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

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2017г.

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

Подсистема обработки и анализа электрокардиограмм

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2017.61.ПЗ ВКР

Руководитель работы, доцент
_____/Т.Ю. Оленчикова
«____»_____2017 г.
Автор работы
Студент группы ЕТ-484
_____/ М.Ю. Усенко
«____»_____2017 г.
Нормоконтролер, доцент
_____/Т.Ю. Оленчикова
«____»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Усенко М. Ю. Подсистема обработки и анализа электрокардиограмм. –
Челябинск: ЮУрГУ, ЕТ-484, 53 с., 39 ил., 8 табл., библиогр. список – 15 наим., 1 прил.

Целью данной работы – разработать подсистему для автоматической обработки и анализа кардиограмм, которая может применяться в медицинских учреждениях для упрощения и ускорения работы медицинских работников. В рамках работы были приведены алгоритмы обработки электрокардиограмм, их анализа, спроектирована база данных и разработан интерфейс приложения. На основе чего была спроектирована программная система, выполнена ее тестирование и отладка.

В первом разделе подробно рассматривается предметная область, описаны ключевые параметры электрокардиограммы и их значение. Описаны их искажения при заболеваниях. Описаны пакеты и языки для цифровой обработки сигналов, а также способы классификации данных.

Второй раздел содержит алгоритмы и формулы, необходимые для обработки и анализа электрокардиограмм.

Третий раздел посвящен разработке подсистемы. В нем приведена диаграмма использования, проектирование базы данных, описание интерфейса программы.

В четвертом разделе приведены результаты экспериментального тестирования программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Сердечно-сосудистые заболевания – одни из самых распространенных и в то же время самых опасных заболеваний. Инфаркты, инсульты, ишемическая болезнь сердца – по данным Роскомстата от них страдают порядка 56.6% россиян. Притом доказано, что у примерно у половины больных даже при регулярном обследовании на электрокардиограммах (далее по тексту ЭКГ) не наблюдается значительных изменений. Кроме того разные школы кардиографии по-разному подходят к чтению ЭКГ, допуская разные трактовки показателей. Значит, что традиционный протокол ЭКГ дает недостаточно информации. Поэтому требуются новые методы обработки ЭКГ.

Для решения этой проблемы медики предлагают цифровую обработку ЭКГ. Поскольку кардиограмма – это электрический сигнал, набор значений, его можно представить в виде функции и, анализируя ее, делать выводы о состоянии пациента и патологиях его сердечно-сосудистой системы. В современной медицине для анализа ЭКГ применяются методы оценки первой и второй производной функции ЭКГ. Анализируя экстремумы функции и промежуток между ними можно выявлять P,Q,R,S,T зубцы – этим кардиологи некоторое время назад занимались вручную - которые соответствуют различным фазам сердечного цикла. В зависимости от интервалов, частоты и прочих параметров можно делать выводы о состоянии здоровья пациента. Именно эти функции (анализ интервалов и выявление отклонений) предлагается выполнять программно.

Целью данной работы является разработка подсистемы для обработки и анализа электрокардиограмм и автоматической диагностики заболеваний.

В первом разделе подробно рассмотрена предметная область. В нем описывается структура кардиограмм, влияние состояние здоровья на их форму. Описаны нормальные показатели и их изменения при различных заболеваниях. Рассмотрены различные алгоритмы классификации.

Во втором разделе описаны алгоритмы обработки и анализа ЭКГ, алгоритм построения дерева решений.

В третьем разделе разработана диаграмма классов, описан процесс проектирования базы данных, спроектирован интерфейс программы.

Четвертый раздел описывает испытания нашей программы на реальных данных, взятых из открытой базы медицинских физиологических сигналов.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Анализ требований к подсистеме исследования ЭКГ. Постановка задачи

Целью данного дипломного проекта является разработка подсистемы для компьютерного анализа ЭКГ. Программа должна выполнять предварительную обработку оцифрованной ЭКГ, определять параметры кардиограммы, и на их основании делать предположения о состоянии здоровья пациента: находится ли он в норме, или присутствуют патологии. Поскольку ЭКГ представляет из себя величину изменяющуюся во времени, то есть сигнал, задача анализа ЭКГ сводится к цифровой обработке сигнала (рисунок 1.1).

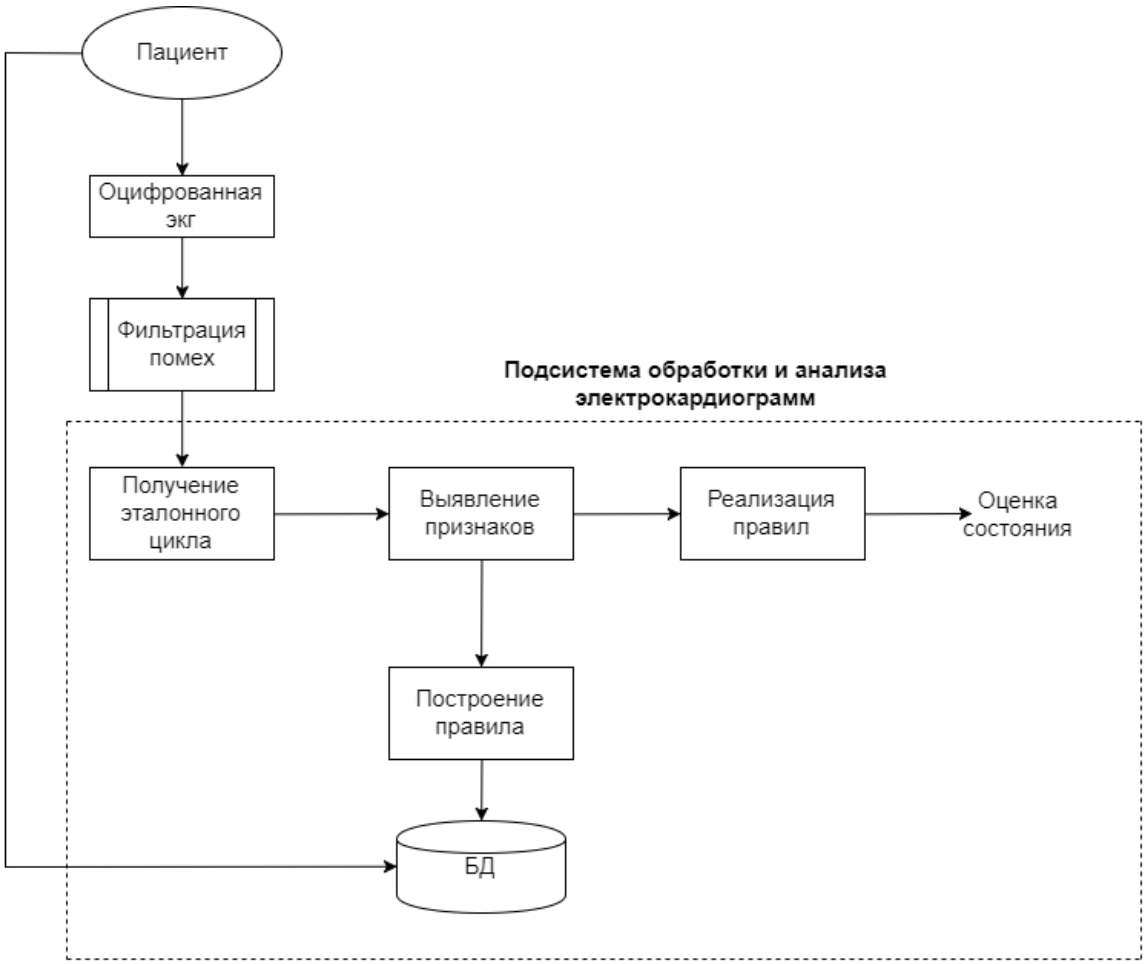

Рисунок 1.1 – Функциональная схема подсистемы

Программа может найти широкое применение в области кардиологии – с ее помощью можно эффективно диагностировать и прогнозировать у пациентов сердечно-сосудистые заболевания.

Данные ЭКГ хранятся в базе данных со всей сопутствующей информацией: информация о пациенте, дата снятия кардиограммы и прочие.

Чтобы достичь поставленной цели необходимо решить следующие задачи:

- изучить существующие методы обработки ЭКГ;
- спроектировать базу данных;
- изучить существующие технологии и инструменты для цифровой обработки сигналов;
- изучить методы принятия решений;
- разработать и отладить модуль выявления признаков;
- разработать модуль принятия решений;
- провести обучение системы

1.2 Анализ существующих пакетов и систем для цифровой обработки сигналов

Аналоговый сигнал представляет из себя изменяющуюся во времени величину. Аналоговый сигнал можно оцифровать, то есть подвергнуть дискретизации по времени и квантованию по уровню. После оцифровки его можно обрабатывать компьютерными методами – этот процесс называется цифровой обработкой сигнала. Цифровая обработка сигнала реализована в различных математических системах и пакетах.

Matlab

Пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете. Язык Matlab является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования.

Matlab предоставляет пользователю большое количествофункций для анализа данных, покрывающие практически все области математики, в частности:

- матрицы и линейная алгебра — алгебра матриц, линейные уравнения, собственные значения и векторы, сингулярности, факторизация матриц и другие;
- многочлены и интерполяция — корни многочленов, операции над многочленами и их дифференцирование, интерполяция и экстраполяция кривых и другие;
- математическая статистика и анализ данных — статистические функции, статистическая регрессия, цифровая фильтрация, быстрое преобразование Фурье и другие;
- обработка данных — набор специальных функций, включая построение графиков, оптимизацию, поиск нулей, численное интегрирование (в квадратурах) и другие;
- дифференциальные уравнения — решение дифференциальных и дифференциально-алгебраических уравнений, дифференциальных уравнений с запаздыванием, уравнений с ограничениями, уравнений в частных производных и другие;

- разреженные матрицы — специальный класс данных пакета Matlab, использующийся в специализированных приложениях;
- целочисленная арифметика — выполнение операций целочисленной арифметики в среде Matlab.

Также Matlab поддерживает множество внешних интерфейсов, с помощью которых организовано взаимодействие с другими программами и периферийными устройствами. Среди возможных интерфейсов есть COM (Component Object Model), с его помощью можно манипулировать как клиентами и серверами. Пакет Matlab в MicrosoftWindows предоставляет доступ к программной платформе .NET Framework. Пакет Matlab содержит функции, которые позволяют ему получать доступ к другим приложениям среды Windows, равно как и этим приложениям получать доступ к данным Matlab, посредством технологии динамического обмена данными (DDE). Интерфейс Matlab, относящийся к общим DLL, позволяет вызывать функции, находящиеся в обычных динамически подключаемых библиотеках, прямо из Matlab. Эти функции должны иметь С-интерфейс.

Конкретно для цифровой обработки сигнала существует расширение DSP SystemToolbox. Оно предназначено для моделирования систем обработки сигналов. Например: радаров, сонаров, систем управления, медицинских приборов.

В DSP SystemToolbox(рисунок 1.2) реализовано большинство существующих алгоритмов для ЦОС. Так же присутствует инструментарий для анализа спектра сигнала, логический анализатор и визуализации результата. Используя DSP SystemToolbox совместно с MatlabCoder и SimulinkCoder можно генерировать код наС/С++.

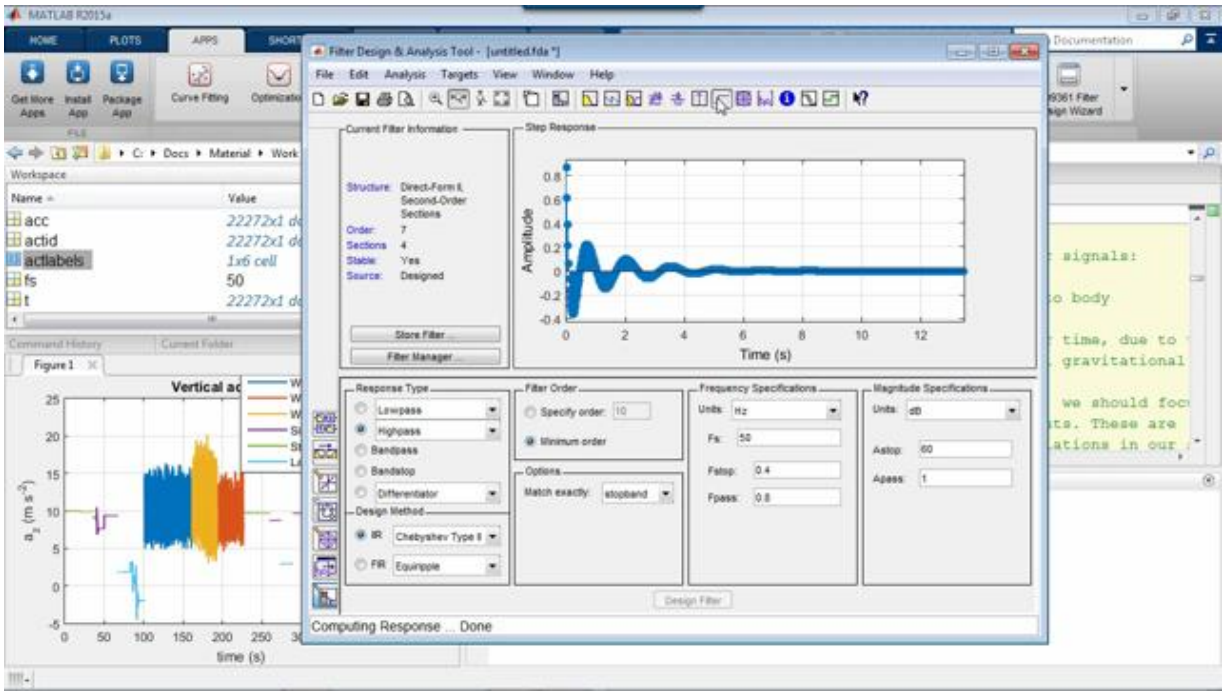

Рисунок 1.2 – Matlab с расширением DSP

Для работы с базами данных предусмотрено расширение DatabaseToolbox, позволяющее организовать работу с базами данных Oracle, Microsoft SQL Server, Sybase, MicrosoftAccess, Informix, Ingres, поддерживает выполнение SQL-запросов из Matlab, а также работу с несколькими базами данных одновременно.

VisSim

Визуальный язык программирования, предназначенный для моделирования динамических систем, а также проектирования, базирующегося на моделях, для встроенных микропроцессоров. VisSim сочетает в себе характерный для Windows интуитивный интерфейс для создания блочных диаграмм и мощное моделирующее ядро.

В реальности любые системы и объекты состоят из различных функциональных элементов. Исходными данными для моделирования в VisSim является структурно-функциональная схема заданной системы и описывающие их уравнения. Вместо таких уравнений могут быть заданы операторы или функции, характеризующие отдельные элементы моделируемой системы, например, передаточнные функции для линейных элементов и статические характеристики для нелинейных элементов.

Сама система и объекты в среде Vissim представлены в виде набора различных блоков, логических, вычислительных, функциональных, цифровых фильтров. Выходной сигнал блока может быть подан на вход следующего – таким образом организовано взаимодействие между блоками и симуляция работы системы в целом (рисунок 1.3).

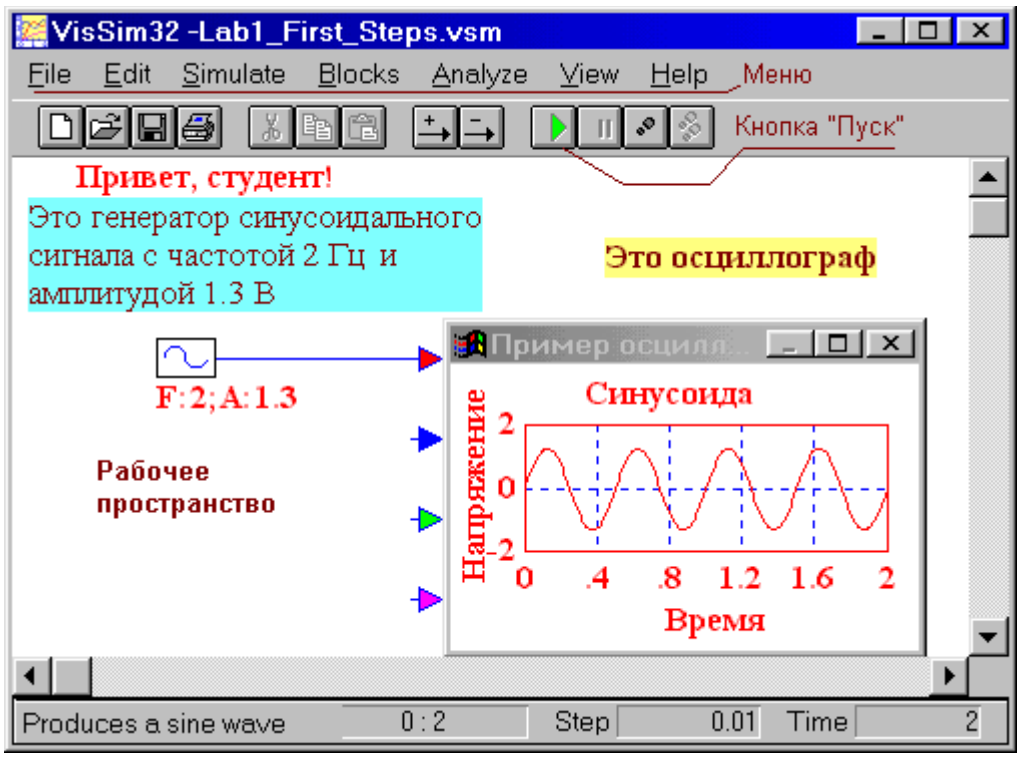

Рисунок 1.3 – Интерфейс VisSim 3.2

Основными областями моделирования являются аэрокосмическая, биологическая и медицинская, электродвигатели, электрические, гидравлические, механические, тепловые процессы, эконометрика.

VisSim/C-Code — расширение для генерации кода на Си, автоматически преобразует модели VisSim в понятный и эффективный код на ANSI C. Код можно скомпилировать и запустить на любой платформе, имеющей компилятор для языка Си.

Однако, Vissim не обладает специализированными инструментами для работы с цифровыми сигналами и подключением к БД.

Python

Строго говоря, Python не является пакетом для обработки сигналов. Это язык программирования общего назначения, стандартная библиотека которого включает в себя многие полезные функции (рисунок 1.4). Помимо стандартных библиотек, существует ряд специализированных, с помощью которых можно проводить цифровую обработку сигналов. Например, NumPy для работы с числамии многомерными массивами; SciPy для научных вычислений–поиск минимумов и максимумов функций, вычисление интегралов, работа с сигналами; Matplotlib для визуализации данных: 2Dи 3D графика, графики, диаграммы, спектральные диаграммы.

Для Python принята спецификация программного интерфейса к базам данныхDB-API и разработаны соответствующие этой спецификации пакеты для доступа к различным СУБД: Oracle, MySQL, PostgreSQL, Sybase, Firebird (Interbase), Informix, Microsoft SQL Server и SQLite. На платформе Windows доступ к БД возможен через ADO (ADOdb).

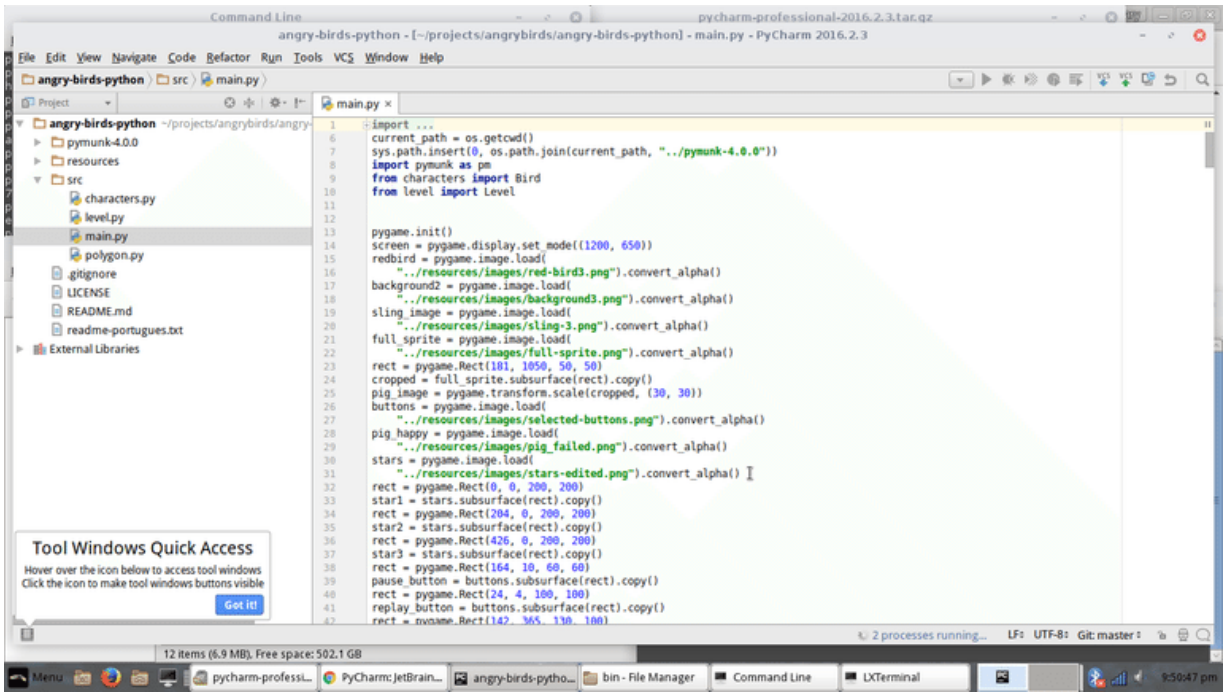

Рисунок 1.4 – Интерфейс IDEPycharm5

AnyLogic

AnyLogic обладает удобным графическим пользовательским интерфейсом, в ней применяется объектно-ориентированный подход, элементы стандарта UML, в качестве языка программирования используется Java. В программе реализованы все три известных подхода к моделированию (рисунок 1.5):

- системная динамика;
- дискретно-событийное (процессное) моделирование;
- агентное моделирование.

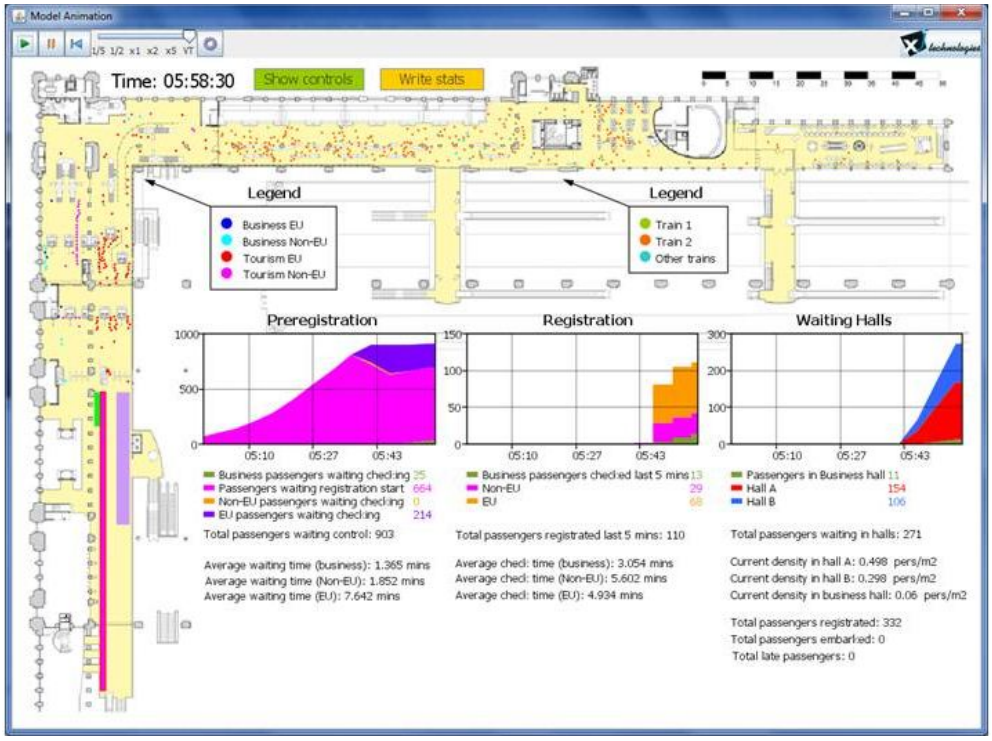

Рисунок 1.5 – Пример моделирования бизнес-процесса в AnyLogic

Так же есть возможность свободно комбинировать все три метода. В настоящее время AnyLogic применяется в следующих сферах: производство, логистика и цепочки поставок, рынок и конкуренция, бизнес-процессы и сфера обслуживания, здравоохранение и фармацевтика, управление активами и проектами, телекоммуникации и информационные системы, социальные и экологические системы, пешеходная динамика, оборона.

AnyLogic поддерживает возможность моделировать процессы графически (как это делал VisSim со своими блоками). А именно существует 4 метода графического моделирования:

- Stock&FlowDiagrams (диаграмма потоков и накопителей) применяется при разработке моделей, используя метод системной динамики;
- Statecharts (карты состояний) в основном используется в агентных моделях для определения поведения агентов. Но также часто используется в дискретно-событийном моделировании, например для симуляции машинных сбоев;

- Actioncharts (блок-схемы) используется для построения алгоритмов. Применяется в дискретно-событийном моделировании (маршрутизация звонков) и агентном моделировании (для логики решений агента);
- Processflowcharts (процессные диаграммы) основная конструкция, используемая для определения процессов в дискретно-событийном моделировании;

Работа с внешними базами данных в AnyLogic была реализована относительно недавно, в версии 7.2 и только в издании professional.

1.3 Выбор средств разработки

В качестве средства разработки был выбран язык Python, так как он является свободно распространяемым и обладает большим количеством готовых библиотек, в которых уже реализовано большинство необходимых нам. В качестве средства разработки базы данных выбран MySQL, также распространяемый по свободной лицензии.

1.4 Анатомия и физиология сердца, связь с ЭКГ

Прежде чем начать разработку программного продукта, необходимо разобраться с предметной областью, в данном случае с кардиограммами. Сделать это, не поднимая вопросов анатомии, физиологии и кардиографии невозможно, поэтому начнем с них.

Электрокардиограмма – графическая запись, которая отображает работу сердца. ЭКГ фиксирует электрический импульс, который вырабатывает организм, вызывая сокращения сердечной мышцы. По оси x откладывается время, а по y – амплитуда сигнала. В случае, когда импульс направлен к регистрирующему датчику, сигнал считается положительным, в противоположную сторону – отрицательным.

Посмотрев на ЭКГ (рисунок 1.6) можно заметить, что на ней циклически повторяются отрезки одного и того же вида. Каждый такой отрезок (рисунок 1.7) соответствует одному сердечному циклу, то есть одному полноценному сокращению сердечной мышцы.

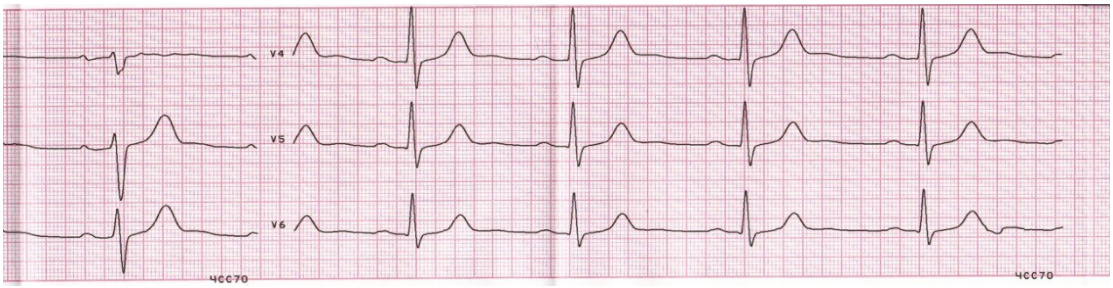

Рисунок 1.6 – Пример ЭКГ

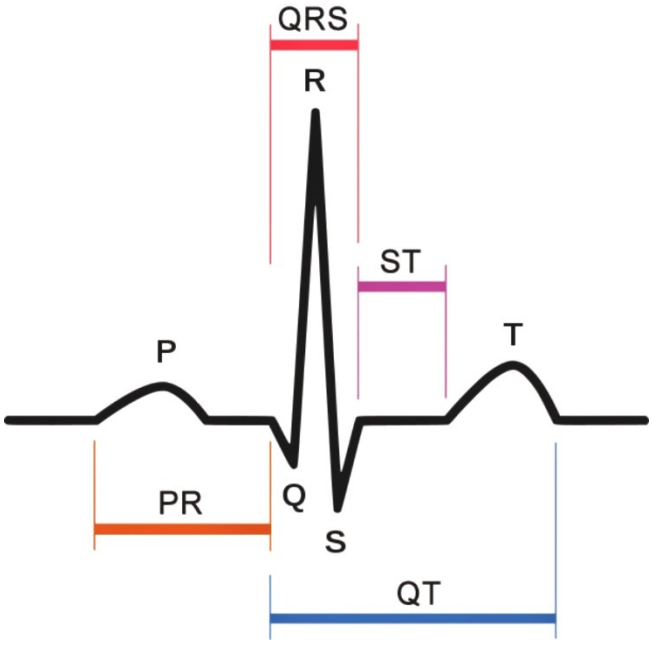

Рисунок 1.7 – Один сердечный цикл

Человеческое сердце состоит из четырех камер (отделов): двух предсердий и желудочков, - последовательное сокращение каждой из них фиксируется на ЭКГ в виде отклонений от изолинии. Такие отклонения называют «зубцами» и именуют латинскими буквами P, Q, R, S, T [1]. Разберем каждый по отдельности.

P-зубец

Ток в сердце поступает через синусовый узел, из-за которого нормальный сердечный ритм называют синусовым. Узел расположен в правом предсердии, так что первым возбуждается именно оно, далее импульс переходит на левое предсердие (рисунок 1.8).

Сигнал от возбуждения каждого предсердия фиксируется. Наслаиваясь, они суммируются и образуют зубец P.

P-Q интервал

Как только импульс покидает предсердия, зубец Рзаканчивается и на графике ЭКГ мы снова видим прямую, то есть отсутствие сигнала [2]. Ток проходит атриовентрикулярное соединение, не вызывая возбуждения прилежащих слоев.

QRS-комплекс

Три пика, Q, Rи Sсовместно образуют QRS-комплекс, который отвечает за сокращение желудочков (рисунок 1.9).

Первой возбуждается межжелудочковая перегородка, образуя зубец Q. Вектор возбуждения при этом направлен вбок от регистрирующего электрода, что дает на ЭКГ отрицательное значение [2].

Далее возбуждается верхушка сердца. Вектор возбуждения направлен прямиком к датчику, что дает на графике самый крупный по амплитуде пик R.

Последним возбуждается основание сердца, вектор возбуждения направлен против датчика, значение отрицательное. На графике образуется пик S.

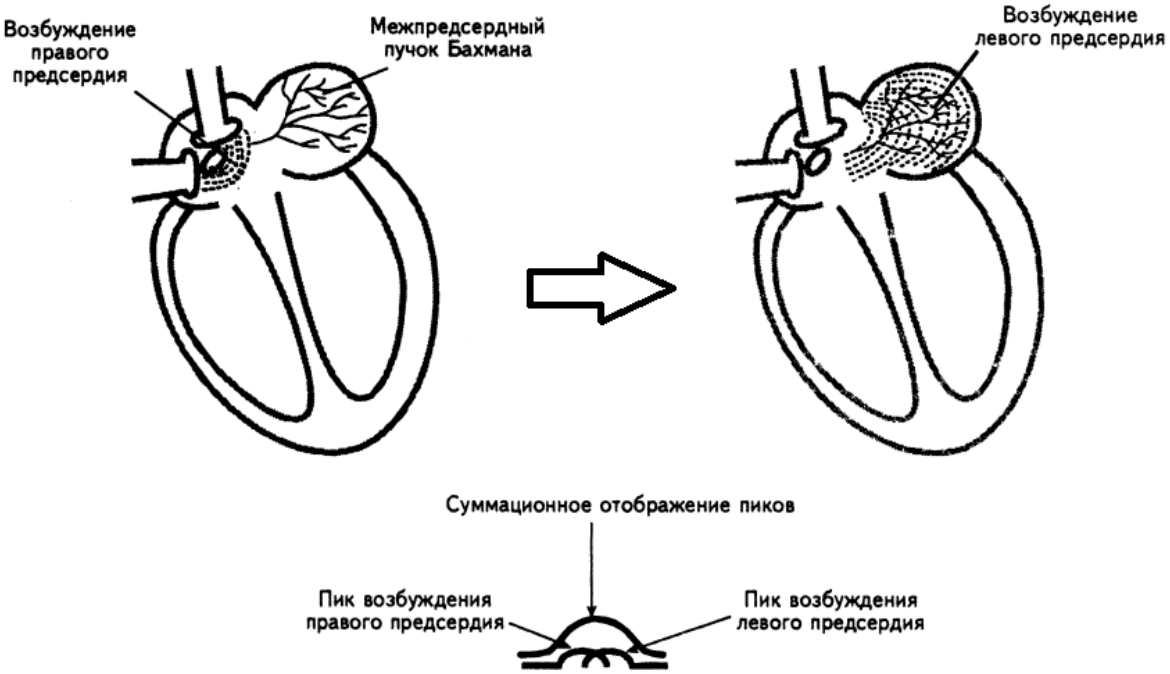

Рисунок 1.8 – Возбуждение предсердий и образование Р-зубца

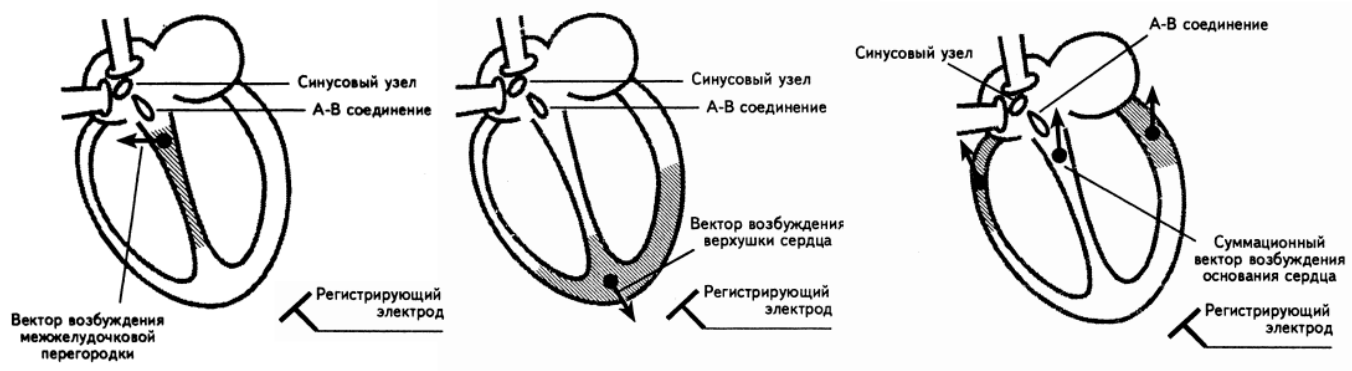

Рисунок 1.9 – Образование Q, Rи Qзубцов

S-T интервал и зубец Т

После возбуждения желудочков начинается процесс реполяризации, то есть восстановления сердца перед началом нового цикла.

J – внутреннее отклонение

Сердечная мышца, миокард, имеет определенную толщину и заключена между двумя стенками: эндокардом и эпикардом, - внутренней и внешней соответственно. Время прохождения импульса от эндокарда до эпикарда имеет свое значение и называется временем внутреннего отклонения[3]. На ЭКГ внутреннее отклонение определяется как время от начала зубца Qдо пика зубца R (рисунок 1.10).

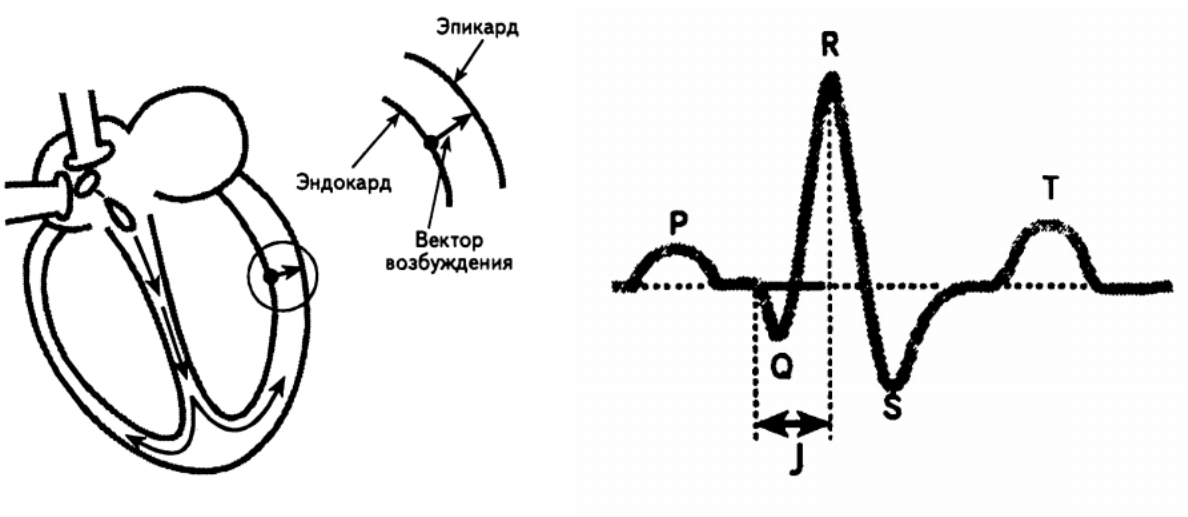

Рисунок 1.10 – Внутреннее отклонение

1.5 Электрокардиографические отведения

При записи ЭКГ регистрируется не сам импульс, а электрическое поле, которое вырабатывает сердце. Для этого на теле пациента размещаются датчики. Все они находятся на одной эквипотенциальной окружности. Так как таких окружностей множество, существует несколько способов расположения датчиков. Каждый такой способ называется отведением.

Существует 12 стандартных (хотя некоторые школы современной кардиологии различают до 42 вариантов), каждое из которых позволяет точнее узнать о работе конкретного отдела сердца. Так, например, I отведение делает акцент на передней стенке сердца, а III на задней.

Однако, концентрация на конкретных отделах сердца приводит к тому, что одна и та же кардиограмма в разных отведениях выглядит по-разному и описанные ранее зубцы получают новые трактовки (рисунок 1.11).

Чтобы унифицировать процесс расшифровки, все измерения показателей в общем производят во IIотведении, которое является результирующим для Iи IIIотведений. А все остальные используются для диагностики конкретных отделов сердца.

1.6 Нормальные значения ЭКГ и отклонения

ЭКГ во IIотведении здорового человека имеет следующие значения.

- Зубец R – 1 mV.
- Высота зубца T и глубина S должны соответствовать 1/3-1/2 зубца R, то есть 0,3-0,5 mV.
- Высота зубца P и глубина Q соответствуют 1/4-1/3 зубца R, то есть 0,25-0,3 mV.
- Время внутреннего отклонения 0,02-0,05с.

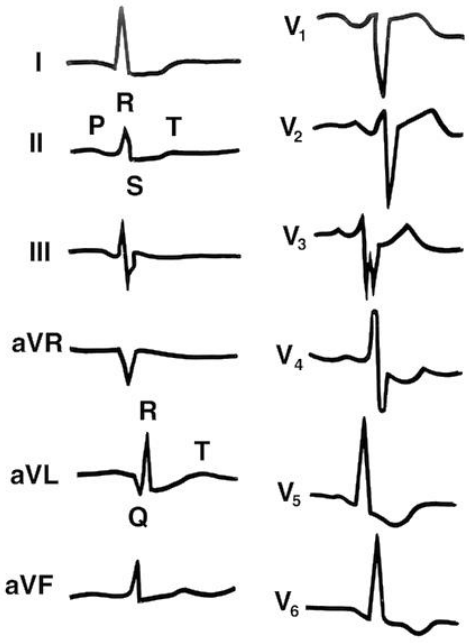

Рисунок 1.11 – Сердечный цикл в 12 стандартных отведениях

Продолжительность различных этапов сердечного цикла имеет одинаковую норму по времени (в терминах кардиологии ее также называют шириной), она равна 10+-2 мс. Такую продолжительность имеет зубец P (каждое из предсердий сокращается за 0,05с), интервал P-Q, комплекс QRS. Соответственно, время реполяризации (восстановления) мышцы равно 30+-2 мс.

Зная показатели ЭКГ здорового человека мы можем определять отклонения от нормы у конкретной кардиограммы, и делать выводы о состоянии здоровья пациента. Рассмотрим некоторые из заболеваний.

Аритмия

При этом заболевании нарушается нормальная ритмика сердца. Нормальной считается частота сокращений 60-90 в минуту (то есть появление зубцов P) и равное временные интервалы между ними. В случае, если интервалы варьируются меньше, чем на 0,12с, говорят о неправильном синусовом ритме[4]. Если разность между наибольшим и наименьшим интервалом превышает 0,12с, то речь идет об аритмии (Рисунок 1.12).

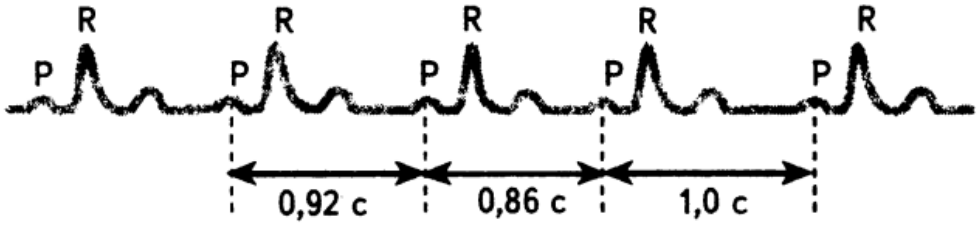

Рисунок 1.12 – Аритмия на ЭКГ

Гипертрофия миокарда

Гипертрофией миокарда называют увеличение сердца, то есть случай, когда сердечная мышца слишком сильно развита. Она больше по размерам и способна создавать куда больше давление.

На ЭКГ это выражается увеличением времени среднего отклонения – расстояние от эндокарда до эпикарда физически больше, и увеличением пика R – сердце развивает куда большее усилие [4]. Если гипертрофировано одно из предсердий, тогда пик P будет ассиметричен, если его вершина смещена влево – увеличено правое предсердие, вправо – левое. Если увеличены оба предсердия – тогда пик P будет симметричным с амплитудой больше нормальной.

Инфаркт миокарда

Инфаркт миокарда – одно из самых опасных, и вместе с тем распространенных сердечно-сосудистых заболеваний. Оно характеризуется отмиранием части сердечной ткани. Чтобы понять, работа какой части сердца нарушена, необходимо просматривать ЭКГ во всех отведениях. Однако во всех случаях, у инфаркта одни и те же признаки (рисунок 1.13).

- Зубец R редуцирован.
- Зубец Q становится патологическим, то есть его длина превышает 0,03с
- Если отведение расположено над областью инфаркта, то сегмент S-T будет расположен выше изолинии. Если отведение на противоположной стороне, сегмент опустится ниже.

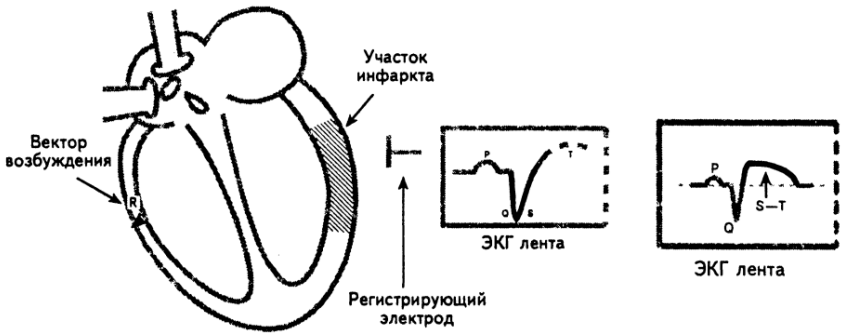

Рисунок 1.13 – Инфаркт миокарда

1.7 Методы принятия решений

Зная нормальные показатели ЭКГ и специфику отдельных заболеваний, нам необходимо определить, присутствует ли на конкретной кардиограмме какое-либо заболевание (или же пациент находится в норме). То есть перед нами стоит задача классификации, где объектом является кардиограмма, а классами - заболевания.

Для исследований были выбраны основные методы классификации, которые используются для обработки разнообразных данных, начиная от экономических сфер и заканчивая медицинской, которая собственно нас и интересует. У каждого метода рассмотрены его плюсы и минусы и, в первую очередь, его погрешность, которая для нас является основным критерием.

Дерево решений

Деревья решений – это способ представления правил в иерархической, последовательной структуре (рисунок 1.14).

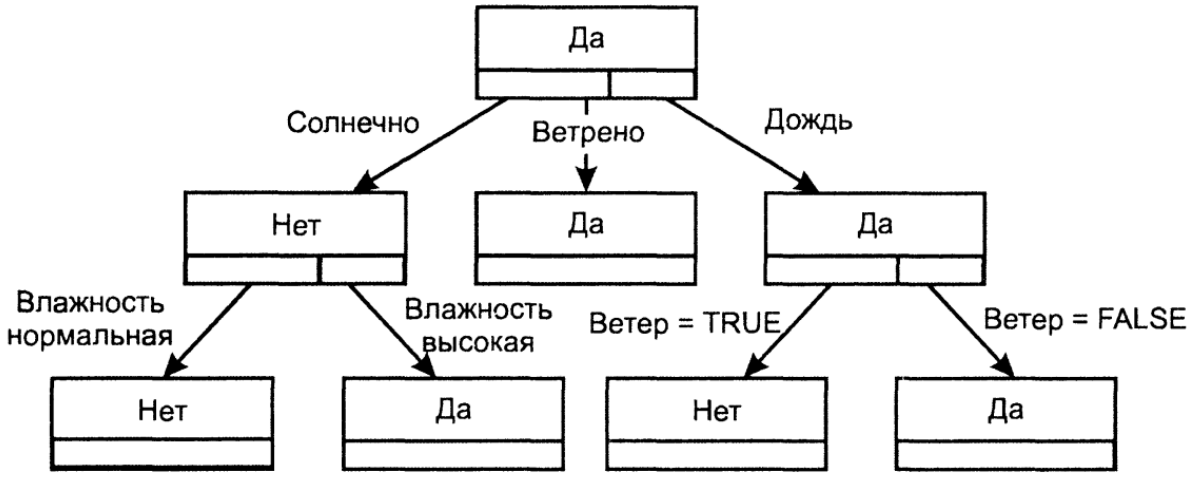

Рисунок 1.14 – Пример дерева решений

Обычно каждый узел дерева включает проверку определенной независимой переменной. Иногда в узле дерева две независимые переменные сравниваются друг с другом или определяется некоторая функция от одной или нескольких переменных.

Если переменная, которая проверяется в узле, принимает категориальные значения, то каждому возможному значению соответствует ветвь, выходящая из узла дерева. Если значением переменной является число, то проверяется, больше или меньше это значение некоторой константы. Иногда область числовых значений разбивают на несколько интервалов. В этом случае выполняется проверка на попадание значения в один из интервалов.

Листья деревьев соответствуют значениям зависимой переменной, т.е. классам. Объект принадлежит определенному классу, если значения его независимых переменных удовлетворяют условиям, записанным в узлах дерева на пути от корня к листу. Соответствующему этому классу.

Если какая-либо независимая переменная классифицируемого объекта не имеет значения, то возникает проблема, связанная с неопределенностью пути, по которому необходимо двигаться по дереву [5]. В некоторых случаях пропущенные значения можно заменять значениями по умолчанию. Если такой подход неприемлем, то необходимо предусмотреть специальные способы обработки таких ситуаций. Другой вариант обработки может быть связан с добавлением специальной ветви к узлу для пропущенных значений.

Деревья решений легко преобразуются в правила. В условную часть таких правил записывается условие, описанное в узлах дерева на пути к листу, в заключительную часть – значение, определенное в листе.

Приведем простой пример, где можно использовать алгоритм дерева решений (рисунок 1.15). Когда человек приходит в банк для получения кредита, то у специалистов работающих там есть алгоритм действий, благодаря которому они решают, выдавать человеку кредит или нет. Часто бывает так, что все эти действия автоматизированы следующим образом:

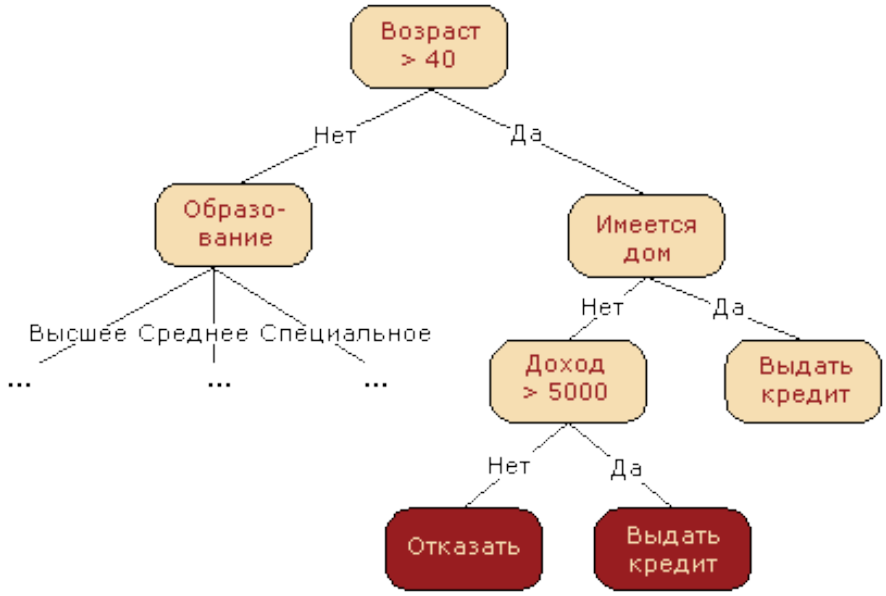

Рисунок 1.15 - Пример выдачи кредита посредством дерева решений

Мало того, что такой алгоритм увеличивает скорость обслуживание клиентов, так еще стоит отметить и то, что он понятен на интуитивном уровне и не требует серьезной подготовки работника. Иными словами в работе этого алгоритма, при его должной реализации разберется любой человек имеющий знания обычного пользователя персонального компьютера.

Итого, можно смело утверждать, что деревья решений являются прекрасным инструментом в системах поддержки принятия решений интеллектуального анализа данных. Они успешно применяются для решения практических задач в следующих областях.

- Банковское дело. Оценка кредитоспособности клиентов банка при выдаче кредитов.
- Промышленность. Контроль над качеством продукции (выявление дефектов), испытания без разрушений (например, проверка качества сварки) и т.д.
- Медицина. Диагностика различных заболеваний.
- Молекулярная биология. Анализ строения аминокислот.

Главным же преимуществом является интуитивность деревьев решений. Классификационная модель, представленная в виде дерева решений, является интуитивной и упрощает понимание решаемой задачи. Результат работы алгоритмов конструирования деревьев решений, в отличие от тех же нейронных

сетей, легко интерпретируется пользователем. Это свойство деревьев решений не только важно при отнесении к определенному классу нового объекта, но и полезно при интерпретации модели классификации в целом. Дерево решений позволяет понять и объяснить, почему конкретный объект относится к тому или иному классу.

Помимо этого можно отметить и то, что деревья решений имеют такую особенность, как быстрое обучение, возможность специальной обработки пропущенных значений, а так же высокую точность, сравнимую с нейронными сетями. Помимо всего прочего алгоритм конструирования дерева решений не требует от пользователя выбора входных атрибутов. На вход алгоритма можно подавать все существующие атрибуты, алгоритм сам выберет наиболее значимые среди них, и только они будут использованы для построения дерева.

Но не все так положительно. Главной проблемой данного алгоритма, является тот факт, что для каждого типа данных предстоит построить новое решение.

Метод «ближайшего соседа»

Метод «ближайшего соседа» относится к классу методов, работа которых основывается на хранении данных в памяти для сравнения с новыми элементами. При появлении новой записи для прогнозирования находятся отклонения между этой записью и подобными наборами данных, и наиболее подобная идентифицируется.

Например, при рассмотрении нового пациента, его атрибуты сравниваются со всеми существующими пациентами данной больницы (возраст, давление и т.д.). Множество «ближайших соседей» пациента поликлиники выбирается на основании ближайшего значения возраста, давления и т.д (рисунок 1.16).

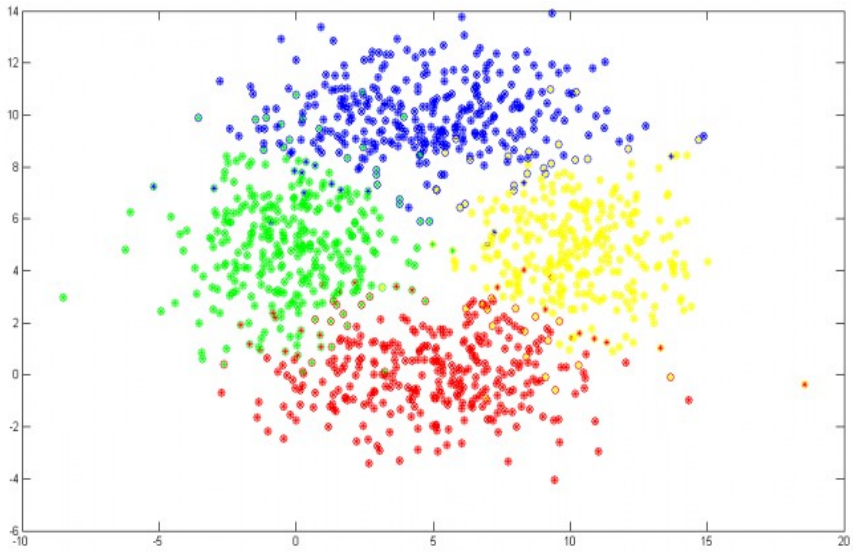

Рисунок 1.16 - Пример работы метода «ближайшего соседа»

При таком подходе используется термин «k-ближайший сосед». Термин означает, что выбирается k «верхних» соседей для их рассмотрения в качества множества «ближайших соседей». Поскольку не всегда удобно хранить все данные, иногда хранится только множество «типичных» случаев. В таком случае используемый метод называют рассуждением по аналогии, рассуждением на

основе аналогичных случаев, рассуждением по прецедентам, т.е. описания ситуации в сочетании с подробным указанием действий, предпринимаемых в данной ситуации.

Подход, основанный на прецедентах, условно можно поделить на следующие этапы:

- сбор подробной информации о поставленной задаче;
- сопоставление этой информации с деталями прецедентов, хранящихся в базе, для выявления аналогичных случаев;
- выбор прецедента, наиболее близкого к текущей проблеме, если это необходимо;
- адаптация выбранного решения к текущей проблеме, если это необходимо;
- проверка корректности каждого вновь полученного решения;
- занесение детальной информации о новом прецеденте в базу прецедентов.

Таким образом, вывод, основанный на прецедентах, представляет собой такой метод анализа данных, который делает заключения относительно данной ситуации по результатам поиска аналогий, хранящихся в базе прецедентов.

Данный метод по своей сути относится к категории «обучения без учителя», т.е. является «самообучающейся» технологией, благодаря чему рабочие характеристики каждой базы прецедентов с течением времени и накоплением примеров улучшаются. Разработка баз прецедентов по конкретной предметной области происходит на естественном для человека языке, следовательно, может быть выполнена наиболее опытными сотрудниками компании – экспертами или аналитиками, работающими в данной предметной области.

Для того, что бы этот метод было более понятен, разберем одну простую задачу. Она заключается в том, что бы правильно классифицировать ирисы (цветы). Имеется набор данных, собранных Р. Фишером, о 150 цветках трех разных классов: IrisSetosa, IrisVersicolour, IrisVirginica, по 50 записей для каждого. Известна длина и ширина чашелистика и лепестка всех этих ирисов.

Для того, что бы ни вдаваться в излишние подробности рассмотрим только два входных поля: длина чашелистика и длина лепестка, а также выходное – класс цветка. Нам потребуется определить класс двух цветок, с определенными значениями длины чашелистика и лепестка (рисунок 1.17).

Очевидно, что первый цветок относится к классу Irissetosa, так как представители этого класса являются его единственными соседями. Что же до второго цветка, то тут все несколько сложнее.

Соседями цветка под номер 2 являются представители сразу двух классов Irisversicolour и Irisvirginica, все они расположены примерно на одном расстоянии от «подопытного» образца, но так, как представителей из класса Irisversicolour больше, то алгоритм отдает предпочтение именно этому классу.

Из плюсов данного метода можно выделить простоту использования полученных результатов, а так же то, что решения не уникальны для конкретной ситуации, их можно использовать для разных типов данных. Стоит отметить и то, что целью поиска является не гарантированно верное решение, а лучшее из возможных.

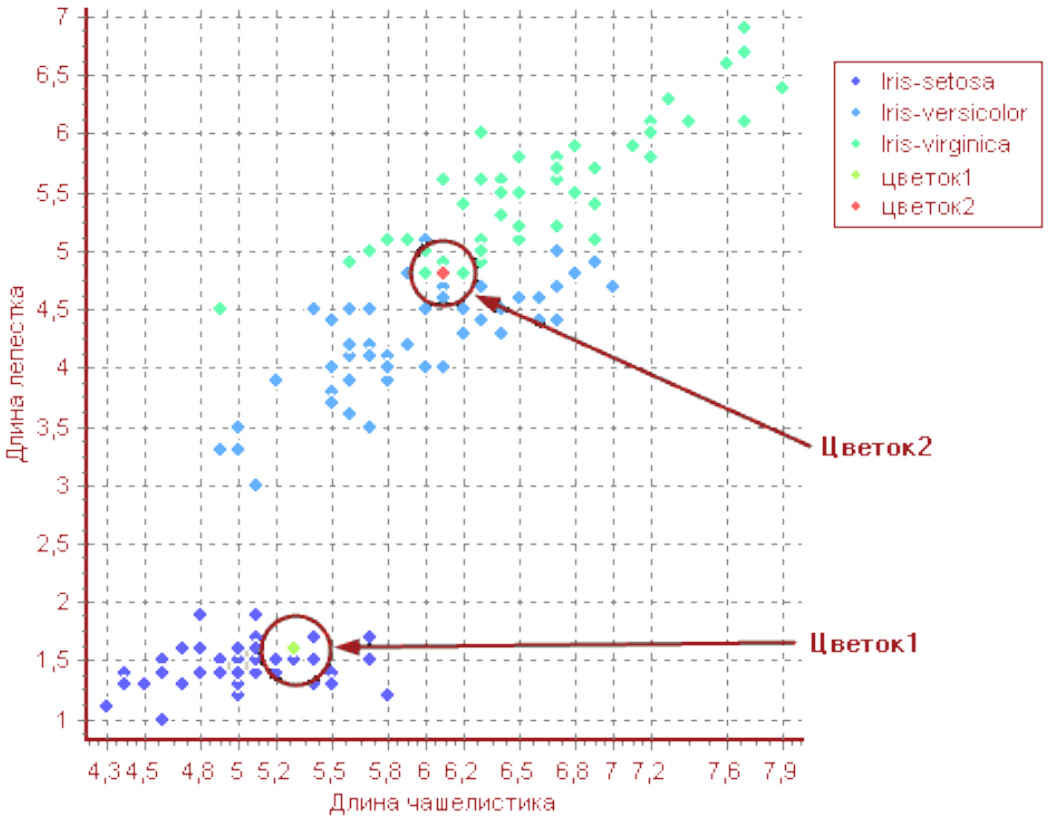

Рисунок 1.17 – Наглядная демонстрация примера работы алгоритма «ближайшего соседа», где нам надо отнести два цветка к одному из трех классов

Одним из недостатков данного метода является то, что он не создает каких-либо моделей или правил, обобщающих предыдущий опыт, - в выборе решения они основываются на всем массиве доступных исторических данных, поэтому невозможно сказать, на каком основании строятся ответы. Не стоит опускать того, что существует сложность выбора меры «близости». От этой меры главным образов зависит объем множества записей, которые нужно хранить в памяти для достижения удовлетворительной классификации или прогноза. Иными словами этот способ хорошо работает для компактных множеств, вроде тех, что мы разобрали выше. Для некомпактных множеств, алгоритм будет давать неправильные результаты.

Стоит упомянуть и то, что при использовании метода возникает необходимость полного перебора обучающей выборки при распознавании, следствие этого – вычислительная трудоемкость. Типичные задачи данного метода – это задачи небольшой размерности по количеству классов и переменных, если же их становится слишком много, то процент погрешности увеличивается.

Регрессионный анализ

Основной особенностью регрессионного анализа, является то, что при его помощи можно получить конкретные сведения о том, какую форму и характер имеет зависимость между исследуемыми переменными.

Решение задачи классификации, с помощью регрессионного анализа, осуществляется следующим образом: линия регрессии делит все множество объектов на два класса, и та часть множества, где значения функции больше нуля, принадлежит к одному классу, а та, где оно меньше нуля – к другому классу (рисунок 1.18).

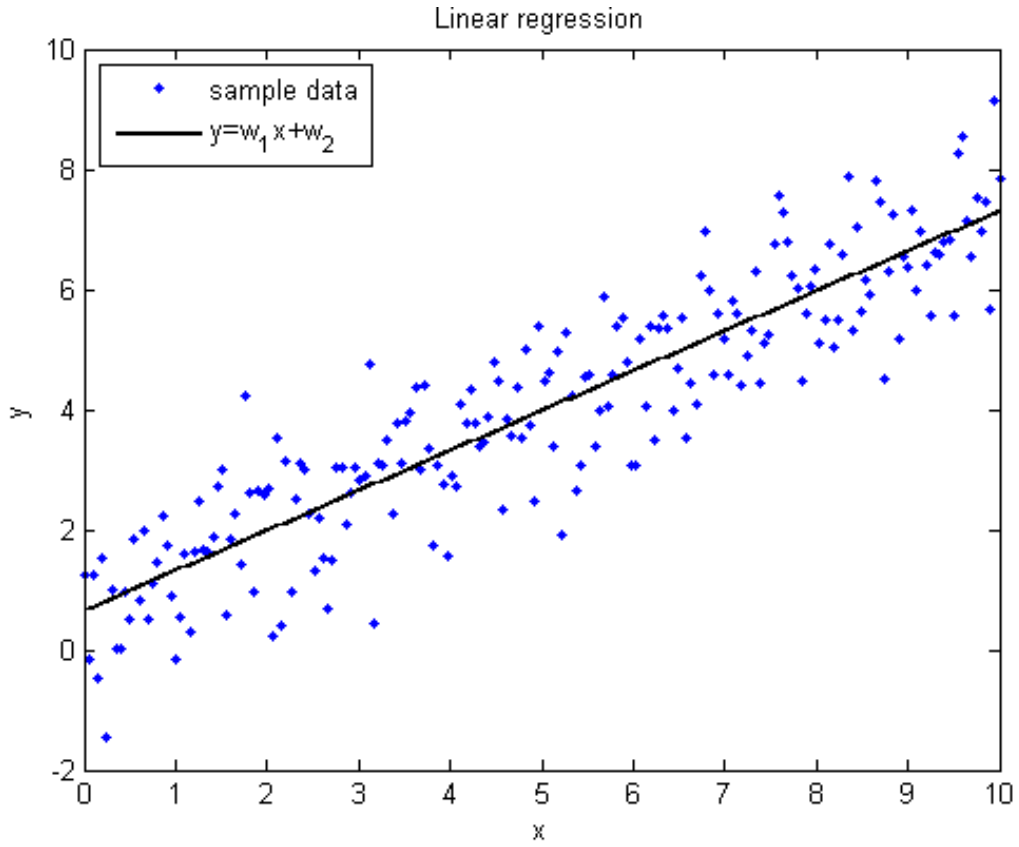

Рисунок 1.18 – Пример работы алгоритма

Основными задачами регрессионного анализа, являются: становление формы зависимости, определение функции регрессии и оценка неизвестных значений зависимой переменной.

Характер и форма зависимости между переменными могут образовывать следующие разновидности регрессии:

- положительная линейная регрессия;
- положительная равноускоренно возрастающая регрессия;
- положительная равнозамедленно возрастающая регрессия;
- отрицательная линейная регрессия;
- отрицательная равноускоренно убывающая регрессия;
- отрицательная равнозамедленно убывающая регрессия.

Стоит отметить то, что описанные разновидности обычно встречаются не в чистом виде, а в сочетании друг с другом. В таком случае говорят о комбинированных формах регрессии.

Вторая задача сводится к выяснению действия на зависимую переменную главных факторов или причин, при неизменных прочих равных условиях, и при условии исключения воздействия на зависимую переменную случайных элементов. Функция регрессии определяется в виде математического уравнения того или иного типа.

Решением третьей задачи заключается в решении задачи одного из следующих типов:

- оценка значений зависимой переменной внутри рассматриваемого интервала исходных данных, т.е. пропущенных значений, при этом решается задача интерполяции;
- оценка будущих значений зависимой переменной, т.е. нахождение значений вне заданного интервала исходных данных, при этом решается задача экстраполяции.

Обе задачи решаются путем подстановки в уравнение регрессии найденных оценок параметров значений независимых переменных. Результат решения уравнения представляет собой оценку значения целевой переменной.

При использовании регрессионного анализа следует учитывать его основное ограничение. Оно состоит в том, что регрессионный анализ позволяет обнаружить лишь зависимости, а не связи, лежащие в основе этих зависимостей.

В итоге, можно сказать, что регрессионный анализ дает возможность оценить степень связи между переменными путем вычисления предполагаемого значения переменной на основании нескольких известных значений.

Нейронные сети

Нейронные сети – это класс моделей, основанных на биологической аналогии с мозгом человека и предназначенных после прохождения этапа, так называемого обучения на имеющихся данных для решения разнообразных задач и анализа данных. При применении этих методов, прежде всего, встает вопрос выбора конкретной архитектуры сети, а именно числа слоев и количества нейронов в каждом из них. Размер и структура сети должны соответствовать существу исследуемого явления. Поскольку на начальном этапе анализа природа явления обычно известна плохо, выбор архитектуры является непростой задачей и часто связан с длительным процессом проб и ошибок.

Затем построенная сеть подвергается процессу обучения, на этом этапе нейроны сети итеративно обрабатывают входные данные и корректируют свои веса так, что бы сеть наилучшим образом прогнозировала данные, на который выполняется обучение. После него на имеющихся данных сеть готова к работе и может использоваться для построения прогнозов.

Нейронная сеть, полученная в результате обучения (рисунок 1.19), выражает закономерности, присутствующие в данных. При таком подходе она оказывается функциональным эквивалентом некоторой модели зависимостей между переменными, подобной тем, которые строятся в традиционном моделировании. Однако, в отличие от традиционных моделей, в случае нейронных сетей эти зависимости не могут быть записаны в явном виде, подобно тому, как это делается в статистике.

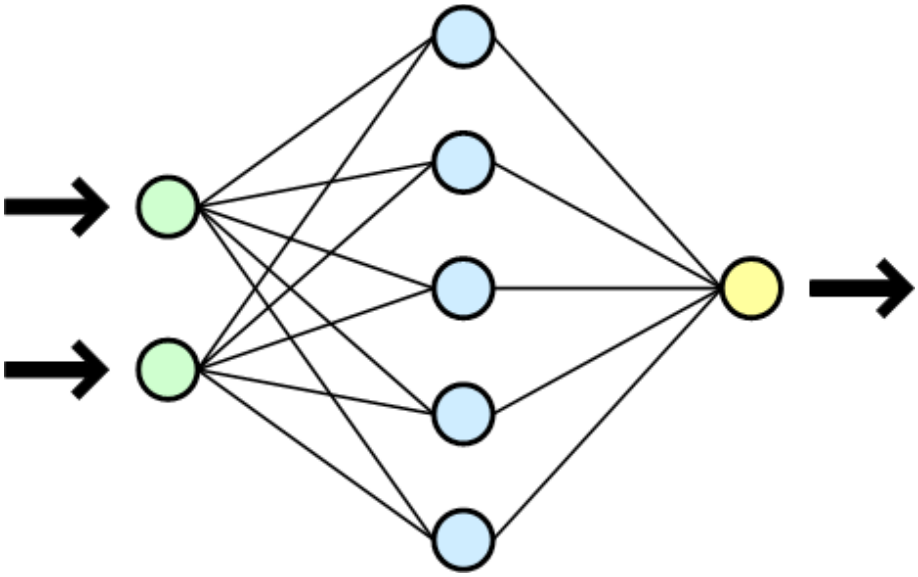

Рисунок 1.19 Пример нейронной сети

Иногда нейронные сети выдают прогноз очень высокого качества, однако они представляют собой типичный пример нетеоретического подхода к исследованию. То есть, не известно, каким именно образом обрабатываются данные, вы посылаете одни данные на вход, и получаете результат на выходе. При таком подходе сосредотачиваются исключительно на практическом результате, в данном случае на точности прогнозов и их прикладной ценности, а не на сути механизмов, лежащих в основе явления или соответствии полученных результатов какой-либо имеющейся теории.

Следует отметить, что методы нейронных сетей могут применяться и в исследованиях, направленных на построение объясняющей модели явления, поскольку нейронные сети помогают изучать данные с целью поиска значимых переменных или групп таких переменных, и полученные результаты могут облегчить процесс последующего построения модели. Более того, сейчас имеются нейросетевые программы, которые с помощью сложных алгоритмов могут находить наиболее важные входные переменные, что уже непосредственно помогает строить модель.

Одно из главных преимуществ нейронных сетей состоит в том, что они, по крайней мере, теоретически, могут аппроксимировать любую непрерывную функцию, и поэтому исследователю нет необходимости заранее принимать какие-либо гипотезы относительно модели и даже, в ряде случаем, о том, какие переменные действительно важны. Однако существенным недостатком нейронных сетей является то обстоятельство, что окончательное решение зависит от начальных установок сети и, как уже отмечалось, его практически невозможно интерпретировать в традиционных аналитических терминах, которые обычно применяются при построении теории явления.

Некоторые авторы отмечают тот факт, что нейронные сети используют, или точнее, предполагают использование вычислительных систем с массовым параллелизмом. Например, существует следующее определение нейронной сети:

Нейронная сеть – это процессор с массивным распараллеливанием операций, обладающий естественной способностью сохранять экспериментальные знания и делать их доступными для последующего использования. Он похож на мозг, так как сеть приобретает знания в результате процесса обучения и для хранения информации использует величины интенсивности межнейронных соединений, которые называются синоптическими весами.

1.8 Выводы по разделу

В данном разделе для лучшего понимания задачи была описана предметная область, то есть вкратце приведены сведения о работе сердца, и ее фиксации на кардиограмме. В дальнейшем сформулированы задачи – обработка ЭКГ относится к цифровой обработке сигнала, дальнейшая постановка диагноза – к задаче классификации.

Рассмотрены пакеты и языки, в которых реализована цифровая обработка сигналов, описаны их достоинства и недостатки. В итоге мы остановили свой выбор на языке Python, в виду его открытости, бесплатной лицензии и большого количества готовых библиотек и расширений.

Для задачи классификации рассмотрены несколько основных вариантов ее решения: нейронные сети, дерево решений, регрессионный анализ и «метод ближайшего соседа». В данной работе используется дерево решений. Несмотря на то, что сейчас большое распространение и развитие получили нейронные сети, они дают определенную погрешность, которая в данном случае, при постановке диагноза, имеет критическое значение. Дерево решений лишено этого недостатка.

2 РАЗРАБОТКА АЛГОРИТМОВ

2.1 Обработка ЭКГ в фазовом пространстве

Реальные ЭКГ значительно отличаются от той модели, что мы привели ранее – на ней нет почти прямых участков, соответствующих отсутствию сигнала. Это обусловлено наличием шумов и помех. Для того чтобы выделить полезный сигнал, ЭКГ фильтруется. Фильтрацию можно осуществить либо аппаратно, установкой фильтров и усилителей, либо путем математических преобразований, либо применив оба подхода сразу. Вопрос фильтрации сигнала достаточно обширен, поэтому в данной работе она не рассматривался, на вход подсистемы подается уже отфильтрованная ЭКГ.

Для диагностики большинства заболеваний достаточно рассмотреть один цикл ЭКГ. Но даже имея на руках очищенную ЭКГ, остается вопрос: какой из большинства циклов выбрать? Для этого вводится понятие эталонного цикла, который является усредненным для всей ЭКГ. Чтобы его найти предлагается [6] обработать ЭКГ в фазовом пространстве. Для этого ЭКГ разбивается на отдельные циклы и строится в плоскости $y(t)-y(t)$.

Для начала разобьем сигнал на отдельные циклы. Каждый цикл ЭКГ y_m представлен в виде последовательности дискретных отсчетов, зафиксированных в моменты времени t_k .

$$y_m[k]=y_m[t_k] \quad ,$$

где $m=1,..M$ – общее количество циклов

$$t_k=k\Delta \quad ,$$

где $k=0,..,K_m-1$;

Δ – шаг квантования.

Посчитаем для всего сигнала значение автокорреляционной функции

$$\Psi(\tau)=\sum_i y_m[i]*y_m[i+\tau] \quad ,$$

где τ – количество шагов квантования, соответствующих длине сердечного цикла.

Нам необходимо определить при каком значении τ величина $\Psi(\tau)$ будет максимальной [7]. Допустимые значения длительности сердечного цикла варьируются от 0,25с до 2,5с. Тогда τ будет находиться в следующих пределах:

$$\frac{0,25}{\Delta}\le\tau\le\frac{2,5}{\Delta}$$

Определив длину сердечного одного цикла, мы можем разбить весь сигнал на циклы. И приступить к построению графика в фазовой плоскости.

Для вычисления производной воспользуемся формулой численного дифференцирования для равноотстоящих узлов с погрешностью Δ^2 :

$$y_m(t_{i+1})=\frac{-3\,y_m(t_{i-1})+4\,y_m(t_i)-y_m(t_i)}{2\Delta} \qquad (2.1)$$

Конечно, существуют формулы и методы, которые могут дать нам меньшую погрешность. Но учитывая, что шаг квантования намного меньше общей длины кардиограммы, погрешности в Δ^2 нам будет достаточно.

Проведем нормировку $y_m^{¿}\in[0,1]$, $y_m^{*¿}\in[0,1]$. В результате получим последовательности Q_i , каждая из которых описывает одну траекторию нормированных векторов $z_m[k]$,которые отображают траектории циклов фильтрованной ЭКГ на плоскости с координатами $y^{¿}-y^{*¿}$, то есть на фазовой плоскости.

$$z_m[k]=\left(y_m^{¿}[k],\dot{y}_m^{¿}[k]\right),$$

где $m=1,\ldots,M$

$$Q_1=\left\{z_1[k],k=0,..,K_1-1\right\} .$$
$$Q_m=\left\{z_m[k],k=0,..,K_m-1\right\} .$$

Так как ЭКГ не является периодической функцией, траектории циклов в фазовом пространстве размываются и притягиваются к некоторой области – аттрактору в виде предельного цикла. Анализ математической модели, описанный в **источнике,** показал, что для любой точки $\theta=\theta_x$ из области определения эталона y_0 соответствующие точки $y_m(\theta_{mx}),y_m(\theta_{mx})$ искаженных сердечных циклов группируются в ограниченной области нормированного фазового пространства с центром в точке $y_0(\theta_x),\dot{y}_0(\theta_x)$ и радиусом $\rho(\theta_x)$, удовлетворяющим следующему условию:

$$\rho^2(\theta_x)=y_0^2(\theta_x)\varepsilon_m^2+\dot{y}_0^2(\theta_x)\left[\frac{\delta_m^i-\varepsilon_m}{1+\delta_m^i}\right] ,$$

где ε_m и δ_m^i случайные величины с матожиданием 0.

Это свойство дает возможность эффективно оценивать полезный сигнал y_0 . Метод основан на усреднени последовательностей Qв фазовом пространстве с применением матрицы $D=\left\|R_h(Q_i,Q_j)\right\|$ хаусдорфовых расстояний R_h между парами фазовых траекторий отдельных циклов:

$$R_h(Q_i,Q_j)=max\left\{\max_{z_j\in Q_j}\min_{z_i\in Q_i}\rho(z_i,z_j),\max_{z_i\in Q_i}\min_{z_j\in Q_j}\rho(z_i,z_j)\right\} ,$$

где $\rho(z_i,z_j)=\left\|z_i-z_j\right\|$ – евклидово расстояние между точками $z_i u\, z_j$ на фазовой плоскости.

После этого можем определить опорный сердечный цикл Q_0 – реальный цикл ЭКГ, который находится на минимальном хаусдорфовом расстоянии от остальных циклов:

$$Q_0 = arg \min_{1 \le j \le M} \sum_{i=1}^{M} R_h(Q_i, Q_j)$$

Далее усредним точки фазовых траекторий остальных циклом в окрестности соответствующих точек опорного цикла Q_0 . В результате получим оценку эталона в двумерном фазовом пространстве и во временной области (рисунок 2.1).

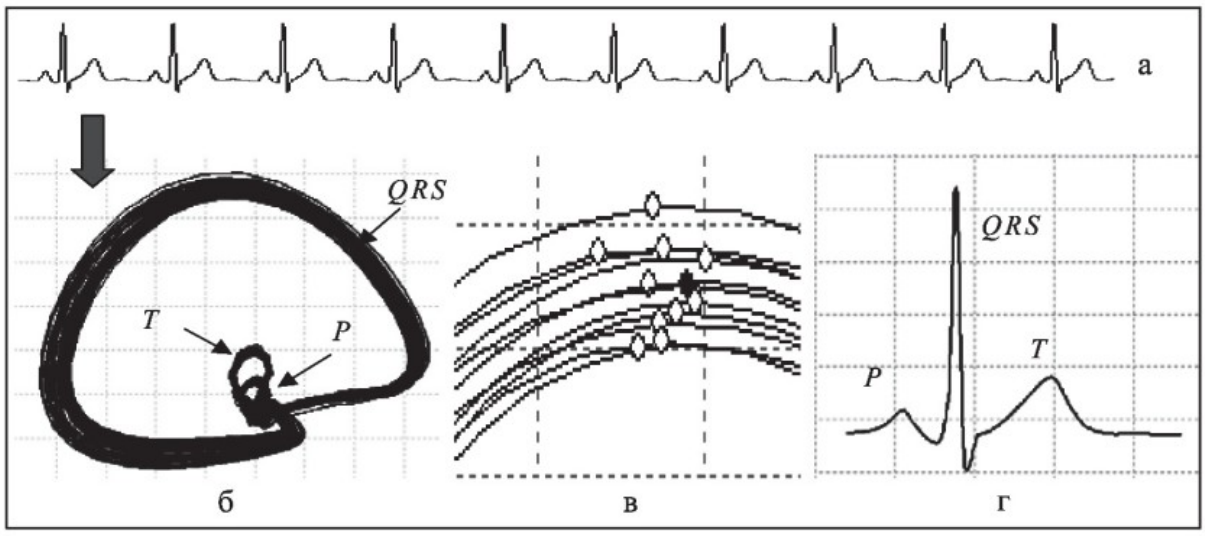

Рисунок 2.1 – иллюстрация алгоритма обработки сигнала в фазовом пространстве: а – исходная кардиограмма, б – фазовая траектория ЭКГ, в – фрагмент фазовых траекторий, г – восстановленный эталонный цикл

Схема алгоритма обработки ЭКГ в фазовом пространстве выглядит следующим образом (рисунок 2.2):

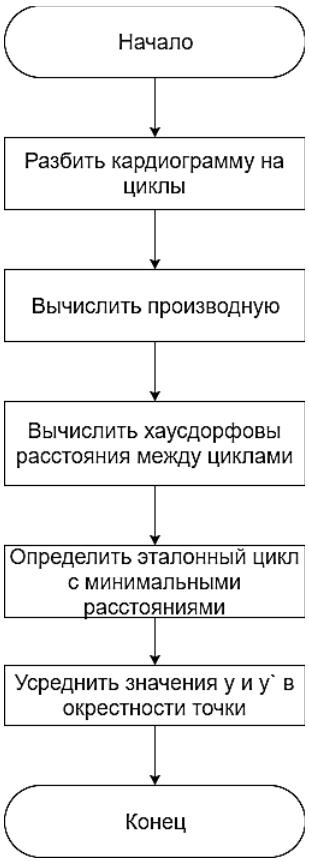

Рисунок 2.2 – Алгоритм обработки ЭКГ в фазовом пространстве

2.2 Алгоритм определения показателей ЭКГ

Классическая кардиография выявляет все необходимые пики и интервалы вручную, осматривая запись кардиограммы. Однако, ручная расшифровка проигрывает в точности и скорости компьютерной обработке.

Для определения показателей используют алгоритм первой производной. Его суть заключается в том, что вместе с циклом ЭКГ мы рассматриваем график его производной и выявляем зубцы. Для вычисления производной воспользуемся формулой (2.1).

Первой на графике ЭКГ идентифицируется точка R [8] как максимальное отклонение от изолинии. В случае патологий, когда R-зубец не так ярко выражен или имеет непривычную форму, он может быть определен вручную через окружающие зубцы.

Точка S определяется как максимальное отклонение от изолинии после точки R на графике производной ЭКГ.

Точка T находится как с использованием графика ЭКГ, так и графика ее производной: максимальное отклонение графика производной от изолинии после комплекса QRS. В случае патологий T-зубец может не проявляться на графике ЭКГ, в таком случае выделенный диапазон сравнивают с базой данных возможных вариантов положения зубца T.

Точки Q и P находятся на графике первой производной ЭКГ как локальные минимумы производной перед зубцом R.

2.3 Дерево решений

Разработан алгоритм классификации по наличию заболевания с использованием бинарного дерева решений. Основными особенностями алгоритма являются[9]:

- выбор иерархии параметров для построения дерева;
- функция построения разбиения в узлах дерева;
- возможность использования числовых и категориальных параметров;
- механизм отсечения ветвей;

Для построения наиболее эффективного дерева решений все данные разделяются на обучающую и тестовую выборки. При обработке обучающей выборки происходит построение бинарного дерева решений, по которому, впоследствии пройдут все данные тестовой выборки. Начинается построение бинарного дерева решений с изучения всех доступных параметров пациентов, которые берутся из исходных данных. Для сокращения трудоемкости задачи дерево строится с помощью жадного алгоритма.

Каждый параметр представлен в виде одного уровня дерева, и как следствие, все узлы на этом уровне олицетворяют собой этот параметр.

Рисунок 2.3 – Пример разделения пациентов по параметру частоты пульса

Приведем простой пример: предположим, что первым узлом в дереве станет параметр частота пульса (рисунок 2.3). Люди, у которых ЧСС меньше 60, будут направлены по левой ветви, остальные по правой.

2.3.1 Определение границ для каждого параметра

Теперь разберем, каким образом происходит разделение по каждому параметру. Всего у нас есть m-количество параметров:

$$x_1, x_2, \ldots, x_m.$$

Каждая переменная x_i может принимать значение из некоторого множества:

$$C_i = \{C_{i1}, C_{i2}, \ldots\}.$$

Иными словами у каждого параметра свой набор значений, отличающийся от других. Например, параметр «амплитуда R-зубца» предлагает интервал значений

от -1.5 до 1.5. Некоторые из них являются бинарными и принимают условные «да» и «нет» (0,1).

В каждом параметре x_i предстоит найти $C_{iн}$ и $C_{iв}$, где $C_{iн}$ – минимальное значение параметра больного пациента, в свою очередь $C_{iв}$ – максимальное значение параметра больного пациента. Делается это для того, что бы отделить одних пациентов от других.

Пациенты, удовлетворяющие условию $C_{iн} \leq C_i \leq C_{iв}$ идут по левой ветви, остальные по правой. В первом случае, риск того, что пациент заболеет, увеличивается, во втором – остается на прежнем уровне, но о системе оценок чуть позже.

На примере параметра «возраст» эти действия можно описать следующим образом: самым молодым пациентом является 19 летний человек ($C_{iн}$¿ , самым старым 65 летний ($C_{iв}$¿ . Все пациенты, входящие в эти границы являются потенциальными больными, остальные же – потенциально здоровыми людьми.

2.3.2 Определение параметров для построения дерева решений

Каждого пациента можно охарактеризовать следующим набором параметров:

$$I_j = (x_1, x_2, \ldots, x_m).$$

Но стоит ли включать все эти параметры в дерево решений? Ведь если они никаким образом не влияют на взаимосвязь с тем, есть ли рису сердечно-сосудистых заболеваний, то они просто будут увеличивать время обработки данных, как при построении дерева решений, так и при проверке тестовой выборки.

Для этого в обучающей выборке каждый из параметров $x_1, x_2, \ldots, x_m$ проходит проверку следующим образом.

Рассмотрим критерий выбора параметра:

$$N = N_б + N_з ,$$

где N – количество пациентов в обучающей выборке;

$N_б$ – количество больных пациентов;

$N_з$ – количество здоровых пациентов.

В каждом узле происходит разбиение пациентов на класс заведомо здоровых ($N_{оз}$) и класс, в котором есть больные и могут встречаться здоровые пациенты (N_a). Иными словами на каждом узле у нас будет встречаться все меньше пациентов, так как алгоритм будет отсекать заведомо здоровых людей. Но при этом не исключается вероятность того, что в этот класс могут попасть больные люди.

$$N_a = N - N_{оз} .$$

Разбиение же, благодаря которому собственно и происходит это отсечение описано в предыдущем пункте (см. п. 2.1.1.).

Добавим к предыдущему пункту пояснение. К первому классу N_a относятся пациенты удовлетворяющие условию $C_{iн} \leq C_i \leq C_{iв}$, остальные относятся ко второму классу $N_{оз}$.

И для того, что бы определить, следует использовать тот или иной параметр, в алгоритме действует проверка следующего рода. У каждого параметра смотрится значение N_a и $N_{оз}$. Если $\frac{N_a}{n} \| \frac{N_{оз}}{n}$ <= 0,01, где n – отвечает за количество пациентов в выборке, то такой параметр считается не существенным и не рассматривается. Этот параметр в дальнейшем исключается из оборота и в последствии алгоритм к нему не обращается.

2.3.3 Ранжирование узлов для построения дерева решений

В алгоритме дерева решений каждый уровень дерева и узлы этого самого уровня, олицетворяют собой определенный параметр, по которому происходит разделение пациентов (рисунок 2.3).

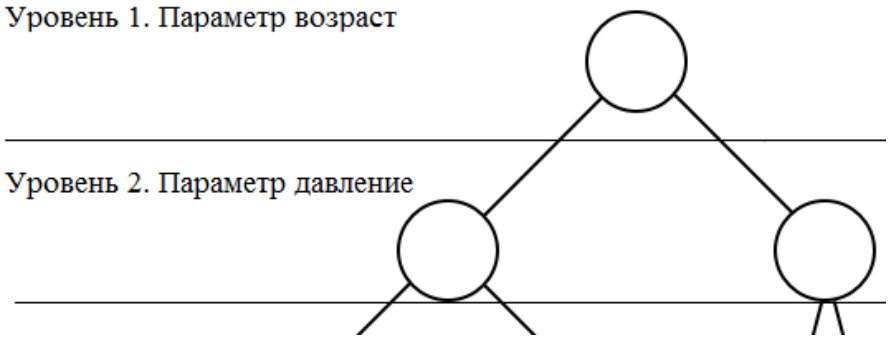

Рисунок 2.3 – Пример взаимодействие параметров и узлов дерева

Но для того, что бы определить какой параметр поставить на тот или иной уровень, требуется сделать следующее.

Предстоит вычислить оценку значимости параметра x_i , т.е. эмпирическую частоту заболевания P:

$$P = \frac{N_б}{N_a} = \frac{N_б}{N - N_{оз}}$$

Критерием выбора параметра x_i в узле i является максимальное разделение больных и здоровых пациентов. Это достигается при максимальном $N_{оз}$, т.е. при максимальном P.

По оценке значимости определяется, какой узел будет стоять выше остальных. Иными словами в зависимости от P узлы ранжируются по убыванию.

Благодаря такому способу разделения уже на первых уровнях дерева происходит максимальное разделение между здоровыми и больными пациентами. Это хорошо сказывается на следующем свойстве алгоритма - механизме отсечения ветвей.

2.3.4 Отсечение ветвей у дерева решений

После всех предыдущих действий переходим к построению самого дерева.

$D = \{T_{11}, T_{21}, T_{22}, \ldots, T_{ii}\}$,

где D – олицетворяет собой множество узлов одного дерева;

T – является узлом определенного уровня.

В зависимости от того, сколько параметров алгоритм допустил к построению нашего дерева, количество узлов на каждом уровне дерева будет равно 2^{u-1}, где u – количество параметров, допущенных к построению дерева. В итоге получаем дерево решений следующего вида (рисунок 2.4).

У стандартного дерева решений по исходным данным, обрабатываемых этим алгоритмом получается 1013 узлов. Вычисляется их количество следующей формулой $\sum_{z=1}^{u} 2^{z-1}$ узлов. Это очень много, поэтому было решено добавить в алгоритм механизм отсечения ветвей[10]. Работает он следующий образом:

- для каждой ветви узла определяется свой «вес». Вес левой ветви узла равен P_i, он высчитывается по формуле (2.1), вес правой равен 0. По левой ветви проходят пациенты удовлетворяющие условию $C_{iн} \leq C_i \leq C_{iв}$, по правой идут остальные;
- получаем «вес» каждого листа дерева;
- определяем P_{min}. Из всех листьев дерева выделяем те, в которые попадают только заболевшие пациенты. После этого среди них вычисляем лист с минимальным «весом» и присваиваем это значение P_{min}. Это делается для того, что бы построить границу оценки между заболевшими и здоровыми пациентами;
- отсечение границ убирает те пути, вес которых меньше P_{min}. Пациент из тестовой выборки направленный по этому пути будет определяться здоровым.

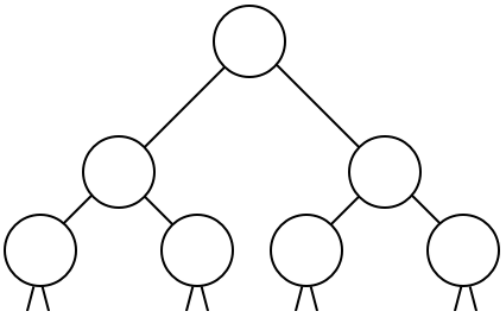

Рисунок 2.4 – Схематический рисунок дерева решений. В примере отображено лишь 3 уровня

Что же касается тех путей, вес которых больше или равен $P_{min,}$ то пациент, направленный по этому пути, будет определяться больным.

В результате мы получим дерево решений следующего вида:

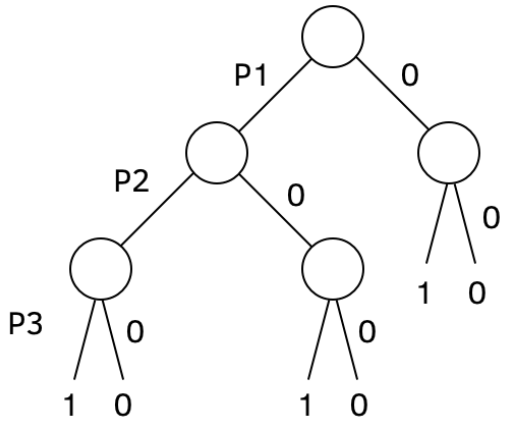

Рисунок 2.5 – Схематический рисунок дерева
решений с отсеченными ветвями

Как показала практика, количество узлов в таком случае уменьшается вдвое, на некоторых выборках получаются еще более значительное сокращение ветвей у дерева. Этот механизм позволяет в значительной степени ускорить работу алгоритма.

2.3.5 Диагностика с применением дерева решений

В исходных данных нам предоставлены восемь зависимых переменных $y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$:

- амплитуда P-зубца;
- амплитуда Q-зубца
- амплитуда R-зубца
- амплитуда S-зубца
- амплитуда T-зубца
- интервал J;
- возраст пациента;
- пол пациента;

Для всех трех типов сердечных заболеваний алгоритм строит отдельное дерево со всеми особенностями, которые описаны выше. Но почему бы просто не объединить эти три зависимых переменных в одну?

Дело в том, что каждому из этих трех типов заболевания принадлежит свой архетип пациента с определенным набором параметров. Например, от стенокардии напряжения страдают в основном люди от 19 до 60, когда нарушению ритма сердца подвержены люди от 24 до 80 и т.д.

Именно поэтому их требуется рассматривать по отдельности иначе повыситься процент погрешности, а этого нам не нужно. Именно по этим причинам алгоритм строит три дерева решений и по каждому из них прогоняет все данные тестовой выборки. Стоит отметить, что у каждого из этих трех деревьев не только разные границы для параметров, но так же отличается расстановка параметров в иерархическом плане. То есть, в первом дереве в самом начале может стоять параметр давление, когда во втором будет стоять возраст.

После завершения всех приготовлений, загружается тестовая выборка и прогоняется через все готовые деревья решений. После, каждый из пациентов этой тестовой выборки получает свой результат, а именно 0 или 1, означающие, что он здоров или болен.

Далее алгоритм выдает процент ошибок, который получился при обработке данных. Обычно он не превышает 4%, что может характеризовать реализацию данного алгоритма, как успешную.

2.4 Общая схема алгоритма дерева решений для диагностики

Для того, что бы разработанный мною алгоритм был нагляднее, было решено дополнительно визуализировать его с помощью схем-алгоритмов. Стоит отметить и то, что именно по этим схемам и была разработана программа, с помощью которой производилась обработка исходных данных.

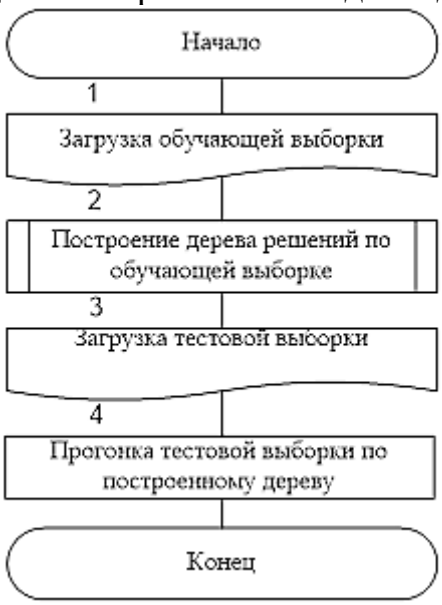

Рисунок 2.6 – Алгоритм построения дерева решений для диагностики сердечно-сосудистых заболеваний

Теперь разберемся в том, как именно происходит построение дерева решений по обучающей выборке.

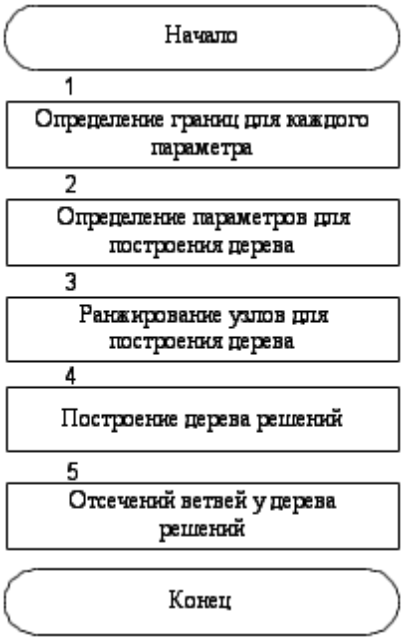

Рисунок 2.7 – Построения дерева решений по обучающей выборке

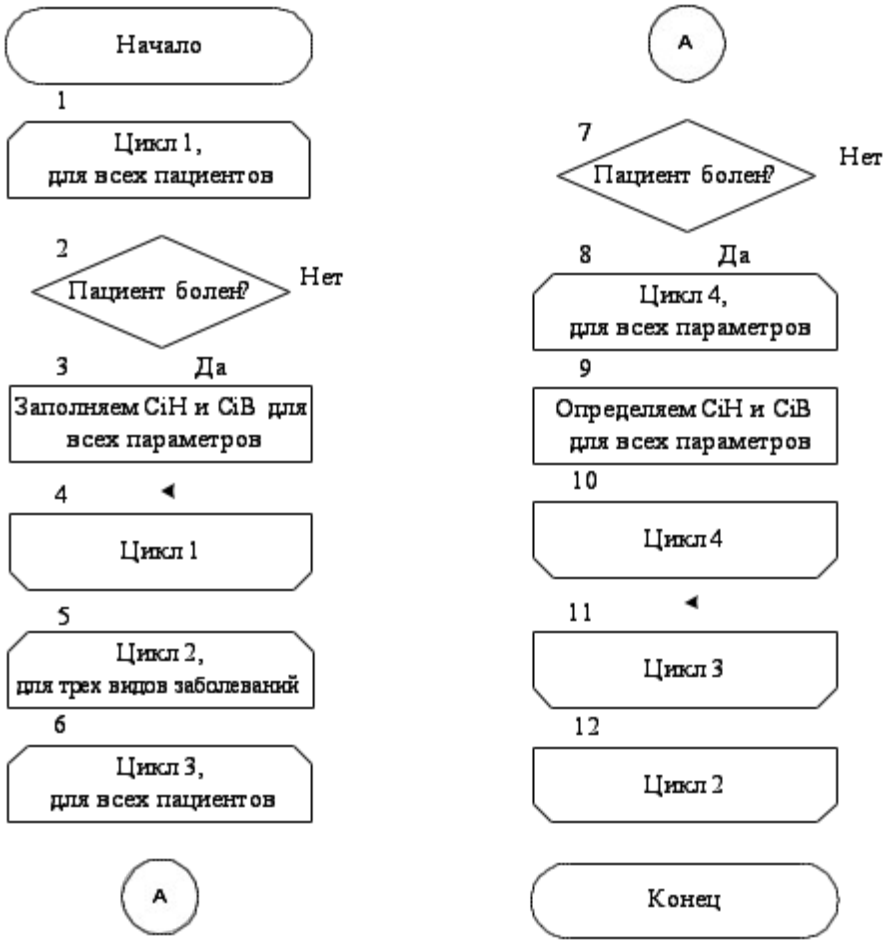

Рисунок 2.8 – Определение границ для каждого параметра

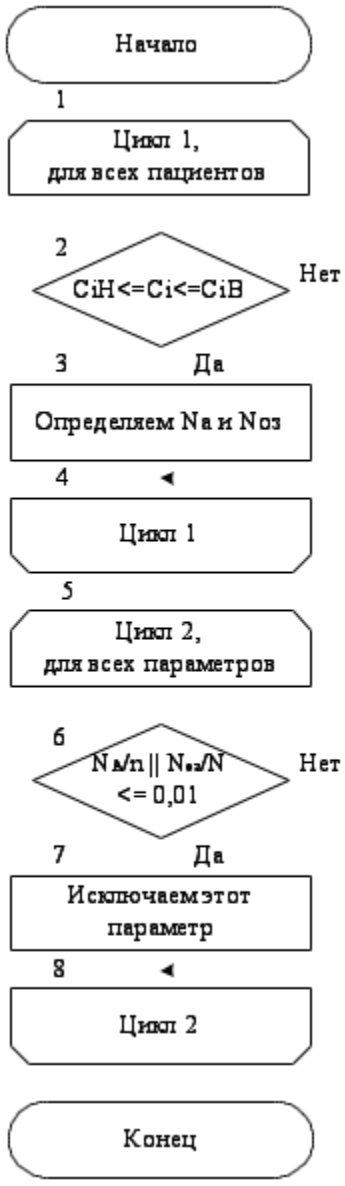

Рисунок 2.9 – Определение параметров для построения дерева

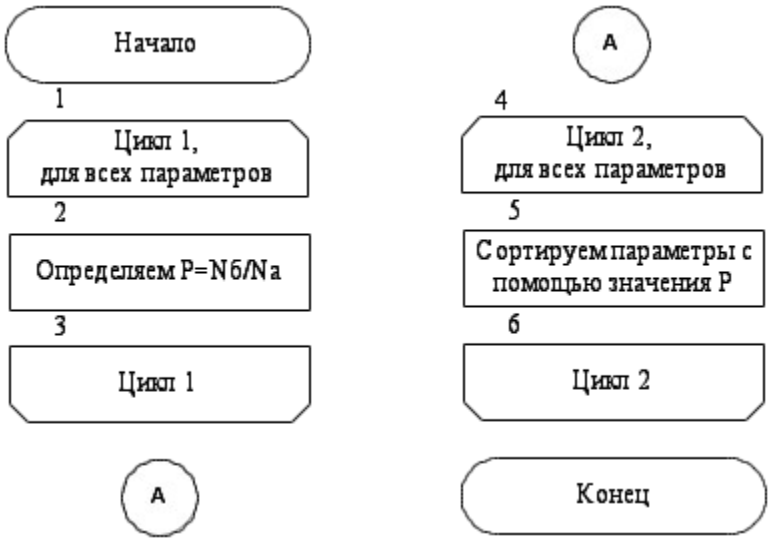

Рисунок 2.10 – Ранжирование узлов для построения дерева

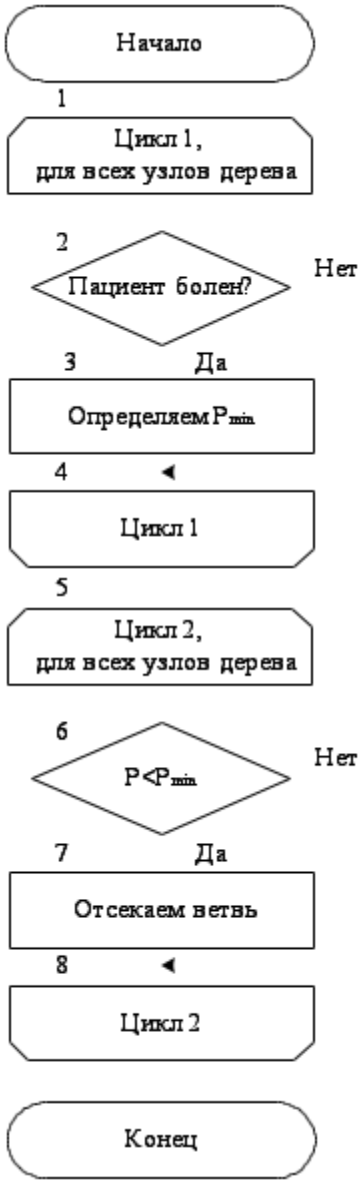

Рисунок 2.11 – Отсечение ветвей у дерева решений

После всех этих действий алгоритм прогоняет данные тестовой выборки через полученное дерево решений и выводит результат.

2.5 Выводы по разделу

В данном разделе были подробно рассмотрены алгоритм обработки ЭКГ в фазовом пространстве и выделения эталонного цикла. Описан алгоритм выявления признаков ЭКГ. Подробно разобран процесс построения дерева решений для классификации ЭКГ с механизмом отсечения ветвей.

3 РЕАЛИЗАЦИЯ ПОДСИСТЕМЫ

3.1 Диаграмма вариантов использования

Чтобы получить более четкое представление о функциональном использовании подсистемы, оно было проиллюстрировано на диаграмме вариантов использования (рисунок 3.1).

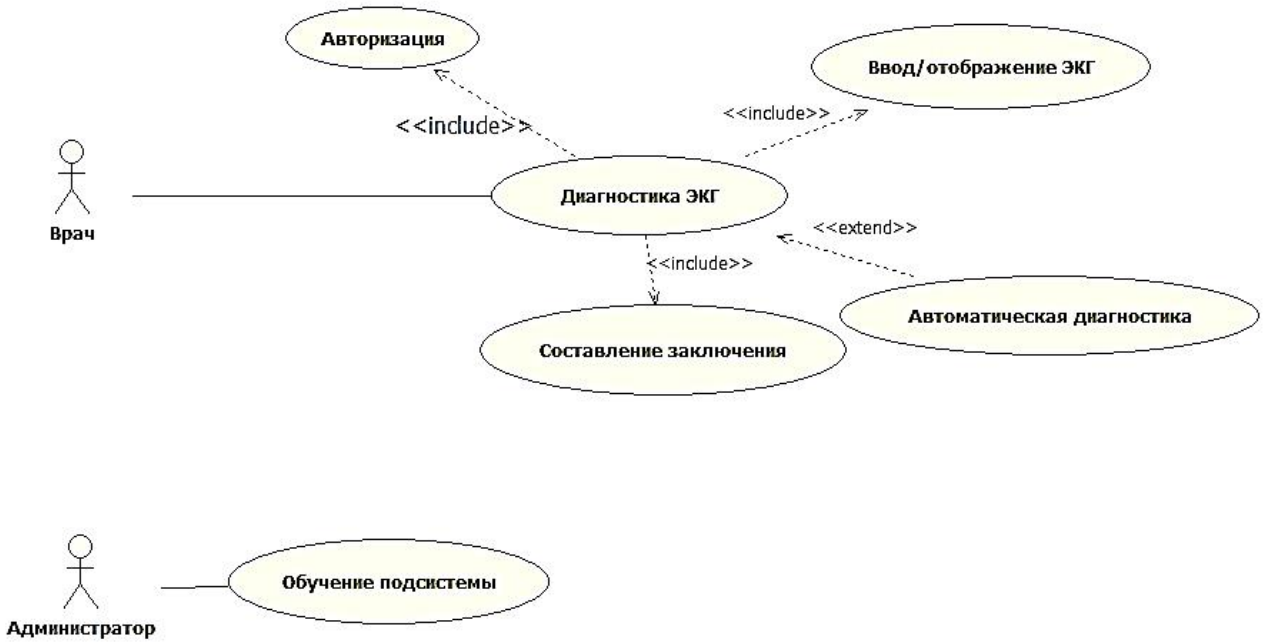

Рисунок 3.1 – диаграмма вариантов использования

У подсистемы обработки и анализа всего один актор – врач.Администратор, занимающийся обучением, действует в рамках другой подсистемы.

Первое, с чего начинает работу врач – авторизуется в программе с помощью логина и пароля. В случае если логин-пароль неправильные или отсутствуют на сервере, подсистема предлагает повторно ввести пароль.

После авторизации начинается работа с кардиограммами. На диаграмме она отображена блоком «Диагностика ЭКГ». Врач может загрузить из базы кардиограмму, она отобразится в программе, и составить по ней заключение.

Кроме того, можно воспользоваться функцией автоматической диагностики, которая выдаст предполагаемый диагноз. Окончательное решение о диагнозе, выявленном на кардиограмме, остается за врачом.

3.2 Проектирование базы данных

3.2.1 Концептуальное проектирование

Проектирование базы данных начинается с концептуального проектирования. Оно производится на основе описания предметной области.

- У каждого пациента нас интересует его ФИО, пол, дата рождения.
- Кардиограмма хранит в себе данные ЭКГ, дату снятия и заключение, которое делает врач. Когда заключение сделано, имя врача сохраняется. У пациента может быть несколько кардиограмм.
- Соответственно, у врача должны быть указаны ФИО и его логин и пароль, для входа авторизации
- После обработки на ЭКГ на ней выделяются необходимые параметры, которые использует дерево решений для диагностики.
- Дерево решений представлено в виде отдельных узлов. В них хранится значение коэффициента, предок и левый и правый потомки.
- У болезней нам нужно только их название.

Изучая данные сердечной активности, врач-кардиолог ставит диагноз. При этом сохраняется информация о том, кто поставил диагноз.

Итого, мы выделили 6 сущностей: пациент, кардиограмма, показатели и диагноз, - и связи: два типа 1 к 1, и две 1 ко многим. Была построена соответствующая ER-диаграмма (рисунок 3.2).

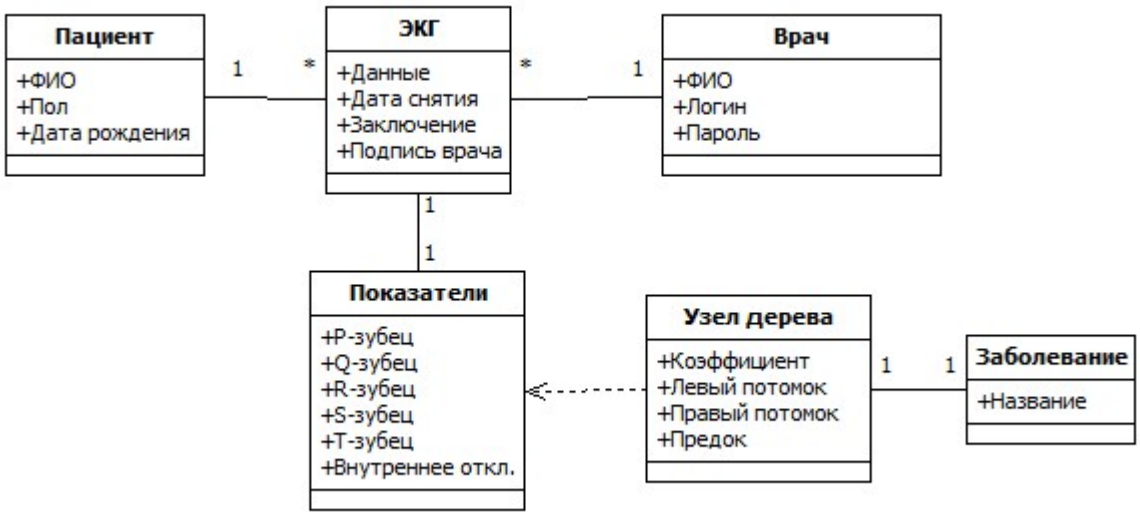

Рисунок 3.2 – ER-диаграмма базы данных

Все таблицы данной базы данных находятся в нормальной форме. Кроме того, каждая таблица имеет только один первичный ключ, а значит, разработанная база данных находится в нормальной форме Бойса-Кодда [11].

3.2.2 Логическое проектирование

На основе построенной ER-диаграммы построим реляционную схему базы данных [12] (рисунок 3.2).

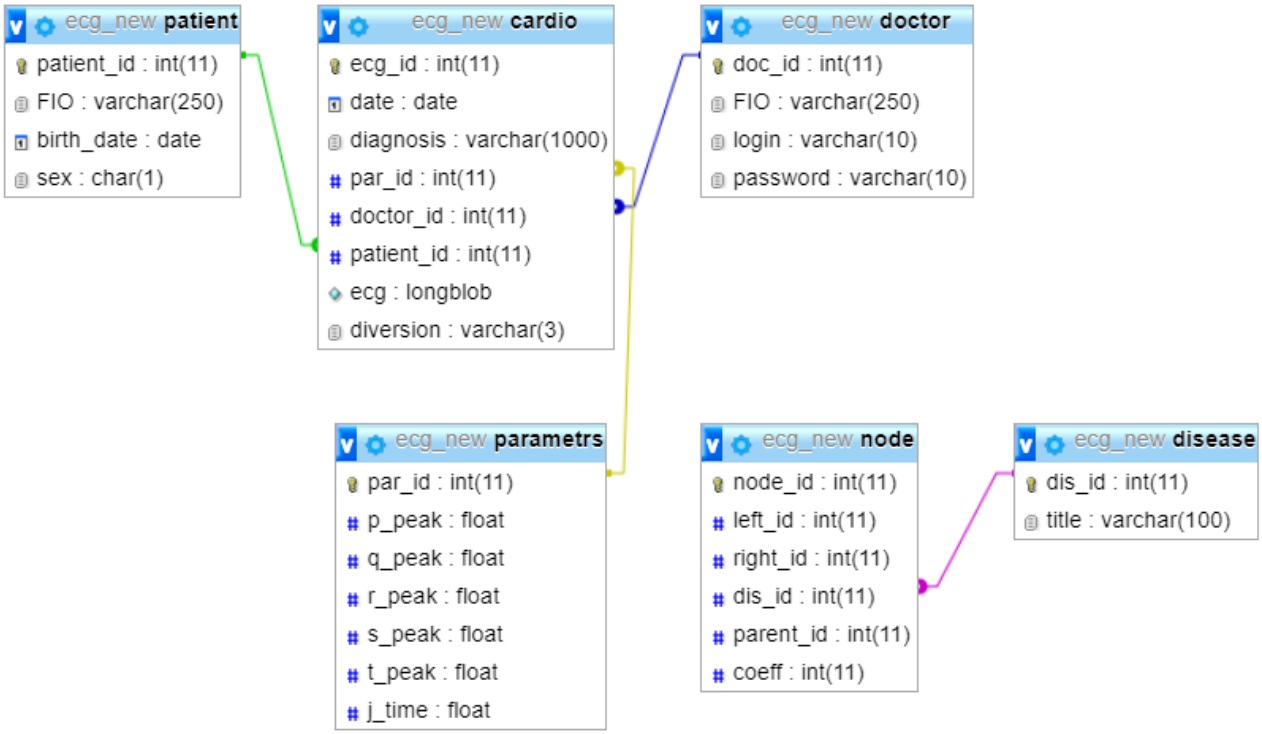

Рисунок 3.3 –Схема реляционной базы данных

Подробное описание таблиц и свойств полей приведено ниже.

Таблица 1

Таблица «Пациент»

	Тип данных	Значение по умолчанию	Обязательность	Первичный ключ	Внешний ключ	Ограничения
patient_id	int	-	+	+	-	-
FIO	varchar(250)	-	+	-	-	-
birthDate	date	-	+	-	-	-
sex	char(1)	-	+	-	-	Либо «м» либо «ж»

Таблица «Пациент» содержит всю информацию о пациентах, ФИО представлено строкой, пол – один символ «м» либо «ж».

Таблица 2

Таблица «Кардиограмма»

	Тип данных	Значение по умолчанию	Обязательность	Первичный ключ	Внешний ключ	Ограничения
ecg_id	int	-	+	+	-	-
ecg_data	blob	-	+	-	-	-
reg_date	date	-	-	-	-	-
diagnosis	shortint	-	-	-	-	-
par_id	int	-	-	-	+	-
diag_id	int	-	-	-	+	-
diversion	varchar(3)	-	+	-	-	-

В первую очередь данная таблица нужна для хранения данных кардиограмм. Изначально они представлены в виде файлов формата CSV, чтобы загрузить их в таблицу используется поле типа blob– большой бинарный объект.

Таблица 3

Таблица «Параметры»

	Тип данных	Значение по умолчанию	Обязательность	Первичный ключ	Внешний ключ	Ограничения
par_id	int	-	+	+	-	-
p_peak	int	-	-	-	-	-
q_peak	int	-	-	-	-	-
r_peak	Int	-	-	-	-	-
s_peak	int	-	-	-	-	-
t_peak	int	-	-	-	-	-
j_time	int	-	-	-	-	-

Параметры хранят значения амплитуд и временных интервалов.

Таблица 4

Таблица «Врач»

	Тип данных	Значение по умолчанию	Обязательность	Первичный ключ	Внешний ключ	Ограничения
doc_id	int	-	+	+	-	-
FIO	varchar(250)	-	+	-	-	-
login	varchar(10)	-	+	-	-	-
password	varchar(10	-	+	-	-	-

Содержит в себе информацию о пользователях системы

Таблица 5

Таблица «Узел дерева»

	Тип данных	Значение по умолчанию	Обязательность	Первичный ключ	Внешний ключ	Ограничения
node_id	int	-	+	+	-	-
left_id	int	-	-	-	+	-
right_id	int	-	-	-	+	-
parent_id	int	-	-	-	+	-
dis_id	int	-	-	-	+	-
coeff	double	-	+	-	-	-
par_name	varchar(10)	-	+	-	-	-

Таблица 6

Таблица «Заболевание»

	Тип данных	Значение по умолчанию	Обязательность	Первичный ключ	Внешний ключ	Ограничения
dis_id	int	-	+	+	-	-
Title	varchar(100)	-	+	-	-	-

3.3 Проектирование интерфейса программы

Работа с приложением начинается с окна авторизации (рисунок 3.4). На нем расположены две формы для ввода логина и пароля и кнопка «Войти», которая выполняет соединение с базой данных по заданному логину и пароля. В случае, если логин и пароль некорректные, отображается соответствующее сообщение.

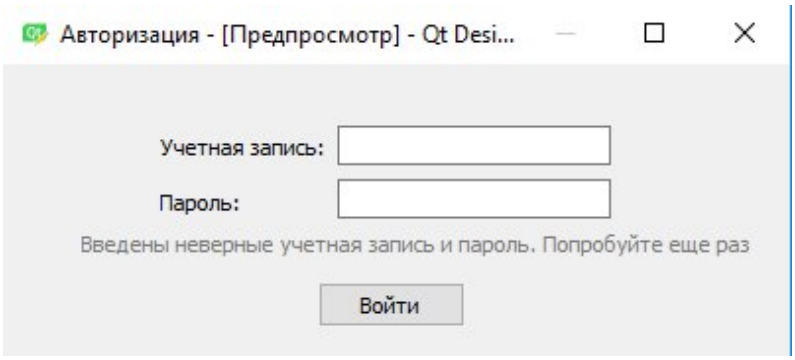

Рисунок 3.4 – Окно авторизации в программе

Если введены правильные логин и пароль, открывается основное окно программы (рисунок 3.5).

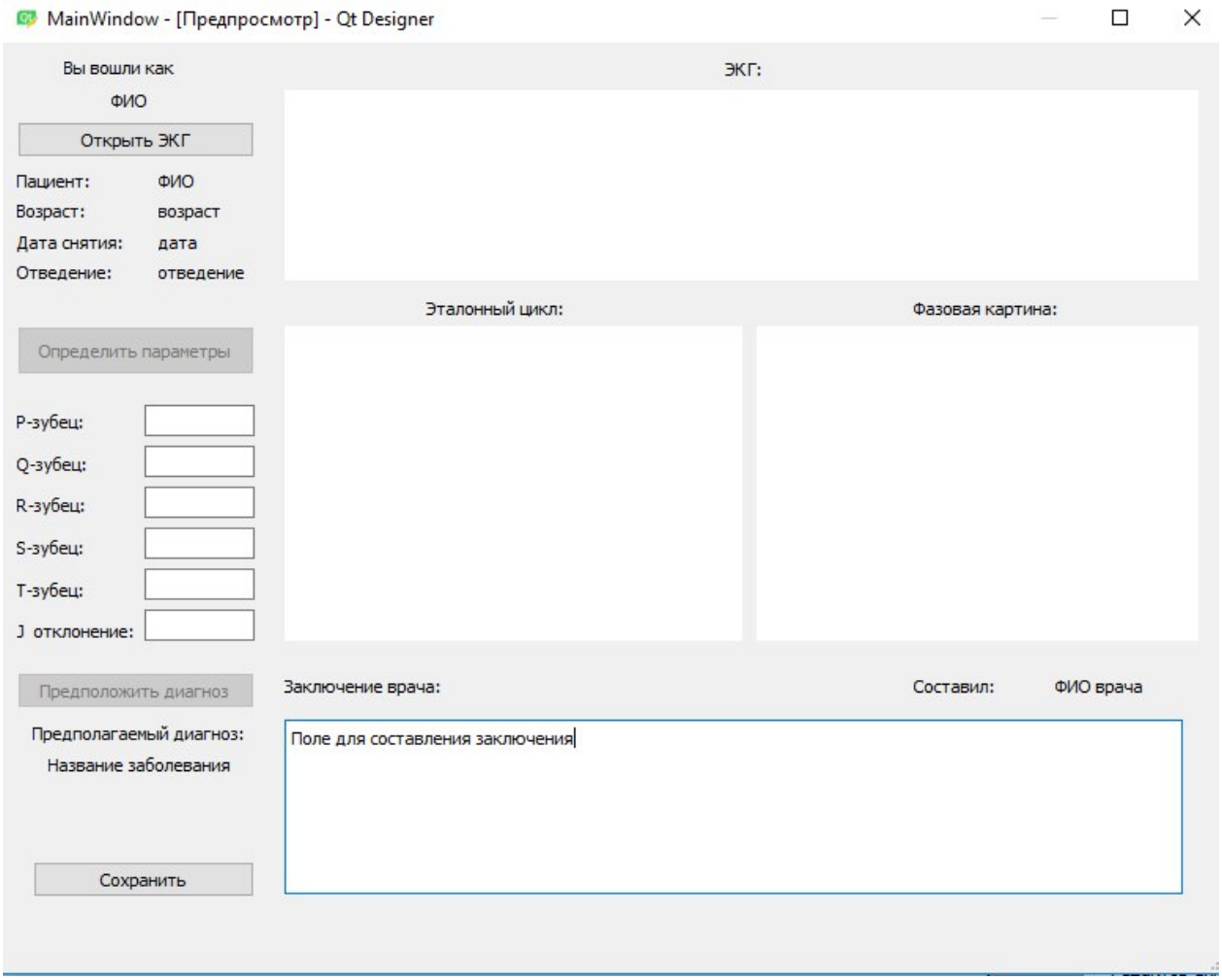

Рисунок 3.5 – Главное окно программы при запуске

Сразу после запуска программы в главном окне нет никакой информации кроме имени авторизовавшегося пользователя. Чтобы работать дальше необходимо выбрать кардиограмму с которой мы будем работать. Для этого нужно нажать кнопку «Открыть ЭКГ», после чего появится модальная форма загрузки ЭКГ (рисунок 3.6).

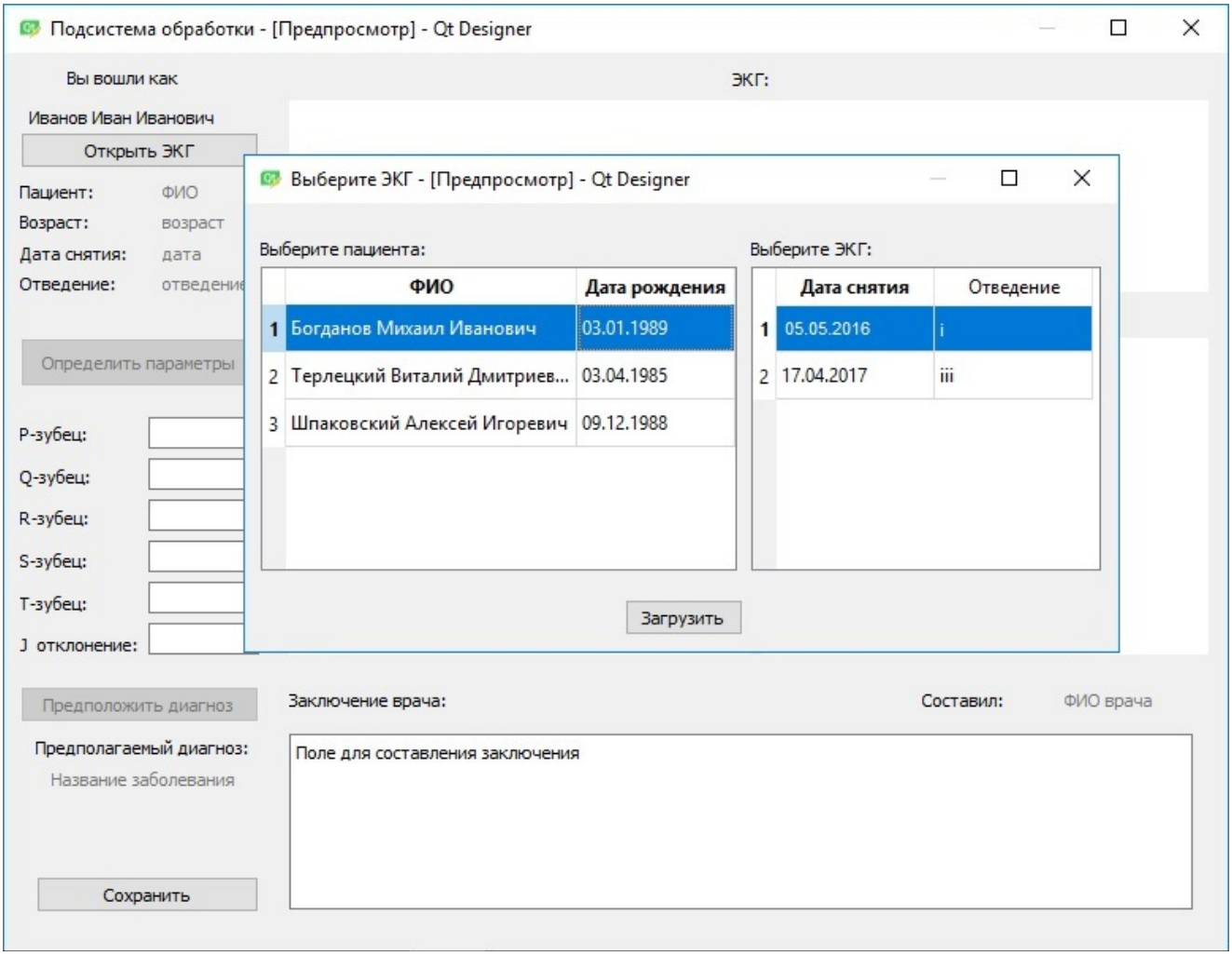

Рисунок 3.6 – Окно выбора кардиограммы

После того как мы выбрали кардиограмму, мы возвращаемся в главное окно программы. Теперь на нем отображаются информация о пациенте, три графика: исходная ЭКГ, ее отображение в фазовом пространстве, и полученный эталонный цикл. Кнопка «Определить параметры» становится доступной (рисунок 3.7).

Сразу после загрузки на ЭКГ нет никаких параметров. Их можно определив автоматически нажав соответствующую кнопку, либо ввести вручную в поля, расположенные в левой части главного окна. Когда все поля заполнены, кнопка «Предположить диагноз» становится доступной. После ее нажатия внизу окна появляется список выявленных болезней, либо надпись «Болезней не выявлено» (рисунок 3.8).

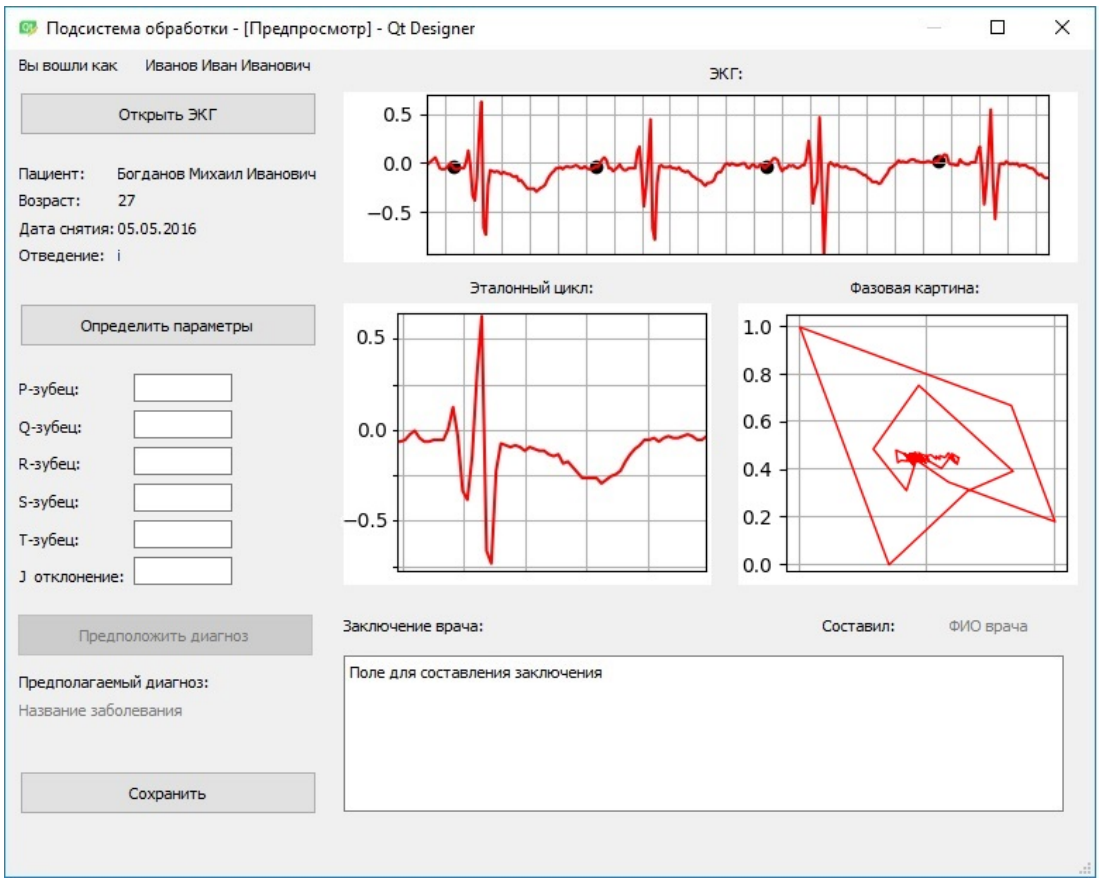

Рисунок 3.7 – Главное окно с загруженной кардиограммой

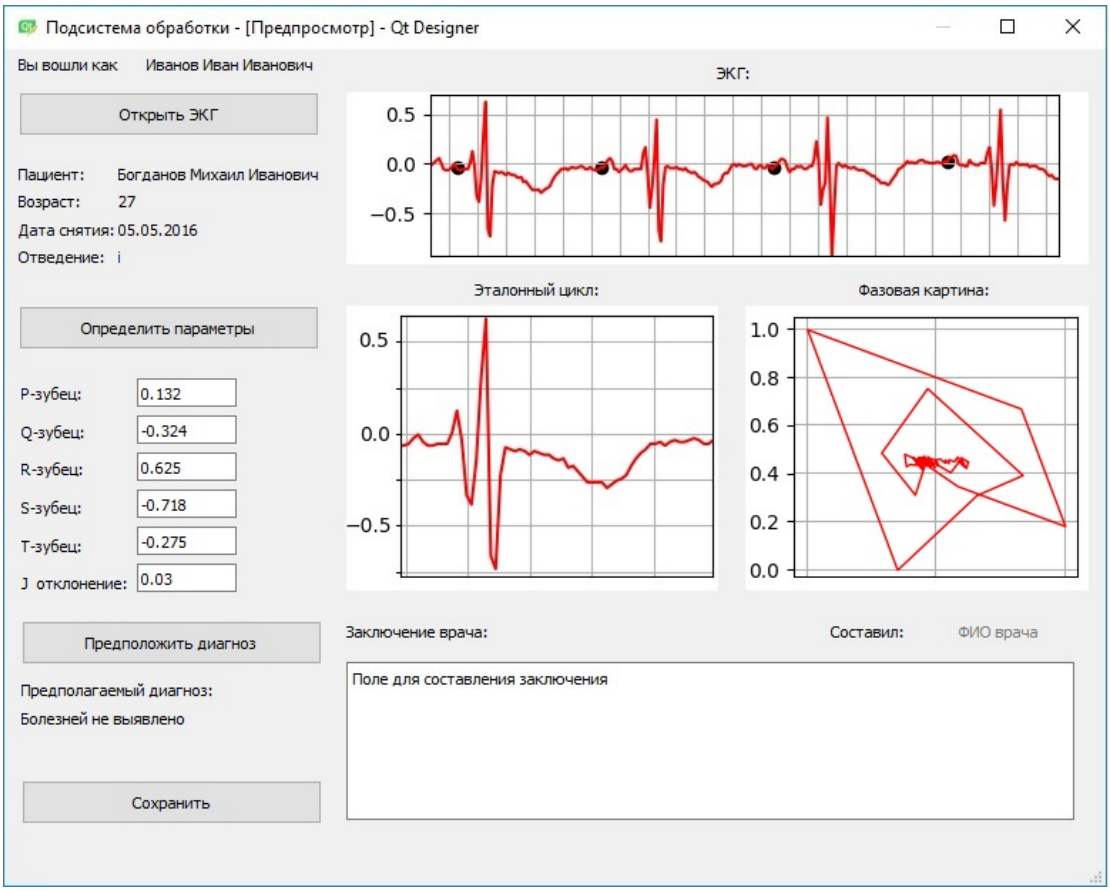

Рисунок 3.8 – Главное окно с определенными показателями и диагнозом

3.4 Выводы по разделу

В данном разделе выполнена разработка архитектуры и интерфейса программы. Представлена диаграмма вариантов использования системы. Спроектирована база данных и приведено описание интерфейса программы, расположившегося в трех окнах.

4 ЭКСПЕРИМЕНТАЛЬНОЕ ТЕСТИРОВАНИЕ СИСТЕМЫ

4.1 Результаты сравнения работы классификаторов

Для проверки корректности работы нашей подсистемы мы проверили ее работу на реальных записях ЭКГ. Данные брались из открытой базы физиологических сигналов physionet.org, нам заранее известен диагноз, пол и возраст пациента. Из нее мы выбрали 100 кардиограмм, которые можно разбить на три группы: с нормальными показателями, с инфарктом, с гипертрофией предсердий. Сотню взятых кардиограмм мы разбили на обучающую и контрольную выборки следующим образом:

Таблица 7

Разбиение набора на обучающие и тестовые выборки

№ набора	Обучающая выборка	Тестовая выборка
1	50	50
2	55	45
3	60	40
4	65	35
5	45	55
6	40	60
7	35	65
8	30	70

Сравним результат обработки выборки несколькими классификаторами. Мы взяли:

- нейронные сети (обозначение в таблице НС);
- линейная регрессия (обозначение ЛР);
- метод «ближайшего соседа» (обозначение БС);
- дерево решений (обозначение ДР).

Для первых трех классификаторов использовались уже готовые алгоритмы, для дерева решений взята наша реализация. В таблице ниже приведены погрешности классификации для каждого из алгоритмов на каждой выборке.

Таблица 8

Погрешности классификации тестовой выборки каждым из классификаторов

№	1	2	3	4	5	6	7	8
НС	12,2%	10,8%	11,7%	12,5%	15,5%	14,2%	18,0%	21,0%
БС	44,1%	41,6%	42,4%	38,9%	42,1%	46,4%	49,2%	50,6%
ЛР	35,1%	29,6%	29,4%	28,2%	36,1%	40,4%	42,2%	41,6%
ДР	3,0%	2,8%	2,7%	2,5%	3,5%	3,5%	3,7%	4,0%

Как и ожидалось, дерево решений обладает наименьшей погрешностью. Нейронные сети так же хорошо себя проявили, однако чем меньше становился объем обучающей выборки, тем больше росло значение погрешности.

Для сравнения быстродействия алгоритмов, в программе был написан соответствующий таймер. Все алгоритмы, кроме нейронной сети, сработали за несколько минут. Нейронной сети понадобилось в 2 раза больше времени. Алгоритм ближайшего соседа на больших объемах обучающей выборки так же будет работать медленней, в виду особенности его работы – полного перебора выборки.

Разработанная программа верно провела диагностику всех трех типов состояния пациента: нормальное (рисунок 4.1), наличие инфаркта (рисунок 4.2), наличие гипертрофии предсердий (4.3).

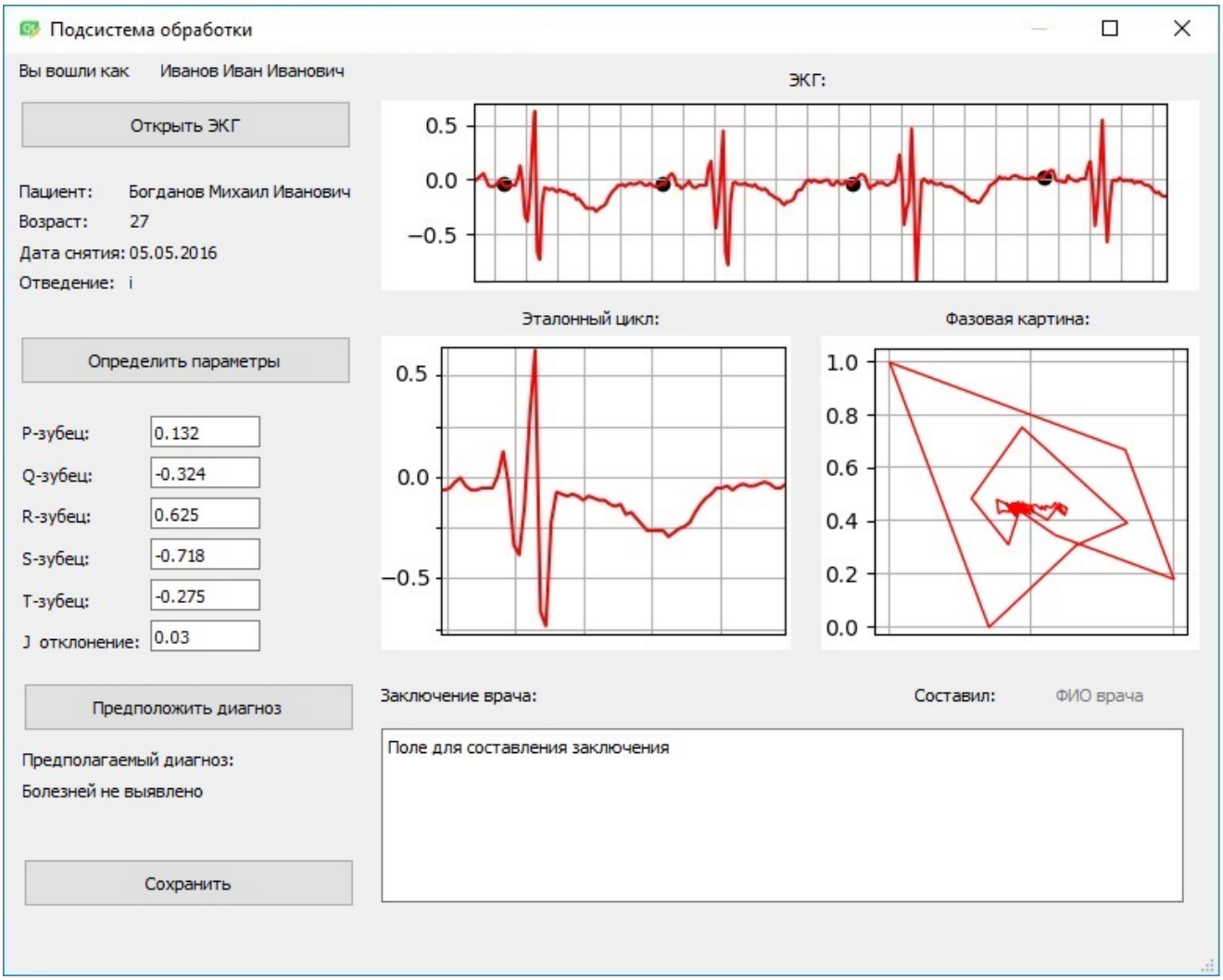

Рисунок 4.1 – результат работы на нормальной кардиограмме

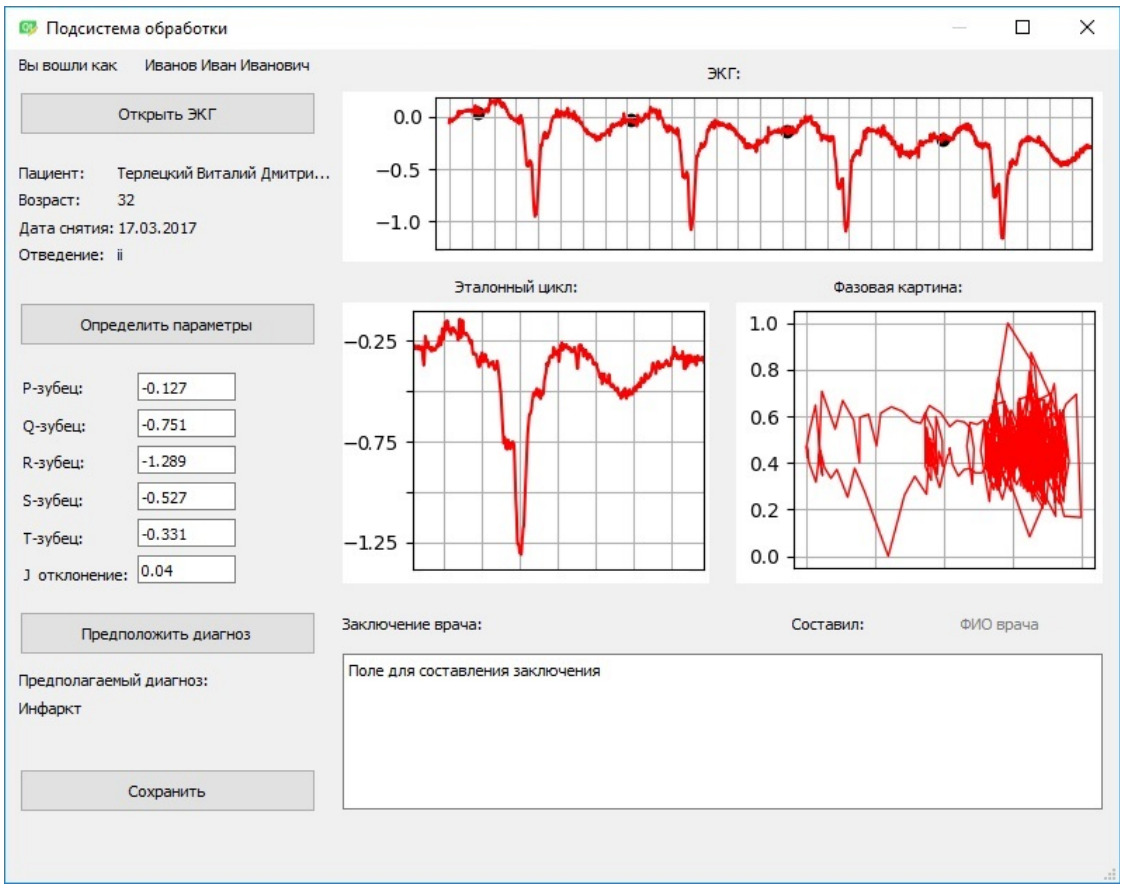

Рисунок 4.2 – результат работы на кардиограмме с инфарктом

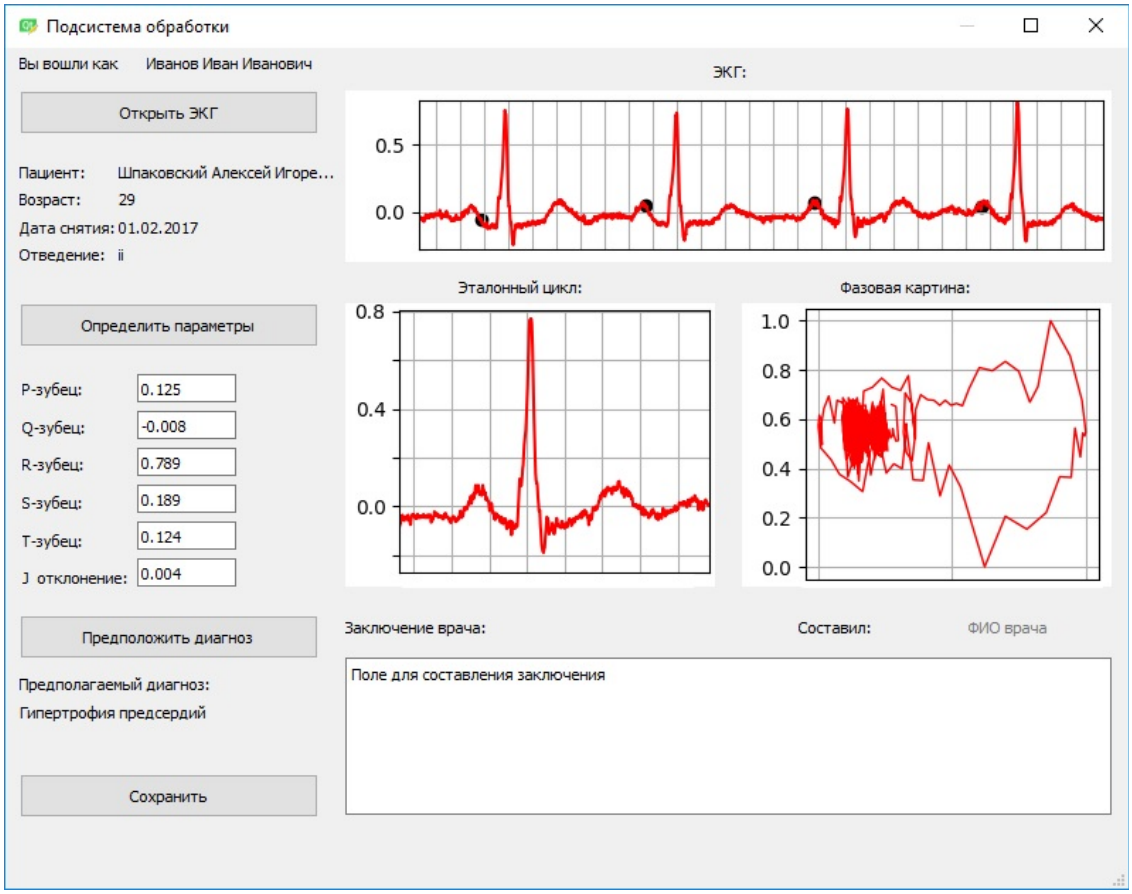

Рисунок 4.3 – результат работы на кардиограмме с гипертрофией предсердий

4.2 Выводы по разделу

В данном разделе были рассмотрены результаты обработки реальных данных ЭКГ различными классификаторами. Всего было отобрано 100 кардиограмм, их разделили на обучающую и тестовую выборки. Они были обработаны нейронной сетью, методом ближайшего соседа, линейной регрессией и деревом решений. Для первых трех классификаторов использовались готовые алгоритмы, дерево решений разработано нами.

В ходе тестирования, дерево решений показало наименьшую погрешность, которая варьируется от 3,0-4,0%. Хуже всего себя проявил метод ближайшего соседа, давая погрешность до 50,6%.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были рассмотрены алгоритмы обработки электрокардиограмм: разбиение на циклы, выявление эталонного цикла с помощью обработки в фазовом пространстве и усреднения с помощью матриц расстояний Хаусдорфа. Был рассмотрен алгоритм выявления основных признаков, а также алгоритм построения дерева решений для автоматической диагностики заболеваний. Спроектирована база данных для хранения кардиограмм, информации об узлах деревьев решений и всей сопутствующей информации, такой как информация о пациентах и врачах, зарегистрированных в системе. Разработана диаграмма использования подсистемы, и спроектирован интерфейс программы.

Разработанная программа прошла успешные испытания на реальных данных ЭКГ, взятых из открытой базы физиологических сигналов physionet.org, верно определив заболевания присутствующие на них.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Лутра, А. ЭКГ понятным языком. / А. Лутра. — М. : Практическая медицина, 2010. — 224 с.
2. Лаун, Б. Дети Гиппократа XXI века. Дела сердечные. / Лаун Б. – М. :Эксмо, 2010. – 285 с.
3. Азбука ЭКГ и Боли в сердце. / Зудбинов Ю. И. – Ростов н/Д. : Феникс, 2008. – 235с.
4. Чазов Е.И. Руководство по нарушениям ритма сердца. / Чазов Е.И., Голицын С.П. – М. : ГЭОТАР-Мелиа, 2008. – 416 с.
5. Демин А.В., Витяев Е.Е., Полоз Т.Л. Реализация универсальной системы извлечения «Discovery» и ее применение в задачах медицинской диагностики // Труды Всероссийская конференция с международным участием «Знания – Онтологии – Теории». – Новосибирск. – 2007. – Т.1. – С. 63–70.
6. Файнзильберг Л. С., Эффективная информационная технология обработки ЭКГ в задаче скрининга ишемической болезни сердца / Файнзильберг Л. С. // Клиническая информатика и телемедицина. – 2010. - № 6. – С. 22-30
7. Солонина, А.И. Основы обработки сигналов: курс лекций. / А.И. Солонина, Д.А. Улахович, С.М. Арбузов, Е.Б. Соловьева. — СПБ. : БВХ-Петербург, 2007. — 768 с.
8. Рангайн, Р.М. Анализ биомедицинских сигналов. Практический подход. / Р.М. Рангайн. – М. :Физматлит , 2007. – 440 с.
9. Деревья решений – общие принципы работы. Дата обновления: 11.03.2007. URL:https://basegroup.ru/community/articles/description (дата обращения: 14.04.17)
10. Методы классификации и прогнозирования. Деревья решений. Дата обновления. Дата обновления: 10.08.2009. URL: http://www.intuit.ru/studies/courses/6/6/lecture/174?page=1 (дата обращения: 28.04.2017)
11. Шварц, Б. MySQL. Оптимизация производительности / Б. Шварц, П. Зайцев, В. Ткаченко, Д. Заводны, А. Ленц и др. – 2-е изд., испр. и доп. – М.: Символ, 2010. – 708 с.
12. Гольцман, В. MySQL 5.0 / В. Гольцман. – СПб.: Питер, 2010. – 253 с.
13. Пособие по работе с базами данных на языке Python. – Дата обновления: 22.11.2014. URL: http://www.internet-technologies.ru/articles/article_2190.html (дата обращения: 28.01.2017).
14. Программирование с Qt : Часть 1. Введение. Инструменты разработчика и объектная модель. – Дата обновления: 27.08.2009
15. Дауни А. Цифровая обработка сигналов на языке Python. / А. Дауни. – М. : ДМК-Пресс, 2017. – 160 с.

ПРИЛОЖЕНИЕ ТЕКСТ ПРОГРАММЫ

Auth.py – класс, описывающий окно авторизации в программе. Содержит функции для создания подключения к базе данных и проверки корректности введенных данных

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(400, 151)
        self.pushButton = QtWidgets.QPushButton(Form)
        self.pushButton.setGeometry(QtCore.QRect(160, 110, 75, 23))
        self.pushButton.setObjectName("pushButton")
        self.label_3 = QtWidgets.QLabel(Form)
        self.label_3.setEnabled(True)
        self.label_3.setGeometry(QtCore.QRect(40, 80, 351, 20))
        font = QtGui.QFont()
        font.setUnderline(True)
        self.label_3.setFont(font)
        self.label_3.setAutoFillBackground(False)
        self.label_3.setObjectName("label_3")
        self.widget = QtWidgets.QWidget(Form)
        self.widget.setGeometry(QtCore.QRect(80, 30, 229, 50))
        self.widget.setObjectName("widget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.widget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.label = QtWidgets.QLabel(self.widget)
        self.label.setObjectName("label")
        self.verticalLayout.addWidget(self.label)
        self.label_2 = QtWidgets.QLabel(self.widget)
        self.label_2.setObjectName("label_2")
        self.verticalLayout.addWidget(self.label_2)
        self.horizontalLayout.addLayout(self.verticalLayout)
        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.lineEdit = QtWidgets.QLineEdit(self.widget)
        self.lineEdit.setObjectName("lineEdit")
        self.verticalLayout_2.addWidget(self.lineEdit)
        self.lineEdit_2 = QtWidgets.QLineEdit(self.widget)
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.verticalLayout_2.addWidget(self.lineEdit_2)
        self.horizontalLayout.addLayout(self.verticalLayout_2)

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Авторизация"))
        self.pushButton.setText(_translate("Form", "Войти"))
         self.label_3.setText(_translate("Form", "Введены неверные учетная запись и пароль. Попробуйте еще раз "))
        self.label.setText(_translate("Form", "Учетная запись:"))
        self.label_2.setText(_translate("Form", "Пароль:"))
```

```
    def connect(self):
def connect():
    """ Соединение с БД """
    try:
        conn = mysql.connector.connect(host='localhost',
                                       database='ecg_new',
                                       user='root',
                                       password='secret')
        if conn.is_connected():
            print('Connected to MySQL database')
            mw=mainWindow.Show()
            self.close()
    except Error as e:
        print(e)
```

MainWindow.py – класс, описывающий главное окно программы

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'interface.ui'
#
# Created by: PyQt5 UI code generator 5.9
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets
import sys

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(773, 587)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.layoutWidget = QtWidgets.QWidget(self.centralwidget)
        self.layoutWidget.setGeometry(QtCore.QRect(10, 80, 70, 72))
        self.layoutWidget.setObjectName("layoutWidget")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.layoutWidget)
        self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.pFIO = QtWidgets.QLabel(self.layoutWidget)
        self.pFIO.setObjectName("pFIO")
        self.verticalLayout_2.addWidget(self.pFIO)
        self.pAge = QtWidgets.QLabel(self.layoutWidget)
        self.pAge.setObjectName("pAge")
        self.verticalLayout_2.addWidget(self.pAge)
        self.pDate = QtWidgets.QLabel(self.layoutWidget)
        self.pDate.setObjectName("pDate")
        self.verticalLayout_2.addWidget(self.pDate)
        self.pPos = QtWidgets.QLabel(self.layoutWidget)
        self.pPos.setObjectName("pPos")
        self.verticalLayout_2.addWidget(self.pPos)
        self.layoutWidget1 = QtWidgets.QWidget(self.centralwidget)
        self.layoutWidget1.setGeometry(QtCore.QRect(100, 80, 57, 72))
        self.layoutWidget1.setObjectName("layoutWidget1")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.layoutWidget1)
        self.verticalLayout_3.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.fio_label = QtWidgets.QLabel(self.layoutWidget1)
        self.fio_label.setObjectName("fio_label")
        self.verticalLayout_3.addWidget(self.fio_label)
```

```
self.age_label = QtWidgets.QLabel(self.layoutWidget1)
self.age_label.setObjectName("age_label")
self.verticalLayout_3.addWidget(self.age_label)
self.label_4 = QtWidgets.QLabel(self.layoutWidget1)
self.label_4.setObjectName("label_4")
self.verticalLayout_3.addWidget(self.label_4)
self.label_10 = QtWidgets.QLabel(self.layoutWidget1)
self.label_10.setObjectName("label_10")
self.verticalLayout_3.addWidget(self.label_10)
self.quickWidget = QtQuickWidgets.QQuickWidget(self.centralwidget)
self.quickWidget.setGeometry(QtCore.QRect(180, 30, 581, 121))
self.quickWidget.setResizeMode(QtQuickWidgets.QQuickWidget.SizeRootObjectToView)
self.quickWidget.setObjectName("quickWidget")
self.label_11 = QtWidgets.QLabel(self.centralwidget)
self.label_11.setGeometry(QtCore.QRect(40, 0, 111, 31))
self.label_11.setObjectName("label_11")
self.buildSample = QtWidgets.QPushButton(self.centralwidget)
self.buildSample.setEnabled(False)
self.buildSample.setGeometry(QtCore.QRect(10, 180, 151, 31))
self.buildSample.setObjectName("buildSample")
self.openEcg = QtWidgets.QPushButton(self.centralwidget)
self.openEcg.setGeometry(QtCore.QRect(10, 50, 151, 23))
self.openEcg.setObjectName("openEcg")
self.textEdit = QtWidgets.QTextEdit(self.centralwidget)
self.textEdit.setGeometry(QtCore.QRect(180, 430, 571, 111))
self.textEdit.setObjectName("textEdit")
self.label_13 = QtWidgets.QLabel(self.centralwidget)
self.label_13.setGeometry(QtCore.QRect(70, 30, 47, 13))
self.label_13.setObjectName("label_13")
self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_2.setGeometry(QtCore.QRect(20, 520, 141, 23))
self.pushButton_2.setObjectName("pushButton_2")
self.layoutWidget2 = QtWidgets.QWidget(self.centralwidget)
self.layoutWidget2.setGeometry(QtCore.QRect(10, 230, 151, 154))
self.layoutWidget2.setObjectName("layoutWidget2")
self.horizontalLayout = QtWidgets.QHBoxLayout(self.layoutWidget2)
self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
self.horizontalLayout.setObjectName("horizontalLayout")
self.verticalLayout_4 = QtWidgets.QVBoxLayout()
self.verticalLayout_4.setObjectName("verticalLayout_4")
self.pPeakLabel = QtWidgets.QLabel(self.layoutWidget2)
self.pPeakLabel.setObjectName("pPeakLabel")
self.verticalLayout_4.addWidget(self.pPeakLabel)
self.qPeakLabel = QtWidgets.QLabel(self.layoutWidget2)
self.qPeakLabel.setObjectName("qPeakLabel")
self.verticalLayout_4.addWidget(self.qPeakLabel)
self.rPeakLabel = QtWidgets.QLabel(self.layoutWidget2)
self.rPeakLabel.setObjectName("rPeakLabel")
self.verticalLayout_4.addWidget(self.rPeakLabel)
self.qPeakLabel_2 = QtWidgets.QLabel(self.layoutWidget2)
self.qPeakLabel_2.setObjectName("qPeakLabel_2")
self.verticalLayout_4.addWidget(self.qPeakLabel_2)
self.tPeakLabel = QtWidgets.QLabel(self.layoutWidget2)
self.tPeakLabel.setObjectName("tPeakLabel")
self.verticalLayout_4.addWidget(self.tPeakLabel)
self.jTimeLabel = QtWidgets.QLabel(self.layoutWidget2)
self.jTimeLabel.setObjectName("jTimeLabel")
self.verticalLayout_4.addWidget(self.jTimeLabel)
self.horizontalLayout.addLayout(self.verticalLayout_4)
self.verticalLayout = QtWidgets.QVBoxLayout()
self.verticalLayout.setObjectName("verticalLayout")
self.pValue = QtWidgets.QLineEdit(self.layoutWidget2)
```

```
self.pValue.setObjectName("pValue")
self.verticalLayout.addWidget(self.pValue)
self.qValue = QtWidgets.QLineEdit(self.layoutWidget2)
self.qValue.setObjectName("qValue")
self.verticalLayout.addWidget(self.qValue)
self.rValue = QtWidgets.QLineEdit(self.layoutWidget2)
self.rValue.setObjectName("rValue")
self.verticalLayout.addWidget(self.rValue)
self.sValue = QtWidgets.QLineEdit(self.layoutWidget2)
self.sValue.setObjectName("sValue")
self.verticalLayout.addWidget(self.sValue)
self.tValue = QtWidgets.QLineEdit(self.layoutWidget2)
self.tValue.setObjectName("tValue")
self.verticalLayout.addWidget(self.tValue)
self.jValue = QtWidgets.QLineEdit(self.layoutWidget2)
self.jValue.setObjectName("jValue")
self.verticalLayout.addWidget(self.jValue)
self.horizontalLayout.addLayout(self.verticalLayout)
self.label_8 = QtWidgets.QLabel(self.centralwidget)
self.label_8.setGeometry(QtCore.QRect(460, 10, 47, 13))
self.label_8.setObjectName("label_8")
self.quickWidget_2 = QtQuickWidgets.QQuickWidget(self.centralwidget)
self.quickWidget_2.setGeometry(QtCore.QRect(180, 180, 291, 200))
self.quickWidget_2.setResizeMode(QtQuickWidgets.QQuickWidget.SizeRootObjectToView)
self.quickWidget_2.setObjectName("quickWidget_2")
self.quickWidget_3 = QtQuickWidgets.QQuickWidget(self.centralwidget)
self.quickWidget_3.setGeometry(QtCore.QRect(480, 180, 281, 200))
self.quickWidget_3.setResizeMode(QtQuickWidgets.QQuickWidget.SizeRootObjectToView)
self.quickWidget_3.setObjectName("quickWidget_3")
self.label_14 = QtWidgets.QLabel(self.centralwidget)
self.label_14.setGeometry(QtCore.QRect(270, 160, 131, 16))
self.label_14.setObjectName("label_14")
self.label_15 = QtWidgets.QLabel(self.centralwidget)
self.label_15.setGeometry(QtCore.QRect(580, 160, 131, 16))
self.label_15.setObjectName("label_15")
self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_3.setEnabled(False)
self.pushButton_3.setGeometry(QtCore.QRect(10, 400, 151, 23))
self.pushButton_3.setObjectName("pushButton_3")
self.label_16 = QtWidgets.QLabel(self.centralwidget)
self.label_16.setGeometry(QtCore.QRect(20, 430, 141, 16))
self.label_16.setObjectName("label_16")
self.label_17 = QtWidgets.QLabel(self.centralwidget)
self.label_17.setGeometry(QtCore.QRect(30, 450, 131, 16))
self.label_17.setObjectName("label_17")
self.label_18 = QtWidgets.QLabel(self.centralwidget)
self.label_18.setGeometry(QtCore.QRect(180, 400, 121, 16))
self.label_18.setObjectName("label_18")
self.label_19 = QtWidgets.QLabel(self.centralwidget)
self.label_19.setGeometry(QtCore.QRect(580, 400, 81, 16))
self.label_19.setObjectName("label_19")
self.label = QtWidgets.QLabel(self.centralwidget)
self.label.setGeometry(QtCore.QRect(670, 400, 81, 16))
self.label.setObjectName("label")
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 773, 21))
self.menubar.setObjectName("menubar")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)
```

```
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pFIO.setText(_translate("MainWindow", "Пациент:"))
        self.pAge.setText(_translate("MainWindow", "Возраст:"))
        self.pDate.setText(_translate("MainWindow", "Дата снятия:"))
        self.pPos.setText(_translate("MainWindow", "Отведение:"))
        self.fio_label.setText(_translate("MainWindow", "ФИО"))
        self.age_label.setText(_translate("MainWindow", "возраст"))
        self.label_4.setText(_translate("MainWindow", "дата"))
        self.label_10.setText(_translate("MainWindow", "отведение"))
        self.label_11.setText(_translate("MainWindow", "Вы вошли как"))
        self.buildSample.setText(_translate("MainWindow", "Определить параметры"))
        self.openEcg.setText(_translate("MainWindow", "Открыть ЭКГ"))
          self.textEdit.setHtml(_translate("MainWindow", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
    "<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
    "p, li { white-space: pre-wrap; }\n"
    "</style></head><body style=\" font-family:\'MS Shell Dlg 2\'; font-size:8.25pt; font-weight:400; font-style:normal;\">\n"
    "<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">В ЭТОМ МЕСТЕ ВРАЧ ПИШЕТ ДИАГНОЗ</p></body></html>"))
        self.label_13.setText(_translate("MainWindow", "ФИО"))
        self.pushButton_2.setText(_translate("MainWindow", "Сохранить"))
        self.pPeakLabel.setText(_translate("MainWindow", "P-зубец:"))
        self.qPeakLabel.setText(_translate("MainWindow", "Q-зубец:"))
        self.rPeakLabel.setText(_translate("MainWindow", "R-зубец:"))
        self.qPeakLabel_2.setText(_translate("MainWindow", "S-зубец:"))
        self.tPeakLabel.setText(_translate("MainWindow", "T-зубец:"))
        self.jTimeLabel.setText(_translate("MainWindow", "J  отклонение:"))
        self.label_8.setText(_translate("MainWindow", "ЭКГ:"))
        self.label_14.setText(_translate("MainWindow", "Эталонный цикл:"))
        self.label_15.setText(_translate("MainWindow", "Фазовая картина:"))
        self.pushButton_3.setText(_translate("MainWindow", "Предположить диагноз"))
        self.label_16.setText(_translate("MainWindow", "Предполагаемый диагноз:"))
        self.label_17.setText(_translate("MainWindow", "Название заболевания"))
        self.label_18.setText(_translate("MainWindow", "Заключение врача:"))
        self.label_19.setText(_translate("MainWindow", "Составил:"))
        self.label.setText(_translate("MainWindow", "ФИО врача"))
```

Ecg_analyzer.py – содержит в себе все необходимые классы для работы с кардиограммами: класс ЭКГ, класс графиков, класс показателей, класс матрицы Хаусдорфа

```
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib
import mysql.connector
from mysql.connector import Error
import csv

class ECG:
    """Класс ЭКГ"""
    def __init__(self, rows):
```

```
    '''Подается бинарная строка, извлеченная курсором'''
    f = open('samples.csv', 'wb')
    for item in rows:
        f.write(item)
    f.close()

    f = open('samples.csv', 'r')
    reader = csv.reader(f)
    self.signal = []
    self.time = []
    tmp = next(reader)
    self.dx = next(reader)[0]
    self.dx = float(self.dx[1:-4])
    for row in reader:
        self.time.append(float(row[0]) * self.dx)
        self.signal.append(float(row[1]))
    f.close()
    self.n=len(self.signal)

def derivative(self):
    '''Вычисляет значения производной, длина n-2'''
    self.dy=[]
    for i in range (0, self.n-2):
        self.dy.append((-3*self.signal[i]+4*self.signal[i+1]-self.signal[i+2])/(2*self.dx))

def separate(self):
    ''' сглаживаем разбиваем на циклы, смотря на разность значения производной'''
    self.smooth=[]
    ndy=len(self.dy)
    self.smooth.append((5 * self.dy[0] + 2 * self.dy[1] - 1 * self.dy[2]) / 6)
    for i in range(1, ndy-1):
        self.smooth.append((self.dy[i - 1] + self.dy[i] + self.dy[i + 1]) / 3)
    self.smooth.append((5 * self.dy[ndy-1] + 2 * self.dy[ndy-2] - 1 * self.dy[ndy-3]) / 6)
    dif=[]
    for i in range(0,ndy-1):
        dif.append(abs(self.smooth[i]-self.smooth[i+1])) #изначально должно быть smooth[i]
    peak=max(dif)
    self.cycles=[]
    for i in range (0,ndy-1):
        if dif[i]>peak*0.8:
            self.cycles.append(i)


    t1 = math.floor(0.25 / myecg.dx)
    t2 = math.floor(2.5 / myecg.dx)
    smax = 0
    tmax = 0
    for t in range(t1, t2):
        sum = 0
        for i in range(0, myecg.n - t):
            sum += myecg.signal[i] * myecg.signal[i + t]
        if sum > smax:
            smax = sum
            tmax = t
    self.cycles=[]
    i=self.signal.index(max(self.signal[0:math.floor(1.0/self.dx)]))
    self.cycles.append(i-math.floor(0.1/self.dx))
    i=1
    while self.cycles[0]+i*tmax<self.n:
        self.cycles.append(self.cycles[0]+i*tmax-13)
        i+=1
```

```
class Graph:
    '''для отображения графиков'''
    def __init__(self, ecg):
        self.y=ecg.signal
        self.t=ecg.time
        self.dy=ecg.dy
        self.cycles=ecg.cycles
    def simple_plot(self):
        fig = plt.figure()
        plt.plot(self.t, self.y,'r')
        list=[i*0.1 for i in range (0,100)]
        plt.xticks(list)
        plt.grid()
        plt.show()
    def der_plot(self):
        fig = plt.figure()
        norm=max(self.dy, key=abs)
        for i in range (0, len(self.dy)):
            self.dy[i]=self.dy[i]/norm
        plt.plot(self.t[0:-2], self.dy,'-or')
        plt.show()
    def cycles_plot(self):
        fig = plt.figure()
        for el in self.cycles:
            plt.scatter(self.t[el], self.y[el],color='black')
        plt.plot(self.t, self.y,'r')
        list=[i*0.1 for i in range (0,100)]
        plt.xticks(list)
        plt.grid()
        plt.show()
    def phase_plot(self):
        fig=plt.figure()
        for i in range (0,len(self.cycles)-1):

            begin=self.cycles[i]
            end=self.cycles[i+1]-1
            ymin=min(self.y[begin:end])
            ymax=max(self.y[begin:end])
            dymin=min(self.dy[begin:end])
            dymax=max(self.dy[begin:end])
            ycoeff=1/(ymax-ymin)
            dycoeff=1/(dymax-dymin)
            ny=[]
            ndy=[]
            for j in range (begin,end):
                ny.append((self.y[j]-ymin)*ycoeff)
                ndy.append((self.dy[j]-dymin)*dycoeff)
        plt.plot(ny,ndy,'r', linewidth=1.0)
        plt.grid()
        plt.show()

        '''
        for i in range (0,len(self.cycles)-1):
            begin=self.cycles[i]
            end=self.cycles[i+1]-1
            ymax=max(self.y[begin:end])
            dymax=max(self.dy[begin:end])
            ny=[]
            ndy=[]
            for j in range (begin,end):
                ny.append(1-math.exp(self.y[j]/ymax))
                ndy.append(1-math.exp(self.dy[j]/dymax))
```

```
            plt.plot(ny,ndy)
            plt.show()
            class Hausdorf():
        def __init__(self,ecg):
            self.n=len(ecg.cycles)
            rd=np.array(self.n,self.n)
            '''Считаем Хаусдорфовы расстояния'''
            for i in range (0,self.n):
                for j in range(0,self.n):
                    rij=max(min(math.sqrt((y[i]-y[j])**2+((dy[i]-dy[j])**2)))
                    rji = max(min(math.sqrt((y[i] - y[j]) ** 2 + ((dy[i] - dy[j]) ** 2)))
                    rd[i][j]=max(rij,rji)
            '''Определяем эталонный цикл'''
            sum=0
            minsum=9999
            imin=0
            for i in range (0,n):
                for j in range (0,n)):
                    sum+=rd[i][j]
                if minsum<sum:
                    imin=i
    '''Усредняем значения'''
        self.yet=[]
        self.dyet=[]
        for i in range (0,t):
            sumy=0
            sumdy=0
            for j in range (0,t):
                if math.sqrt((y[i]-y[j])**2+((dy[i]-dy[j])**2))<rd[j]:\
                    sumy+=y[i]
                    sumdy+=dy[j]
            self.yet.append(sumy/len(sumy))
            self.yet.append(sumy / len(sumy))

class QRS:
    """Выделяем показатели ЭКГ"""
    def __init__(self,etalon):
        self.y=etalon.yet
        self.dy=etalon.dyet
        self.iso=sum(self.y[0:5])/5
        self.y[:]=abs([x-self.iso for x in self.y]))
        self.dy[:] = abs([x - self.iso for x in self.dy]))
        self.rpeak=max(y)
        self.rtime=self.y.index(self.rpeak)
        self.speak=max(dy[:self.rtime])
        self.stime=self.y.index(self.speak)
        self.tpeak=max(dy[:self.stime])
        self.ttime=self.y.index(self.tpeak)
        self.qpeak=math.locmin(self.[:self.rtime],-1)
        self.qtime=self.y.index(self.qpeak)
        self.ppeak=math.locmin(self.[:self.rtime],-2)
        self.ptime=self.y.index(self.ppeak)
        self.jtime=(rtime-qtime)
```

Хранимые процедуры, расположенные в базе данных. Обеспечивают проверку логина и пароля, загрузку информации из таблиц для отображения в формах.

```
-- загружает фио при авторизации, проверяет логин-пароль
SELECT fio
FROM doctor
```

```
WHERE login=log and password=pass

--загружает список пациентов
SELECT fio, birthdate
FROM patient

--загружает список кардиограмм выбранного пациента
SELECT date, diversion
FROM cardio
WHERE patient_id=pat_id

--загружает информацию о пациенте и кардиограмме в главное окно
DECLARE @age
SELECT @age=date
FROM cardio
WHERE patient_id=pat_id

SELECT fio, birth_date-@age, diversion, date, ecg
FROM cardio, patient
WHERE cardio.patient_id=patient.patient_id
AND cardio.patient_id=pat_id
--загружает значения параметров
SELECT *
FROM parametrs, cardio
WHERE cardio.par_id=parametrs.pat_id
AND ecg_id=ecg

--сохраняет результат обработки экг
UPDATE cardio
SET diagnosis=result, doctor_id=doc
WHERE ecg_id=ecg

INSERT into parametrs
values p,q,r,s,t,j
WHERE par_id=par

--построение дерева
SELECT *
FROM node
WHERE parent_id=par_id
```