Real estate recommender system using case-based reasoning approach. // Journal of Telecommunication, Electronic and Computer Engineering (JTEC). Vol. 8 No. 2. p. 177-182.
23. X. Yuan, J.-H. Lee, S.-J. Kim, Y.-H. Kim. Toward a user-oriented recommendation system for real estate websites. // Information Systems 38 (2013). p. 231–243.

24. Доска объявлений от частных лиц и компаний на Avito. [Электронный ресурс] URL:<https://www.avito.ru/> (дата обращения:12.12.2016).
25. Model-View-Controller – Wikipedia [Электронный ресурс] URL:<https://ru.wikipedia.org/wiki/Model-View-Controller> (дата обращения:12.12.2016).

Приложение 1 Листинг ClusterAnalysis.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RSP0
{
    public class DataSet<T> // Класс абстрактных анализаторов
    {
        // protected

        public DataSet()
        {
            input = new List<T>();
        }

        public void Add(T row)
        // Вообще могут быть несколько таблиц данных
        // Т.е. матрица квартир.
        // Матрица пользовательского выбора.
        {
            input.Add(row);
        }

        public void Clear()
        {
            input.Clear();
        }

        public int Count
        {
            get
            {
                return input.Count;
            }
        }

        // Входные данные, являющиеся результатом выборки из БД
        // Их еще предстоит преобразовать в матрицы, вектора и т.п.
        protected List<T> input = null;
    }
}
```

```

    }

    public class Analyzer<T> : DataSet<T>
    {
        public Analyzer() : base() {}
        public virtual bool Process() { return false; }
        public bool DEBUG = false;
    }

    public class ClusterAnalyzer<T> : Analyzer<T> // Класс кластерных
        ↪ анализаторов
    {
        public ClusterAnalyzer() : base()
        {
            // Инициализация кластеризатора.....
            alglb.clusterizercreate(out state);
        }

        public override bool Process()
        {
            double[,] distances = new double[Count, Count]; // It's matrix!
            //
            // пример.
            // = new double[,]{ {0,3,1},{3,0,3},{1,3,0}};

            prepareDistMatrix(distances);

            // Капец. Привык, что все динамично.

            Console.WriteLine("->Doing clustering.");
            alglb.clusterizersetdistances(state, distances, true);
            Console.WriteLine("->Retrieveing result.");
            alglb.clusterizerrunahc(state, out report);
            Console.WriteLine("Done clustering.");
            return true;
        }

        protected int[] cidx;
        protected int[] cz;

        public int[] Cidx

```

```

{
    get
    {
        return cidx;
    }
}

public int[] Cz
{
    get
    {
        return cz;
    }
}

public void PrepareClusters(int k)
{
    alglib.clusterizergetkclusters(report, k, out cidx, out cz);
}

protected alglib.clusterizerstate state; // Объект, выполняющий
    ↪ кластерный анализ
protected alglib.ahcreport report;      // Результат кластеризации.

public int[,] Z
{
    get
    {
        return report.z;
    }
}

protected void prepareDistMatrix(double[,] m)
{
    Console.WriteLine("->Constructing dissimilarity matrix.");
    foreach (var o1 in input.Select((v, i) => new {v,i})) // Верхний
        ↪ объект
    {
        Console.WriteLine(" - Row: "+o1.i);
        int j = o1.i + 1; // Заполняем столбцы от диагонали, направо до
            ↪ конца
    }
}

```

```

// и вниз до конца.
while (j<Count)
{
    T o2 = input[j];
    double v = compare(o1.v,o2);
    m[o1.i, j] = v;
    m[j, o1.i] = v; // Теперь симметрично.
    j++; // Пока кластер с 1000 квартирами поюзаем. Ночью
        ↪ посчитаем все.
}
}
Console.WriteLine("Done.");
/* // Печать матрицы различий
for (var row=0; row<Count; row++)
{
    for (var col=0; col<Count; col++)
    {
        Console.Write(""+m[row,col]+" ");
    }
    Console.WriteLine();
}
*/
}

protected virtual double compare(T v1, T v2)
{
    throw new NotImplementedException(); // Сейчас я не знаю, что есть
        ↪ тип T.
}
}

// TODO: Реализовать анализатор рекомендательной системы
// Weighted Slope One:
    ↪ http://www.cnblogs.com/kuber/articles/SlopeOne\_CSharp.html
// Я пишу как художник.

public class WeightedSlopeOne<T1, T2> : Analyzer<T1>
{
    public WeightedSlopeOne() : base() {}

    public override bool Process()

```

```

    {
        // TODO: Тело Метода.
        return true;
    }

    public DataSet<T2> Choices = new DataSet<T2>();
}

public class ClusterList : EntityList<IClassName> { }

public class ClusterListView : EntityListView<ClusterList>
{
    public ClusterListView(ClusterList context): base (context) {}

    public int CountObjects(IClassName o)
    {
        MyEntityContext ctx = Application.Context;
        Console.Write("Query for count:" + o.Cluster + "(" + o.Name + ") objs:");
        int num = ctx.ObjectClassss.Where(x => x.Cluster == o.Cluster).Count();
        Console.WriteLine(num);
        return num;
    }
}
}

```


Приложение 2 Листинг FlatClusterAnalysis.cs

```
using System;
using System.Linq;

namespace RSP0
{
    public class FlatClusterAnalyzer : ClusterAnalyzer<IObject>
    {
        public FlatClusterAnalyzer() : base() { }

        protected double diff(double a, double b)
        {
            if (Math.Abs(a + b) < 0.01) return 0.0;
            double res = Math.Abs(a - b) / (a + b);
            return res;
        }

        protected override double compare(IObject o1, IObject o2)
        {
            if (o1 == o2) return 0.0; // Работа протестирована
            // Надо сравнить два оъекта по полям.
            double s = 0.0; // Чем больше s, тем сильнее различаются
            ↪ объекты.
            double q = 0.0; // Сумма максимумов важностей
            double v = 0;

            // Используем манхэттеновскую меру. |x1-x2| + |y1-y2| +
            ↪ ...

            // + нам надо что-то вроде важности поля.
            // (район города) - это очень важно.
            v = 10;
            if (o1.Location != o2.Location)
            {
                s += v;
            }
            q += v; // s суммируется не всегда.

            // Address compare
        }
    }
}
```

```

v = 25;
s += diff(o1.Price, o2.Price) * v;
q += v;

v = 35;
s += diff(o1.Area, o2.Area) * v;
q += v;

v = 100;
bool r1 = o1.Rooms > 0, r2 = o2.Rooms > 0;
bool ro1 = o1.RoomsOffered > 0, ro2 = o2.RoomsOffered >
↪ 0;
if (r1 != r2)
{
    s += 1 * v; // Разные объекты
}
else // Объекты похожи в смысле измерения комнат
{
    if (!r1) // Оба объекта без комнат.
    {
        // Ничего не добавляем к s
    }
    else // Объекты с комнатами
    {
        double v1 = 100;
        if (ro1 == ro2)
        {
            if (ro1) // Сдают комнаты.
            {
                s +=
                ↪ diff(o1.RoomsOffered,
                ↪ o2.RoomsOffered) *
                ↪ v1;
            }
            // Иначе они одинаковы
            ↪ относительно доли комнат.
        }
        else
        {

```

```

        s += v1; // Один сдает комнаты,
        ↪ другой все сразу.
    }

    q += v1;

    s += diff(o1.Rooms, o2.Rooms) * v1;
}
}
q += v; // Суммируем всегда

v = 10; // Размер здания не сильно важен.
bool f1 = o1.FloorTotal > 0, f2 = o2.FloorTotal > 0;
if (f1 == f2)
{
    if (f1) // Здания с этажами
    {
        s += diff(o1.FloorTotal, o2.FloorTotal) *
        ↪ v;
    }
    // else Здания без этажей
}
else
{
    s += v; // Один с этажами, др. нет.
}
q += v;

v = 30; // Этажи одни и те же.
f1 = o1.Floor > 0;
f2 = o2.Floor > 0;
if (f1 == f2)
{
    if (f1)
    {
        s += diff(o1.Floor, o2.Floor) * v;
    }
}
else
{
    s += v;
}

```

```

    }
    q += v;

    v = 30;
    if (o1.BuildingType != o2.BuildingType) s += v;
    q += v;

    v = 30;
    if (o1.BuildingSeries != o2.BuildingSeries) s += v;
    q += v;

    v = 30;
    if (o1.PropertyType != o2.PropertyType) s += v;
    q += v;

    v = 100;
    if (o1.Category != o2.Category) s += v;
    q += v;

    // v=20; // Один и тот же агент продает
    // if (o1.Agents == o2.Agents) s+=v;
    // q+=v;

    double rate = s / q;

    if (rate < 0.0 || rate > 1.0)
        throw new Exception(string.Format("wrong rate
        ↪ {0}", rate));

    // Console.WriteLine("R->" + rate + " " + s + "/" + q);
    return rate;
}

public static FlatClusterAnalyzer AnalyzeFlatWithCluster(int?
    ↪ cmax = 10)
{
    FlatClusterAnalyzer fca = new FlatClusterAnalyzer();
    MyEntityContext ctx = Application.Context;

    Console.WriteLine("-> Adding data from database");

```

```

        int counter = 0;
        var allSet = ctx.Objects;
        IQueryable<RSP0.IObject> qSet;

        if (cmax != null)
        {
            qSet = allSet.Take((int)cmax);
        }
        else
        {
            qSet = allSet;
        }

        foreach (IObject o in qSet)
        {
            if (counter % 100 == 0)
            {
                Console.WriteLine("" + counter + "    \r");
            }
            fca.Add(o);
            counter++;
        }

        Console.WriteLine("\nStarting cluster analysis....");
        if (!fca.Process())
            throw new ProcessingException("cannot process
            ↪ data");
        Console.WriteLine("Finished....");

        return fca;
    }

    // Сохранить результат в базе данных

    public bool Store(int k)
    {
        MyEntityContext ctx = Application.Context;

        PrepareClusters(k);
    }

```

```

ctx.ObjectClassss.ToList().ForEach(ctx.DeleteObject);
ctx.SaveChanges();

foreach (var cl in Cidx.Select((c, i) => new {c,i}))
{
    IObjectClass c = ctx.ObjectClassss.Create();
    c.Cluster = cl.c;
    IObject ob = input[cl.i];
    c.Object = ob;
    Console.WriteLine("ob:"+ob.GUID+"->" +cl.c+"["+cl.i+"]");
}

ctx.SaveChanges();

ctx.ClassNames.ToList().ForEach(ctx.DeleteObject);

for(int i=0; i<k; i++) // Заготовки имен кластеров.
{
    IClassName cn = ctx.ClassNames.Create();
    cn.Name=i.ToString();
    cn.Cluster=i;
}

ctx.SaveChanges();

        return true;
    }

    // В винде сейчас регенерирую БД. // грузится ...
}

public class ProcessingException : Exception
{
    public ProcessingException(string msg) : base(msg) { }
}
}

```

Приложение 3 Листинг SlopeOne.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RSP0
{
    public class SlopeOne : Analyzer<ILikes>
    {
        // Класс рассчитывает по квартирам в кластере (это фича будет)
        // вес квартиры, которую пользователь еще не смотрел.
        {
            public SlopeOne() : base() { }

            public List<int> Clusters; // Список кластеров, которые нас
            ↪ интересует

            public List<ILikes> Likes = null;

            public override bool Process()
            {
                // Итак, у нас список объектов ILikes, хранящихся в
                ↪ input;
                // Надо рассчитать матрицу весов: сколько раз посещали
                ↪ квартиру.
                // Для этого надо матрицу создать n x n, где n - это
                ↪ количество
                // квартир в выборке элементов всех кластеров.
                //
                // Не, правильно я сделал. Иначе два раза запросы к БД
                ↪ делать.
                PrepareInput(likes: Likes); // Загрузка данных лайков.

                calcMatrix();

                return true;
            }

            private double[,] m = null; // Матрица оценок.
            private double[,] d = null; // Differences
        }
    }
}
```

```

private int[] v = null; // Sum of votes per object

protected void calcMatrix()
{
    m = new double[agents.Count, objects.Count]; // Bce
    ↪ обнулено.

    foreach (var i in input.Select((v, i) => new { v, i }))
    {
        m[agents[i.v.Agent], objects[i.v.Object]] +=
        ↪ i.v.Value;
        //
        ↪ https://ru.wikipedia.org/wiki/Slope\_One#Коллаборат
    }

    d = new double[objects.Count, objects.Count];
    v = new int[objects.Count];

    for (int row = 0; row < objects.Count; row++)
    {
        int col = row + 1;
        while (col < objects.Count)
        {
            double s = 0;
            int num = 0;
            for (int ag = 0; ag < agents.Count; ag++)
            {
                double a = m[ag, row], b = m[ag,
                ↪ col];
                if (a > 0 && b > 0)
                {
                    s += a - b;
                    num++;
                }
            }
            s /= num;
            d[row, col] = s;
            d[col, row] = -s;
            col++;
        }
    }
}

```



```

    int agnum = 0;
    for (int ag = 0; ag < agents.Count; ag++)
    {
        if (m[ag, row] > 0) agnum++;
    }
    v[row] = agnum;
}
if (DEBUG)
{
    Console.WriteLine("--Indexes--");
    foreach (KeyValuePair<IAgent, int> kvp in agents)
    {
        Console.WriteLine(string.Format("Key =
            ↳ {0}, Value = {1}", kvp.Key.Name,
            ↳ kvp.Value));
    }

    foreach (KeyValuePair<IObject, int> kvp in
        ↳ objects)
    {
        Console.WriteLine(string.Format("Key =
            ↳ {0}, Value = {1}", kvp.Key.Name,
            ↳ kvp.Value));
    } // A bug found.

    Console.WriteLine("-- Source matix --");
    int rowLength = m.GetLength(0);
    int collength = m.GetLength(1);

    for (int i = 0; i < rowLength; i++)
    {
        for (int j = 0; j < collength; j++)
        {
            Console.Write(string.Format("{0}
                ↳ ", m[i, j]));
        }
        Console.WriteLine();
    }

    Console.WriteLine("-- distance matix --");
    rowLength = d.GetLength(0);

```

```

        collength = d.GetLength(1);

        for (int i = 0; i < rowLength; i++)
        {
            for (int j = 0; j < collength; j++)
            {
                Console.Write(string.Format("{0:f}
                ↪ ", d[i, j]));
            }
            Console.WriteLine();
        }

        Console.WriteLine("— Sum values vector —");
        for (int i = 0; i < v.GetLength(0); i++)
        {
            Console.Write(string.Format("{0} ",
            ↪ v[i]));
        }
        Console.WriteLine();
    }
}

public List<ILikes> Estimate(IAgent agent, int cluster)
// Оценивает все квартиры в клатере cluster
// для агента agent
{
    List<ILikes> res = new List<ILikes>();
    MyEntityContext ctx = Application.Context;
    foreach (IObjClass oc in ctx.ObjectClassss.Where(x =>
    ↪ x.Cluster == cluster).ToList())
    {
        ILikes like = ctx.Likess.Create();
        double? eval = Estimate(agent, oc.Object); //
        ↪ Оценить данный объект
        if (eval != null)
        {
            like.Agent = agent;
            like.Object = oc.Object;
            like.Value = (double)eval;
            like.Quality = OriginatingEnum.Evaluated;
            res.Add(like);
        }
    }
}

```

```

        // Эти лайки мы не записываем в БД.!!
    }
}
// Теперь надо упорядочить по убыванию Value
res.Sort(delegate (ILikes x, ILikes y)
    {
        if (x.Value == y.Value) return
            ↪ 0;
        if (x.Value > y.Value) return 1;
        return -1;
    });
return res;
}

public double? Estimate(IAgent agent, IObject obj)
// Оценка одного объекта.
{
    if (m == null) throw new EstimationException("matrix is
        ↪ not constructed");
    int ia, io;
    if (!agents.TryGetValue(agent, out ia))
    {
        return null;
    }
    if (!objects.TryGetValue(obj, out io))
    {
        return null;
    }

    if (m[ia, io] > 0)
    {
        // Оценка им самим уже есть, вернуть ее
        return m[ia, io];
    }

    double value = 0;
    int num = 0;

    for (int row = 0; row < objects.Count; row++)
    {

```

```

        double a = d[io, row];
        if (!(a > 0)) continue;
        double theirest = m[ia, row]; // Моя оценка
        ↪ квартиры, с кот. сравниваем
        if (!(theirest > 0)) continue;
        theirest += a; // Добавить среднюю оценку FIXME:
        ↪ Проверить знак.
        int cnt = v[row];
        num += cnt;
        theirest *= cnt;
        value += theirest;
    }

    if (num == 0) return null; // Никто не голосовал за эту
    ↪ квартиру вообще

    return value / num;
}

protected void addLike(ILikes like)
{
    agents.SetDefault(like.Agent, agents.Count);
    objects.SetDefault(like.Object, objects.Count);
    Add(like); // in input
}

protected void PrepareInput(int? cluster = null, List<ILikes>
    ↪ likes = null)
{
    MyEntityContext ctx = Application.Context;
    Console.WriteLine(" RS: importing data ");

    agents = new Dictionary<IAgent, int>();
    objects = new Dictionary<IObject, int>();
    if (likes != null)
    {
        foreach (ILikes like in likes)
        {
            addLike(like);
        }
    }
}

```

```

        return; // Этот вариант задуман для тестирования.
    }

    if (cluster != null)
    {
        foreach (IObjectClass oc in
            ↪ ctx.ObjectClasses.Where(x => x.Cluster ==
            ↪ cluster).ToList())
        {
            foreach (ILikes l in ctx.Likess.Where(x
                ↪ => x.Object.GUID ==
                ↪ oc.Object.GUID).ToList())
            {
                addLike(l);
            }
        }
    }
    else
    {
        foreach (int c in Clusters)
        {
            PrepareInput(c); // Да. Сие является
                ↪ рекурсией.
        }
    }
}

Dictionary<IAgent, int> agents; // Отображение агента на его
    ↪ индекс в матрице.
Dictionary<IObject, int> objects; // Аналогично... для квартирке.
}

public class EstimationException : Exception
{
    public EstimationException(string msg) : base(msg) { }
}

}

```