Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
«Институт космических и информационных технологий»
Кафедра «Высокопроизводительных вычислений»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Д.А. Кузьмин
подпись инициалы, фамилия
«____» _____20 __ г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Система поиска решений на базе открытых данных

09.04.01 «Информатика и вычислительная техника»

09.04.01.01 «Высокопроизводительные вычислительные системы»

Научный руководитель	подпись, дата	к.т.н., доцент должность, ученая степень	Д.А. Кузьмин инициалы, фамилия
Выпускник	подпись, дата		А.Н. Хохлов инициалы, фамилия
Нормоконтролер	подпись, дата		В.И. Иванов инициалы, фамилия
Рецензент	подпись, дата	к.т.н., доцент должность, ученая степень	Ф.А. Казаков инициалы, фамилия

Красноярск 2018

РЕФЕРАТ

Магистерская диссертация по теме «Система поиска решений на базе открытых данных» содержит 61 страницу, 58 иллюстраций, 5 таблиц, 9 формул, 1 приложение, 11 использованных источников.

ОТКРЫТЫЕ ДАННЫЕ, КЛАССИФИКАЦИЯ, КЛАСТЕРИЗАЦИЯ, МАШИННОЕ ОБУЧЕНИЕ, АНАЛИЗ ДАННЫХ, РАСПОЗНАВАНИЕ ОБРАЗОВ.

Цель работы: разработка, обоснование и реализация прототипа программного комплекса для работы с открытыми научными данными.

Задачи работы:

провести анализ предметной области концепции открытых данных в научных исследованиях, требований, предъявляемых к форматам их представления и выполнить обзор уже разработанных программных комплексов, выявить их достоинства и недостатки;

- выбрать подходящую платформы и язык программирования для реализации программного комплекса;
- реализовать и исследовать алгоритмы поиска и алгоритмы кластеризации: FOREL, k-средних, кластеризатор на основе самоорганизующейся сети Кохонена;
- проанализировать особенности работы алгоритмов на модельных и реальных данных, сравнить полученные результаты;
- программно реализовать прототип программной системы поиска решений на базе открытых данных;
- оценить скорость и качество работы созданного программного обеспечения в сравнении с уже имеющимися программными продуктами.

Основные результаты, связанные с исследованием и разработкой программной системы, получены автором лично. Постановка задач и обсуждение возможных путей их решения осуществлялись совместно с руководителем.

Актуальность и практический аспект проблемы открытых данных связаны с широким распространением практических задач в жизни общества, решение которых без дополнительных разработок и исследований невозможно.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Высокопроизводительные вычисления
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Д.А. Кузьмин
подпись инициалы, фамилия
« _____» _____20 ___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме магистерской диссертации
бакалаврской работы, дипломного проекта, дипломной работы, магистерской диссертации

Студенту Хохлову Александру Николаевичу
фамилия, имя, отчество

Группа КИ16-01-1М Направление (специальность) 09.04.01
номер код

Информатика и вычислительная техника
наименование

Тема выпускной квалификационной работы «Система поиска решений на базе открытых данных».

Утверждена приказом по университету № _____ от _____

Руководитель ВКР Кузьмин Д.А., канд. техн. наук, доцент, заведующий кафедрой «Высокопроизводительных вычислений».

Исходные данных для ВКР: язык программирования C++, доступ к системе электронного обучения Сибирского Федерального Университета.

Перечень разделов ВКР: введение, аналитическая часть, исследование и построение решения, разработка и тестирование программного комплекса, заключение, список использованных источников, приложение (листинг программы).

Перечень графического материала: скриншоты программ, скриншоты распределения данных, блок-схема работы приложения, презентация в формате «pptx», выполненная в Microsoft Office PowerPoint.

Руководитель ВКР _____ Д.А. Кузьмин
подпись, дата инициалы, фамилия

Задание принял к исполнению _____ А.Н. Хохлов
подпись, дата инициалы, фамилия

« ___ » _____ 2018 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ....3
1 Аналитическая часть....5
1.1 Общие сведения....5
1.2 Задача интеллектуального анализа данных....8
1.3 Задачи классификации и кластеризации....10
1.4 Обзор решаемых задач....11
1.5 Выбор программных средств разработки....19
1.6 Выводы....20
2 Исследование и построение решения....21
2.1 Постановка задачи....21
2.2 Требования к системе....21
2.3 Структурная схема программной системы....22
2.4 Анализ существующих систем....24
2.5 Обзор методов классификации и кластеризации....27
2.6 Выводы....44
3 Разработка и тестирование программного комплекса....45
3.1 Модель базы данных....45
3.2 Схема работы программы....46
3.3 Визуализация многомерных данных....48
3.4 API для поддержки....49
3.5 Численные исследования....50
3.6 Выводы....59
ЗАКЛЮЧЕНИЕ....60
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ....61
ПРИЛОЖЕНИЕ А Исходный код программы....62

ВВЕДЕНИЕ

В последние годы развитие концепции «открытых данных» идёт семимильными шагами: из сугубо специализированной темы она превратилась в предмет, имеющий актуальность и оказывающий влияние во всех странах мира. Открытые данные приобретают всё большее значение в качестве элемента разработки социально-экономической политики, причём как в развитых, так и в развивающихся странах.

В основе концепции открытых данных лежит идея о том, что определённые данные должны находиться в свободном доступе для неоднократного использования любыми заинтересованными сторонами. Данные, которые действительно «открыты», как правило, имеют следующие характеристики: они доступны в Интернете, представлены в цифровом и машиночитаемом формате, обеспечивающем операционную совместимость с другими данными; кроме того, отсутствуют какие-либо ограничения на их использование или повторную публикацию.

Цели движения открытых данных похожи на другие «открытые» движения, такие как открытое программное обеспечение (open source), открытый контент (open content) и открытый доступ (open access). Рост популярности идеи об открытых данных во второй половине 2000-х годов связан, прежде всего, с запуском правительственных инициатив, таких как Data.gov.

Открытые данные часто ассоциируются с нетекстовыми материалами, такими как карты, геномы, химические компоненты, математические и научные формулы, медицинские данные, данные о биологическом разнообразии. Проблемы чаще всего возникают по той причине, что эти данные могут быть коммерчески ценными или могут быть собраны в некие ценные продукты.

Доступ к данным, как и последующее их использование, контролируется организациями - государственными и частными. Контроль может быть через ограничения, лицензии, копирайт, патенты и требования оплаты для доступа или повторного использования. Сторонники идеи «открытых данных» считают, что подобные ограничения идут против общественного блага и данные должны быть доступны без ограничений или оплаты. Также важно что данные должны быть доступны без последующих запросов на разрешение, хотя и способы повторного использования, такие как создание продуктов на базе данных, могут контролироваться лицензией. Государственные данные представляют один из ключевых интересов для общества, и многочисленные некоммерческие организации и отдельные активисты добиваются открытости государственной информации в машиночитаемой форме. Многие национальные правительства в рамках стратегий «открытого государства» создали веб-сайты для распространения части данных, обрабатываемых в секторе государственного управления.

Существует немало областей, где «открытые данные» весьма полезны, а также много примеров того, как они используются и обеспечивают значительный эффект. Так, благодаря наличию «Открытого правительства», граждане могут принимать более информированные решения и совершать более обоснованный выбор относительно получения доступа к ресурсам общества и их использования, может поощряться более активная гражданская позиция населения.

Главной целью магистерской диссертации является разработка, обоснование и реализация прототипа программного комплекса для работы с открытыми научными данными. Соответственно, необходимым функционалом системы является поиск, обработка открытых данных и визуализация полученных на их основе результатов.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- провести анализ предметной области концепции открытых данных в научных исследованиях, требований, предъявляемых к форматам их представления и выполнить обзор уже разработанных программных комплексов, выявить их достоинства и недостатки;
- выбрать подходящую платформы и язык программирования для реализации программного комплекса;
- реализовать и исследовать алгоритмы поиска и алгоритмы кластеризации: FOREL, k-средних, кластеризатор на основе самоорганизующейся сети Кохонена;
- проанализировать особенности работы алгоритмов на модельных и реальных данных, сравнить полученные результаты;
- программно реализовать прототип программной системы поиска решений на базе открытых данных;
- оценить скорость и качество работы созданного программного обеспечения в сравнении с уже имеющимися программными продуктами.

Актуальность данной работы связана с тем, что сегодня концепция открытых данных является одним из наиболее развивающихся направлений современной науки во всем мире. Эта область науки определена приоритетной как в США, так и во всех других развитых странах.

1 Аналитическая часть

1.1 Общие сведения

Концепция открытых данных на сегодняшний день не является чем-то новым и уникальным, при том, что формализация данного определения относительно новая. Основная мысль данной концепции лежит в термине «Открытое определение», которое можно в общих словах описать следующим: «Часть информации является открытой только в том случае, если любой желающий может свободно использовать, повторно использовать и распространять данные».

Цели движения открытых данных похожи на другие «открытые» движения, такие как открытое программное обеспечение (open source), открытый контент (open content) и открытый доступ (open access). Рост популярности идеи об открытых данных во второй половине 2000-х годов связан, прежде всего, с запуском правительственных инициатив, таких как Data.gov.

Открытые данные часто ассоциируются с нетекстовыми материалами, такими как карты, геномы, химические компоненты, математические и научные формулы, медицинские данные, данные о биологическом разнообразии. Проблемы чаще всего возникают по той причине, что эти данные могут быть коммерчески ценными или могут быть собраны в некие ценные продукты.

Доступ к данным, как и последующее их использование, контролируется организациями - государственными и частными. Контроль может быть через ограничения, лицензии, копирайт, патенты и требования оплаты для доступа или повторного использования. Сторонники идеи «открытых данных» считают, что подобные ограничения идут против общественного блага и данные должны быть доступны без ограничений или оплаты. Также важно, что данные должны быть доступны без последующих запросов на разрешение, хотя и способы повторного использования, такие как создание продуктов на базе данных, могут контролироваться лицензией.

Государственные данные представляют один из ключевых интересов для общества, и многочисленные некоммерческие организации и отдельные активисты добиваются открытости государственной информации в машиночитаемой форме. Многие национальные правительства в рамках стратегий «открытого государства» создали веб-сайты для распространения части данных, обрабатываемых в секторе государственного управления.

Термин «Открытое определение» дает полное описание требования к открытым данным и содержимому этих данных. Открытые данные являются одним из блоков так называемых открытых сведений. Сами открытые сведения являются квинтэссенцией полезности и применяемости открытых данных.

Ключевые особенности открытых данных следующие:

• доступность. Данные должны быть доступны в целом и иметь разумную цену воспроизводства, опирающуюся на доступ к данным средствами Интернет. Данные должны быть доступны в удобной и изменяемой форме;

• повторное использование и распространение. Данные должны предоставляться на условиях повторного использования и распространения, включая объединение с другими наборами данных. Данные должны быть машиночитаемыми;

• всеобщее участие. Любой вправе использовать, повторно использовать, распространять открытые данные. Не должно быть никаких препятствий, мешающих выполнению этих условий для человека или группы людей. Например, "некоммерческие" ограничения, которые бы препятствовали «коммерческому» использованию, или ограничения пользования для определенных целей (например, только в образовании), не допускаются.

На сегодняшний день существует множество типов открытых данных, потенциально используемых. Некоторые из них:

• культурные, такие как сведения о раскопках и артефактах, музеях, культурных мероприятиях и прочее;

• научные. Данные, публикуемые как часть научных исследований, начиная от астрономии и заканчивая зоологией;

• финансовые, такие как сведения о правительственных счетах (доходы и расходы государства) и сведения о состоянии финансовых рынков;

• метеорологические. Данные, которые бы позволили следить за изменениями в погоде и климате;

• статистические, т.е. данные, полученные с помощью статистических управлений, таких как переписи населения и основные социально-экономические показатели;

• данные об окружающем пространстве. Информация, связанная с окружающей природной средой, например, присутствия и уровень загрязняющих веществ, качество рек и морей;

• данные о транспорте. Всевозможные сведения о маршрутах, расписания движений и т.д.

• Ответ на вопрос об необходимости открытых данных основывается на типе сведений в открытых данных. И все же, основные причины следующие:

• прозрачность. В корректно функционирующем демократическом обществе, граждане должны знать, что делает их правительство. Для этого они должны иметь возможность свободно получать доступ к правительственным данным и информации, а также обмениваться информацией с другими гражданами. Прозрачность должна быть не только в доступе, но также и в совместном использовании. Часто, чтобы понять материал, его необходимо проанализировать и визуализировать, а это в свою

очередь требует от материала быть используемым, повторно используемым и распространяемым;

- свободная социальная и коммерческая ценность. В век цифровых технологий, данные являются ключевым пунктом социальной и коммерческой деятельности. Все, начиная от нахождения вашего местного почтового отделения, для построения поисковой системы требует доступа к данным, большая часть которых создается или предоставляется правительством. Открывая данные, правительство может помочь управлять созданием инновационного бизнеса и услуг, имеющие социальную и коммерческую ценность;
- участие и взаимодействие. Большинство граждан имеют возможность взаимодействовать с правительством лишь эпизодически, а то и вовсе только на выборах, проходящих раз в 4 - 5 лет. Открывая данные, правительство предоставляет гражданам возможность быть более информированными и принимать участие в процессе принятия решений. И это даже больше чем прозрачность: это полный доступ к обществу, не только знание того, что происходит в деятельности власти, но и возможность внести свой вклад в процесс развития общества. Таким образом, можно утверждать, что развитие концепции открытых данных, ее повсеместное внедрение является шагом на пути к процветающему самоконтролируемому демократическому обществу.

Тим Бернес-Ли предложил пятизвездочную шкалу классификации открытых данных:

- одна звезда: данные доступны в Вебе (в любом формате), но подпадают под лицензию открытых данных правительства Великобритании (Open Government Licence for public sector information);
- две звезды: открытые данные доступны в качестве машиночитаемых структурированных данных (например, в виде Excel-таблицы вместо отсканированного изображения таблицы);
- три звезды: открытые данные соответствуют двум звездам плюс представлены в непроприетарном формате (например, в формате CSV вместо Excel-формата);
- четыре звезды: открытые данные соответствуют трем звездам плюс представлены в открытых стандартах консорциума W3C (RDF и SPARQL), предназначенных для идентификации данных;
- пять звезд: открытые данные соответствуют четырем звездам, плюс они связаны с другими данными с учетом контекста их использования.

Открытые данные представляют один из ключевых интересов для общества и многочисленные некоммерческие организации и отдельные активисты добиваются открытости государственной информации в машиночитаемой форме. Многие национальные правительства в рамках стратегий «открытого государства» создали веб-сайты для распространения части данных, обрабатываемых в секторе государственного управления.

Поэтому так важно развивать это движение. Развитие открытых данных позволит гражданам получать актуальную и достоверную информацию об интересующих их аспектах жизни государства и работы его правительства. Открытые данные позволят гражданам активно принимать участие в жизни страны, определяя курс ее развития в направлении, удовлетворяющем требования большинства граждан. Порталы открытых данных как раз являются унифицированным инструментов в достижении этих целей. Именно этот фактор обуславливает необходимость развития технологий ПОД, делая их, по возможности, наиболее универсальными.

1.2 Задача интеллектуального анализа данных

На протяжении всей жизни человек познает большое количество информации. В связи с развитием компьютерных технологий, появлением интернета, различных баз данных на людей обрушились огромные информационные потоки. Для восприятия и понимания этих данных возникло такое направление как Data Mining.

Data Mining (рус. добыча данных, интеллектуальный анализ данных) – технология, используемая для обнаружения в исходных данных заранее неизвестных полезных знаний. Термин был введен Григорием Пятецким-Шапиро в 1989 году.

Data Mining применяется в самых различных сферах человеческой деятельности: для реализации масштабных проектов в бизнесе, промышленности, геологии, медицине и других областях.

Предполагается, что существует некоторая база данных, в которой имеют место «скрытые знания». Под ними понимают какую-то новую информацию, нетривиальные знания, обнаружить которые не так просто. Они обязательно должны иметь практическую ценность для выполнения поставленной цели. Такие знания могут быть представлены в наглядной форме и быть легко интерпретированы.

В основе Data Mining лежит простая схема применения:

- постановка задачи;
- предварительная обработка данных: сбор, подготовка, внедрение;
- построение модели, оценка точности;
- применение разработанной модели.

Data Mining позволяет решить задачи, которые требуют большого количества вычислений и не могут быть решены без помощи компьютера. Высокий уровень развития современных технологий позволяет значительно упростить обработку данных и сократить время работы алгоритмов.

Основные алгоритмы Data Mining

Классификация – один из разделов машинного обучения, который заключается в следующем: имеется множество объектов с известными признаками и принадлежностью к какому-либо из классов, а также набор

объектов с похожими параметрами, класс которых необходимо определить на основе имеющейся информации.

Пример задачи классификации изображен на рисунке 1, объекты в форме треугольника и квадрата принадлежат разным классам, принадлежность круглых объектов необходимо определить.

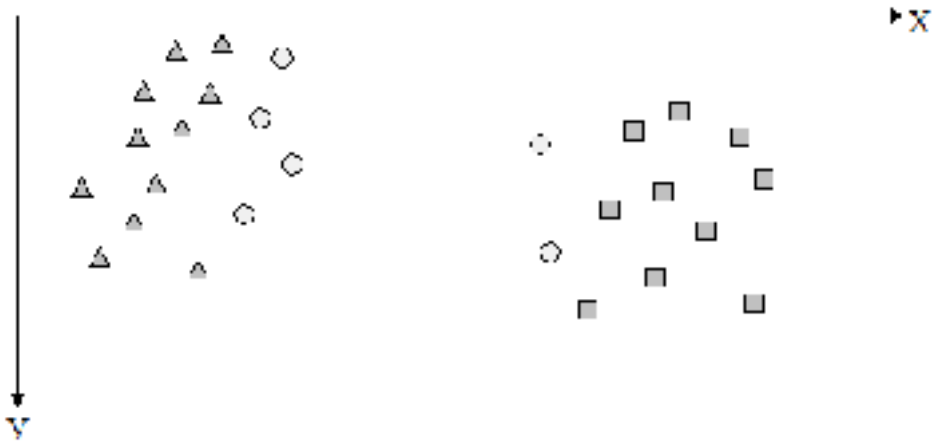

Рисунок 1 – Задача классификации

Кластеризация – один из разделов машинного обучения, где имеется массив объектов с известными признаками, которые необходимо разделить на отдельные непересекающиеся множества, называемые кластерами. На рисунке 2 представлен пример задачи кластеризации и ее решение.

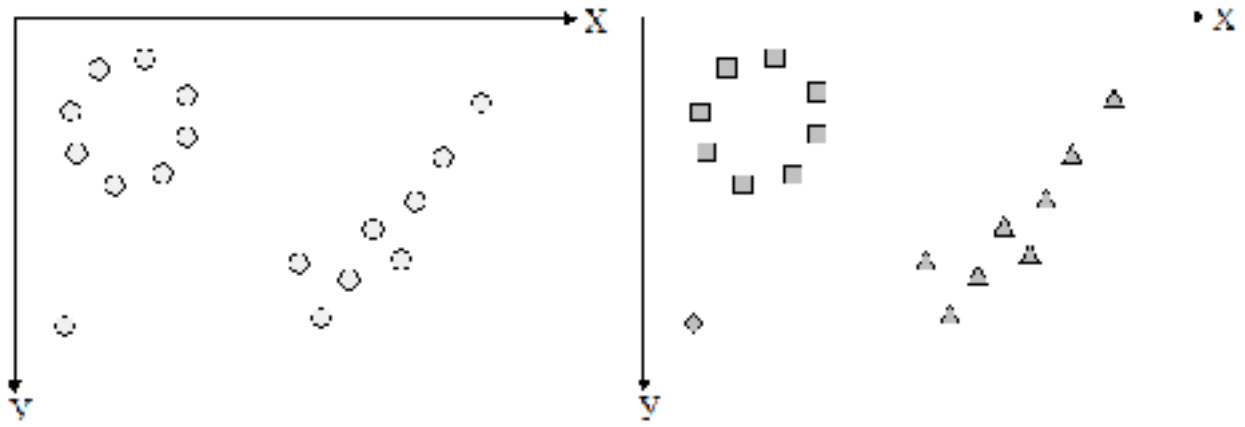

Рисунок 2 – Задача кластеризации

Регрессия – метод для поиска или восстановления зависимости между отдельными переменными для исследования их влияния на зависимую переменную или результат. На рисунке 3 представлен пример задачи регрессии и ее решение.

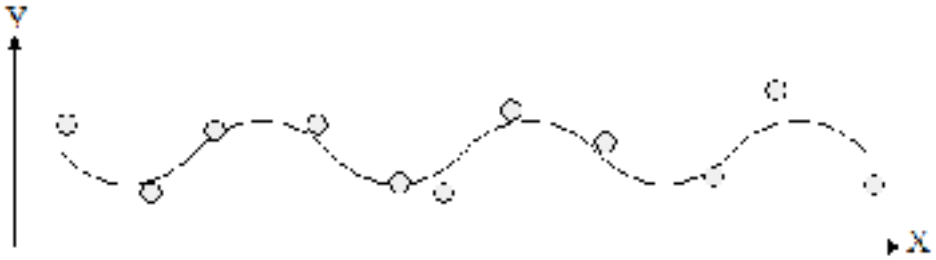

Рисунок 3 – Задача регрессии

Ассоциация – выявление закономерностей между взаимосвязанными событиями. Если из события A следует событие B, то такое правило называется ассоциативным.

Анализ отклонений – выявление данных, заметно отличающихся от общего шаблона, что позволяет сократить объем информации для выявления главных компонент.

Генетические алгоритмы – методы, основанные на эволюции поколений итеративным способом с использованием механизмов, аналогичных естественному отбору в природе. Основными этапами являются:

- создание начальной популяции;
- отбор;
- выбор родителей;
- размножение;
- мутации.

1.3 Задачи классификации и кластеризации

Задача классификации относится к разделу обучения с учителем. Для задач данного типа характерно контролируемое обучение с присутствием обучающего множества с указанной принадлежностью объектов к классам.

Определяемые классы могут быть как пересекающиеся, так и непересекающиеся. Пример таких распределений отображен на рисунке 4.

На сегодняшний день существует большое множество алгоритмов классификации, которые могут давать решения с различной степенью точности. Точной систематизации методов классификации не существует, однако можно выделить основные используемые:

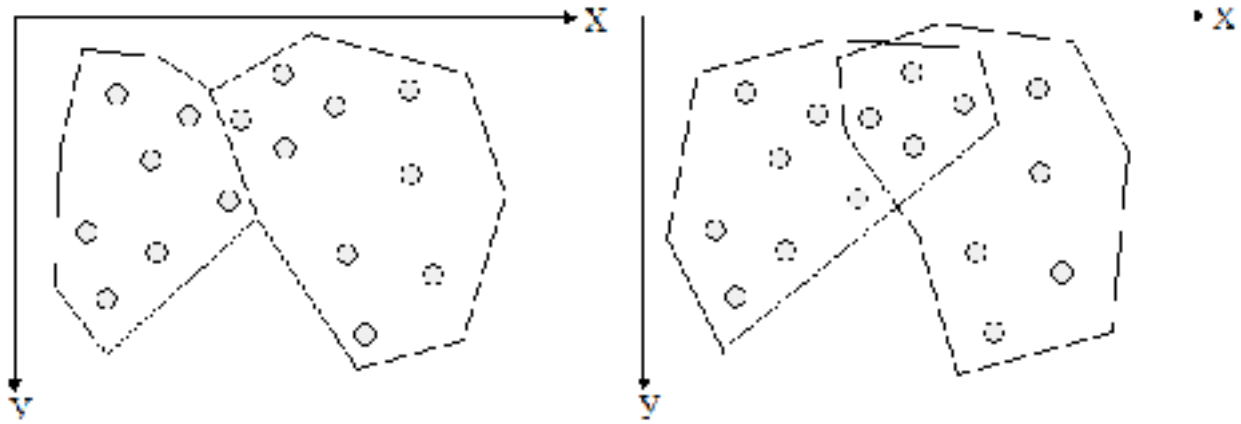

Рисунок 4 – Непересекающиеся и пересекающиеся множества

- деревья решений – средство для поддержки принятия решений. Структура деревьев представляет собой «ветки» и «листья». Первые содержат атрибуты, оказывающие влияние на итоговый результат, а вторые содержат значения целевой функции;
- байесовская классификация – представляет из себя классификатор, основанный на теореме Байеса: если плотности распределения каждого из классов известны, то искомый алгоритм можно записать в одной формулой;
- искусственные нейронные сети – система взаимодействующих между собой нейронов, каждый из которых имеет входную и выходную связь. На вход поступают данные для обучения сети, они влияют на текущее состояние нейрона, что влечет за собой изменение выходных сигналов;
- статистические методы – научные методы анализа статистических данных, полученные в результате наблюдений или экспериментов;
- методы ближайшего соседа – заключаются в нахождении определенного количества наиболее похожих наблюдений каких-либо явлений на основе евклидова расстояния. Найденные объекты используются для определения класса объекта.

Задача кластеризации относится к разделу обучения без учителя. Для задач данного типа характерно неконтролируемое обучение, когда принадлежность объектов к каким-либо классам не указана. Множество объектов необходимо разделить на один или более кластеров данных.

Приведем основные способы решения задачи кластеризации:

- итеративные алгоритмы, основанные на разделении исходной выборки на N классов;
- иерархические алгоритмы. Изначально каждый объект является отдельным кластером. В процессе работы мелкие кластеры объединяются в более крупные;
- методы основанные на концентрации объектов;
- модельные методы, основанные на использовании моделей, наиболее приближенных к искомым данным.

Иногда определенные задачи могут иметь не единственное решение. Это применимо как к задаче классификации, так и кластеризации.

1.4 Обзор решаемых задач

В работе в качестве примера рассматривался вопрос технической диагностики электрорадиоизделий (ЭРИ). Под технической диагностикой понимают определение технического состояния устройства. Техническая диагностика перед эксплуатацией каких-либо механизмов или микросхем имеет большую значимость. Пример ЭРИ представлен на рисунке 5.

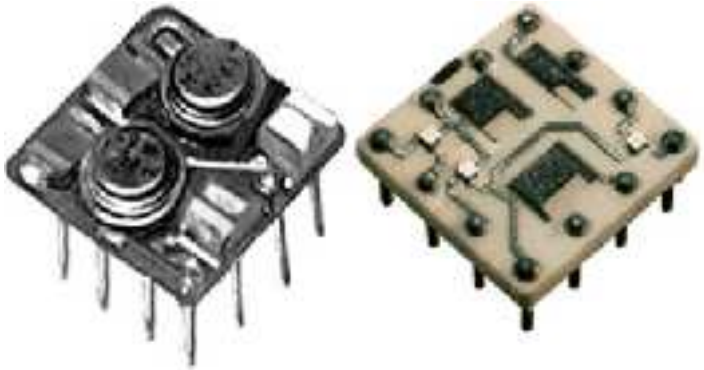

Рисунок 5 – Электрорадиоизделия

На сегодняшний день существует множество предприятий, осуществляющих анализ отказов и выявление скрытых дефектов ЭРИ и полупроводниковых приборов. Проведение контроля качества ЭРИ позволяет обнаружить изделия с отклонениями или обеспечить гарантию их работы.

На территории России имеется несколько испытательных центров ракетно-космической промышленности, занимающихся испытаниями ЭКБ для бортовой аппаратуры. Они взаимодействуют с производителями и поставщиками, а также участвуют при разработке методик входного контроля. Одним таких центров Красноярского края является Железногорское ОАО «ИТЦ – НПО ПМ» (рисунок 6).

Рисунок 6 – Логотип предприятия

Существует три вида испытаний ЭРИ:

- отбраковочные испытания;

- разрушающий физический анализ (РФА);
- диагностический неразрушающий контроль.

Отбраковочные испытания дают подробную информацию о качестве электронной компонентной базы (ЭКБ). Они включают в себя проверку соответствия ЭКБ требованиям документации и позволяют отбраковать потенциально ненадежные изделия.

РФА заключается в проведении физико-технической экспертизы разрушающими методами для выборочных изделий из одной партии. По полученным результатам проводятся дополнительные испытания для партий с отрицательными результатами, и принимаются решения о дальнейшей их эксплуатации.

Диагностический неразрушающий контроль проводится с целью выявления потенциально ненадежной ЭКБ. Его особенностью является то, что отбраковываются изделия, соответствующие нормативно-технической документации, но имеющие признаки скрытых дефектов и низкого качества. Он позволяет выявить дефекты, обнаружить которые невозможно при проведении отбраковочных испытаний или РФА.

Для настройки алгоритмов, определения оптимальных значений параметров в качестве используемой выборки был взят массив данных под названием Ирисы Фишера (от англ. – Fisher’s Iris data set), взятые из репозитория UCI задач машинного обучения, содержащего как реальные, так и модельные наборы данных. Это одна из самых известных баз данных, используемых для распознаваний образов. Массив данных состоит из 150 экземпляров цветков ириса, по 50 экземпляров каждого из трех видов:

- Ирис Щетинистый (от англ. – Iris Setosa);
- Ирис Виргинский (от англ. – Iris Virginica);
- Ирис Разноцветный (от англ. – Iris Versicolor).

Для каждого экземпляра измерялись четыре характеристики (в сантиметрах):

- длина наружной части цветка (от англ. – sepal length);
- ширина наружной части цветка (от англ. – sepal width);
- длина внутренней части цветка (от англ. – petal length);
- ширина внутренней части цветка (от англ. – petal width).

Отображение многомерных данных на плоскости является затруднительным, поэтому на рисунках 7 и 8 изображены распределения объектов выборки относительно двух параметров: ширины и длины внутренней доли цветка, а также ширины и длины наружной доли цветка. Данные проекции позволяют в достаточной степени увидеть плотность распределения объектов и их расположение относительно друг друга.

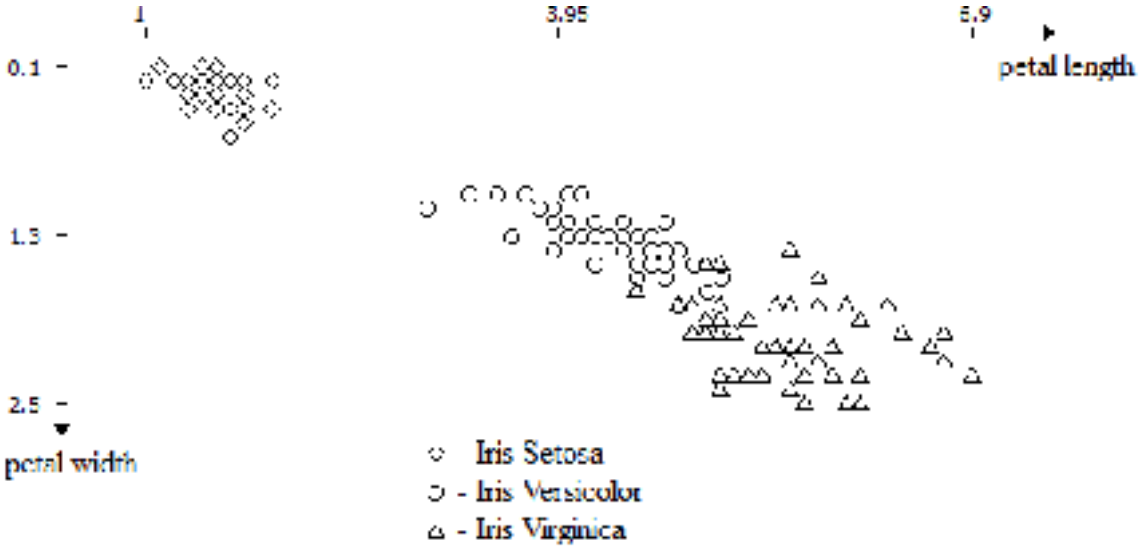

Рисунок 7 – Распределение данных об ирисах в срезе

В описании к базе данных сказано, что ирисы Iris Setosa линейно отделимы от остальных классов ирисов. Это можно увидеть как на рисунке выше по признаку длины или ширины лепестка, так и по любому другому набору из двух признаков. В то же время ирисы оставшихся классов Iris Versicolor и Iris Virginica неотделимы линейно один от другого ни по одному из шести возможных наборов из двух признаков.

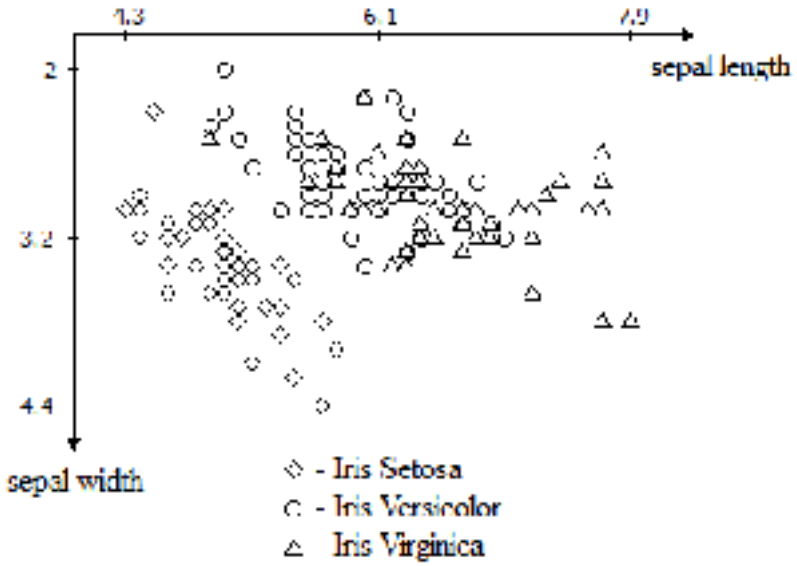

Рисунок 8 – Распределение данных об ирисах в срезе

Для проведения диагностического контроля предприятием «ИТЦ – НПО ПМ» были предоставлены два набора данных, содержащих данные об измерениях ЭРИ. Значения измеряемых параметров для интегральных микросхем 1526ЛЕ5, 1526ЛЕ5ММ представлены в таблице 1 (результаты измерений для токов представлены в миллиамперах, напряжения – в вольтах). Известно, что наборы данных содержат по 3 кластера.

Таблица 1 – Исходные данные для проведения диагностического контроля

№	Контролируемые параметры, единицы измерения	Тесты	Режим измерения
1	Входной ток низкого уровня, I_{IL}(УР), мкА	11-18	Ucc = 11 В U_{IH} = 11 В
2	Входной ток высокого уровня, I_{IH}(УР), мкА	19-26	Ucc = 11 В U_{IL} = 0 В
3	Максимальное выходное напряжение низкого уровня, U_{OL}max(УР), В	27-30	Ucc = 5 В U_{IH} = 3.5 В
4	Минимальное выходное напряжение высокого уровня, U_{OH}min(УР), В	31-38	Ucc = 5 В U_{IL} = 1.5 В
5	Выходной ток низкого уровня, I_{OL}(УР), мА	39-42	Ucc = 5 В Uo = 0.4 В
6	Выходной ток высокого уровня, I_{OH}(УР), мА	43-46	Ucc = 5 В Uo = 2.5 В
7	Ток потребления, Icc(УР), мкА	47-55	Ucc = 11 В U_{IH} = 11 В
8	Прямая ВАХ верхнего диода, U, В	56,59,62,65, 68,71,74,77	I_I = 0.1 мкА
		57,60,63,66, 69,72,75,78	I_I = 100 мкА
		58,61,64,67, 70,73,76,79	I_I = 1 мА
9	Прямая ВАХ нижнего диода, U, В	80,83,86,89, 92,95,98,101	I_I = 0.1 мкА
		81,84,87,90, 93,96,99,102	I_I = 100 мкА
		82,85,88,91, 94,97,100,103	I_I = 1 мА
10	Прямая ВАХ диода цепи питания, U, В	104	I_I = 0.1 мкА
		105	I_I = 100 мкА
		106	I_I = 10 мА
11	Выходная ВАХ нижнего транзистора, U, В	107,110,113,116	Io = 0.1 мА
		108,111,114,117	Io = 0.3 мА
		109,112,115,118	Io = 1 мА
12	Выходная ВАХ верхнего транзистора, U, В	119,122,125,128	Io = 0.1 мА
		120,123,126,129	Io = 0.3 мА
		121,124,127,130	Io = 1 мА

Исходные данные были предоставлены в формате .xls (рисунок 9).

	AC	AD	AE	AF	AG
1	Тест 38	Тест 39	Тест 40	Тест 41	Тест 42
2	4,990E+0	2,440E+0	2,420E+0	2,410E+0	2,390E+0
3	4,990E+0	2,690E+0	2,690E+0	2,670E+0	2,670E+0
4	4,990E+0	3,050E+0	3,050E+0	3,030E+0	3,010E+0
5	4,990E+0	2,200E+0	2,190E+0	2,220E+0	2,210E+0
6	4,990E+0	2,660E+0	2,640E+0	2,620E+0	2,620E+0
7	4,990E+0	2,610E+0	2,600E+0	2,530E+0	2,550E+0
8	4,990E+0	2,460E+0	2,510E+0	2,520E+0	2,520E+0
9	4,990E+0	2,490E+0	2,520E+0	2,540E+0	2,540E+0
10	4,990E+0	2,510E+0	2,510E+0	2,510E+0	2,500E+0
11	4,990E+0	2,130E+0	2,120E+0	2,190E+0	2,150E+0
12	4,990E+0	2,320E+0	2,360E+0	2,340E+0	2,340E+0

Рисунок 9 – Массив данных в программе Excel

Данные представляют собой два файла программы Excel:

- одна партия – 198 объектов, 120 признаков;
- несколько партий – 620 объектов, 120 признаков.

Значения атрибутов были нормализованы минимаксным методом путем сдвига значений каждого компонента по следующей формуле:

$$z = \frac{x - min(x)}{\left[\max(x) - min(x)\right]}, \quad (1)$$

где z – новое значение атрибута,

x – старое значение атрибута,

$min(x)$ – минимальное значение атрибута среди всех значений выборки,

$max(x)$ – максимальное значение атрибута среди всех значений выборки.

На рисунках 10 и 11 представлены распределения данных о 198 объектах в различных срезах.

На рисунках 12 и 13 представлены распределения данных о 620 объектах в различных срезах.

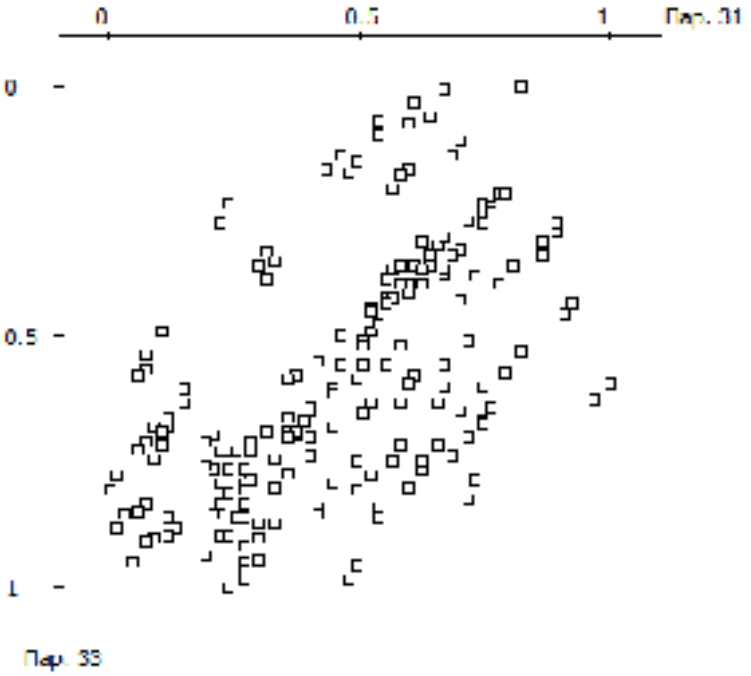

Рисунок 10 – Распределение данных в срезе

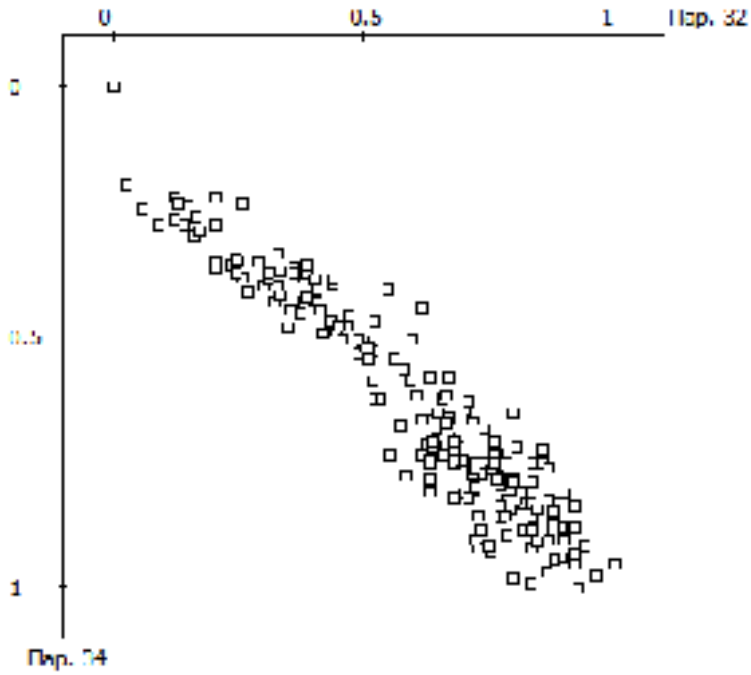

Рисунок 11 – Распределение данных в срезе

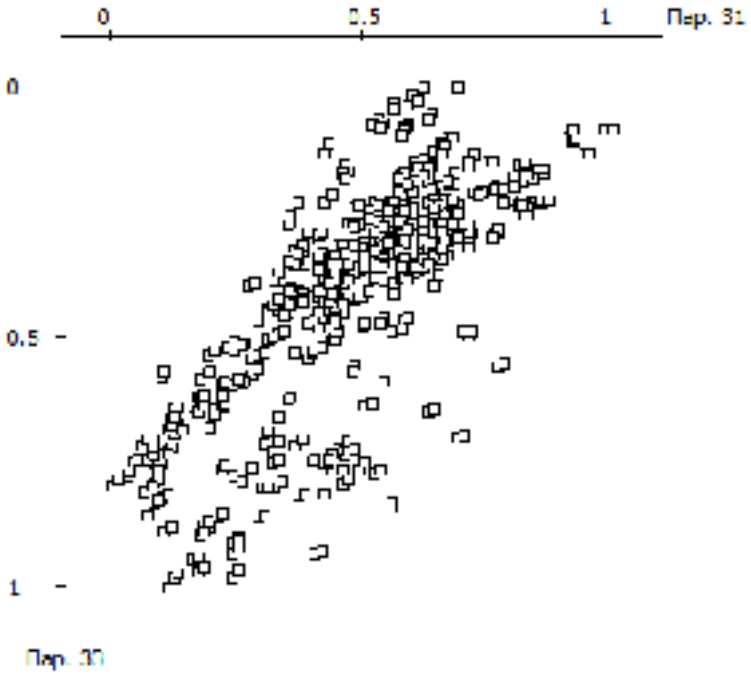

Рисунок 12 – Распределение данных в срезе

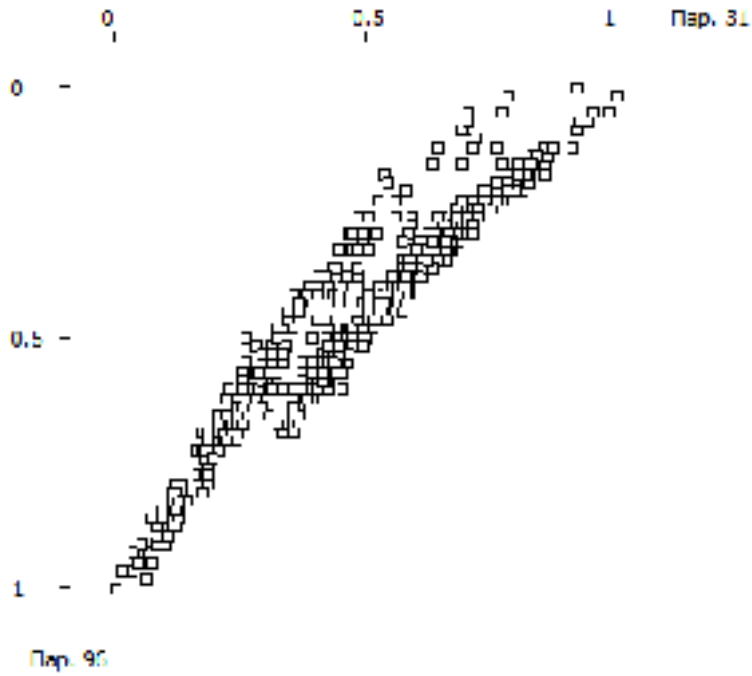

Рисунок 13 – Распределение данных в срезе

За меру сходства двух объектов было использовано евклидово расстояние:

$$\rho\left(\bar{x},\bar{y}\right)=\sqrt{\sum_{i=0}^{n}\left(x_i\text{-}y_i\right)^2}, \tag{2}$$

где $\bar{x}$ – вектор значений параметров объекта x размерности n;

$\bar{}$

y – вектор значений параметров объекта y размерности n;

x_i – значение i-го параметра объекта x;

y_i – значение i-го параметра объекта y;

n – размерность данных.

1.5 Выбор программных средств разработки

Для разработки программной системы было необходимо выбрать такой язык программирования, который поддерживает возможности объектно-ориентированного программирования, типизацию данных, может использовать библиотеки для работы с графикой и мультимедиа.

На основе необходимых требований был выбран язык C++, поскольку является наиболее удобным и гибким в использовании, а также он был основным языком программирования в процессе прохождения обучения.

Стоит также отметить основные преимущества данного языка:

- возможность написания программ практически любого назначения: утилиты, драйверы, библиотеки и т. д.;
- поддержка различных стилей и технологий программирования: объектно-ориентированное программирование, процедурное программирование, метапрограммирование;
- эффективность: язык предоставляет программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы;
- кроссплатформенность: стандарт языка накладывает минимальные требования на ЭВМ для запуска скомпилированных программ.

Использование графических библиотек

Для отображения массива данных на плоскости было решено использовать графическую библиотеку Open Graphics Library (OpenGL).

OpenGL является одним из самых популярных прикладных программных интерфейсов (API – Application Programming Interface) для разработки приложений в области двумерной и трехмерной графики.

Описать возможности OpenGL можно через функции его библиотеки. Все функции можно разделить на пять категорий:

- функции описания примитивов определяют объекты нижнего уровня иерархии (примитивы), которые способна отображать графическая подсистема. В OpenGL в качестве примитивов выступают точки, линии, многоугольники и т.д.;
- функции описания источников света служат для описания положения и параметров источников света, расположенных в трехмерной сцене;
- функции задания атрибутов. С помощью задания атрибутов программист определяет, как будут выглядеть на экране отображаемые объекты. Другими словами, если с помощью примитивов определяется, что появится на экране, то атрибуты

определяют способ вывода на экран. В качестве атрибутов OpenGL позволяет задавать цвет, характеристики материала, текстуры, параметры освещения;

- функции визуализации позволяет задать положение наблюдателя в виртуальном пространстве, параметры объектива камеры. Зная эти параметры, система сможет не только правильно построить изображение, но и отсечь объекты, оказавшиеся вне поля зрения;
- набор функций геометрических преобразований позволяют программисту выполнять различные преобразования объектов – поворот, перенос, масштабирование.

При этом OpenGL может выполнять дополнительные операции, такие как использование сплайнов для построения линий и поверхностей, удаление невидимых фрагментов изображений, работа с изображениями на уровне пикселей и т.д.

Интегрированная среда разработки

Одной из основных причин выбора данной IDE является наличие опыта работы с ней: она неоднократно использовалась как при выполнении курсовых, так и лабораторных работ.

На сегодняшний день существует множество различных сред программирования, предоставляющие различный функционал для разработки приложений. В качестве инструментария разработки программного обеспечения была использована программа Microsoft Visual C++, разработанная компанией Microsoft.

Visual C++ поддерживает перечень приложений как на Managed C++ и C++/CLI, так и на обычном C++, и тем самым позволяет генерировать код как для платформы .NET Framework, так и для исполнения в среде «чистой» Windows. В этом отношении Visual C++ является уникальным среди других языковых средств, предоставляемых средой Visual Studio, поскольку ни Visual Basic .NET, ни Visual J# не способны генерировать код для чистого Win32, в отличие от предыдущих версий (Visual Basic и Visual J++ соответственно).

1.6 Выводы

В данном разделе были определены: выявлены ключевые особенности открытых данных, определены их типы, обозначена необходимость использования открытых данных, понятие Data Miming, принцип работы, основные аспекты задач классификации, кластеризации и технической диагностики, а также способы их решения. Были рассмотрены задачи, решаемые в ходе выполнения работы: срезы данных, описание атрибутов. Был обоснован выбор программных средств разработки.

2 Исследование и построение решения

2.1 Постановка задачи

Главной целью магистерской диссертации является разработка, обоснование и реализация прототипа программного комплекса для работы с открытыми научными данными. Соответственно, необходимым функционалом системы является поиск, обработка открытых данных и визуализация полученных на их основе результатов.

В соответствии с целью были поставлены следующие задачи:

- провести анализ предметной области концепции открытых данных в научных исследованиях, требований, предъявляемых к форматам их представления и выполнить обзор уже разработанных программных комплексов, выявить их достоинства и недостатки;
- выбрать подходящую платформы и язык программирования для реализации программного комплекса;
- реализовать и исследовать алгоритмы поиска и алгоритмы кластеризации: FOREL, k-средних, кластеризатор на основе самоорганизующейся сети Кохонена;
- проанализировать особенности работы алгоритмов на модельных и реальных данных, сравнить полученные результаты;
- программно реализовать прототип программной системы поиска решений на базе открытых данных;
- оценить скорость и качество работы созданного программного обеспечения в сравнении с уже имеющимися программными продуктами.

2.2 Требования к системе

Чтобы раскрыть весь потенциал открытых данных, они должны быть доступны в максимально удобном для использования виде. Чтобы сразу начать ими пользоваться, а не тратить время на приведение их к корректному виду.

Приложение «Система поиска решений на базе открытых данных» должно иметь следующие характеристики:

- язык разработки программного обеспечения: C++ с использованием графической библиотеки OpenGL;
- компиляция осуществляется в Microsoft Visual C++ Studio 2012;
- главное окно программы должно предоставлять удобный пользовательский интерфейс и быть интуитивно понятным;
- для полноценной работы необходим компьютер под управлением операционной системы Windows;
- визуализация. Возможность визуализировать данные так, как хочет автор и пользователи, а не так как это сделает система;

- стандартизованность. Возможность задать структуру данных, отклонение от которой выдаст ошибку и не позволит загрузить данные;
- наличие API и возможность интеграции с различными источниками данных.

2.3 Структурная схема программной системы

Перед составлением программной реализации системы необходимо определить ее составные части, выбрать необходимые библиотеки для реализации основных функций. В процессе выполнения выпускной квалификационной работы была разработана следующая структурная схема программной системы, состоящая из массива данных, обработчика, основной программы, блока настройки параметров и формирования результатов эксперимента. Структура автоматизированной системы представлена на рисунке 14.

Рисунок 14 – Структурная схема программной системы

Массивы данных

Рациональнее всего использовать файлы формата .txt, так как он наиболее удобен при чтении и записи данных, поэтому данный блок состоит из следующих наборов задач, сохраненных в текстовых файлах:

- модельные задачи классификации;
- реальные задачи классификации;
- модельные задачи кластеризации;
- реальные задачи кластеризации.

Файл состоит из двух блоков, отделенных друг от друга дополнительным отступом строки:

- первый блок: описание переменных;
- второй блок: таблица с данными.

Для описания каких-либо переменных используется следующая конструкция:

[идентификатор] = [значение]

Для идентификатора имеется ряд условий:

- имена переменных должны быть уникальны;
- имя переменной должно начинаться с буквы;
- остальными символами могут быть буквы (строчные или прописные), цифры и символ подчеркивания. Пробел, точка, запятая и другие специальные знаки – недопустимы;
- длина имени не должна превышать 255 символов;
- имя переменной не должно совпадать с ключевыми символами языка программирования (например: if, else, for, while, do и другими).

Значение может быть как текстовым, так и числовым. Строка записывается в двойных кавычках, число – цифрами. Если число является десятичной дробью, целая часть отделяется от дробной точкой.

Каждая строчка первого блока содержит описание отдельной переменной.

Каждая строчка второго блока текстового файла содержит в себе информацию об отдельных наблюдениях, разделенных запятыми. Количество рассматриваемых атрибутов не ограничено и должно быть одинаковым для всех строчек файла.

Пример файла представлен на рисунке 15.

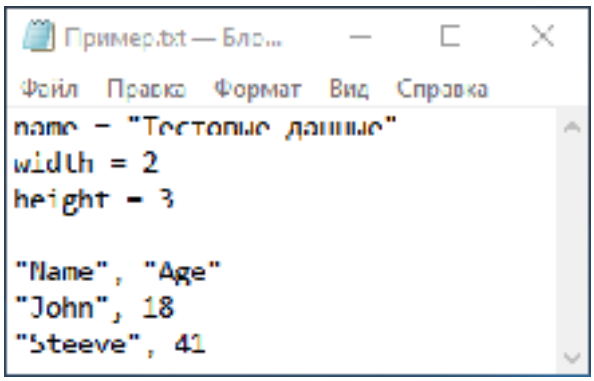

Рисунок 15 – Входной файл

Обработчик

В связи с большим разбросом значений, данные, пересылаемые в программу, должны быть предварительно обработаны:

- нормализация параметров, чтобы их значения находились в промежутке от 0 до 1;
- приведение бинарных переменных к значению 0 или 1.

Блок настройки параметров.

Поскольку в разрабатываемой программе будет реализовано несколько алгоритмов, каждый из которых обладает своими особенностями, необходимо добавить отдельное окно, которое позволит подобрать

оптимальные значения коэффициентов и параметров для правильной работы алгоритмов.

Основная программа и формирование результатов эксперимента

Блок формирования результатов позволяет просмотреть информацию об исходной выборке: принадлежность данных к классам, значение отдельных параметров. Также имеется возможность просмотра гистограммы, отображающей плотность распределения объектов, графика, отражающей распределение данных в каком-либо срезе по двум переменным.

2.4 Анализ существующих систем

Пользователям необходимо самостоятельно искать нужные данные, при этом, какого-либо универсального подхода в публикации этих данных нет. Для решения описанной выше проблемы, а также ряда других, таких как актуализация данных, единый реестр и интерфейс и прочее, применяют порталы открытых данных.

Портал открытых данных (ПОД) - Web-ресурс в задачи которого входит хранение, версионирования, катологизация, актуализация, сбор и предоставление доступа к открытым данным.

Ниже представлены некоторые из систем, активно использующихся в России.

www.data.gov.ru - портал открытых данных Российской Федерации. Здесь сосредотачиваются наиболее актуальные сведения об открытых данных федеральных органов власти, органов региональной власти и иных организаций, размещаются документированные наборы данных, ссылки и метаданные опубликованных наборов данных, информация о созданных на основе открытых данных программных продуктах и информационных услугах. Здесь же публикуются нормативные правовые акты, регламентирующие деятельность государственных органов по раскрытию данных, методические и публицистические ресурсы. Здесь же реализованы коммуникационные интерфейсы для взаимодействия с организациями, выступающими в качестве владельцев социально-значимых данных.

www.opendatasoft.com - одна из лидирующих платформ для публикации и обмена открытыми данными. Это единственная в своем роде платформа, способная обрабатывать потоки данных в режиме реального времени.

www.opensciencedatacloud.org - с помощью Open Science Data Cloud научному сообществу предоставляются ресурсы для хранения, совместного использования и анализа научных наборов данных, измеряемых в терабайтах и петабайтах. Наборы данных доступны в различных форматах.

www.opendata.mkrf.ru – официальный сайт министерства культуры Российской Федерации. Министерство культуры Российской Федерации (Минкультуры России) - федеральный орган исполнительной власти, осуществляющий функции по выработке и реализации государственной политики и нормативно-правовому регулированию в сфере культуры,

искусства, культурного наследия (в том числе археологического наследия), кинематографии, архивного дела, туристской деятельности, авторского права и смежных прав и функции по управлению государственным имуществом и оказанию государственных услуг в сфере культуры и кинематографии, а также по охране культурного наследия, авторского права и смежных прав, по контролю и надзору в указанной сфере деятельности. Деятельность Минкультуры России регулируется нормативно-правовыми актами, нормативными документами, постановлениями Правительства Российской Федерации, административными регламентами, приказами и распоряжениями, полный перечень которых доступен на официальном сайте Министерства Культуры Российской Федерации, расположенном в сети Интернет по адресу *http://mkrf.ru/*. Министерства культуры Российской Федерации осуществляет координацию деятельности подведомственных учреждений культуры и Федерального агентства по туризму (Ростуризм).

Ввиду отсутствия рекомендованной методики проведения анализа находящейся информации в распоряжении ФОИВа в соответствии с полномочиями, а также содержащейся в его информационных ресурсах, реестрах и регистрах, Минкультуры России была выбрана следующая методика анализа:

- анализ информации, находящейся в распоряжении Минкультуры России в соответствии с полномочиями министерства;
- анализ реестров и регистров, образующихся в процессе оказания государственных услуг и осуществления функций Минкультуры России;
- анализ сведений, находящихся в информационных ресурсах Минкультуры России;
- подготовка сводного перечня выявленных наборов данных.

Выявлены следующие реестры, регистры и перечни:

- фонд (реестр) перемещенных в Союз ССР в результате Второй мировой войны и находящихся на территории Российской Федерации культурных ценностей, предназначенных для обмена на культурные ценности Российской Федерации, разграбленные бывшими неприятельскими государствами в период Второй мировой войны и находящиеся на территории государства, не востребовавшего свои культурные ценности;
- перечень перемещенных культурных ценностей, не подлежащих передаче иностранным государствам, международным организациям и (или) вывозу из Российской Федерации, а также правила обеспечения режима их хранения;
- реестр книжных памятников;
- реестр объектов культурного наследия (памятников истории и культуры) народов Российской Федерации;
- государственный свод особо ценных объектов культурного наследия народов Российской Федерации;

- реестр социально-ориентированных некоммерческих организаций - получателей государственной поддержки (в рамках компетенции Минкультуры России);
- государственный регистр фильмов;
- государственный каталог Российской Федерации;
- регистр культурные ценности, ввозимые и временно ввозимые на территорию Российской Федерации;
- регистр фактов пропажи, утраты, хищения культурных ценностей, организует и обеспечивает оповещение государственных органов и общественности в Российской Федерации и за ее пределами об этих фактах.

www.minfin.ru/opendata/ - На официальном сайте Министерства финансов Российской Федерации размещается информация в соответствии с порядком, определенным приказом Минфина России от 28.09.2016 № 397 «Об официальном сайте Минфина России в информационно-телекоммуникационной сети «Интернет».

Главная страница Сайта содержит список основных рубрик, ссылки на официальный сайт Президента, интернет портал Правительства Российской Федерации, новостную ленту официальной информации, а также контактную информацию и форму для обращения граждан.

Основные рубрики Сайта содержат сведения о структуре и правовых основах деятельности Минфина России, федеральных службах и подведомственных организациях, входящих в состав Минфина России, мероприятиях и высказываниях руководства, нормативно-правовых документах Минфина России, публикуемых на Сайте.

Вниманию пользователей Сайта представлена информация по федеральному бюджету, бюджетной политике, электронному бюджету, Резервному фонду и Фонду национального благосостояния, международным финансовым и налоговым отношениям, государственному долгу, бухгалтерскому учету и аудиту, государственным услугам и другим видам деятельности Минфина России.

Рубрика «Открытое министерство» содержит ключевые показатели эффективности деятельности Минфина России, приоритетные направления, а также ссылки на Единый портал бюджетной системы Российской Федерации, Портал государственных программ, официальный сайт для размещения информации о государственных (муниципальных) учреждениях, информационно-аналитический раздел Сайта, а также ссылки к информационным ресурсам органов государственной власти, сайтам федеральных служб, порталу правовой информации и сайтам неправительственных организаций.

В таблице 2 приведено сравнение рассматриваемых систем.

Таблица 2 - сравнение рассматриваемых систем

Критерий/Портал	Кол-во наборов данных	Наличие поиска	Разделение на категории	Качество документации
data.gov.ru	13653	Есть	Есть	Нет
opendatasoft.com	9175	Есть	Нет	Нет
opensciencedatacloud.org	Неизвестно	Нет	Нет	Высокое
opendata.mkrf.ru	68	Есть	Есть	Среднее
minfin.ru	137	Есть	Есть	Среднее

На сегодняшний день, все наборы данных предоставляются пользователям в виде простых таблиц содержащих текстовую информацию. Внешний вид наборов остается примитивным и зачастую не соответствует их реальному виду. Среди рассмотренных ПОД так же не обнаружилось какой-либо сущности, которая бы описывала структуры наборов. Данный нюанс крайне негативно влияет на усваиваемость информации пользователем, а возможность использования данных наборов сторонними системами затруднительна из-за того, что разработчики данных систем могут не знать о назначении отдельных элементов данных.

Ко всему прочему, стоит упомянуть о самой информации размещаемой на порталах. В первую очередь, вся информация должна быть машиночитаемой. Таким образом все типы данных предоставляемой информации являются простыми, такими как текст, числа, URL, и прочие, которые явно могут быть представлены в виде текста. Большинство ПОД расширяют этот перечень тем, что предоставляет карту с геометками, в случае, если набор содержит геолокационную информацию. Определение «машиночитаемый» подразумевает тот факт, что вычислительная машина может обрабатывать эти данные. Современный уровень развития вычислительной техники и программного обеспечения достаточно высок, чтобы расширить список машиночитаемых форматов до мультимедийной информации, такой как изображения, аудио- и видеофайлы.

2.5 Обзор методов классификации и кластеризации

Метод k ближайших соседей

Метод k ближайших соседей (англ. k-nearest neighbors algorithm, k-NN) – простейший метрический алгоритм для классификации объектов. Предполагается, что существует определенный набор объектов с уже имеющейся классификацией. Рассмотрим принцип работы алгоритма.

Классифицируемый объект X относится к тому классу У, к которому принадлежит большинство из k его ближайших соседей.

Входные данные для алгоритма следующие:

- классифицируемая выборка;
- классифицированная выборка;

- параметр k – натуральное число, определяющее максимальное количество соседей, используемых для классификации нового объекта;
- параметр R – радиус поиска соседей.

Приведем пошаговое описание алгоритма:

1) выбрать случайный неклассифицированный объект выборки X;
2) найти k ближайших классифицированных объектов для X, находящихся на расстоянии меньше R от него;
3) найти наиболее распространенный класс У среди найденных объектов;
4) объект X добавить в класс У и исключить его из дальнейшего рассмотрения. Переход к пункту а.

Алгоритм завершает свою работу, когда все объекты выборки являются классифицированными.

Исследование алгоритма на модельных данных

Проверим работу алгоритма на следующем примере (рисунок 16). Объект, обозначенный квадратом, необходимо классифицировать на основе имеющихся данных. При значении k в промежутке [1; 5] объект будет классифицирован как треугольник, поскольку большинство его ближайших соседей это треугольники, а при значении большем шести будет определен как круг.

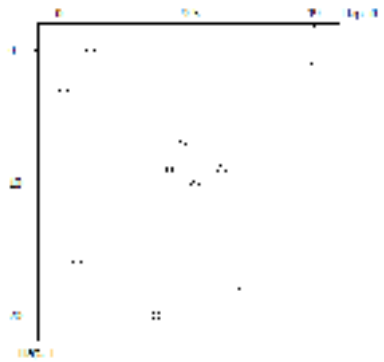

Рисунок 16 – Классифицируемый объект и обучающая выборка

В данном случае имеет смысл ограничить радиус поиска ближайших соседей, если тестовый квадрат должен быть классифицирован как треугольник.

Выводы

Достоинство – простой в реализации алгоритм.

Недостатки:

- высокая вычислительная трудоемкость;
- подходит только для задач небольшой размерности по количеству классов и переменных.

Алгоритм FRiS-STOLP

Алгоритм FRiS-СТОЛП используется для нахождения эталонных объектов в каждом имеющемся классе объектов на основе FRiS-функции. Данная функция позволяет не только определить как далеко или близко объекты расположены в пространстве или похожи ли они друг на друга, но и дать количественную оценку ответа на вопрос “по сравнению с чем?”. Данный способ определения сходства позволяет учитывать большее количество факторов при классифицировании объектов.

Входные данные для алгоритма следующие:

- классифицируемая выборка;
- классифицированная выборка.

Приведем пошаговое описание алгоритма:

1) для каждого объекта класса А относительно всех остальных объектов этого же класса находим среднее арифметическое значение оценки обороноспособности относительно одного ближайшего объекта из других классов. Данная функция сходства имеет следующий вид:

$$C'_{ij/b} = \left(R_2 - R_1 \right) / \left(R_2 + R_1 \right), \quad (3)$$

где i – объект класса А,

j – другой объект класса А ($i \neq j$),

b – ближайший объект класса В для i,

R_1 – расстояние между объектами i и j,

R_2 – расстояние между объектами i и b;

2) для каждого объекта класса А находим среднее арифметическое значение оценки толерантности относительно одного ближайшего объекта из других классов относительно всех остальных объектов этого же класса В. Данная функция сходства имеет следующий вид:

$$C''_{qs/i} = \left(R_2 - R_1 \right) / \left(R_2 + R_1 \right), \quad (4)$$

где i – объект класса А,

s – ближайший объект класса В для i,

q – другой объект класса В ($s \neq q$),

R_1 – расстояние между объектами q и s,

R_2 – расстояние между объектами q и i;

3) для каждого объекта класса А находим общую оценку эффективности по следующей формуле:

$$F_i = \left(C'_i + C''_i \right) / 2, \quad (5)$$

где i – объект класса А,

C' – среднее значение обороноспособности объекта i,

C'' – среднее значение толерантности объекта i;

4) в качестве столпа для класса А выбираем объект, который имеет наибольшую величину F_i.

5) Повторяем шаги а-г для всех остальных классов.

Поиск значений эффективности позволяет наиболее точно определить столпы классов и предотвратить возникновения ошибок: чем выше обороноспособность объекта, тем меньше вероятность появления ошибок первого рода, чем выше толерантность объекта, тем меньше вероятность появления ошибок второго рода.

Первые столпы были выбраны в тех условиях, когда им противостояли ближайшие объекты из других классов. Теперь же в роли этих объектов могут выступать уже известные столпы. Пересчет столпов достаточно провести один или два раза в зависимости от количества имеющихся объектов, впоследствии список столпов практически не меняется.

Определение принадлежности объекта к какому-либо классу можно определять двумя способами:

- объект относится к тому классу, к которому принадлежит ближайший для него столп;
- с учетом расстояния до ближайшего из конкурирующих образов, используя FRiS-функцию.

Исследование алгоритма на модельных данных

На рисунке 17 представлены два класса, а также два столпа, обозначенные крестами. Исследуемый объект в форме квадрата расположен несколько ближе к эталону класса формы круга, однако при использовании FRiS-функции он будет определен как треугольник, поскольку, судя по структуре классов, он является типичным представителем треугольных объектов.

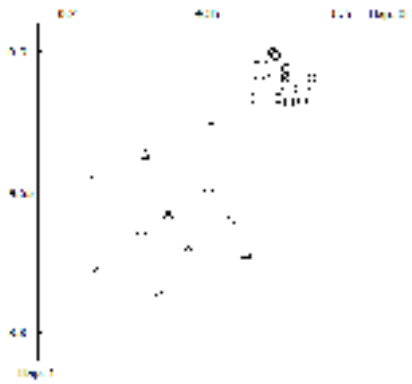

Рисунок 17 – Исходная выборка, столпы и неизвестный объект

Выводы

Достоинства:

- простой процесс распознавания с опорой;
- возможность избавления от ошибок и выбросов.

Недостатки:

- высокая вычислительная трудоемкость;
- подходит только для задач небольшой размерности по количеству классов и переменных.

Алгоритм FOREL

FOREL (от англ. Formal Element – Формальный Элемент) — алгоритм кластеризации, основанный на идее объединения в один кластер объектов в областях их наибольшего сгущения. Рассмотрим принцип работы алгоритма.

На каждом шаге строится гиперсфера минимального радиуса, которая охватывает все объекты с центром в случайном объекте выборки. Если бы был необходим один кластер, то он был бы представлен именно этой начальной сферой, как отображено на рисунке 18.

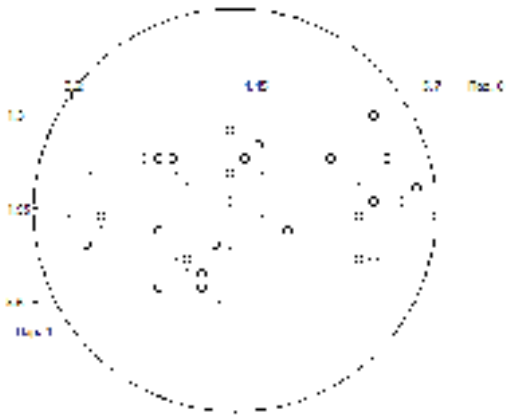

Рисунок 18 – Кластер, содержащий все объекты выборки

Для получения большего числа кластеров необходимо постепенно уменьшать радиус сферы и двигать ее в сторону локального сгущения объектов выборки. Перенос центра сферы с уменьшением радиуса продолжается до тех пор, пока она не остановится, то есть пока на очередном шаге координаты центра не останутся теми же. Этот процесс показан на рисунке 19.

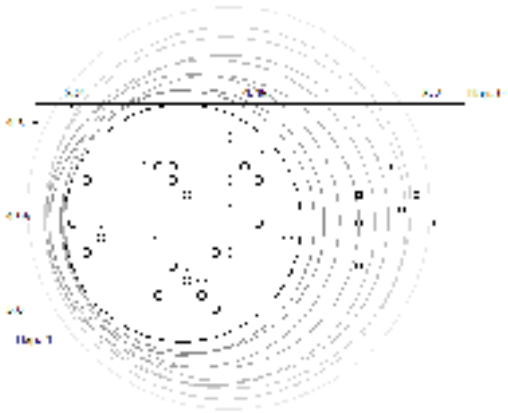

Рисунок 19 – Уменьшение радиуса и перемещение сферы

Полученные объекты внутри сферы помечаются как кластеризованные и убираются из выборки. Этот процесс повторяется до тех пор, пока вся выборка не будет кластеризована.

Входные данные для алгоритма следующие:

- кластеризуемая выборка;
- параметр α – коэффициент в пределах (0; 1) уменьшения радиуса сферы. При меньшем значении получится меньшее количество кластеров.

Приведем пошаговое описание алгоритма:

1) Выбрать случайный объект выборки;
2) Провести сферу минимального радиуса с центром в данном объекте, охватывающую все объекты выборки;
3) Уменьшить радиус сферы по формуле

$$R=R*\alpha, \quad (6)$$

где R – радиус сферы;

α – коэффициент уменьшения радиуса сферы;

4) Пометить объекты, находящиеся внутри сферы;
5) Передвинуть центр сферы в центр масс помеченных объектов;
6) Если центр сферы изменился, то переходим к пункту в, иначе к пункту ж;
7) Помеченные объекты добавить в один кластер и исключить их из дальнейшего рассмотрения. Переход к пункту а.

Алгоритм завершает свою работу, когда все объекты выборки являются кластеризованными.

Исследование алгоритма на модельных данных

На рисунках 20 и 21 можно заметить, что результат кластеризации сильно зависит от выбора объекта, с которого начинается построение начальной сферы.

Во избежание возникновения таких случаев в программной реализации алгоритма в качестве начального объекта выбирался тот, который наиболее близко расположен к центру масс некластеризованных объектов, что позволяет добиться одинаковой кластеризации при повторных запусках алгоритма.

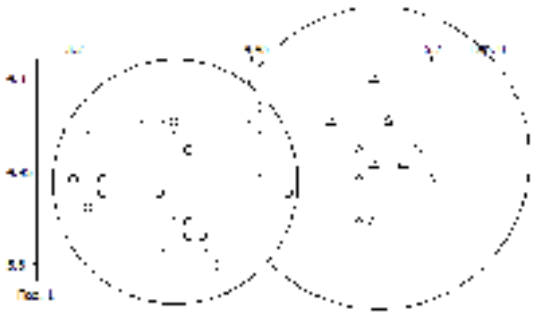

Рисунок 20 – Результат кластеризации

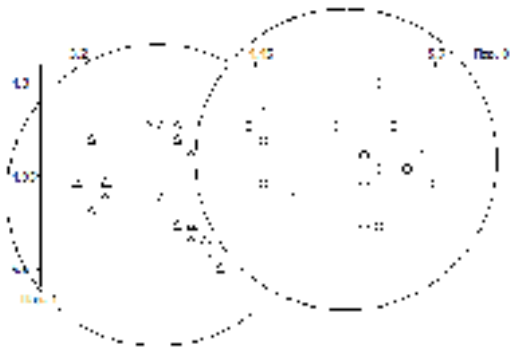

Рисунок 21 – Результат кластеризации

Рассмотрим работу алгоритма на наборе объектов, отображенных на рисунке 22. Визуально оценив расположение объектов, можно распределить их по трем кластерам.

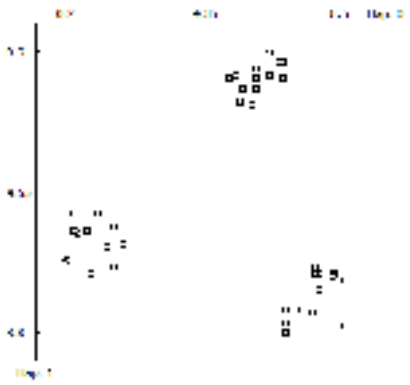

Рисунок 22 – Кластеризуемая выборка

Однако, варьируя коэффициент α , можно получить различное количество кластеров. На рисунке 23 показан график зависимости коэффициента α и количества кластеров, полученных в результате работы алгоритма.

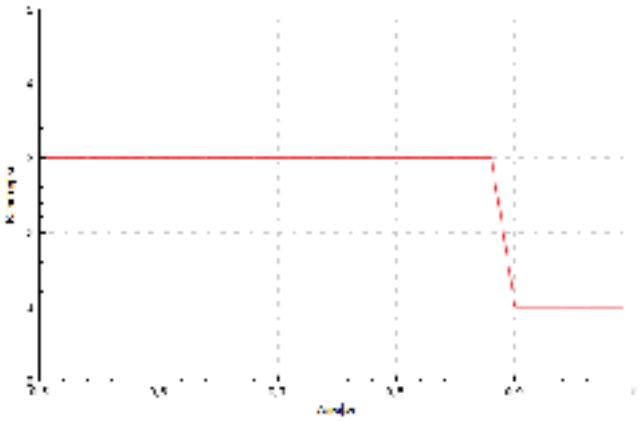

Рисунок 23 – График зависимости α и количества кластеров

Результаты кластеризации, соответствующие данному графику, с использованием разных параметров можно увидеть на рисунках 24-26.

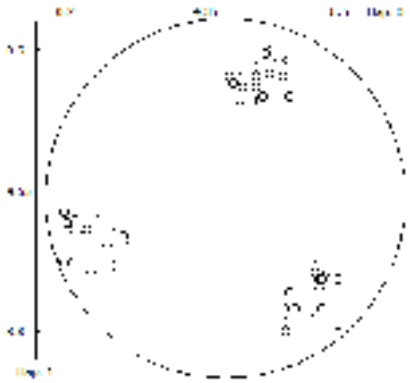

Рисунок 24 – Результат кластеризации при α=0.9

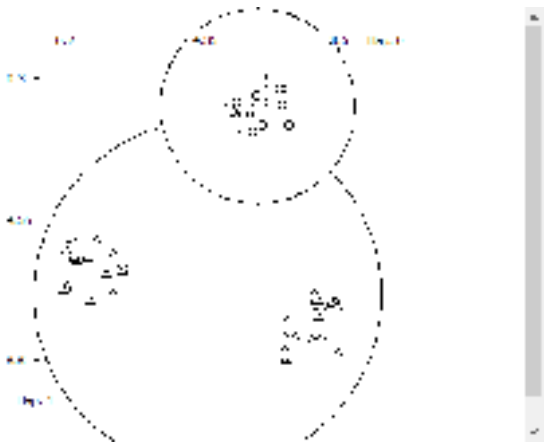

Рисунок 25 – Результат кластеризации при α=0.89

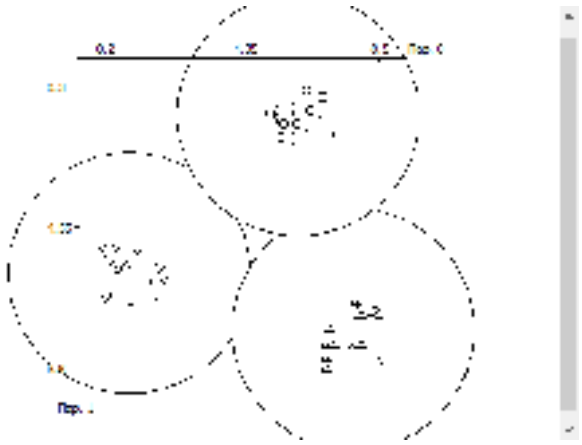

Рисунок 26 – Результат кластеризации при α=0.7

Проверим работу алгоритма, добавив к исходной выборке еще один отдельный объект, расположенный между кластерами. Как видно из графика на рисунке 27, желаемый результат с 4 кластерами получается при использовании параметра α в промежутке от 0.64 до 0.77.

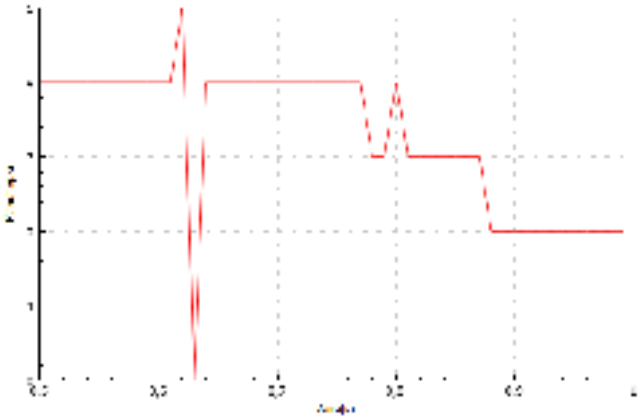

Рисунок 27 – График зависимости α и количества кластеров

Несмотря на то, что в результате работы программы получится 4 кластера, их состав может не соответствовать ожиданиям (рисунок 28).

Такое расположение кластеров вызвано тем, что некоторая часть объектов кластера размещена довольно близко друг к другу, а добавленный новый объект немного притягивает центр сферы к себе. Чтобы предотвратить возникновение таких результатов, необходимо ввести дополнительные модификации алгоритма или использовать другое значение α(рисунок 29).

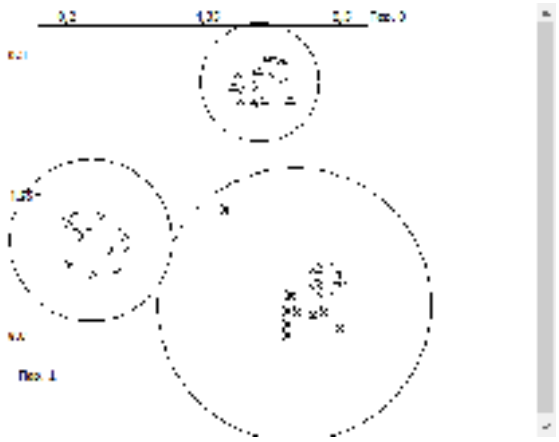

Рисунок 28 – Результат кластеризации при α=0.77

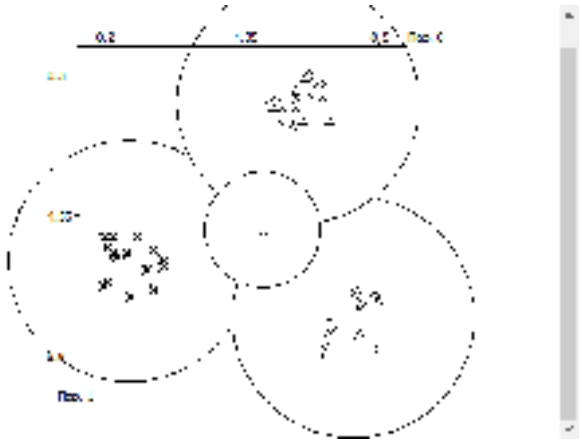

Рисунок 29 – Результат кластеризации при α=0.7

Также на графике имеется такое значение α, при котором количество кластеров равно нулю. Причину возникновения этого случая можно рассмотреть на рисунке 30. При уменьшении радиуса сферы на каждой итерации состав ее внутренних объектов постоянно меняется, вследствие чего сфера уменьшается до такого размера, что перестает содержать в себе какие-либо элементы. Поэтому имеет смысл взять одно из предыдущих состояний сферы либо увеличивать параметр α в процессе работы алгоритма.

Выводы

Достоинства:

- высокое сходство объектов кластеров с центральным элементом;
- сходимость алгоритма за конечное число шагов.

Недостатки:

- произвольное по количеству разбиение на классы;
- плохая применимость алгоритма при близко расположенных кластерах;
- полученные кластеры имеют сферическую форму;

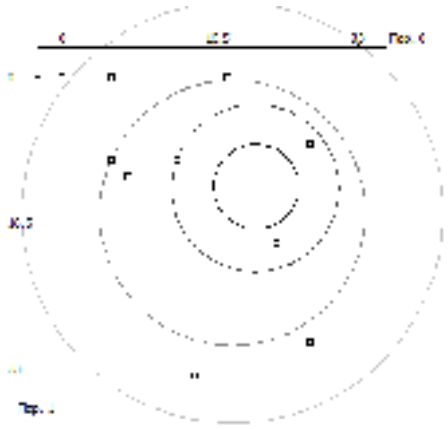

Рисунок 30 – Ошибка при кластеризации

- неустойчивость алгоритма (зависимость от выбора начального объекта);
- необходимость предварительной нормализации значений выборки;
- зацикливание алгоритма при малом значении α.

Метод k-средних

Метод k-средних (англ. – k-means) – один из методов кластеризации, основанный на вычислении центра масс для главных объектов кластеров.

Входные данные для алгоритма следующие:

- кластеризуемая выборка;
- параметр α – количество результирующих кластеров.

Приведем пошаговое описание алгоритма:

1) создать α главных объектов выборки, присвоить им случайные значения;
2) для каждого главного объекта вычислить центр масс;
3) перенести все главные объекты в найденные центры масс;
4) если хоть один из главных объектов сдвинулся, то переход к пункту б, иначе алгоритм завершает свою работу.

Исследование алгоритма на модельных данных

Рассмотрим работу алгоритма на наборе объектов, отображенных на рисунке 31. Добавим 3 главных объекта, которые будут характеризовать центры будущих кластеров и поместим их в случайные объекты выборки.

После проведения нескольких итераций алгоритма главные объекты сдвигаются к локальным сгущениям элементов выборки (рисунок 32). Таким образом, каждый из объектов выборки будет принадлежать к ближайшему для него главному объекту, который и образует отдельный кластер.

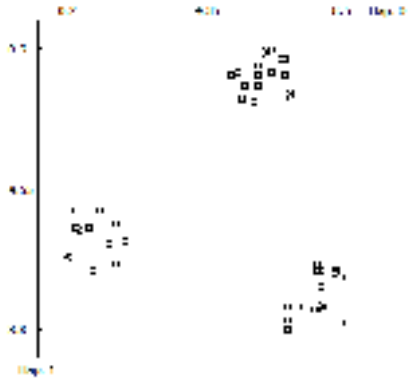

Рисунок 31 – Исходная выборка и три главных объекта

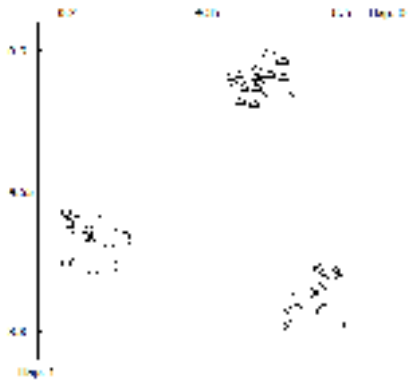

Рисунок 32 – Результат кластеризации

Выбор начальных объектов случайным образом позволяет получить различные варианты кластеризации, пример одной из вариаций можно увидеть на рисунке 33.

Выводы

Достоинства:

- разбиение на заранее известное количество кластеров;
- возможность избавления от ошибок и выбросов.

Недостатки:

- результат кластеризации зависит от выбора исходных центров кластеров;
- подходит только для задач небольшой размерности по количеству классов и переменных.

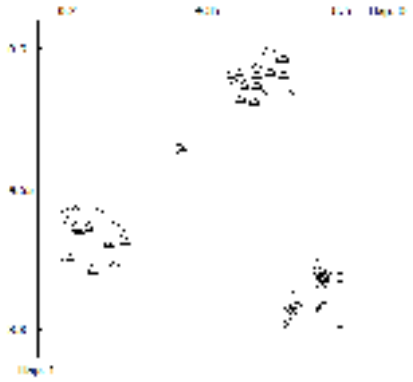

Рисунок 33 – Результат кластеризации

Кластеризатор на основе нейронной сети Кохонена

Самоорганизующаяся нейронная сеть (карта) Кохонена выполняет проецирование многомерных данных в пространство меньшей размерности. Она представляет собой двухслойную сеть (рисунок 34). Каждый нейрон первого (входного) слоя соединен со всеми нейронами второго (выходного) слоя, которые расположены в виде двумерной решетки. Количество входных нейронов совпадает с размерностью кластеризуемых данных.

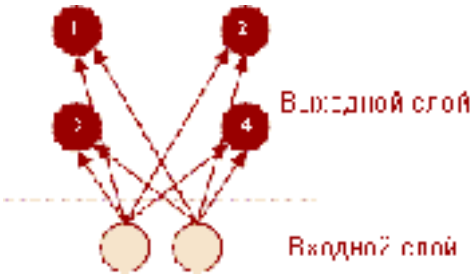

Рисунок 34 – Два слоя сети Кохонена

Нейроны выходного слоя называются кластерными элементами, их количество определят максимальное количество групп, на которые система может разделить входные данные. Увеличивая количество нейронов второго слоя можно увеличивать детализацию результатов процесса кластеризации.

При визуализации данных таким способом можно увидеть наличие или отсутствие кластерной структуры данных, число кластеров, законы совместного распределения признаков, зависимости между переменными. Рассмотрим принцип работы алгоритма.

Система работает по принципу соревнования – нейроны второго слоя соревнуются друг с другом за право наилучшим образом сочетаться с входным вектором сигналов. На каждом шаге обучения из исходного набора данных случайно выбирается один вектор (объект). Затем производится поиск нейрона выходного слоя, для которого расстояние между его вектором весов и входным вектором минимально. По определённому правилу производится корректировка весов для нейрона-победителя и нейронов из

его окрестности (рисунок 35), которая задаётся соответствующей функцией окрестности.

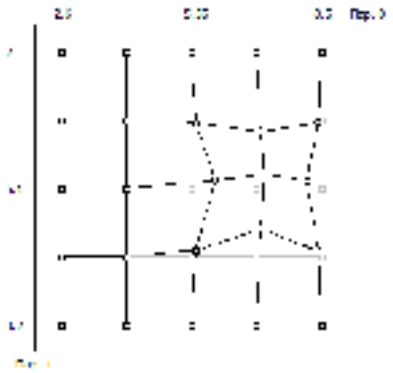

Рисунок 35 – Корректировка весов для нейрона-победителя и его соседей

Радиус действия функции окрестности будет уменьшаться с увеличением количества итераций. В качестве такой функции была использована функция Гаусса, плотность распределения которой показана на рисунке 36. Функция Гаусса имеет вид:

$$h\left(u,c,i\right)=exp\left(-\frac{\rho(c,u)}{\sigma(i)}\right), \tag{7}$$

где u – номер нейрона в двумерной решетке второго слоя сети, для которого вычисляем значение h;

c – номер нейрона-победителя в двумерной решетке второго слоя сети;

i – номер текущей итерации,

$$\sigma\left(i\right)=\frac{1}{exp\left(i^{-2}\right)}, \tag{8}$$

где i – номер текущей итерации.

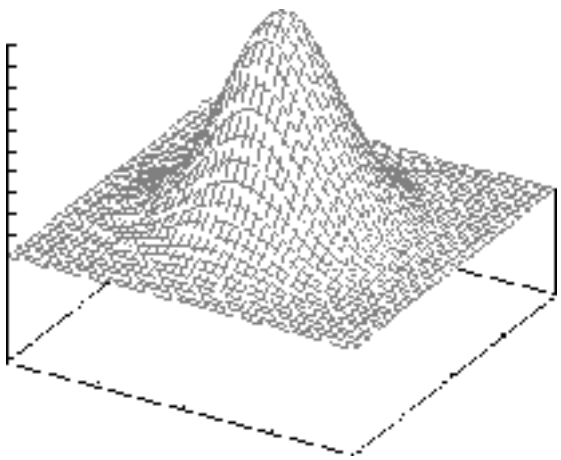

Рисунок 36 – Функция Гаусса

Входные данные для алгоритма следующие:

- кластеризуемая выборка;
- параметр N – размерность сети NxN в пределах [1; 10], при меньшем значении получится меньшее количество кластеров;
- параметр I – количество выполняемых операций;
- параметр η – коэффициент скорости обучения в пределах (0; 1).

Приведем пошаговое описание алгоритма:

1) инициировать матрицу весов случайными значениями в пределах [-1; 1];
2) установить счетчик итераций i равным 0;
3) выбрать случайный объект выборки x;
4) найти нейрона-победителя c для данного объекта;
5) изменить вес каждого нейрона w_u по формуле:

$$\overline{w_u} = \overline{w_u} + \left(\overline{w_u} - \overline{x} \right) * h\left(u,c,i \right) * \eta, \quad (9)$$

где $\overline{w_u}$ – вектор весовых коэффициентов нейрона с индексом u;

$\overline{x}$ – вектор весовых коэффициентов объекта выборки;

$h\left(u,c,i \right)$ – значение функции окрестности (7) для нейрона с номером u и нейрона-победителя с номером c на i-ой итерации;

η – коэффициент скорости обучения;

6) увеличить счетчик итераций i на единицу, переход к пункту в.

Алгоритм завершает свою работу после выполнения I итераций.

Для остановки процесса обучения также можно использовать следующие критерии:

- выход сети стабилизируется, т.е. входные вектора не переходят между кластерными элементами;
- изменения весов становятся незначительными.

Исследование алгоритма на модельных данных

В программной реализации алгоритма для оптимизации его работы в качестве начальных значений матрицы весов выходного слоя использовались промежуточные значения между минимальным и максимальным значением входных векторов. Таким образом, карта получается натянутой на крайние элементы кластеризуемой выборки в соответствии с рисунками 37 и 38.

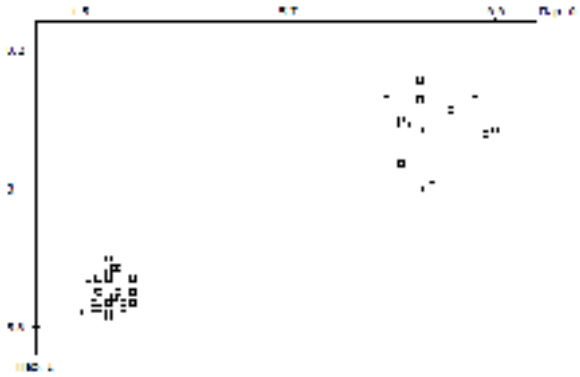

Рисунок 37 – Кластеризуемая выборка

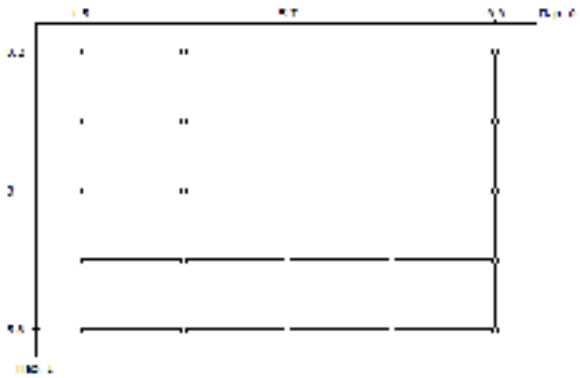

Рисунок 38 – Сеть Кохонена

Для того чтобы можно было провести кластерный анализ, проведем 1000 итераций обучения сети. На рисунке 39 видно, что некоторые нейроны заметно сдвинулись со своего изначального положения в сторону локального сгущения множества объектов, поскольку во время работы алгоритма они неоднократно выступали в качестве нейронов-победителей.

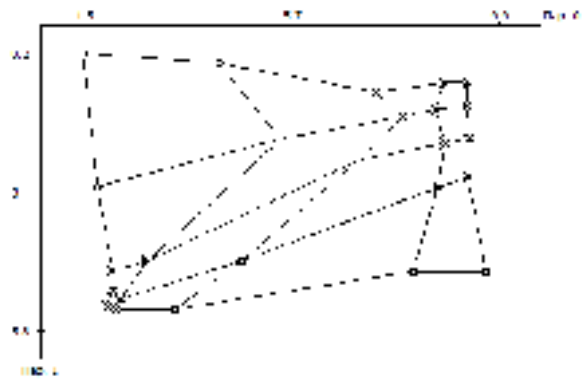

Рисунок 39 – Сеть Кохонена, 1000 итераций обучения

Однако, несмотря на проделанную работу, часть нейронов осталась практически на том же месте. Проведя еще 9000 итераций, можно заметить, что сеть уже является сформированной (рисунок 40): при проведении дополнительных итераций изменения весов нейронов становятся незначительными.

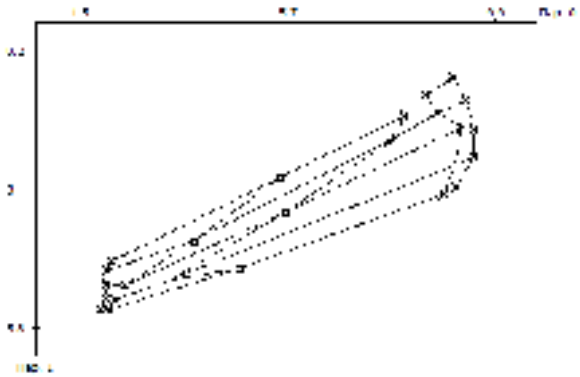

Рисунок 40 – Сеть Кохонена, 10000 итераций обучения

В данный момент нейроны, обозначенные крестом, являются нейронами-победителями для определенного набора объектов и образуют вместе с ними отдельный кластер. Построим на основе выходного слоя нейронов диаграмму, отображающую распределение объектов по кластерам (рисунок 41).

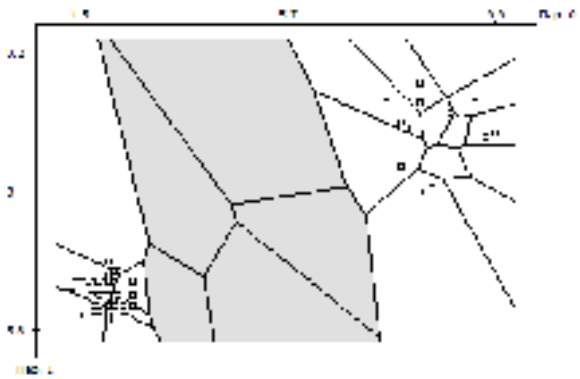

Рисунок 41 – Распределение исходной выборки

Из данного рисунка видно, что области, закрашенные серым цветом, не содержат в себе ни единого объекта, а, следовательно, не образуют никаких кластеров. Кластеры, образованные остальными нейронами, при необходимости можно объединить в один поиском в ширину или другими способами.

Выводы

Достоинства:

- высокая точность работы алгоритма;

- результирующие кластеры не пересекаются и имеют произвольную форму;
- наглядное представление многомерных данных на плоскости;
- регулируемое по количеству разбиение на классы;
- наличие эталонного объекта кластеров.

Недостатки:

- сложность алгоритма и медленная скорость работы;
- необходимость введения алгоритма объединения отдельных нейронов;
- наличие нейронов-победителей, расположенных между отдельными предполагаемыми кластерами.

Сами по себе нейронные сети задачу кластерного анализа и идентификации зависимостей не решают. Они только позволяют по внешнему виду карты выдвинуть гипотезы о наличии кластерной структуры и числе кластеров, зависимостях между значениями отдельных переменных. Выдвинутые гипотезы должны проверяться и подтверждаться иными способами. Более того, карты Кохонена могут приводить как к формированию ложных гипотез, так и к невозможности увидеть отдельные реально имеющиеся и статистически достоверные зависимости в данных.

2.6 Выводы

В данном разделе были исследованы основные моменты процесса разработки системы поиска решений на базе открытых данных. Выявлены особенности разработанных алгоритмов. Даны краткие описания алгоритмов, используемые параметры, коэффициенты и входные данные. Описан порядок выполнения алгоритмов, приведены примеры работы алгоритмов на модельных данных с иллюстрациями, определены основные достоинства и недостатки рассматриваемых методов.

Также в ходе анализа предметной области было выяснено, что в мире существуют аналоги разрабатываемой системы, но они имеют ряд недостатков, которые препятствуют удобной работе конечного пользователя: отсутствие единого механизма поиска, интуитивно непонятный интерфейс, перегруженность функционала или узкая специализация, высокие системные требования. Разработанная модель будет соответствовать мировому стандарту и лишена этих недостатков.

Рассмотренные идеи направлены на максимальную универсализацию и автоматизацию процессов жизнедеятельности открытых данных, увеличивая актуальность, достоверность и доступ к открытым данным.

3 Разработка и тестирование программного комплекса

3.1 Модель базы данных

Из статистики по документам data.gov.ru видно, что большая часть данных размещены в CSV-формате. Дело в том, что большая часть CSV-файлов имеют невалидный формат. В CSV легко допустить ошибку, а если пользователь не разбирается в стандарте, то вероятность ошибки близка к 100%. Следующие ошибки встречаются чаще всего:

- лишние кавычки. Неправильная кавычка может сделать нечитабельным программой весь документ;
- разное количество колонок в строках данных;
- разрозненность данных и отсутствие стандартов. Каждая служба публикует данные в произвольном виде;
- данные разбросаны по разным региональным ресурсам;
- отсутствие единого механизма поиска;
- отсутствие API для доступа к данным.

Поэтому для разработанной системы данные хранятся на диске в виде текстовых файлов. В текстовых файлах (с расширением txt) данные представлены на специальном языке хранения данных, описанных в пункте 2.3. В этом виде они могут приготавливаться и корректироваться вне системы, например, в любом текстовом редакторе.

Основной рабочий цикл программы, состоит из загрузки базы данных, работы с данными, т.е. просмотра и изменения, и сохранения измененной версии базы данных.

Сначала пользователь загружает базу данных из текстового файла или из ПОДа в системную память, где данные формируют сеть объектов. Пользовать имеется возможность просматривать и корректировать сеть. В процессе работы в сеть может считываться содержимое ещё нескольких файлов.

Если в сети присутствуют правила, то при любом изменении данных, состоящем в корректировке значения или добавления связи, может быть запущен продукционный механизм, действие которого состоит либо в множестве вторичных изменений сети с целью соблюдения всех правил, либо в полном отказе от изменения, в том числе первичного, если правила сети несовместимы с данным изменением.

Пользователь имеет возможность перемещать блоки данных между сетью и текстовым редактором в обоих направлениях с помощью мыши или через системный буфер. Системный буфер может быть использован также и для переноса данных между двумя сетями открытыми в различных запусках программы.

После того как данные в сети изменены или добавлены, полученная сеть сохраняется на диск в виде текстового файла базы данных.

Поскольку база данных может иметь вид текстового файла и язык данных прост и читабелен, то иногда пользователю может быть удобно исправить данные непосредственно в файле базы данных. Для этого разработанная система может исполнять функцию обычного текстового редактора, с тем дополнением, что между этим редактором и окнами возможен перенос данных с помощью мыши.

Пользователь может работать единовременно только с одной базой данных из одного вызова программы. Если нужно работать с двумя или более базами данных параллельно, следует вызвать программу многократно.

3.2 Схема работы программы

Основной рабочий цикл программы состоит из следующих частей, отображенных на рисунке 42.

Инициализация

На данном этапе происходят следующие действия:

- определение размера рабочего окна;
- инициализация состояния клавиатуры: какая из клавиш в текущий момент времени нажата, зажата или отпущена;
- инициализация состояния мыши: позиция курсора на экране, какая из клавиш в текущий момент времени нажата, зажата или отпущена, было ли осуществлено вращение колесика или его нажатие;
- создание рабочего окна, инициализация необходимых переменных;
- определение прототипа функции WndProc;
- установки, касаемые работы библиотеки OpenGL;
- проверка на наличие ошибок.

Активная ли система?

Выход их программы осуществляется одним из двух способов:

- нажатие клавищи Esc;
- нажатие по крестику в заголовке окна.

Обновление состояния

Данный этап составляет основу работу программы. В течение каждой секунды необходимо многократно обновить состояния мыши, клавиатуры, чтобы пользователь мог взаимодействовать с программой. Также осуществляется обновление остальных объектов программы, если это необходимо.

Рисунок 42 – Основной цикл программы

Вывод на экран

Если программа активна, и не нажата ESC, визуализируется сцена и меняется буфер (используя двойную буферизацию исключается мерцание при анимации). Используя двойную буферизацию, рисуется всё на "скрытом экране" (второй буфер) так, чтобы пользователи не могли видеть этого. Когда меняется буфер, экран (первый буфер), который видит пользователь, становится скрытым, а скрытый - становится видимым. Таким образом, виден не процесс прорисовки сцены, а только результат визуализации.

Для этого необходимо выполнить:

- очистку экрана и буфера глубины;
- сброс просмотра;

- поворот и сдвиг сцены;
- отрисовка полигонов;
- смена буффера.

Приостановление работы

Чтобы обеспечить одинаковую скорость работы программы на различных устройствах, был введен счетчик кадров. При использовании 60 кадров в секунду, каждый фрейм требует 16,(6) миллисекунд. Если мощность компьютера позволяет выполнить необходимые действия за меньший промежуток времени, то освободившееся миллисекунды процессор будет находиться в режиме ожидания.

Деинициализация

Следующая секция используется только перед выходом из программы. Ее задача освободить контекст рендеринга, контекст устройства и, наконец, дескриптор окна. Необходима также проверка на наличие ошибок. Если программа неспособна удалить какую-нибудь часть из контекстов окна, появится окно с соответствующим сообщением об ошибке. После производится удаление всех используемых объектов. После того, как все описанные действия выполнены, окно уничтожается, и программа закрывается без каких-либо утечек памяти.

3.3 Визуализация многомерных данных

Визуализация данных – задача, с которой сталкивается в своей работе любой исследователь. К задаче визуализации данных сводится проблема представления в наглядной форме данных эксперимента или результатов теоретического исследования. Традиционные инструменты в этой области – графики и диаграммы – плохо справляются с задачей визуализации, когда возникает необходимость изобразить более трех взаимосвязанных величин.

Разработанная система позволяет визуализировать до 6 парамертов:

- x координата;
- y координата;
- z координата;
- размер объекта;
- цвет объекта;
- форма объекта.

К сожалению, цвет и форма позволяют отразить только ограниченные интервалы. В случае с цветом возможны два варианта:

- значение может пролегать в интервале от 0 до 255. Соответственно – 256 состояний. Можно пользоваться данным способом при большом разбросе параметра;
- значение может быть одним из различных цветов: красный, зеленый, белый, черный, синий, серый, фиолетовый, голубой, желтый и т.д.

Возможен также вариант с использованием всех комбинаций синего, красного и зеленого оттенков. Однако данный способ слишком трудно воспринимается для глаза.

При наличии булевых параметров, к примеру, «мужчина или женщина», «да или нет», «черный или белый», можно использовать различную форму объекта: куб или шар.

3.4 API для поддержки

Для поддержки приложения важно разработать удобный API для использования клиентом. API должно быть легко использовать, но сложно использовать неправильно.

Первым шагом предлагается перечислить все типы данных, которые клиентское приложение может захотеть получить или передать с помощью сервиса. Описание отображает назначения данных в приложении. Пользователь должен работать с точки зрения клиентского приложения, а не сервиса.

Было решено отказаться от использования некоторых классов во избежание возникновения дубликатов экземпляров объектов. К примеру, класс Display можно было случайно создать повторно. Использование namespace вместо class позволяет этого избежать. В данном случае нет необходимости выделять память под объект и в дальнейшем ее освобождать.

Список использованных namespace представлен ниже:

- Display – содержит значения высоты и ширины дисплея компьютера;
- Keyboard – содержит по 256 состояний клавиш клавиатуры: Нажата ли кнопка, зажата ли кнопка, отжата ли кнопка, а также идентификатор последней нажатой кнопки;
- Mouse – содержит значения X и У курсора в окне приложения, нажата, зажата или отжата левая и права кнопка мыши, а также функции для работы с колесикком мыши;
- Camera – содержит значения позиции камеры в пространстве, повороты относительно осей. Все преобразования для установки камеры и ее поворота берет на себя данная часть программы;
- Window – содержит значения X и У, ширины и высоты окна. Реализованы функции для создания, уничтожения, передвижения окна.

Загрузка данных возможна как из текстового файла, так и с сайта data.gov.ru. Описание функций приведено в приложении Б. Обработка данных возможна как встроенными алгоритмами, так и алгоритмами, добавленными пользователем.

3.5 Численные исследования

В рамках выполнения выпускной квалификационной работы было предложено рассмотреть работу нескольких алгоритмов классификации на наборе данных с ирисами.

На рисунках 43 и 44 представлены исходные данные для алгоритмов. На рисунках 45-49 представлены результаты работы алгоритмов. Результаты исследований представлены в таблице 3.

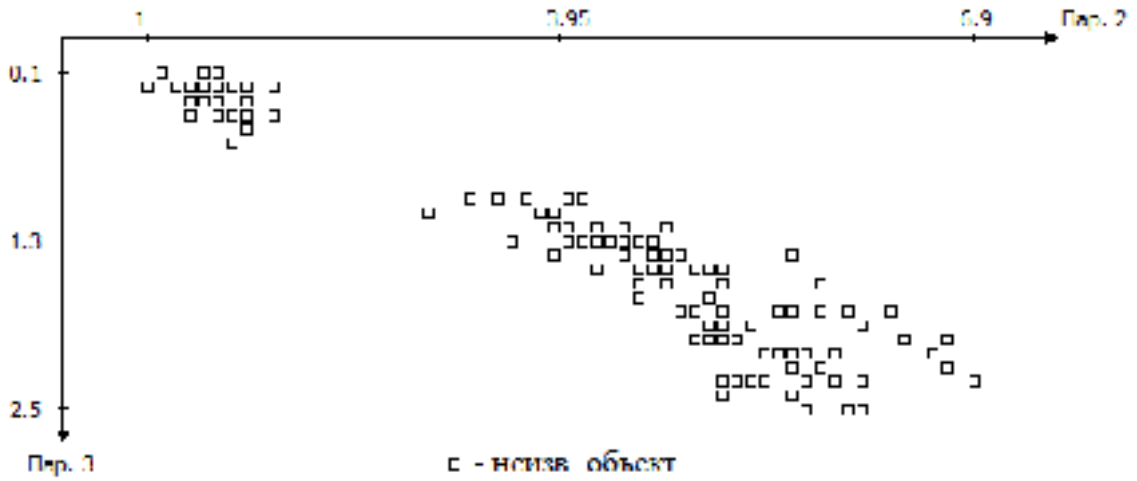

Рисунок 43 – Исходные данные для задачи кластеризации

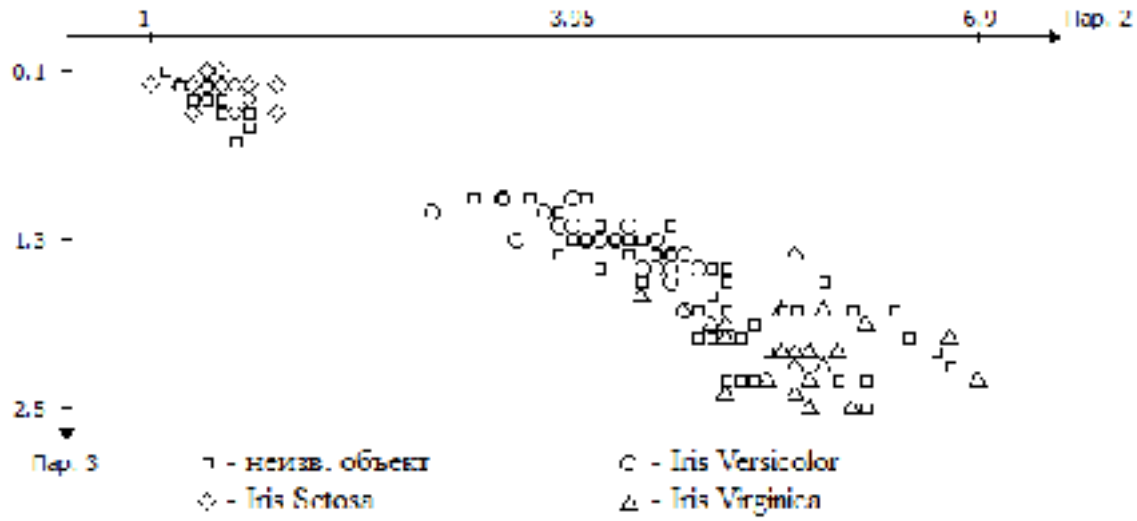

Рисунок 44 – Исходные данные для задачи классификации

Метод k ближайших соседей

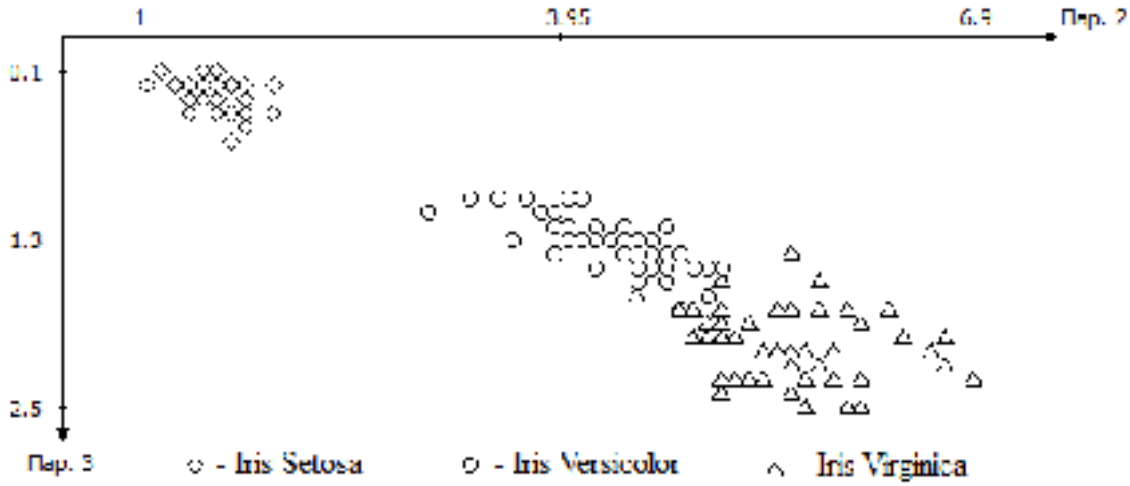

Рисунок 45 – Результат классификации методом k ближайших соседей

Алгоритм FRiS-STOLP

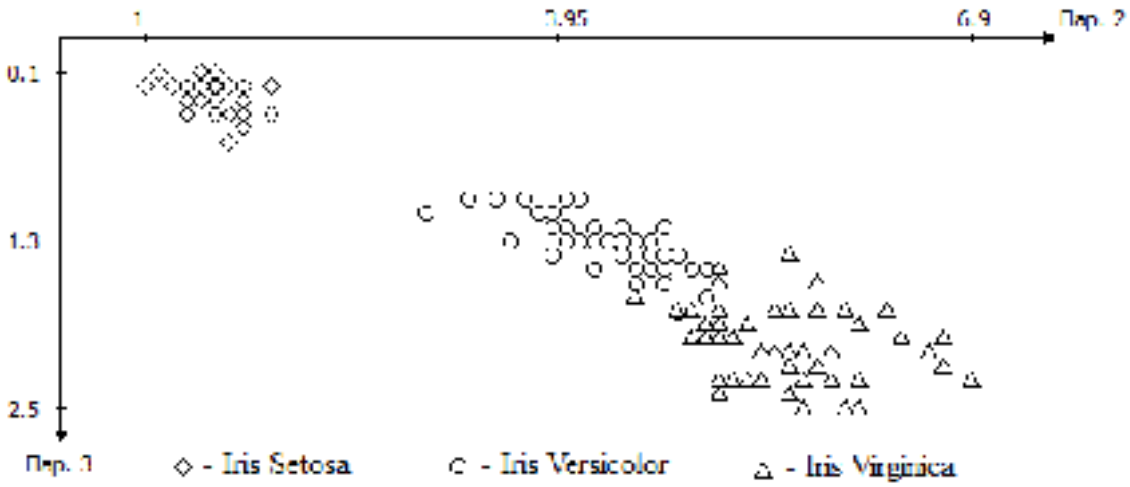

Рисунок 46 – Результат классификации алгоритмом FRiS-STOLP

Алгоритм FOREL

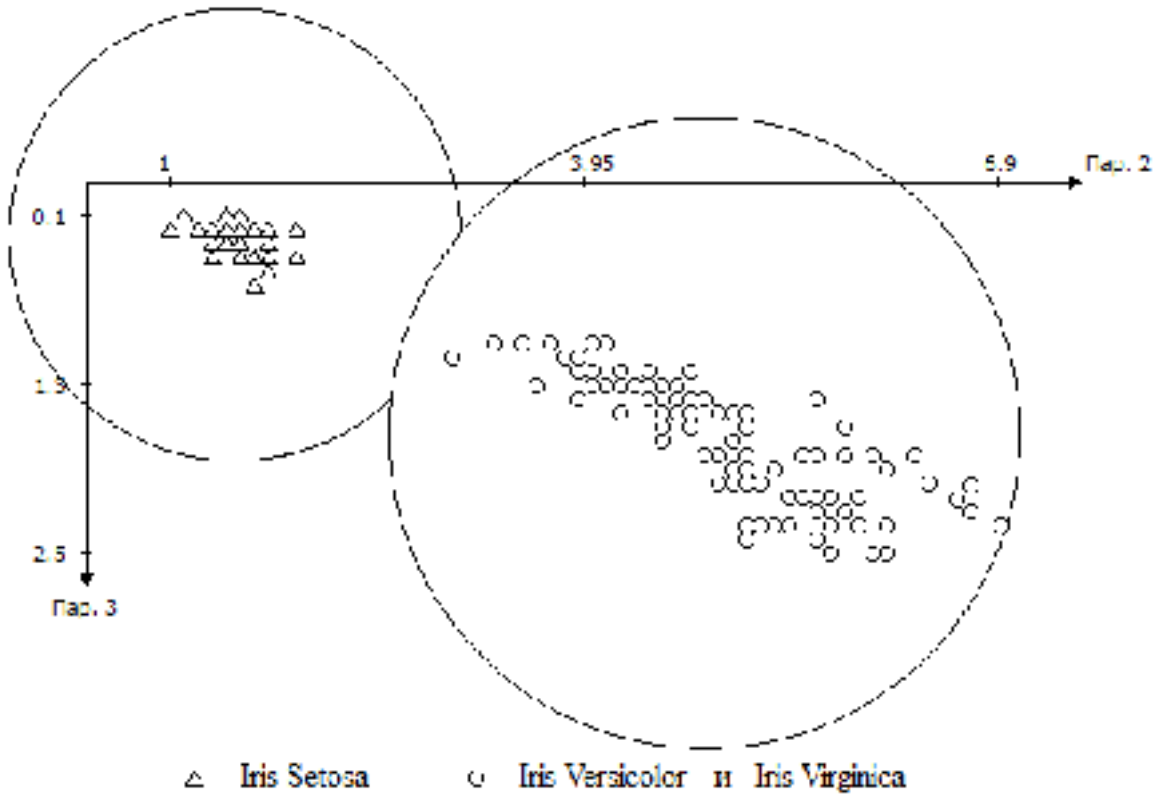

Рисунок 47 – Результат кластеризации алгоритмом FOREL

Метод k-средних

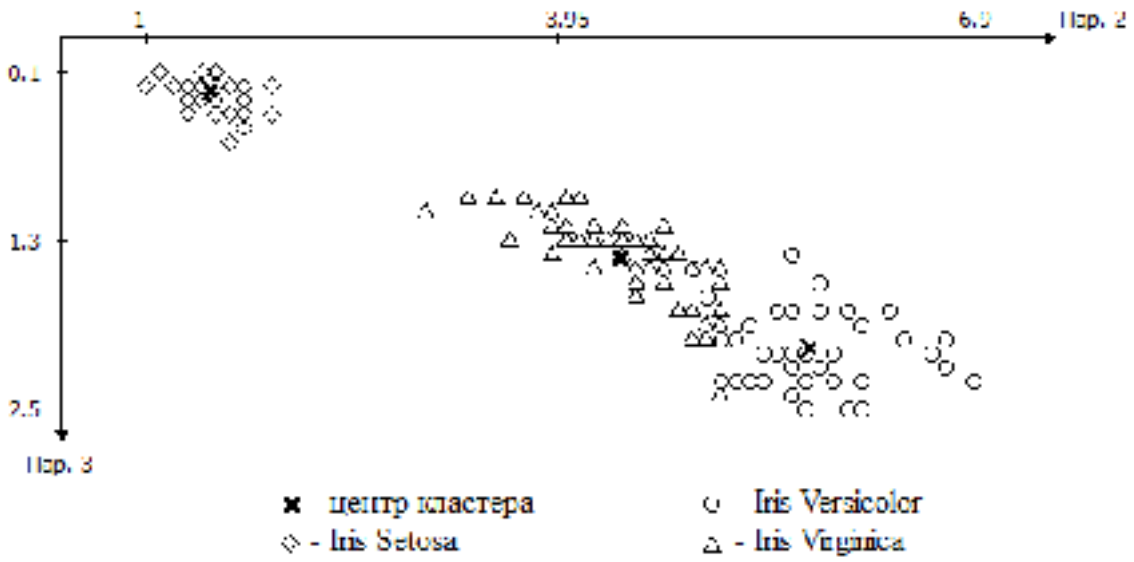

Рисунок 48 – Результат кластеризации методом k-средних

Сеть Кохонена

Рисунок 49 – Результат кластеризации при использовании нейронной сети

Таблица 3 – Результаты исследований

Алгоритм	Исх. данные	Результат	
		объекты	верно определено
k ближ. соседей	Неизвестно – 75 объектов 1 класс – 25 объектов 2 класс – 25 объектов 3 класс – 25 объектов	1 класс – 26 2 класс – 24 3 класс – 25	98.6%
FRiS-STOLP		1 класс – 25 2 класс – 25 3 класс – 25	100%
FOREL	Неизвестно – 150 объектов	1 кластер – 50 2 кластер – 100	66.6%
k-средних		1 кластер – 50 2 кластер – 69 3 кластер – 31	86.6%
Сеть Кохонена		1 кластер – 50 2 кластер – 49 3 кластер – 51	99.3%

Алгоритм FOREL был исключен из дальнейшего рассмотрения, поскольку он приспособлен к работе с кластерами, имеющие небольшое внутрикластерное расстояние и большое межкластерное. На рисунке 50 представлены исходные данные для алгоритмов. На рисунках 51 и 52 представлены результаты работы алгоритмов. Разработанные алгоритмы позволяют выделить 3 кластера. Результаты исследований представлены в таблице 4.

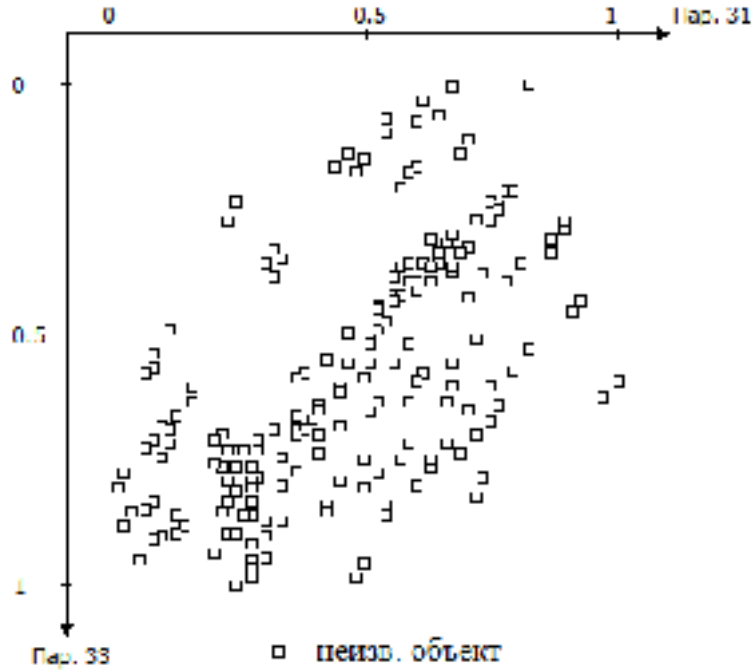

Рисунок 50 – Исходные данные ЭРИ для задачи кластеризации

Метод k-средних

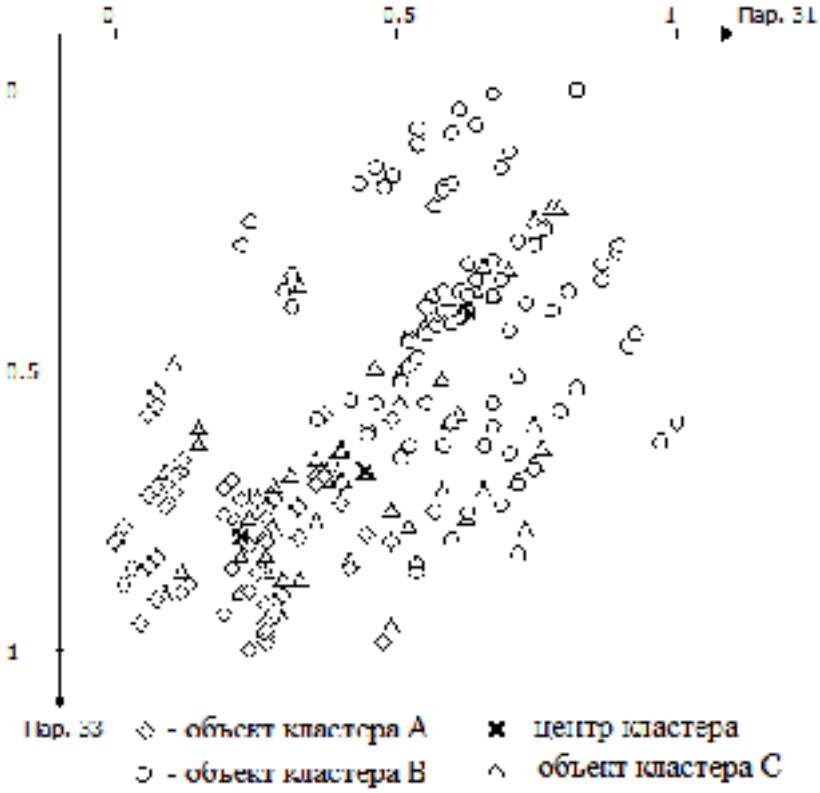

Рисунок 51 – Результат кластеризации методом k-средних

Сеть Кохонена

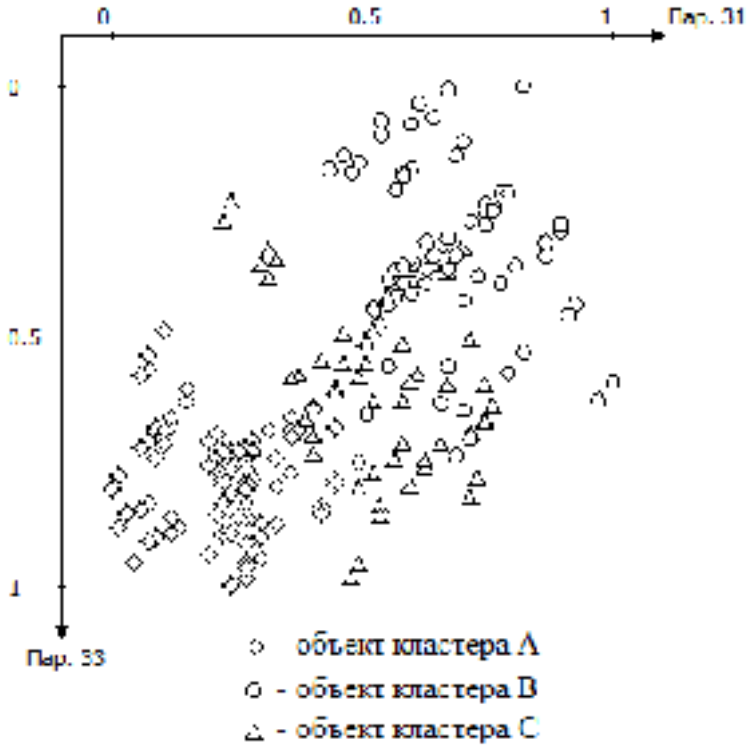

Рисунок 52 – Результат кластеризации при использовании нейронной сети

Таблица 4 – Результаты исследований

Алгоритм	Исх. данные	Результат (объекты)
k-средних	Неизвестно – 198 объектов	1 кластер – 85 2 кластер – 55 3 кластер – 58
Сеть Кохонена		1 кластер – 78 2 кластер – 55 3 кластер – 65

На рисунках 53 и 54 представлены исходные данные для алгоритмов в различных срезах. На рисунках 55-58 представлены результаты работы алгоритмов. Разработанные алгоритмы позволяют выделить 2-3 кластера различными способами. Результаты исследований представлены в таблице 5.

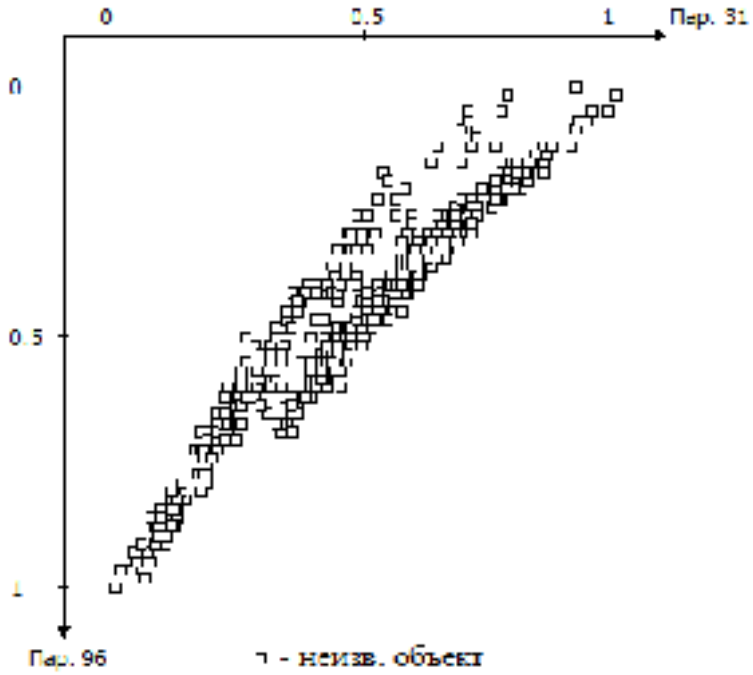

Рисунок 53 – Исходные данные ЭРИ для задачи кластеризации

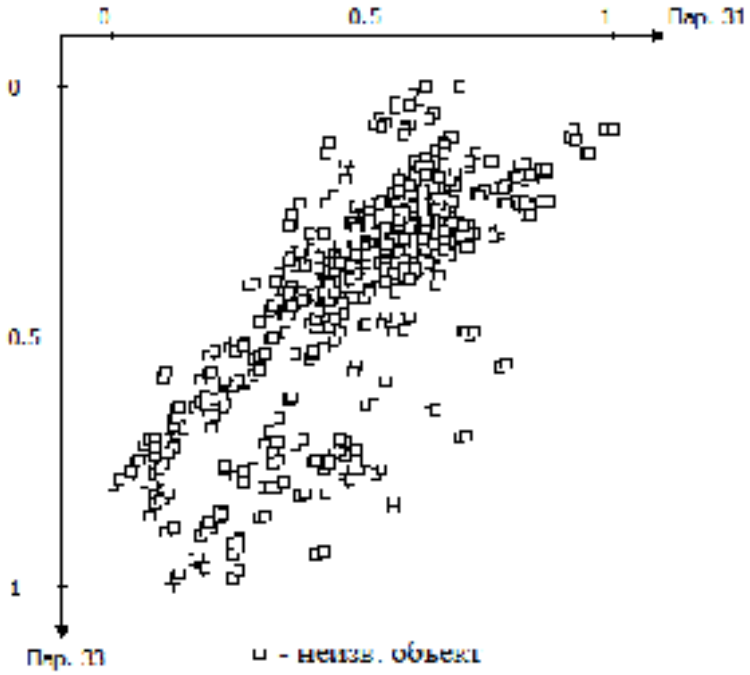

Рисунок 54 – Исходные данные ЭРИ для задачи кластеризации

Метод k-средних

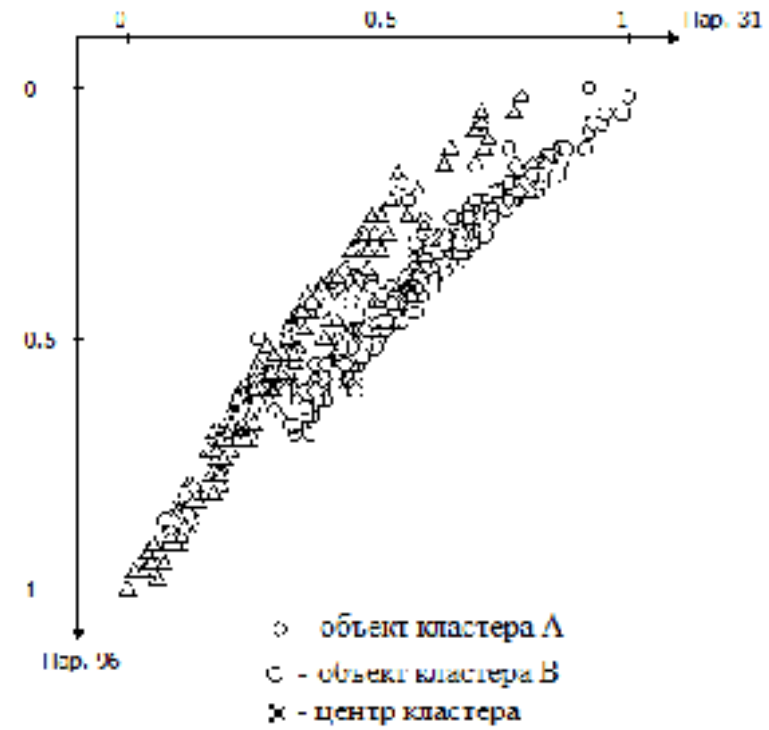

Рисунок 55 – Результат кластеризации методом k-средних

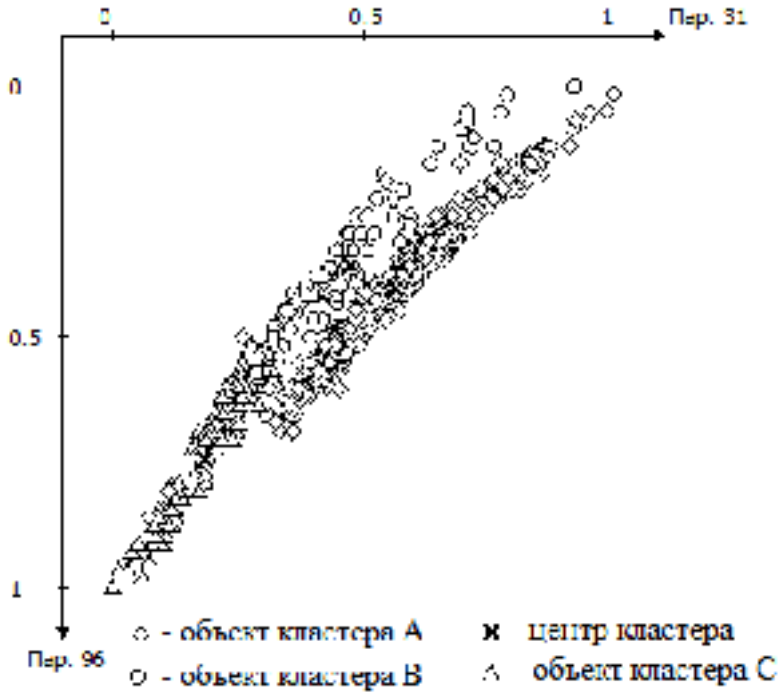

Рисунок 56 – Результат кластеризации методом k-средних

Сеть Кохонена

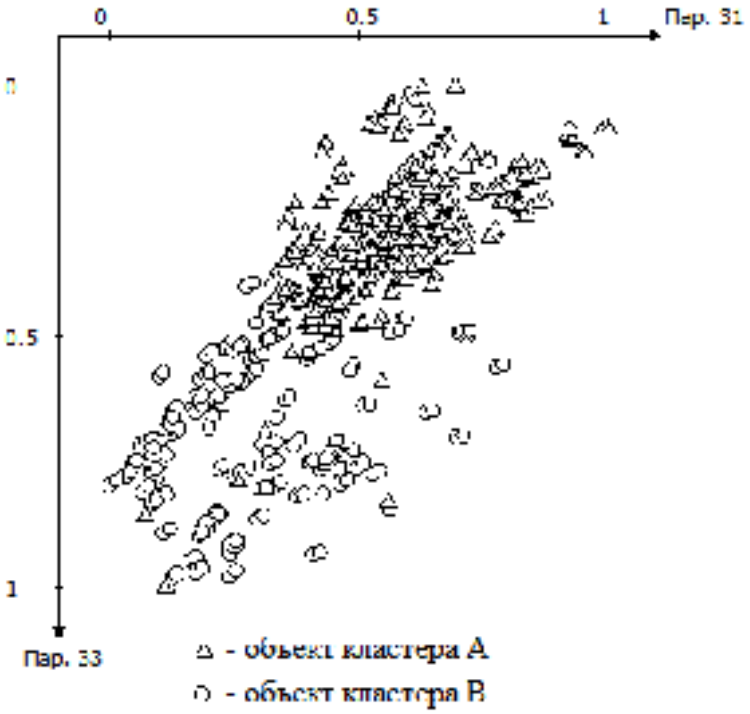

Рисунок 57 – Результат кластеризации при использовании нейронной сети

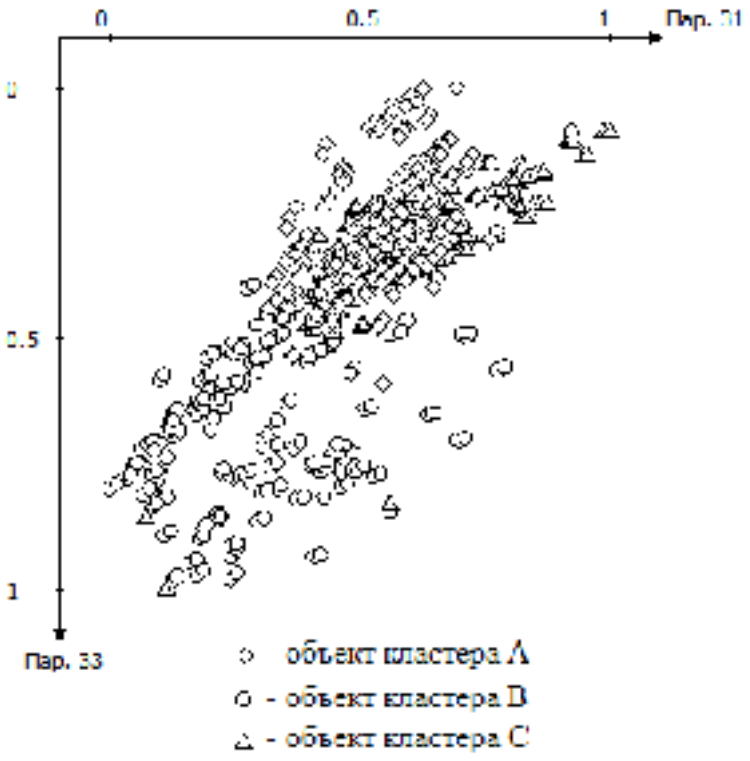

Рисунок 58 – Результат кластеризации при использовании нейронной сети

Таблица 5 – Результаты исследований

Алгоритм	Исх. данные	Результат	
		кол-во кластеров	распределение объектов
k-средних	Неизвестно – 620 объектов	2	1 кластер – 420 2 кластер – 199
		3	1 кластер – 417 2 кластер – 122 3 кластер – 80
		3	1 кластер – 215 2 кластер – 205 3 кластер – 199
Сеть Кохонена		2	1 кластер – 417 2 кластер – 202
		3	1 кластер – 216 2 кластер – 202 3 кластер – 201

3.6 Выводы

В данном разделе была разработана модель базы данных, представлена схема работы программы, разработан API для дальнейшей поддержки программы, исследована работа различных алгоритмов на реальных данных. Выборка «Ирисы Фишера» была использована для подбора оптимальных значений параметров алгоритмов. Были классифицированы объекты, определено наличие трех необходимых кластеров. В малой выборке ЭРИ было выделено 3 кластера, наличие которых было известно. В большое выборке ЭРИ было выделено 2-3 кластера различными способами.

ЗАКЛЮЧЕНИЕ

Целью данной работы было моделирование системы поиска решений на базе открытых данных, а также выявление всех возможных трудностей при её создании и эксплуатации. В результате проделанной работы было сделано следующее:

- проведен анализ предметной области концепции открытых данных в научных исследованиях, требований, предъявляемых к форматам их представления и выолнен обзор уже разработанных программных комплексов, вявлены их достоинства и недостатки;
- выбрана подходящая платформы и язык программирования для реализации программного комплекса;
- реализованы и исследованы алгоритмы поиска и алгоритмы кластеризации: FOREL, k-средних, кластеризатор на основе самоорганизующейся сети Кохонена;
- проанализированы особенности работы алгоритмов на модельных и реальных данных, сравнить полученные результаты;
- программно реализован прототип программной системы поиска решений на базе открытых данных.

Оценить качество работы на многомерных данных сложно, так как размерность исследуемой выборки высока, а просмотреть срезы данных по всем наборам признаков является физически невозможным. Однако полученные результаты позволяют сделать определенные выводы о данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТО 4.2-07-2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Введ. 09.01.2014. – Красноярск: СФУ, 2014. – 60 с.

2. Лафоре, Р. Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е изд. / Р. Лафоре – СПб.: Питер, 2014. – 928 с.

3. Муравьев, А. С. Модифицированный алгоритм растущего нейронного газа применительно к задаче классификации / А. С. Муравьев, А. А. Белоусов // Вестник науки Сибири. – 2014. – №4(14). – С. 105-111.

4. Муртазаев, А. К. Фазовые переходы в антиферромагнитной модели Изинга на квадратной решетке с взаимодействиями вторых ближайших соседей / А. К. Муртазаев // Журнал экспериментальной и теоретической физики. – 2013. – Т. 144, вып. 6. – С. 1236-1245.

5. Павловская, Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб.: Питер, 2012. – 461 с.

6. Попова, О. А. Модели и методы интеллектуального анализа данных: учебно-методическое пособие [Электронный курс] / О. А. Попова. – Красноярск: Сиб. федер. ун-т, 2012.

7. Сараев, В. Отдайтесь большой цифре / В. Сараев // Эксперт. – 2015. – №9. – С. 51-55.

8. Удалова, Ю. В. Математические и алгоритмические основы объектно-ориентированного программирования: [Электронный курс]: учеб.-метод. пособие / Ю. В. Удалова. – Красноярск: Сиб. федер. ун-т, 2013.

9. Фазылов, Ш. Х. Модель распознающих операторов, основанных на принципе ближайшего соседа, в условиях взаимосвязанности признаков / Ш. Х. Фазылов, Чье Ен Ун // Информатика и системы управления. – 2012. – №4(34). – С. 34-42.

10. Царев, Р. Ю. Алгоритмы и структуры данных: учеб. пособие / Р. Ю. Царев. – Красноярск: Сиб. федер. ун-т, 2013. – 160 с.

11. Шаграев, А. Г. Трансдуктивное обучение логистической регрессии в задаче классификации текстов / А. Г. Шаграев, И. А. Бочаров, В. Н. Фальк // Программные продукты и системы. – 2014. – №2. – С. 114-118.

ПРИЛОЖЕНИЕ А Исходный код программы

Листинг А.1 – Основной цикл программы

```
// Libraries
#pragma comment (lib, "opengl32.lib")
#pragma comment (lib, "glu32.lib")
#pragma comment (lib, "wsock32.lib")
#define _CRT_SECURE_NO_WARNINGS

// Includes
#include <winsock.h>
#include <windows.h>
#include <math.h>
#include <time.h>
#include <string>
#include <gl/glew.h>
#include <gl/gl.h>
#include <gl/glu.h>
using namespace std;

// Extra types
typedef unsigned char  uchar;
typedef unsigned short ushort;
typedef unsigned int   uint;

// Engine
#include "functions.cpp"
#include "keyboard.cpp"
#include "mouse.cpp"
#include "camera.cpp"
#include "display.cpp"
#include "window.cpp"
#include "object.cpp"

// Main function
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
	// Initialize everything
	Display::initialize();
	Keyboard::initialize();
	Mouse::initialize();

	// Create window
	if (Window::create(PRGRM_TITLE, PRGRM_FULL,
		(Display::width  - PRGRM_WIDTH)  / 2,
		(Display::height - PRGRM_HEIGHT) / 2,
		PRGRM_WIDTH, PRGRM_HEIGHT)) {
		// Seed random generator
		srand((unsigned)time(nullptr));

		// Variables
		constexpr float timeForFrame = (float)1000.0 / PRGRM_FPS;
		int frame       = 0;
		int timeCurrent = clock();
		int timeStart   = timeCurrent;
		int timeEnd     = timeStart + 1000;
		int timeSleep;
		MSG msg;

		// Main cycle
		while (Engine::prgrm) {
			// Retrieve message
			while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
				TranslateMessage(&msg);
				DispatchMessage(&msg);
			}
```

```
                        // Do frame tick
                        Engine::step();
                        if (!room.doTransition()) {
                                Engine::draw();
                        }

                        // Wait some mseconds if needed
                        if ((timeSleep = timeStart + timeForFrame * ++frame - (timeCurrent =
clock())) > 0) {
                                timeCurrent += timeSleep;
                                Sleep(timeSleep);
                        }

                        // Check FPS
                        if (timeEnd <= timeCurrent) {
                                Engine::fps = frame;
                                frame       = 0;
                                timeStart   = timeCurrent;
                                timeEnd     = timeCurrent + 1000;
                        }
                }

                // App end
                Engine::deleteObjects();
                Window::destroy();
        }
        return 0;
}
```

Листинг А.2 – Дополнительные функции

```
// constants
constexpr float PI        = (float)3.141592741;
constexpr float PI_180    = (float)180.0 / PI;
constexpr float PI_HALF   = PI / 2.0;
constexpr float PI_DOUBLE = PI * 2.0;
constexpr float SQRT2     = (float)1.414213538;
constexpr float SQRT3     = (float)1.732050776;
constexpr float SQRT5     = (float)2.236067977;

string getTime() {
	uint t = clock();

	uint msec = t % 1000;
	uint sec  = (t / 1000) % 60;
	uint min  = (t / 1000) / 60;

	string time = "";

	if (min < 10) time += "0";
	time += to_string(min);
	time += ":";
	if (sec < 10) time += "0";
	time += to_string(sec);
	time += ":";
	if (msec < 100) time += "0";
	if (msec < 10)  time += "0";
	time += to_string(msec);

	return time;
}

// constexpr round
constexpr int rnd(float value) {
	return int(value) + (value - int(value) >= 0.5 ? 1 : 0);
}

// [-127; 128] to float [-1.0; 1.0]
constexpr float to_float(char value) {
	return value / (float)100.0;
}

// float [-1.0; 1.0] to char [-127; 128]
constexpr char to_char(float value) {
	return rnd(value * 100.0);
}

// Show message
void showMessage(string text) {
	MessageBox(nullptr, text.c_str(), "Message", MB_OK);
}

// string to int
int to_int(string value) {
	return stoi("0" + value);
}

// string to float
float to_float(string value) {
	return stof(value);
}

// [0 - 0xFFFFFFFF]
uint _random() {
	return (rand() & 0xFF) | ((rand() & 0xFF) << 8) | ((rand() & 0xFF) << 16) | ((rand() &
0xFF) << 24);
```

```
}

// [0 - value]
uint random(uint value) {
	return _random() % (value + 1);
}

// [from - to]
int random(int from, int to) {
	return from + _random() % (abs(to - from) + 1);
}

// [0.0 - 1.0]
float random() {
	return (float)rand() / RAND_MAX;
}

// [0.0 - value]
float random(float value) {
	return random((uint)value) + random() * (value - (int)value);
}

// [from - to]
float random(float from, float to) {
	return from + random(fabs(to - from));
}

// Length direction x
float xAngleLength(float length, float xAngle, float yAngle = 90.0) {
	return length * cos(xAngle / PI_180) * sin(yAngle / PI_180);
}

// Length direction y
float yAngleLength(float length, float xAngle, float yAngle = 90.0) {
	return length * sin(xAngle / PI_180) * sin(yAngle / PI_180);
}

// Length direction z
float zAngleLength(float length, float xAngle, float yAngle = 90.0) {
	return length * sin(yAngle / PI_180 - PI_HALF);
}

// Squared number
constexpr float sqr(float x) {
	return x * x;
}

// Distance between (x1,y1) and (x2,y2)
float distanceBetween(float x1, float y1, float x2, float y2) {
	return sqrt(sqr(x2 - x1) + sqr(y2 - y1));
}

// Distance between (x1,y1,z1) and (x2,y2,z2)
float distanceBetween(float x1, float y1, float z1, float x2, float y2, float z2) {
	return sqrt(sqr(x2 - x1) + sqr(y2 - y1) + sqr(z2 - z1));
}

// Horizontal direction from (x1,y1) to (x2,y2)
float angleBetween(float x1, float y1, float x2, float y2) {
	const float xx = x2 - x1;
	if (xx != 0.0) {
		float direction = atan((y2 - y1) / xx) * PI_180;

		if (x2 > x1) {
			if (y1 > y2)
				return direction + (float)360.0;
			else
				return direction;
```

```
                }
                return direction + (float)180.0;
        }
        if (y1 > y2) return 270.0;
        if (y1 < y2) return 90.0;
        return 0.0;
}

// Vertical direction from (x1,y1,z1) to (x2,y2,z2)
float angleBetween(float x1, float y1, float z1, float x2, float y2, float z2) {
        const float distance = angleBetween(x1, y1, x2, y2);

        if (distance != 0.0) return atan((z2 - z1) / distance) * PI_180 + (float)90.0;
        if (z1 < z2) return 180.0;
        return 0.0;
}
```

Листинг А.3 – Управление клавиатурой

```
// Keyboard
namespace Keyboard {
	constexpr int KEYS_COUNT = 256;
	bool isReleased[KEYS_COUNT];
	bool isPressed[KEYS_COUNT];
	bool isHeld[KEYS_COUNT];
	uchar lastPressedKey;

	// Reset keyboard status
	void reset() {
		memset(isReleased, false, KEYS_COUNT);
		memset(isPressed,  false, KEYS_COUNT);
		lastPressedKey = 0;
	}

	// Initialize
	void initialize() {
		memset(isHeld, false, KEYS_COUNT);
		reset();
	}

	// Get last pressed key
	uchar getLastPressedKey() {
		if (lastPressedKey >= 'A' && lastPressedKey <= 'Z') {
			return isHeld[VK_SHIFT] ? lastPressedKey : tolower(lastPressedKey);
		}

		if (!isHeld[VK_SHIFT]) {
			switch (lastPressedKey) {
			case '0':
			case '1':
			case '2':
			case '3':
			case '4':
			case '5':
			case '6':
			case '7':
			case '8':
			case '9':
				return lastPressedKey;

			case VK_OEM_1:      return ';';
			case VK_OEM_PLUS:   return '=';
			case VK_OEM_COMMA:  return ',';
			case VK_OEM_MINUS:  return '-';
			case VK_OEM_PERIOD: return '.';
			case VK_OEM_2:      return '/';
			case VK_OEM_3:      return '`';

			case VK_OEM_4: return '[';
			case VK_OEM_5: return '\\';
			case VK_OEM_6: return ']';
			case VK_OEM_7: return '\'';
			}
		} else {
			switch (lastPressedKey) {
			case '0': return ')';
			case '1': return '!';
			case '2': return '@';
			case '3': return '#';
			case '4': return '$';
			case '5': return '%';
			case '6': return '^';
			case '7': return '&';
			case '8': return '*';
```

```
                        case '9': return '(';

                        case VK_OEM_1:      return ':';
                        case VK_OEM_PLUS:   return '+';
                        case VK_OEM_COMMA:  return '<';
                        case VK_OEM_MINUS:  return '_';
                        case VK_OEM_PERIOD: return '>';
                        case VK_OEM_2:      return '?';
                        case VK_OEM_3:      return '~';

                        case VK_OEM_4: return '{';
                        case VK_OEM_5: return '|';
                        case VK_OEM_6: return '}';
                        case VK_OEM_7: return '\"';
                        }
                }

                return (lastPressedKey == ' ') ? ' ' : 0;
        }
}
```

Листинг А.4 – Управление мышкой

```
// Mouse buttons
enum MouseButton : uchar {
	MOUSE_LEFT	= 0,
	MOUSE_RIGHT	= 1,
	MOUSE_WHEEL	= 2,
	MOUSE_MIDDLE	= 2,
	MOUSE_CENTER	= 2,
	MOUSE_WHEEL_UP	= 0,
	MOUSE_WHEEL_DOWN = 1
};

// Mouse
namespace Mouse {
	int  x;
	int  y;
	bool isReleased[3];
	bool isPressed[3];
	bool isHeld[3];
	bool isWheelRotated[2];
	bool isVisible;
	bool isOutside = false;

	// Reset buttons status
	void reset() {
		memset(isPressed,  false, 3);
		memset(isReleased, false, 3);
		isWheelRotated[MOUSE_WHEEL_UP] = isWheelRotated[MOUSE_WHEEL_DOWN] = false;
	}

	// Move mouse
	void move(int x, int y) {
		if (GetActiveWindow()) {
			SetCursorPos(x, y);
		}
	}

	// Show mouse
	void show() {
		while (ShowCursor(true) < 0);
		isVisible = true;
	}

	// Hide mouse
	void hide() {
		while (ShowCursor(false) >= 0);
		isVisible = false;
	}

	// Initialize
	void initialize() {
		memset(isHeld, false, 3);
		reset();
	}
}
```

Листинг А.5 – Управление камерой

```
// Camera
namespace Camera {
	float x      = 0.0;
	float y      = 0.0;
	float z      = 0.0;
	float xAngle = 0.0;
	float yAngle = 0.0;
	float zAngle = 0.0;
	float zNear  = 0.01;
	float zFar   = 512.0;
	float fov    = 60.0;

	// Rotate world
	void update() {
		glLoadIdentity();
		glRotatef(-yAngle, 1.0, 0.0, 0.0);
		glRotatef(-zAngle, 0.0, 1.0, 0.0);
		glRotatef(-xAngle, 0.0, 0.0, 1.0);
		glTranslatef(-x, -y, -z);
	}

	// Set camera state
	void set(float _x, float _y, float _z, float _xAngle, float _yAngle = 90.0, float
_zAngle = 0.0) {
		x = _x;
		y = _y;
		z = _z;
		xAngle = _xAngle - (float)90.0;
		yAngle = _yAngle;
		zAngle = _zAngle;

		update();
	}

	// 2d mode
	void set2d(int x1, int y1, int x2, int y2) {
		glMatrixMode(GL_PROJECTION);
		glPushMatrix();
		glLoadIdentity();

		glOrtho(x1, x2, y2, y1, 0.0, 1.0);
		glMatrixMode(GL_MODELVIEW);
		glLoadIdentity();
	}

	// 3d mode
	void set3d() {
		glMatrixMode(GL_PROJECTION);
		glPopMatrix();
		glMatrixMode(GL_MODELVIEW);
		glLoadIdentity();
		update();
	}
}
```

Листинг А.6 – Дисплей

```
// Display
namespace Display {
	int width;
	int height;

	void initialize() {
		width  = GetSystemMetrics(SM_CXSCREEN);
		height = GetSystemMetrics(SM_CYSCREEN);
	}
}
```

Листинг А.7 – Рабочее окно

```
// Declaration WndProc
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

namespace Window {
	HGLRC     hRC       = nullptr; // Permanent rendering context
	HDC       hDC       = nullptr; // Private GDI device context
	HWND      hWnd      = nullptr; // Window handle
	HINSTANCE hInstance = nullptr; // Application instance

	int    x;
	int    y;
	int    width;
	int    height;
	string title;
	bool   hasFocus;
	bool   isFullscreen;

	// Move
	void move(int x, int y) {
		SetWindowPos(hWnd, hWnd, x, y, width, height, SWP_NOZORDER | SWP_NOSIZE);
	}

	// Resize
	void resize(int width, int height) {
		RECT client, window;
		GetClientRect(hWnd, &client);
		GetWindowRect(hWnd, &window);

		SetWindowPos(hWnd, hWnd, x, y,
			width  + window.right  - window.left - client.right,
			height + window.bottom - window.top  - client.bottom,
			SWP_NOZORDER | SWP_NOMOVE);
	}

	// Update
	void update(int width, int height) {
		glViewport(0, 0, width, height);
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();

		// Calculate window aspect ratio
		gluPerspective(Camera::fov, (float)width / height, Camera::zNear, Camera::zFar);
		glMatrixMode(GL_MODELVIEW);
		glLoadIdentity();
	}

	// Initialization
	void initialize() {
		glClearColor(0.0, 0.0, 0.0, 0.0);
		glClearDepth(1.0);

		glShadeModel(GL_SMOOTH);
		glPolygonMode(GL_BACK, GL_LINE);
		//glPolygonMode(GL_FRONT, GL_LINE);
		glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
		glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
		glAlphaFunc(GL_GREATER, (float)0.01);
		glDepthFunc(GL_LEQUAL);

		constexpr GLenum caps[] = {
			//GL_CULL_FACE,
			GL_DEPTH_TEST,
			GL_ALPHA_TEST,
			GL_BLEND,
			GL_TEXTURE_2D
```

```
		};
		for (int i = 0; i < 4; i++)
			glEnable(caps[i]);
	}

	// Set title
	void setTitle(string title) {
		Window::title = title;
		SetWindowText(hWnd, title.c_str());
	}

	// Destroy
	void destroy() {
		if (isFullscreen) {
			ChangeDisplaySettings(nullptr, 0);
		}

		if (hRC) {
			if (!wglMakeCurrent(nullptr, nullptr)) {
				showMessage("Release of DC and RC failed.");
			}
			if (!wglDeleteContext(hRC)) {
				showMessage("Release RC failed.");
			}
		}

		if (hDC && !ReleaseDC(hWnd, hDC)) {
			showMessage("Release DC Failed.");
		}
		if (hWnd && !DestroyWindow(hWnd)) {
			showMessage("Could not release hWnd.");
		}
		if (!UnregisterClass("OpenGL", hInstance)) {
			showMessage("Could not unregister class.");
		}

		hRC       = nullptr;
		hDC       = nullptr;
		hWnd      = nullptr;
		hInstance = nullptr;
	}

	// Create
	bool create(string _title, bool _isFullscreen, int _x, int _y, int _width, int _height)
{
		title        = _title;
		isFullscreen = _isFullscreen;

		// Fullscreen or not
		if (!isFullscreen) {
			x      = _x;
			y      = _y;
			width  = _width;
			height = _height;
		} else {
			x      = 0;
			y      = 0;
			width  = Display::width;
			height = Display::height;
		}

		// Grab an instance for window
		hInstance = GetModuleHandle(nullptr);

		// Window class
		WNDCLASS wc = {
			CS_HREDRAW | CS_VREDRAW | CS_OWNDC, // Redraw on move, and own DC for
window
```

```
                        (WNDPROC) WndProc,                  // WndProc handles messages
                        0,                                  // No extra window data
                        0,                                  // No extra window data
                        hInstance,                          // Instance
                        LoadIcon(nullptr, IDI_WINLOGO),     // Default icon
                        LoadCursor(nullptr, IDC_ARROW),     // Arrow pointer
                        nullptr,                            // No background
                        nullptr,                            // No menu
                        "OpenGL"                            // Class name
                };

                // Attempt to register the window class
                if (!RegisterClass(&wc)) {
                        showMessage("Failed to register the window class.");
                        return false;
                }

                // Attempt fullscreen mode
                if (isFullscreen) {
                        DEVMODE dmScreenSettings;
// Device mode
                        memset(&dmScreenSettings, 0, sizeof(dmScreenSettings));
// Clear memory
                        dmScreenSettings.dmSize       = sizeof(dmScreenSettings);
// Size of DEVMODE
                        dmScreenSettings.dmPelsWidth  = width;
// Width
                        dmScreenSettings.dmPelsHeight = height;
// Height
                        dmScreenSettings.dmBitsPerPel = 32;
// Bits per pixel
                        dmScreenSettings.dmFields     = DM_BITSPERPEL | DM_PELSWIDTH |
DM_PELSHEIGHT; // Fields

                        // Try to set selected mode and get results
                        if (ChangeDisplaySettings(&dmScreenSettings, CDS_FULLSCREEN) !=
DISP_CHANGE_SUCCESSFUL) {
                                // Set it back
                                x      = _x;
                                y      = _y;
                                width  = _width;
                                height = _height;
                                isFullscreen = false;
                        }
                }

                // Choose window styles
                DWORD dwExStyle; // Window extended style
                DWORD dwStyle;   // Window style
                if (isFullscreen || PRGRM_NOBORDER) {
                        dwExStyle = WS_EX_APPWINDOW;
                        dwStyle   = WS_POPUP;
                } else {
                        dwExStyle = WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;
                        dwStyle   = WS_OVERLAPPEDWINDOW;
                        if (!PRGRM_MINIMIZE) dwStyle &= ~WS_MINIMIZEBOX;
                        if (!PRGRM_MAXIMIZE) dwStyle &= ~WS_MAXIMIZEBOX;
                }

                // Adjust window to true requested size
                RECT windowRect = { 0, 0, width, height };
                AdjustWindowRectEx(&windowRect, dwStyle, false, dwExStyle);

                if (!(hWnd = CreateWindowEx(
                        dwExStyle,                              // Extended style
                        "OpenGL",                               // Class name
                        title.c_str(),                          // Title
                        WS_CLIPSIBLINGS | WS_CLIPCHILDREN |     // Required style
```

```
                        dwStyle,                                // Selected style
                        x, y,                                   // Position
                        windowRect.right  - windowRect.left,    // Calculate width
                        windowRect.bottom - windowRect.top,     // Calculate height
                        nullptr,                                // No parent
                        nullptr,                                // No menu
                        hInstance,                              // Instance
                        nullptr)))                              // Do not pass anything to
WM_CREATE
                {
                        destroy();                              // Reset the display
                        showMessage("Window creation error."); // Error
                        return false;                           // Return false
                }

                // Pixel format descriptor
                static PIXELFORMATDESCRIPTOR pfd = {
                        sizeof(pfd),         // Size of pfd
                        1,                   // Version
                        PFD_DRAW_TO_WINDOW | // Format must support Window
                        PFD_SUPPORT_OPENGL | // Format must support OpenGL
                        PFD_DOUBLEBUFFER,    // Must support double buffering
                        PFD_TYPE_RGBA,       // Request an RGBA format
                        32,                  // Color bits
                        0, 0, 0, 0, 0, 0,    // RGB bits ignored
                        0, 0,                // Alpha bits ignored
                        0,                   // No accumulation buffer
                        0, 0, 0, 0,          // Accumulation bits ignored
                        16,                  // 16-bit Z-Buffer
                        0,                   // No stencil buffer
                        0,                   // No auxiliary buffer
                        PFD_MAIN_PLANE,      // Main drawing layer
                        0,                   // Reserved
                        0, 0, 0              // Layer masks ignored
                };

                // Results after searching for a match
                GLuint pixelFormat;

                // Check errors
                if (hDC = GetDC(hWnd)) {
                        if (pixelFormat = ChoosePixelFormat(hDC, &pfd)) {
                                if (SetPixelFormat(hDC, pixelFormat, &pfd)) {
                                        if (hRC = wglCreateContext(hDC)) {
                                                if (wglMakeCurrent(hDC, hRC)) {
                                                        ShowWindow(hWnd, SW_SHOW); // Show the
window
                                                        SetForegroundWindow(hWnd); // Higher

priority
                                                        SetFocus(hWnd);            // Set keyboard

focus
                                                        update(width, height);     // Perspective
                                                        initialize();              // Other things
                                                        return true;               // Good!
                                                } else showMessage("Can't activate the GL rendering
context.");
                                        } else showMessage("Can't create a GL rendering context.");
                                } else showMessage("Can't set the pixelFormat.");
                        } else showMessage("Can't find a suitable pixelFormat.");
                } else showMessage("Can't create a GL device context.");

                // An error
                destroy();
                return false;
        }

        // Change window mode
        void changeFullscreen() {
```

```
			destroy();
			create(title, !isFullscreen, x, y, width, height);
		}
	}

	// Window procedure
	LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
		// Check message
		switch (uMsg) {
			// Key pressed
		case WM_KEYDOWN:
			if (!Keyboard::isHeld[Keyboard::lastPressedKey = wParam]) {
				Keyboard::isHeld[wParam] = Keyboard::isPressed[wParam] = true;
			}
			return 0;

			// Key released
		case WM_KEYUP:
			Keyboard::isHeld[wParam]     = false;
			Keyboard::isReleased[wParam] = true;
			return 0;

			// Mouse moved
		case WM_MOUSEMOVE:
			Mouse::x = LOWORD(lParam);
			Mouse::y = HIWORD(lParam);
			return 0;

			// Mouse wheel rotated
		case WM_MOUSEWHEEL:
			Mouse::isWheelRotated[HIWORD(wParam) == WHEEL_DELTA ? MOUSE_WHEEL_UP :
	MOUSE_WHEEL_DOWN] = true;
			return 0;

			// Left mouse button pressed
		case WM_LBUTTONDOWN:
			Mouse::isHeld[MOUSE_LEFT] = Mouse::isPressed[MOUSE_LEFT] = true;
			return 0;

			// Right mouse button pressed
		case WM_RBUTTONDOWN:
			Mouse::isHeld[MOUSE_RIGHT] = Mouse::isPressed[MOUSE_RIGHT] = true;
			return 0;

			// Middle mouse button pressed
		case WM_MBUTTONDOWN:
			Mouse::isHeld[MOUSE_MIDDLE] = Mouse::isPressed[MOUSE_MIDDLE] = true;
			return 0;

			// Left mouse button released
		case WM_LBUTTONUP:
			Mouse::isHeld[MOUSE_LEFT]     = false;
			Mouse::isReleased[MOUSE_LEFT] = true;
			return 0;

			// Right mouse button released
		case WM_RBUTTONUP:
			Mouse::isHeld[MOUSE_RIGHT]     = false;
			Mouse::isReleased[MOUSE_RIGHT] = true;
			return 0;

			// Middle mouse button released
		case WM_MBUTTONUP:
			Mouse::isHeld[MOUSE_MIDDLE]     = false;
			Mouse::isReleased[MOUSE_MIDDLE] = true;
			return 0;

			// Window focus
```

```
	case WM_ACTIVATE:
		Window::hasFocus = !HIWORD(wParam);
		return 0;

		// System command
	case WM_SYSCOMMAND:
		if (wParam == SC_SCREENSAVE || wParam == SC_MONITORPOWER)
			return 0;
		break;

		// Window resized
	case WM_SIZE:
		Window::update(Window::width = LOWORD(lParam), Window::height = HIWORD(lParam));
		return 0;

		// Window moved
	case WM_MOVE:
		Window::x = LOWORD(lParam);
		Window::y = HIWORD(lParam);
		return 0;

		// Window close
	case WM_CLOSE:
		Engine::prgrm = false;
		return 0;
	}

	// Pass all unhandled messages to DefWindowProc
	return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

Листинг А.8 – Объекты программы

```
// With all objects named as 'object'
#define with(object)                                  \
for (OBJECT_##object* object = ::object;              \
    object > (OBJECT_##object*)0x0000FFFF;            \
    object = (OBJECT_##object*)object->PRGRM_right) \
    if (((Engine::Object*)object)->PRGRM_id() !=    \
              ObjectId::object)                             \
        break;                                        \
    else if (object->isActive)

// With single object named as 'object' with 'id'
#define withId(object, id) \
for (OBJECT_##object* object = (OBJECT_##object*)::object->getObjectById(id); \
    object != nullptr; object = nullptr)

// is object exists? macros
#define objectExists(object) (object > (Engine::Object*)0x0000FFFF)

// Cast & get address for the 4th parameter
#define objectCreate(x, y, z, object) ((OBJECT_##object*)_objectCreate(x, y, z,
(Engine::Object**)&object))

// Engine
namespace Engine {
       float x;
       float y;
       float z;

       // Object
       class Object {
       public:
              Object* PRGRM_left = nullptr;
              Object* PRGRM_right;

              float x;
              float y;
              float z;

              char zIndex       = 0; // delete later
              char id           = -1;
              bool isActive     = true;
              bool isPersistent = false;
              bool isSolid      = false; // delete later

              // Constructor
              Object() {
                     // Set position
                     x = Engine::x;
                     y = Engine::y;
                     z = Engine::z;

                     // Return it back to _objectCreate() function
                     object = this;
              }

              // Destructor
              void destroy(Object** ptr) {
                     // Change first-same-object pointer
                     if (*ptr == this) {
                            if (PRGRM_right != nullptr) if (PRGRM_right->PRGRM_id() ==
PRGRM_id()) *ptr = PRGRM_right;
                            if (*ptr == this) *ptr = (Object*)PRGRM_id();
                     }

                     // Change neighbours links to each other
```

```
		(PRGRM_left != nullptr ? PRGRM_left->PRGRM_right : objectList) =
PRGRM_right;
		if (PRGRM_right != nullptr) PRGRM_right->PRGRM_left = PRGRM_left;

		// Delete
		delete this;
	}

	// Destroy
	void destroy() {
		isActive = false;
	}

	// Get object with needed id
	Object* getObjectById(char id) {
		Object* obj = this;
		while (objectExists(obj)) {
			if (PRGRM_id() != obj->PRGRM_id())
				break;
			if (obj->isActive && obj->id == id)
				return obj;
			obj = obj->PRGRM_right;
		}
		return nullptr;
	}

	// Get id that is not used by this kind of object
	char getId() {
		int idNeed = -1;
		while (getObjectById(++idNeed) != nullptr);
		return idNeed;
	}

	int objectCount() {
		int count = 0;
		Object* obj = this;
		while (objectExists(obj)) {
			if (PRGRM_id() != obj->PRGRM_id())
				break;
			if (obj->isActive)
				count++;
			obj = obj->PRGRM_right;
		}
		return count;
	}

	// Insert object to the object list
	void PRGRM_insert(Object* object) {
		PRGRM_right = object;

		// If there is the same object giving a place before him
		if (objectExists(PRGRM_right)) {
			// Put this new object before him
			PRGRM_left = PRGRM_right->PRGRM_left;
		} else {
			// Find a place in whole list to insert this new object
			PRGRM_right = objectList;
			while (PRGRM_right != nullptr) {
				if (zIndex > PRGRM_right->zIndex) {
					PRGRM_left  = PRGRM_right;
					PRGRM_right = PRGRM_right->PRGRM_right;
				} else break;
			}
		}

		// Change neighbours links to this new object
		(PRGRM_left != nullptr ? PRGRM_left->PRGRM_right : objectList) = this;
		if (PRGRM_right != nullptr) PRGRM_right->PRGRM_left = this;
```

```
		}

		// Move at position
		void moveTo(Object* obj) {
			if (objectExists(this)) {
				if (objectExists(obj)) {
					x = obj->x;
					y = obj->y;
					z = obj->z;
				} else showMessage("moveTo error obj");
			} else showMessage("moveTo error this");
		}

		// Other virtual funcs
		virtual void     PRGRM_kill()     = 0;
		virtual void     PRGRM_step()     = 0;
		virtual void     PRGRM_draw()     = 0;
		virtual ObjectId PRGRM_id() const = 0;
	} *objectList = nullptr, *object = nullptr;

	// Delete everything
	void deleteObjects() {
		Object* ptr = objectList;
		while (ptr != nullptr) {
			Object* del = ptr;
			ptr = ptr->PRGRM_right;
			del->PRGRM_kill();
		}
	}

	// Global step
	void step() {
		// Step active
		Object* ptr = objectList;
		while (ptr != nullptr) {
			if (ptr->isActive) ptr->PRGRM_step();
			ptr = ptr->PRGRM_right;
		}

		// Delete inactive
		ptr = objectList;
		while (ptr != nullptr) {
			Object* del = ptr;
			ptr = ptr->PRGRM_right;
			if (!del->isActive) del->PRGRM_kill();
		}

		// Reset mouse & keyboard
		Keyboard::reset();
		Mouse::reset();
	}

	// Global draw
	void draw() {
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		// Draw
		Object* ptr = objectList;
		while (ptr != nullptr) {
			ptr->PRGRM_draw();
			ptr = ptr->PRGRM_right;
		}

		SwapBuffers(Window::hDC);
	}
};
```

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
«Институт космических и информационных технологий»
Кафедра «Высокопроизводительных вычислений»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Д.А. Кузьмин
подпись инициалы, фамилия
«____» _____20 __ г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Система поиска решений на базе открытых данных

09.04.01 «Информатика и вычислительная техника»

09.04.01.01 «Высокопроизводительные вычислительные системы»

Научный руководитель	подпись, дата	к.т.н., доцент должность, ученая степень	Д.А. Кузьмин инициалы, фамилия
Выпускник	подпись, дата		А.Н. Хохлов инициалы, фамилия
Нормоконтролер	подпись, дата		В.И. Иванов инициалы, фамилия
Рецензент	подпись, дата	к.т.н., доцент должность, ученая степень	Ф.А. Казаков инициалы, фамилия

Красноярск 2018