Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, _____
_____
_____(И.О. Ф.)
«____»_____2017г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____/А.А.Замышляева
«____»_____2017 г.

Адаптивная модель распознавания текстовых запросов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–010402.2017.146.ПЗ ВКР

Руководитель работы, к.т.н.
_____/ Б.М. Кувшинов
«____»_____2017 г.
Автор работы
Студент группы ЕТ-222
_____/ А.С. Амбросова
«____»_____2017 г.
Нормоконтролер, к.э.н., доцент кафедры ПМиП
_____/Д.А. Дрозин
«____»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Амбросова А.С. Адаптивная модель распознавания текстовых запросов. – Челябинск: ЮУрГУ, ЕТ-222, 94 с., 9 ил., 1 табл., библиогр. список – 50 наим., 3 прил.

В работе исследованы основные подходы к распознаванию текстовой информации. Разработан комбинированный подход распознавания текстовых запросов на основе поиска по документу-образцу и фасетного поиска. Разработаны алгоритмы по разрешению конфликтных ситуаций, возникающих при распознавании текстовых запросов, с использованием номенклатурного справочника и ассоциативных правил. Реализована самообучающаяся система распознавания текстовых запросов для определения лекарства из каталога, требуемого клиенту.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Уже в течение многих лет неуклонно растет интерес к методам обнаружения знаний в базах данных (knowledge discovery in databases) [37]. В результате возникают задачи, связанные с необходимостью обработки больших массивов данных с целью поиска новых закономерностей, установления и выявления новых знаний.

Задачи прикладного характера, связанные с необходимостью обработки больших массивов данных, возникают на промышленных предприятиях для обнаружения скрытых тенденций и закономерностей развития производственных процессов или прогнозирования качества изделия в зависимости от некоторых параметров технологического процесса, а также в организациях, занимающихся розничной торговлей, финансовым анализом, например, для прогнозирования остатка на счетах клиента, в логистике и коммуникациях.

Для анализа данных в настоящее время широко применяют методы и средства искусственного интеллекта, в частности нейронные сети, нечеткие модели, деревья решений, байесовские сети, методы регрессионного анализа и другие [1, 31]. Однако такие методы, как правило, используются для обработки структурированных данных, представленных в виде массивов, содержащих значения признаков и выходных параметров экземпляров выборки.

В то же время чаще всего данные не структурированы [12, 43], то есть каждая единица хранения не может быть представлена конечным числом признаков (атрибутов). Такие данные могут содержать, например, информацию о товарах, купленных одним покупателем у предприятия розничной торговли; результаты ответов респондента при проведении анкетирования; набор установленных диагнозов и результатов лабораторных исследований у пациентов лечебных учреждений; набор различного рода данных о клиентах предприятий и др.

Размер каждой транзакции (множества событий, произошедших одновременно) не является фиксированным.

В связи с этим возникают задачи:

1. сокращения объемов неструктурированных данных путем удаления избыточных транзакций, исключение которых из дальнейшего рассмотрения не повлияет на качество синтезируемых правил и моделей;
2. выявления интересных правил, позволяющих извлекать новые знания на основе имеющихся неструктурированных данных;
3. построения моделей на основе больших массивов неструктурированных данных для решения практических задач прогнозирования, классификации и кластеризации данных.

Для обработки больших массивов неструктурированных данных и решения указанных задач целесообразно рассмотреть средства для распознавания и анализа текстовых документов, а также использовать методы поиска ассоциативных правил, позволяющие выявить новые закономерности на основе хранящейся информации.

1. МЕТОДЫ РАСПОЗНАВАНИЯ ТЕКСТОВОЙ ИНФОРМАЦИИ

Информационный поиск – самостоятельное направление исследований, изучающее вопросы поиска документов, обработки результатов поиска, а также целый ряд смежных вопросов: моделирования, классификации, кластеризации и фильтрации документов, проектирования архитектур поисковых систем и пользовательских интерфейсов, языки запросов и т.д.

Документ – это содержательно законченная единица информации, представленная на каком-либо естественном языке, которая идентифицируется уникальным образом [50].

Информационно-поисковая система – это комплекс программных средств, обеспечивающих избирательный отбор по заданным признакам документов, хранимых в оцифрованном представлении.

Способы поиска можно разделить на две большие группы.

1. Библиографический поиск или поиск по каталогу.
2. Тематический поиск или поиск по тексту.

Поиск по каталогу обеспечивает нахождение документов по их выходным данным, например, по названию документа, по его тематике, по именам авторов, датам публикаций. Эти выходные данные составляют реквизиты документа [34, 36, 38].

Основой каталога является предварительно заданная модель представления реквизитов, реализованная в виде базы данных, в соответствии с которой обеспечивается запись отдельных элементов реквизитов и последующий поиск по ним.

Основная проблема и недостаток такого варианта поиска – это необходимость выполнения значительного объема работ по предварительной организации, наполнению каталога. Как правило, это ручная классификация на основе привлечения экспертов. Учитывая колоссальные объемы информационных ресурсов, накопленных к настоящему времени, в совокупности с возрастающими темпами их роста становится понятным проблематичность структурирования и организации всего сегодняшнего информационного пространства.

Поиск по тексту [4, 8, 14] ориентирован на нахождение документов по их содержанию. Сюда же относится так называемый полнотекстовый поиск. Общая схема такого поиска заключается в формулировании некоторого запроса пользователем относительно содержания документа и отборе из множества доступных документов, тех, которые удовлетворяют запросу. Такой вариант поиска удобен, прежде всего, тем, что нет необходимости в предварительном разделении документов по различным категориям. Особенно это актуально при значительном объеме доступных документов, высокой динамики их обновления или отсутствии некоторых реквизитов.

Основная проблема такого поиска [16] – это сложность однозначной автоматической интерпретации содержания текстов документов и формулировок информационных потребностей пользователей. Сложность интерпретации затрудня-

ет определение соответствия рассматриваемого документа информационным потребностям пользователя.

Эти проблемы обусловлены отсутствием какой-либо регулярной структуры у текстовых документов на естественном языке. Такие информационные ресурсы принято называть неструктурированными или слабоструктурированными.

В соответствии с вышеприведенной классификацией способов поиска принято выделять два основных класса информационно-поисковых систем:

1. поисковые каталоги;
2. поисковые системы.

Поисковые каталоги в большей степени ориентированы на структурную организацию тематических коллекций с удобной системой ссылок и иерархией документов по тематическим коллекциям. Это позволяет пользователю самостоятельно находить требуемый документ, просматривая структуру каталога, либо использовать механизмы поиска, ориентированные на данный каталог. В любом случае, организация информации ее структурирование и предварительное наполнение тематического каталога является в информационно-поисковой системе первостепенным критерием, определяющим качество и эффективность поиска. Наполнение тематического каталога документами может выполняться как в ручном, так и в автоматическом режиме. Однако наиболее качественным все же остается ручной подбор документов для таких каталогов с привлечением экспертов по конкретным тематическим разделам или полуавтоматический вариант с предварительным «грубым» поиском документов и последующей их селекцией.

Поисковые системы ориентированы на поиск слабоструктурированной информации. Как правило, они используются для поиска документов в больших и динамичных информационных коллекциях, например, в Интернете. Особенностью таких коллекций является отсутствие четко выраженной структурной организации, позволяющей упорядочить и однозначно классифицировать хранящиеся в них документы по тематической направленности.

Процесс поиска текстовой информации, реализуемый типичной поисковой системой, включает в себя следующие этапы:

1. формализация пользователем поискового запроса (представление пользователем, в том или ином виде, своих информационных потребностей);
2. предварительный отбор документов по формальным признакам наличия интересующей информации (например, наличие в тексте документа одного из слов запроса, если запрос формулируется на естественном языке);
3. анализ отобранных документов (лингвистический, статистический);
4. оценка соответствия смыслового содержания найденной информации требованиям поискового запроса (ранжирование).

На сегодняшний день существуют хорошо зарекомендовавшие себя решения в области текстового поиска. Рассмотрим некоторые из них в ракурсе текстового анализа и идентификации документов.

1.1. Модели поиска

Одним из ключевых понятий, характеризующим выбор того или иного метода анализа текстовой информации, а также реализацию конкретного варианта поиска, является модель поиска [17, 3].

Модель поиска – это сочетание следующих составляющих [24, 6]:

1. способ представления документов;
2. способ представления поисковых запросов;
3. вид критерия релевантности документов.

Вариации этих составляющих определяют большое число всевозможных реализаций систем текстового поиска. Рассмотрим некоторые из них, наиболее популярные в настоящее время.

1.1.1. Простейшие модели поиска

Это модели, в которых документ представляется в виде набора ассоциированных с ним внешних атрибутов [12, 43]. К простейшим моделям поиска относится модель дескрипторного поиска и модель, основанная на Дублинском ядре.

В простейших системах дескрипторного поиска представление документа описывается совокупностью слов или словосочетаний лексики предметной области, которые характеризуют содержание документа. Эти слова и словосочетания называются дескрипторами. Индексирование документа в таких системах реализуется назначением для него совокупности дескрипторов. При этом дескрипторы могут приписываться документу:

1. на основе его содержания;
2. на основе его названия.

Эти два процесса называются соответственно индексированием по содержанию и индексированием по заголовкам документов [18].

В некоторых дескрипторных системах индексирование документов осуществляется вручную экспертами в предметной области системы, в других оно выполняется автоматически. Представление документа в дескрипторных системах называется поисковым образом документа.

Дескрипторные системы можно отнести к классу систем, ориентированных на библиографический поиск или поиск «по каталогу».

Дублинское ядро (Dublin Core) [11, 28]– это набор элементов метаданных, смысл которых зафиксирован в спецификации определяющего его стандарта. В терминах значений этих элементов можно описывать содержание различного рода текстовых документов.

Первоначальная версия Дублинского ядра была предложена в 1995 году на состоявшемся в Дублине (США) симпозиуме, организованном Online Computer Library Center (OCLC) и National Center for Supercomuting Applications (NCSA) для описания информационных ресурсов библиотечных систем.

В модели поиска, основанной на Дублинском ядре, представлением -го документа является множество пар:

$$D_k = \{(N_{i_k}, V_{i_k})\}, \quad (1)$$

где N_{i_k} – имя -го элемента метаданных Дублинского ядра в описании содержания k-го документа;
V_{i_k} – значение этого элемента метаданных.

Представлением запроса также является множество пар некоторых элементов Дублинского ядра и их значений:

$$Q = \{(N_j, V_j)\}, \quad (2)$$

где N_j – имя -го элемента метаданных Дублинского ядра в описании пользовательского запроса;
V_j – значение этого элемента метаданных.

Критерий релевантности -го документа выглядит следующим образом:

$$Q \subseteq D_k. \quad (3)$$

1.1.2. Модели, основанные на классификаторах

Это одна из разновидностей простейших моделей поиска. Документ в данной модели представляется в виде совокупности ассоциированных с ним атрибутов.

Атрибутами являются идентификаторы классов, к которым относится данный документ. Классы формируют иерархическую структуру классификатора [2, 7, 23].

Запрос может быть представлен двумя способами:

1. Простой вариант – запросом является идентификатор какого-либо класса из заданного классификатора. Критерий релевантности документа запросу – класс документа совпадает с классом в представлении запроса или является его подклассом.
2. Сложный вариант – в запросе можно указать несколько классов классификатора. Критерий релевантности документа запросу – класс документа совпадает с каким-либо из указанных в запросе классов или является его подклассом.

Модели, основанные на классификаторах, близки к булевским моделям.

1.1.3. Булевские модели

В булевских моделях [11, 40] поиска пользователь может формулировать запрос в виде булевского выражения, используя для этого операторы И, ИЛИ, НЕТ. Термы запроса зависят от конкретного варианта модели поиска. В булевской модели, ориентированной на поиск «по тексту», термами будут слова, соответственно, критерием релевантности будет условие вхождения некоторого слова или словосочетания в текст документа. В булевской модели, ориентированной на поиск по классификаторам, термами выражения будут идентификаторы классов класси-

фикатора. В булевской модели поиска с использованием Дублинского ядра термом будет значения элементов метаданных. Документ, имеющий совпадающие значения элементов метаданных со значениями, заданными в запросе, считается релевантным.

В общем случае критерием релевантности документа запросу в булевских моделях поиска является истинность булевского выражения, заданного в запросе.

Одним из несомненных достоинств булевской модели поиска является простота ее реализации. Главными недостатками считаются:

1. отсутствие возможности ранжирования найденные документы по степени релевантности, поскольку отсутствуют критерии ее оценки;
2. сложность использования – далеко не каждый пользователь может свободно оперировать булевскими операторами при формулировке своих запросов.

1.1.4. Векторные модели

В настоящее время векторные модели [5, 9, 13] являются самыми распространенными и применяемыми на практике моделями поиска. Векторные модели, в отличие от булевых, без труда позволяют ранжировать результирующее множество документов запроса.

Суть таких моделей сводится к представлению документов и запросов в виде векторов.

Каждому терму t_i в документе d_j и запросе q сопоставляется некоторый неотрицательный вес w_{ij} (w_i для запроса). Таким образом, каждый документ и запрос может быть представлен в виде -мерного вектора:

$$d_j = (w_{1j}, w_{2j}, \dots, w_{kj}), \tag{4}$$

где k- общее количество различных термов во всех документах.

Согласно векторной модели, близость документа d_i к запросу q оценивается как корреляция между векторами их описаний. Эта корреляция может быть вычислена, например, как скалярное произведение соответствующих векторов описаний.

Существуют различные подходы к выбору указанных весов. Одним из самых простых является использование нормализованной частоты данного терма в документе:

$$w_{ij} = \frac{n_{ij}}{N_j}, \tag{5}$$

где n_{ij} – количество повторений данного терма в документе;
N_j – общее количество всех термов в документе.

Более сложные варианты расчета весов учитывают частоту использования данного терма в других документах коллекции, т.е. учитывают дискриминационную силу терма. Но эти варианты возможны только при наличии статистики использования термов в коллекции.

Вариации всевозможных способов назначения весов термов и оценки меры близости векторов определяют широкий спектр различных модификаций данной модели поиска.

1.1.5. Вероятностные модели

Впервые идеи таких моделей были предложены в 1960 году [35, 20, 39, 41]. В их основе лежит принцип вероятностного ранжирования (Probabilistic Ranking Principle, PRP). Этот принцип заключается в следующем – наивысшая общая эффективность поиска достигается в случае, когда результирующие документы ранжируются по убыванию вероятности их релевантности запросу. Сначала для каждого для каждого документа оценивается вероятность того, что он релевантен запросу, а затем по этим оценкам выполняется ранжирование документов.

Существуют различные способы получения этих оценок, а также дополнительные предположения и гипотезы на основе априорных сведений относительно документов коллекции, которые и определяют конкретную реализацию вероятностной модели поиска.

Например, эта оценка может быть вычислена, в соответствии с теоремой Байеса, по некоторой функции вероятностей вхождения термов данного документа в релевантные и нерелевантные документы. С помощью запроса определяется вероятность вхождения заданного терма в релевантные документы, а по полной коллекции документов определяется вероятность вхождения этого терма в нерелевантные документы.

1.1.6. Сети вывода

Так же, как и вероятностные модели, сети вывода [44, 49] основаны на принципе вероятностного ранжирования результирующих документов поиска. Главное их отличие от вероятностных моделей заключается в том, что используется оценка не вероятности релевантности документа запросу, а вероятности того, что он удовлетворяет информационным потребностям пользователя.

В рамках данной модели процесс поиска документов описывается как процесс рассуждений в условиях неопределенности. В процессе такого рассуждения оценивается вероятность того, что информационные потребности пользователя, выраженные с помощью одного или нескольких запросов, удовлетворены.

Сеть вывода основана на Байесовской сети, которая включает узлы четырех видов. Узлами первого вида являются документы коллекции, изученные пользователем в процессе поиска. Узлами второго вида являются термы, которыми описывается содержание документов. Узлами третьего вида являются запросы, состоящие из термов, которыми описывается содержание документов. Узел четвертого типа в сети только один, и он соответствует информационным потребностям пользователя, которые не известны поисковой системе. Все узлы первого и второго вида формируются заранее для заданной коллекции. Узлы третьего вида и их связи с узлами термов, описывающих документы, и узлом информационных потребностей формируются для каждого конкретного запроса.

После того, как сеть построена, осуществляется оценка документов коллекции. Это реализуется распространением по сети оценки вероятности узла конкретного документа. Результатом распространения является вычисление вероятности узла информационных потребностей. При этом оценка для каждого документа строится независимо от оценок других документов, с учетом матриц, описывающих связи между узлами документов и узлами термов, узлами термов и узлами запросов.

Процесс оценки повторяется для каждого документа, затем они ранжируются на основе вычисленных оценок вероятности узла информационных потребностей.

1.1.7. Поиск по документу-образцу

Одной из частных задач информационного поиска является задача поиска по документу-образцу.

Документ-образец выступает в качестве одной из форм представления информационных потребностей пользователя. Целью поиска является, обнаружение тематически близких документов. При этом, как правило, речь идет не о поиске идентичных или синтаксически близких документов, а о поиске документов, близких по содержанию, близких по смыслу [21, 22].

Существует документ-образец «Рис. 1.1.» и некоторая коллекция доступных документов. Выполняется предварительный отбор из коллекции документов, и затем для отобранных документов вычисляется тематическая близость. Вычисленные оценки тематической близости используются при ранжировании документов по тематической близости к документу образцу.

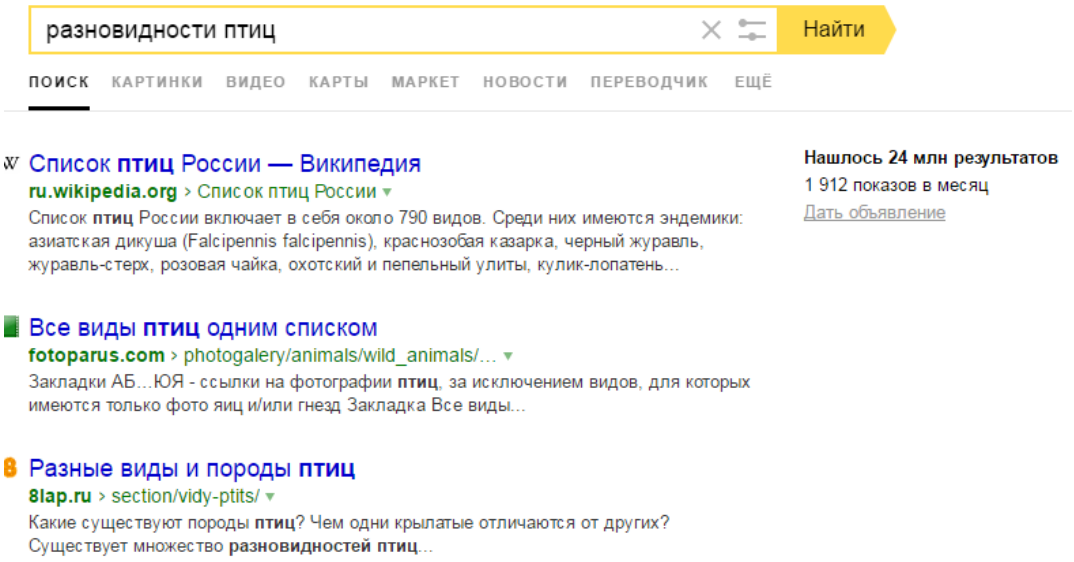

Рис. 1.1. Пример поиска документов по образцу

Самым простым подходом к решению задачи поиска документов по образцу является использование всех слов документа-образца в качестве запроса. Однако длина такого запроса может оказаться очень большой, что отрицательно скажется на качестве поиска, так как результатом поиска будут все документы, в которых присутствовали данные слова, и таких документов может быть очень много.

Приемлемым вариантом в данном случае является выделение тематики документа. Под тематикой понимается множество ключевых слов, описывающих, с некоторой степенью адекватности, содержание документа. Тематика – это приближенное представление документа. Для повышения точности и адекватности описания содержания документа ключевые слова используются с некоторыми весовыми коэффициентами, которые соотносятся с частотой повторений этих слов в тексте. Вопросы выделения тематики и вычисления тематической близости документов по их тематическому представлению во многом и определяют возможность и эффективность поиска по документу-образцу.

В работе [37] предлагается один из вариантов реализации поиска документов по образцу, автором предлагается следующая последовательность действий:

1. для каждого документа определяется некоторое относительно небольшое множество документов, представляющее его аппроксимированное тематическое окружение;
2. построенные тематические окружения анализируются с целью формирования множеств, состоящих из ключевых слов, характеризующих тематику исходного документа относительно остальных документов коллекции;
3. полученные наборы ключевых слов используются для дальнейшего вычисления относительных оценок тематического подобия.

Однако реализация такого варианта поиска предполагает предварительный анализ коллекции доступных документов, что во многом ограничивает применение данного подхода в коллекциях с высокой динамикой обновления и большим числом доступных документов.

1.1.8. Фасетный поиск

Большинство документов имеют смешанную структуру: обычно часть документа содержит явно выделенные признаки (метаданные), а часть документа неструктурированна. Чаще всего метаданные документа состоят из структурированных признаков, по которым можно классифицировать документ [32, 46].

Фасетная классификация – это совокупность нескольких независимых классификаций, осуществляемых одновременно по различным основаниям, в которой:

1. понятия представлены в виде пересечения ряда признаков (фасетной структуры);
2. классификационные индексы синтезируются посредством комбинирования фасетных признаков в соответствии с фасетной формулой.

Основой классификации является соотнесение объекта разным категориям (задание множества и его элементов).

Если структурное содержание соответствует фасетной классификации, то можно совместить поиск по тексту в неструктурированном тексте с фасетной навигацией.

Для просмотра различных аспектов содержимого документа можно сконфигурировать фасеты различных типов.

1. При одноуровневом фасете все возможные пути находятся на одном уровне. Например, фасет с именем «Область» включает в себя значения Московская, Тверская, Тульская и так далее. Выбрав фасеты, можно быстро ограничить результаты документами, содержащих указания на одну или несколько конкретных областей.
2. Иерархический фасет позволяет исследовать вложенные уровни классификации. Например, категорию «Музыка», внутри которой можно выбрать классификацию по жанру, имени исполнителя, названию песни и так далее.
3. Фасет дат представляет собой особый тип четырехуровневого иерархического фасета, который строится на основе параметрического поля даты. В состав этого фасета входят значения для года, месяца, числа и времени.
4. Фасет диапазона десятичных значений позволяет категоризировать содержимое в соответствии с диапазоном десятичных числовых значений, например, помогает ограничить поиск автомобилей моделями, доступными в конкретном диапазоне цен.

Рассмотрим на примере принцип работы фасетного поиска «Рис. 1.2.». Каждый документ связан с некоторым текстовым описанием. Пользователь выполняет поиск товара, в описании которого содержится слово «sophisticated». Затем пользователь сужает область поиска, применяя фасет «below 10$». Программа возвращает 15 товаров, соответствующих данным фильтрам, отсортировав их по рейтингу (другой фасет). Пользователь может конкретизировать свой запрос, выбрав дополнительные фасеты: тип, страна и так далее.

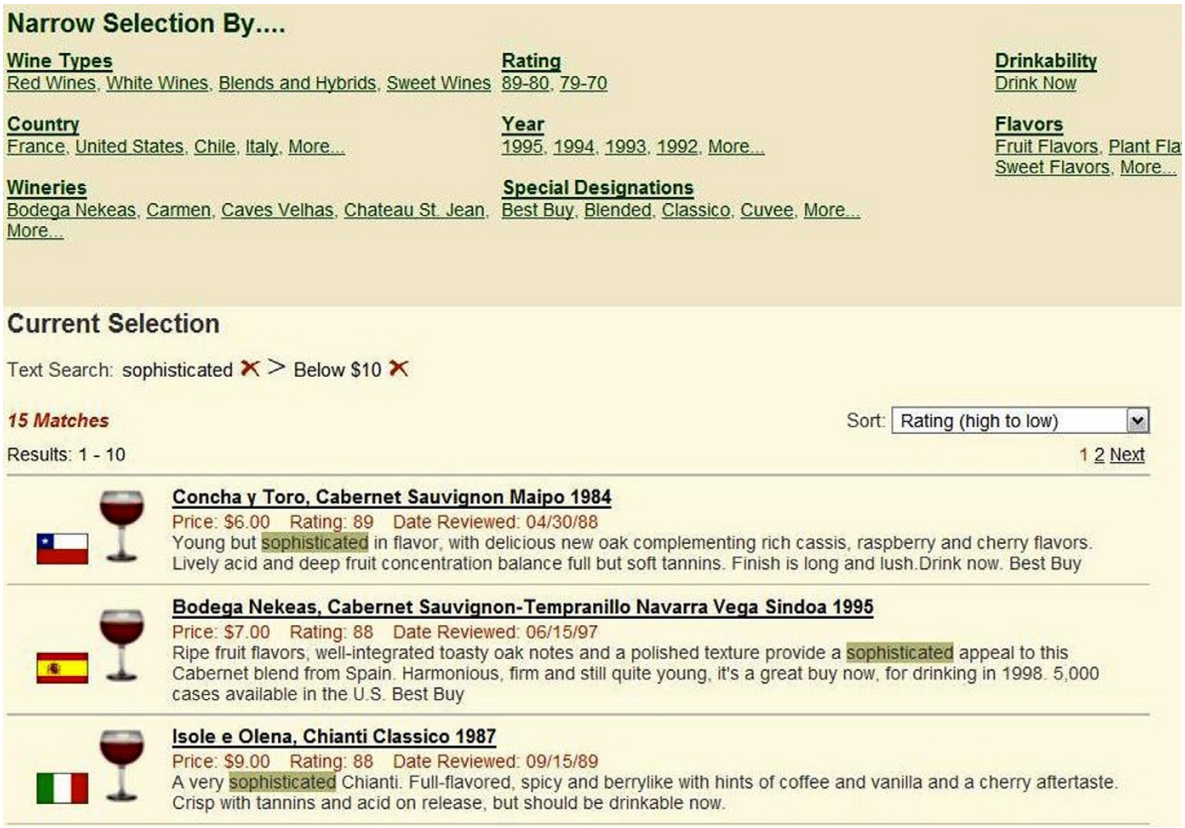

Рис. 1.2. Пример фасетного поиска

1.1.9. Комбинированный подход

Представленные на сегодняшний день в большинстве популярных поисковых систем способы организации полнотекстового поиска и методы анализа документов не учитывают в достаточной мере человеческий фактор. А именно, не учитывается тот факт, что во многом поиск определяется слабо формализуемыми и нечеткими условиями, в значительной степени зависящими от опыта и предпочтений самого человека. Далеко не всегда пользователь информационно-поисковой системы может четко и однозначно сформулировать именно тот набор ключевых слов, который приведет его к искомому результату.

В данной работе рассматривается задача анализа слабоструктурированного документа с использованием номенклатурного медицинского справочника. То есть необходимо разработать такую модель распознавания текстового запроса, которая по выделенным в тексте фармацевтическим параметрам будет выбирать результат из имеющейся коллекции выходных данных – лекарств.

Рассмотрев описанные раннее методы поиска, было принято решение скомбинировать фасетный метод с методом поиска по документу-образцу.

Исходные данные для поиска по документу-образцу – текстовый документ, который сравнивается с документами-образцами. Результатом простого поиска по документу-образцу является коллекция всех документов, содержащих хотя бы одну из указанных характеристик. К тому же метод основан на частоте появления ключевых слов в документе. Однако в поставленной задачи документом-образцом является каждая строка номенклатурного справочника, которая содержит значения признаков лекарства. Соответственно, частоту появления ключевых слов в строке подсчитать невозможно.

Исходными данными для фасетного поиска являются фасетные признаки. Фасетный поиск в большинстве случаев для данной задачи будет давать или большую коллекцию результатов или же не давать результатов совсем. Это обусловлено тем, что лекарства могут содержать большое количество одинаковых значений признаков, и при выделении малого числа фасетных признаков, поиск будет выдавать большое количество результатов. С другой стороны, текстовый запрос может содержать противоречивые фасетные признаки, тогда запрос не даст результатов.

Учитывая вышеизложенные доводы, внесем следующие изменения в алгоритм распознавания.

1. Исходными данными для нашей задачи являются документы-образцы и фасетные признаки.
2. Документом-образцом является каждая строка номенклатурного справочника, которая содержит значения признаков лекарства.
3. Для выделения фармацевтических параметров в тексте будет использоваться фасетный поиск.
4. Результат распознавания на этапе обучения модели будет проверяться пользователем.

5. Выбор набора ключевых фасетных признаков данного текстового запроса будет зависеть от оценки качества распознавания предыдущих текстовых запросов.

1.2. Ассоциативные правила

Ассоциация – это одна из задач Data Mining. [1, 30] Целью поиска ассоциативных правил (association rule) является нахождение закономерностей между связанными событиями в базах данных.

Ассоциативным правилом [10] называется импликация $X \to Y$, в которой наборы X и Y не пересекаются:

$$X \to Y: X \subset I, Y \subset I, X \bigcap Y = \emptyset. \quad (6)$$

То есть ассоциативное правило описывает закономерности вида: «Из события X следует событие Y» или «если условие, то действие». Задача поиска ассоциативных правил заключается в том, чтобы на основе имеющегося набора транзакционной базы данных D найти закономерности между событиями:

$$\tau_a \in I, a = 1,2, \dots, N_i. \quad (7)$$

1.2.1. Характеристики ассоциативных правил

Задача построения ассоциативных правил связана с необходимостью вычисления поддержки и достоверности правил a. Набор $X \subset I$ из базы D имеет поддержку $supp(X)$, определяемую как отношение количества транзакций T в наборе данных D, содержащих множество элементов X, к общему количеству транзакций в базе данных D.

Поддержкой $s = supp(X \to Y)$ правила $X \to Y$ является поддержка множества $X \cup Y$:

$$supp(X \to Y) = supp(X \cup Y). \quad (8)$$

Достоверностью $c = conf(X \to Y)$ правила $X \to Y$ называют отношение его поддержки $supp(X \to Y)$ к поддержке $supp(X)$множества X. Достоверность правила показывает, что $c\%$ транзакций из всего множества, содержащих набор элементов X, также содержат набор элементов Y.

При помощи использования алгоритмов поиска ассоциативных правил аналитик может получить все возможные правила вида "Из X следует Y", с различными значениями поддержки и достоверности. Однако в большинстве случаев, количество правил необходимо ограничивать заранее установленными минимальными и максимальными значениями поддержки и достоверности.

Если значение поддержки правила слишком велико, то в результате работы алгоритма будут найдены правила очевидные и хорошо известные. Слишком низкое значение поддержки приведет к нахождению очень большого количества правил, которые, возможно, будут в большей части необоснованными, но не известными

и не очевидными для аналитика. Таким образом, необходимо определить такой интервал, который с одной стороны обеспечит нахождение неочевидных правил, а с другой – их обоснованность.

В результате процесс синтеза ассоциативных правил можно разбить на два этапа:

1. генерирование всех наборов X с уровнем поддержки, не ниже заданного экспертом порогового значения $minsupport(X)$, в результате чего формируются часто встречаемые наборы $X \subset I$;
2. генерирование всех правил $X \to Y$ с уровнем достоверности, не ниже заданного экспертом порогового значения $minconfidence(X \to Y)$.

1.2.2. Виды ассоциативных правил

При поиске взаимосвязей между различными элементами в транзакционных базах данных $D = \{T_1, T_2 \dots T_{N_T}\}$ часто необходимо выявлять не только, так называемые, позитивные ассоциативные правила (positive association rules) $X \to Y$, но и другие виды правил. К таким правилам относятся: негативные, численные, обобщенные, временные и нечеткие ассоциативные правила.

Негативные ассоциативные правила (negative association rules) характеризуют отрицательную взаимосвязь между различными событиями типа: «Если произошло событие X, то событие Y не наступит» ($X \to \bar{Y}$) или «Если не произошло событие X, то наступит событие Y» ($\bar{X} \to Y$).

Необходимость извлечения негативных ассоциативных правил $X \to \bar{Y}$ или $\bar{X} \to Y$ наряду с позитивными правилами $X \to Y$ обуславливается следующей причиной. Построение полного набора ассоциативных правил ($X \to Y$, $X \to \bar{Y}$, $\bar{X} \to Y$) между различными объектами $\tau_a \in I, a = 1,2, \dots, N_i$ базы данных $D = \{T_1, T_2 \dots T_{N_T}\}$ позволит более детально описать исследуемые зависимости, что в свою очередь приведет к более точным результатам прогнозирования по синтезированной базе правил.

Для поиска интересных негативных правил (таких, которые представляют интерес в конкретной прикладной области в соответствии с заданным набором данных D) необходимо учитывать уровень их интереса, определяемый в соответствии с критерием Пятецкого-Шапиро следующим образом:

$$supp(X \to \bar{Y}) - supp(X)supp(\bar{Y}) \geq \varepsilon_1. \quad (9)$$

При выполнении неравенства (9) правила $X \to \bar{Y}$ считаются интересными. Аналогичным образом можно определить и неравенства для поиска интересных негативных правил типа $\bar{X} \to Y$.

Таким образом, при извлечении негативных правил $X \to \bar{Y}$ из набора данных D происходит поиск таких транзакций T_j, в результате чего извлекается набор негативных правил $X \to \bar{Y}$, удовлетворяющих списку условий:

$$\begin{cases} (supp(X) \geq minsupport) \cap (supp(Y) \geq minsupport) \cap \\ \cap (supp(X \to \bar{Y}) \geq minsupport); \\ conf(X \to Y) \geq minsupport; \\ |supp(X \to \bar{Y}) - supp(X)supp(\bar{Y})| \geq \varepsilon_1. \end{cases} \quad (10)$$

Приведенные условия позволяют выявлять достоверные негативные правила $X \to \bar{Y}$ с приемлемым уровнем поддержки и являющиеся интересными в исследуемой предметной области.

При идентификации негативных ассоциативных правил необходимо обрабатывать нечастые последовательности в заданной базе данных D. Однако большинство методов синтеза ассоциативных правил основаны на извлечении и анализе часто встречаемых наборов, что затрудняет их применение на практике для поиска негативных ассоциативных правил, и обуславливает необходимость разрабатывать новые методы синтеза ассоциативных правил, позволяющие извлекать как позитивные, так и негативные ассоциативные правила.

Часто признаки $\tau_a \in I, a = 1,2, \dots, N_i$ могут принимать не только бинарные, но и численные значения из некоторого диапазона значений $\tau_a \in \left[\tau_{a_{min}}; \tau_{a_{max}}\right]$ или множества значений $T(\tau_a) = \left\{\tau_{a1}, \tau_{a2}, \dots, \tau_{aN_{\tau_a}}\right\}$.

Поэтому актуальной является задача выделения правил вида: если $X \in [X_{min}; X_{max}]$, то $Y \in [Y_{min}; Y_{max}]$.

Численным ассоциативным правилом (quantitative association rule) называется импликация вида:

$$\langle X, v(X)\rangle \to \langle Y, v(Y)\rangle, v(X) \in T(X), v(Y) \in T(Y), \quad (11)$$

где $v(X), v(Y)$ – значения переменных X и Y,
$T(X)$ и $T(Y)$ – множества возможных значений.

Поддержка $supp(X \to Y)$ численного ассоциативного правила $X \to Y$ вида (11) определяется:

$$supp(X \to Y) = \frac{1}{|D|}\sum_{j=1}^{|D|}\prod_{i=1}^{N_X+N_Y} v_j(\tau_a), \tau_a \in (X \cup Y), \quad (12)$$

где N_X и N_Y – количество элементов $\tau_a \in I$ в множествах $X = \left\{\tau_1, \tau_2, \dots, \tau_{N_X}\right\}$ и $Y = \left\{\tau_{N_X+1}, \tau_{N_X+2}, \dots, \tau_{N_X+N_Y}\right\}$ соответственно;
$v\_j(\tau_a)$ – значение a -го признака τ_a в j-й транзакции T_j базы данных D .

Достоверность $conf(X \to Y)$ численного ассоциативного правила $X \to Y$ вида (11) определяется аналогично позитивным АП. При этом поддержка $supp(X)$ множества X вычисляется:

$$supp(X) = \frac{1}{|D|}\sum_{j=1}^{|D|}\prod_{i=1}^{N_X} v_j(\tau_a), \tau_a \in X. \quad (13)$$

Процесс поиска численных ассоциативных правил вида (11) по заданным наборам данных D связан с необходимостью разбиения на интервалы диапазонов возможных значений элементов $\tau_a \in I$, входящих в транзакции $T_j, j = 1,2, \dots N_T$. В результате такого разбиения каждая j-я транзакция $T_j = \left(tid_j, item_j\right)$ представляется списком элементов $item_j = \left\{t_{1j}, t_{2j}, \dots, t_{N_{item_j}j}\right\} \subseteq I$, в котором каждый i-й элемент t_{ij}представляется в виде:

$$t_{ij} = (\text{элемент } \tau_a \in I;\ \text{диапазон значений элемента } \tau_a),$$
$$I = \left\{\tau_1, \tau_2, \dots, \tau_{N_I}\right\}, \tag{14}$$

где I – множество возможных значений, которые могут входить в список элементов $item_j$ каждой транзакции T_j, содержит элементы τ_a:

$$\tau_a \in [\tau_{a\,min}; \tau_{a\,max}\,] = \bigcup_{c=1}^{N_{\text{разб}\,\tau_a}} [\tau_{a\,\min c}; \tau_{a\,\max c}\,], a = 1, 2, \dots, N_1\,, \tag{15}$$

где $\tau_{a\,min}$, $\tau_{a\,max}$ – минимальное и максимальное значение, которые может принимать a -й элемент τ_a множества I;

$\tau_{a\,\min ñ}$ и $\tau_{a\,\max ñ}$ – минимальное и максимальное значение c -го интервала разбиения значений a-го элемента τ_a множества I;

$N_{\text{разб}\,\tau_a}$ – количество интервалов разбиения a -го элемента τ_a.

После дискретизации значений численных переменных выполняется поиск ассоциативных правил $X \to Y$. При этом используются методы извлечения ассоциативных правил, удовлетворяющих приведенным выше условиям к позитивным правилам вида $X \to Y$, но при таком поиске каждый диапазон дискретизации каждой переменной считается отдельным элементом, который может быть использован при построении ассоциативного правила. Однако необходимость дискретизации диапазонов значений переменных для извлечения численных ассоциативных правил существенно увеличивает пространство поиска и требования к вычислительным ресурсам ЭВМ.

1.2.3. Методы поиска ассоциативных правил

На сегодняшний день существует большое количество методов поиска ассоциативных правил в разных источниках данных. Основными являются методы AIS и SETM. Рассмотрим подробнее каждый из этих методов [47, 25 – 27, 33, 42].

1.2.3.1. Алгоритм AIS

Первый алгоритм поиска ассоциативных правил, называвшийся AIS, (предложенный Agrawal, Imielinski and Swami) был разработан сотрудниками исследовательского центра IBM Almaden в 1993 году.

В алгоритме AIS кандидаты множества наборов генерируются и подсчитываются «на лету», во время сканирования базы данных. Каждая транзакция проверяется на наличие больших наборов, выявленных при предыдущем проходе. Соот-

ветственно, новые наборы формируются путем расширения имеющихся наборов. Этот алгоритм неэффективен, поскольку генерирует и учитывает слишком много наборов-кандидатов, которые недостаточно большие (нечастые).

1.2.3.2. Алгоритм SETM

Создание этого алгоритма было мотивировано желанием использовать язык SQL для вычисления часто встречающихся наборов товаров. Как и алгоритм AIS, SETM также формирует кандидатов «на лету», основываясь на преобразованиях базы данных. Чтобы использовать стандартную операцию объединения языка SQL для формирования кандидата, SETM отделяет формирование кандидата от их подсчета.

Неудобство алгоритмов AIS и SETM − излишнее генерирование и подсчет слишком многих кандидатов, которые в результате не оказываются часто встречающимися. Для улучшения их работы был предложен алгоритм Apriori.

1.2.3.3. Алгоритм Apriori

Работа данного алгоритма [45, 48] состоит из нескольких этапов, каждый из этапов состоит из следующих шагов:

1. формирование кандидатов;
2. подсчет кандидатов.

Формирование кандидатов (candidate generation) − этап, на котором алгоритм, сканируя базу данных, создает множество i -элементных кандидатов (i − номер этапа). На этом этапе поддержка кандидатов не рассчитывается.

Подсчет кандидатов (candidate counting) − этап, на котором вычисляется поддержка каждого i -элементного кандидата. Здесь же осуществляется отсечение кандидатов, поддержка которых меньше минимума, установленного пользователем *(min_sup)*. Оставшиеся i -элементные наборы называются часто встречающимися.

Рассмотрим работу алгоритма Apriori на примере базы данных D «Рис. 1.3.». Минимальный уровень поддержки равен 3.

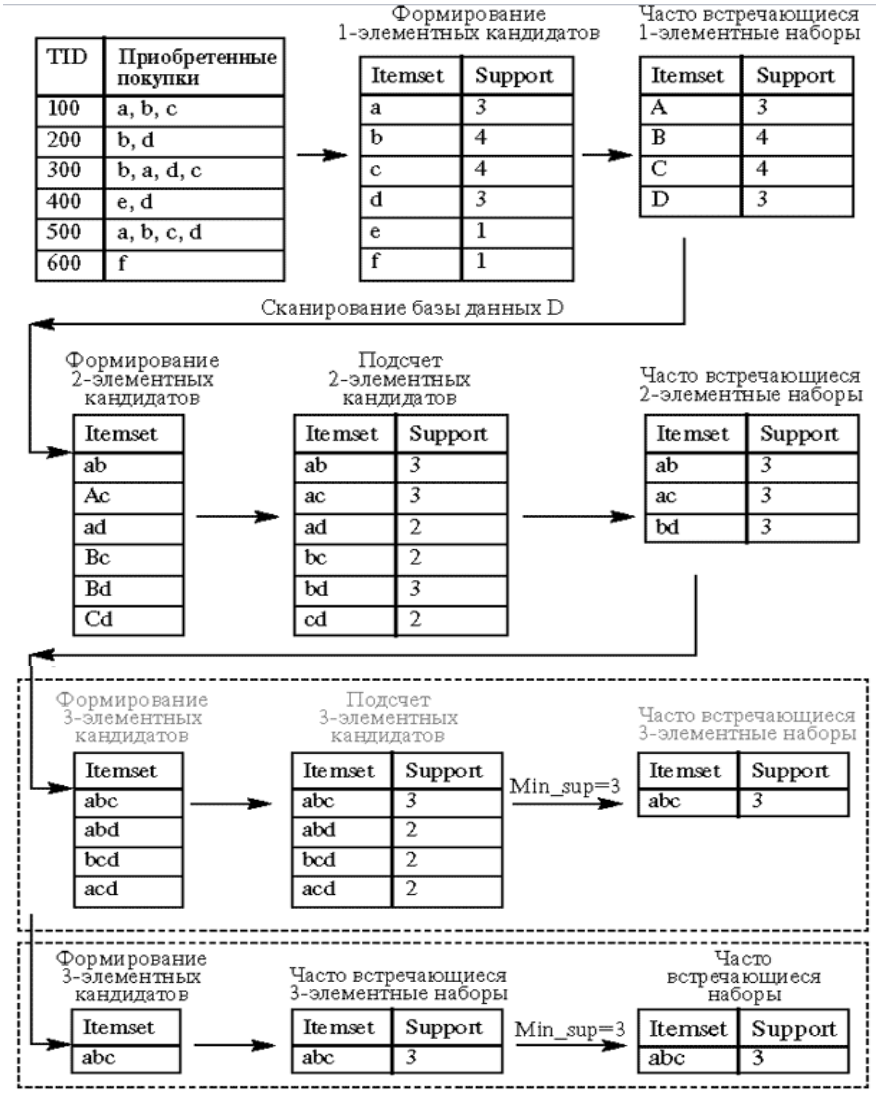

Рис. 1.3. Пример работы алгоритма Apriori

На первом этапе происходит формирование одноэлементных кандидатов. Далее алгоритм подсчитывает поддержку одноэлементных наборов. Наборы с уровнем поддержки меньше установленного − 3, отсекаются. В нашем примере это наборы e и f, которые имеют поддержку, равную 1. Оставшиеся наборы товаров считаются часто встречающимися одноэлементными наборами товаров: это наборы a, b, c, d.

Далее происходит формирование двухэлементных кандидатов, подсчет их поддержки и отсечение наборов с уровнем поддержки, меньшим 3. Оставшиеся двухэлементные наборы товаров, считающиеся часто встречающимися двухэлементными наборами ab, ac, bd, принимают участие в дальнейшей работе алгоритма.

Если смотреть на работу алгоритма прямолинейно, на последнем этапе алгоритм формирует трехэлементные наборы товаров: abc, abd, bcd, acd, подсчитывает их поддержку и отсекает наборы с уровнем поддержки, меньшим 3. Набор товаров abc может быть назван часто встречающимся.

Однако алгоритм Apriori уменьшает количество кандидатов, отсекая тех, которые заведомо не могут стать часто встречающимися, на основе информации об отсеченных кандидатах на предыдущих этапах работы алгоритма.

Отсечение кандидатов происходит на основе предположения о том, что у часто встречающегося набора товаров все подмножества должны быть часто встречающимися. Если в наборе находится подмножество, которое на предыдущем этапе было определено как нечасто встречающееся, этот кандидат уже не включается в формирование и подсчет кандидатов.

Так наборы товаров ad, bc, cd были отброшены как нечасто встречающиеся, алгоритм не рассматривал товаров abd, bcd, acd.

При рассмотрении этих наборов формирование трехэлементных кандидатов происходило бы по схеме, приведенной в верхнем пунктирном прямоугольнике. Поскольку алгоритм априори отбросил заведомо нечасто встречающиеся наборы, последний этап алгоритма сразу определил набор abc как единственный трехэлементный часто встречающийся набор.

Алгоритм Apriori рассчитывает также поддержку наборов, которые не могут быть отсечены априори. Это так называемая негативная область (negative border), к ней принадлежат наборы-кандидаты, которые встречаются редко, их самих нельзя отнести к часто встречающимся, но все подмножества данных наборов являются часто встречающимися.

1.2.4. Использование ассоциативных правил в поставленной задаче

Ассоциативные правила определяют закономерности между связанными событиями в базах данных. В поставленной задаче ассоциативное правило будет описывать закономерности вида: «Из набора найденных в тексте фармацевтических признаков X следует лекарство Y из номенклатурного справочника». Задача поиска ассоциативных правил будет заключаться в том, чтобы на основе имеющегося набора текстовых запросов выявить закономерности между наборами фармацевтических признаков и лекарствами из справочника.

Каждый выявленный набор фармацевтических признаков записывается в базу данных с соответствующими характеристиками: поддержкой S и достоверностью C. Поддержка набора будет показывать, что $S\%$ наборов фармацевтических признаков, содержащихся в базе данных, содержат данный набор. Достоверность набора будет показывать, что $C\%$ наборов фармацевтических признаков из всего множества, содержащих набор фармацевтических признаков X были распознаны правильно.

1.3. Выводы по разделу

В настоящее время существуют различные методы текстового поиска, основанные на разных способах представления документов, поисковых запросов и релевантности документов. Такое многообразие выбора входных и выходных дан-

ных модели позволяет комбинировать методы распознавания текстовых запросов для решения конкретных задач.

В данной работе рассматривается задача анализа слабоструктурированного документа с использованием номенклатурного медицинского справочника. Для решения данной задачи принято решение скомбинировать фасетный метод с методом поиска по документу-образцу. Исходными данными будут являться документы-образцы: строки номенклатурного справочника и фасетные признаки: фармацевтические параметры лекарств. Оценка качества распознавания на этапах обучения модели будет определяться пользователем.

Для выявления соответствий между фасетными признаками и медицинскими лекарствами из номенклатурного справочника будет решаться задача поиска ассоциативных правил. Ассоциативное правило будет описывать закономерности вида: «Из набора найденных в тексте фармацевтических признаков X следует лекарство Y из номенклатурного справочника». Каждый выявленный набор фармацевтических признаков имеет соответствующие характеристики: поддержку S и достоверность C.

2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ДЛЯ РАСПОЗНАВАНИЯ ТЕКСТОВЫХ ЗАПРОСОВ

Исходными данными для разработки адаптивной модели распознавания текстовых запросов являются номенклатурный справочник лекарственных препаратов с двадцатью фармацевтическими параметрами, а также шаблоны для поиска фармацевтических параметров, используемых при распознавании наименований лекарств.

Необходимо разработать самообучающуюся систему распознавания текстовых запросов для определения лекарства из каталога, требуемого клиенту.

Обучающая выборка состоит из более чем ста тысяч текстовых запросов с известными результатами распознавания – лекарственными препаратами.

Имеется набор лекарств:

$$Drug = \{Drug_1, \dots, Drug_x\}, \tag{16}$$

где $Drug_i$ – i -е лекарство, $i = \overline{1, x}$;
x – количество всевозможных лекарств.

Дан номенклатурный справочник – каталог используемых лекарств $Card$,

$$Card = (cardID, A_{m\times n}, Drug_i), i = \overline{1, m}, j = \overline{1, n}, \tag{17}$$

где m – количество строк в каталоге;
n – количество признаков в каталоге;
$cardID$ – уникальный идентификатор в каталоге;
$A_{m\times n} = \{a_{ij}\}$– значения признаков для -го лекарства.

Дана база данных:

$$D = \{T_1, \dots, T_{N_D}\}, \tag{18}$$

где T_j – транзакция, $j = \overline{1, N_D}$;
$N_D = |D|$ – мощность БД.

Элементы T_j представлены в виде:

$$T_j = \left(T_{ID\,j}, Text_j, item_j, waiting_{ID}, reality_{ID}\right), \tag{19}$$

где $T_{ID\,j}$ – идентификатор -й транзакции T_j;
$Text_j$ – строка запроса пользователя (транзакция);
$item_j = \{\dots (e_l, mark_l,) \dots\}, 1 \le l \le p$ – набор эвристических правил в -ой транзакции;
e_l – номер эвристического правила, найденного в текущей транзакции;
$mark_{ji}$ – значение эвристического правила e_l;
$waiting_{ID}$ – идентификатор $cardID$ лекарства, распознанного системой;
$reality_{ID}$ – идентификатор $cardID$ лекарства, распознанного человеком.

Существует набор шаблонов $Pattern$которые применяются для распознавания заданной строки:

$$Pattern = \{pat_1, \dots, pat_k\}, \quad (20)$$

где k – количество всевозможных шаблонов.

Каждому шаблону в соответствие ставится некоторое эвристическое правило $E = \{e_1, \dots, e_p\}$, которое определяет значение только одного из признаков $Feature = \{f_1, \dots, f_n\}$.

Одному эвристическому правилу может соответствовать несколько шаблонов, одному признаку может соответствовать несколько эвристических правил.

Каждое эвристическое правило e_i имеет две характеристики: S_{e_i} и C_{e_i}, [1, 17] с помощью которых можно оценить качество распознавания в тексте,

где S_{e_i} – поддержка эвристического правила,

C_{e_i} – достоверность эвристического правила.

Набор эвристических правил [12], найденный в процессе распознавания из $Text_j$, с соответствующими характеристиками записываются в базу данных эвристик:

$$D_e = \{set_1, \dots, set_i, \dots\}, \quad (21)$$

где $set_i = \{< \dots, \mathrm{e}_l, \dots >, S, C\}$, – набор состоящий из эвристических правил e_l, найденных в процессе распознавания,

S – поддержка всего набора эвристических правил,

C – достоверность всего эвристических правил.

Необходимо для строки $s_p, p = N_D + 1$ выявить имеющиеся признаки, и по ним, в соответствии с каталогом, определить лекарство $Drug_i$, требуемое пользователю. С помощью обучения настроить заданную модель.

Обучение модели происходит непрерывно, после каждого распознавания нового текстового запроса. Пользователь принимает решение согласиться или не согласится с ответом, предложенным программой. В зависимости от результатов распознавания необходимо пересчитывать характеристики набров эвристических правил, повлиявших на результат.

2.1. Допущения для математической модели

Так как в данном виде поставленная задача сложна для реализации, введем некоторые допущения, для упрощения математической модели.

1. В одном текстовом запросе пользователя содержится информация только об одном заказе. Программа должна выбрать наиболее подходящий под выявленные признаки результат.

2. В каждой строке номенклатурного справочника у каждого фармацевтического признака может быть только одно значение, то есть не рассматриваются конструкции вида «И», «ИЛИ».
3. Значения признаков представлены в виде числа или элемента из конечного множества. То есть для распознавания текстовых запросов в задаче будут применяться текстовые шаблоны.
4. Каждое эвристическое правило определяет значение только одного признака.
5. Каждый раз, после нахождения результата распознавания, пересчитываются характеристики для всего найденного эвристического набора. Будем считать, что все эвристики, входящие в данный набор, в одинаковой степени повлияли на полученный результат. Такое допущение позволит не отслеживать взаимосвязи между признаками и эвристическими правилами, по которым находятся данные признаки.
6. Человек отвечает на те текстовые запросы, в которых невозможно автоматизировано определить результат.

2.2. Простая и сложная постановки задачи

Постановку задачи можно рассматривать в двух аспектов. В первом случае одинаковым текстовым запросам соответствуют одинаковые варианты ответа. То есть, если при первом распознавании текстового запроса программа выбрала правильный ответ, то, в следующий раз, выделив из такого же текстового запроса такой же набор эвристических правил, программа гарантированно получит правильный ответ. Данную формулировку будем считать легкой постановкой задачи. Такая постановка задачи характерна для текстовых запросов с четко заданными параметрами, подходящими только к конкретному лекарству из номенклатурного справочника.

В сложной постановке задачи одинаковым текстовым запросам могут соответствовать разные варианты ответа. То есть, если при первом распознавании текстового запроса программа выбрала правильный ответ, то, в следующий раз выделив из такого же текстового запроса такой же набор эвристических правил, программа может не получить правильный ответ. Данная постановка задачи характерна для текстовых запросов, содержащих большое количество параметров, подходящих для нескольких лекарств, а также для текстовых запросов почти или совсем не содержащих информации о параметрах лекарств.

В работе будет разработана модель распознавания для легкой постановки задачи «Приложение 2. Пример работы адаптивной модели распознавания текстового запроса».

2.3. Характеристики эвристических наборов

Закономерности между наборами фармацевтических признаков и медицинскими лекарствами из справочника будут определяться с помощью ассоциативных правил. Ассоциативное правило будет описывать закономерности вида: «Из набора найденных в тексте фармацевтических признаков X следует лекарство Y из номенклатурного справочника».

Каждый набор эвристических правил set_i, с помощью которых были выявлены фармацевтические признаки, хранится в базе данных эвристик D_e с соответствующими характеристиками: поддержкой S_i и достоверностью C_i. Поддержка набора показывает, что $S_i\%$ наборов эвристических правил, содержащихся в базе данных, содержат данный набор. Достоверность набора показывает, что $C_i\%$ наборов эвристических правил из всего множества, содержащих эвристический набор X были распознаны правильно.

Поддержка эвристического набора определяется как:

$$S_i = \frac{\text{количество примеров содержащих такой же набор эвристических правил}}{\text{общее количество примеров}}. \quad (22)$$

Достоверность эвристического набора определяется как:

$$C_i = \frac{\text{количество правильных примеров} - \text{количество неправильных примеров, содержащих такой же набор}}{\text{общее количество примеров, содержащих данный набор}}. \quad (23)$$

Поддержка эвристического набора всегда неотрицательна, достоверность эвристического набора может принимать отрицательные значения.

Начальное значение поддержки первого набора попавшего в базу данных эвристик D_e:

$$S_0 = 0{,}2. \quad (24)$$

Начальное значение достоверности первого набора попавшего в базу данных эвристик D_e:

$$C_0 = 0{,}2. \quad (25)$$

Поддержка и достоверность наборов эвристических правил будут меняться в процессе обучения модели. Если программа выбрала лекарство, совпавшее с эталонным, то характеристики набора, выбравшего фармацевтические признаки, увеличатся. Соответственно, если выбранный программой ответ был ошибочным, характеристики набора уменьшатся.

2.4. Выводы по разделу

Описана математическая модель задачи. Исходными данными модели распознавания являются номенклатурный справочник и фармацевтические параметры лекарств. Необходимо разработать самообучающуюся систему распознавания текстовых запросов, которая по выделенным в тексте фармацевтическим пара-

метрам будет выбирать результат из имеющейся коллекции выходных данных – лекарств. Для корректировки системы распознавания каждому эвристическому набору ставится в соответствие две характеристики: достоверность и поддержка набора.

Для упрощения математической модели описаны правила, которых будем придерживаться при разработке программы.

В главе представлены две различных постановки задачи. В простой постановке задачи одинаковым текстовым запросам соответствуют одинаковые варианты ответа. В сложной постановке задачи одинаковым текстовым запросам могут соответствовать разные лекарства.

3. РАЗРАБОТКА МОДЕЛИ РАСПОЗНАВАНИЯ ТЕКСТА

На вход модели подается текстовый запрос. Модель выявляет набор признаков медицинских лекарств, содержащихся в запросе. Выделенному набору признаков ставится в соответствие список лекарств из номенклатурного справочника. Пользователь выбирает наиболее подходящее лекарство из полученного списка. Выходным значением модели является лекарство, выбранное пользователем.

Опишем описанную выше модель распознавания более подробно.

3.1. Конфликтные ситуации при распознавании текстовых запросов

В каждом текстовом запросе необходимо выделять фармацевтические параметры для дальнейшей процедуры выбора соответствующего лекарства. Для выявления фармацевтических параметров в тексте будет использоваться фасетный поиск.

В текстовом запросе последовательно ищутся все фасетные признаки (эвристические правила). Для этого к тексту применяются все шаблоны, которые соответствуют различным эвристическим правилам «Приложение 1. Формат входных и выходных данных». В процессе распознавания значений признаков, в текстовом запросе могут возникать следующие конфликтные ситуации между эвристическими правилами:

1. два и более эвристических правила в одной и той же части текста выявили разные значения признаков;
2. два и более эвристических правила выявили разные значения одного и того же признака в тексте;
3. найденному набору эвристических правил не соответствует ни одно лекарство из номенклатурного справочника.

Рассмотрим подробнее каждую конфликтную ситуацию и алгоритм её разрешения.

3.1.1. Эвристические правила, принадлежат одной части текста

Данный вид конфликта возникает при применении каждого шаблона к тексту. Для каждого найденного шаблона определяется интервал покрытия в тексте. Всем найденным шаблонам ставятся в соответствие эвристические правила. Интервалы покрытия текста могут пересекаться только в том случае, если соответствующие им шаблоны принадлежат одному эвристическому правилу. В противном случае возникает конфликтная ситуация.

На «Рис. 3.1.»приведен пример данной конфликтной ситуации. К данному тексту были применены два шаблона: «#г/#мл» и «#мл», которые выявили соответственно следующие эвристические правила «дозировка 100,0мг/5мл» и «объем

5мл». Необходимо выбрать какое из найденных эвристических правил нужно исключить.

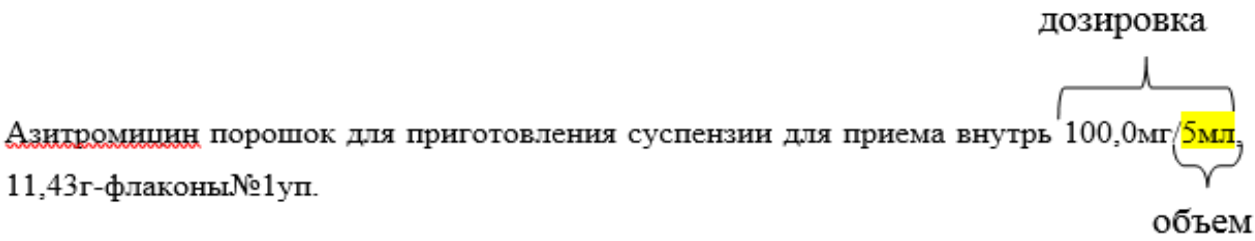

Рис. 3.1. Пример конфликтной ситуации

Для разрешения данного конфликта был предложен следующий алгоритм.

1. Генерируется набор, содержащий все найденные неконфликтные эвристические правила, и к нему добавляются всевозможные наборы из конфликтных значений (кроме $\emptyset$ и всего конфликтного набора).
2. Среди этого множества выбираются 10% от всех сгенерированных наборов с усредненным лучшим показателем характеристик $S * C$. Если такого набора еще не существует в базе данных эвристик, считаем его S, C. Они определяются как минимально установленные значения (т.е. $S := S_0, C := C_0$).

3.1.2. Эвристические правила, выявили разные значения одного признака

Данный вид конфликта возникает при сопоставлении каждому найденному эвристическому правилу соответствующего признака. Мы накладывали условия на задачу о том, что в текстовом запросе результатом распознавания может быть только один лекарственный препарат и в каждой строке номенклатурного справочника каждый признак имеет не более одного значения. Соответственно, если в текстовом запросе находятся два и более различных значений одного и того же признака, возникает конфликтная ситуация.

На «Рис. 3.2.» приведен пример данной конфликтной ситуации. В тексте найдены два разных значения одного и того же признака – лекарственная форма. Необходимо выбрать одну из форм лекарственного препарата.

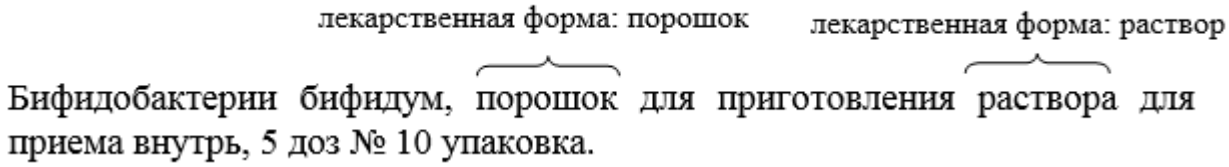

Рис. 3.2. Пример конфликтной ситуации

3.1.3. Набор эвристических правил при пересечении с номенклатурным справочником дает пустое множество

Данная конфликтная ситуация возникает при сопоставлении выявленных признаков в текстовом запросе со строками номенклатурного справочника. Для того чтобы выбрать лекарство из справочника необходимо, чтобы найденные признаки в тексте были подмножеством признаков в строке справочника. Если найденные признаки не являются подмножеством ни одной из строк номенклатурного справочника возникает конфликтная ситуация, так как невозможно сопоставить лекарство текстовому запросу.

На «Рис. 3.3» приведен пример данной конфликтной ситуации. Выделенные признаки не являются подмножеством ни одной из строк номенклатурного справочника. Необходимо выделить другой набор эвристических правил или уменьшить данный набор, так чтобы он давал соответствие со справочником.

Номенклатурный справочник

Название лекарственного препарата	Название	Вид	Лекарственная форма	Объем (мл; г)	№
Аира корневища КЛС уп 75г	Аира корневища	корневища	измельченные	75	1
Аира корневища уп 50г	Аира корневища	корневища	измельченные	50	1
Аира корневища ф-пак 1,5г №20	Аира корневища	корневища	измельченные	1,5	20
Аира корневища КЛС ф-пак 1,5г №20	Аира корневища	корневища	порошок	1,5	20

Запрос:

Необходимы аира корневища, в виде измельченного порошка,50г.

Выявленные признаки: Аира корневища, порошок, 50г.

Рис. 3.3. Пример конфликтной ситуации

Для выбора одного значения признака из найденных эвристическими правилами и устранения пустого множества при пересечении набора эвристических правил с каталогом был предложен следующий алгоритм.

1. Если одно и то же эвристическое правило дает разные значения одного и того же признака, то выбирается первый сгенерированный набор.
2. Если разные эвристические наборы выявляют одинаковое значение одного и того же признака, то генерируется набор, содержащий все найденные эвристические правила.
3. Генерируются всевозможные наборы из значений первых N значимых признаков.
4. Среди них выбираются наборы, которые есть в каталоге. Если ни один из наборов не найден, то уменьшается число значимых признаков, переходим к предыдущему шагу.
5. Среди найденных наборов выбираются 10% наборов с усредненным лучшим показателем характеристик $S * C$. Зафиксированы первые N признаков для каждого оставшегося набора.
6. Для каждого последующего признака генерируются всевозможные значения и добавляются к существующим зафиксированному набору.

7. Среди них выбираются наборы, которые есть в каталоге.
8. Среди найденных наборов выбираем наборы с усредненным лучшим показателем характеристик $s * c$. Если ни один из наборов не найден, убираем значение данного признака.
9. Добавляем найденное значение признака к зафиксированному набору, переходим к следующему признаку.

Все разработанные алгоритмы разрешают конфликтные ситуации руководствуясь соответствием с номенклатурным справочником и выбирая эвристические правила с лучшими показателями характеристиками. Данный подход позволяет учитывать предыдущий опыт распознавания текстовых запросов и адаптировать модель в соответствии с полученными результатами распознавания.

3.2. Процедура пересчета характеристик эвристических наборов

После распознавания текстового запроса и выбора лекарственного препарата происходит корректировка обучающей модели распознавания.

Пересчет характеристик эвристических наборов происходит следующим образом.

1. Набор эвристических правил set_i, с помощью которого были выявлены фармацевтические признаки, записывается в базу данных эвристик D_e вместе со своими характеристиками S, C. Если такой набор уже существует, то пересчитываются только характеристики набора S, C в соответствии с формулами (22,23).
2. Корректируется число правильно и неправильно распознанных текстов с помощью данного набора.
3. Генерируются всевозможные наборы $newset_k$ из данного эвристического набора set_i и проверяется наличие каждого в базе данных эвристик.
$$newset_k \, \epsilon \, set_i, k = \overline{1..2^l}, l = |set_i|.$$
4. Если набор $newset_k$ существует, то у него корректируется число правильно или неправильно распознанных текстов. Таким образом корректируются характеристики всех подмножеств выявленного набора эвристических правил.

3.3. Процедура выбора строки из номенклатурного справочника по набору эвристических правил

После разрешения всех конфликтных ситуаций формируется окончательный набор эвристических правил set_i, выявленных в текстовом запросе $Text_j$. Этому набору необходимо поставить в соответствие строку из номенклатурного справочника.

Выбираются все строки номенклатурного справочника, которые содержат выявленный набор эвристических правил. Если таких строк меньше 10, то из них выбирается строка, лекарственный препарат $Drug_i$ которой сходится с эталонным

$reality\_ID$. Если такой строки нет, то текстовый запрос считается распознанным неверно.

Если выбранных строк номенклатурного справочника больше 10, считается, что ответ неоднозначный и пользователю лучше самому проверить данный текстовый запрос.

3.4. Алгоритм адаптивной модели распознавания текстовых запросов

В данном алгоритме указана последовательность применения выше описанных процедур для корректной работы модели распознавания текстовых запросов.

1. Объявление входных данных: запрос пользователя в формате строки s_p; каталог используемых лекарств $Card$; набор признаков $Feature$; набор эвристических правил E; набор шаблонов $Pattern$.
2. К исходной строке последовательно применяются все шаблоны pat_i. Определяется интервал покрытия текста каждым шаблоном $[a_{pat_i}; b_{pat_i}]$. Каждому найденному шаблону pat_i ставится в соответствие эвристическое правило e_k и определяется значение соответствующего признака f_l.
3. Находятся и разрешаются все конфликтные ситуации, в которых эвристические правила принадлежат одной части текстового запроса.
4. Находятся и разрешаются все конфликтные ситуации, в которых одному признаку соответствуют разные значения и набор эвристических правил не является подмножеством ни одной строки номенклатурного справочника.
5. Выдается результат распознавания программы – лекарство $Drug_i$, определенное из выбранной строки каталога $Card$.
6. Пользователь подтверждает или отклоняет результат распознавания программы.
7. Обновление поддержки S и достоверности C для набора эвристик и всех его подмножеств, найденных в процессе распознавания.

3.5. Архитектура программы

Разработанная модель написана с использование объектно-ориентированного программирования и состоит из 6 основных классов, взаимодействующих между собой:

1. класс $Pattern$;
2. класс $Euristic$;
3. класс $Feature$;
4. класс $Entry$;
5. класс $Card$;
6. класс $Driver$.

3.5.1. Класс $Pattern$

Класс $Pattern$ описывает шаблоны для распознавания текстовых запросов. Каждый объект класса имеет текстовую часть шаблона, заданную в виде строки $m\_strPattern$, и указатель на объект класса $Euristic$ – m_pEur – определяет эвристическое правило, которому принадлежит данный шаблон.

В классе описывается метод $FindEntriesIn$, который находит и возвращает числовое значение шаблона,если оно есть, и интервалы покрытия текста данным шаблоном.

3.5.2. Класс $Euristic$

Класс $Euristic$ описывает эвристические правила. Каждый объект класса имеет название эвристического правила, заданного в виде строки, $m\_strName$, текстовое значение эвристического правила $m\_strEuristicValue$, указатели на объекты класса $Pattern$ $m\_vecPatterns$, которые принадлежат данному эвристическому правилу и имя признака $m\_strParentFeatureName$, которому принадлежит эвристическое правило.

В классе описываются метод $AddPattern$, который добавляет шаблон к данному эвристическому правилу, и метод $FindEntriesIn$, который описывается объектами класса $Entry$ и содержит информацию о найденнных эвристических правилах в текстовом запросе.

3.5.3. Класс $Feature$

Класс $Feature$ описывает фармацевтические признаки лекарств. Каждый объект имеет имя признака, заданного в виде строки $m\_strName$, набор эвристических правил $m\_vecEuristics$ класса $Euristic$,которые принадлежат данному признаку.

Класс содержит метод $AddEuristic$, с помощью которого добавляются эвристические правила к данному признаку.

3.5.4. Класс $Entry$

Класс $Entry$ – вспомогательный класс, который содержит информацию о найденнных эвристических правилах. Объект класса содержит имя эвристического правила для найденного шаблона $m\_strEuristicName$, начальную и конечную позиции найденного шаблона m_iFrom, m_iTo и числовое значение шаблона m_iValue, если оно есть.

Для этого класса специальным образом переопределен оператор сравнения, позволяющий проверять эвристические правила на равенство.

3.5.5. Класс $Card$

Класс $Card$ описывает номенклатурный справочник. Объект класса содержит список названий признаков лекарств $CardFeatureName$ и список значений лекарственных признаков для каждой строки справочника $CardValue$.

С помощью метода $ReadCard$ происходит считывание номенклатурного справочника из файла, а с помощью метода $ParseString$ выделение из строки отдельные признаков, и с помощью метода $ParseValues$ выделение числовых значений признаков.

3.5.6. Класс $Driver$

Класс $Driver$ – основной класс управления. Объект класса содержит эвристические правила и их названия $m\_mpEuristics$, названия признаков $m\_mpFeatures$ и объект класса $Card - m\_bd$. В классе объявлены названия входных и выходных файлов. В $m\_vecLastResponse$ с помощью класса $Entry$ описываются все найденные эвристически наборы в запросе. $m\_vecLastRequest$ – исходный текстовый запрос, записанный в виде строки, а $m\_vecChangedRequest$ – предобработанный текстовый запрос. $m\_vecAnswerTree$ – дерево ответов для каждого рассматриваемого набора эвристических правил. Наборы, создающие конфликтную ситуацию, в которых эвристические правила принадлежат одной части текстового запроса, хранятся в $m\_SegmentConflicts$. Наборы, создающие конфликтные ситуации, описанные в (3.1.2. и 3.1.3.), хранятся в $m\_FeaturesConflicts$. В $m\_mpSetEuristicInDB$ хранятся все найденные наборы эвристических правил со своими характеристиками.

С помощью метода $PreprocessString$ происходит предобработка текстового запроса, с помощью функций $ReadRightAnswer, ReadEuristics, ReadFeatures$ – чтение файла с эталонными ответами, файла с эвристическими правилами и файла с признаками соответственно. Методы $CalcTargetFuncForEntries$ и $CalcTargetFuncForPartialSet$ вычисляют среднее значение характеристик для полного и частичного набора эвристических правил, хранящихся в базе данных. С помощью методов $FindOutSegmentConflicts$ и $ResolveSegmentConflicts$ находятся и разрешаются конфликтные ситуации, в которых эвристические правила принадлежат одной части текстового запроса. Аналогично методы $FindOutFeaturesValueConflicts$ и $ResolveFeaturesValueConflicts$ находят и разрешают конфликтные ситуации, в которых одному признаку соответствуют разные значения и набор эвристических правил не является подмножеством ни одной строки номенклатурного справочника. Метод $FindLinesStrInDB$ находит все строчки из номенклатурного справочника, в которых содержатся найденные признаки. Метод $RenewSupportAndConfidence$ обновляет характеристики всех наборов эвристических правил, хранящихся в базе данных эвристик. В классе содержатся метод чтения всех входных данных $ReadData$ и методы для записи выходных данных $PrintWholeLastResponce, PrintStatistics, ReWriteEuristicInDB$.

3.6. Выводы по разделу

В данной главе описывались основные этапы разработки модели распознавания текстовых запросов.

В процессе распознавания значений признаков, в текстовом запросе могут возникать следующие конфликтные ситуации между эвристическими правилами:

1. два и более эвристических правила в одной и той же части текста выявили разные значения признаков;
2. два и более эвристических правила выявили разные значения одного и того же признака в тексте;
3. найденному набору эвристических правил не соответствует ни одно лекарство из номенклатурного справочника.

Представлены алгоритмы разрешения данных конфликтных ситуаций. Описаны процедуры пересчета характеристик выявленного эвристического набора в процессе распознавания, а также процедура выбора строки номенклатурного справочника в соответствии с выявленными признаками. Разработан алгоритм адаптивной модели распознавания и представлена архитектура программы.

4. АНАЛИЗ РЕЗУЛЬТАТОВ РАСПОЗНАВАНИЯ РАЗРАБОТАННОЙ МОДЕЛИ

Результатом распознавания текстового запроса является множество лекарств, выбранных программой «Приложение 3. Код программы» в соответствии с выявленными признаками и номенклатурным справочником. Пользователь проверяет наличие эталонного ответа в найденном множестве. Если эталонный ответ принадлежит найденному множеству, то считается, что программа нашла верный ответ и характеристики найденного эвристического набора увеличиваются, иначе результат распознавания считается неверным и характеристики набора уменьшаются.

Важно заметить, что пользователь не проверяет правильность выбранных признаков при распознавании текстового запроса. Проверять наборы выбранных признаков – трудно затратно для пользователя. Соответственно найденный правильный ответ распознавания не означает того, что все выявленные признаки были выбраны верно. Такое условие проверки результатов распознавания замедляет процесс обучения модели, но это компенсируется тем, что пользователь ограничен от анализа большого количества информации.

4.1. Качество распознавания текстовых запросов

Качество распознавания текстовых запросов будем оценивать по трём показателям: число правильно распознанных текстов, число неправильно распознанных текстов, число нераспознанных текстов. В нераспознанных текстовых запросах программа не предлагает пользователю множество ответов для выбора.

В результате обучения модели на 2134 текстовых запросах были получены следующие результаты: правильных ответов 1616, неправильных – 155, не распознанных – 363. В 76% случаев пользователь соглашается с выбранным ответом, в 7% случаев программа принимает неправильные решения и в 17% случаев программа не смогла распознать текстовые запросы.

4.2. Причины неправильного распознавания текстовых запросов

Разработанная модель распознавания текстовых запросов в 7% случаев неправильно распознала текстовый запрос и в 17% не смогла принять никакого решения.

Рассмотрим факторы, которые могли привести к неправильному распознаванию:

1. Пользователь соглашается с лекарством, выбранным программой, а не с набором выявленных признаков. Соответственно правильный ответ, полученный на предыдущих тестируемых выборках, не означает того, что выявлен-

ный набор эвристических правил был выбран верно, но характеристики надежности этого набора повысятся.

2. Модель распознавания адаптивна, то есть обучение модели происходит постоянно и выбор решения основывается на накопленных результатах распознавания. В сложной постановке задачи одинаковым текстовым запросам могут соответствовать разные лекарства, соответственно в некоторых случаях модель будет давать неверный ответ.
3. При процедуре пересчета характеристик эвристических наборов влияние каждого выбранного эвристического правила на результат считается одинаковым. Однако некоторые признаки могут быть более значимыми, чем другие при выборе лекарственного препарата.
4. Текстовые запросы могут быть слабо информативными – содержать недостаточный набор признаков. Соответственно список лекарств, подходящих под заданный текстовый запрос будет избыточным.
5. Программа не рассматривает все наборы из эвристических наборов, встречающихся в текстовом запросе. Рассматриваются только 10% наборов с усредненным лучшим показателем характеристик $S * C$.

4.3. Оценка производительности программы

Распознавание текстовых запросов происходит с учетом истории прошлых результатов распознавания. Соответственно время выполнения и количество совершенных вычислений разработанной модели зависит от размера обучающей выборки.

Для установления данных зависимостей программный продукт был протестирован «Таблица 1» на отличных по объему входных данных. Тестирование проводилось с использованием 64-х разрядной операционной системы, на процессоре AMD Phenom(tm) II N870 Triple − Core Processor 2.30 GHz и оперативной памятью 3,00ГБ.

Таблица 1 Результаты тестирования

Размер обучающей выборки	Количество обращений к БД эвристик	Количество рассматриваемых множеств	Время работы (с)
500	43682063	12166	37
1000	106688742	26218	85
1500	111994170	41853	100
2000	193507580	57628	165

Количество рассматриваемых множеств линейно «Рис. 4.1.» зависит от объема обучающей выборки. В среднем в одном текстовом запросе рассматривается 27 различных наборов эвристических правил.

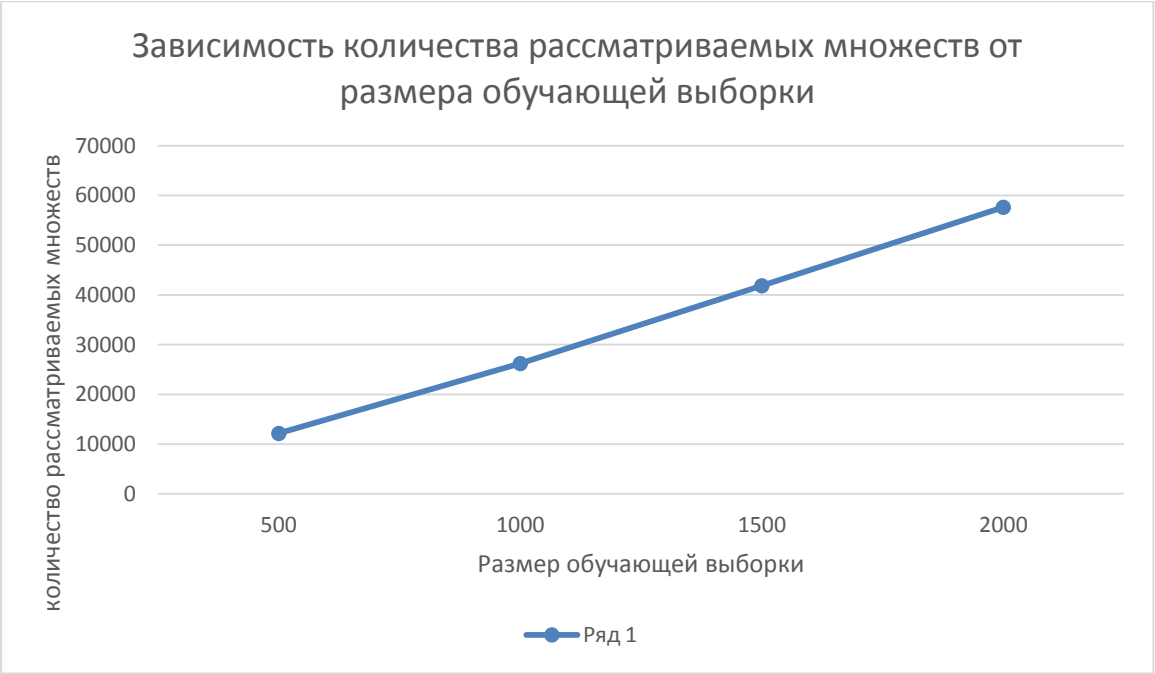

Рис. 4.1. График зависимости рассматриваемых множеств от объема обучающей выборки

В результате тестирования было выявлено, что число обращений к базе данных эвристик линейно «Рис. 4.2.» зависит от числа тестируемых примеров. Полученная зависимость может быть объяснена тем, что с количеством текстовых запросов, увеличивается количество наборов эвристических правил, записываемых в базу данных эвристик. Для определения набора с наилучшими показателями ведется поиск по всей базе данных, соответственно, чем больше база данных, тем больше операций будет совершаться. Данный показатель отрицательно влияет на производительность и в дальнейшем необходимо усовершенствовать процедуру поиска в базе данных эвристик.

Рис. 4.2. График зависимости для поиска в БД эвристик

Время выполнения модели распознавания текстовых запросов линейно «Рис. 4.3.» зависит от количества текстовых запросов. В среднем 1 текстовый запрос программа обрабатывает за 4 мс.

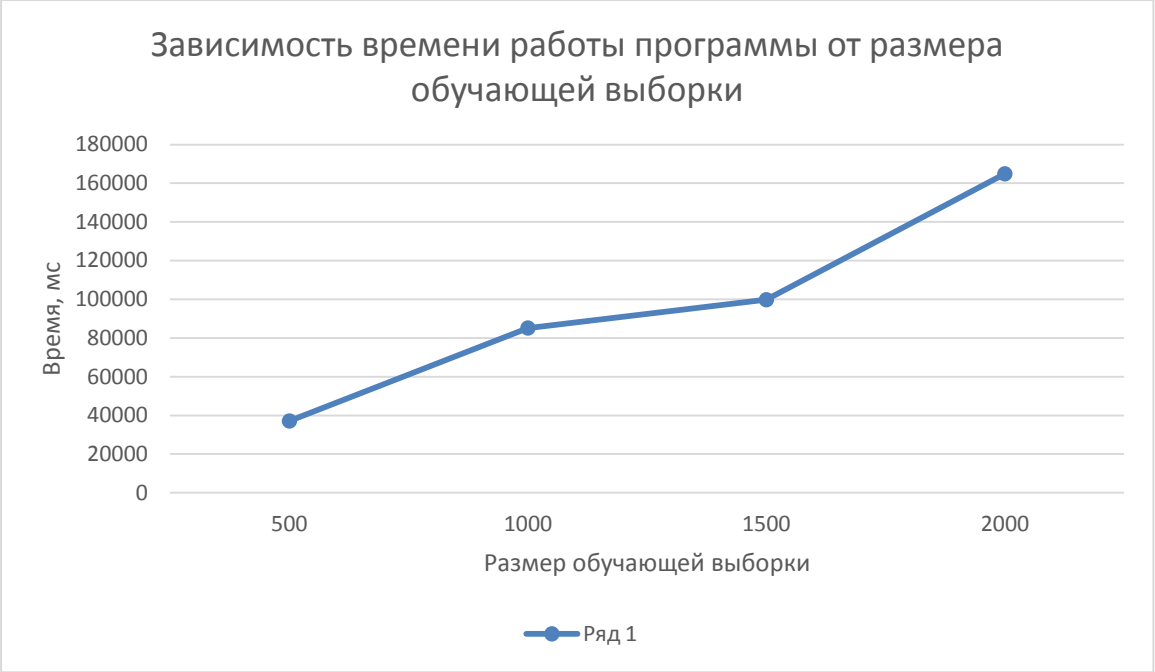

Рис. 4.3. График зависимости времени от объема обучающей выборки

4.4. Дальнейшие пути усовершенствования модели

Распознавание слабо структурированного документа – обширная задача. На данный момент задача решена с учетом допущений к формату данных. В дальнейшем можно усовершенствовать модель распознавания, учитывая логические операторы: И, ИЛИ, НЕ. С использованием логических операторов признаки лекарства в номенклатурном справочнике будут многозначными, то есть номенклатурный справочник будет точнее отражать характеристики лекарств.

В разработанной модели значения распознаваемых признаков либо числовые, либо текстовые и выбираются из конечного множества. В дальнейшем числовые данные можно выбирать из промежутка, задействовав такие шаблоны, как: не более, не менее, от …до и т.д.

Для более точной оценки качества распознавания необходимо изменять показатели характеристик в соответствии с долей влияния эвристического правила на результат. Для этого можно с помощью ассоциативных правил попробовать найти зависимости между эвристическими правилами и признаками.

Ключевым аспектом дальнейшего усовершенствования модели должна стать её производительность. Необходимо уменьшить число обращений в базу данных эвристик, вследствие этого уменьшится время работы программы.

4.5. Выводы по разделу

Разработанная модель в 76% случаев принимает правильные решения, в 7% случаев программа неверно распознает текстовые запросы, и в 17% случаев программа не смогла распознать текстовые запросы. Данные показатели средние для распознавания текстовых запросов. Прежде всего необходимо уменьшить число не распознанных текстов. Достичь этого можно за счёт учёта логических операторов, выбора числовых значений из диапозана и усовершенствования процедуры пересчёта характеристик эвристических наборов.

Тестирование модели показало, что число обращений к базе данных эвристик линейно зависит от количества распознанных текстовых запросов. Данный показатель отрицательно влияет на производительность и в дальнейшем необходимо усовершенствовать процедуру поиска в базе данных эвристик.

Рассмотрены факторы, влияющие на некачественное распознавание модели. Описаны методы по дальнейшему улучшению эффективности распознавания разработанной модели.

ЗАКЛЮЧЕНИЕ

В данной работе рассматривалась задача анализа слабоструктурированного документа с использованием номенклатурного медицинского справочника. Для распознавания текстовых запросов был разработан алгоритм, сочетающий фасетный метод с методом поиска по документу-образцу. Исходными данными являются документы-образцы: строки номенклатурного справочника и фасетные признаки: фармацевтические параметры лекарств. Необходимо разработать самообучающуюся систему распознавания текстовых запросов, которая по выделенным в тексте фармацевтическим параметрам будет выбирать результат из имеющейся коллекции выходных данных – лекарств.

Для выявления соответствий между фасетными признаками и медицинскими лекарствами из номенклатурного справочника решается задача поиска ассоциативных правил. Ассоциативное правило описывает закономерности вида: «Из набора найденных в тексте фармацевтических признаков X следует лекарство Y из номенклатурного справочника». Для корректировки системы распознавания каждому эвристическому набору ставится в соответствие две характеристики: достоверность C и поддержка S набора. Поддержка набора показывает, что $S\%$ наборов фармацевтических признаков, содержащихся в базе данных, содержат данный набор. Достоверность набора показывает, что $C\%$ наборов фармацевтических признаков из всего множества, содержащих набор фармацевтических признаков X были распознаны правильно.

В процессе распознавания значений признаков, к тексту применяются все шаблоны, которые соответствуют различным эвристическим правилам. Поэтому в текстовом запросе могут возникнуть конфликтные ситуации между эвристическими правилами:

1. два и более эвристических правила в одной и той же части текста выявили разные значения признаков;
2. два и более эвристических правила выявили разные значения одного и того же признака в тексте;
3. найденному набору эвристических правил не соответствует ни одно лекарство из номенклатурного справочника.

Для разрешения конфликтных ситуаций были разработаны 2 алгоритма. Данные алгоритмы разрешают конфликтные ситуации, руководствуясь соответствием с номенклатурным справочником и выбирая эвристические правила с лучшими показателями характеристиками. Данный подход позволяет учитывать предыдущий опыт распознавания текстовых запросов и адаптировать модель в соответствии с полученными результатами распознавания.

Качество распознавания текстовых запросов оценивается по трём показателям: число правильно распознанных текстов, число неправильно распознанных текстов, число нераспознанных текстов. Текст считается распознанным верно, если эталонный ответ пользователя принадлежит множеству лекарств, выбранных программой в соответствии с выявленными признаками и номенклатурным справочником. Пользователь не проверяет правильность выбранных признаков при распознавании текстового запроса, соответственно найденный правильный ответ распознавания не означает того, что все выявленные признаки были выбраны верно.

В результате обучения модели в 76% случаев пользователь соглашается с выбранным ответом, в 7% случаев программа принимает неправильные решения и в 17% случаев программа не может распознать текстовые запросы. Тестирование модели показало, что число обращений к базе данных эвристик линейно зависит от количества распознанных текстовых запросов. Данный показатель отрицательно влияет на производительность и в дальнейшем необходимо усовершенствовать процедуру поиска в базе данных эвристик.

В дальнейшем для улучшения эффективности распознавания разработанной модели необходимо усовершенствовать алгоритм поиска набора эвристических правила в базе данных. Так же для более точной оценки качества распознавания можно изменять показатели характеристик в соответствии с долей влияния эвристического правила на результат.

ЛИТЕРАТУРА

1. Алеев, А.В. Поиск и использование шаблонов последовательных событий при анализе логов / А.В. Алеев. – СПб., 2012. – 59 с.
2. Батура, Т.В. Методы автоматической классификации текстов / Т.В. Батура // Программные продукты и системы. – 2017. – том 30. – №1. – с. 85 – 99.
3. Добров, Б.В. Автоматическая рубрикация полнотекстовых документов по классификаторам сложной структуры / Б.В. Добров, Н.В. Лукашевич // VIII Нац. конф. по искусственному интеллекту КИИ-2002. – М.: Физматлит, 2002. – том 1. – с. 178-186. – Режим доступа: http://www.cir.ru/docs/ips/publications/2002_cai_rubr.pdf/ – (Дата обращения: 25.12.2016).
4. Добрынин, В.Ю. Задача выбора тематических коллекций, релевантных запросу / В.Ю. Добрынин, И.С. Некрестьянов // Труды Всероссийской научно-методической конференции «Интернет и современное сообщество». – СПб, 1998.
5. Дубинский, А. Некоторые вопросы применения векторной модели представления документов в информационном поиске / А. Дубинский // Управляющие системы и машины. – 2001. – №4. – с. 77-83.
6. Дубинский, А.Г. Разработка моделей и совершенствование структуры систем информационного поиска в глобальной компьютерной сети / А.Г. Дубинский // Днепропетровский национальный университет. – Днепропетровск, 2002.
7. Ермаков, А.Е. Ассоциативная модель порождения текста в задаче классификации / А.Е. Ермаков, В.В. Плешко // Информационные технологии. – 2000. – № 12.
8. Ермаков, А.Е. Полнотекстовый поиск: проблемы и их решение / А.Е. Ермаков // Мир ПК. – 2000. – №5.
9. Журавлёв, Ю.И. Об алгебраическом подходе к решению задач распознавания или классификации / Ю.И. Журавлёв // Проблемы кибернетики. – 1978. – т. 33. – с. 5-68.
10. Зайко, Т.А. Ассоциативные правила интеллектуальном анализе данных / Т.А. Зайко, А.А. Олейник, С.А. Субботин// Вісник НТУ "ХПІ". Серія: Інформатика та моделювання. – Харьков: НТУ "ХПИ", 2013. – № 39 (1012). – с. 82– 96.
11. Когаловский, М. Перспективные технологии информационных систем / М. Когаловский. – М.: ДМК Пресс; М.: Компания АйТи. – 2003. – 288 с.
12. Кураленок, И.Е. Оценка систем текстового поиска / И.Е. Кураленок, И.С. Некрестьянов Оценка систем текстового поиска // Программирование. – 2002. – №4. – с. 226-242.

13. Моченов, С.В. Векторная модель представления текстовой информации / С.В. Моченов, А.М. Бледнов, Ю. А. Луговских // Материалы международной научной конференции. – Ижевск, 2006.
14. Некрестьянов, И.С. Оценка тематического подобия текстовых документов / И.С. Некрестьянов, В.Ю. Добрынин, В.В. Клюев // Труды второй всероссийской научной конференции “Электронные библиотеки”. – Протвино, 2000. – с. 204-210.
15. Поиск на основе фасетов. – Дата обновления: декабрь,2009. URL: https://www.ibm.com/support/knowledgecenter/ru/SS5RWK_2.1.0/com.ibm.discovery.es.ad.doc/iiysafacets.htm (дата обращения: 20.11.2016).
16. Поляков, И.В. Проблема классификации текстов и дифференцирующие признаки / И.В. Поляков, Т.В. Соколова, А.А. Чеповский [и др.] // Вестник НГУ. Серия: Информационные технологии. – 2015. – Т. 13. – №2. – с. 55–63.
17. Сидорова, Е.А. Тематический анализ запросов пользователей на основе предметно-ориентированного словаря / Е.А. Сидорова // Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2014. – том 12. – №4. – с. 83 – 94.
18. Фасетный метод классификации товаров в интернет-магазине. – Режим доступа: http://elbuz.com/faceted-classification-in-online-store – (Дата обращения: 20.11.2016).
19. Чубукова, И.А. Data Mining / И.А. Чубукова. –2008. – 324 с. – Режим доступа: http://portal.tpu.ru/SHARED/a/AAPONOMAREV/metod/Tab/lections%20data%20mining.pdf. – (Дата обращения: 25.10.2015).
20. Чугреев, В.Л. Анализ структуры текста и прогнозирование нечисловых величин / В.Л. Чугреев, С.А. Яковлев // ВУЗОВСКАЯ НАУКА – РЕГИОНУ: Материалы 1-й Общероссийской научно-технической конференции. – Вологда: ВоГТУ, 2003. – с. 202-204.
21. Чугреев, В.Л. Анализ текста, применительно к решению задач поиска документов по образцу / В.Л. Чугреев, С.А. Яковлев // Информатизация процессов формирования открытых систем на основе САПР, АСНИ, СУБД и систем искусственного интеллекта (ИНФОС - 2003): Материалы 2-й Межд. науч.-техн. конф. – Вологда: ВоГТУ, 2003. – с. 49-52.
22. Чугреев, В.Л. Выделение критериев поиска текста на основе подобия значимых документов / В.Л. Чугреев, С.А. Яковлев // ВУЗОВСКАЯ НАУКА – РЕГИОНУ: Материалы 1-й Общероссийской нучн.-техн. конф. – Вологда: ВоГТУ, 2003. – с. 200-202.
23. Aggarwal, C. Data classification: algorithms and applications / C. Aggarwal. – CRC Press, 2014. – № 9. – p. 245–273.
24. Agichtein, E. Learning search engine specific query transformations for question answering / E. Agichtein, S. Lawrence, L. Gravano // In Proc. of the WWW10. – 2001. – p. 169-178.
25. Agrawal, R. Fast Discovery of Association Rules / R. Agrawal, R. Srikant // In Proc. of the 20th International Conference on VLDB. –Chile, September 1994.

26. Agrawal, R. Mining Associations between Sets of Items in Massive Databases / R. Agrawal, T. Imielinski, A. Swami // In Proc. of the 1993 ACM-SIGMOD Int'l Conf. on Management of Data. – 1993.
27. Brinetal, S. Dynamic Itemset Counting and Implication Rules for Market Basket Data / S. Brinetal // In Proc. ACM SIGMOD Int'l Conf. Management of Data. – ACM Press. – New York, 1997.
28. Dublin Core Metadata Element Set Reference Description // Version 1.1. – 1999. – Режим доступа: http:/purl.org/proposed_recommendations/pr-dces-19990702.html – (Дата обращения: 25.10.2016).
29. Finkelstein, L. Placing search in context: the concept revisited / L. Finkelstein, E. Gabrilovich, Y. Matias [и др.] // In Proc. of the WWW10. – 2001. – p. 406-414.
30. Fuller, C.M. An investigation of data and text mining methods for real world deception detection / C.M. Fuller, D.P. Biros, D. Delen // Expert Syst. with Applications. – 2011. – №38. – p. 8392–8398.
31. Ghiassi, M. Automated text classification using a dynamic artificial neural network model / M. Ghiassi, M. Olschimke, B. Moon [и др.] // Expert Syst. with Applications. – 2012. – №39. – p. 10967–10976.
32. Hearst, M. Clustering versus faceted categories for information exploration / M. Hearst // Communications of the ACM. – 2006. – p. 56- 61.
33. Hipp, J. Algorithms for Association Rule Mining – A General Survey and Comparison / J. Hipp, U. Guntzer, G. Nakaeizadeh // In Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2000.
34. Ju, R. An Efficient Method for Document Categorization Based on Word and Latent Semantic Analysis / R. Ju // 2015 IEEE Intern. Conf. on Comp. and Inform. Technology. – UK, 2015. – p. 2276–2283.
35. Maron, M. On relevance, probabilistic indexing and information retrieval / M. Maron, J. Kuhns // Jornal of the ACM. – 1960. – №. 7. – 216-244.
36. Medhat, W. Sentiment analysis algorithms and applications: a survey / W. Medhat, A. Hassan, H. – Ain Shams Eng. Jour. – 2014.– №5. – p. 1093–1113.
37. Parkhi, R. Decimal Classification and Colon Classification in Perspective / R. Parkhi. – London: Asia Publishing House. – 1964. – p. 130.
38. Pontiki, M. Aspect based sentiment analysis / M. Pontiki, D. Galanis, J. Pavlopoulos [и др.] // The 8th Intern. Workshop on Semantic Evaluation (SemEval 2014). –Ireland, 2014. – p. 27–35.
39. Rudakov, K.V. Methods of Optimization and Monotone Correction in the Algebraic Approach to the Recognition Problem / K.V. Rudakov, K.V. Vorontsov // Doklady Mathematics. – 1999. – v. 60.
40. Salton, G. Extended Boolean information retrieval / G. Salton, E. Fox, H. Wu // Communications of the ACM. – December 2001. – Vol. 26. – No. 4. – p. 35-43.
41. Santner, T.J. The Statistical Analysis of Discrete Data / T.J. Santner, D.E. Duffy // Springer-Verlag. – 1989.
42. Savasere, A. An Efficient Algorithm for Mining Association Rules in Large Databases / A.Savasere, E. Omiecinski, S. Navathe // In Proc. 21st Int'l Conf. Very Large Data Bases. – San Francisco, 1995.

43. Sebastiani, F. Machine learning in automated text categorization / F. Sebastiani // ACM Computing Surveys. – 2002. – Vol.34. – №1. – P.1-47. – Режим доступа: http://nmis.isti.cnr/sebastiani/Publications/ACMCS02.pdf – (Дата обращения: 25.10.2015).
44. Singhal, A. Modern Information Retrieval: A Brief Overview / A. Singhal // Data Enginering Bulletin – IEEE Computer Society, December 2001. – Vol. 24. – № 4. – p. 35-43.
45. Tsochantaridis, I. Large margin methods for structured and interdependent output variables / I. Tsochantaridis, T. Joachims, T. Hofmann [и др] // Journal of Machine Learning Research. – 2005. – v. 6. – p. 1453-1484.
46. Tunkelang, T. Faceted Search / T. Tunkelang // Synthesis Lectures on Information Concepts, Retrieval, and Services. – 2009.
47. Vo1, B. Fast Algorithm for Mining Generalized Association Rules / B. Vo1, B. Le // International Journal of Database Theory and Application. – 2009. – Vol.2. – №3. – p. 1 – 12.
48. Xiang, Z. Character-level convolutional networks for text classification / Z. Xiang, Z. Junbo, Y. // Proc. Neural Inform. Processing Systems Conf. (NIPS 2015). – Canada, 2015. – Режим доступа: https://arxiv.org/abs/1509.01626 – (Дата обращения: 25.06.2016).
49. Yang, Y. An evaluation of statistical approaches to text categorization. / Y. Yang // Information Retrieval Journal. – 1999. – №. 1. – p. 69–90.
50. Zhang, J. Relational Browser++: a fast and contextualized searching and browsing tool / J. Zhang, G. Marchionini // Technical Report, SILS-TR-2004-01. – 2004.

ПРИЛОЖЕНИЕ 1. ФОРМАТ ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

Титульник

Рассмотрим, как описываются входные и выходные данные в разработанной модели распознавания текстовых запросов.

П1.1. Файл EuristicsAndFeatures.txt

В данном файле записаны входные данные: набор признаков; набор эвристических правил; набор шаблонов.

В начале файла подается количество существующих эвристических правил. Далее с новой строки записываются имя эвристического правила и её текстовое значение. С новой строки записывается соответствующий шаблон эвристического правила.

Шаблон состоит из текстовой и числовой части. Если шаблон содержат в себе символ #, то на данном месте должно стоять число, иначе числовая часть по умолчанию равна -1.

Следующее число в файле – количество существующих признаков. Далее с новой строки записывается имя признака и количество эвристических правил, принадлежащих данному признаку. С новых строк записываются названия эвристических правил, принадлежащих данному признаку.

П1.2. Файл test.csv

В данном файле записаны входные данные: каталог $Card$ используемых лекарств с соответствующими значениями признаков.

Первая строка таблицы содержит в себе название всех признаков. Каждая следующая строка содержит в себе значения не обязательно всех признаков.

Каждая строка таблицы представлена в формате вида «Рис. П1.1.».

Текстовое значение 1 признака	:	Числовое значение 1 признака	,	Текстовое значение 2 признака	:	Числовое значение 2 признака	,	…	Текстовое значение n-го признака	:	Числовое значение n-го признака

Рис. П1.1. Формат строк в номенклатурном справочнике

Текстовые значения таблицы могут принимать следующие значения:

1. текст;
2. «*» - данный признак принимает любое текстовое значение или может быть пустым;
3. «!» - данный признак не должен содержать никакого текстового значения.

Числовые значения таблицы могут принимать следующие значения:

1. число;
2. «-2» - данный признак принимает любое числовое значение;
3. «-1» - данный признак не должен содержать никакого числового значения.

Все десятичные числа записываются через «.».

П1.3. Файл seteuristicinbd.txt

В данном файле записаны выходные данные: база данных эвристических правил.

Каждый набор эвристических правил записывается в файл в виде тройки строк. Первая строка представляет собой набор эвристических правил, найденных в процессе распознавания, на следующей строке содержатся соответствующие характеристики, разделенные между собой пробелами. В последней строке содержится статистика по данному набору: число правильно распознанных лекарств и число неправильно распознанных лекарств.

Строки представлены в формате вида «Рис. П1.2.».

Набор эвристических правил, расположенных в алфавитном порядке, разделенных между собой пробелом.
Поддержка набора(S) Достоверность набора(C)
Число правильных распознанных лекарств Число неправильно распознанных лекарств

Рис. П1.2. Формат строк в базе данных эвристических правил

ПРИЛОЖЕНИЕ 2. ПРИМЕР РАБОТЫ АДАПТИВНОЙ МОДЕЛИ РАСПОЗНАВАНИЯ ТЕКСТОВОГО ЗАПРОСА

Титульник

Дан номенклатурный справочник с 12-ю параметрами лекарств. Фрагмент справочника представлен в «Таблица П2.1.». Поля содержащие «!» не могут иметь никакие значения данного признака. Поля, содержащие «*» могут иметь любое значения данного признака. Результат распознавания – уникальный номер каждой строки справочника.

Таблица П2.1. фрагмент номенклатурного справочника

МНН	**ТН группировочное**	**ЛФ1**	**ЛФ1.5**	**ЛФ2**	**ЛФ3**	**дозировка**	**Объем**	**№**	**Тара**	**Материал тары**	**Детализация материала тары**	**Результат распознавания**
Азелаиновая кислота	Азелик	гель	!	!	для наружного применения	дозировка мг/г:150	объем г:30	№:1	туба	!	Алюминий	4601969006322
Азелаиновая кислота	Скинорен	гель	!	!	для наружного применения	дозировка мг/г:150	объем г:15	№:1	туба	!	!	4260085521210
Азелаиновая кислота	Скинорен	гель	!	!	для наружного применения	дозировка мг/г:150	объем г:30	№:1	туба	!	Алюминий	4260085520305
Азелаиновая кислота	Скинорен	гель	!	!	для наружного применения	дозировка мг/г:150	объем г:5	№:1	туба	!	!	4260085520312
Азелаиновая кислота	Скинорен	крем	!	!	для наружного применения	дозировка мг/г:200	объем г:30	№:1	туба	!	!	4029668001125
Азитромицин	Азитрокс	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:20	объем г:15.9	№:1	флакон	стекло	темное	4601669006875
Азитромицин	Сумамед	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:40	объем г:16.74	№:1	флакон	полимер	полиэтилен	3850114216432
Азитромицин	Сумамед	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:40	объем г:29.295	№:1	флакон	полимер	полиэтилен	3850114216449
Азитромицин	Сумамед	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:40	объем г:35.573	№:1	флакон	полимер	полиэтилен	3850114216456

Окончание таблицы П2.1.

МНН	ТН группировочное	ЛФ1	ЛФ1.5	ЛФ2	ЛФ3	дозировка	Объем	№	Тара	Материал тары	Детализация материала тары	Результат распознавания
Азитромицин	Хемомицин	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:20	объем г:11.43	№:1	флакон	стекло	темное	8600097306072
Азитромицин	Хемомицин	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:40	объем г:10	№:1	флакон	стекло	темное	8600097303262
Азитромицин	Экомед	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:20	объем г:16.5	№:1	флакон	*	*	4607003668983
Азитромицин	Экомед	порошок	порошок для приготовления суспензии	!	для приема внутрь	дозировка мг/мл:40	объем г:16.5	№:1	флакон	*	*	4607003668990

Каждому признаку лекарства соответствуют эвристические правила. Эвристические правила находятся с помощью текстовых и числовых шаблонов. Примеры шаблонов и эвристических правил для признака «дозировка» представлены в «Таблица П2.2.».

Таблица П2.2. Эвристические правила для признака "дозировка"

Названия эвристических правил	Значение эвристического правила	шаблон
дозировка_1	дозировка мг/г	# мг/г
дозировка_2	дозировка АТрЕ	# АТрЕ
дозировка_3	дозировка ЕД	# ЕД
дозировка_4	дозировка ЕД	#ЕД
дозировка_5	дозировка мг/г	#мг/г
дозировка_6	дозировка мг/мл	# мг/мл
дозировка_7	дозировка мг/мл	#мг/мл
дозировка_8	дозировка МЕ	#МЕ
дозировка_9	дозировка мкг/доза	#мкг/доза
дозировка_10	дозировка мл/доза	#мл/доза
дозировка_11	дозировка млн КОЕ	# млн КОЕ

Текстовые запросы неструктурированны и представлены в виде строк «Таблица П2.3.».

Таблица П2.3. Примеры текстовых запросов

Текстовый запрос	**Номер строки, соответствующей данному запросу**
Азитромицин порошок для приготовления суспензии для приема внутрь 100,0 мг / 5 мл 11,43г- флаконы №1 уп	8600097306072
Азитромицин азитрок спорд/ при г сусп внутрь 100мг/5мл № 1 фл фл	4601669006875
Азитромицин показания: инфекционно-воспалительные заболевания, вызванные чувствительными к препарату микроорганизмами : инфекции верхних дыхательных путей и лор-органов (фарингит/тонзиллит,синусит,средний отит); инфекции нижних дыхательных путей (острый бронхит, обострение хронич)	3850114216432
Азитромицин хемомицин порошок для приготовления суспензии для приема внутрь 100мг/5мл, 11.43г-флаконы темного стекла/в комплекте с ложкой мерной/-пачки картонные	8600097306072
Азитромицин азитромицин,сумамед 200мг/5мл,порошок для приготовления суспензии,флакон полимерный для приготовления суспензии 29,295г.уп	3850114216449
мнн: азитромицин 24.42.11. 239 порошок для приготовления суспензии для приема внутрь с концентрацией 200мг/5мл16,74г,флак.№1уп	3850114216432
Азитромицин порошок для приготовления суспензии для приема внутр ь100мг/5мл20мл№1уп.	4601669006875
рчш № юьшшэбьрьхфдю № хяю № ююъфыя № шуюютыхэшееяхэчшшфыя № шхьртэ № 200ьу/5ьы;16,74уы	3850114216432
Азитромицин порошок д/сусп. Для приема внутрь 200мг/5мл 29,295г (30мл) фл.флакон	3850114216449
рчш№юьшшэяю№ююъфыя№шуюютыхэшееяхэчшшфыя№шхьртэ№100ьу/5ьы,11.43у-ыръюэхьэюуюехъыр/тъюьяыхъхеыюцъющьх№эющ/-яръшър№юээхя	8600097306072
Азитромицин эко мед порошок для приготовления суспензии для приема внутрь 100мг/5мл,-16,5г уп	4607003668983
рчш№юъеяю№ююъфыя№шуюютыхэшееяхэчшщфыя№шхьртэ№100ьу/5ьы15,9у20ьыя	4601669006875

В результате работы программы для каждого текстового запроса получим набор эвристических правил «Рис. П2.1.», который был найден в процессе распознавания. По данному набору ищется соответствующие строчки в справочнике, и если таких строк меньше 10, генерируется ответ, который сравнивается с эталонным. Если таких строк больше 10, то считается, что ответ неоднозначный и пользователю лучше самому проверить данный текстовый запрос.

Например, в первом текстовом запросе программа выбрала следующий набор признаков: МНН: Азитромицин; ЛФ3: для приема внутрь; Объем: объем г 11,43; №: 1. Результат выбранной программой совпал с эталонным ответом.

В 8,10,12-м текстовых запросах программа не смогла выявить признаки и выбрать строку номенклатурного справочника. Данные тексты сильно зашумленные.

Всего программа распознала 12 текстовых запросов, из которых 9 распознались верно и 3 запроса программа не смогла распознать.

```
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 43 To 57 with value -1 Euristic name: ЛФ3_7
3 From 65 To 66 with value 11.43 Euristic name: объем_2
4 From 76 To 78 with value 1 Euristic name: номер_4
Пользователь подтвердил выбранное  программой лекарство.
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 11 To 18 with value -1 Euristic name: ТН_3
3 From 45 To 46 with value 1 Euristic name: номер_1
4 From 47 To 48 with value -1 Euristic name: Тара_13
Пользователь подтвердил выбранное  программой лекарство.
1 From 0 To 10 with value -1 Euristic name: МНН_2
Пользователь подтвердил выбранное  программой лекарство.
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 11 To 19 with value -1 Euristic name: ТН_99
3 From 52 To 66 with value -1 Euristic name: ЛФ3_7
4 From 75 To 76 with value 11.43 Euristic name: объем_2
5 From 78 To 83 with value -1 Euristic name: Тара_5
6 From 92 To 96 with value -1 Euristic name: материал_4
7 From 85 To 91 with value -1 Euristic name: детали_11
Пользователь подтвердил выбранное  программой лекарство.
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 23 To 29 with value -1 Euristic name: ТН_92
3 From 112 To 113 with value 29.295 Euristic name: объем_2
4 From 71 To 76 with value -1 Euristic name: Тара_5
5 From 77 To 83 with value -1 Euristic name: материал_1
Пользователь подтвердил выбранное  программой лекарство.
1 From 4 To 14 with value -1 Euristic name: МНН_2
2 From 50 To 64 with value -1 Euristic name: ЛФ3_7
3 From 86 To 87 with value 16.74 Euristic name: объем_2
4 From 95 To 97 with value 1 Euristic name: номер_4
Пользователь подтвердил выбранное  программой лекарство.
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 43 To 57 with value -1 Euristic name: ЛФ3_7
3 From 69 To 71 with value 1 Euristic name: номер_4
Пользователь подтвердил выбранное  программой лекарство.
8
Лекарство не найдено.
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 25 To 39 with value -1 Euristic name: ЛФ3_7
3 From 47 To 48 with value 29.295 Euristic name: объем_2
4 From 54 To 55 with value -1 Euristic name: Тара_13
Пользователь подтвердил выбранное  программой лекарство.
10
Лекарство не найдено.
1 From 0 To 10 with value -1 Euristic name: МНН_2
2 From 11 To 16 with value -1 Euristic name: ТН_104
3 From 49 To 63 with value -1 Euristic name: ЛФ3_7
4 From 73 To 74 with value 16.5 Euristic name: объем_2
Пользователь подтвердил выбранное  программой лекарство.
12
Лекарство не найдено.
```

Рис. П2.1. Найденные наборы эвристических правил

После распознавания каждого текстового запроса изменяются характеристики эвристических наборов «Рис. П2.2.», хранящихся в базе данных эвристик.

```
ЛФ3_1 МНН_10 Тара_13 дозировка_2 номер_4
0.2 0.2
1 0
ЛФ3_7 МНН_2 ТН_104 объем_2
0.2 0.2
1 0
ЛФ3_7 МНН_2 ТН_99 Тара_5 детали_11 материал_4 объем_2
0.2 0.2
1 0
ЛФ3_7 МНН_2 Тара_13 объем_2
0.2 0.2
1 0
ЛФ3_7 МНН_2 номер_4
0.2 0.2
1 0
ЛФ3_7 МНН_2 номер_4 объем_2
1 0.22
2 0
МНН_2
1 0.58
7 0
МНН_2 ТН_3 Тара_13 номер_1
0.2 0.2
1 0
МНН_2 ТН_92 Тара_5 материал_1 объем_2
0.2 0.2
1 0
```

Рис. П2.2. Наборы эвристических правил с их характеристиками

ПРИЛОЖЕНИЕ 3. КОД ПРОГРАММЫ

Титульник

Файл classes.h - содержит в себе объявления классов;
файл classes.cpp - содержит методы классов;
файл main.h – заголовочный файл программы;
файл main.cpp – содержит функцию main.

П3.1. Файл classes.h

```
#pragma once
#include "main.h"

using namespace std;

#define VALUE_DOES_NOT_EXISTS -1
#define VALUE_COULD_BE_ANY -2

class Euristic;
class Pattern;
class Driver;
class Entry;
class Card;



class Pattern
{
private:


public:

    typedef vector<pair<pair<int, int>, double> > patternResponse;

    string m_strPattern;
    Euristic * m_pEur;

    Pattern() {}

    Pattern(const string& pat, Euristic *pEur) : m_strPattern(pat), m_pEur(pEur)
    {

    }

    patternResponse FindEntriesIn(const string& strRequest,  unordered_map<int, string>& mpReplacedNumbers);        //find all entries, like < <from, to>, value (default -1)>;

    bool operator<(const Pattern& rh)
    {
        return (this->m_strPattern < rh.m_strPattern);
    }
```

```
    friend bool operator==(const Pattern&, const Pattern&);

};

inline bool operator==(const Pattern &x, const Pattern &y)
{
    return x.m_strPattern == y.m_strPattern;
}

class Euristic
{
private:



public:
    string m_strName;
    string m_strEuristicValue;     // string constant value for current euristic
    vector<Pattern> m_vecPatterns;
    string m_strParentFeatureName;

    Euristic() {}

    Euristic(const string& eurName, const string& strEurValue) : m_strName(eurName),
m_strEuristicValue(strEurValue)
    {

    };

    void SetParentFeatureName(const string& str)
    {
        m_strParentFeatureName = str;
    }

    void AddPattern(const string & patName)
    {
        m_vecPatterns.push_back(Pattern(patName, this));
        sort(m_vecPatterns.begin(), m_vecPatterns.end());
    }

    vector<Entry> FindEntriesIn(const string& strRequest,  unordered_map<int, string>&
mpReplacedNumbers);

    void ResolveInternalConflicts(Pattern::patternResponse & vecPattResponse);

};

class Feature
{
private:
```

```
        string m_strName;
        vector<Euristic *> m_vecEuristics;

    public:
        Feature() {}
        Feature(string FeatureName) : m_strName(FeatureName)
        {

        }

        void AddEuristic(Euristic * eur)
        {
                m_vecEuristics.push_back(eur);
        }
    };

    /*
    Entry - class describes an entry of some pattern and all needed information about this entry
    */
    class Entry
    {
    private:


    public:

        string m_strEuristicName;     //name of euristic for founded pattern
        int m_iFrom, m_iTo; //borders of pattern in request - used for checking for intersections that create conflicts
        double m_iValue; // value for numeric patterns, -1 if not a numeric pattern

        bool operator==(Entry& right) const
        {
                return m_strEuristicName == right.m_strEuristicName && m_iFrom == right.m_iFrom && m_iTo == right.m_iTo;
        }

        Entry() {}
        Entry(string strEurName, int iFrom, int iTo, double iValue = VALUE_DOES_NOT_EXISTS) : m_strEuristicName(strEurName), m_iFrom(iFrom), m_iTo(iTo), m_iValue(iValue)
        {

        }

        void PrintEntry(fstream& output)
        {
                output << "From " << m_iFrom << " To " << m_iTo << " with value " << m_iValue << " Euristic name: " << m_strEuristicName << endl;
        }

    };
```

```
class Card
{


private:
    pair<string, double> ParseValues(const string & str);

public:
    vector<vector<pair<string, double> > > CardValue;//feature text values in each line

    vector <pair<string, double> > CardFeatureName;//NAME of features
    Card() {};
    vector<pair<string, double> > ParseString(const string & str, int pos);
    void ReadCard();

};

struct NumericalValuesInDB
{
    double Confidence;
    double Support;
    int RightAns;
    int WrongAns;
    int NumbOfParentsSet;//number of sets, which have such set
};

class Driver
{
private:

    const double DEFAULT_SUPPORT = 0.2;
    const double DEFAULT_CONFIDENCE = 0.2;
    int NumberOfGivenExamples;//number of all examples
    int brunch=0;//number of current brunch
    int OutRightAnsw = 0;//numbers for statistic in out file
    int OutWrongAnsw=0;
    int OutEmptyAnsw = 0;
    int OutChooseUser = 0;
    //int secondbrunch = 0;//number of cur variant for ans

    map<string, Euristic> m_mpEuristics; //name, euristic
    map<string, Feature> m_mpFeatures; //name, feature

    string m_strInputName;//for patterns,euristic,feature
    string m_strUserInputName;//for user request
    string m_strAnswerName;// for answer on user request
    string m_strOutName;//for out results
    string m_strOutStatisticName;//for out statistic

    fstream m_inputFile;
```

```
        fstream m_inputUserFile;
        fstream m_inputAnswerFile;
        fstream m_out;
        fstream m_outstatistic;

        bool m_boolIsResultRight;
        bool m_boolIsResultExist;

        Card m_bd;

        vector <vector<Entry>> m_vecLastResponse;
        vector<vector <vector<Entry>>> m_vecAnswerTree;//answers of every set from every brunch
        vector <string> m_vecLastRequest;
        vector <string> m_vecChangedRequest;      //request with replaced numbers and without spac-
es
        unordered_map<int, string> m_mpReplacedNumbers;

        vector<vector<int> > m_SegmentConflicts; //graph of conflicts of first type (segment intersec-
tions) (numeration according to m_vecLastResponce)

        unordered_map<string, vector<Entry > >m_FeaturesConflicts; // all possible Entries for partic-
ular feature
        //map<vector<string>, pair<double, double> > m_mpSetEuristicInBd;//all finded sets of
euristic with their credibility and possibility (достоверность и вероятность) after solving of all con-
flicts
        //map<vector<string>, pair<int, int> > m_mpStatisticsOfResultsInBd;//all finded sets of
euristic with their statistics: right answers and wrong answers after solving of all conflicts

        map<vector<string>, NumericalValuesInDB> m_mpSetEuristicInDB; // all finded sets of
euristic with their credibility and possibility(достоверность и вероятность) and statistics: right an-
swers and wrong answers after solving of all conflicts

        string PreprocessString(string str);
        string ReplaceNumbers(string str);

        vector<string> RightAnswer;//right answer of request

        void ReadRightAnswer();

        void ReadEuristics();

        void ReadFeatures();

        void ClearAllForNewRequest();

        double CalcTargetFuncForEntries(vector<int> set);//calculate parameters(S,C) for set of en-
tries, that contains in DB

        double CalcTargetFuncForPartialSet(vector<int> set, int featuresCnt);

        void FindOutSegmentConflicts();
```

```
	void ResolveSegmentConflicts();

	void FindOutFeaturesValueConflicts();

	void ResolveFeaturesValueConflicts();
	vector<vector<Entry>> ChooseValuesForMainFeatures(int featuresCnt);
	void GenerateAllValues(vector<vector<Entry> > & answer, vector<Entry> curVec, int ptr, int totalCnt);
	vector<int> GetEntriesIndexes(vector<Entry> cur);

	vector <int> FindLinesStrInDB(vector<int> set, int featuresCnt = 1e9);//return vector of numbers of lines in which first featuresCnt are founded in Card
	bool IsConsistInDB(vector<int> set, int featuresCnt = 1e9);
	void AddCardFeatureValuesInSet(vector <int> set);
	void DeleteCardFeatureValuesInSet();
	void ReadEuristicsInDB();
	double FindSetEuristicInDB(vector <string> set);
	double FindPartialSetEuristicInBD(vector <string> set);
	void RenewSupportAndConfidence();
	void FindNumbOfParentsSet();//find all parents in DB, which have such set
	void ChangeParametersInSubset(vector <string> set);//change all parents in DB, which have such set

	bool IsSubset(vector<string> a, vector<string> b);//check if b is subset of a

	vector <string> OrderEuristic(vector <int> set);

	void PrintLastEntries(fstream& out)
	{
		for (int i = 0; i < m_vecLastResponse[brunch].size(); ++i)
		{
			out << i + 1 << ' ';
			m_vecLastResponse[brunch][i].PrintEntry(out);
		}
	}
	void PrintSegmentConflicts(fstream& out)
	{
		out << "SEGMENT CONFLICTS:\n";
		for (int i = 0; i < m_SegmentConflicts.size(); ++i)
		{
			out << i + 1 << ' ' << m_vecLastResponse[brunch][i].m_strEuristicName << " ["<< m_vecLastResponse[brunch][i].m_iFrom<<','<< m_vecLastResponse[brunch][i].m_iTo<< "] intersect with: ";
			for (int j = 0; j < m_SegmentConflicts[i].size(); ++j)
			{
				int iNum = m_SegmentConflicts[i][j];
				out << m_vecLastResponse[brunch][iNum].m_strEuristicName << " [" << m_vecLastResponse[brunch][iNum].m_iFrom << ',' << m_vecLastResponse[brunch][iNum].m_iTo << "] ";
			}
```

```
                out << "\n";
            }
        }

        void PrintFeatureConflicts(fstream& out)
        {
            out << "FEATURES CONFLICTS:\n";
            int iCnt = 0;
            for (auto it = m_FeaturesConflicts.begin(); it != m_FeaturesConflicts.end(); ++it)
            {
                iCnt++;
                out << iCnt << ' ' << it->first << " with following different values: ";
                for (int i = 0; i < it->second.size(); ++i)
                {
                    Euristic& tmp = m_mpEuristics[it->second[i].m_strEuristicName];
                    out << it->second[i].m_iValue << " (" << tmp.m_strEuristicValue << ")
";
                }
                out << "\n";
            }
        }

        void UserAnswer( int pos);//agree or disagree with results
        vector<string> AnswerOfFunc(int pos);//retrn string of ID of the drug from Card

    public:
        vector<vector<pair<string, double> > > CardFeatureValuesInSet; //collection of all values in
set for every particular feature
        int NumbOfUserRequest;//size of m_vecLastRequest
        Driver(const string& inputName, const string& inputUserName, const string&
inputAnswerName, const string& strOutputFileName,const string& strOutStatisticName) :
m_strInputName(inputName), m_inputFile(inputName, std::fstream::in),
m_strUserInputName(inputUserName), m_inputUserFile(inputUserName, std::fstream::in),
m_inputAnswerFile(inputAnswerName, std::fstream::in),
m_strOutName(strOutputFileName),m_strAnswerName(inputAnswerName),m_strOutStatisticName(s
trOutStatisticName),m_bd(Card())
        {
            m_out.open(strOutputFileName, std::ios::out);
            m_out.close();
            ReadData();
        }

        void ReadData()
        {
            ReadEuristics();
            ReadFeatures();
            m_bd.ReadCard();
            ReadEuristicsInDB();
            FindNumbOfParentsSet();
            ReadRightAnswer();
        }
```

```
	void ReadUserRequest();

	void ReWriteEuristicInDB();

	void ProcessLastRequest( int pos);

	void CloseAll()
	{
		m_out.close();
	}

	void PrintWholeLastResponce(int pos)
	{
		//fstream out(strOuputFileName, std::fstream::out);
		m_out.open(m_strOutName,std::fstream::app);
		if (m_boolIsResultExist)//result is finded
		{
			PrintLastEntries(m_out);
		//m_out << endl << endl;
		//PrintSegmentConflicts(m_out);
		//m_out << endl << endl;
		//PrintFeatureConflicts(m_out);

		//m_out << endl << endl;
			if (m_boolIsResultRight)//пользователь согласился
			{
				m_out << "Пользователь подтвердил выбранное  программой лекарство." << endl;
				OutRightAnsw++;
			}
			else
			{
				m_out << "Пользователь НЕ подтвердил выбранное программой лекарство." << endl;
				OutWrongAnsw++;
			}
		}
		else
		{
			m_out << pos << endl;
			m_out << "Лекарство не найдено." << endl;
			OutEmptyAnsw++;
		}
		m_out.close();
	}

	void PrintStatistics()
	{
		m_outstatistic.open(m_strOutStatisticName, std::fstream::app);
```

```
            m_outstatistic << "Правильных:" << OutRightAnsw << " Неправильных:" << OutWrongAnsw << " Не распознанных:" << OutEmptyAnsw << " Выбранных пользователем:"<< OutChooseUser <<endl;

        }

    };
```

П3.2. Файл classes.cpp

```
    #include "classes.h"


    using namespace std;

    //Driver class implementation

    void Driver::ReadEuristics()
    {
        int iEurNum;
        m_inputFile >> iEurNum;

        for (int i = 0; i < iEurNum; ++i)
        {
                string curEurName;
                string curEurValue;
                int iPatNum;
                m_inputFile >> curEurName;// >> iPatNum;          //for now we'll assume that only one pattern corresponds to every heuristic
                iPatNum = 1;
                m_inputFile.get();
                getline(m_inputFile, curEurValue);  //read value for current euristic (separated from name by space)
                Euristic curEur(curEurName, curEurValue);

                for (int j = 0; j < iPatNum; ++j)        //every patttern on a single line
                {
                        string curPat;
                        //m_inputFile >> curPat;
                        getline(m_inputFile, curPat);
                        curPat = PreprocessString(curPat);
                        curEur.AddPattern(curPat);
                }
                //vecAllEuristics.push_back(curEur);
                m_mpEuristics.insert(make_pair(curEurName, curEur));
        }

    }
```

```
void Driver::ReadFeatures()
{
    int iFeatNum;
    m_inputFile >> iFeatNum;
    for (int i = 0; i < iFeatNum; ++i)
    {
        string curFeatureName;
        int iEurNumber;
        m_inputFile >> curFeatureName >> iEurNumber;
        m_inputFile.get();      //TODO: -//-

        Feature curFeature(curFeatureName);

        for (int j = 0; j < iEurNumber; ++j)
        {
            string curEurName;
            //m_inputFile >> curEurName;
            getline(m_inputFile, curEurName);
            auto it = m_mpEuristics[curEurName];
            //it.SetParentFeatureName(curFeatureName);
            m_mpEuristics[curEurName].SetParentFeatureName(curFeatureName);
            //curFeature.AddEuristic(&m_mpEuristics[curEurName]);
        }

        m_mpFeatures.insert(make_pair(curFeatureName, curFeature));
    }

    //delete dublicate Euristics
    vector<Euristic> vecAllEuristics;
    for (auto it = m_mpEuristics.begin(); it != m_mpEuristics.end(); ++it)
    {
        vecAllEuristics.push_back(it->second);
    }

    m_mpEuristics.clear();
    for (int i = 0; i < vecAllEuristics.size(); ++i)
    {
        for (int j = i + 1; j < vecAllEuristics.size(); ++j)
        {
            if (vecAllEuristics[i].m_vecPatterns == vecAllEuristics[j].m_vecPatterns &&
                vecAllEuristics[i].m_strParentFeatureName ==
vecAllEuristics[j].m_strParentFeatureName &&
                PreprocessString(vecAllEuristics[i].m_strEuristicValue) ==
PreprocessString(vecAllEuristics[j].m_strEuristicValue))
            {
                swap(vecAllEuristics[j], vecAllEuristics[vecAllEuristics.size() - 1]);
                vecAllEuristics.pop_back();
                j--;
            }
        }
    }
```

```
    m_mpEuristics.clear();
    for (int i = 0; i < vecAllEuristics.size(); ++i)
    {
            m_mpEuristics.insert(make_pair(vecAllEuristics[i].m_strName, vecAllEuristics[i]));
    }
    for (auto it = m_mpEuristics.begin(); it != m_mpEuristics.end(); ++it)
    {

            m_mpFeatures[it->second.m_strParentFeatureName].AddEuristic(&it->second);
    }

}

void Driver::ReadUserRequest()
{
    while (!m_inputUserFile.eof())
    {
            string s,tmp;
            getline(m_inputUserFile, s);
            m_vecLastRequest.push_back(s);
            tmp = PreprocessString(s);
            m_vecChangedRequest.push_back(tmp);
    }
    NumbOfUserRequest = m_vecLastRequest.size();


    //int a = 123;
/*
    int iRequestNum;
    m_inputUserFile >> iRequestNum;
    m_inputUserFile.get();          //TODO: be sure readed correctly
    //Request curRequest;
    for (int i = 0; i < iRequestNum; ++i)
    {
                    string curText;
                    getline(m_inputUserFile, curText);
                    nameRequest->AddRequest(curText);
                    //curRequest.AddRequest(curText);
    }
    */
}

void Driver::ReadRightAnswer()
{
    while (!m_inputAnswerFile.eof())
    {
            string str;
            m_inputAnswerFile >> str;
            RightAnswer.push_back(str);
    }
}
```

```
void Driver::ClearAllForNewRequest()
{

}

string Driver::PreprocessString(string str)
{
    //preprocessing of request - all characters to lowercase chars and delete spaces
    string tmp="";
    for (int i = 0; i < str.size(); ++i)
    {
            if (str[i] != ' ')
            {
                    str[i]= tolower(str[i]); //TODO: correct processing russian characters!
                    if (str[i] >= 'A' && str[i] <= 'Я')       //process russian characters
                    {
                            str[i]= str[i] - 'A' + 'a';
                    }
                    tmp += str[i];
            }
    }
    return tmp;
}
string Driver::ReplaceNumbers(string str)
{
    string ans;
    string lastNum;
    bool bDotWas = false;          //only one dot
    m_mpReplacedNumbers.clear();
    for (int i = 0; i < str.size(); ++i)
    {
            if (str[i] == ' ')
            {
                    if (!lastNum.empty())
                    {
                            m_mpReplacedNumbers.insert(make_pair(ans.size(), lastNum));
                            ans += '#';
                            lastNum.clear();
                            bDotWas = false;
                    }
                    continue;
            }
            if (str[i] >= '0' && str[i] <= '9')
            {
                    lastNum += str[i];
                    continue;
            }

            if ((str[i] == '.' || str[i] == ',') && bDotWas == false && !lastNum.empty())
            {
```

```
                bDotWas = true;
                lastNum += '.';
                continue;
            }

            if (!lastNum.empty())
            {
                m_mpReplacedNumbers.insert(make_pair(ans.size(), lastNum));
                ans += '#';
                lastNum.clear();
                bDotWas = false;
            }

            ans += str[i];

        }
        if (!lastNum.empty())
        {
            m_mpReplacedNumbers.insert(make_pair(ans.size(), lastNum));
            ans += '#';
            lastNum.clear();
            bDotWas = false;
        }
        return ans;
    }


    void Driver::ProcessLastRequest(int pos)
    {
        brunch = 0;
        m_vecLastResponse.clear();
        m_SegmentConflicts.clear();
        vector<Entry> tmpvec;
        m_vecChangedRequest[pos] = ReplaceNumbers(m_vecChangedRequest[pos]);
        for (auto it = m_mpEuristics.begin(); it != m_mpEuristics.end(); ++it)      //walking through the
Map, it - reference to pair: first - name, second - element
        {
            vector<Entry> curEntries = it->second.FindEntriesIn(m_vecChangedRequest[pos],
m_mpReplacedNumbers);
            tmpvec.insert(tmpvec.end(), curEntries.begin(), curEntries.end()); //add cur entries into
vector, which stores all entries for last request
        }
        m_vecLastResponse.push_back(tmpvec);
        FindOutSegmentConflicts();
        ResolveSegmentConflicts();
        int bestset;
        double bestvalue;
        bestvalue= -1e9;
        for ( int i = 0; i <m_vecLastResponse.size(); i++)
        {
            brunch = i;
```

```
                FindOutFeaturesValueConflicts();
                ResolveFeaturesValueConflicts();
                vector<int> tmp = GetEntriesIndexes(m_vecLastResponse[i]);
                if (CalcTargetFuncForEntries(tmp)> bestvalue)
                {
                        bestset = i;
                        bestvalue = CalcTargetFuncForEntries(tmp);
                }
        }
        brunch = bestset;
        UserAnswer(pos);
        RenewSupportAndConfidence();
    }


    void Driver::FindOutSegmentConflicts()
    {
        vector < pair<int, pair<int, int> > > scan;     //scanning segments <border index, <type (begin =
0/end = 1), euristic index> >
        const int SEGMENT_BEGIN = 0;
        const int SEGMENT_END = 1;
        scan.reserve(2 * m_vecLastResponse[brunch].size());
        for (int i = 0; i < m_vecLastResponse[brunch].size(); ++i)
        {
                scan.push_back(make_pair(m_vecLastResponse[brunch][i].m_iFrom,
make_pair(SEGMENT_BEGIN, i)));
                scan.push_back(make_pair(m_vecLastResponse[brunch][i].m_iTo,
make_pair(SEGMENT_END, i)));
        }
        sort(scan.begin(), scan.end());
        unordered_set<int> currentSet;
        m_SegmentConflicts.clear();
        m_SegmentConflicts.resize(m_vecLastResponse[brunch].size()); //for every Euristic we will
store list of conflict entries
        for (int i = 0; i < scan.size(); ++i)
        {
                int curNumber = scan[i].second.second;
                if (scan[i].second.first == SEGMENT_BEGIN)
                {
                        for (auto it = currentSet.begin(); it != currentSet.end(); ++it)
                        {
                                m_SegmentConflicts[curNumber].push_back(*it);
                                m_SegmentConflicts[*it].push_back(curNumber);
                        }
                        currentSet.insert(curNumber);
                }
                else
                {
                        currentSet.erase(curNumber);
                }
        }
```

```
}

void Driver::ResolveSegmentConflicts()
{
    vector<int> currentBestSet;
    double currentBestValue = -1e9;
    vector<pair <double, vector<int>>> AllValuesOfGeneratedSet;//curvalue,curset
    vector<int> nonConfilctSet;
    vector<int> conflictSet;

    for (int i = 0; i < m_SegmentConflicts.size(); ++i)
    {
        if (m_SegmentConflicts[i].size() == 0)
        {
            nonConfilctSet.push_back(i);
        }
        else
        {
            conflictSet.push_back(i);
        }
    }
    if (conflictSet.size() == 0)
    {
        currentBestSet = nonConfilctSet;

    AllValuesOfGeneratedSet.push_back(make_pair(CalcTargetFuncForEntries(currentBestSet),
currentBestSet));
    }
    for (int i = 1; i < (1 << conflictSet.size()) - 1 && i<10000; ++i)     //look over all subsets
    {
        vector<int> subset;
        for (int j = 0; j < conflictSet.size() ; ++j)
        {
            if (i & (1 << j))
            {
                subset.push_back(conflictSet[j]);
            }
        }
        //create current subset
        vector<int> curSet = nonConfilctSet;
        curSet.insert(curSet.end(), subset.begin(), subset.end());
        //process currents subset
        double curValue = CalcTargetFuncForEntries(curSet);
        if (curValue > currentBestValue)
        {
            currentBestValue = curValue;
            currentBestSet = curSet;
        }
        AllValuesOfGeneratedSet.push_back(make_pair(curValue, curSet));
    }
    //find 10% of AllValuesOfGeneratedSet
```

```
        int count = ceil(AllValuesOfGeneratedSet.size()*0.1);
        if (count < 2)
        {
                count = AllValuesOfGeneratedSet.size();
        }
        sort(AllValuesOfGeneratedSet.begin(), AllValuesOfGeneratedSet.end());//10% are at the end;
        vector <vector <Entry>> proba;
        for (int i = 0; i < count;i++)
        {
                vector<Entry> tmp;
                int ind = AllValuesOfGeneratedSet.size() - 1 - i;
                for (int j = 0; j < AllValuesOfGeneratedSet[ind].second.size(); ++j)//change old set on
the finded set
                {

        tmp.push_back(m_vecLastResponse[0][AllValuesOfGeneratedSet[ind].second[j]]);
                }
                proba.push_back(tmp);
        }
        m_vecLastResponse = proba;
        //vector<Entry> tmp;
        //for (int i = 0; i < currentBestSet.size(); ++i)//change old set on the finded set
        //{
        //        tmp.push_back(m_vecLastResponse[currentBestSet[i]]);
        //}
        //m_vecLastResponse = tmp;

        //FindOutSegmentConflicts();
        //check for correct finding out (after resolving must be no conflicts)
    }
    double Driver::CalcTargetFuncForEntries(vector<int> set)
    {

        if (IsConsistInDB(set)==0)//not found in DB
        {
                return -1e9;
        }
        else
        {
                //возвращаем корень из параметров
                vector<string> tmp =OrderEuristic(set);
                return FindSetEuristicInDB(tmp);
                //находим такой набор в бд эвристик
        }
    }
    bool Driver::IsConsistInDB(vector<int> set, int featuresCnt)
    {
        //TODO: implement  working with Card
        AddCardFeatureValuesInSet(set);
        //compare with Card
        for (int i = 0; i < m_bd.CardValue.size(); i++)
```

```
	{
		bool fl = 1;//continue search
		for (int j = 0; j < min((int)m_bd.CardValue[0].size(), featuresCnt); j++)
		{
			if (m_bd.CardValue[i][j].first == "!" && CardFeatureValuesInSet[j].size() != 0)// must be empty value
			{
				fl = 0;
				break;
			}
			else
			{
				if (CardFeatureValuesInSet[j].size() == 0)//empty //m_bd.CardValue[i][j].first == "!" &&
				{
					fl = 1;
					//break;
				}
				else
				{
					if (m_bd.CardValue[i][j].first != "*" && m_bd.CardValue[i][j].first != "!")// text value
					{
						fl = 0;
						auto ptrBd = m_bd.CardValue[i][j];
						for (int k = 0; k < CardFeatureValuesInSet[j].size(); ++k)
						{
							auto ptrSet = CardFeatureValuesInSet[j][k];
							if (ptrSet.first == ptrBd.first)
							{
								if ((ptrBd.second == VALUE_DOES_NOT_EXISTS && ptrSet.second == VALUE_DOES_NOT_EXISTS) || ptrBd.second == VALUE_COULD_BE_ANY)
								{
									fl = 1;
									break;
								}
								if (ptrBd.second == ptrSet.second)
								{
									fl = 1;
									break;
								}
							}
						}
					}
				}
				if (fl == 0)
					break;
			}
		}
		if (fl == 1)
```

```
                {
                        DeleteCardFeatureValuesInSet();
                        return 1;

                }
        }
        DeleteCardFeatureValuesInSet();
        return 0;

    }
    vector<int> Driver::FindLinesStrInDB(vector<int> set, int featuresCnt)
    {
        //TODO: implement  working with Card
        AddCardFeatureValuesInSet(set);
        vector <int> allNumbLines;
        //compare with Card
        for (int i = 0; i < m_bd.CardValue.size(); i++)
        {
                bool fl = 1;//continue search
                for (int j = 0; j < min((int)m_bd.CardValue[0].size(), featuresCnt); j++)
                {
                        if (m_bd.CardValue[i][j].first == "!" && CardFeatureValuesInSet[j].size() !=
0)// must be empty value
                        {
                                fl = 0;
                                break;
                        }
                        else
                        {
                                if (CardFeatureValuesInSet[j].size() == 0)//empty
//m_bd.CardValue[i][j].first == "!" &&
                                {
                                        fl = 1;
                                        //break;
                                }
                                //}
                                else
                                {
                                        if (m_bd.CardValue[i][j].first != "*" &&
m_bd.CardValue[i][j].first != "!")// text value
                                        {
                                                fl = 0;
                                                auto ptrBd = m_bd.CardValue[i][j];
                                                for (int k = 0; k < CardFeatureValuesInSet[j].size(); ++k)
                                                {
                                                        auto ptrSet = CardFeatureValuesInSet[j][k];
                                                        if (ptrSet.first == ptrBd.first)
                                                        {
                                                                if ((ptrBd.second == VAL-
UE_DOES_NOT_EXISTS && ptrSet.second == VALUE_DOES_NOT_EXISTS) || ptrBd.second ==
VALUE_COULD_BE_ANY)
```

```
                                                {
                                                        fl = 1;
                                                        break;
                                                }
                                                if (ptrBd.second == ptrSet.second)
                                                {
                                                        fl = 1;
                                                        break;
                                                }
                                        }
                                }
                        }
                        if (fl == 0)
                                break;
                }
        }
    }
    if (fl == 1)
    {
        //DeleteCardFeatureValuesInSet();
        //return 1;
        allNumbLines.push_back(i);

    }
  }
  DeleteCardFeatureValuesInSet();
  return allNumbLines;
}

void Driver::FindOutFeaturesValueConflicts()
{
  m_FeaturesConflicts.clear();
  for (int i = 0; i < m_vecLastResponse[brunch].size(); ++i)
  {
        string strCurEurName = m_vecLastResponse[brunch][i].m_strEuristicName;
        Euristic& curEur = m_mpEuristics[strCurEurName];
        string strFeatureName = curEur.m_strParentFeatureName;
        m_FeaturesConflicts[strFeatureName].push_back(m_vecLastResponse[brunch][i]);

  }
}
void Driver::ResolveFeaturesValueConflicts()
{
  for (int i = 0; i < m_bd.CardFeatureName.size(); ++i)
  {
        vector<Entry>& it = m_FeaturesConflicts[m_bd.CardFeatureName[i].first];
        if (it.size() <= 1)
        {
                //конфликтов по этому признаку нет
        }
        else
```

```
		{
			//for every particular Euristic we need only first entry

			unordered_map < string, vector<Entry> > eurToEntry;
			for (int j = 0; j < it.size(); ++j)
			{
				string eurName = it[j].m_strEuristicName;
				eurToEntry[eurName].push_back(it[j]);
			}

			vector<Entry> newVec;
			for (auto itEur = eurToEntry.begin(); itEur != eurToEntry.end(); ++itEur)
			{
				Entry cur = itEur->second[0];
				int curFrom = itEur->second[0].m_iFrom;
				for (int j = 0; j < itEur->second.size(); ++j)
				{
					if (itEur->second[j].m_iFrom < curFrom)
					{
						curFrom = itEur->second[j].m_iFrom;
						cur = itEur->second[j];
					}
				}
				newVec.push_back(cur);
			}

			it = newVec;
		}
	}
	//according to catalog and bd we may choose first 5 (or not 5) values
	int firstnumb = 3;
	vector<vector<Entry>> tmpAnswer;
	vector<vector<Entry>> allbestvalues=ChooseValuesForMainFeatures(3);
	for (int st = 0; st < allbestvalues.size(); ++st)
	{
		vector<Entry> bestvalues=allbestvalues[st];
		//solve other conflicts
		vector<string> NamesOfBestValues;
		for (int i = 0; i < bestvalues.size(); ++i)
		{
			NamesOfBestValues.push_back(bestvalues[i].m_strEuristicName);
		}
		sort(NamesOfBestValues.begin(), NamesOfBestValues.end());
		vector<int> curSet = GetEntriesIndexes(bestvalues);
		for (int i = bestvalues.size(); i < m_bd.CardFeatureName.size(); ++i)
		{
			vector<Entry>& it = m_FeaturesConflicts[m_bd.CardFeatureName[i].first];
			double value = -1e9;
			int numb;
			for (int j = 0; j < it.size(); ++j)
			{
```

```
                                vector<Entry> tmpGenSet = bestvalues;
                                vector <string> curName;
                                curName.push_back(it[j].m_strEuristicName);
                                if (IsSubset(NamesOfBestValues, curName))//such component is already
exist in  current set
                                {
                                        break;
                                }
                                tmpGenSet.push_back(it[j]);
                                vector <int> tmpSet = GetEntriesIndexes(tmpGenSet);
                                if (value < CalcTargetFuncForPartialSet(tmpSet, i+1))
                                {
                                        value = CalcTargetFuncForPartialSet(tmpSet, i+1);
                                        numb = j;
                                }

                        }
                        if (value == -1e9)//make sign that means any value of this feature
                        {

                        }
                        else
                        {
                                bestvalues.push_back(it[numb]);//add feature
                        }

                }
                //m_vecAnswerTree[brunch].push_back(bestvalues);
                tmpAnswer.push_back(bestvalues);
        }
        //m_vecAnswerTree.push_back(tmpAnswer);
        //find the best  set from all this brunch
        double bestvalue = -1e9;
        int bestindex;
        for (int i = 0; i < tmpAnswer.size(); ++i)
        {
                vector<int> tmp = GetEntriesIndexes(tmpAnswer[i]);
                if (CalcTargetFuncForEntries(tmp)> bestvalue)
                {
                        bestindex = i;
                        bestvalue = CalcTargetFuncForEntries(tmp);
                }
        }
        m_vecLastResponse[brunch]=tmpAnswer[bestindex];
        //FindOutSegmentConflicts(); // нельзя теперь проверять снова!!!
        //FindOutFeaturesValueConflicts();

    }


    double Driver::CalcTargetFuncForPartialSet(vector<int> set, int featuresCnt)
```

```
{
    if (IsConsistInDB(set, featuresCnt)==0)
    {
            return -1e9;
    }
    else
    {
            vector<string> tmp = OrderEuristic(set);
            return FindPartialSetEuristicInBD(tmp);
    }
}

double Driver::FindPartialSetEuristicInBD(vector<string> set)
{
    double maxVal = -1e9;
    for (auto it = m_mpSetEuristicInDB.begin(); it != m_mpSetEuristicInDB.end(); ++it)
    {
            if (IsSubset(it->first, set))
            {
                    double a = it->second.Confidence*it->second.Support;
                    maxVal = max(maxVal, a);
            }
    }
    if (maxVal == -1e9)
    {
            return DEFAULT_CONFIDENCE*DEFAULT_SUPPORT;
    }
    return maxVal;
}

bool Driver::IsSubset(vector<string> a, vector<string> b)       //check if b is subset of a
{
    int ptr1 = 0, ptr2 = 0;
    while (ptr2 < b.size())
    {
            if (ptr1 >= a.size())
            {
                    return false;
            }
            if (a[ptr1] == b[ptr2])
            {
                    ptr1++;
                    ptr2++;
                    continue;
            }
            if (a[ptr1] < b[ptr2])
            {
                    ptr1++;
                    continue;
            }
            if (a[ptr1] > b[ptr2])
```

```
                {
                        return false;
                }
        }
        return true;
}

vector <vector<Entry>> Driver::ChooseValuesForMainFeatures(int featuresCnt)
{
        vector<vector<Entry> > allSets;
        vector<pair <double, vector<int>>> AllValuesOfGeneratedSet;//curvalue,curset
        GenerateAllValues(allSets, vector<Entry>(), 0, featuresCnt);
        int ind = -1;
        double bestval = -1e9;
        for (int i = 0; i < allSets.size(); ++i)
        {
                vector<int> curSet = GetEntriesIndexes(allSets[i]);
                double val= CalcTargetFuncForPartialSet(curSet, featuresCnt+1);
                if (val> bestval)
                {
                        bestval = val;
                        ind = i;
                }
                if (val == -1e9)//can't find in bd such value
                {

                }
                else
                {
                        AllValuesOfGeneratedSet.push_back(make_pair(val, curSet));
                }
        }

        if (bestval == -1e9)
        {
                //TODO:: featuresCnt not found!!!!!!!!!! PLEASE, DO SMTH!
                if (featuresCnt == 1)
                {
                        //throw new exception("Cant find main features");
                }

                return ChooseValuesForMainFeatures(featuresCnt-1);
        }
        else
        {
                //find 20% sets with best characters;
                int count = ceil(AllValuesOfGeneratedSet.size()*0.1);
                sort(AllValuesOfGeneratedSet.begin(), AllValuesOfGeneratedSet.end());//20% are at
the end;
                vector <vector <Entry>> proba;
                for (int i = 0; i < count; i++)
```

```
		{
			vector<Entry> tmp;
			int ind = AllValuesOfGeneratedSet.size() - 1 - i;
			for (int j = 0; j < AllValuesOfGeneratedSet[ind].second.size(); ++j)//change old
set on the finded set
			{

	tmp.push_back(m_vecLastResponse[brunch][AllValuesOfGeneratedSet[ind].second[j]]);
			}
			proba.push_back(tmp);
		}
		return proba;


		//return allSets[ind];
	}

}

vector<int> Driver::GetEntriesIndexes(vector<Entry> cur)
{
	vector<int> ans;
	for (int i = 0; i < cur.size(); ++i)
	{
		bool found = false;
		for (int j = 0; j < m_vecLastResponse[brunch].size(); ++j)
		{
			if (cur[i] == m_vecLastResponse[brunch][j])
			{
				ans.push_back(j);
				found = true;
				break;
			}
		}
		if (!found)
		{
			throw new exception("cant find index for given Entry");
		}
	}
	return ans;
}

void Driver::GenerateAllValues(vector<vector<Entry> > & answer, vector<Entry> curVec, int ptr,
int totalCnt)
{
	if (ptr == totalCnt)
	{
		answer.push_back(curVec);
		return;
	}
	vector<Entry>& it = m_FeaturesConflicts[m_bd.CardFeatureName[ptr].first];
```

```
        if (it.size() == 0)
        {
                GenerateAllValues(answer, curVec, ptr + 1, totalCnt);
        }
        for (int i = 0; i < it.size(); ++i)
        {
                curVec.push_back(it[i]);
                GenerateAllValues(answer, curVec, ptr + 1, totalCnt);
                curVec.pop_back();
        }

    }

    void Driver::AddCardFeatureValuesInSet(vector <int> set)
    {
        CardFeatureValuesInSet.resize(m_bd.CardFeatureName.size());

        for (int i = 0; i < set.size(); i++)
        {
                for (int j = 0; j < m_bd.CardFeatureName.size(); j++)
                {
                        //записать эвристики по соотв признакам
                        bool fl = 0;

                        Euristic&                                                it                                                =
m_mpEuristics[m_vecLastResponse[brunch][set[i]].m_strEuristicName];
                        /*
                        for (auto it = m_mpEuristics.begin(); it != m_mpEuristics.end(); ++it)
        //walking through the Map, it - reference to pair: first - name, second - element
                        {
                                if (m_vecLastResponse[i].m_strEuristicName==it->second.m_strName)
                                {
                                */
                        if (it.m_strParentFeatureName == m_bd.CardFeatureName[j].first)
                        {
                                pair<string, double> tmp;
                                tmp.first = it.m_strEuristicValue;
                                tmp.second = m_vecLastResponse[brunch][set[i]].m_iValue;
                                CardFeatureValuesInSet[j].push_back(tmp);
                                fl = 1;
                                break;
                        }
                        if (fl)//feature was founded
                                break;
                }
        }

    }
    void Driver::DeleteCardFeatureValuesInSet()
    {
        CardFeatureValuesInSet.clear();
```

```
}

void Driver::ReadEuristicsInDB()
{
    ifstream f;
    f.open("seteuristicinbd.txt");
    NumberOfGivenExamples = 0;
    while (!f.eof())
    {
            string curLine;
            getline(f, curLine);
            stringstream tmp;
            tmp << curLine;
            vector<string> set;
            while (tmp)
            {
                    string curEurName;
                    tmp >> curEurName;
                    if (!tmp)
                            break;
                    set.push_back(curEurName);
            }
            getline(f, curLine);
            stringstream tmp2;
            tmp2 << curLine;
            //pair<double, double> curVal;
            //tmp2 >> curVal.first >> curVal.second;
            //m_mpSetEuristicInBd[set] = curVal;

            NumericalValuesInDB curVal;
            tmp2 >> curVal.Confidence >> curVal.Support;
            getline(f, curLine);
            stringstream tmp3;
            tmp3 << curLine;
            tmp3 >> curVal.RightAns >> curVal.WrongAns;
            NumberOfGivenExamples+= curVal.RightAns+curVal.WrongAns;
            curVal.NumbOfParentsSet = 0;
            //m_mpStatisticsOfResultsIn[set]= curVal2;
            m_mpSetEuristicInDB[set] = curVal;

    }

    f.close();
}
void Driver::ReWriteEuristicInDB()
{
    ofstream f;
    f.open("seteuristicinbd.txt", std::ios::out | std::ios::trunc);
    //auto it2 = m_mpStatisticsOfResultsInBd.begin();
    for (auto it = m_mpSetEuristicInDB.begin(); it != m_mpSetEuristicInDB.end(); ++it)
    {
```

```
            if (it != m_mpSetEuristicInDB.begin())
            {
                f << endl;
            }
            for (int i = 0; i < it->first.size(); ++i)
            {
                f << it->first[i] << ' ';
            }
            f << endl;
            f << setprecision(2)<<it->second.Confidence<< ' ' << it->second.Support << endl;
            f << it->second.RightAns << ' ' << it->second.WrongAns;
    }

    //ToDo::заполнить m_mpSetEuristicInBd в файл
    f.close();
}
double Driver::FindSetEuristicInDB(vector<string> set)
{
    auto it = m_mpSetEuristicInDB.find(set);
    if (it == m_mpSetEuristicInDB.end())
    {
        return DEFAULT_CONFIDENCE*DEFAULT_SUPPORT;
    }
    return it->second.Confidence*it->second.Support;
}
vector <string> Driver::OrderEuristic(vector <int> set)
{
    vector <string> tmp;
    for (int i = 0; i < set.size(); i++)
    {
            //записать эвристики в алфавитном порядке

        //TODO: to be sure, that we have to push value
        tmp.push_back(m_vecLastResponse[brunch][set[i]].m_strEuristicName);
        //Euristic& it = m_mpEuristics[m_vecLastResponse[set[i]].m_strEuristicName];
        //tmp.push_back(it.m_strEuristicValue);

    }
    if (tmp.size() <= 1)
    {
        return tmp;
    }
    sort(tmp.begin(), tmp.end());
    vector <string> diftmp;
    for (int i = 0; i < tmp.size()-1; i++)
    {
        if (tmp[i] != tmp[i + 1] )
        {
            diftmp.push_back(tmp[i]);
        }
    }
```

```
        if (tmp.size() > 1)
        {
                if (tmp[tmp.size() - 2] != tmp[tmp.size() - 1])
                {
                        diftmp.push_back(tmp[tmp.size() - 1]);
                }
        }
        return diftmp;
    }

    vector<string> Driver::AnswerOfFunc(int pos)//return string of ID of the drug from Card
    {
        vector<int> curSet = GetEntriesIndexes(m_vecLastResponse[brunch]);
        vector<string> ans;
        ans.clear();
        if (curSet.empty())
        {
                //throw new exception("Can't recognize the string.");
                return ans;
        }
        vector <int> NumbOfLines = FindLinesStrInDB(curSet);
        //correct answers
        //if (NumbOfLines.size() == 1)//answer is definite
        //{
        //      ans.push_back(m_bd.CardValue[NumbOfLines[0]][m_bd.CardValue[0].size() -
1].first);
        //      return ans;
        //}
        if (NumbOfLines.size() < 10)//may be we have right answer throw others
        {
                //// ask user to choose best variant from the list
                //setlocale(LC_CTYPE, "rus");
                //cout << "Ваш запрос №"<<pos<<":" << endl;
                //cout << m_vecLastRequest[pos] << endl;
                //cout << "Выберите наиболее подходящий вариант из предложенных. Введите
номер варианта." << endl;
                //for (int i = 0; i < NumbOfLines.size(); ++i)
                //{
                //      cout << "Вариант " << i << " :" << endl;
                //      for (int j = 0; j < m_bd.CardValue[0].size() - 1; ++j)
                //      {
                //              if      (m_bd.CardValue[NumbOfLines[i]][j].first      !=      "!"      &&
m_bd.CardValue[NumbOfLines[i]][j].first != "*")
                //              {
                //                      cout << m_bd.CardValue[NumbOfLines[i]][j].first << ' ';
                //              }
                //              if      (m_bd.CardValue[NumbOfLines[i]][j].second      !=-1      &&
m_bd.CardValue[NumbOfLines[i]][j].second != -2)
                //              {
                //                      cout<<m_bd.CardValue[NumbOfLines[i]][j].second << ';';
                //              }
```

```
                //        }
                //        cout << endl;
                //}
                ////write user's answer
                //int variant_answ;
                //cin >> variant_answ;
                ////TODO:: check the right format of variant_answ
                //string ans= m_bd.CardValue[NumbOfLines[variant_answ]][m_bd.CardValue[0].size()
- 1].first;
                //OutChooseUser++;
                //return ans;
                for (int i = 0; i < NumbOfLines.size(); ++i)
                {
                        ans.push_back(m_bd.CardValue[NumbOfLines[i]][m_bd.CardValue[0].size()   -
1].first);
                }
                return ans;
        }
        if (NumbOfLines.size() >= 10)//bad recosnition of rensponse
        {
                //throw new exception("Can't recognize the string.");
                return ans;
        }
    }
    void Driver::UserAnswer(int pos)//agree or disagree with results
    {
        vector<string> ans=AnswerOfFunc(pos);
        bool fl = false;
        for (int  i = 0; i < ans.size(); ++i)
        {
                if (ans[i] == RightAnswer[pos])
                {
                        //right result
                        m_boolIsResultRight = true;
                        m_boolIsResultExist = true;
                        fl = true;
                        break;
                }
        }
        if (!fl)//didn't find the result
        {
                //empty result
                if (ans.size() == 0)
                {
                        m_boolIsResultExist = false;
                }
                else
                {//false result
                        m_boolIsResultRight = false;
                        m_boolIsResultExist = true;
                }
```

```
    }
    //if (ans == RightAnswer[pos])
    //{
    //        //right result
    //        m_boolIsResultRight = true;
    //        m_boolIsResultExist = true;
    //}
    //else
    //{
    //        //empty result
    //        if (ans == "")
    //        {
    //                m_boolIsResultExist = false;
    //        }
    //        else
    //        {//false result
    //                m_boolIsResultRight = false;
    //                m_boolIsResultExist = true;
    //        }
    //}
}
void Driver::RenewSupportAndConfidence()
{

    //if set exists in DB
    vector<int> curSet = GetEntriesIndexes(m_vecLastResponse[brunch]);
    if (!m_boolIsResultExist)      //cant find the result at all
    {
            //don't renew the file
    }
    else
    {
            vector<string> tmp = OrderEuristic(curSet);
            auto it = m_mpSetEuristicInDB.find(tmp);
            if (it == m_mpSetEuristicInDB.end())//set doesn't exist in DB
            {
                    //add new line in DB
                    NumericalValuesInDB curVal;
                    curVal.Confidence = DEFAULT_CONFIDENCE;
                    curVal.Support = DEFAULT_SUPPORT;
                    curVal.NumbOfParentsSet = 1;
                    if (m_boolIsResultRight)
                    {
                            curVal.RightAns = 1;
                            curVal.WrongAns = 0;
                    }
                    else
                    {
                            curVal.RightAns = 0;
                            curVal.WrongAns = 1;
                    }
```

```
                m_mpSetEuristicInDB[tmp] = curVal;
                NumberOfGivenExamples++;
            }
            else
            {
                //change parameters
                if (m_boolIsResultRight)
                {
                    ++m_mpSetEuristicInDB[tmp].RightAns;
                }
                else
                {
                    ++m_mpSetEuristicInDB[tmp].WrongAns;
                }
                NumberOfGivenExamples++;
                // count right,wrong answers in lines which have such set
                int tmp_all = 0, tmp_right = 0, tmp_wrong = 0;
                for (auto i = m_mpSetEuristicInDB.begin(); i != m_mpSetEuristicInDB.end();
++i)
                {
                    if (IsSubset(i->first, tmp))
                    {
                        tmp_all += i->second.RightAns + i->second.WrongAns;
                        tmp_right += i->second.RightAns;
                        tmp_wrong += i->second.WrongAns;
                    }
                }
                // change parameters for current set
                m_mpSetEuristicInDB[tmp].Confidence = (tmp_right - tmp_wrong + 0.0) /
tmp_all;
                m_mpSetEuristicInDB[tmp].Support = tmp_all / (NumberOfGivenExamples +
0.0);
            }
            // change parents for other lines, which are
            ChangeParametersInSubset(tmp);
            //change parameters for other lines
        }

    }

    void Driver::FindNumbOfParentsSet()
    {
        for (auto i = m_mpSetEuristicInDB.begin(); i != m_mpSetEuristicInDB.end(); ++i)
        {
            for (auto j = m_mpSetEuristicInDB.begin(); j != m_mpSetEuristicInDB.end(); ++j)
            {
                if (IsSubset(j->first, i->first))//check if i is subset of j
                {
                    i->second.NumbOfParentsSet+=j->second.RightAns+j-
>second.WrongAns;
                }
```

```
		}
	}
}
void Driver::ChangeParametersInSubset(vector <string> set)
{
		for (auto j = m_mpSetEuristicInDB.begin(); j != m_mpSetEuristicInDB.end(); ++j)
		{
			if (IsSubset(set,j->first)&&(j->first!=set))//check if j is subset of set
			{
				++(j->second.NumbOfParentsSet);
				if (m_boolIsResultRight)
				{
					j->second.RightAns++;
				}
				else
				{
					j->second.WrongAns++;
				}
				j->second.Support = j->second.NumbOfParentsSet / (NumberOfGivenExamples+0.0);
				j->second.Confidence = (j->second.RightAns - j->second.WrongAns+0.0) / j->second.NumbOfParentsSet;
			}
		}

}

//Card class methods implementation

pair<string, double> Card::ParseValues(const string& str)
{
    int ptr = str.find(':');
    pair<string, double> ans;
    ans.first = str.substr(0, ptr);
    ans.second = atof(str.substr(ptr + 1).c_str());
    return ans;
}

vector<pair<string, double> > Card::ParseString(const string & str, int pos)
{
    vector<pair<string, double> > ans;
    int i = str.find(';', pos);
    int j= str.find(':', pos);
    if (i == -1)
    {
		ans.push_back(ParseValues(str.substr(pos)));
    }
    else
    {
		string s = str.substr(pos, i - pos);
		ans.push_back(ParseValues(s));
```

```
            vector<pair<string, double> > tmp = ParseString(str, i + 1);
            ans.insert(ans.end(), tmp.begin(), tmp.end());
        }
        return ans;
    }


    void Card::ReadCard()
    {
        //read Card
        ifstream f;
        f.open("test.csv");
        string s;
        getline(f, s);//read name of features in card
        CardFeatureName = ParseString(s,0);
        //AddCardFeatureName(s);
        while (!f.eof())
        {
            getline(f, s);
            CardValue.push_back(ParseString(s, 0));
        }

    }

    //Euristic class methods implementation


    vector<Entry> Euristic::FindEntriesIn(const string& strRequest, unordered_map<int, string>&
mpReplacedNumbers)
    {
        vector<Entry> vecAnswer;
        Pattern::patternResponse vecAllPattResponses;
        for (int i = 0; i < m_vecPatterns.size(); ++i)
        {
            Pattern::patternResponse vecPattResponse =
m_vecPatterns[i].FindEntriesIn(strRequest, mpReplacedNumbers);
            vecAllPattResponses.insert(vecAllPattResponses.end(), vecPattResponse.begin(),
vecPattResponse.end());
        }

        ResolveInternalConflicts(vecAllPattResponses);

        for (int i = 0; i < vecAllPattResponses.size(); ++i)
        {
            vecAnswer.push_back(Entry(m_strName, vecAllPattResponses[i].first.first,
vecAllPattResponses[i].first.second, vecAllPattResponses[i].second));
        }

        return vecAnswer;

    }
```

```
void Euristic::ResolveInternalConflicts(Pattern::patternResponse & strRequest)
{
    //TODO: important: resolve all internal conflicts (similar entries)
}

//Pattern class methods implementation

Pattern::patternResponse Pattern::FindEntriesIn(const string& strRequest, unordered_map<int, string>& mpReplacedNumbers)
{
    //TODO: important, implement finding entries for different types of patterns

    int iPtr = m_strPattern.find('#');        //pointer to '#'



    //naive implementation: (works only for non-numerical patterns)
    Pattern::patternResponse answer;
    int indexFrom = 0;
    int indexOfPat = 0;
    while (1)
    {
        indexOfPat = strRequest.find(m_strPattern, indexFrom);
        //indexOfPat = str.find_first_of(curPatt, indexFrom);

        if (indexOfPat == -1)
        {
            break;
        }
        else
        {
            //entry is found, add it to the answer
            indexFrom = indexOfPat + 1;
            double iCurValue = VALUE_DOES_NOT_EXISTS;
            if (iPtr != -1)
            {

                int tmp = strRequest.find('#', indexOfPat);
                //iCurValue = atof(mpReplacedNumbers[tmp].c_str());
                stringstream stream;
                stream<< mpReplacedNumbers[tmp].c_str();
                stream >> iCurValue;
            }
            answer.push_back(make_pair(make_pair(indexOfPat, indexOfPat + m_strPattern.size() - 1), iCurValue));

        }
    }
    return answer;
}
```

П3.3. Файл main.h

```
#pragma once

#include <stdio.h>
#include <cstring>
#include <stdlib.h>
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <map>
#include <unordered_map>
#include <unordered_set>
#include <iomanip>
#include <functional>
```

П3.4. Файл main.cpp

```
#include "main.h"
#include "classes.h"

using namespace std;



int main()
{

    Driver
MainDriver("EuristicsAndFeatures.txt","UserRequest.txt","RightAnswer.txt","out.txt","statistic.txt");

    //MainDriver.ReadData();
    MainDriver.ReadUserRequest();
    for (int pos = 0; pos < MainDriver.NumbOfUserRequest; ++pos)
    {
            MainDriver.ProcessLastRequest(pos);
            //MainDriver.FindOutSegmentConflicts();
            MainDriver.PrintWholeLastResponce(pos);
            cerr << pos << endl;
    }
    MainDriver.ReWriteEuristicInDB();
    MainDriver.PrintStatistics();
    MainDriver.CloseAll();
    return 0;
}
```