|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Направление подготовки (специальность) | 09.03.04 Программная инженерия |
| Выпускающая кафедра | Математическое и программное обеспечение электронно-вычислительных машин |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

|  |  |
| --- | --- |
| Название работы | Разработка программного обеспечения для построения и |
| интеграции модели Random Forest | |
|  | |
|  | |
|  | |

|  |  |
| --- | --- |
| Студента | Осичева Дмитрия Павловича |
|  | Ф.И.О. |
|  | |

ДОПУСТИТЬ К ЗАЩИТЕ

|  |  |  |
| --- | --- | --- |
| Директор института |  | Е.В. Ершов |
| Заведующий выпускающей кафедрой |  | Е.В. Ершов |
| Руководитель выпускной квалификационной работы |  | Е.В. Ершов |
| Нормоконтролер |  | Л.Н. Виноградова |
| Выпускник |  | Д.П. Осичев |

# Оглавление

[Введение 6](#_Toc484073153)

[1. Сравнительный анализ отечественных и зарубежных аналогов проектируемой системы 8](#_Toc484073154)

[2. Выбор технологии, среды и языка программирования 9](#_Toc484073155)

[2.1 Выбор модели жизненного цикла 9](#_Toc484073156)

[2.2 Выбор технологии проектирования 11](#_Toc484073157)

[2.3 Выбор подхода к разработке 12](#_Toc484073158)

[2.4 Выбор языка и среды программирования 12](#_Toc484073159)

[2.5 Выбор CASE – средств 14](#_Toc484073160)

[3. Анализ процесса обработки информации, выбор структур данных для её хранения, выбор методов и алгоритмов решения задачи 14](#_Toc484073161)

[4. Разработка спецификаций проектируемой системы 17](#_Toc484073162)

[4.1 Построение диаграммы вариантов использования 17](#_Toc484073163)

[4.2 Построение контекстной диаграммы классов 25](#_Toc484073164)

[4.3 Построение диаграмм последовательностей системы для основных вариантов использования 27](#_Toc484073165)

[4.4 Построение диаграмм деятельностей сценариев вариантов использования 35](#_Toc484073166)

[4.5 Проектирование структур данных и построение диаграмм отношений компонентов данных 37](#_Toc484073167)

[5. Проектирование системы 38](#_Toc484073168)

[5.1 Проектирование структуры и построение диаграмм пакетов 38](#_Toc484073169)

[5.2 Проектирование классов 41](#_Toc484073170)

[5.2.1 Проектирование классов пакета «Model» 41](#_Toc484073171)

[5.2.2 Проектирование классов пакета «ViewModel» 47](#_Toc484073175)

[5.3 Построение диаграммы компонентов 56](#_Toc484073181)

[5.4 Спецификация модулей 57](#_Toc484073182)

[5.5 Построение диаграммы размещения 73](#_Toc484073187)

[6. Проектирование интерфейса пользователя 73](#_Toc484073188)

[6.1 Построение графа диалогов 73](#_Toc484073189)

[6.2 Разработка форм ввода-вывода информации 75](#_Toc484073190)

[7. Выбор стратегии тестирования, разработка тестов, программа и методика испытаний 79](#_Toc484073191)

[7.1. Объект испытаний 79](#_Toc484073192)

[7.2. Цель испытаний 79](#_Toc484073193)

[7.3 Требования к программному обеспечению 79](#_Toc484073194)

[7.4 Требование к документации 79](#_Toc484073195)

[7.5 Состав и порядок испытаний 80](#_Toc484073196)

[7.5.1 Состав испытаний 80](#_Toc484073197)

[7.5.2 Порядок проведения испытаний 80](#_Toc484073198)

[7.6 Методы испытаний 81](#_Toc484073199)

[7.7 Результаты тестирования 81](#_Toc484073200)

[7.7.1 Модульное тестирование 81](#_Toc484073201)

[7.7.2 Системное тестирование 90](#_Toc484073202)

[Заключение 93](#_Toc484073203)

[Список литературы 95](#_Toc484073204)

[Приложение 1. Техническое задание 98](#_Toc484073205)

[Приложение 2. Текст программы 106](#_Toc484073220)

[Приложение 3. Спецификации 127](#_Toc484073221)

[Приложение 4. Руководство пользователя 128](#_Toc484073222)

[Приложение 5. Набор тестовых данных 144](#_Toc484073223)

# Введение

В настоящее время в промышленности активно используются технологии машинного обучения, которые позволяют с помощью анализа исторических данных о производстве, а также информации, которая доступна в реальном времени от датчиков, сенсоров и других устройств повысить эффективность и автоматизацию производства.

На данный момент существуют различные методы машинного обучения с различным уровнем сложности. Одним из самых сложных методов являются нейросети, которые являются универсальным аппроксиматором. Однако в большинстве случаев используются простые методы: метод ближайших соседей, дерево принятия решения, Random Forest. Для таких методов осуществляется быстрая настройка данных, что позволяет за короткое время получить результаты прогнозирования. Кроме того данные методы машинного обучения легче интерпретировать — используя более простую и наглядную модель, что в свою очередь позволяет понять логику работу алгоритма и объяснить, почему для каждого набора признаков был получен тот или иной прогноз [9].

На сегодняшний день Random Forest является одним из самых популярных и крайне эффективных методов решения задач машинного обучения, таких как классификация и регрессия. По эффективности он конкурирует с машинами опорных векторов, нейронными сетями и бустингом.

В основе Random Forest лежит дерево принятия решений. Дерево принятия решений – это средство поддержки принятия решений при прогнозировании, широко применяющееся в статистике и анализе данных. Ветви (ребра графа) хранят в себе значения атрибутов, от которых зависит целевая функция; на листьях же записывается значение целевой функции. Существуют также и другие узлы – родительские и потомки – по которым происходит разветвление, и можно различить случаи [10]. При построении Random Forest строиться ансамбль деревьев принятия решения, результат которых, на конечном этапе, усредняется.

Наиболее пригодным для построения модели является язык программирования R. Данный язык служит для статистической обработки данных и является свободным программным обеспечением.

Ввиду того, что полученную модель с помощью языка R, можно использовать только в этом языке, то возникает проблема внедрения модели в программные продукты, которые используются на производстве.

В настоящее время наиболее популярными программными продуктами на производстве является СУБД Oracle и Visual Studio, которая позволяет разрабатывать .NET приложения.

Основным назначением разрабатываемого программного обеспечения, является генерация и трансляция модели Random Forest в SQL-код, C#-код и DLL библиотеку, с помощью языка R.

## Сравнительный анализ отечественных и зарубежных аналогов проектируемой системы

С развитием информационных систем, а также автоматизацией многих процессов стали очень популярны алгоритмы машинного обучения. В большинстве процессов, которые происходят на производстве, разбирается технолог. В таком случае необходима система, которая без особых знаний специфичных языков программирования и изучений сложных систем позволит построить модель и внедрить ее на производство, а именно в базу данных Oracle или в проект в Visual Studio от Microsoft.

На сегодняшний день для обработки данных можно выделить язык программирования R. Данный язык является бесплатным и имеет множество библиотек. Но для того чтобы работать в нем необходимо изучать специфичный для этого языка синтаксис. Также не существует метода для интеграции модели в базу данных, для непосредственной работы с данными и в проекты Visual Studio.

Вторым программным решением является Oracle R Enterprise. Данная система представляет собой базу данных Oracle с интегрированным языком R [11]. Такое решение позволяет использовать возможности языка R, непосредственно в базе данных, где хранятся данные, поступающие с производства. Ввиду того Oracle R Enterprise транслирует все команды и функции R в SQL процедуры, пользователю необходимо знать синтаксис языка R [12]. Так как Oracle R Enterprise является базой данных, то нет возможности использования модели в проекте Visual Studio. Основным недостатком данной системы является большая стоимость: 593 100 рублей (10450 $) [13].

IBM SPSS Modeler - это платформа прогнозной аналитики, позволяющая применять прогнозную информацию при принятии решений на уровне отдельных пользователей, групп, систем или предприятия [14].

Для работы с данной системой не надо изучать специальные языки программирования, что позволяет работать с ней технологу. Также в данной системе реализовано большое количество методов для анализа данных. В качестве недостатков можно выделить: не способность дальнейшей интеграции, полученной модели, в базы данных и программные проекты, большая стоимость системы – 249 726 – 624 316 рублей (4 400 – 11 000 $) [15].

Основными недостатками всех представленных решений являются большая стоимость и сложность работы с ними. Исходя из этого, было принято решение разработки приложения, которое позволит без особых знаний специализированных языков программирования построить модель машинного обучения, а именно Random Forest, и внедрить ее в свою базу данных Oracle и программный проект .NET.

## Выбор технологии, среды и языка программирования

### Выбор модели жизненного цикла

При проектировании и разработке программного обеспечения используется понятие жизненного цикла (ЖЦ).

Жизненный цикл (ЖЦ) информационной системы – это непрерывный процесс, начинающийся с момента принятия решения о создании информационной системы и заканчивающийся в момент полного изъятия ее из эксплуатации [16].

В настоящее время существует большое разнообразие моделей ЖЦ разработки программного обеспечения. Наиболее популярными моделями ЖЦ, являются каскадная и спиральная модель.

Каскадная модель (рис. 1) предусматривает четко последовательную организацию разработки ПО. При этом весь процесс разработки разбивается на этапы: разработка и анализ требований; проектирование; реализация; тестирование; ввод в действие [5]. Переход к выполнению следующего этапа осуществляется только тогда, когда все работы на предыдущем этапе будут полностью завершены.



Рис. 1. Каскадная модель

Данная модель подходит для проектов, в которой требования к программному обеспечению (ПО) четко определенны на ранних этапах проектирования и с большой вероятностью не будут изменяться в дальнейшем.

Спиральная модель (рис. 2) предполагает итерационный процесс разработки информационной системы. Данная модель обычно ориентируется на случай, когда требования проекта постоянно корректируются. Заказчик работы и исполнитель, в таком случае, работают в постоянном сотрудничестве.

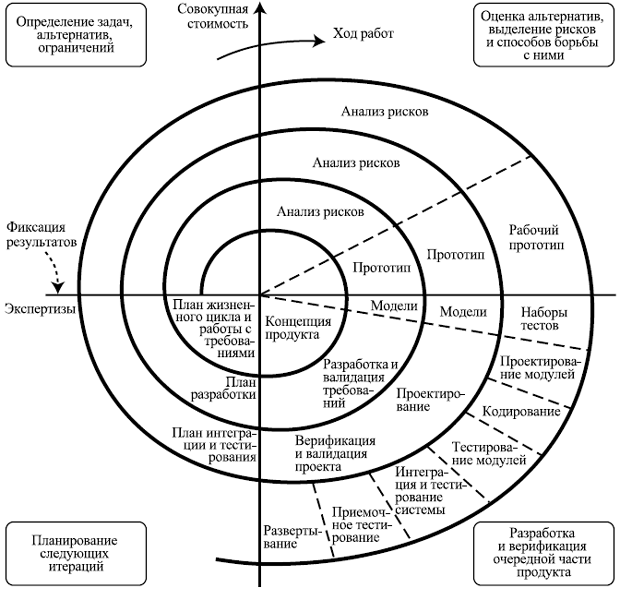


Рис. 2. Спиральная модель

Каждый виток спирали соответствует созданию фрагмента или версии программного изделия, на нем уточняются цели и характеристики проекта, определяется его качество, планируются работы на следующем витке спирали [5].

Для разработки данного программного обеспечения предъявляются следующие требования:

1. короткие сроки разработки;
2. меняющиеся или дополняющие требования к программному продукту;
3. быстрая разработка прототипа.

Исходя из данных требований, была выбрана спиральная модель жизненного цикла программного продукта. Данная модель обладает следующим рядом преимуществ [2]:

1. итерационная разработка существенно упрощает внесение изменений в проект при изменении требований заказчика;
2. при использовании спиральной модели отдельные элементы информационной системы интегрируются в единое целое постепенно;
3. уменьшения уровня рисков, так как риски обнаруживаются во время интеграции;
4. итерационная разработка обеспечивает большую гибкость в управлении проектом, давая возможность внесения тактических изменений в разрабатываемую информационную систему;
5. данная модель позволяет получить более надежную и устойчивую систему.

### Выбор технологии проектирования

При проектировании программного обеспечения для генерации и интеграции модели Random Forest было решено использовать методологию быстрой разработки приложений RAD (Rapid Application Development). RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементного прототипирования [17].

Известно, что методология RAD поддерживает спиральную модель ЖЦ, поэтому выбор данной методологии оправдан в рамках данного проекта. Одним из основных принципов данной методологии является использование CASE-средств.

### Выбор подхода к разработке

Основными подходами к разработке ПО являются структурный и объектный [1].

В основе структурного подхода лежит принцип декомпозиции – разбиения на части сложных систем с целью последующей реализации в виде отдельных подпрограмм.

Объектно-ориентированный подход основан на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию с наследованием свойств. Взаимодействие осуществляется путем передачи сообщений.

Для разрабатываемого ПО легко выделить основные объекты предметной области, и определить их взаимодействие, поэтому выбран объектно-ориентированный подход к программированию.

Для проектирования ПО был выбран унифицированный язык моделирования UML - стандартный язык для написания моделей анализа, проектирования и реализации объектно-ориентированных программных систем [7].

## Выбор языка и среды программирования

На данный момент существует большое множество различных объектно-ориентированных языков программирования. Одним из популярных на сегодняшний день языков программирования является C#.

C# – объектно-ориентированный язык программирования. Он относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML [18].

Для разработки ПО была выбрана платформа пользовательского интерфейса - Windows Presentation Foundation (WPF). Данная платформа служит для создания интерактивных клиентских приложений [19]. В WPF используется расширяемый язык разметки для приложений (XAML). При использовании WPF, чаще всего применяют паттерн MVVM (Model-View-ViewModel). Шаблон MVVM имеет три основных компонента: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса XAML, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления [20].

В качестве среды разработки была выбрана Microsoft Visual Studio 2013. Microsoft Visual Studio — продукт компании Microsoft, включающий интегрированную среду разработки программного обеспечения и ряд других инструментальных средств [21]. Данная среда позволяет разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Presentation Foundation.

Для реализации основного функционала приложения используется язык программирования R. R — язык программирования для статистической обработки данных и работы с графикой [22].

В качестве среды разработки для языка R была выбрана RStudio. Данная среда включает в консоль; редактор, с поддержкой прямого выполнения кода; инструменты для отладки и управления рабочим проектом [23].

## Выбор CASE – средств

Для автоматизации процессов проектирования ПО целесообразно использовать CASE-средства. CASE-средства представляют собой программные средства, поддерживающие процессы создания и сопровождения приложения, включая анализ и формулировку требований, проектирования прикладного ПО, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы [3].

В связи с тем, что выбран объектный подход к разработке, оправдано использование современный редактор Software Ideas Modeler.

Software Ideas Modeler – это программа для моделирования диаграмм UML (Unified Modeling Language). UML – это язык графического описания для объектного моделирования в области разработки программного обеспечения, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. Также поддерживается генерация кода при выполнении модели UML [24].

## Анализ процесса обработки информации, выбор структур данных для её хранения, выбор методов и алгоритмов решения задачи

В настоящее время методы машинного обучения применяются для решения задач классификации и регрессии. Решить задачу классификации – это значит определить, к какому классу относиться объект множества по определенным входным параметрам. Суть решения задачи регрессии сводиться к получению вещественного значения на основании обучающих примеров.

На сегодняшний день разработано большое количество алгоритмов машинного обучения для решения задач классификации и регрессии. Наиболее популярными считаются: метод опорных векторов (SVM); метод ближайших соседей; нейронные сети; Random Forest.

Метод опорных векторов (Support Vector Machine) был предложен Вапниковым В. Н. и служит для решения задачи классификации. Цель данного метода – найти плоскость, которая будет разделять два множества объектов. Из преимуществ метода выделяют [28]:

* быстрота нахождения решающей функции;
* нахождения разделяющей полосы максимальной ширины, что позволяет получить более точную классификацию.

Из недостатков выделяют следующее:

* метод чувствителен к шумам;
* не существует общего подхода к автоматическому выбору ядра;
* медленное обучение.

Метод ближайших соседей является простейшим метрическим классификатором, который основан на оценивании сходства объектов. Исследуемый объект относиться к тому классу, к которому принадлежат ближайшие к нему объекты обучающей выборки. К преимуществам данного метода относят:

* простота использования результатов.

К недостаткам относят:

* сложность выбора меры «близости»;
* сложность вычисления при распозновании;
* способность решать задачи с небольшим количеством классов и переменных.

Нейронные сети относятся к универсальным алгоритмам. Нейронные сети состоят из искуственных нейронов, которые преобразуют несколько входных фактов в один выходной. К преиуществам данного метода относят:

* устойчивость к шумам входных данных;
* адаптирование к изменениям окружающей среды;
* быстродействие.

Из недостатков выделяют следующее:

* сложная настройка;
* обучение сети может привести к тупиковым ситуациям;
* результат обученной нейронной сети не всегда может быть однозначно предзказуем, что увеличивает риск использования данного метода на производстве.

Алгоритм Random Forest был предложен Лео Брейманом и Адель Катлером и был реализован в R Анди Льюав и Мэтью Винером.

Метод основан на построении большого количества деревьев принятия решений. Дерево решений – метод с помощью которого можно получить прогнозируемое значение, полученное в результате оценки входных атрибутов.

Random Forest служит для решения задач классификации и регрессии. В задачи регрессии ответы решающих деревьев усредняется, в классификации принимается решение голосованием по большинству [25]. Определяющим фактором, от которого зависит тип модели, является выходное значение. Если выходной параметр является непрерывным значением, то решается задачи регресси, а если выходной параметр имеет дискретное значение, то решается задача классификации.

Преимущества данного метода:

* высокая скорость обучения;
* масштабируемость (способность обрабатывать большие объемы данных);
* малое количество настраиваемых параметров;

К недостаткам относят:

* большой объем занимаемой памяти.

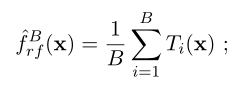
Для обучения Random Forest должны быть предусмотрены примеры данных. Такие данные могут быть подготовлены экспертом или накоплены в ходе выполнения определенной работы.

Алгоритм построения случайного леса представлен в следующем виде [8]:

1. Для *i = 1,2, …, B* (*B* – количество деревьев в ансамбле) выполнить

* Сформировать бутстреп выборку *S* размера *l* по исходной обучающей выборке ;
* По бутстреп выборке *S* индуцировать неусеченное дерево решений *Ti* с минимальным количеством наблюдений в терминальных вершинах равным *nmin*, рекурсивно следуя следующему подалгоритму:
  + 1. из исходного набора *n* признаков случайно выбрать *p* признаков;
    2. из *p* признаков выбрать тот признак, который обеспечивает наилучшее расщепление (расщепление (split) – правило разбиения множества, связанное с каждой вершиной дерева решений);
    3. расщепить выборку, соответствующую обрабатываемой вершине, на две подвыборки.

1. В результате 1 шага получается ансамбль деревьев решений ;
2. Предсказание новых наблюдений осуществлять следующим образом:
3. для регрессии:



1. для классификации:

пусть *ωi*(x) ∈ {*ω1, ω2,..., ωc*} – класс, предсказанный деревом решений *Ti*; тогда – класс, наиболее часто встречающийся в множестве .

## 4. Разработка спецификаций проектируемой системы

### 4.1 Построение диаграммы вариантов использования

Разработка спецификаций программного обеспечения начинается с анализа требований к функциональности, в соответствии с техническим заданием. В процессе анализа выявляют внешних пользователей разрабатываемого программного обеспечения и перечень отдельных аспектов его поведения в процессе взаимодействия с конкретным пользователем [4].

ВИ представляет собой характерную процедуру применения разрабатываемой системы конкретным действующим лицом, в качестве которого могут выступать пользователи или другие системы. На диаграмме вариантов использования представлены только отношения между субъектами. Данная диаграмма не указывает порядок выполнения шагов для достижения цели варианта использования.

Диаграмма вариантов использования для данной разработки представлена на рис. 3, описание представлено в табл. 1-14.

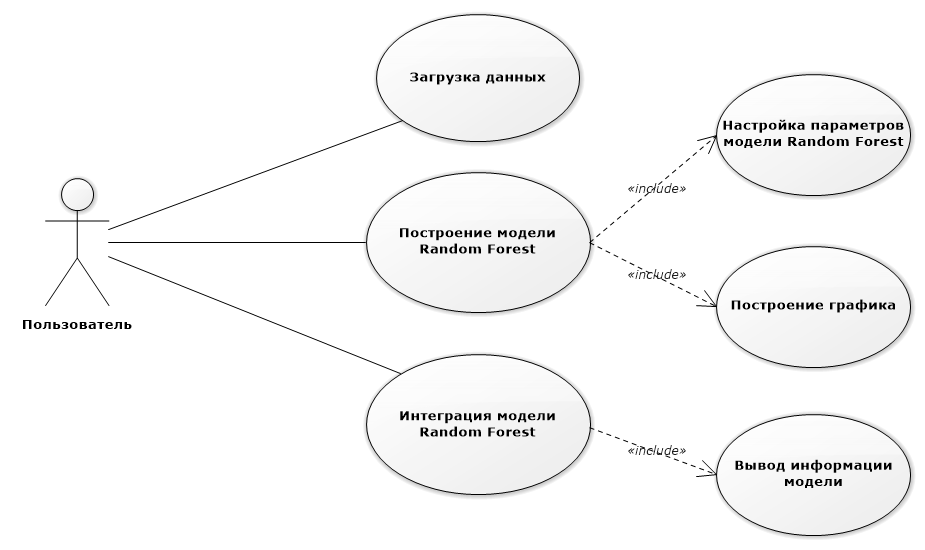


Рис. 3. Диаграмма вариантов использования

Действующие лица:

Пользователь – лицо, использующее программное обеспечения для построения и интеграции модели Random Forest.

Варианты использования:

* загрузка данных;
* построение модели Random Forest;
* интеграция модели Random Forest;
* настройка параметров модели Random Forest;
* построение графика;
* вывод информации модели.

Описание варианта использования «Загрузка данных» представлено в табл. 1, 2.

Таблица 1

Краткое описание варианта «Загрузка данных»

|  |  |
| --- | --- |
| Название ВИ | Загрузка данных |
| Цель | Загрузка и отображение данных для построения модели |
| Действующие лица | Пользователь |
| Краткое описание | Предоставляет пользователю возможность загрузки и отображение данных в формате xls, xlsx и csv |
| Тип варианта | Основной |

Таблица 2

Типичный ход событий для сценария «Загрузка данных»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1) Пользователь выбирает опцию «Загрузка данных» | 2) Система загружает форму выбора файла |
| 3) Пользователь выбирает файл | 4) Система загружает данные в таблицу на форме для просмотра |

Описание варианта использования «Построение модели Random Forest» (табл. 3, 4).

Таблица 3

Краткое описание варианта «Построение модели Random Forest»

|  |  |
| --- | --- |
| Название ВИ | Построение модели Random Forest |
| Цель | Построение модели Random Forest по загруженным данным |
| Действующие лица | Пользователь |
| Краткое описание | Предоставляет пользователю возможность построения модели Random Forest, с указанием параметров |
| Тип варианта | Основной |

Таблица 4

Типичный ход событий для сценария «Построение модели Random Forest»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1) Пользователь выбирает опцию «Построить модель Random Forest» | 2) Система загружает форму построения модели Random Forest |
| 3) Пользователь указывает необходимые параметры | 4) Система генерирует модель Random Forest, отображает график и загружает форму для сохранения модели Random Forest |
| 5) Пользователь указывает имя и место для сохранения построенной модели Random Forest | 6) Система сохраняет построенную модель |

Описание варианта использования «Интеграция модели Random Forest» (табл. 5-8).

Таблица 5

Краткое описание варианта «Интеграция модели Random Forest»

|  |  |
| --- | --- |
| Название ВИ | Интеграция модели |
| Цель | Интеграция модели Random Forest в SQL и C# код |
| Действующие лица | Пользователь |
| Краткое описание | Предоставляет пользователю возможность транслированить модель Random Forest для использования в базе данных Oracle и в .NET проектах |
| Тип варианта | Основной |

Альтернатива 1: Генерация SQL кода (табл. 6).

Альтернатива 2: Генерация С# кода (табл. 7).

Альтернатива 3: Генерация dll библиотеки (табл. 8).

Таблица 6

Типичный ход событий для сценария «Генерация SQL кода»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1 | 2 |
| 1) Пользователь выбирает опцию меню «Интеграция модели» | 2) Система загружает форму для интеграции модели |
| 3) Пользователь выбирает опцию «Обзор» для загрузки модели | 4) Система загружает форму для выбора файла в формате RData |
| 5) Пользователь выбирает необходимый файл и нажимает «Открыть» | 6) Система загружает выбранный файл и отображает информацию и график по данной модели |
| 7) Пользователь выбирает команду трансляции модели в SQL код и нажимает опцию «Выполнить» | 8) Система загружает форму для ввода имени таблицы |
| 9) Пользователь указывает имя таблицы, в которую будет сохранена модель Random Forest | 10) Система транслирует модель в SQL код и загружает форму для сохранения кода |

Продолжение табл. 6

|  |  |
| --- | --- |
| 1 | 2 |
| 11) Пользователь указывает имя и место для сохранения файла | 12) Система сохраняет код |

Таблица 7

Типичный ход событий для сценария «Генерация С# кода»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1) Пользователь выбирает опцию меню «Интеграция модели» | 2) Система загружает форму для интеграции модели |
| 3) Пользователь выбирает опцию «Обзор» для загрузки модели | 4) Система загружает форму для выбора файла в формате Rdata |
| 5) Пользователь выбирает необходимый файл и нажимает «Открыть» | 6) Система загружает выбранный файл и отображает информацию и график по данной модели |
| 7) Пользователь выбирает команду трансляции модели в C# код и нажимает опцию «Выполнить» | 8) Система загружает форму для ввода имени класса |
| 9) Пользователь указывает имя класса, для сохранения модели Random Forest | 10) Система транслирует модель в набор файлов С# кода и загружает форму для сохранения кода |
| 11) Пользователь указывает место для сохранения файлов | 12) Система сохраняет файлы с C# кодом |

Таблица 8

Типичный ход событий для сценария «Генерация dll библиотеки»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1 | 2 |
| 1) Пользователь выбирает опцию меню «Интеграция модели» | 2) Система загружает форму для интеграции модели |

Продолжение табл. 8

|  |  |
| --- | --- |
| 1 | 2 |
| 3) Пользователь выбирает опцию «Обзор» для загрузки модели | 4) Система загружает форму для выбора файла в формате RData |
| 5) Пользователь выбирает необходимый файл и нажимает «Открыть» | 6) Система загружает выбранный файл и отображает информацию и график по данной модели |
| 7) Пользователь выбирает команду трансляции модели в dll библиотеку и нажимает опцию «Выполнить» | 8) Система загружает форму для ввода имени класса |
| 9) Пользователь указывает имя класса, для сохранения модели Random Forest | 10) Система транслирует модель в dll библиотеку и загружает форму для сохранения |
| 11) Пользователь указывает имя и место для сохранения библиотеки | 12) Система сохраняет dll библиотеку |

Описание варианта использования «Настройка параметров модели Random Forest» (табл. 9, 10).

Таблица 9

Краткое описание варианта «Настройка параметров модели Random Forest»

|  |  |
| --- | --- |
| Название ВИ | Настройка параметров модели Random Forest |
| Цель | Настройка модели Random Forest |
| Действующие лица | Пользователь |
| Краткое описание | Предоставляет пользователю возможность указать необходимые параметры для построения модели Random Forest |
| Тип варианта | Дополнительный |

Таблица 10

Типичный ход событий для сценария «Настройка параметров модели Random Forest»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1) Пользователь указывает количество деревьев, целевую переменную и тип модели (классификация или регрессия) | 2) Система применяет параметры для выборки |

Описание варианта использования «Построение графика» (табл. 11, 12).

Таблица 11

Краткое описание варианта «Построение графика»

|  |  |
| --- | --- |
| Название ВИ | Построение графика |
| Цель | Вывод графика построенной модели |
| Действующие лица | Пользователь |
| Краткое описание | Выводит график зависимости ошибки от количества деревьев принятия решений |
| Тип варианта | Дополнительный |

Таблица 12

Типичный ход событий для сценария «Построение графика»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1) Пользователь выбирает входной файл, выполняет настройку Random Forest и выполняет построение модели | 2) Система отображает график |

Описание варианта использования «Вывод информации модели»   
(табл. 13, 14).

Таблица 13

Краткое описание варианта «Вывод информации модели»

|  |  |
| --- | --- |
| Название ВИ | Вывод информации модели |
| Цель | Вывод информации по загруженной модели Random Forest |
| Действующие лица | Пользователь |
| Краткое описание | Выводит пользователю информацию по выбранной модели Random Forest |
| Тип варианта | Дополнительный |

Таблица 14

Типичный ход событий для сценария «Построение графика»

|  |  |
| --- | --- |
| Действие пользователя | Отклик системы |
| 1) Пользователь выбирает файл с моделью Random Forest | 2) Система выводит количество деревьев принятия решений, тип задачи и целевую переменную |

### 

### 4.2 Построение контекстной диаграммы классов

На рис. 4 показана контекстная диаграмма классов. Контекстная диаграмма классов демонстрирует связи между основными понятиями предметной области. На этапе проектирования происходит уточнение контекстной диаграммы классов с помощью уточненных диаграмм классов и детальных диаграмм классов.

Описание классов приведено в табл. 15, описание связей представлено в табл. 16.

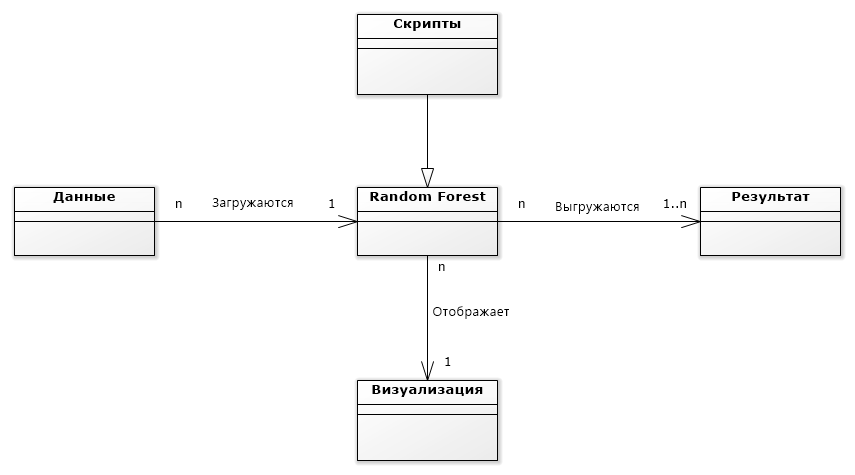


Рис. 4. Контекстная диаграмма классов

Таблица 15

Основные понятия концептуальной модели предметной области

|  |  |
| --- | --- |
| Класс | Описание |
| Данные | Содержит методы для работы с данными: загрузка, конвертация, визуализация |
| Random Forest | Содержит методы для генерации и трансляции модели Random Forest |
| Результаты | Позволяет сохранять полученные результаты, полученные на этапе генерации и трансляции модели |
| Визуализация | Предоставляет возможность отображать график модели Random Forest |
| Скрипты | Содержит методы, разработанные на языке R для трансляции Random Forest в SQL код, C# код. |

Таблица 16

Описание связей

|  |  |
| --- | --- |
| Связь | Описание |
| Данные – Random Forest | Набор данных загружается в Random Forest |
| Random Forest – Визуализация | Множество построенных деревьев принятия решений отображаются на графике |
| Random Forest – Результат | Сгенерированная модель Random Forest, а также ее трансляция в SQL-код, C# код и DLL библиотеку передаются для сохранения во внешней памяти |
| Скрипты - Random Forest | Random Forest использует скрипты на языке R для выполнения трансляции модели |

### 

### 4.3 Построение диаграмм последовательностей системы для основных вариантов использования

Диаграмма последовательностей системы (sequence diagram) – графическая модель, которая для определённого сценария варианта использования показывает генерируемые действующими лицами события и их порядок. При этом система рассматривается как единое целое.

Диаграмма последовательности системы для варианта использования «Загрузка данных» представлена на рис. 5, операции описаны в табл. 17.



Рис. 5. Диаграмма последовательностей системы для варианта использования «Загрузка данных»

Таблица 17

Описание операции «Загрузка файла»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Загрузка файла |
| Обязанности | Загрузка и отображение данных из файла |
| Тип | Системная |
| Ссылки | Вариант использования «Загрузка данных» |
| Примечания | - |
| Исключения | - |
| Вывод | - |
| Предусловия | - |
| Постусловие | Вывод данных в виде таблице на экран |

Диаграмма последовательности системы для варианта использования «Построение модели Random Forest» представлена на рис. 6, операции описаны в табл. 18-23.

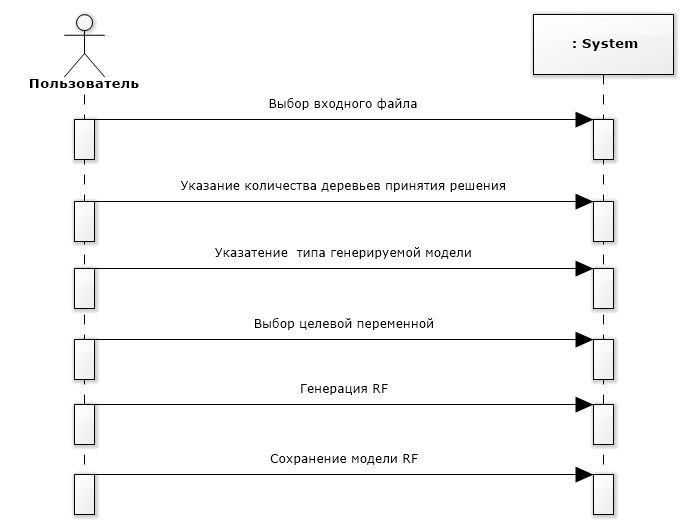


Рис. 6. Диаграмма последовательностей системы для варианта использования «Построение модели Random Forest»

Таблица 18

Описание операции «Выбор входного файла»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Выбор входного файла |
| Обязанности | Указания пути к входному файлу |
| Тип | Системная |
| Ссылки | Вариант использования «Построение модели Random Forest» |
| Примечания | - |
| Исключения | - |
| Вывод | Отображение пути к файлу на форме |
| Предусловия | - |
| Постусловие | **-** |

Таблица 19

Описание операции «Указание количества деревьев принятия решений»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Указание количества деревьев принятия решений |
| Обязанности | Внесение в систему информацию о количестве деревьев принятия решений |
| Тип | Системная |
| Ссылки | Вариант использования «Построение модели Random Forest» |
| Примечания | - |
| Исключения | Указание только положительного целого числа |
| Вывод | - |
| Предусловия | - |
| Постусловие | - |

Таблица 20

Описание операции «Указание типа генерируемой модели»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Указание типа генерируемой модели |
| Обязанности | Внесение в систему информацию о типе генерируемой модели |
| Тип | Системная |
| Ссылки | Вариант использования «Построение модели Random Forest» |
| Примечания | - |
| Исключения | - |
| Вывод | - |
| Предусловия | - |
| Постусловие | - |

Таблица 21

Описание операции «Выбор целевой переменной»

|  |  |
| --- | --- |
| Раздел | Описание |
| 1 | 2 |
| Имя | Выбор целевой переменной |
| Обязанности | Внесение в систему информацию о переменной, по которой будет строиться Random Forest |
| Тип | Системная |
| Ссылки | Вариант использования «Построение модели Random Forest» |
| Примечания | Загрузить все возможные переменные из входного файла |
| Исключения | - |
| Вывод | - |

Продолжение табл. 21

|  |  |
| --- | --- |
| 1 | 2 |
| Предусловия | Должен быть выбран входной файл |
| Постусловие | - |

Таблица 22

Описание операции «Генерация RF»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Генерация RF |
| Обязанности | Построение модели Random Forest |
| Тип | Системная |
| Ссылки | Вариант использования «Построение модели Random Forest» |
| Примечания | - |
| Исключения | - |
| Вывод | 1. Вывод индикатора выполнения трансляции 2. Построение графика модели RF |
| Предусловия | Должен быть выбран входной файл и указаны все параметры |
| Постусловие | - |

Таблица 23

Описание операции «Сохранение модели RF»

|  |  |
| --- | --- |
| Раздел | Описание |
| 1 | 2 |
| Имя | Сохранение модели RF |
| Обязанности | Предоставляет возможность сохранения модели Random Forest |
| Тип | Системная |

Продолжение табл. 23

|  |  |
| --- | --- |
| 1 | 2 |
| Ссылки | Вариант использования «Построение модели Random Forest» |
| Примечания | - |
| Исключения | - |
| Вывод | - |
| Предусловия | Построена модель Random Forest |
| Постусловие | - |

Диаграмма последовательности системы для варианта использования «Интеграция модели Random Forest» представлена на рис. 7, операции описаны в табл. 24-27

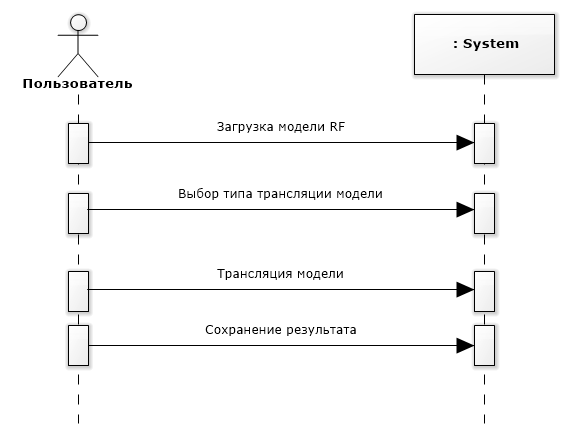


Рис. 7. Диаграмма последовательностей системы для варианта использования «Трансляция модели Random Forest»

Таблица 24

Описание операции «Загрузка модели RF»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Загрузка модели RF |
| Обязанности | Указания пути к файлу с моделью RF |
| Тип | Системная |
| Ссылки | Вариант использования «Трансляция модели Random Forest» |
| Примечания | - |
| Исключения | - |
| Вывод | Отображение информации, выбранной модели RF |
| Предусловия | - |
| Постусловие | **-** |

Таблица 25

Описание операции «Выбор типа трансляции модели»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Выбор типа трансляции модели |
| Обязанности | Внесение в систему информацию о типе генерируемой модели |
| Тип | Системная |
| Ссылки | Вариант использования «Трансляция модели Random Forest» |
| Примечания | - |
| Исключения | - |
| Вывод | - |
| Предусловия | - |
| Постусловие | - |

Таблица 26

Описание операции «Трансляция модели»

|  |  |
| --- | --- |
| Раздел | Описание |
| Имя | Трансляция модели |
| Обязанности | Транслирование модели в SQL код, C# код или DLL библиотеку |
| Тип | Системная |
| Ссылки | Вариант использования «Трансляция модели Random Forest» |
| Примечания | Предусмотреть возможность ввода названия таблицы или названия класса для трансляции модели в SQL код или C# код соответственно |
| Исключения | Название должно быть указано на латинском языке |
| Вывод | Вывод индикатора выполнения трансляции |
| Предусловия | Указан файл с моделью RF |
| Постусловие | - |

Таблица 27

Описание операции «Сохранение результата»

|  |  |
| --- | --- |
| Раздел | Описание |
| 1 | 2 |
| Имя | Сохранение результата |
| Обязанности | Предоставляет возможность сохранения транслируемой модели Random Forest |
| Тип | Системная |
| Ссылки | Вариант использования «Трансляция модели Random Forest» |
| Примечания | - |
| Исключения | - |

Продолжение табл. 27

|  |  |
| --- | --- |
| 1 | 2 |
| Вывод | - |
| Предусловия | - |
| Постусловие | - |

### 

### 4.4 Построение диаграмм деятельностей сценариев вариантов использования

Диаграмма деятельностей (activity diagram) описывает бизнес-процесс или программный алгоритм как последовательность действий. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов. Стрелки-соединители показывают, как управление последовательно передается от одного действия к другому. Обычно действие может начаться только после завершения предыдущего действия [4].

Диаграмма деятельности для варианта использования «Загрузка данных» представлена на рис. 8.

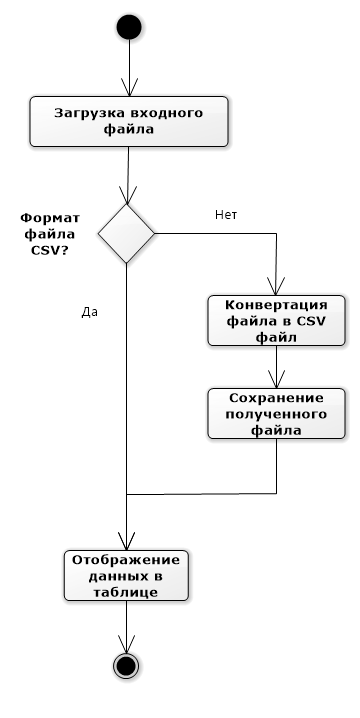


Рис. 8. Диаграмма деятельностей для варианта использования «Загрузка данных»

После выбора входного файла выполняется его загрузка в систему. Загружать можно файлы в формате xls, xlsx, csv. В случаи если файл представлен в формате xls или xlsx, то он автоматически конвертируется в формат csv и сохраняется рядом с исходным файлом. После данные из файла csv выводятся в таблицу.

Диаграмма деятельности для варианта использования «Построение модели Random Forest» представлена на рис. 9.

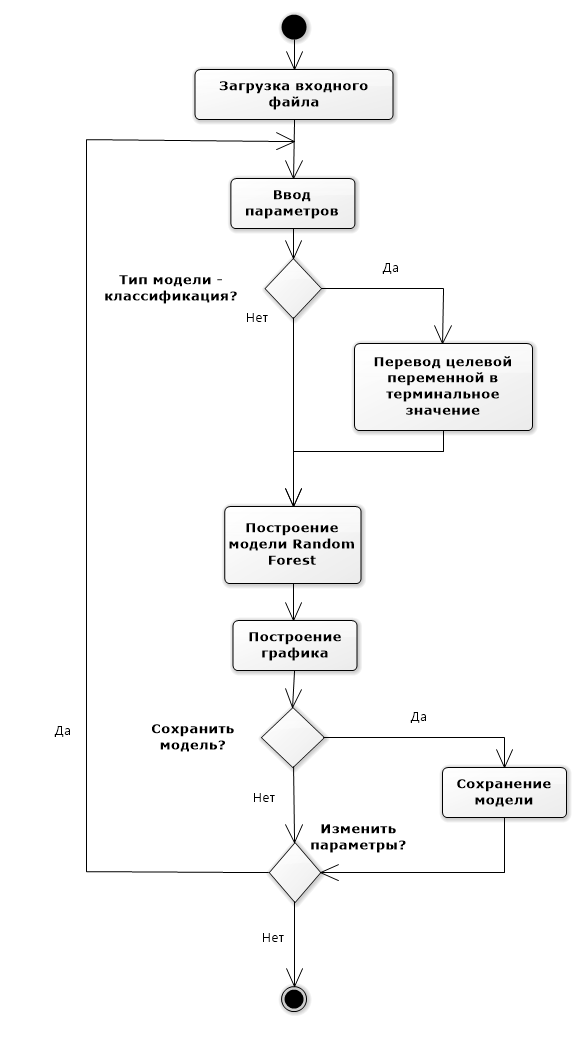


Рис. 9. Диаграмма деятельностей для варианта использования «Построение модели Random Forest»

После загрузки входного файла, необходимо указать параметры для построения Random Forest. В случае если решается задача классификации, то выбранная целевая переменная будет переведена в терминальное значение. Потом строиться сама модель Random Forest и ее график зависимости ошибки прогнозирования от количества деревьев принятия решений. В результате, полученную модель можно сохранить на локальном диске. Также, можно перестроить модель с другими параметрами.

### 4.5 Проектирование структур данных и построение диаграмм отношений компонентов данных

Одной из задач данного ПО является транслирование модели Random Forest в SQL код, который потом может быть запущен в базе данных (БД) Oracle. Данный SQL код создает в базе данных таблицу, куда сохраняет все данные по построенной модели Random Forest.

Таблица для хранения модели Random Forest имеет следующие поля: ID, номер дерева (TREE\_NUM), имя параметра (PARAM\_NAME), значение условия перехода (COND\_VALUE), ссылка на узел дерева при истинном результате выполнения условия (TRUE\_ID), ссылка на узел дерева при ложном результате выполнения условия (TRUE\_ID), тип узла (NODE\_TYPE), значение для терминального узла (VALUE).

На рис. 10 представлена структура таблицы в базе данных для хранения деревьев, построенных в процессе построения модели Random Forest.

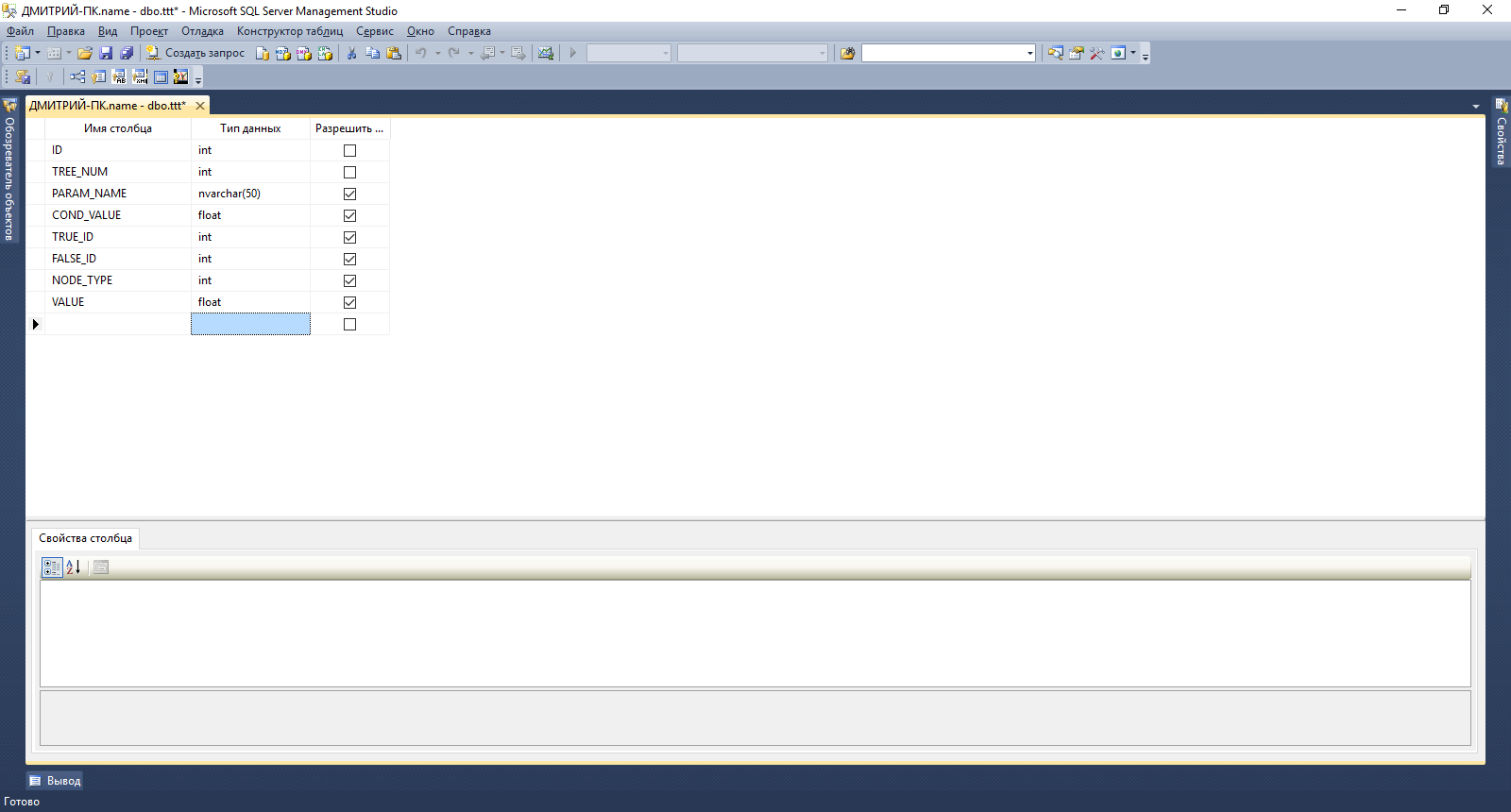


Рис. 10. Структура таблицы для интеграции модели в БД Oracle

## 5. Проектирование системы

Одна из задач логического проектирования при объектном подходе является разработка классов с полным описанием полей и методов, разрабатываемых классов.

На этапе физического проектирования происходит объединение классов и других программных ресурсов в программные компоненты.

### 5.1 Проектирование структуры и построение диаграмм пакетов

Диаграмма пакетов – это диаграмма, основным содержанием которой являются пакеты и отношения между ними. Жёсткого разделения между разными диаграммами не проводится, поэтому данное название предлагается исключительно для удобства и не имеет семантического значения (пакеты и диаграммы пакетов могут присутствовать на других структурных диаграммах). Диаграммы пакетов служат, в первую очередь, для организации элементов в группы по какому-либо признаку с целью упрощения структуры и организации работы с моделью системы. В случае сложных систем, состоящих из множества подсистем, применение диаграмм пакетов упрощает проектирование и делает возможным поэтапную реализацию системы из подсистем, которые соответствуют пакетам [4].

В терминах программирования пакеты в UML соответствуют таким группирующим конструкциям, как пакеты в Java и пространства имен в .NET [26]. Каждый пакет представляет пространство имен (namespace), а это означает, что каждый класс внутри собственного пакета должен иметь уникальное имя.

На рис. 11 представлена диаграмма пакетов для данной предметной области, описание пакета представлено в табл. 28.

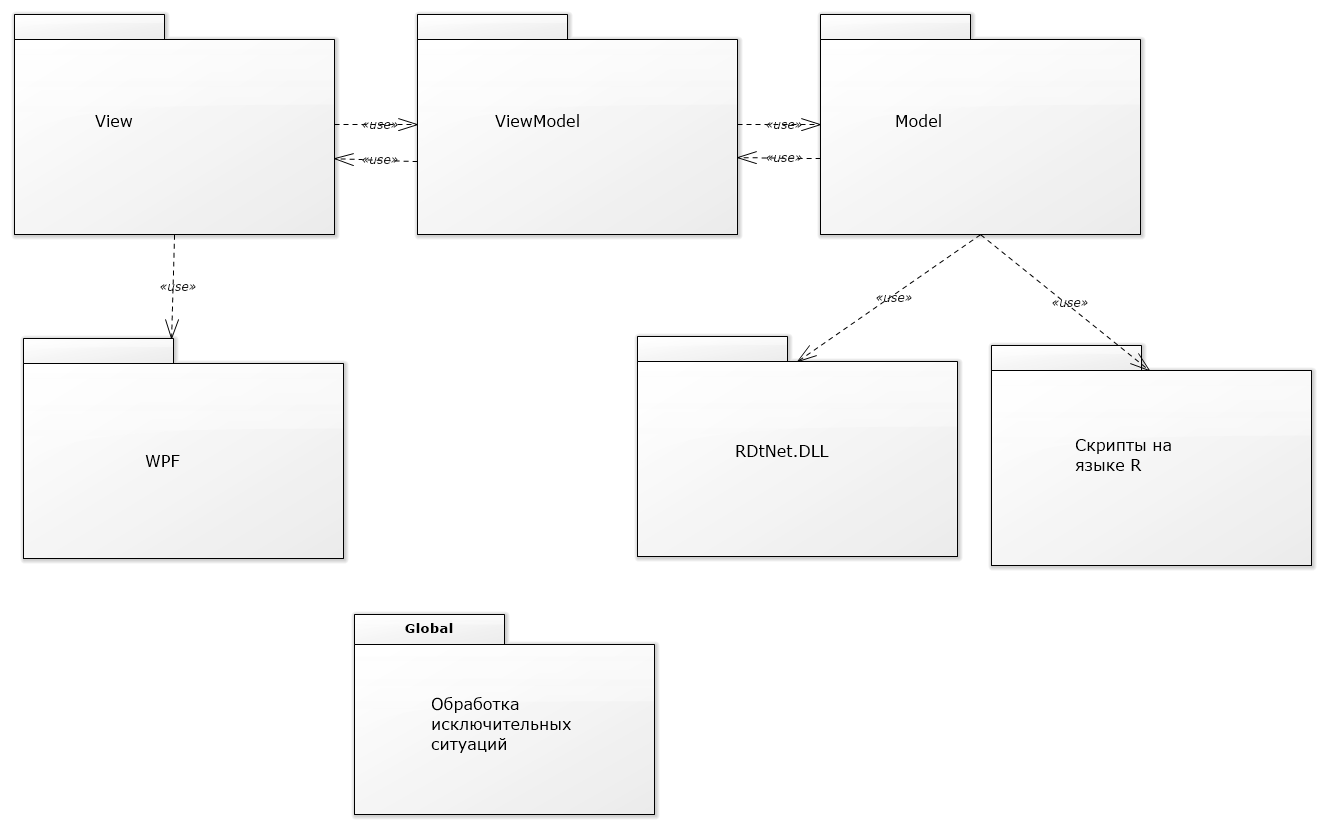


Рис. 11. Диаграмма пакетов

Таблица 28

Описание пакетов

|  |  |
| --- | --- |
| Наименование | Описание |
| 1 | 2 |
| View | Содержит классы, отвечающие за пользовательский интерфейс программного обеспечения |
| Model | Содержит классы, описывающие предметную область программного обеспечения |
| ViewModel | Содержит классы, отвечающие за логуки программного обеспечения и взаимодействие между пакетом View и Model |

Продолжение табл. 28

|  |  |
| --- | --- |
| 1 | 2 |
| WPF | Содержит классы для разработки Windows приложений |
| RDоtNet.DLL | Библиотека компонентов для использования языка R в С# |
| Скрипты на языке R | Содержит методы на языке R, которые выполняют трансляцию модели Random Forest |
| Обработка исключительных ситуаций | Отвечает за обработку ошибок, возникших во время выполнения программного обеспечения |

Пакет «Обработка исключительных ситуаций» является глобальным, так как обработка исключительных ситуаций применяется во всех пакетах.

Пакет «View» связан с пакетом «ViewModel», так как выполнения операции, относящейся к обработке данных из пакета «Model», происходит через управляющий пакет «ViewModel». Также «ViewModel» отправляет результат выполнения операций на пользовательский интерфейс, который представлен пакетом «View».

Пакет «View» связан с пакетами «WPF», так как «WPF» содержит элементы, которые служат для построения пользовательского интерфейса.

Пакет «Model» представляет собой библиотеку классов. Он взаимосвязан с внешним пакетом «RDоtNet.DLL», так как для выполнения некоторых функций используются методы из данного пакета. Также для выполнения трансляции модели используется внешний пакет «Скрипты на языке R», который он подгружает. Используя классы пакета «Model» в классах пакета «ViewModel» создаются объекты.

## 5.2 Проектирование классов

### 5.2.1 Проектирование классов пакета «Model»

### 5.2.1.1 Построение исходной диаграммы классов

Пакет «Model» представляет собой библиотеку классов, которая необходима для работы приложения. Данный пакет содержит функционал приложения.

На рис. 12 представлена исходная диаграмма классов пакета «Model», описание классов представлено в таблице 29.

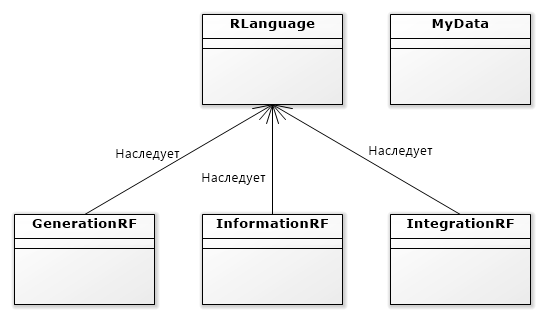


Рис. 12. Исходная диаграмма классов пакета «Model»

Таблица 29

Описание классов пакета «Model»

|  |  |
| --- | --- |
| Название класса | Описание |
| 1 | 2 |
| RLanguage | Класс содержит в себе поля и функции для инициализации и работы с языком R |

Продолжение табл. 29

|  |  |
| --- | --- |
| 1 | 2 |
| MyData | Класс служит для работы с данными: загрузка, конвертация xls и xlsx в csv |
| GenerationRF | Класс служит для генерации модели Random Forest |
| InformationRF | Класс содержит в себе методы для отображения графиков и основной информации модели Random Forest |
| IntegrationRF | Класс содержит в себе методы для трансляции модели Random Forest, полученной на шаге генерации, в SQL код, С# код или dll библиотеку. |

### 

### 5.2.1.2 Построение уточненной диаграммы классов

Данная диаграмма строится для уточнения отношений между классами. На рис. 13 представлена уточенная диаграмма классов пакета «Model».

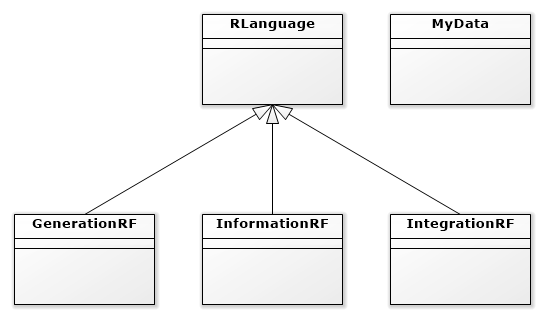


Рис. 13. Уточненная диаграмма классов пакета «Model»

Классы «GenerationRF», «InformationRF» и «IntegrationRF» связаны с классом «RLanguage» типом обобщения, так как эти классы «GenerationRF», «InformationRF» и «IntegrationRF» наследуются от класса «RLanguage».

Класс «MyData» не связан с остальными классами пакета, так как нет прямого взаимодействия с этими классами.

### 5.2.1.3 Разработка детальной диаграммы классов

При детальном проектировании классов описываются атрибуты и методы классов. Детальная диаграмма классов пакета «Model» представлена на рис. 14, описание данных классов представлено в табл. 30-38.

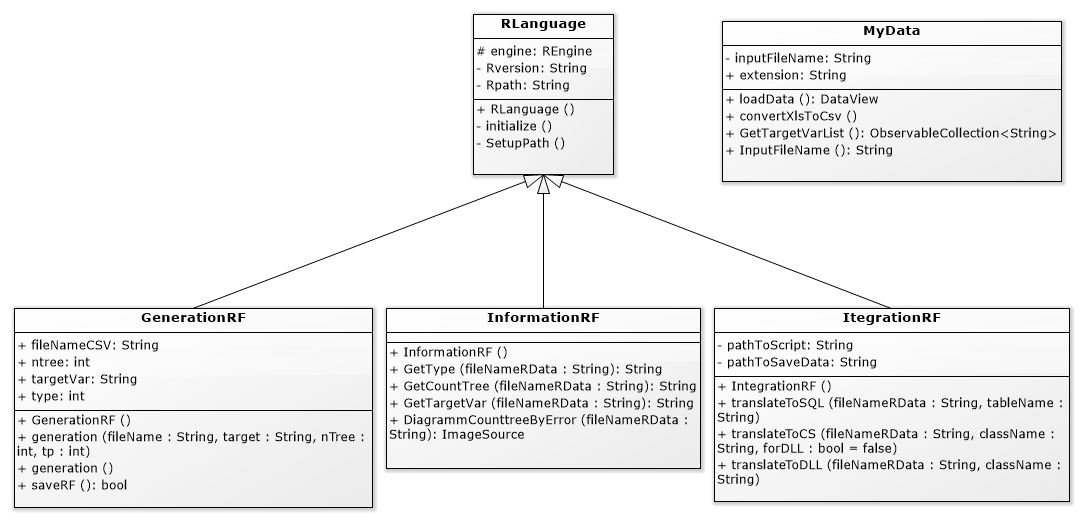


Рис. 14. Детальная диаграмма классов пакета «Model»

Таблица 30

Описание полей класса «RLanguage»

|  |  |
| --- | --- |
| Название поля | Назначение |
| REngine engine | Переменная, для работы с языком R |
| String Rversion | Версия языка R |
| String Rpath | Путь, где лежат библиотеки R |

Таблица 31

Описание методов класса «RLanguage»

|  |  |
| --- | --- |
| Название метода | Назначение |
| RLanguage() | Конструктор класса по умолчанию |
| void initialize() | Инициализация языка R |
| void SetupPath() | Установка переменной среды для языка R |

Таблица 32

Описание полей класса «GenerationRF»

|  |  |
| --- | --- |
| Название поля | Назначение |
| String fileNameCSV | Путь к входному файлу в формате CSV |
| int ntree | Количество деревьев принятия решений |
| String targetVar | Целевая переменная |
| int type | Тип задачи (классификации (0) или регрессии (1)) |

Таблица 33

Описание методов класса «GenerationRF»

|  |  |
| --- | --- |
| Название метода | Назначение |
| GenerationRF() | Конструктор класса по умолчанию |
| void generation() | Генерация модели Random Forest |
| void generation(String fileName, String target, int nTree, int tp) | Перегрузка метода generation() |
| bool saveRF() | Сохранение полученной модели Random Forest |

Таблица 34

Описание методов класса «InformationRF»

|  |  |
| --- | --- |
| Название метода | Назначение |
| InformationRF() | Конструктор класса по умолчанию |
| String GetType(String fileNameRData) | Возвращает тип, указанной модели Random Forest |
| String GetCountTree(String fileNameRData) | Возвращает количество деревьев принятия решений, указанной модели Random Forest |
| String GetTargetVar(String fileNameRData) | Возвращает целевую переменную, указанной модели Random Forest |
| ImageSource DiagrammCounttreeByError(String fileNameRData) | Возвращает изображение графика зависимости количества деревьев принятия решений от ошибки построенной модели, указанной модели Random Forest |

Таблица 35

Описание полей класса «IntegrationRF»

|  |  |
| --- | --- |
| Название поля | Назначение |
| String pathToScript | Путь к скриптам на языке R для выполнения трансляции модели в SQL код или C# код |
| String pathToSaveData | Путь для системного сохранения файлов, полученных после трансляции модели |

Таблица 36

Описание методов класса «IntegrationRF»

|  |  |
| --- | --- |
| Название метода | Назначение |
| IntegrationRF() | Конструктор класса по умолчанию |
| void translateToSQL(String fileNameRData, String tableName) | Трансляция модели Random Forest для указанной таблицы в SQL-код |
| translateToCS(String fileNameRData, String className, bool forDLL = false) | Трансляция модели Random Forest для указанного класса в C#-код |
| translateToDLL(String fileNameRData, String className) | Трансляция модели Random Forest для указанного класса в DLL библиотеку |
| bool saveSQL() | Сохранения полученного SQL-кода на локальном диске |
| bool saveCsharp(String className) | Сохранения полученного набора файлов С#-кода на локальном диске |
| bool saveDLL(String className) | Сохранения полученной DLL библиотеки на локальном диске |

Таблица 37

Описание полей класса «MyData»

|  |  |
| --- | --- |
| Название поля | Назначение |
| String inputFileName | Переменная для хранения полного имени входного файла |
| String extension | Переменная для хранения расширения входного файла |

Таблица 38

Описание методов класса «MyData»

|  |  |
| --- | --- |
| Название метода | Назначение |
| String InputFileName (object value) | Возвращает и принимает полное имя входного файла |
| DataView loadData() | Загрузка данных из входного файла |
| void convertXlsToCsv() | Конвертация xls или xlsx файла в csv файл |
| ObservableCollection<String> | Коллекция названий столбцов csv файла |

### 

### 5.2.2 Проектирование классов пакета «ViewModel»

### 5.2.1.1 Построение исходной диаграммы классов

Диаграмма классов описывает типы объектов системы и различного рода статические отношения, которые существуют между ними.

На рис. 15 представлена исходная диаграмма классов, описание классов представлено в табл. 39.

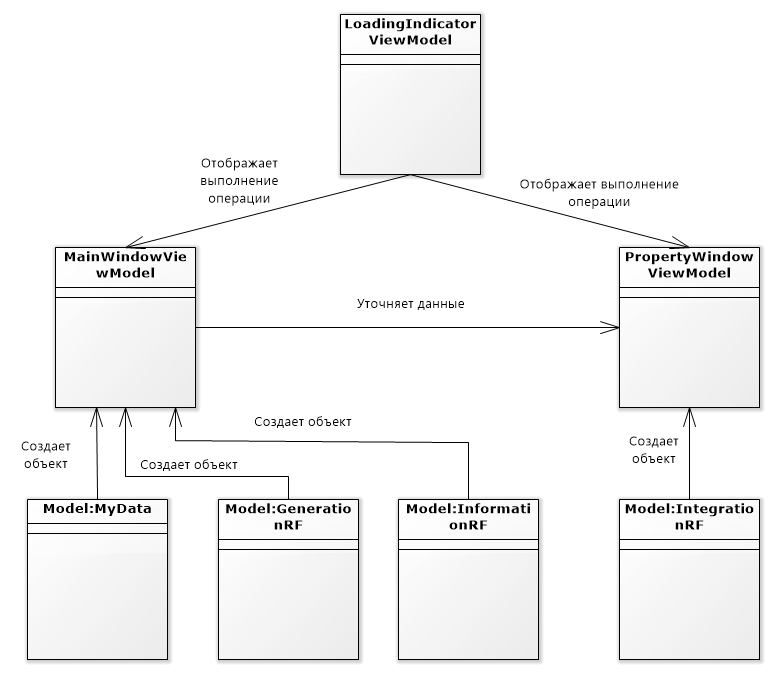


Рис. 15. Исходная диаграмма классов пакета «ViewModel»

Таблица 39

Описание классов пакета «ViewModel»

|  |  |
| --- | --- |
| Название класса | Описание |
| MainWindowViewModel | Класс предоставляет набор методов и свойств, позволяющий реализовать основную логику данной программы. Данный класс осуществляет взаимодействие пользовательского интерфейса с функциональной логикой приложения из пакета «Model» |
| PropetryWindowViewModel | Данный класс содержит методы и свойства, которые позволяют уточнить входные данные на определенном шаге, также осуществляет взаимодействие функциональной логикой приложения из пакета «Model» |
| LoadingIndicatorViewModel | Класс представляет собой набор методов и свойств для визуального отображения процесса выполнения операции |

### 

### 5.2.2.2 Построение диаграмм последовательностей действий

Диаграмма последовательностей системы (sequence diagram) – графическая модель, которая для определённого сценария варианта использования показывает генерируемые действующими лицами события и их порядок. При этом система рассматривается как единое целое [4].

На рис. 16-18 представлены диаграммы последовательностей действий для варианта использования «Интеграция модели Random Forest»

На рис. 16 представлена диаграмма нормального хода событий работы программного обеспечения, при которой пользователь правильно выполняет следующую последовательность действий:

1. загрузка файла;
2. указание типа трансляции модели;
3. ввод названия таблицы или класса, в которую будет транслирована модель;
4. сохранение результата.

На рис. 17 показана диаграмма для случая, когда пользователь прерывает выполнения трансляции модели. На шаге ввода имени таблицы или класса, пользователь закрывает окно, тем самым отправив команду объекту класса MainWindowViewModel об остановке трансляции модели.

На рис. 18 изображена диаграмма, когда во время работы возникает ошибка загрузки скрипта. Процесс, в котором происходит исключительная ситуация, выполняет безопасное завершение и передает на главное окно сообщение об ошибке.

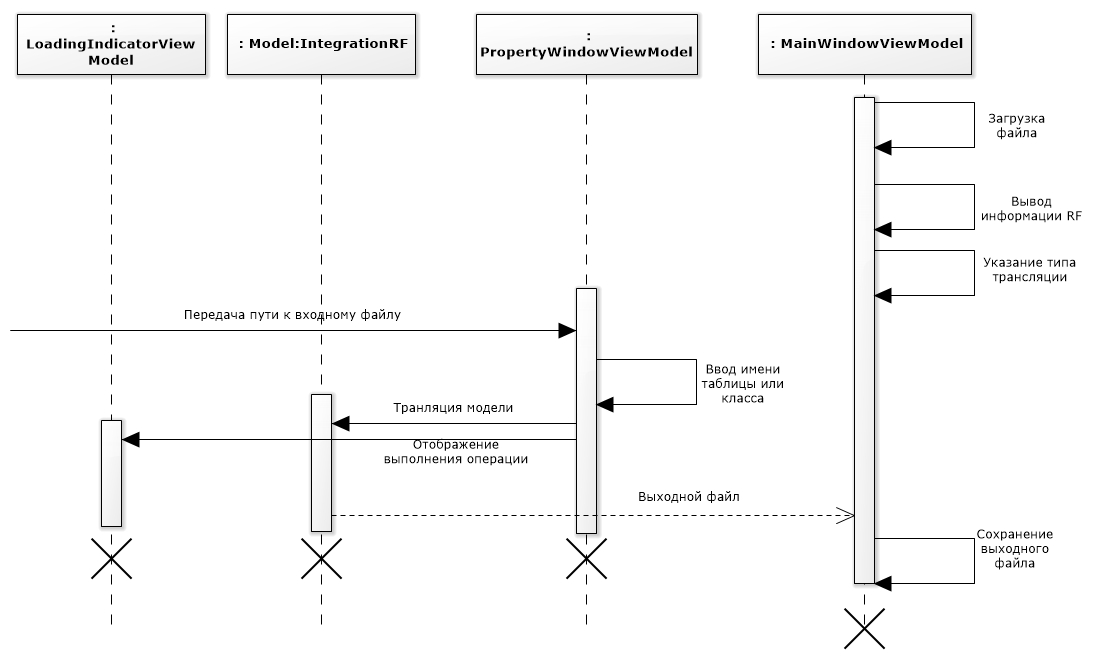


Рис. 16. Диаграмма последовательностей действий нормального хода событий

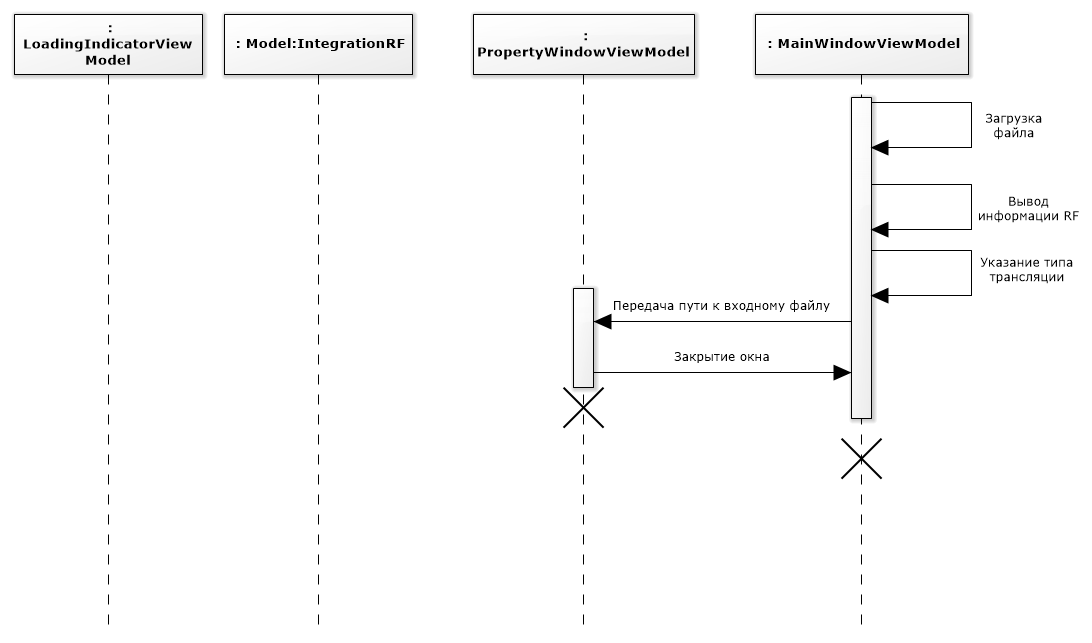


Рис. 17. Диаграмма последовательностей действий прерывания процесса пользователем

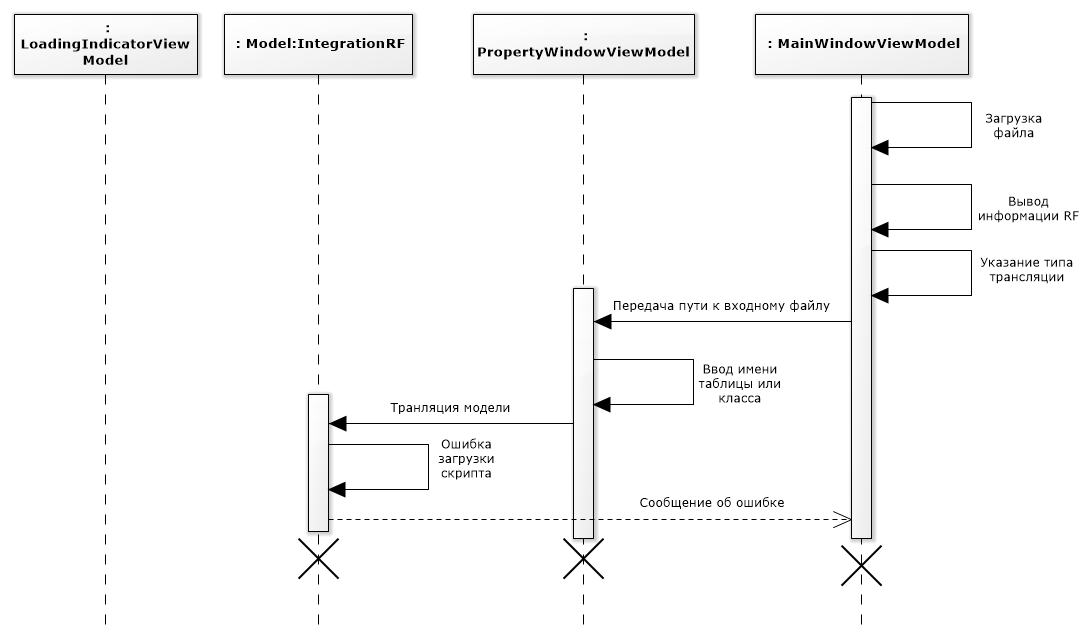


Рис. 18. Диаграмма последовательностей действий возникновения нештатной ситуации

### 5.2.2.3 Построение диаграммы кооперации

Диаграмма кооперации - это альтернативный способ представления взаимодействия объектов в процессе реализации сценария, который позволяет по-другому взглянуть на ту же информацию. В отличие от диаграмм последовательностей диаграммы кооперации показывают потоки данных между объектами классов, что позволяет уточнить связи между ними [4].

Диаграмма кооперации для пакета «ViewModel» представлена на рис. 19.

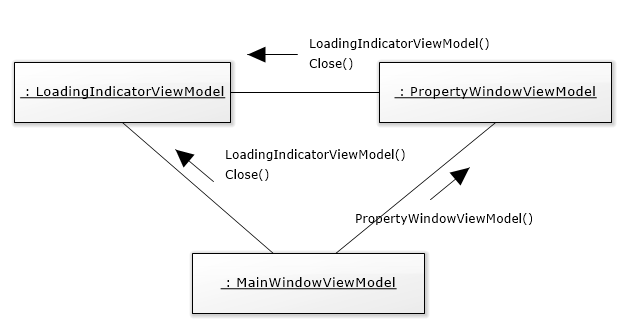


Рис. 19. Диаграмма кооперации

Классы «MainWindowViewModel» и «PropertyWindowViewModel» создают экземпляр класса «LoadingIndicatorViewModel» для отображения процесса выполнения операций. После завершения операции «MainWindowViewModel» и «PropertyWindowViewModel» посылают завершения работы класса «LoadingIndicatorViewModel».

Класс «MainWindowViewModel» вызывает экземпляр класс «PropertyWindowViewModel» для выполнения трансляции модели.

### 5.2.2.4 Построение уточненной диаграммы классов

Данная диаграмма строиться для уточнения отношений между классами.

Уточненная диаграмма представлена на рис. 20.

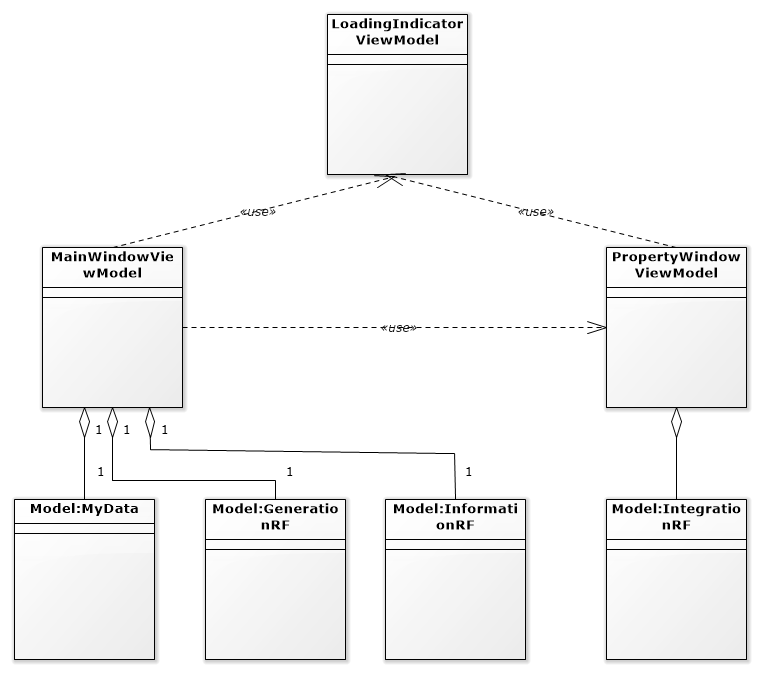


Рис. 20. Уточненная диаграмма классов

Класс «MainWindowViewModel» использует объекты классов «PropertyWindowViewModel» и «LoadingIndicatorViewModel» в своих функциях.

Классы из пакета «Model»: MyData, GenerationRF, InformationRF и IntegrationRF, всегда имеют тип связи – агрегация. Объекты этих классов могут существовать независимо.

### 5.2.2.5 Детальная диаграмма классов

Полная диаграмма классов пакета «ViewModel» представлена на рис. 21, описание полей и методов в табл. 40-45.

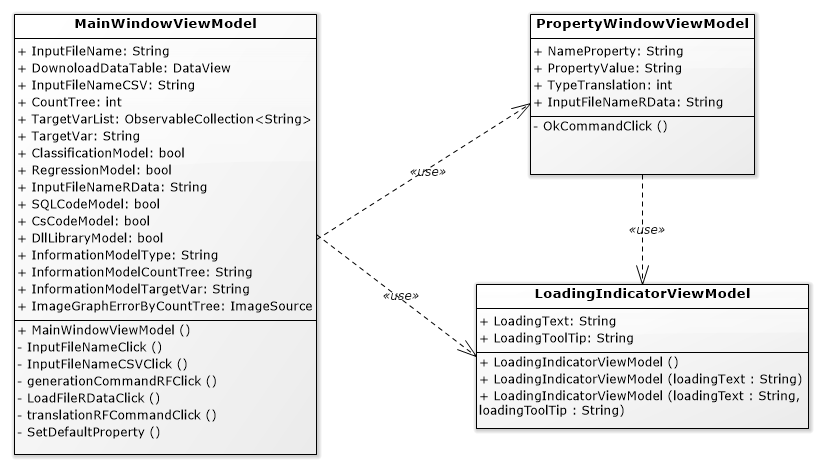


Рис. 21. Детальная диаграмма классов

Таблица 40

Описание полей класса «MainWindowViewModel»

|  |  |
| --- | --- |
| Название поля | Назначение |
| 1 | 2 |
| String InputFileName | Путь к входному файлу для отображения его в таблице |
| DataView DownoloadDataTable | Загруженные данные для отображения в таблице |
| String InputFileNameCSV | Путь к входному файлу .csv для построения модели Random Forest |

Продолжение табл. 40

|  |  |
| --- | --- |
| 1 | 2 |
| int CountTree | Количество деревьев для построения модели Random Forest |
| ObservableCollection<String> TargetVarList | Список целевых переменных из файла |
| String TargetVar | Целевая переменная для построения модели Random Forest |
| bool ClassificationModel | Модель классификации |
| bool RegressionModel | Модель регрессии |
| String InputFileNameRData | Путь к файлу .RData с моделью Random Forest |
| bool SQLCodeModel | Трансляция модели в SQL код |
| bool CsCodeModel | Трансляция модели в C# код |
| bool DllLibraryModel | Трансляция модели в Dll библиотеку |
| String InformationModelType | Информация о модели. Тип |
| String InformationModelCountTree | Информация о модели. Количество деревьев |
| String InformationModelTargetVar | Информация о модели. Целевая переменная |
| ImageSource ImageGraphErrorByCountTree | График зависимости ошибки от количества деревьев принятия решений |

Таблица 41

Описание методов класса «MainWindowViewModel»

|  |  |
| --- | --- |
| Название метода | Назначение |
| 1 | 2 |
| MainWindowViewModel() | Конструктор класса по умолчанию |

Продолжение табл. 41

|  |  |
| --- | --- |
| 1 | 2 |
| InputFileNameClick() | Обработка нажатия кнопки для загрузки файла |
| InputFileNameCSVClick() | Обработка нажатия кнопки для загрузки CSV файла |
| generationCommandRFClick() | Обработка нажатия кнопки для построения модели Random Forest |
| LoadFileRDataClick() | Обработка нажатия кнопки для загрузки RData файла с моделью |
| translationRFCommandClick() | Обработка нажатия кнопки для трансляции модели Random Forest |
| SetDefaultProperty() | Установка значений полей в начальное состояние |

Таблица 42

Описание полей класса «PropertyWindowViewModel»

|  |  |
| --- | --- |
| Название поля | Назначение |
| String NameProperty | Наименования свойства |
| String PropertyValue | Значение свойства |
| int TypeTranslation | Тип трансляции модели |
| String InputFileNameRData | Путь к файлу RData с моделью Random Forest |

Таблица 43

Описание методов класса «PropertyWindowViewModel»

|  |  |
| --- | --- |
| Название метода | Назначение |
| OkCommandClick() | Обработка нажатия кнопки Ок |

Таблица 44

Описание полей класса «LoadingIndicatorViewModel»

|  |  |
| --- | --- |
| Название поля | Назначение |
| String LoadingText | Информационная надпись |
| String LoadingToolTip | Всплывающая подсказка |

Таблица 45

Описание методов класса «LoadingIndicatorViewModel»

|  |  |
| --- | --- |
| Название метода | Назначение |
| LoadingIndicatorViewModel() | Конструктор по умолчанию |
| LoadingIndicatorViewModel(String loadingText) | Конструктор с одним параметром, содержащим информационную надпись |
| LoadingIndicatorViewModel(String loadingText, String loadingToolTip) | Конструктор с двумя параметрами, содержащими информационную надпись и всплывающую подсказку |

### 

### 5.3 Построение диаграммы компонентов

Диаграмма компонетов строиться с целью показать из каких частей программное обеспечение выглядит на физическом уровне.

Диаграмма на каторой показано из какие компоненты использует приложение представлена на рис. 22.

Компонент «RandomForestWork» представляет собой программное обеспечение, разработанное с помощью .NET технологий. Данный компонент служит для загрузки данных и генерации дополнительных параметров; для построения модели Random Forest и трансляции данной модели в SQL-код, C# код и dll библиотеку. Для работы данного компонента необходимы два компонента: «Компилятор языка R» и «Компилятор .NET Framework 4.5».

Компонент «Компилятор языка R» представляет собой библиотеку для выполнения команд компилятора языка R в компоненте «RandomForestWork».

Компонент «Компилятор .NET Framework 4.5» представляет собой компилятор .NET Framework 4.5 компании Microsoft. Данный компонент предназначен для трансляции модели Random Forest в DLL библиотеку. Также данный компонент необходим для работы системы.

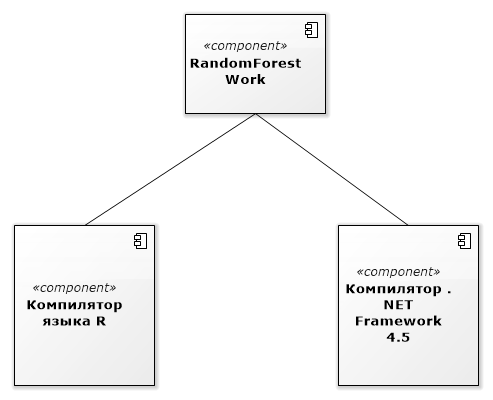


Рис. 22. Диаграмма компонентов

### 5.4 Спецификация модулей

Программное обеспечение для генерации и интеграции модели Random Forest можно разделить на 4 модуля (рис. 23):

* пользовательский интерфейс;
* модуль для работы с языком R;
* модуль для работы с электронными таблицами;
* скрипты.

Рис. 23. Модульная схема приложения

### 5.4.1 Описание модуля «Пользовательский интерфейс»

Данный модуль предназначен для осуществления взаимодействия между пользователем и программным обеспечением. Компоненты, которые входят в состав данного модуля представлены в табл. 46, их описание представлено в табл. 47-49.

Таблица 46

Компоненты модуля «Пользовательский интерфейс»

|  |  |
| --- | --- |
| Название компонента | Описание |
| MainWindowViewModel.cs | Предоставляет пользовательский интерфейс приложения и служит для взаимодействия пользователя с функционалом приложения |
| PropertyWindowViewModel.cs | Предоставляет пользовательский интерфейс для уточнения информации на этапе трансляции модели |
| LoadingIndicatorViewModel.cs | Предоставляет пользователю информацию о выполнении системой определенной операции |

Таблица 47

Спецификация компонента MainWindowViewModel.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| 1 | 2 |
| InputFileName | Обеспечивает взаимосвязь между интерфейсом и системой, в который указывается путь к входному файлу для отображения его в таблице |
| DownoloadDataTable | Обеспечивает взаимосвязь между интерфейсом и системой, в которой хранятся данные из входного файла, для отображения его в таблице |
| InputFileNameCSV | Обеспечивает взаимосвязь между интерфейсом и системой, в который указывается путь к входному файлу в формате csv |
| CountTree | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться количество деревьев принятия решений |
| TargetVarList | Обеспечивает взаимосвязь между интерфейсом и системой, в который храниться список переменных из входного файла csv |
| TargetVar | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться целевая переменная, выбранная из списка TargetVarList |
| ClassificationModel | Обеспечивает взаимосвязь между интерфейсом и системой, которая указывает, что необходимо строить модель классификации. Выбрано может только одно значение: ClassificationModel или RegressionModel |

Продолжение табл. 47

|  |  |
| --- | --- |
| 1 | 2 |
| RegressionModel | Обеспечивает взаимосвязь между интерфейсом и системой, которая указывает, что необходимо строить модель регрессии. Выбрано может только одно значение: ClassificationModel или RegressionModel |
| InputFileNameRData | Обеспечивает взаимосвязь между интерфейсом и системой, в который указывается путь к входному файлу в формате RData. Данный файл должен содержать построенную модель Random Forest |
| SQLCodeModel | Обеспечивает взаимосвязь между интерфейсом и системой, которая указывает, что необходимо модель транслировать в SQL код. Выбрано может только одно значение: SQLCodeModel, CsCodeModel или DllLibraryModel |
| CsCodeModel | Обеспечивает взаимосвязь между интерфейсом и системой, которая указывает, что необходимо модель транслировать в C# код. Выбрано может только одно значение: SQLCodeModel, CsCodeModel или DllLibraryModel |
| DllLibraryModel | Обеспечивает взаимосвязь между интерфейсом и системой, которая указывает, что необходимо модель транслировать в DLL библиотеку. Выбрано может только одно значение: SQLCodeModel, CsCodeModel или DllLibraryModel |

Продолжение табл. 47

|  |  |
| --- | --- |
| 1 | 2 |
| InformationModelType | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться информация о типе модели (классификация или регрессия), которую необходимо транслировать |
| InformationModelCountTree | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться информация о количестве деревьев принятий решений модели, которую необходимо транслировать |
| InformationModelTargetVar | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться информация о целевой переменной модели, которую необходимо транслировать |
| ImageGraphErrorByCountTree | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться график зависимости ошибки прогноза от количества деревьев принятий решений |
| InputFileNameClick() | Обработчик нажатия кнопки для загрузки входного файла. Вызывает окно для выбора файла.  Сохраняет полный путь к данному файлу в InputFileName, InputFileNameCSV.  Заносит данные в DownoloadDataTable и TargetVarList.  Вызывает экземпляр класса LoadingIndicator с указанием о выполнении операции загрузки данных. |

Продолжение табл. 47

|  |  |
| --- | --- |
| 1 | 2 |
| InputFileNameCSVClick() | Обработчик нажатия кнопки для загрузки входного csv файла для построения модели Random Forest. Вызывает окно для выбора файла. Сохраняет полный путь к данному файлу InputFileNameCSV.  Заносит данные в TargetVarList. |
| generationCommandRFClick() | Обработчик нажатия кнопки для построения модели Random Forest. Создаются объекты классов GenerationRF и InformationRF. Выполнятся корректность введенных данных: fileNameCSV, ntree и targetVar.  Вызывает экземпляр класса LoadingIndicator с указанием о выполнении операции генерации модели. Заносит данные в InputFileNameRData, ImageGraphErrorByCountTree, InformationModelType, InformationModelCountTree, InformationModelTargetVar |
| LoadFileRDataClick() | Обработчик нажатия кнопки для загрузки входного файла в формате RData для трансляции модели Random Forest. Вызывает диалоговое окно для выбора входного файла. Создает экземпляр класса InformationRF.  Заносит данные в InputFileNameRData, InformationModelType, InformationModelCountTree, InformationModelTargetVar |

Продолжение табл. 47

|  |  |
| --- | --- |
| 1 | 2 |
| translationRFCommandClick() | Обработчик нажатия кнопки для выполнения трансляции модели.  Сверяет какой тип трансляции был выбран (SQLCodeModel, CsCodeModel или DllLibraryModel).  Создает экземпляр класса PropertyWindowViewModel с указанием дополнительных свойств: если выбрана трансляция в SQL код, то запрашивать имя таблицы; если выбрана трансляция в C# код, то запрашивать имя класса; если выбрана трансляция в DLL библиотеку, то запрашивать имя библиотеки.  Вызывается экземпляр класса PropertyWindowViewModel |
| MainWindowViewModel() | Конструктор по умолчанию, вызывает метод SetDefaultProperty() |
| SetDefaultProperty() | Заносит начальные значения в InputFileName, DownoloadDataTable, InputFileNameCSV, CountTree, TargetVarList, TargetVar, ClassificationModel, InputFileNameRData, SQLCodeModel, InformationModelType, InformationModelCountTree, InformationModelTargetVar, ImageGraphErrorByCountTree |

Таблица 48

Спецификация компонента PropertyWindowViewModel.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| NameProperty | Обеспечивает взаимосвязь между интерфейсом и системой, в которой указывается наименование свойства |
| PropertyValue | Обеспечивает взаимосвязь между интерфейсом и системой, в которой указывается значение свойства |
| TypeTranslation | Хранит значение, которое указывает на тип транслируемой модели |
| InputFileNameRData | Хранит путь к входному файлу в формате RData. Данный файл должен содержать построенную модель Random Forest |
| OkCommandClick() | Обработчик нажатия кнопки для выполнения трансляции модели.  Создает экземпляр класса IntegrationRF.  Проверяет какой тип трансляции был выбран и вызывает соответствующий метод экземпляра класса IntegrationRF.  Вызывает экземпляр класса LoadingIndicator с указанием о выполнении операции трансляции модели. |

Таблица 49

Спецификация компонента LoadingIndicatorViewModel.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| LoadingText | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться текст с указанием наименования выполняемой операции |
| LoadingToolTip | Обеспечивает взаимосвязь между интерфейсом и системой, в которой храниться текст с описанием выполняемой операции |
| LoadingIndicatorViewModel() | Конструктор по умолчанию.  Вносит данные по умолчанию в LoadingText и LoadingToolTip |
| LoadingIndicatorViewModel(loadingText) | Конструктор с одним параметром.  Вносит данные по умолчанию в LoadingToolTip и в LoadingText значение loadingText, передаваемое конструктору |
| LoadingIndicatorViewModel(loadingText, loadingToolTip) | Конструктор с двумя параметрами. Вносит данные в LoadingText из loadingText и в LoadingToolTip из loadingToolTip. |

### 

### 5.4.2 Описание модуля для работы с языком R

Данный модуль предназначен для осуществления работы приложения с компилятором языка R. Компоненты, которые входят в состав данного модуля представлены в табл. 50, их описание представлено в табл. 51-54.

Таблица 50

Компоненты модуля для работы с языком R

|  |  |
| --- | --- |
| Название компонента | Описание |
| Rlanguage.cs | Служит для инициализации и настройки движка языка R |
| GenerationRF.cs | Служит для построения модели Random Forest. Наследуется от Rlanguage |
| IntegrationRF.cs | Служит для трансляции модели Random Forest. Наследуется от Rlanguage |
| InformationRF.cs | Служит для получения информации модели Random Forest. Наследуется от Rlanguage |

Таблица 51

Спецификация компонента Rlanguage.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| engine | Переменная служит для вызова функций языка R |
| Rversion | Переменная для хранения версии языка R, которая используется для работы |
| Rpath | Переменная для хранения полного пути к исполняемому файлу |
| RLanguage() | Конструктор по умолчанию.  Запускает метод initialize() |
| initialize() | Метод для инициализации движка R.  Вызывает метод SetupPath(), если инициализация была не произведена.  Инициализирует переменную engine |
| SetupPath() | Метод для настройки переменной среды PATH Windows к библиотеке R.dll |

Таблица 52

Спецификация компонента GenerationRF.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| 1 | 2 |
| fileNameCSV | Переменная для хранения пути к входному файлу в формате csv |
| ntree | Переменная для хранения количества деревьев принятия решений |
| targetVar | Переменная для хранения целевой переменной |
| type | Переменная для хранения типа модели (классификация, регрессия) |
| pathToSaveData | Переменная для хранения полного пути, сохранения системой получившейся модели |
| SaveFileName | Переменная для хранения полного пути, куда пользователь сохранил модель |
| generation() | Метод для построения модели Random Forest.  Для работы с языком R используется переменная engine из класса Rlanguage, которая наследуется.  Строит модель по параметрам указанным в fileNameCSV, ntree, targetVar  Сохраняет построенную модель по пути, который указан в pathToSaveData.  После построения модели вызывается метод saveRF() |
| generation(fileName, target, nTree, tp) | Метод для построения модели Random Forest.  Вносит данные в fileNameCSV, ntree, targetVar, которые получает в качестве параметров и вызывает метод generation() |

Продолжение табл. 52

|  |  |
| --- | --- |
| 1 | 2 |
| saveRF() | Метод, который позволяет сохранить полученную модель на локальном диске.  Вызывает окно, которое позволяет пользователю выбрать место и указать имя для сохранения модели. |
| GenerationRF() | Конструктор по умолчанию.  Вызывает базовый конструктор из класса Rlanguage.  Вносит начальное значение в pathToSaveData |

Таблица 53

Спецификация компонента IntegrationRF.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| 1 | 2 |
| pathToScript | Переменная для хранения пути к скриптам на языке R, которые хранятся на локальном диске |
| pathToSaveData | Переменная для хранения пути, куда система будет сохранять результаты транслирования модели Random Forest |
| IntegrationRF() | Конструктор по умолчанию.  Вызывает базовый конструктор из класса Rlanguage.  Вносит начальное значение в pathToScript и pathToSaveData |

Продолжение табл. 53

|  |  |
| --- | --- |
| 1 | 2 |
| translateToSQL(fileNameRData, tableName) | Метод для трансляции модели в SQL код.  В качестве параметров принимает путь к файлу с моделью Random Forest (fileNameRData) и название таблицы (tableName), в которую будет транслирована модель. После трансляции модели вызывает метод saveSQL |
| translateToCS(fileNameRData, className, forDLL = false) | Метод для трансляции модели в C# код.  В качестве параметров принимает путь к файлу с моделью Random Forest (fileNameRData), название класса (className), в которую будет транслирована модель и флаг (forDLL) о том будет ли полученный класс использоваться для построения DLL библиотеке. После трансляции модели вызывает метод saveCsharp(), если флаг forDLL равен false |
| translateToDLL(fileNameRData, className) | Метод для трансляции модели в C# код.  В качестве параметров принимает путь к файлу с моделью Random Forest (fileNameRData) и название класса (className), в которую будет транслирована модель. Вызывается метод translateToCS() с флагом forDLL равным true. Для сборки dll библиотеки вызывает csc консоль .NET Framework 4.5. После трансляции модели вызывает метод saveDLL() |

Продолжение табл. 53

|  |  |
| --- | --- |
| 1 | 2 |
| saveSQL() | Метод для сохранения получившегося SQL кода.  Вызывает окно, которое позволяет пользователю выбрать место и указать имя для сохранения. |
| saveCsharp(className) | Метод для сохранения получившегося C# кода. В качестве параметра принимает имя класса.  Вызывает окно, которое позволяет пользователю выбрать место для сохранения. |
| saveDLL(className) | Метод для сохранения получившейся dll библиотеки. В качестве параметра принимает имя класса. Вызывает окно, которое позволяет пользователю выбрать место для сохранения. |

Таблица 54

Спецификация компонента IntegrationRF.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| 1 | 2 |
| InformationRF() | Конструктор по умолчанию.  Вызывает базовый конструктор из класса Rlanguage. |
| GetType(fileNameRData) | Метод для получения информации по типу выбранной модели. В качестве параметра принимает путь к файлу с моделью. |

Продолжение табл. 54

|  |  |
| --- | --- |
| 1 | 2 |
| GetCountTree(fileNameRData) | Метод для получения информации по количеству деревьев решений выбранной модели. В качестве параметра принимает путь к файлу с моделью. |
| GetTargetVar(fileNameRData) | Метод для получения информации по целевой переменной выбранной модели. В качестве параметра принимает путь к файлу с моделью. |
| DiagrammCounttreeByError (fileNameRData) | Метод для построения графика зависимости количества деревьев от ошибки выбранной модели. В качестве параметра принимает путь к файлу с моделью. |

### 

### 5.4.3 Описание модуля для работы с электронными таблицами

Данный модуль предназначен для чтения и конвертации электронных таблиц. Компоненты, которые входят в состав данного модуля представлены в табл. 55, их описание представлено в табл. 56.

Таблица 55

Компоненты модуля для работы с электронными таблицами

|  |  |
| --- | --- |
| Название компонента | Описание |
| MyData.cs | Служит для работы с электронными таблицами |

Таблица 56

Спецификация компонента MyData.cs

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| inputFileName | Переменная для хранения полного имени к входному файлу |
| extension | Переменная для хранения расширения входного файла |
| InputFileName() | Метод для указания полного пути к входному файлу и получения его расширения |
| loadData() | Метод для считывания данных из файла и представления их в виде таблицы.  Производиться сравнения расширения и если оно совпадает со значением .xls или .xlsx, то вызывается метод convertXlsToCsv() |
| convertXlsToCsv() | Метод для конвертации файла в формате .xls или .xlsx в csv файл с выводом советующего сообщения |
| GetTargetVarList() | Метод для получения списка заголовков таблицы входного файла |

### 

### 5.4.4 Описание модуля «Скрипты»

Данный модуль содержит методы, написанные на языке R, для выполнения трансляции модели. Компоненты, которые входят в состав данного модуля представлены в табл. 57, их описание представлено в табл. 58, 59.

Таблица 57

Компоненты модуля для работы с электронными таблицами

|  |  |
| --- | --- |
| Название компонента | Описание |
| RFToCS.RData | Служит для трансляции модели в C# код |
| RFToSQL.RData | Служит для трансляции модели в SQL код |

Таблица 58

Спецификация компонента RFToCS.RData

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| RFToCS (model, className, pathToSave) | Метод для трансляции модели в C# код. В качестве параметров принимает имя модели Random Forest (model), названия класса (className) и путь (pathToSave), куда необходимо сохранить полученный C# код |

Таблица 59

Спецификация компонента RFToSQL.RData

|  |  |
| --- | --- |
| Функция или свойство | Описание |
| RFToSQL (model, tableName, fileName) | Метод для трансляции модели в SQL код. В качестве параметров принимает имя модели Random Forest (model), названия таблицы (className) и имя файла (pathToSave), который будет полученный в результате трансляции модели |

### 

### 5.5 Построение диаграммы размещения

Данная разработка представляет собой одиночное приложение для работы на локальном персональном компьютере. Из этого следует, что диаграмма размещения не требуется в данной разработке.

## 6. Проектирование интерфейса пользователя

### 6.1 Построение графа диалогов

Пользовательский интерфейс – совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером [4].

Основой задачей проектирования пользовательского интерфейса является построение графа диалога. Граф диалога – ориентированный граф, где каждая вершина представляет собой конкретную картинку на экране или состояние диалога, которое характеризуется набором доступных действий для пользователю. Дуги показывают возможные изменения состояний, которые возникают при выполнении пользователем указанных действий. Каждый маршрут соответствует возможному варианту диалога.

На рис. 24 представлен граф диалогов разрабатываемого программного обеспечения. В табл. 60 находиться описание вершин графа диалога.



Рис. 24. Граф диалога

Таблица 60

Описание вершин графа диалога

|  |  |
| --- | --- |
| Вершина графа | Описание |
| 1 | 2 |
| Окно открытия данных | Окно, в котором отображается содержимое файла в таблице |

Продолжение табл. 60

|  |  |
| --- | --- |
| 1 | 2 |
| Окно генерации Random Forest | Окно, в котором пользователь производит настройку построения Random Forest и просматривает график построенной модели |
| Окно интеграции Random Forest | Окно, в котором пользователь выбирает тип трансляции модели и производит ее трансляцию |
| Окно загрузки данных | Окно, через которое выбирается файл для загрузки |
| Окно сохранения файла | Окно, через которое выбирается место для сохранения файла на локальном диске |
| Окно сохранения папки | Окно, через которое выбирается место для сохранения папки на локальном диске |
| Окно указания имени класса | Окно, в котором указывается имя класса, для трансляции модели в C# код и DLL библиотеку |
| Окно указания имени таблицы | Окно, в котором указывается имя таблицы, для трансляции модели в SQL код |

### 

### 6.2 Разработка форм ввода-вывода информации

Для данного программного обеспечения был выбран интерфейс со свободной навигацией. Данный тип интерфейса поддерживает концепцию интерактивного взаимодействия с ПО, визуальную обратную связь с пользователем и возможность прямого манипулирования объектом (кнопки, индикаторы, строки состояния) [4].

Интерфейсы со свободной навигацией реализуют, используя событийное программирование и объектно-ориентированные библиотеки, что предполагает применение визуальных сред разработки программного обеспечения [27]. Данный тип интерфейса при взаимодействии с пользователем предлагает на выбор только те операции, которые имеют смысл в конкретной ситуации.

Разработка интерфейса производилась с помощью технологии WPF, данная технология позволяет создавать интерфейс который будет автоматически изменять размеры элементов управления, в зависимости от того какой размер окна выберет пользователь.

После запуска программы открывается окно приложения с возможностью выбора одной из трех функциональных вкладок, при старте открывается окно «Открыть данные» (рис. 25). Здесь пользователь может загрузить и просмотреть данные.

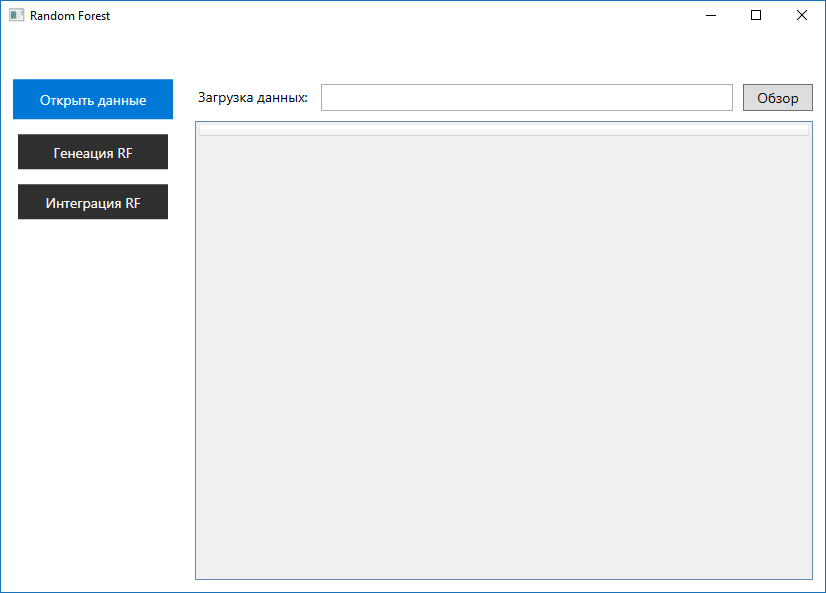


Рис. 25. Окно приложения после его запуска

На рис. 26 представлена вкладка «Генерация RF». На данной вкладке пользователь может выбрать входной файл с данными и построить модели Random Forest, выполнив необходимые настройки. Также пользователь может увидеть график зависимости ошибки от количества деревьев принятия решений.

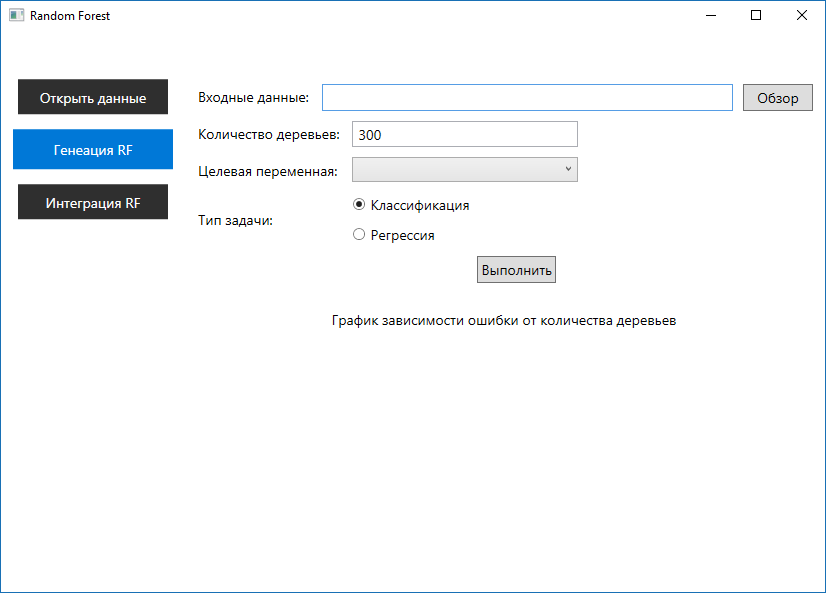


Рис. 26. Вкладка «Генерация RF»

Вкладка «Интеграция RF» (рис. 27) позволяет загрузить готовую модель Random Forest с последующей ее трансляцией в SQL код, C# код или DLL библиотеку.

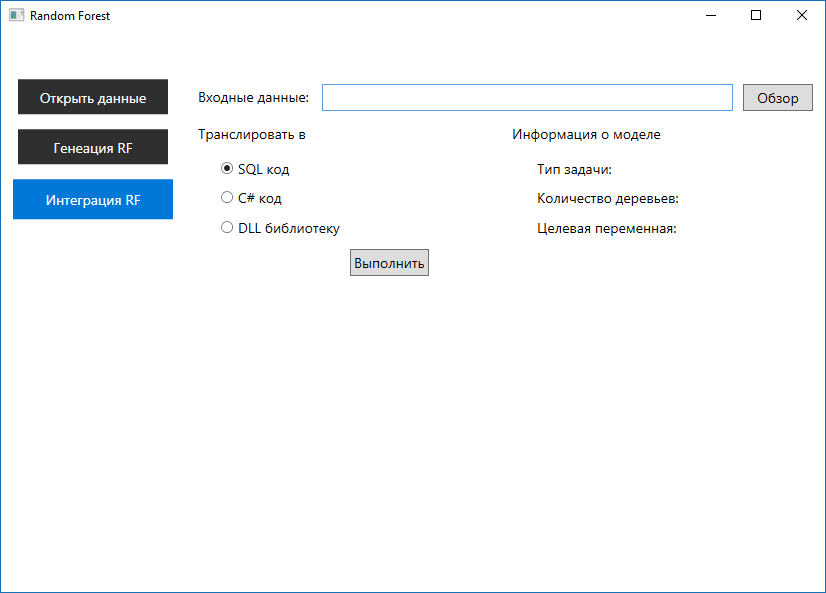


Рис. 27. Вкладка «Интеграция RF»

В случаи возникновения ошибки, программа выдаст сообщения об ошибки. Пример сообщения об ошибке представлен на рис 28.

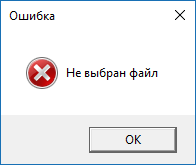


Рис. 28. Сообщение об ошибке

Для сохранения файла используется стандартное окно, которое предоставляет операционная система Windows (рис. 29).

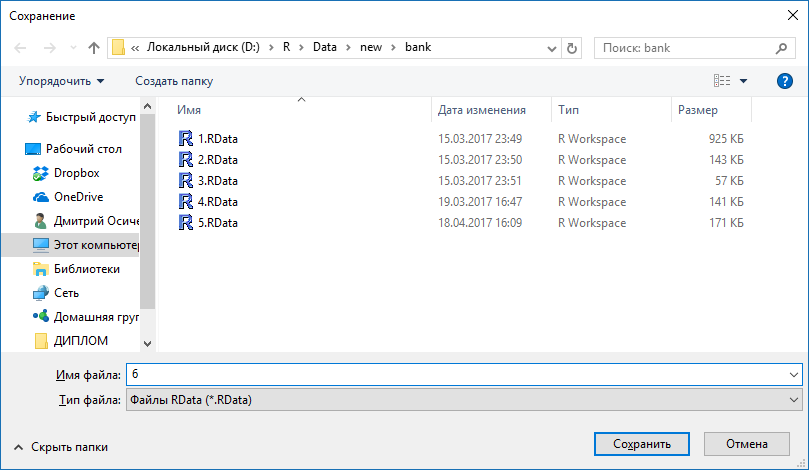


Рис. 29. Окно сохранения файла

Подробное описание окон приложения и последовательность действий по работе с ними представлено в руководстве пользователя (прилож. 3)

## 7. Выбор стратегии тестирования, разработка тестов, программа и методика испытаний

### 7.1. Объект испытаний

Испытанию подвергается программное обеспечение для генерации и интеграции модели Random Forest.

### 7.2. Цель испытаний

Цель проводимых испытаний:

* поиск максимального количества ошибок;
* проверка правильности работы отдельных частей программы и всего программного продукта в целом;
* проверка соответствия, разработанного ПО требованиям, обозначенным в техническом задании.

### 7.3 Требования к программному обеспечению

Разработанная программа должна удовлетворять функциональным требованиям, перечисленным в техническом задании (прил. 1). Функционирование программы должно быть стабильным, исключительные ситуации должны обрабатываться самим приложением. Организация диалога с пользователем должна предусматривать проверку введенных данных.

Также в соответствии с техническим заданием на выпускную квалификационную работу показатели надежности разрабатываемой системы должны отвечать всем требованиям ГОСТ 28806-90.

### 7.4 Требование к документации

Программная документация, которая должна быть доступна при проведении испытаний:

* техническое задание (прил. 1);
* текст программы (прил. 2);
* руководство пользователя (прил. 3).

### 7.5 Состав и порядок испытаний

Программа запускается в соответствии с условиями эксплуатации. Состав средств IBM-совместимой рабочей станции, на которой выполняется тестирование:

1. Процессор: Intel Core i5 с тактовой частотой 2,5 ГГц;
2. ОЗУ: 12 Гб;
3. Жесткий диск: 1 Тб;
4. ЖК дисплей 17.3”.

### 7.5.1 Состав испытаний

Испытания проводились в следующем составе:

* тестирование пользовательского интерфейса;
* тестирование минимальных конфигураций технических средств;
* тестирование надёжности.

### 7.5.2 Порядок проведения испытаний

Испытания проводились в следующем порядке:

1. запуск приложения;
2. тестирование удобства эксплуатации;
3. тестирование функциональных возможностей;
4. тестирование граничных значений;
5. тестирование надежности.

### 7.6 Методы испытаний

Для тестирования на начальном этапе применяется метод ручного контроля. Ручной контроль обеспечивает обнаружение 30-70% ошибок. В качестве исходных данных для такого контроля выступают техническое задание, спецификации, структурные и функциональные схемы, схемы отдельных компонентов, а для более поздних этапов – алгоритмы и тексты программ, а также тестовые наборы [6]. В качестве ручного тестирования был выбран метод проверки за столом. Данный метод основан на проверке исходного текста одним человеком, который читает текст программы. Он пропускает через программу тестовые данные. Данный метод тестирования должен проводить не автор программы.

Для тестирования отдельных модулей ПО был выбран один из методов функционального тестирования – метод граничных значений. Исходными данными для этого метода являются: программные модули, спецификации на компоненты, правильные и неправильные данные для модулей.

### 7.7 Результаты тестирования

### 7.7.1 Модульное тестирование

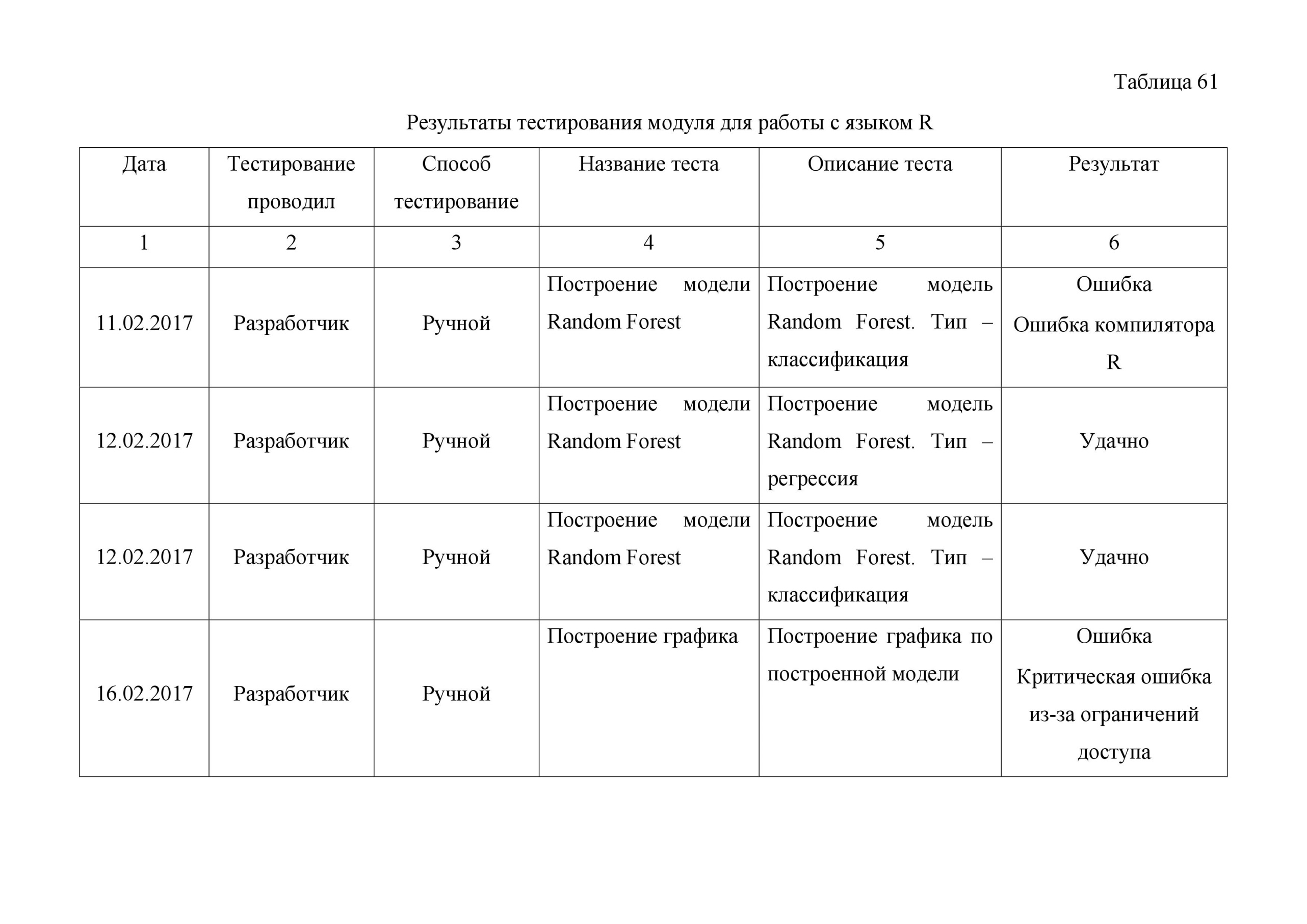
Данный способ тестирования подразумевает проведение испытаний каждого модуля системы в отдельности. Модулем в данном случае называют компонент или набор компонентов, объединённых общим функционалом.

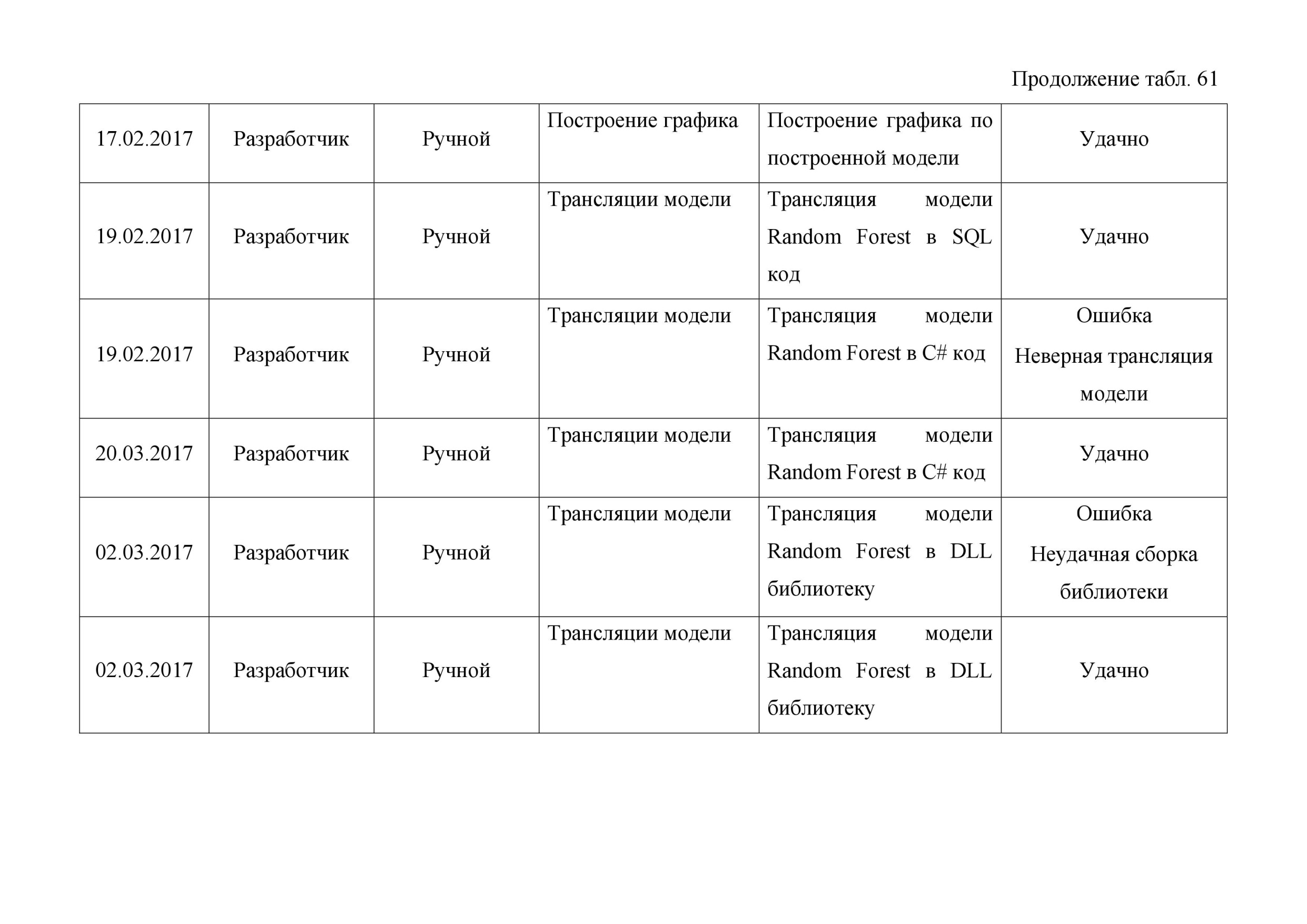
Набор тестовых данных представлен в прил. 4.

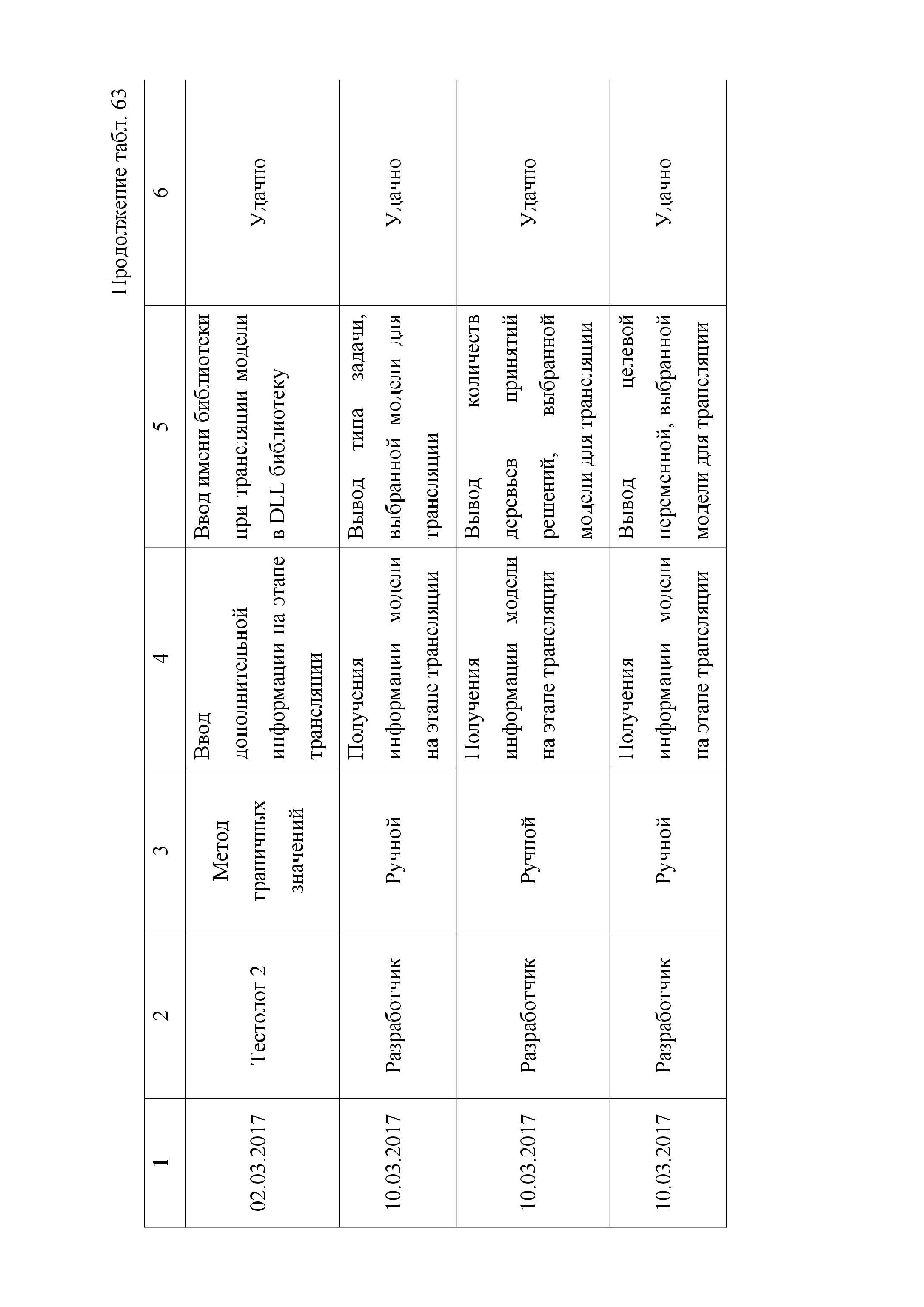
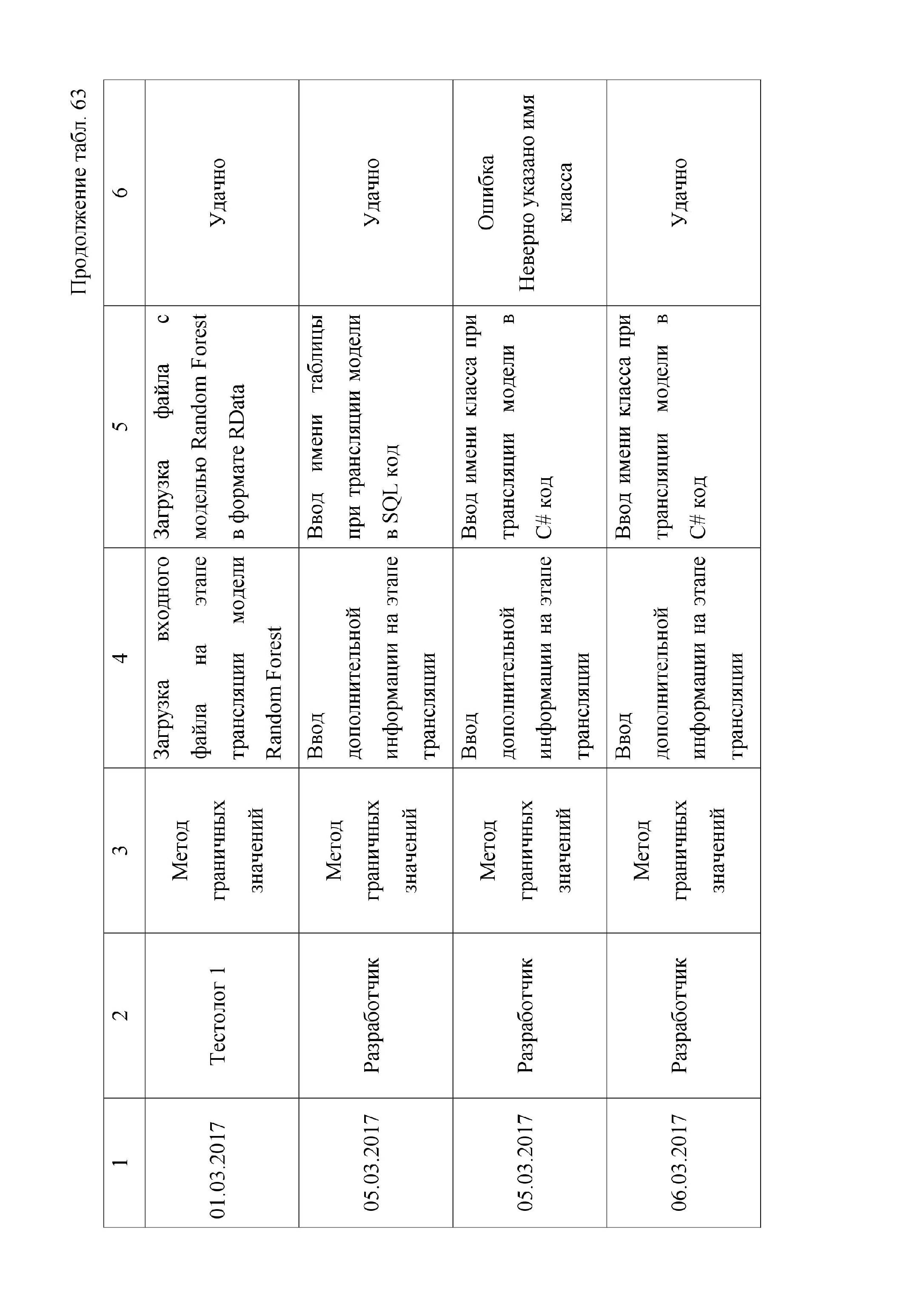
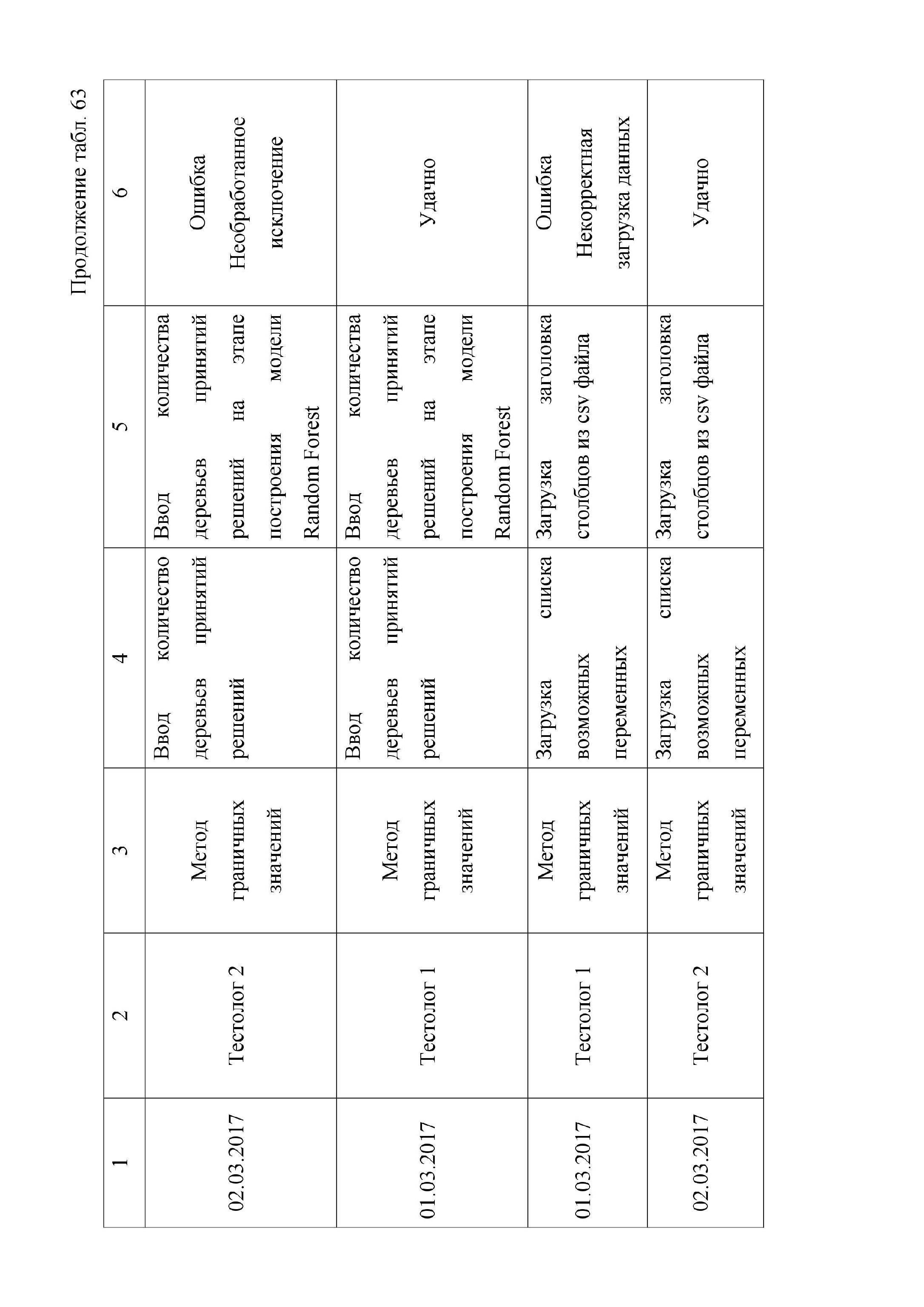
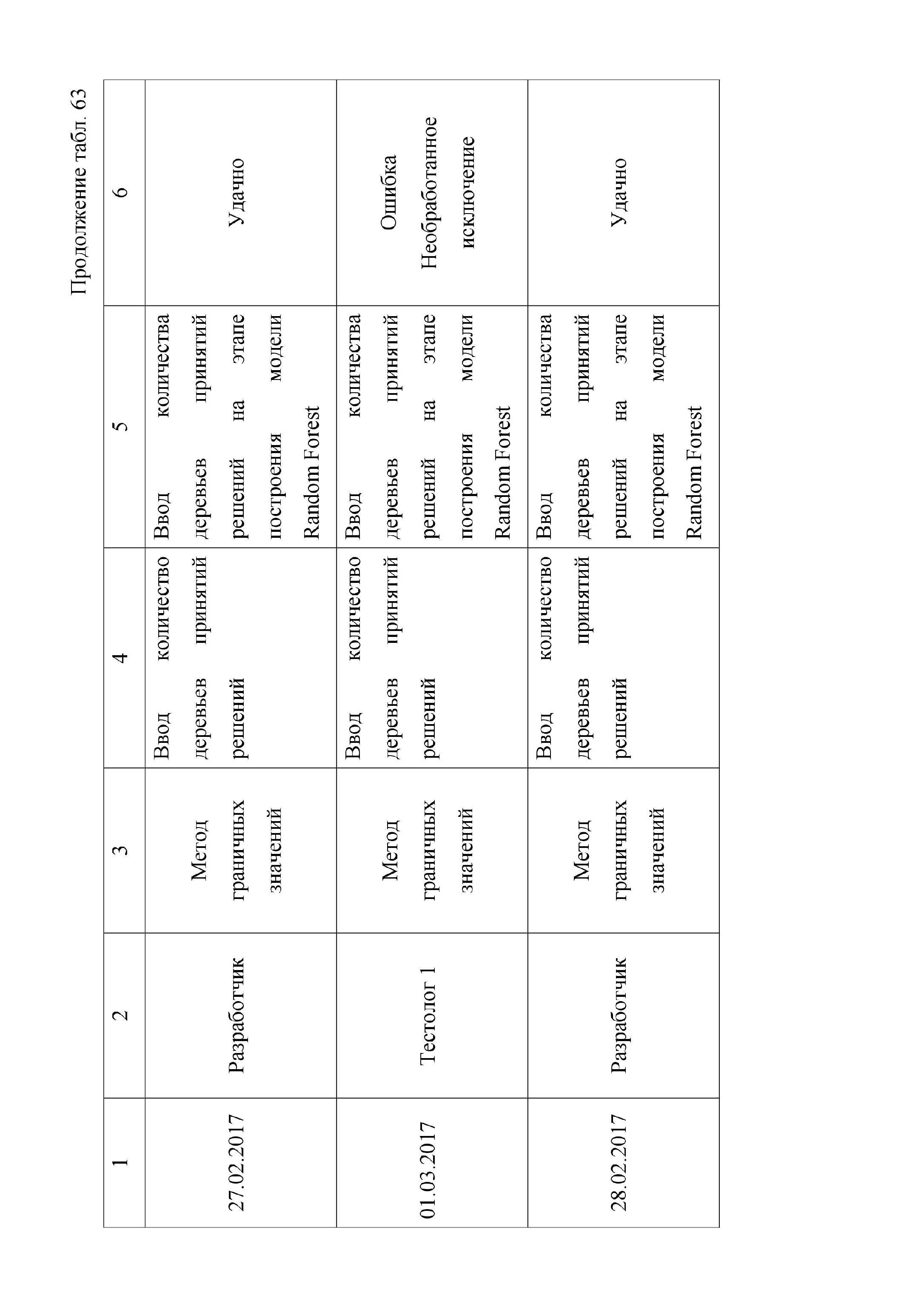
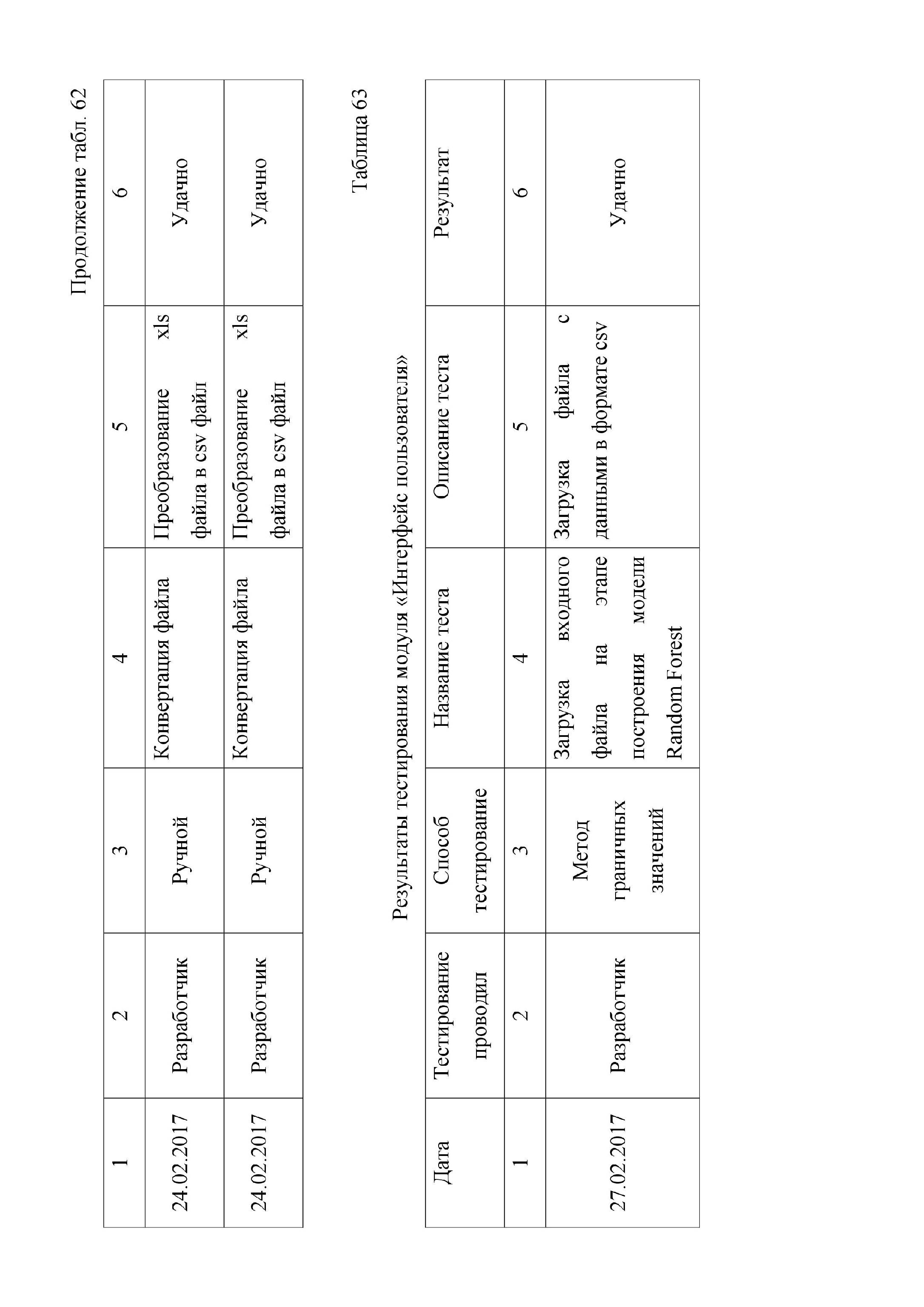
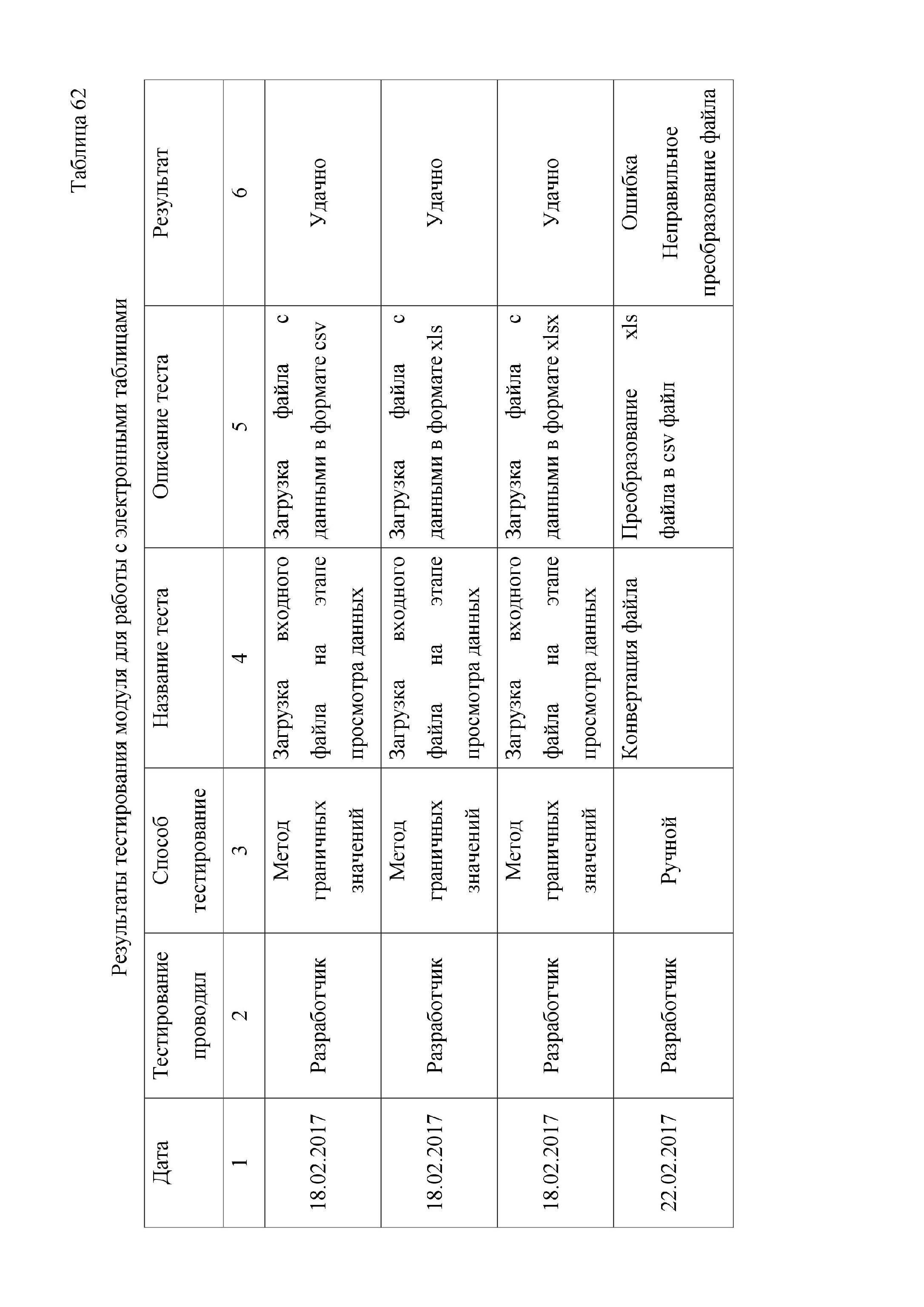
Тестированию подверглись следующие модули ПО:

* модуль для работы с языком R;
* модуль для работы с электронными таблицами;
* пользовательский интерфейс.

Результаты модульного тестирования представлены в табл. 61-63.





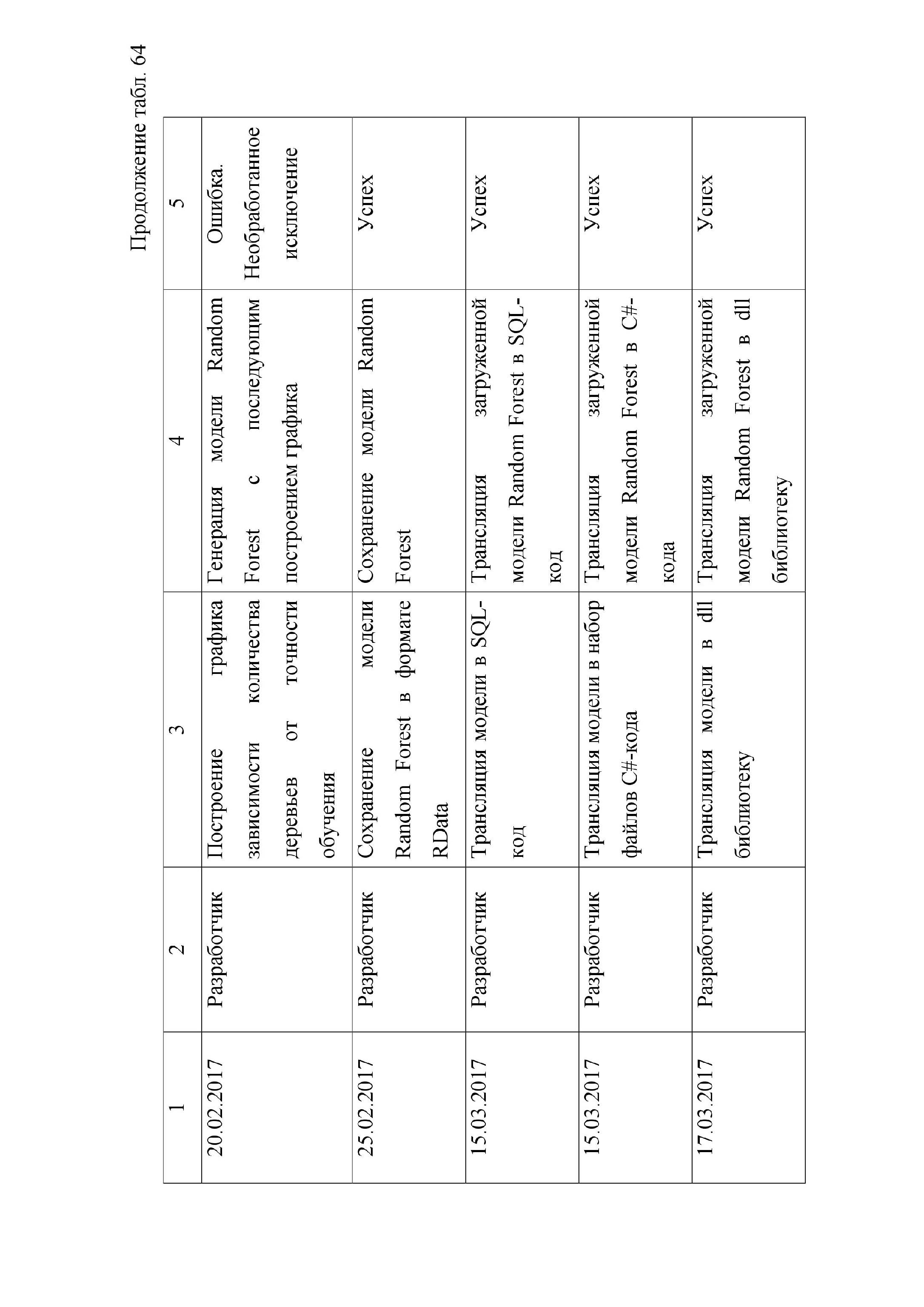
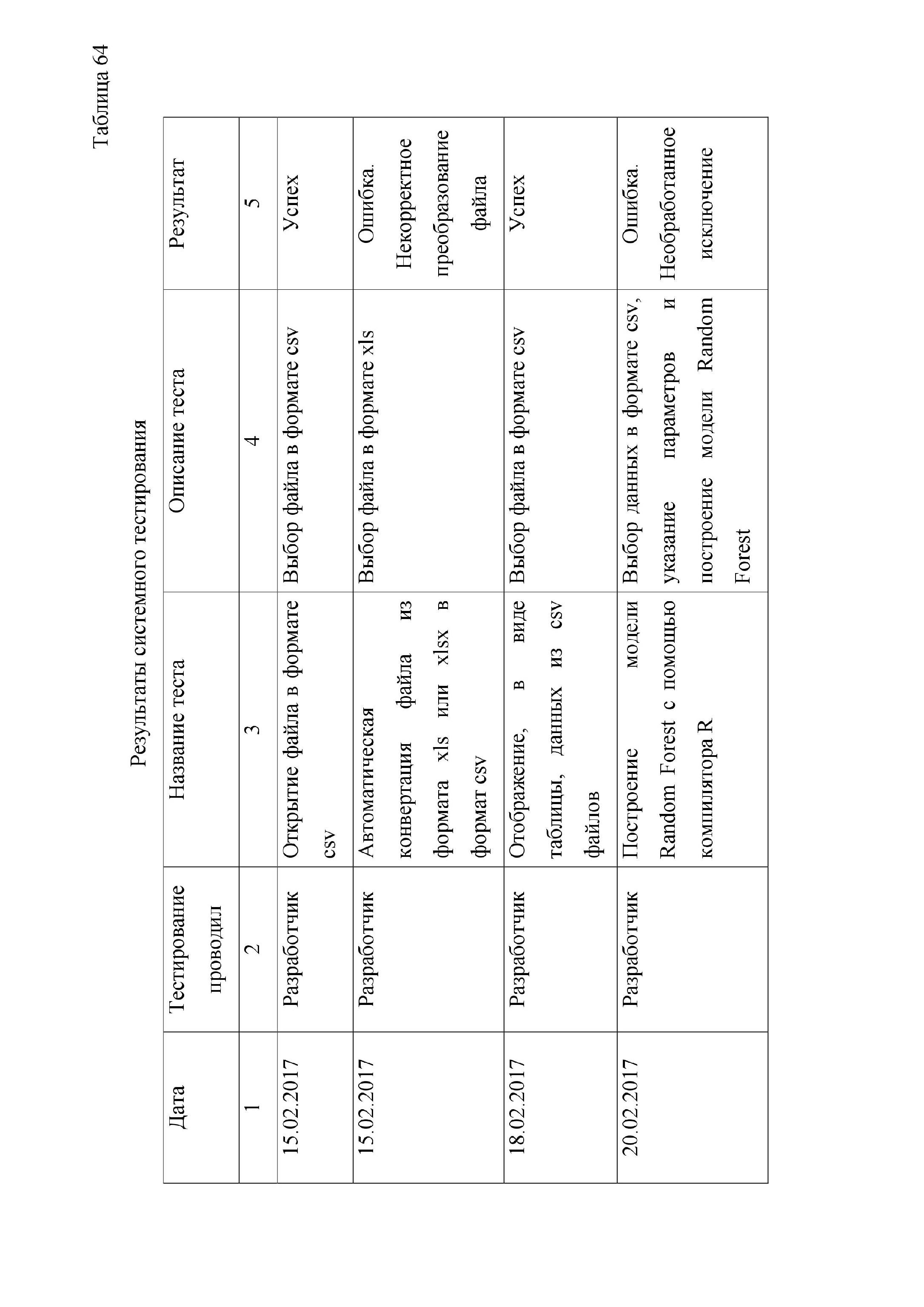


### 7.7.2 Системное тестирование

При системном тестировании производиться проверка всего приложения, которое было полностью разработано и которое готово к релизу. С помощью данного способа тестирования проверяется соответствие функциональных требований приложения и требований представленных в техническом задании (прил. 1).

Так как, основная задача разработанного приложения является генерация и интеграция модели Random Forest по входному набору данных, то для тестирования был выбран файл с большим количеством входных данных. Пример данного файла и его описание представлено в прил. 4.

Результаты системного тестирования представлены в табл. 64.



# Заключение

В ходе выполнения выпускной квалификационной работы было спроектировано и разработано программное обеспечение для автоматизации и облегчения работы по процессу генерации и интеграции модели машинного обучения Random forest. Для реализации поставленной задачи были выполнены все этапы разработки программного обеспечения согласно выбранной спиральной модели жизненного цикла.

На начальном этапе было разработано техническое задание, которое содержит требование к программному обеспечению. Составление данного документа позволило четко определить требуемый функционал разработки.

На этапе проектирования были разработаны спецификации программного обеспечения, пакеты и классы, которые входят в пакеты. При проектировании приложения использовался паттерн MVVM.

На этапе реализации было написано ПО с использованием среды разработки Visual Studio 2013 на языке C#. При разработке использовалась технология WPF с поддержкой XAML кода, для реализации пользовательского интерфейса. Также на языке R были разработаны скрипты для выполнения трансляции модели Random forest. Также у разработанного скрипта имеется свидетельство о государственной регистрации программы для ЭВМ.

На завершающем этапе проводилось тестирования ПО. В ходе тестирования были выявлены и исправлены ошибки при работе приложения. В результате тестирование показало, что данное ПО соответствует всем требованиям, представленным в техническом задании.

Также в ходе работы в рамках выпускной квалификационной работы были получены и усвоены следующие компетенции:

1. Владение навыками использования различных технологий разработки программного обеспечения (ПК-3);
2. Владение концепциями и атрибутами качества программного обеспечения (надежности, безопасности, удобства использования), в том числе роли людей, процессов, методов, инструментов и технологий обеспечения качества (ПК-4);
3. Готовность к использованию методов и инструментальных средств исследования объектов профессиональной деятельности (ПК-13);
4. Готовность обосновать принимаемые проектные решения, осуществлять постановку и выполнение экспериментов по проверке их корректности и эффективности (ПК-14);
5. Способность формализовать предметную область программного проекта и разработать спецификации для компонентов программного продукта (ПК-16);

Использование созданного программного обеспечения позволит без изучения специальных программных решений строить модель Random forest, что облегчает и автоматизирует работу пользователя, которому необходимо использовать данную модель в СУБД Oracle или .NET проектах.

# Список литературы

1. Вершинина Е.В. Обзор моделей жизненного цикла разработки программного обеспечения: учебник. Е.В. Вершинина, М.С. Гонченко. – НГУ им. Н.И. Лобачевского, 2010. – 38 с.
2. Гвоздева Т.В. Проектирование информационных систем: учебное пособие. Т.В. Гвоздева, Б.А. Баллод. – Ростов на Дону: Феникс, 2009. – 508 c.
3. Емельянова Н.З. Основы построения автоматизированных информационных систем: учебное пособие. Н. З. Емельянова, Т. Л. Партыка, И. И. Попов. – М.: ФОРУМ: ИНФРА-М, 2007. – 416 с.
4. Иванова Г.С. Технология программирования: Учебник для вузов. Г. С. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. – 320 с.
5. Избачков Ю.С. Информационные системы: учебник. Ю.С. Избачков, В.Н. Петров – СПб: Питер, 2006. – 656 с.
6. Котляров В.П. Основы тестирования программного обеспечения: учебное пособие. В. В. Котляров – СПб.: Питер, 2006. – 170 с.
7. Орлов С.А. Технология разработки программного обеспечения. Разработка сложных программных систем: учебник. С. А. Орлов. – СПб.: Питер, 2002. – 464 с.
8. Чистяков С. П. Случайные леса: обзор / С. П. Чистяков // Труды Карельского научного центра РАН. – 2013. - №11. – С. 117-136
9. Практические аспекты машинного обучения. [Электронный ресурс] / Режим доступа: http://www.osp.ru/os/2016/01/13048648/
10. Деревья решений и алгоритмы их построения. [Электронный ресурс] / DataReview – Режим доступа: http://datareview.info/article/derevya-resheniy-i-algoritmyi-ih-postroeniya/
11. Oracle R Enterprise. [Электронный ресурс] / Oracle – Режим доступа: http://www.oracle.com/technetwork/database/database-technologies/r/r-enterprise/overview/index.html
12. Oracle R Enterprise. [Электронный ресурс] / ФОРС – Режим доступа: http://www.fors.ru/business-solutions/analytical-systems-and-data-warehouse/tools-advanced-analytics/oracle-r-enterprise/
13. Oracle pricing. [Электронный ресурс] / Oracle – Режим доступа: http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf
14. IBM SPSS Modeler. [Электронный ресурс] / IBM – Режим доступа: http://www-03.ibm.com/software/products/ru/spss-modeler
15. IBM SPSS Modeler Pricing. [Электронный ресурс] / CROWD – Режим доступа: https://www.g2crowd.com/products/ibm-spss-modeler/pricing
16. Жизненный цикл информационной системы. [Электронный ресурс] / Жизненный цикл – Режим доступа: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema2
17. Rapid Application Development – Быстрая разработка приложений. [Электронный ресурс] / INFORMICUS – Режим доступа: http://www.informicus.ru/ErrorPage.aspx?aspxerrorpath=/Default.aspx
18. Язык C#. [Электронный ресурс] / Википедия – свободная энциклопедия – Режим доступа: https://ru.wikipedia.org/wiki/C\_Sharp
19. WPF. [Электронный ресурс] / Microsoft – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ms742119(v=vs.110).aspx
20. Smith, J. WPF Apps With The Model-View-ViewModel Design Pattern. [Электронный ресурс] / Josh Smith – Режим доступа: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
21. Microsoft Visual Studio. [Электронный ресурс] / Википедия – свободная энциклопедия – Режим доступа: https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio#Visual\_Studio\_2013
22. R. [Электронный ресурс] / MachineLearning – Режим доступа: http://www.machinelearning.ru/wiki/index.php?title=R
23. RStudio. [Электронный ресурс] / RStudio – Режим доступа: https://www.rstudio.com/products/RStudio/
24. Software Ideas Modeler. [Электронный ресурс] / f1cd – Режим доступа: http://www.f1cd.ru/soft/base/software\_ideas\_modeler/
25. Случайный лес (Random Forest). [Электронный ресурс] / Анализ малых данных – Режим доступа: https://alexanderdyakonov.wordpress.com/2016/11/14/%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B9-%D0%BB%D0%B5%D1%81-random-forest/
26. Введение в UML 2.0. [Электронный ресурс] / Интуит. Национальный открытый университет – Режим доступа: http://www.intuit.ru/studies/courses/1041/218/lecture/27263
27. Разработка пользовательских интерфейсов. [Электронный ресурс] / Технология программирования – Режим доступа: http://www.tehprog.ru/index.php\_page=lecture42.html
28. Машина опорных векторов. [Электронный ресурс] / Распознование – Режим доступа: http://www.machinelearning.ru/wiki/index.php?title= %D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0\_%D0%BE%D0%BF%D0%BE%D1%80%D0%BD%D1%8B%D1%85\_%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BE%D0%B2

# Приложение 1

МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра математического и программного обеспечения ЭВМ

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

д. т.н., профессор \_\_\_\_\_\_\_\_\_\_ Ершов Е.В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_ г.

Разработка программного обеспечения построения и интеграции модели Random Forest

Техническое задание на выпускную квалификационную работу

Листов 8

Руководитель:

Ершов Евгений Валентинович

Исполнитель: студент гр. 1ПИб-01-41оп

Осичев Дмитрий Павлович

Череповец, 2017 г.

# ВВЕДЕНИЕ

Целью работы является проектирование и разработка программного обеспечения с использованием языков программирования .NET C# и R. Данный продукт предназначен для генерации и интеграции модели машинного обучения Random Forest. Под генерацией понимается: генерация дополнительных параметров, построение модели Random Forest по заданной выборке и генерация SQL и C# кода по построенной модели. Под интеграцией понимается возможность дальнейшего использование SQL кода в СУБД Oracle и C# кода в .NET технологиях.

1. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

Основанием для разработки данного программного обеспечения служит задание на выпускную квалификационную работу от руководителя по производственной и преддипломной практике в «Череповецком государственном университете» на кафедре «МПО ЭВМ»

# НАЗНАЧЕНИЕ РАЗРАБОТКИ

Разработанный программный продукт предназначен для генерации и интеграции модели машинного обучения Random Forest с использованием языка программирования R.

# ТРЕБОВАНИЯ К ПРОГРАММЕ

## 3.1 Требования к функциональным характеристикам

### 3.1.1 Требование к составу выполняемых функций

Программное обеспечение должно включать в себя следующий набор функций:

* открытие файла в формате csv;
* автоматическая конвертация файла из формата xls или xlsx в формат csv;
* отображение, в виде таблицы, данных из csv файлов;
* построение модели Random Forest с помощью компилятора R;
* выбор количества деревьев для построения модели Random Forest;
* выбор целевой переменной для построения модели Random Forest;
* построение графика зависимости количества деревьев от точности обучения;
* сохранение полученной модели Random Forest в формате RData;
* загрузка модели Random Forest в формате RData для трансляции в SQL-код, C#-код или dll библиотеку;
* трансляция модели в SQL-код;
* трансляция модели в набор файлов C#-кода;
* трансляция модели в dll библиотеку;
* назначение имени таблицы при трансляции модели Random Forest в SQL-код;
* назначение имени класса при трансляции модели Random Forest в C#-код;
* назначение имени класса библиотеки при трансляции модели Random Forest в dll библиотеку.

### 3.1.2 Требование к организации входных данных

Входные данные должны быть представлены в виде отдельного файла.

Входные данные программы для просмотра содержимого должны быть представлены в xls, xlsx или csv формате.

Входные данные программы для генерации модели Random Forest должны быть представлены в csv формате.

Входные данные программы для трансляции модели должны быть представлены в формате RData.

Файлы указанных форматов должны размещаться на локальных или съемных носителях, отформатированных согласно требованиям операционной системы.

Файлы с другим форматом не должны открываться.

### 3.1.3 Требования к организации выходных данных

Сгенерированная модель Random Forest сохраняется в формате RData. Транслированная модель Random Forest в SQL-код, сохраняется в формате sql. Транслированная модель Random Forest в dll библиотеку, сохраняется в формате dll. Транслированная модель Random Forest в C#-код, сохраняется в виде набора файлов формата cs, в отдельной папке.

Файлы данных форматов могут быть сохранены на локальном носителе.

## 3.2 Требования к надежности

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:

* организацией бесперебойного питания технических средств;
* использованием лицензионного программного обеспечения;
* регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
* регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов.

## 3.3 Условия эксплуатации

Условия эксплуатации программного обеспечения должны соответствовать условиям эксплуатации вычислительных средств, на которых предполагается использовать проектируемое программное обеспечение.

### 3.4 Требования к составу и параметрам технических средств

Система предназначена для работы на IBM-совместимых персональных компьютерах с установленной ОС Windows 7 / 8 / 8.1 / 10.

Минимальная конфигурация:

* процессор Intel Pentium 1 ГГц 64-разрядный (х64) или выше;
* объём свободного места на жёстком диске – 30 Мб;
* объём оперативной памяти: 512 Мб или больше;
* пакет .NET Framework 4.5 или выше;
* стандартный VGA-монитор;
* манипулятор "мышь";
* клавиатура.

### 3.5 Требования к информационной и программной совместимости

Среда программирования: Visual Studio 2013, RStudio

Язык программирования: C# .NET 4.5, R.

# ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

В состав документации, сопровождающей данный программный продукт, входят следующие документы:

* расчётно-пояснительная записка;
* текст программы;
* руководство пользователя.

# СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

Стадии и этапы разработки выпускной квалификационной работы представлен в табл. П1.1.

Таблица П1.1

Стадии и этапы разработки.

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| 1 | 2 | 3 | 4 |
| Изучение предметной области и формирование требований | 6.03.17 – 16.03.17 | Перечень основных требований к разрабатываемой системе | Выполнено |
| Разработка и утверждение технического задания | 6.03.17 –  15.03.17 | Утвержденное техническое задание | Выполнено |
| Проектирование ИС: построение диаграммы вариантов использования, контекстной диаграммы классов, диаграммы последовательностей системы для всех вариантов использования и диаграммы деятельностей для одного из вариантов использования | 16.03.17 –  20.03.17 | Разработанные диаграммы вариантов использования, контекстной диаграммы классов, диаграммы последовательностей системы для всех вариантов использования и диаграммы деятельностей для одного из вариантов использования | Выполнено |

Продолжение табл. П1.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Проектирование ИС: построение диаграммы пакетов, исходной диаграммы классов, диаграмм последовательностей взаимодействия объектов классов для различных вариантов завершения, диаграммы кооперации, уточненной диаграммы классов и полной диаграммы классов из пакетов | 21.03.17 – 26.03.17 | Разработанные диаграммы пакетов, исходная диаграмма классов из пакетов, диаграммы последовательностей взаимодействия объектов классов для различных вариантов завершения, диаграммы кооперации, уточненная диаграмма классов и полная диаграмма классов из пакетов | Выполнено |
| Проектирование ИС: построение диаграммы компонентов и диаграммы размещения | 27.03.17 – 29.03.17 | Разработанные диаграммы компонентов и диаграммы размещения | Выполнено |
| Тестирование модулей ИС | 30.03.17 – 06.04.17 | Тестирование модуля взаимодействия с базой данных | Выполнено |
| Разработка и оформление документации | 07.04.17 – 09.04.17 | Расчетно-пояснительная записка, содержащая все построенные диаграммы и спецификации | Выполнено |

1. ПОРЯДОК КОНТРОЛЯ И ПРИЁМКИ

Приемка результатов работы осуществляется в соответствии с планом приемки, утвержденным на кафедре и согласованным с руководителем ВКР.

Виды испытаний, проводимые при приёме ВКР:

1. Проверка программной документации;
2. Проверка программных спецификаций на соответствие требованиям технического задания;
3. Защита выпускной квалификационной работы.

# Приложение 2

Текст программы

Модуль RLanguage.cs

Данный модуль служит для инициализации и работы с языком R. Текст программы представлен на рис. П2.1.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RDotNet;

using System.IO;

namespace RandomForestWork.Model{

class RLanguage{

/// <summary>

/// Движок языка R

/// </summary>

protected REngine engine;

/// <summary>

/// Версия R

/// </summary>

private string Rversion = "R-3.3.1";

/// <summary>

/// Путь к библиотеке R

/// </summary>

private string Rpath = Directory.GetCurrentDirectory();

public RLanguage()

{

initialize();

}

/// <summary>

/// Инициализация движка R

/// </summary>

private void initialize()

{

if (REngine.GetInstanceFromID("RDotNet") == null)

{

SetupPath();

REngine.CreateInstance("RDotNet");

engine = REngine.GetInstanceFromID("RDotNet");

engine.Initialize();

}

else

engine = REngine.GetInstanceFromID("RDotNet");}

/// <summary>

/// Настройка пути к R.dll

/// </summary>

private void SetupPath()

{

var oldPath = System.Environment.GetEnvironmentVariable("PATH");

var rPath = System.Environment.Is64BitProcess ?

string.Format(@"{0}\{1}\x64", Rpath, Rversion) :

string.Format(@"{0}\{1}\i386", Rpath, Rversion);

if (!Directory.Exists(rPath))

throw new DirectoryNotFoundException(

string.Format(" R.dll not found in : {0}", rPath));

var newPath = string.Format("{0}{1}{2}", rPath,

System.IO.Path.PathSeparator, oldPath);

System.Environment.SetEnvironmentVariable("PATH", newPath);}}}

Рис. П2.1. Текст программы модуля RLanguage.cs

Модуль IntegrationRF.cs

Данный модуль служит для трансляции модели Random Forest в SQL-код C#-код и собирает DLL библиотеку из cs файлов. Текст программы представлен на рис. П2.2.

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RandomForestWork.Model

{

// Данный класс служит для трансляции

// модели random forest в sql-код C# код и собирает DLL библиотеку из cs файлов

class IntegrationRF : RLanguage

{

// Путь к скриптам для трансляции модели на языке R

private String pathToScript;

// Путь для сохранения транслированных моделей (папака data)

private String pathToSaveData;

public IntegrationRF()

: base()

{

// Указание пути к скриптам

pathToScript = Directory.GetCurrentDirectory();

pathToScript += @"\fun";

pathToScript = pathToScript.Replace(@"\", @"\\");

// Указание пути для сохранения данных

pathToSaveData = Directory.GetCurrentDirectory();

pathToSaveData += @"\data";

}

public void translateToSQL(String fileNameRData, String tableName)

{

// Замена одинарного / на двойной // в пути к загружаемому файлу

fileNameRData = fileNameRData.Replace(@"\", @"\\");

// Загрузка функции для трансляции модели RF в SQL код

engine.Evaluate(string.Format("load('{0}')", pathToScript + @"\\RFToSQL.RData"));

// Загрузка модели RF в формате RData

engine.Evaluate(string.Format("load('{0}')", fileNameRData));

// Процесс трансляции

engine.Evaluate(string.Format("RFToSQL(rf, '{0}', 'RF')", tableName));

// Сохранения SQL кода

saveSQL();

}

public void translateToCS(String fileNameRData, String className, bool forDLL = false)

{

// Замена одинарного / на двойной // в пути к загружаемому файлу

fileNameRData = fileNameRData.Replace(@"\", @"\\");

// Создание папки для сохранения набора CS файлов

if (Directory.Exists(pathToSaveData + @"\" + className))

Directory.Delete(pathToSaveData + @"\" + className, true);

Directory.CreateDirectory(pathToSaveData + @"\" + className);

// Загрузка функции для трансляции модели RF в CS код

engine.Evaluate(string.Format("load('{0}')", pathToScript + @"\\RFToCS.RData"));

// Загрузка модели RF в формате RData

engine.Evaluate(string.Format("load('{0}')", fileNameRData));

Рис. П2.2. Текст программы модуля IntegrationRF.cs

П2.2. Продолжение

// Процесс трансляции

engine.Evaluate(string.Format("RFToCS(rf, '{0}', '{1}')", className, pathToSaveData.Replace(@"\", @"\\") + @"\\" + className));

if (!forDLL)

{

// Сохранения CS кода

saveCsharp(className);

}

}

public void translateToDLL(String fileNameRData, String className)

{

// Переменная для запуска приложений

ProcessStartInfo psi = new ProcessStartInfo();

// Создание списка CS файлов

String CsFileList = "";

// Путь к csc.exe для запуска компилятора C#. Находиться в пакете .NET

String cscFile = @"C:\Windows\Microsoft.NET\Framework\v4.5\";

translateToCS(fileNameRData, className, true);

string[] files = Directory.GetFiles(pathToSaveData + @"\" + className, "\*.cs");

foreach (string file in files)

{

CsFileList += '\"' + file + '\"' + " ";

}

// Имя запускаемого приложения

psi.FileName = "cmd";

psi.Arguments = string.Format(@"/c {0}csc /target:library /out:{1}.dll {2}", cscFile, className, CsFileList);

Process.Start(psi);

String sourceFile = Directory.GetCurrentDirectory() + @"\" + className + @".dll";

// Остановка выполнения программы и ожидание, когда файл библиотеки будет создан

while (!File.Exists(sourceFile))

;

saveDLL(className);

}

// Сохранение SQL файла

private bool saveSQL()

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

//Задаем фильтр для файлов

saveFileDialog.Filter = "Файлы SQL (\*.sql)|\*.sql";

if (saveFileDialog.ShowDialog() == true)

{

//Копируем модель RF в указанный каталог с указанным именем

File.Copy(pathToSaveData + @"\RF\_tb1.sql", saveFileDialog.FileName);

File.Delete(pathToSaveData + @"\RF\_tb1.sql");

}

else

{

File.Delete(pathToSaveData + @"\RF\_tb1.sql");

}

return true;

}

// Сохранение cs файлов

private bool saveCsharp(String className)

{

System.Windows.Forms.FolderBrowserDialog folderBrowser = new System.Windows.Forms.FolderBrowserDialog();

if (folderBrowser.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

if(Directory.Exists(folderBrowser.SelectedPath + @"\" + className))

{

if(System.Windows.MessageBox.Show("Заменить существующую папку", "Предупреждение", System.Windows.MessageBoxButton.OKCancel, System.Windows.MessageBoxImage.Asterisk) == System.Windows.MessageBoxResult.OK)

{

П2.2. Продолжение

Directory.Delete(folderBrowser.SelectedPath + @"\" + className);

}

else

return false;

}

Directory.CreateDirectory(folderBrowser.SelectedPath + @"\" + className);

foreach (string s in Directory.GetFiles(pathToSaveData + @"\" + className))

{

string s2 = folderBrowser.SelectedPath + @"\" + className + @"\" + Path.GetFileName(s);

File.Copy(s, s2);

}

Directory.Delete(pathToSaveData + @"\" + className, true);

}

else

{

Directory.Delete(pathToSaveData + @"\" + className, true);

}

return true;

}

// Сохранения DLL

private bool saveDLL(String className)

{

// Исходное место библеотеки

String sourceFile = Directory.GetCurrentDirectory() + @"\" + className + @".dll";

SaveFileDialog saveFileDialog = new SaveFileDialog();

//Задаем фильтр для файлов

saveFileDialog.Filter = "DLL файлы (\*.dll)|\*.dll";

saveFileDialog.FileName = Path.GetFileName(sourceFile);

if (saveFileDialog.ShowDialog() == true)

{

//Копируем модель RF в указанный каталог с указанным именем

File.Copy(sourceFile, saveFileDialog.FileName);

File.Delete(sourceFile);

Directory.Delete(pathToSaveData + @"\" + className, true);

}

else

{

File.Delete(sourceFile);

Directory.Delete(pathToSaveData + @"\" + className, true);

}

return true;

}

}

}

Модуль GenerationRF.cs

Данный модуль служит для генерации модели машинного обучения Random Forest. Текст программы представлен на рис. П2.3.

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

Рис. П2.3. Текст программы модуля GenerationRF.cs

П2.3. Продолжение

using System.Windows;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace RandomForestWork.Model{

// Данный класс служит для генерации

// модели машинного обучения Random Forest.

class GenerationRF : RLanguage

{

// Путь к входным данным в формате CSV

public string fileNameCSV { get; set; }

// Количество деревьев принятия решений

public int ntree { get; set; }

// Целевая переменная

public string targetVar { get; set; }

// Тип задачи (классификации (0) или регрессии (1))

public int type { get; set; }

private String pathToSaveData { get; set; }

/// <summary>

/// Путь куда была сохранена модель в формате RData

/// </summary>

public String SaveFileName { get; set; }

public GenerationRF()

: base()

{

// Указание пути для сохраненного файла

pathToSaveData = Directory.GetCurrentDirectory();

pathToSaveData += @"\data\RF.RData";

}

// Генерация модели Random Forest с помощью языка R

public void generation(string fileName, string target, int nTree, int tp)

{

fileNameCSV = fileName;

targetVar = target;

ntree = nTree;

type = tp;

generation();

}

public void generation()

{

try

{

// Проверка расширения

if (Path.GetExtension(fileNameCSV).Equals(".csv"))

{

// Замена одинарного / на двойной // в пути к загружаемому файлу

fileNameCSV = fileNameCSV.Replace(@"\", @"\\");

// Загрузка входной выборки в train, для пстроения модели RF

engine.Evaluate(string.Format(@"train<-read.csv('{0}', sep=';')", fileNameCSV));

// Подключение библиотеки Random Forest

engine.Evaluate("library(randomForest)");

// Определение типа решаемой задачи

if (type == 0)

{

// Построение модели RF

engine.Evaluate(string.Format("rf <- randomForest(as.factor(train${0})~., data=train, ntree={1})", targetVar, ntree));

}

else

{

// Построение модели RF

engine.Evaluate(string.Format("rf <- randomForest(train${0}~., data=train, ntree={1})", targetVar, ntree));

}

// Удаление предыдущего сохраненного файла

if (File.Exists(pathToSaveData))

File.Delete(pathToSaveData);

// Сохранение модели RF

engine.Evaluate(string.Format(@"save(rf, file='{0}\\data\\RF.RData')", Directory.GetCurrentDirectory().Replace(@"\", @"\\")));

saveRF();

}

else

{

MessageBox.Show("Неверный формат файла. Укажите файл с расширением CSV", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;}}

catch (RDotNet.ParseException ex)

{

MessageBox.Show(ex.Message);}}

П2.3. Продолжение

// Функция сохранения модели RF в формате RData

private bool saveRF()

{SaveFileDialog saveFileDialog = new SaveFileDialog();

// Задаем фильтр для файлов

saveFileDialog.Filter = "Файлы RData (\*.RData)|\*.RData";

if (saveFileDialog.ShowDialog() == true)

{

// Удаляем файл если он там уже есть

if (File.Exists(saveFileDialog.FileName))

File.Delete(saveFileDialog.FileName);

// Копируем модель RF в указанный каталог с указанным именем

File.Copy(pathToSaveData, saveFileDialog.FileName);

File.Delete(pathToSaveData);

SaveFileName = saveFileDialog.FileName;

}

return true;

}

}

}

Модуль InformationRF.cs

Данный модуль служит для получения информации и построения графиков модели Random Forest. Текст программы представлен на рис. П2.4.

using RDotNet;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace RandomForestWork.Model

{

/// <summary>

/// Данный класс служит для получения информации и построения графиков модели Random Forest

/// </summary>

class InformationRF : RLanguage

{

public InformationRF()

: base()

{

}

public String GetType(String fileNameRData)

{

fileNameRData = fileNameRData.Replace(@"\", @"\\");

// Загрузка модели RF в формате RData

engine.Evaluate(string.Format("load('{0}')", fileNameRData));

// Считывание типа модели Random Forest

engine.Evaluate("type<-rf$type");

CharacterVector s = engine.GetSymbol("type").AsCharacter();

String result = "";

if (s[0].Equals("classification"))

result = "Классификация";

else if (s[0].Equals("regression"))

result = "Регрессия";

return result;

}

public String GetCountTree(String fileNameRData)

{

fileNameRData = fileNameRData.Replace(@"\", @"\\");

// Загрузка модели RF в формате RData

engine.Evaluate(string.Format("load('{0}')", fileNameRData));

Рис. П2.4. Текст программы модуля InformationRF.cs

П2.4. Продолжение

// Считывание количества деревьев модели Random Forest

engine.Evaluate("countTree<-rf$ntree");

NumericVector n = engine.GetSymbol("countTree").AsNumeric();

String result = n[0].ToString();

return result;

}

public String GetTargetVar(String fileNameRData)

{

fileNameRData = fileNameRData.Replace(@"\", @"\\");

// Загрузка модели RF в формате RData

engine.Evaluate(string.Format("load('{0}')", fileNameRData));

engine.Evaluate("Call<-rf$call");

engine.Evaluate("Call<-as.character(Call)");

CharacterVector s = engine.GetSymbol("Call").AsCharacter();

String result = s[1].Replace("as.factor(train$", "");

result = result.Replace(") ~ .", "");

return result;

}

/// <summary>

/// Построение графика зависимости количества деревьев от ошибки

/// </summary>

/// <returns></returns>

public ImageSource DiagrammCounttreeByError(String fileNameRData)

{

fileNameRData = fileNameRData.Replace(@"\", @"\\");

// Загрузка модели RF в формате RData

engine.Evaluate(string.Format("load('{0}')", fileNameRData)); engine.Evaluate(string.Format(@"png(file='{0}\\data\\GraphtreesByError.png')", Directory.GetCurrentDirectory().Replace(@"\", @"\\")));

engine.Evaluate("plot(rf)");

engine.Evaluate("dev.off()");

BitmapImage bi = new BitmapImage();

byte[] imageData;

imageData = File.ReadAllBytes(Directory.GetCurrentDirectory() + @"\data\\GraphtreesByError.png");

MemoryStream ms = new MemoryStream(imageData);

ms.Position = 0;

bi.BeginInit();

bi.CacheOption = BitmapCacheOption.OnLoad;

bi.CreateOptions = BitmapCreateOptions.PreservePixelFormat;

bi.UriSource = null;

bi.StreamSource = ms;

bi.EndInit();

return bi;

}

}

}

Модуль MyData.cs

Данный модуль служит для загрузки данных и конвертации из одного формата в другой. Текст программы представлен на рис. П2.5.

using Excel;

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Data;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

Рис. П2.5. Текст программы модуля MyData.cs

П2.5. Продолжение

namespace RandomForestWork.Model

{

class MyData

{

// Переменная для хранения полного имени входного файла

private String inputFileName;

// Переменная для хранения расширения входного файла

public String extension { get; set; }

public String InputFileName

{

get

{

return inputFileName;

}

set

{

inputFileName = value;

extension = Path.GetExtension(value);

}

}

public System.Data.DataView loadData()

{

DataSet ds = new DataSet();

if ((extension.Equals(".xls")) || (extension.Equals(".xlsx")))

{

convertXlsToCsv();

}

StreamReader DataStreamReader = new StreamReader(inputFileName);

string[] columnNames = DataStreamReader.ReadLine().Split(new string[] { ";" }, StringSplitOptions.None);

DataTable table = new DataTable("CSVData");

DataColumn column = null;

for (int i = 0; i < columnNames.Length; i++)

{

column = new DataColumn(columnNames[i]);

table.Columns.Add(column);

}

ds.Tables.Add(table);

while (!DataStreamReader.EndOfStream)

{ ds.Tables["CSVData"].Rows.Add(DataStreamReader.ReadLine().Split(new string[] { ";" }, StringSplitOptions.None));

}

DataStreamReader.Close();

return ds.Tables[0].DefaultView; }

/// <summary>

/// Функция для конвертация xls в csv

/// </summary>

public void convertXlsToCsv()

{

//Путь выходного файла в формате CSV

String outputFileName = inputFileName.Substring(0, inputFileName.Length - extension.Length);

outputFileName += ".csv";

// Интерфейс для работы с Excel файлами

IExcelDataReader excelReader = null;

FileStream stream = File.Open(inputFileName, FileMode.Open, FileAccess.Read);

StreamWriter csv = new StreamWriter(outputFileName);

if (extension.Equals(".xlsx"))

excelReader = ExcelReaderFactory.CreateOpenXmlReader(stream);

else if (extension.Equals(".xls"))

excelReader = ExcelReaderFactory.CreateBinaryReader(stream);

// Считывание данных в DataSet

DataSet result = excelReader.AsDataSet();

for(int i=0; i< result.Tables[0].Rows.Count; i++)

{

String csvData = "";

for (int j = 0; j < result.Tables[0].Columns.Count; j++)

{

csvData += result.Tables[0].Rows[i][j].ToString();

if (j < result.Tables[0].Columns.Count - 1)

csvData += ";";

}

csv.WriteLine(csvData);

}

csv.Close();

stream.Close();

//Указание входным файлом, файл в формате CSV

InputFileName = outputFileName;

П2.5. Продолжение

MessageBox.Show("Файл был переформатирован. Новый файл лежит в " + InputFileName, "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Asterisk);

}

/// <summary>

/// Функция для вывода загаловка таблицы

/// </summary>

public ObservableCollection<String> GetTargetVarList()

{

ObservableCollection<String> result = new ObservableCollection<String>();

StreamReader DataStreamReader = new StreamReader(inputFileName);

String[] columnNames = DataStreamReader.ReadLine().Split(new String[] { ";" }, StringSplitOptions.None);

foreach (String column in columnNames)

{

result.Add(column);

}

return result;

}

}

}

Модуль MainWindowViewModel.cs

Данный модуль служит для организации взаимодействия пользовательского интерфейса с основным функционалом приложения. Текст программы представлен на рис. П2.6.

using RandomForestWork.Model;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using RandomForestWork;

using System.Collections.ObjectModel;

using System.IO;

using System.Drawing;

using System.Windows.Media.Imaging;

using System.Windows.Media;

namespace RandomForestWork.ViewModel

{

class MainWindowViewModel : DependencyObject

{

/// <summary>

/// Путь к входному файлу для отображения его в таблице

/// </summary>

public String InputFileName

{

get { return (String)GetValue(InputFileNameProperty); }

set { SetValue(InputFileNameProperty, value); }}

// Using a DependencyProperty as the backing store for InputFileName. This enables animation, styling, binding, etc...

public static readonly DependencyProperty InputFileNameProperty = DependencyProperty.Register("InputFileName", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Загруженные данные для отображения в таблице

/// </summary>

public DataView DownoloadDataTable

{

get { return (DataView)GetValue(DownoloadDataTableProperty); }

set { SetValue(DownoloadDataTableProperty, value); }}

Рис. П2. 6. Текст программы модуля MainWindowViewModel.cs

П2.6. Продолжение

// Using a DependencyProperty as the backing store for DownoloadDataTable. This enables animation, styling, binding, etc...

public static readonly DependencyProperty DownoloadDataTableProperty =

DependencyProperty.Register("DownoloadDataTable", typeof(DataView), typeof(MainWindowViewModel), new PropertyMetadata(null));

/// <summary>

/// Путь к входному файлу .csv для построения модели Random Forest

/// </summary>

public String InputFileNameCSV

{

get { return (String)GetValue(InputFileNameCSVProperty); }

set { SetValue(InputFileNameCSVProperty, value); }

}

// Using a DependencyProperty as the backing store for InputFileNameCSV. This enables animation, styling, binding, etc...

public static readonly DependencyProperty InputFileNameCSVProperty =

DependencyProperty.Register("InputFileNameCSV", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Количество деревьев для построения модели Random Forest

/// </summary>

public int CountTree

{

get { return (int)GetValue(CountTreeProperty); } set { SetValue(CountTreeProperty, value); }

}

// Using a DependencyProperty as the backing store for CountTree. This enables animation, styling, binding, etc...

public static readonly DependencyProperty CountTreeProperty =

DependencyProperty.Register("CountTree", typeof(int), typeof(MainWindowViewModel), new PropertyMetadata(0));

/// <summary>

/// Список целевых переменных из файла

/// </summary>

public ObservableCollection<String> TargetVarList

{

get { return (ObservableCollection<String>)GetValue(TargetVarListProperty); }

set { SetValue(TargetVarListProperty, value); }

}

// Using a DependencyProperty as the backing store for TargetVarList. This enables animation, styling, binding, etc...

public static readonly DependencyProperty TargetVarListProperty =

DependencyProperty.Register("TargetVarList", typeof(ObservableCollection<String>), typeof(MainWindowViewModel), new PropertyMetadata(null));

/// <summary>

/// Целевая переменная для построения модели Random Forest

/// </summary>

public String TargetVar

{

get { return (String)GetValue(TargetVarProperty); }

set { SetValue(TargetVarProperty, value); }

}

// Using a DependencyProperty as the backing store for TargetVar. This enables animation, styling, binding, etc...

public static readonly DependencyProperty TargetVarProperty =

DependencyProperty.Register("TargetVar", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Модель классификации

/// </summary>

public bool ClassificationModel

{

get { return (bool)GetValue(ClassificationModelProperty); }

П2.6. Продолжение

set { SetValue(ClassificationModelProperty, value); }

}

// Using a DependencyProperty as the backing store for ClassificationModel. This enables animation, styling, binding, etc...

public static readonly DependencyProperty ClassificationModelProperty =

DependencyProperty.Register("ClassificationModel", typeof(bool), typeof(MainWindowViewModel), new PropertyMetadata(false));

/// <summary>

/// Модель регрессии

/// </summary>

public bool RegressionModel

{

get { return (bool)GetValue(RegressionModelProperty); }

set { SetValue(RegressionModelProperty, value); }

}

// Using a DependencyProperty as the backing store for RegressionModel. This enables animation, styling, binding, etc...

public static readonly DependencyProperty RegressionModelProperty =

DependencyProperty.Register("RegressionModel", typeof(bool), typeof(MainWindowViewModel), new PropertyMetadata(false));

/// <summary>

/// Путь к файлу .RData с моделью Random Forest

/// </summary>

public String InputFileNameRData

{

get { return (String)GetValue(InputFileNameRDataProperty); }

set { SetValue(InputFileNameRDataProperty, value); }

}

// Using a DependencyProperty as the backing store for InputFileNameRData. This enables animation, styling, binding, etc...

public static readonly DependencyProperty InputFileNameRDataProperty = DependencyProperty.Register("InputFileNameRData", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Трансляция модели в SQL код

/// </summary>

public bool SQLCodeModel

{

get { return (bool)GetValue(SQLCodeModelProperty); }

set { SetValue(SQLCodeModelProperty, value); }

}

// Using a DependencyProperty as the backing store for SQLCodeModel. This enables animation, styling, binding, etc...

public static readonly DependencyProperty SQLCodeModelProperty = DependencyProperty.Register("SQLCodeModel", typeof(bool), typeof(MainWindowViewModel), new PropertyMetadata(false));

/// <summary>

/// Трансляция модели в C# код

/// </summary>

public bool CsCodeModel

{

get { return (bool)GetValue(CsCodeModelProperty); }

set { SetValue(CsCodeModelProperty, value); }

}

// Using a DependencyProperty as the backing store for CsCodeModel. This enables animation, styling, binding, etc...

public static readonly DependencyProperty CsCodeModelProperty =

DependencyProperty.Register("CsCodeModel", typeof(bool), typeof(MainWindowViewModel), new PropertyMetadata(false));

/// <summary>

/// Трансляция модели в Dll библиотеку

/// </summary>

public bool DllLibraryModel

{

get { return (bool)GetValue(DllLibraryModelProperty); }

set { SetValue(DllLibraryModelProperty, value); }

}

П2.6. Продолжение

// Using a DependencyProperty as the backing store for DllLibraryModel. This enables animation, styling, binding, etc...

public static readonly DependencyProperty DllLibraryModelProperty =

DependencyProperty.Register("DllLibraryModel", typeof(bool), typeof(MainWindowViewModel), new PropertyMetadata(false));

/// <summary>

/// Информация о модели. Тип

/// </summary>

public String InformationModelType

{

get { return (String)GetValue(InformationModelTypeProperty); }

set { SetValue(InformationModelTypeProperty, value); }

}

// Using a DependencyProperty as the backing store for InformationModelType. This enables animation, styling, binding, etc...

public static readonly DependencyProperty InformationModelTypeProperty =

DependencyProperty.Register("InformationModelType", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Информация о модели. Количество деревьев

/// </summary>

public String InformationModelCountTree

{

get { return (String)GetValue(InformationModelCountTreeProperty); }

set { SetValue(InformationModelCountTreeProperty, value); }

}

// Using a DependencyProperty as the backing store for InformationModelCountTree. This enables animation, styling, binding, etc...

public static readonly DependencyProperty InformationModelCountTreeProperty = DependencyProperty.Register("InformationModelCountTree", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Информация о модели. Целевая переменная

/// </summary>

public String InformationModelTargetVar

{

get { return (String)GetValue(InformationModelTargetVarProperty); }

set { SetValue(InformationModelTargetVarProperty, value); }

}

// Using a DependencyProperty as the backing store for InformationModelTargetVar. This enables animation, styling, binding, etc...

public static readonly DependencyProperty InformationModelTargetVarProperty =

DependencyProperty.Register("InformationModelTargetVar", typeof(String), typeof(MainWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// График зависимости ошибки от количества деревьев принятия решений

/// </summary>

public ImageSource ImageGraphErrorByCountTree

{

get { return (ImageSource)GetValue(ImageGraphErrorByCountTreeProperty); }

set { SetValue(ImageGraphErrorByCountTreeProperty, value); }

}

// Using a DependencyProperty as the backing store for ImageGraphErrorByCountTree. This enables animation, styling, binding, etc...

public static readonly DependencyProperty ImageGraphErrorByCountTreeProperty =

DependencyProperty.Register("ImageGraphErrorByCountTree", typeof(ImageSource), typeof(MainWindowViewModel), new PropertyMetadata(null));

// Обработка нажатия кнопки для загрузки файла

#region

private ICommand openFileDialog;

public ICommand OpenFileDialog

П2.6. Продолжение

{get

{

return openFileDialog ?? (openFileDialog = new CommandHandler(() => InputFileNameClick(), true));

}

}

private void InputFileNameClick()

{

MyData dt = new MyData();

// Диалоговое окно загрузки файлов

Microsoft.Win32.OpenFileDialog ofd = new Microsoft.Win32.OpenFileDialog();

// Установим фильтр(маску) для файлов

ofd.DefaultExt = "\*.xls; \*.xlsx; \*.csv";

ofd.Filter = "File|\*.xls; \*.xlsx; \*.csv|Excel 2003(\*.xls)|\*.xls|Excel 2007(\*.xlsx)|\*xlsx|CSV(\*.csv)|\*.csv";

// Заголовок диалогового окна

ofd.Title = "Выберите файл для загрузки данных";

if (ofd.ShowDialog() == true)

{

SetDefaultProperty();

dt.InputFileName = ofd.FileName;

LoadingIndicator loadind = new LoadingIndicator("Загрузка данных", "Открытие файла");

loadind.Show(Application.Current.MainWindow);

DownoloadDataTable = dt.loadData();

loadind.Close();

InputFileName = dt.InputFileName;

InputFileNameCSV = InputFileName;

TargetVarList = dt.GetTargetVarList();

}

}

#endregion

// Обработка нажатия кнопки для загрузки CSV файла

#region

private ICommand loadFileCSV;

public ICommand LoadFileCSV

{

get

{

return loadFileCSV ?? (loadFileCSV = new CommandHandler(() => InputFileNameCSVClick(), true));

}

}

private void InputFileNameCSVClick()

{

// Диалоговое окно загрузки файлов

Microsoft.Win32.OpenFileDialog ofd = new Microsoft.Win32.OpenFileDialog();

// Установим фильтр(маску) для файлов

ofd.DefaultExt = "\*.csv";

ofd.Filter = "File|\*.csv|CSV(\*.csv)|\*.csv";

// Заголовок диалогового окна

ofd.Title = "Выберите файл для загрузки данных";

if (ofd.ShowDialog() == true)

{

MyData dt = new MyData();

SetDefaultProperty();

InputFileNameCSV = ofd.FileName;

dt.InputFileName = InputFileNameCSV;

TargetVarList = dt.GetTargetVarList();

}

}

#endregion

// Обработка нажатия кнопки для построения модели Random Forest

#region

private ICommand generationRFCommand;

public ICommand GenerationRFCommand

{

get

{

return generationRFCommand ?? (generationRFCommand = new CommandHandler(() => generationCommandRFClick(), true));

}

}

private void generationCommandRFClick()

{

if (!InputFileNameCSV.Equals(""))

{

GenerationRF genRF = new GenerationRF();

InformationRF inform = new InformationRF();

genRF.fileNameCSV = InputFileNameCSV;

genRF.ntree = CountTree;

genRF.targetVar = TargetVar;

П2.6. Продолжение

if (ClassificationModel)

genRF.type = 0;

else if (RegressionModel)

genRF.type = 1;

LoadingIndicator loadind = new LoadingIndicator("Генерация Random Forest", "Выполняется построение модели Random Forest");

loadind.Show(Application.Current.MainWindow);

genRF.generation();

loadind.Close();

InputFileNameRData = genRF.SaveFileName;

ImageGraphErrorByCountTree = inform.DiagrammCounttreeByError(genRF.SaveFileName);

InformationModelType = inform.GetType(InputFileNameRData);

InformationModelCountTree = inform.GetCountTree(InputFileNameRData);

InformationModelTargetVar = inform.GetTargetVar(InputFileNameRData);

}

else

MessageBox.Show("Не выбран файл", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

#endregion

// Обработка нажатия кнопки для загрузки RData файла с моделью

#region

private ICommand loadFileRData;

public ICommand LoadFileRData

{

get

{

return loadFileRData ?? (loadFileRData = new CommandHandler(() => LoadFileRDataClick(), true));

}

}

private void LoadFileRDataClick()

{

// Диалоговое окно загрузки файлов

Microsoft.Win32.OpenFileDialog ofd = new Microsoft.Win32.OpenFileDialog();

// Установим фильтр(маску) для файлов

ofd.DefaultExt = "\*.RData";

ofd.Filter = "File|\*.RData|RData(\*.RData)|\*.RData";

// Заголовок диалогового окна

ofd.Title = "Выберите файл для загрузки данных";

if (ofd.ShowDialog() == true)

{InformationRF inform = new InformationRF();

SetDefaultProperty();

InputFileNameRData = ofd.FileName;

InformationModelType = inform.GetType(InputFileNameRData);

InformationModelCountTree = inform.GetCountTree(InputFileNameRData);

InformationModelTargetVar = inform.GetTargetVar(InputFileNameRData);

}

}

#endregion

// Обработка нажатия кнопки для трансляции модели Random Forest

#region

private ICommand translationRFCommand;

public ICommand TranslationRFCommand

{

get

{

return translationRFCommand ?? (translationRFCommand = new CommandHandler(() => translationRFCommandClick(), true));

}

}

private void translationRFCommandClick()

{

if(SQLCodeModel)

{

var propWindow = new PropertyWindow

{

DataContext = new PropertyWindowViewModel()

{

NameProperty = "Имя таблицы:",

TypeTranslation = 1,

InputFileNameRData = InputFileNameRData

}

};

П2.6. Продолжение

propWindow.ShowDialog(Application.Current.MainWindow); }

else if(CsCodeModel)

{

var propWindow = new PropertyWindow

{

DataContext = new PropertyWindowViewModel()

{

NameProperty = "Имя класса:",

TypeTranslation = 2,

InputFileNameRData = InputFileNameRData

}

};

propWindow.ShowDialog(Application.Current.MainWindow);

}

else if(DllLibraryModel)

{

var propWindow = new PropertyWindow

{

DataContext = new PropertyWindowViewModel()

{

NameProperty = "Имя библиотеки:",

TypeTranslation = 3,

InputFileNameRData = InputFileNameRData

}

};

propWindow.ShowDialog(Application.Current.MainWindow);

}

}

#endregion

public MainWindowViewModel()

{

SetDefaultProperty();

}

/// <summary>

/// Установка значений полей в начальное состояние

/// </summary>

private void SetDefaultProperty()

{

InputFileName = "";

DownoloadDataTable = null;

InputFileNameCSV = "";

CountTree = 300;

TargetVarList = null;

TargetVar = "";

ClassificationModel = true;

InputFileNameRData = "";

SQLCodeModel = true;

InformationModelType = "";

InformationModelCountTree = "";

InformationModelTargetVar = "";

ImageGraphErrorByCountTree = null;

}

}

}

Модуль PropertyWindowViewModel.cs

Данный модуль служит для организации взаимодействия пользовательского интерфейса с основным функционалом приложения на этапе трансляции модели. Текст программы представлен на рис. П2.7.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using RandomForestWork.Model;

using System.Windows.Input;

namespace RandomForestWork.ViewModel{

class PropertyWindowViewModel : DependencyObject{

Рис. П2.7 Текст программы модуля PropertyWindowViewModel.cs

П2.7. Продолжение

public String NameProperty

{

get { return (String)GetValue(NamePropertyProperty); }

set { SetValue(NamePropertyProperty, value); }

}

// Using a DependencyProperty as the backing store for NameProperty. This enables animation, styling, binding, etc...

public static readonly DependencyProperty NamePropertyProperty =

DependencyProperty.Register("NameProperty", typeof(String), typeof(PropertyWindowViewModel), new PropertyMetadata(""));

public String PropertyValue

{

get { return (String)GetValue(PropertyValueProperty); }

set { SetValue(PropertyValueProperty, value); }

}

// Using a DependencyProperty as the backing store for PropertyValue. This enables animation, styling, binding, etc...

public static readonly DependencyProperty PropertyValueProperty =

DependencyProperty.Register("PropertyValue", typeof(String), typeof(PropertyWindowViewModel), new PropertyMetadata(""));

/// <summary>

/// Тип трансляции модели: 1-SQL, 2-C#, 3-DLL

/// </summary>

public int TypeTranslation { get; set; }

/// <summary>

/// Путь к файлу RData с моделью Random Forest

/// </summary>

public String InputFileNameRData { get; set; }

// Обработка нажатия кнопки Ок

#region

private ICommand okCommand;

public ICommand OkCommand

{

get

{

return okCommand ?? (okCommand = new CommandHandler(() => OkCommandClick(), true));

}

}

private void OkCommandClick()

{

IntegrationRF integRF = new IntegrationRF();

switch (TypeTranslation)

{

case 1:

{integRF.translateToSQL(InputFileNameRData, PropertyValue);

break;

}

case 2:

{ integRF.translateToCS(InputFileNameRData, PropertyValue);

break;

}

case 3:

{

integRF.translateToDLL(InputFileNameRData, PropertyValue);

break;

}

}

WindowCollection windows = Application.Current.Windows;

foreach(Window w in windows)

{

if (w.IsActive)

w.Close();

}

}

#endregion

}

}

Модуль LoadingIndicatorViewModel.cs

Данный модуль служит для отображения уровня выполнения операции. Текст программы представлен на рис. П2.8.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RandomForestWork.ViewModel

{

class LoadingIndicatorViewModel

{

public String LoadingText { get; set; }

public String LoadingToolTip { get; set; }

public LoadingIndicatorViewModel()

{

LoadingText = "Загрузка...";

LoadingToolTip = "Подождите, выполняется операция.";

}

public LoadingIndicatorViewModel(String loadingText)

{

LoadingText = loadingText;

LoadingToolTip = "Подождите, выполняется операция.";

}

public LoadingIndicatorViewModel(String loadingText, String loadingToolTip)

{

LoadingText = loadingText;

LoadingToolTip = loadingToolTip;

}

}

}

Рис. П2.8. Текст программы модуля LoadingIndicatorViewModel.cs

Модуль RFToSQL.RData

Данный модуль разработан на языке программирования R и служит для трансляции модели Random Forest в SQL код. Текст программы представлен на рис. П2.9.

# Функция для генерации модели Random Forest в БД.

# Создается таблица с деревьями, и две хранимые процедуры:

# одна считает по одному дереву, другая инициирует расчет по всем деревьям и усрелняет результат

rf2tablesqlfunc <- function (model, tableName, fileName) {

require (randomForest, quietly=TRUE)

if (!("randomForest" %in% class(model)))

{

stop ("Функция работает только с RandomForest")

return

}

#=================================================================================

#===================== СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ ДЕРЕВЬЕВ ====================

#=================================================================================

sink(paste(fileName, "\_tbl.sql", sep=""), type="output")

cat(paste("set feedback off", "\n", sep=""))

cat(paste("set define off", "\n\n", sep=""))

cat(paste("CREATE TABLE ", tableName, "\n", sep=""))

cat(paste("(", "\n", sep=""))

cat(paste("ID NUMBER not null, ", "\n", sep=""))

Рис. П2.9. Текст программы модуля RFToSQL.RData

П2.9. Продолжение

cat(paste("TREE\_NUM NUMBER not null, ", "\n", sep=""))cat(paste("PARAM\_NAME VARCHAR2(50), ", "\n", sep=""))

cat(paste("COND\_VALUE NUMBER, ", "\n", sep=""))

cat(paste("TRUE\_ID NUMBER, ", "\n", sep=""))

cat(paste("FALSE\_ID NUMBER, ", "\n", sep=""))

cat(paste("NODE\_TYPE NUMBER, ", "\n", sep=""))

cat(paste("VALUE NUMBER", "\n", sep=""))

cat(paste("); ", "\n", sep=""))

#cat(paste("commit;", "\n\n", sep=""))

cat(paste("comment on column ", tableName, ".ID is 'PK';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".TREE\_NUM is 'Номер дерева';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".PARAM\_NAME is 'Имя параметра';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".COND\_VALUE is 'Значение для сравнения';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".TRUE\_ID is 'FK (откл.) ссылка на ID узла если условие выполняется';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".FALSE\_ID is 'FK (откл.) ссылка на ID узла если условие НЕ выполняется';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".NODE\_TYPE is 'Тип узла: 0 - корневой, 1 - условие, 2 - терминальный';", "\n", sep=""))

cat(paste("comment on column ", tableName, ".VALUE is 'Результат (для терминального узла)';", "\n", sep=""))

cat(paste("alter table ", tableName, " add constraint PK\_", tableName, " primary key (ID) using index;", "\n", sep=""))

cat(paste("commit;", "\n\n", sep=""))

treeStartId = 0

# Для каждого дерева создаем пачку insert'ов

for (tree.num in 1:(model$ntree))

{

# Нужен еще признак root

generateInsert <- function(model, tree.data, tree.row.num, id, nodeType, tableName) {

nType = nodeType

curNodeId = id + 1

tree.row <- tree.data[tree.row.num,]

split.var <- as.character(tree.row[,"split var"])

split.point <- tree.row[,"split point"]

if(tree.row[,"status"] != -1) # Разделяющий узел

{

if(is.numeric(unlist(model$forest$xlevels[split.var])))

{

# ветка true

trueId = curNodeId + 1

lastTrueId = generateInsert(model, tree.data, tree.row[,"left daughter"], curNodeId, 1, tableName) # в рекурсии происходит увеличение curNodeId на один и получим trueId

# ветка false

falseId = lastTrueId + 1

lastFalseId = generateInsert(model, tree.data, tree.row[,"right daughter"], lastTrueId, 1, tableName) # в рекурсии происходит увеличение lastTrueId на один и получим falseId

# INSERT INTO table\_name VALUES (value1,value2,value3,...)

# curNodeId, trueId, falseId, nodeType = nType

cat(paste("INSERT INTO ", tableName, " VALUES (", curNodeId, ", ", tree.num, ", '", gsub("[.]","\_",split.var), "', ", split.point, ", ", trueId, ", ", falseId, ", ", nType, ", ", "null", ");", sep=""))

cat("\n") # переход на следующую строку

return (lastFalseId)

}

else # категориальные величины

{

stop ("Функция работает только с числовыми значениями")}}

else # узел с результатом{

nType = 2

# INSERT INTO table\_name VALUES (value1,value2,value3,...)

# curNodeId, nodeType = 2 строго

cat(paste("INSERT INTO ", tableName, " VALUES (", curNodeId, ", ", tree.num, ", ", "null", ", ", "null", ", ", "null", ", ", "null", ", ", nType, ", ", tree.row[,"prediction"], ");", sep=""))

П2.9. Продолжение

cat("\n") # переход на следующую строку

return (curNodeId)

}}

treeStartId = generateInsert(model, getTree(model, k=tree.num, labelVar=TRUE), 1, treeStartId, 0, tableName)

}

cat(paste("commit;", "\n", sep=""))

cat(paste("set feedback on", "\n", sep=""))

cat(paste("set define on", "\n", sep=""))

sink() #закончили работу с файлом

#=======================================================================================

#===================== КОНЕЦ СОЗДАНИЯ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ ДЕРЕВЬЕВ ====================

#=====================================================================================

#=================================================================================

#===================== СОЗДАНИЕ ХРАНИМЫХ ПРОЦЕДУР ================================

#=================================================================================

sink(paste(fileName, "\_pkg.sql", sep=""), type="output")

cat(paste("create or replace package body PKG\_", tableName, " as", "\n", sep="")) # открываем пакет

paramsWithType = ""

paramsNoType = ""

duals = ""

for(xNum in 1:(length(names(model$forest$xlevels))))

{

del = ", "

if(paramsWithType == "")

{

del = ""

}

xName = names(model$forest$xlevels)[xNum]

paramsWithType = paste(paramsWithType, del, "p", xName, " number", sep="")

paramsNoType = paste(paramsNoType, del, "p", xName, sep="")

del1 = "union "

if(duals == "")

{

del1 = ""

}

duals = paste(duals, del1, "select p", xName, " val, '", xName, "' param\_name from dual", "\n", sep="")

}

cat(paste("function CALC\_TREE\_", tableName, "(pROOTID number, ", paramsWithType, ") return number is Result number;", "\n", sep="")) # заголовок функции

cat(paste("begin", "\n", sep=""))

cat(paste("select max(value) val into Result", "\n", sep=""))

cat(paste("from (select \*", "\n", sep=""))

cat(paste(" from (select t.id,", "\n", sep=""))

cat(paste(" case", "\n", sep=""))

cat(paste(" when p.val <= t.cond\_value then true\_id else false\_id", "\n", sep=""))

cat(paste(" end node\_id", "\n", sep=""))

cat(paste(" from ", tableName, " t,", "\n", "(" , sep=""))

cat(paste(duals, "\n", sep=""))

cat(paste(") p", "\n" , sep=""))

cat(paste(" where t.param\_name = p.param\_name) t", "\n" , sep=""))

cat(paste(" connect by t.id = prior t.node\_id", "\n" , sep=""))

cat(paste(" start with t.id = pROOTID) t1,", "\n" , sep=""))

cat(paste(" " ,tableName, " t0", "\n" , sep=""))

cat(paste("where t1.node\_id = t0.id;", "\n" , sep=""))

cat(paste("return(Result);", "\n" , sep=""))

cat(paste("end Calc\_Tree\_", tableName, ";", "\n" , sep=""))

cat(paste("\n\n" , sep=""))

cat(paste("function CALC\_", tableName, "(", paramsWithType, ") return number is Result number;", "\n", sep="")) # заголовок функции

cat(paste("begin", "\n", sep=""))

cat(paste(" select avg(Calc\_Tree\_", tableName, "(t.id, ", paramsNoType, ")) val into Result", "\n", sep=""))

П2.9. Продолжение

cat(paste(" from ", tableName, " t", "\n", sep=""))

cat(paste(" where t.node\_type = 0;", "\n", sep=""))

cat(paste(" return(Result);", "\n", sep=""))

cat(paste("end Calc\_", tableName, ";", "\n", sep=""))

cat(paste("end PCK\_", tableName, ";", "\n", sep="")) # закрываем пакет

sink() #закончили работу с файлом

# КОНЕЦ СОЗДАНИЯ ХРАНИМЫХ ПРОЦЕДУР

}

Модуль RFToCS.RData

Данный модуль разработан на языке программирования R и служит для трансляции модели Random Forest в C# код. Текст программы представлен на рис. П2.10.

RFToCS <- function (model, className, pathToSave)

{

require (randomForest, quietly=TRUE)

if (!("randomForest" %in% class(model)))

{

stop ("Функция работает только с RandomForest")

return

}

sink(paste(pathToSave, "\\", className,"Main.cs", sep=""), type="output") # Создаем главный файл partial-classа

cat(paste("namespace RandomForest.", className, sep="")) # открыли namespace

cat(paste("\n", "{", "\n", sep=""))

cat(paste("public partial class", className))

cat(paste("\n", "{", sep=""))

# в зависисмоти от задачи определяем тип данных

if (model$type == "classification")

{

pred.type <- "VARCHAR"

typeText = "string"

}

else

{

pred.type <- "FLOAT"

typeText = "double"

}

#Создание строки с списком параметров с объявлением и без

param.listAndType = "" #Строка для хранения списка переменных с типом данных (для объявления функций)

param.list = "" #Строка для хранения списка переменных без типом данных (для вызова функций)

#Сохраним в список l все нетерминальные символы

l<-rownames(as.matrix(model$forest$xlevels))

param.num = 1

while(!is.na(l[param.num]))

{

if( is.numeric(unlist(model$forest$xlevels[l[param.num]])))

{

param.listAndType = paste(param.listAndType, "double ", l[param.num], sep="")

}

else

{

Рис. П2.10. Текст программы модуля RFToCS.RData

П2.10. Продолжение

param.listAndType = paste(param.listAndType, "string ", l[param.num], sep="")

}

param.list = paste(param.list, l[param.num], sep="")

if(!is.na(l[param.num+1]))

{

param.listAndType = paste(param.listAndType, ", ", sep="")

param.list = paste(param.list, ", ", sep="")

}

param.num=param.num+1}

# Создаем функцию, которая вызывает расчеты для всех деревьев

cat(paste("\n", "public ", typeText, "[] CalculateAllTrees(", param.listAndType, ")", sep=""))

cat(paste("\n", "{"))

cat(paste("\n", "var v = new ", typeText, "[",model$ntree,"];", sep=""))

#вызовы функций по каждому дереву

for (tree.num in 1:(model$ntree))

{

cat(paste("\n", "v[", tree.num-1 ,"] = CalculateTree", tree.num, "(", param.list, ");", sep=""))

}

cat("\n", "\n", "return v;")

cat("\n", "}") # закрыли функцию расчета всех деревьев

cat(paste("\n", "}")) # закрыли класс

cat(paste("\n", "}", sep="")) # закрыли namespace

sink() # закончили работу с главным файлом

# Для каждого дерева создаем файл и описываем в нем дерево как функцию partial-класса

for (tree.num in 1:(model$ntree))

{

sink(paste(pathToSave, "\\", className, "Tree", tree.num, ".cs", sep=""), type="output")

# Для каждого дерева создаем новую функцию в partial-class'е

cat(paste("using System.Collections.Generic;"))

cat(paste("\n", "namespace RandomForest.", className, sep="")) # открыли namespace

cat(paste("\n", "{", sep=""))

cat(paste("\n\t", "public partial class ", className, sep =""))

cat(paste("\n\t{", sep =""))

cat(paste("\n\t\t", "private ", typeText, " CalculateTree", tree.num, "(", param.listAndType, ")", sep=""))

cat(paste("\n\t\t", "{", sep =""))

cat(paste("\n\t\t\t", typeText, " v;", sep=""))

cat("\n\n\t\t\t","#region tree ", tree.num, sep="")

cat("\n\n\t\t\t", sep="")

# model - модель RF, tree.data - данные дерева, полученные с помощью getTree(),

recurse.rf <- function(model, tree.data, tree.row.num, ind=4) {

tree.row <- tree.data[tree.row.num,]

#берем строчку из tree.data под номером tree.row.num

indent.str <- paste(rep("\t", ind), collapse="")

#строка с табами

split.var <- as.character(tree.row[,"split var"]) # название переменной

split.point <- tree.row[,"split point"] # Значение с которым сравнивается split.var

if(tree.row[,"status"] != -1) # Разделяющий узел

{

if(is.numeric(unlist(model$forest$xlevels[split.var])))

{

cat("if(" ,gsub("[.]" ,"\_" ,split.var), " <= ", split.point, ")", sep="")

cat("\n",indent.str, sep="")

recurse.rf(model, tree.data, tree.row[,"left daughter"], ind=(ind+1))

cat("\n",indent.str,"else ", sep="")

recurse.rf(model, tree.data, tree.row[,"right daughter"], ind=(ind+1))

}

else # категориальные величины

{

conv2binary <- function (ncat, num.to.convert) # Функция для конвертирования категориальной величины в бинарный массив

{

ret <- numeric()

if((2^ncat) <= num.to.convert)

{

return (NULL)

}

else

{

for (x in (ncat - 1):0)

{

if (num.to.convert >= (2^x))

{

П2.10. Продолжение

num.to.convert <- num.to.convert - (2^x)

ret <- c(ret, 1)

}

else

{

ret <- c(ret, 0)

}}

return(ret) }}

categ.bin <- conv2binary(model$forest$ncat[split.var], split.point)

categ.flags <- (categ.bin[length(categ.bin):1] == 1)

categ.values <- unlist(model$forest$xlevels[split.var])

cat(paste("if(new List<string> ", "{\"",

paste(categ.values[categ.flags], sep="", collapse="\", \""),

"\"}.Contains(", gsub("[.]","\_",split.var), "))", sep=""))

cat("\n",indent.str, sep="")

recurse.rf(model, tree.data, tree.row[,"left daughter"], ind=(ind+1))

cat("\n",indent.str,"else ", sep="")

recurse.rf(model, tree.data, tree.row[,"right daughter"], ind=(ind+1)) }}

else # узел с результатом

{

if (is.numeric(tree.data$prediction))

{

cat(paste("v[",tree.num,"]="))

cat(paste(tree.row[,"prediction"], " ", sep=""))

cat(paste(";"))

}

else

{

cat(paste("v = "))

cat(paste("\"", tree.row[,"prediction"], "\"", sep=""))

cat(paste(";", sep=""))

}

}

}

recurse.rf(model, getTree(model,k=tree.num,labelVar=TRUE), 1)

cat("\n\n\t\t\t","#endregion", sep="")

cat("\n\n\t\t\t","return v;", sep="")

cat(paste("\n\t\t", "}", sep="")) # закрыли функцию

cat(paste("\n\t", "}", sep="")) # закрыли класс

cat(paste("\n", "}", sep="")) # закрыли namespace

sink() #закончили работу с файлом

}

}

# Приложение 3

Спецификации

Спецификация на документацию и компоненты представлена в табл. П3.1.

Таблица П3.1

Спецификация

|  |  |  |
| --- | --- | --- |
| Обозначение | Наименование | Примечание |
| Документация | | |
| Расчётно-пояснительная записка | «Разработка программного обеспечения построения и интеграции модели Random Forest» |  |
| Приложение 2 | Текст программы |  |
| Приложение 4 | Руководство пользователя |  |
| Приложение 5 | Наборы тестовых данных |  |
| Компоненты | | |
| RandomForestWork.exe | Файл установки |  |

# Приложение 4

Руководство пользователя

Данное программное обеспечения предназначено для упрощения и автоматизации процесса построения модели Random Forest, которая может решать задачу классификации или регрессии. Также данное ПО служит для трансляции полученной модели в SQL код, C# код и DLL библиотеку, чтобы можно было использовать построенную модель в таких программных продуктах как СУБД Oracle и MS Visual Studio.

Для работы с данным ПО использовуются наборы данных, которые представлены в формате csv, xls или xlsx. Также для выполнения отедльных операций используются файлы с построенной моделью Random Forest в файле формата RData.

Описание установки

Перед началом работы с программой на рабочем ПК должно быть предустановленн следующее программаное обеспечение:

* Microsoft .NET Framework 4.5;
* R 3.3.1 с установленным пакетом «randomForest».

Описание запуска

Для работы с программой, необходимо запустить исполняемый файл *RandomForestWork.exe*, находящийся в рабочем каталоге программного обеспечения. В случае успешного запуска, на экране компьютера будет отображено начальное окно программы (рис. П4.1).

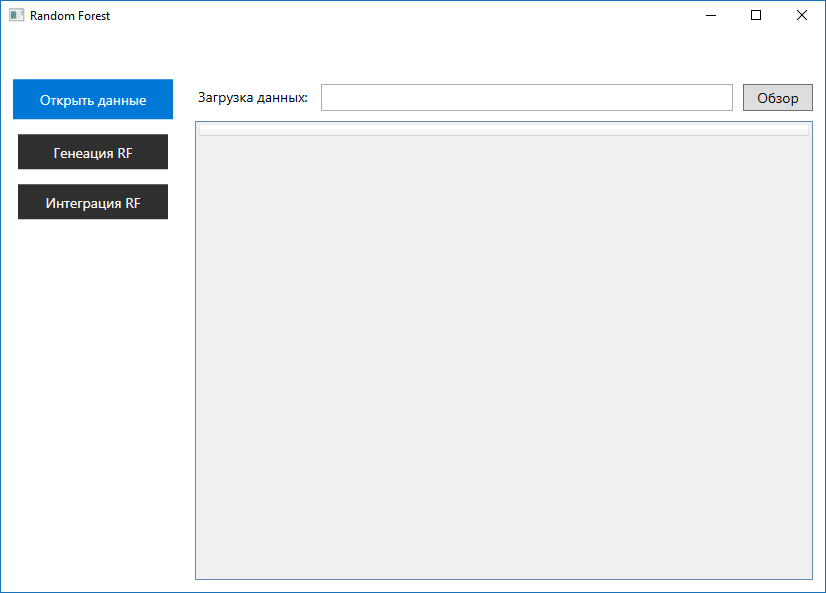


Рис. П4.1. Начальное окно приложения

Работа с программой

Работу приложения можно рассматривать как последевательность шагов:

* просмотр данных;
* построение модели Random Forest;
* трансляция модели Random Forest;

Просмотр данных

Для загрузки данных для просмотра необходимо выбрать вкладку «Открыть данные» (рис. П4.1). Чтобы загрузить файл необходимо нажать кнопку «Обзор» (рис. П4.2).

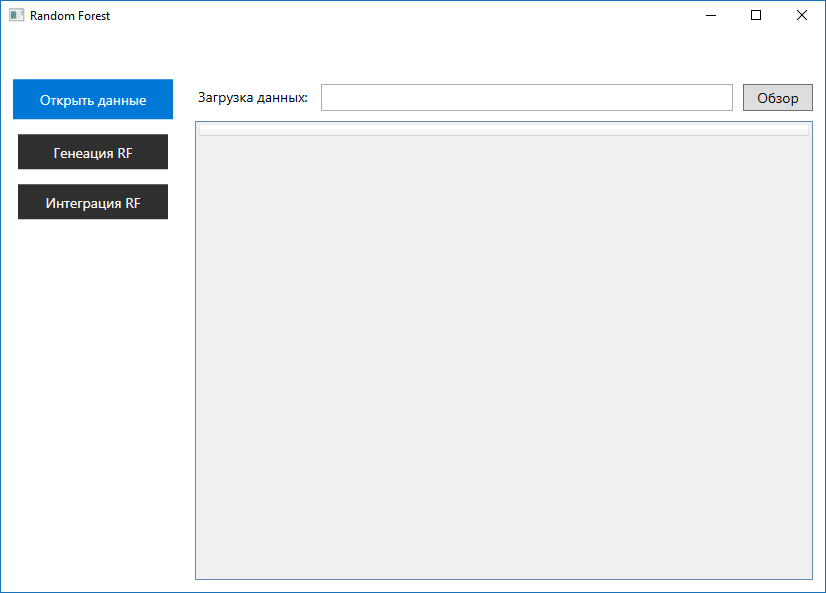


Рис. П4.2. Просмотр данных. Кнопка обзор

После нажатия на кнопку «Обзор» появиться окно (рис. П4.3), с помощью которого можно выбрать файл в формате csv, xls или xlsx на локальном компьютере. Файл должен содержать набор данных для решения задачи классификации или регрессии.

После выбора нужного файла необходимо нажать на кнопку «Открыть». После чего система начнет считывание данных файла в таблицу на форме.

Во время считывания данных появится иформационное окно с отображением выполнения операции (рис. П4.4).

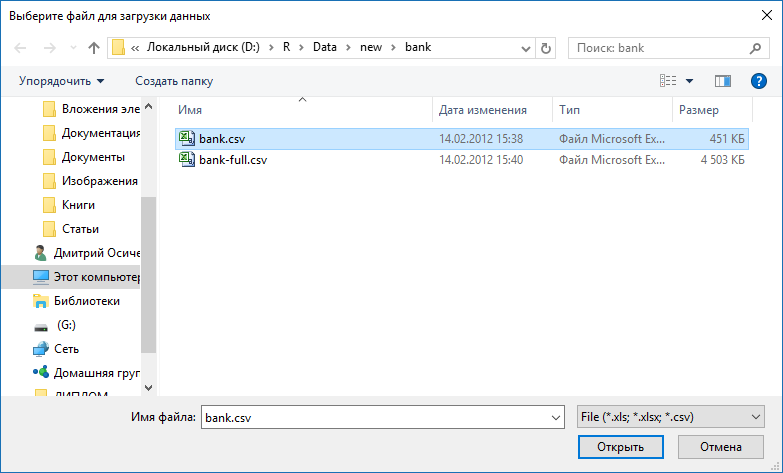


Рис. П4.3. Окно для выбора входного файла с выборкой

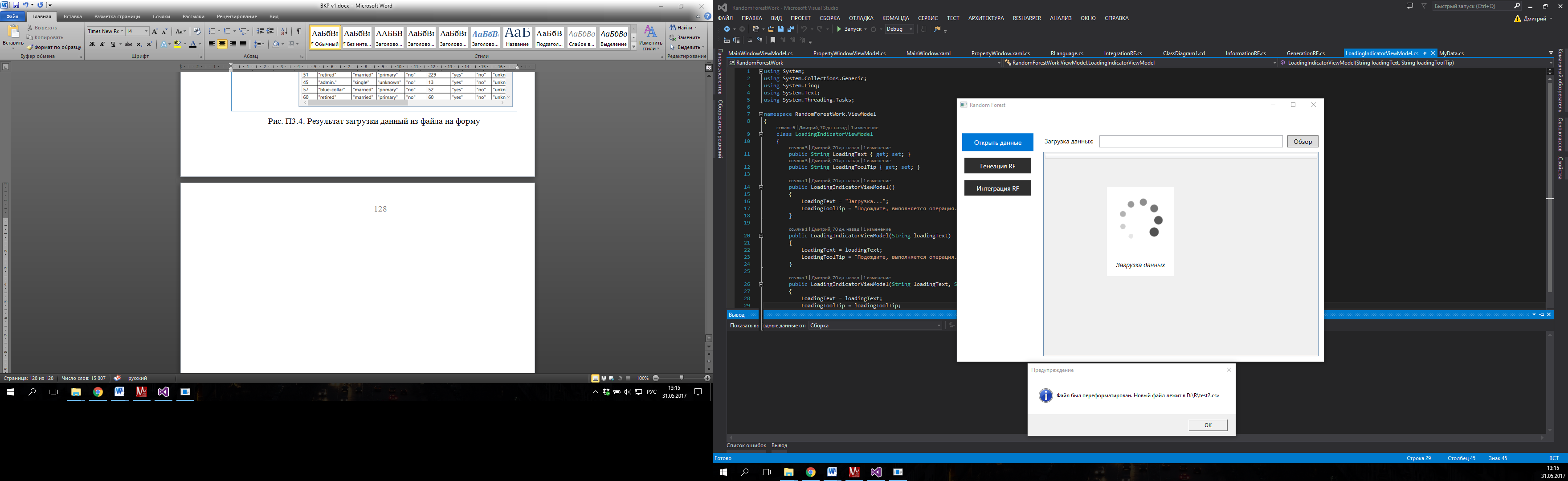


Рис. П4.4. Идикатор выполнения загрузки данных

В случае успешной загрузки данных, они будут представлены в таблице на форме, также будет представлен полный путь к входному файлу (рис. П4.5). Если был выбран файл формата xls или xlsx приложение автоматически переформатирует его в формат csv и отобразит соотвествующее информационное окно (рис. П4.6).

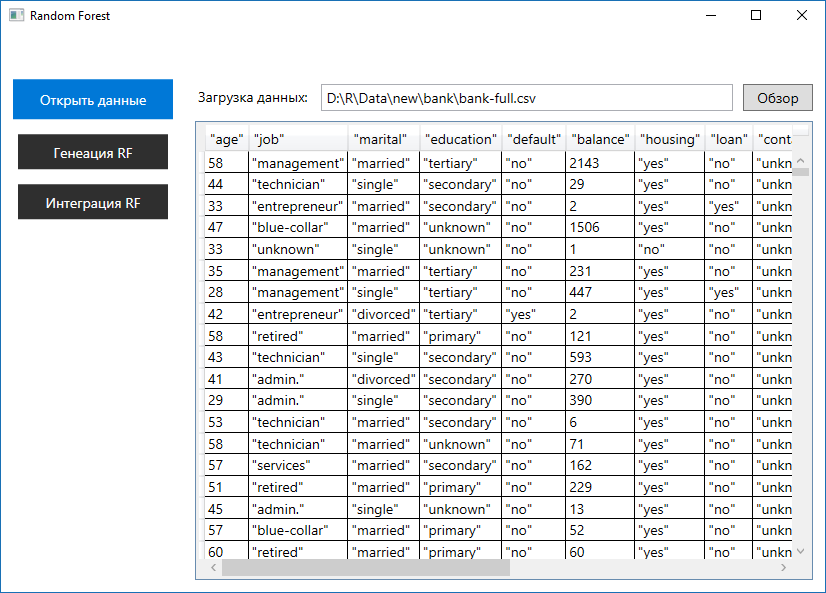


Рис. П4.5. Результат загрузки данный из файла на форму

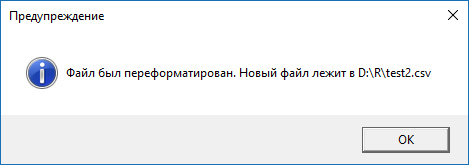


Рис. П4.6. Информационное окно о переформатировании файла

Построение модели Random Forest

Окно для построения модели Random Forest представлено на рис. П4.7.

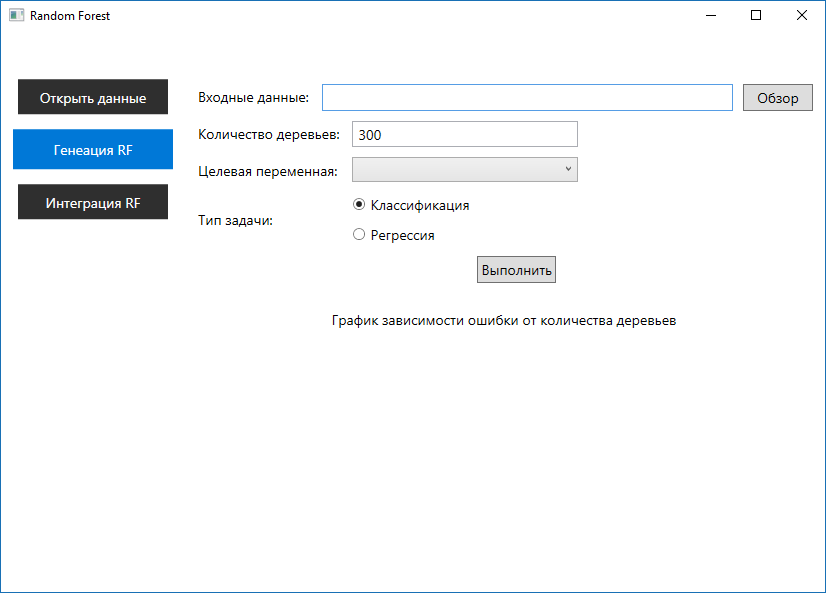


Рис. П4.7. Окно для построения модели Random Forest

Для построения модели необходим файл в формате csv. Если файл был выбран на шаге «Просмотр данных», то путь до него отобразиться в графе «Входные данные» (рис. П4.8). Или можно выбрать другой файл, нажав кнопку «Обзор», в этом случае появиться окно для выбора файла аналогично выбору файлна на этапе просмотра данных (рис. П4.3).

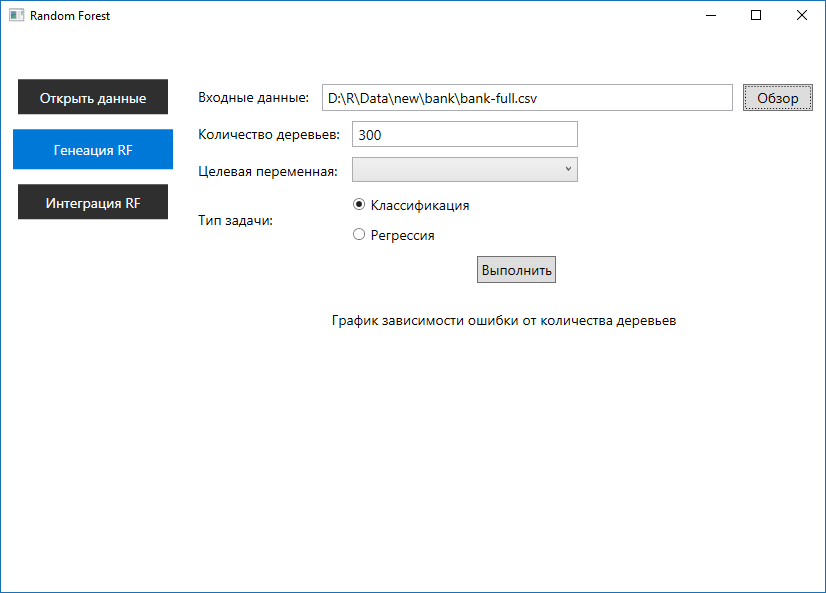


Рис. П4.8. Отображения пути к файлу

Из входного файла приложение считывает название загаловков и предлагает выбрать один из данных заголовков в качестве целевой переменной (рис. П4.9).

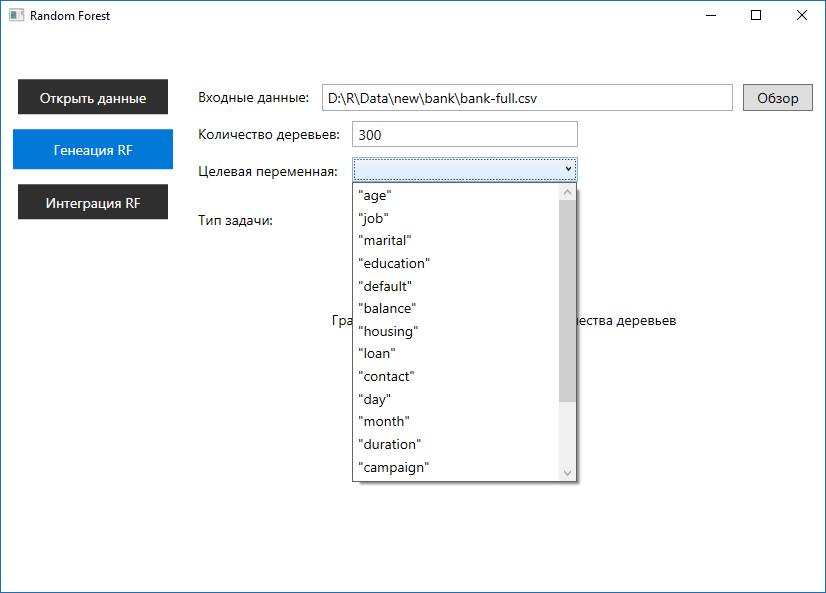


Рис. П4.9. Выбор целевой переменной

Также необходимо указать количество деревьев принятий решений. Допустиме значение: 1-1000. Дальше необходимо выбрать один из типов задач: регрессия или классификация.

Для построения модели необходимо нажать на кнопку «Выполнить». В случае если не все параметры были указаны приложение выдаст сообщение об ошибке (рис. П4.10).

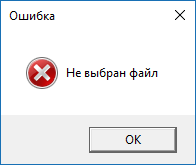


Рис. П4.10. Сообщение об ошибке на этапе построения модели

В случае успешного построения модели приложение построит график зависимости количества деревьев принятия решений от точности прогноза модели (рис. П4.11), с помощью которого пользователь может поддкоректировать количество деревьев принятия решений.

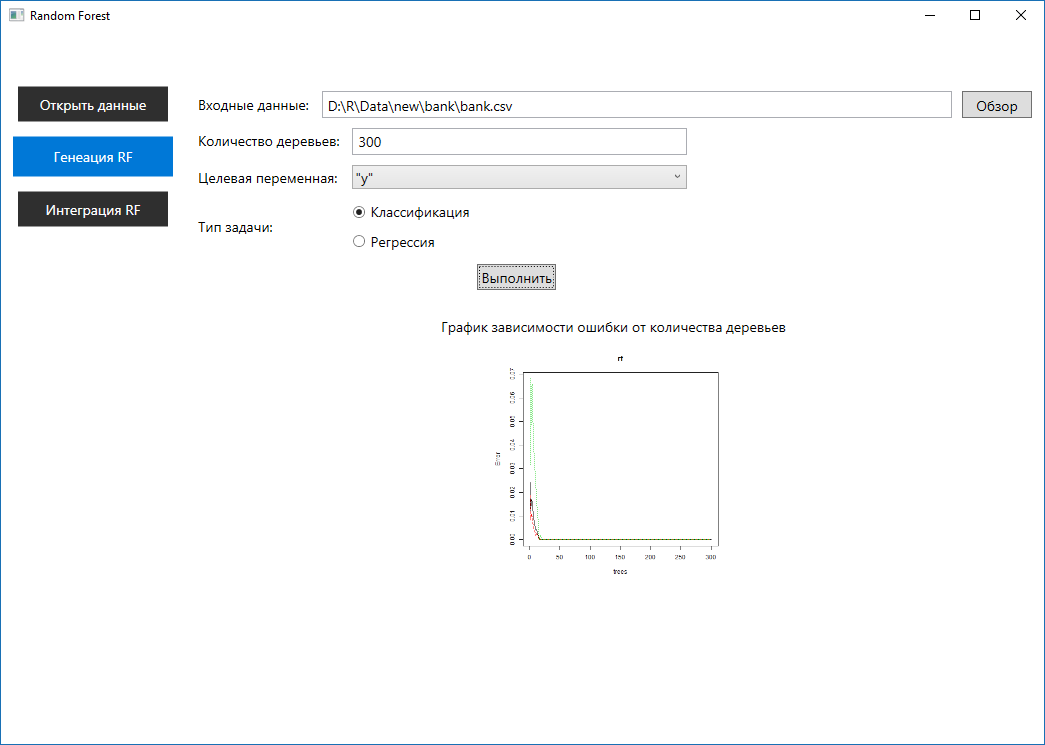


Рис. П4.11. Построение графика

После приложение предложит указать имя и место для сохранения модели на локальном диске в формате RData (рис. П4.12).

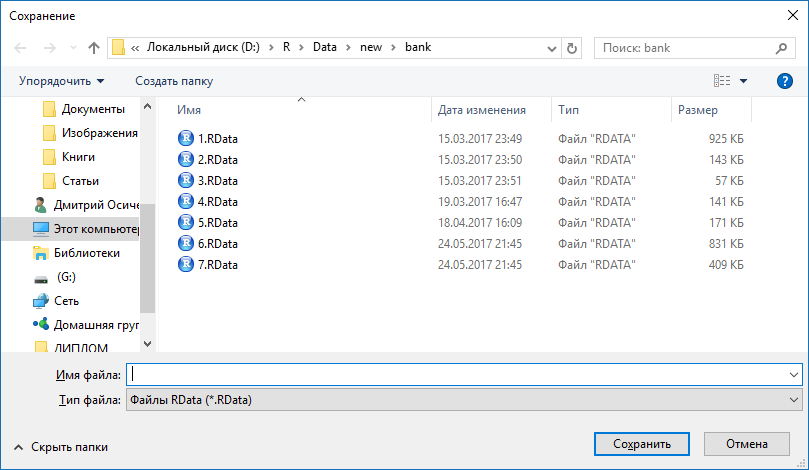


Рис. П4.12. Сохранение полученной модели

Трансляция модели Random Forest

Окно для трансялции модели Random Forest представлено на рис. П4.13.

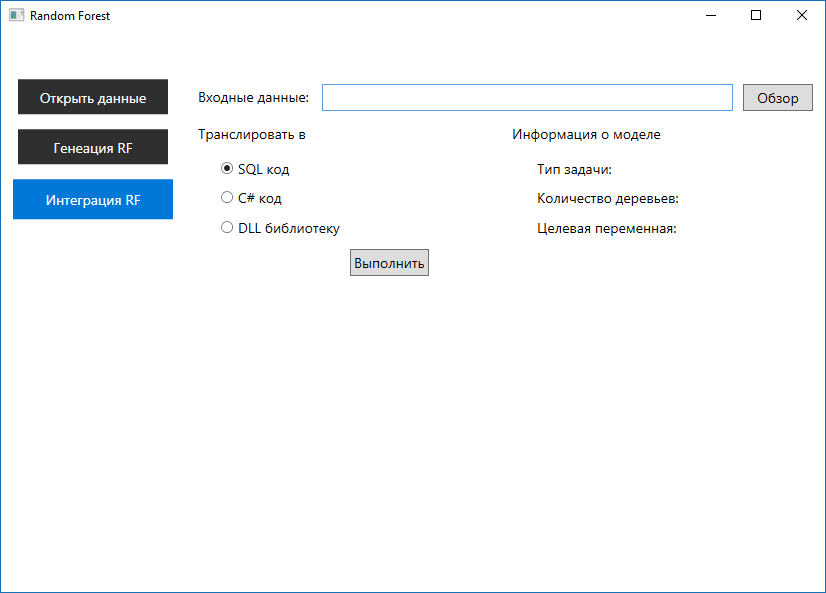


Рис. П4.13. Окно для трансляции модели Random Forest

Для трансляции модели необходим файл в формате RData с построенной моделью Random Forest. Для загрузки файла необходимо нажать кнопку «Обзор», в этом случае появиться окно для выбора файла аналогично выбору файлна на этапе просмотра данных (рис. П4.3).

После выбора файла приложение подгрузит информацию о выбранной модели: тип задачи, количество деревьев решений и целевую переменную (рис. П4.14).

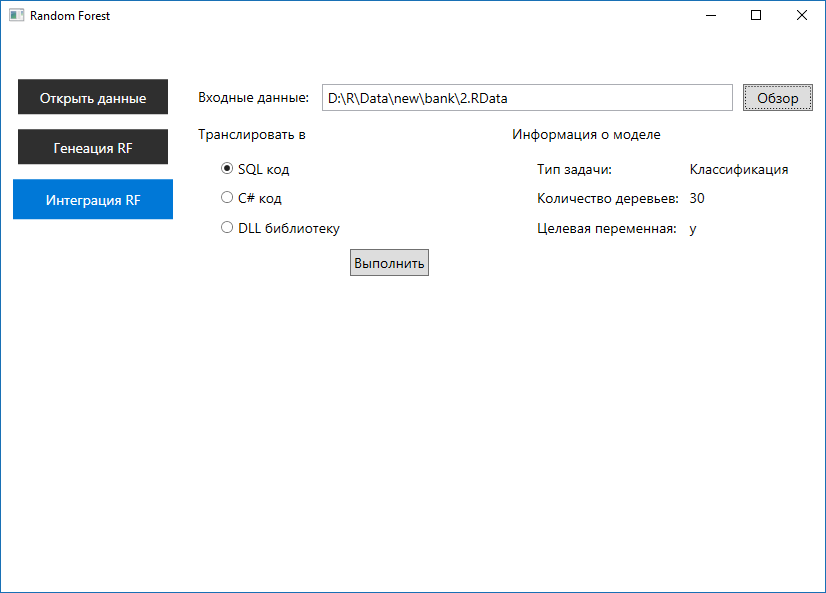


Рис. П4.14. Вывод информации о модели Random Forest

Далее необходимо выбрать тип один из трех типов, в который произодить трансляцию модели. После выбора типа трансляции необходимо нажать на кнопку выполнить.

Для трансляции модели в SQL код, необходимо чтобы модель не содержала категориальных величин, а только числовые. Для этого необходимо все категориальные переменные сделать целочисленными переменными, обозначив каждому классу свое числовое значение.

Трансляция в SQL код

После нажатия кнопки «Выпонить» появиться окно с просьбой ввести названия имени таблицы (рис. П4.15).

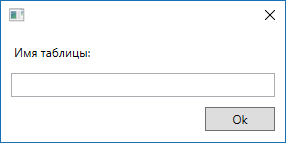


Рис. П4.15. Ввод имени таблицы

После того как трансляция будет выполнена появиться окно для сохранения SQL кода. В данном окне можно указать имя и место для сохранения файла. Вид окна аналогичен окну представленном на рис. П4.12.

В результате трансляции будет создано два файла. Один содержит операции для создания и заполнения таблицы. Второй предназначен для создания хранимой процедуры, с помощью которой осуществляется обход деревьев.

Трансляция в C# код

После нажатия кнопки «Выпонить» появиться окно с просьбой ввести названия класса (рис. П4.16). Трансляция будет производиться в набор файлов, каждый из которых представляет одно дерево плюс один класс для их объединения.

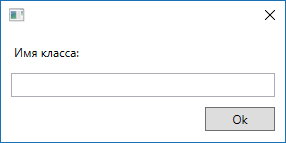


Рис. П4.16. Ввод имени класса

После того как трансляция будет выполнена появиться окно для выбора места для сохранения папки с C# кодом (рис. П4.17).

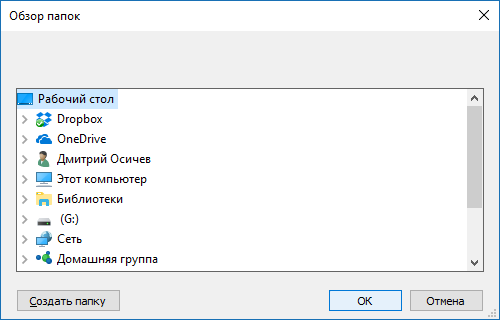


Рис. П4.17. Выбор места для сохранения C# кода

Трансляция в DLL библиотеку

После нажатия кнопки «Выпонить» появиться окно с просьбой ввести название имени библиотеки (рис. П4.18). Во время выполнения трансляции будет вызвана консоль для выполнения необходимых операций.

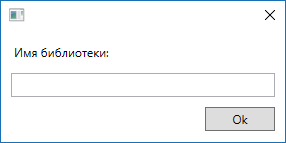


Рис. П4.18. Ввод имени библиотеки

Для завершения работы с приложением достаточно закрыть его, нажав на значок «крестика» в правом верхнем углу приложения.

Подключение полученной библиотеки к проекту .NET

Для использования библиотеки с полученной модели Random Forest в своем проекте необходимо сделать следующее:

1. В обозревателе решений на вкладке References, с помощью правой клавиши мыши, необходимо выбрать «Добавить ссылку…» (рис. П4.19)

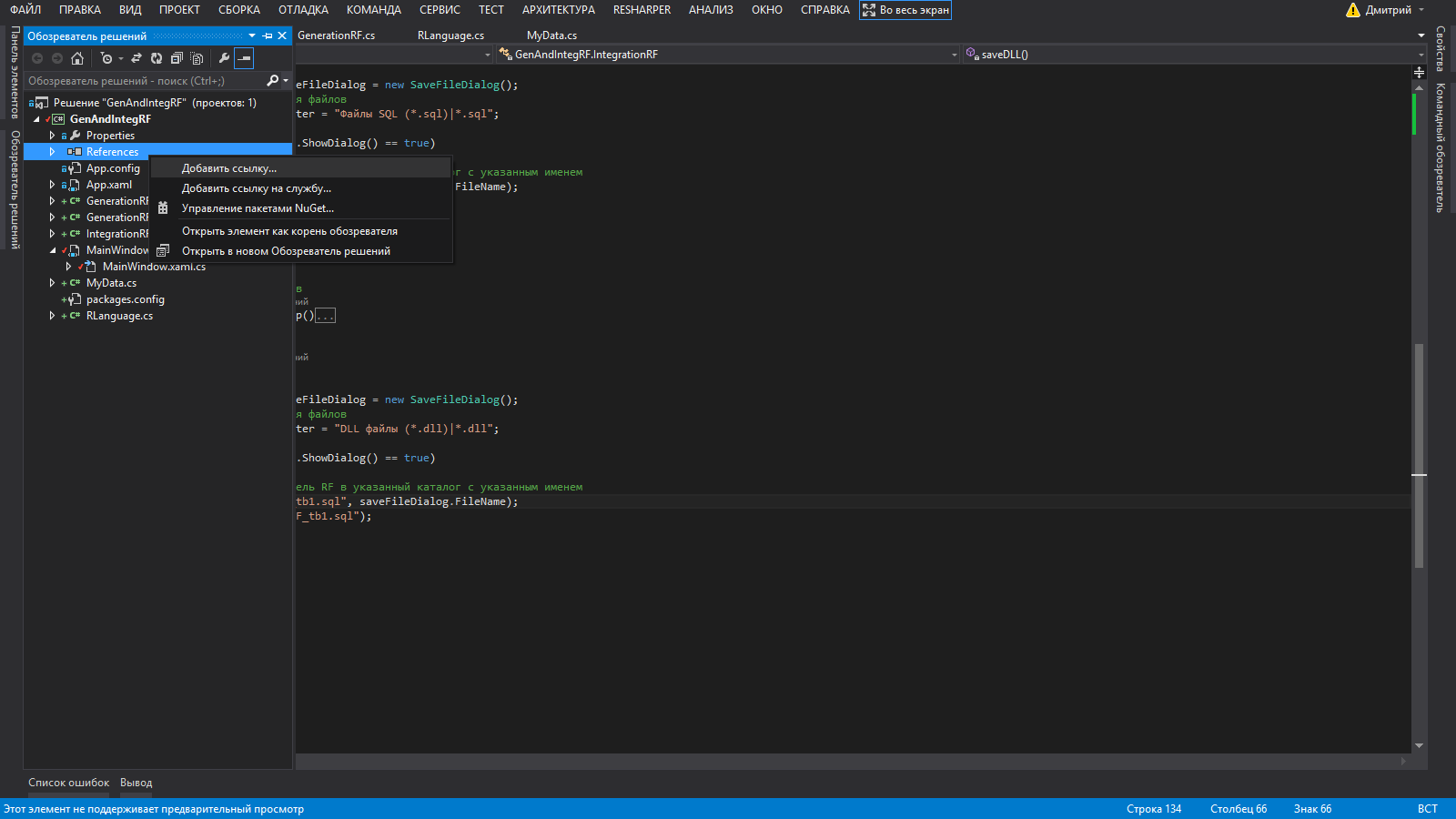


Рис. П4.19. Добавление ссылки

1. В менеджере ссылок, необходимо нажать на кнопку обзор и выбрать на локальном диске полученную библиотеку (рис. П4.20)

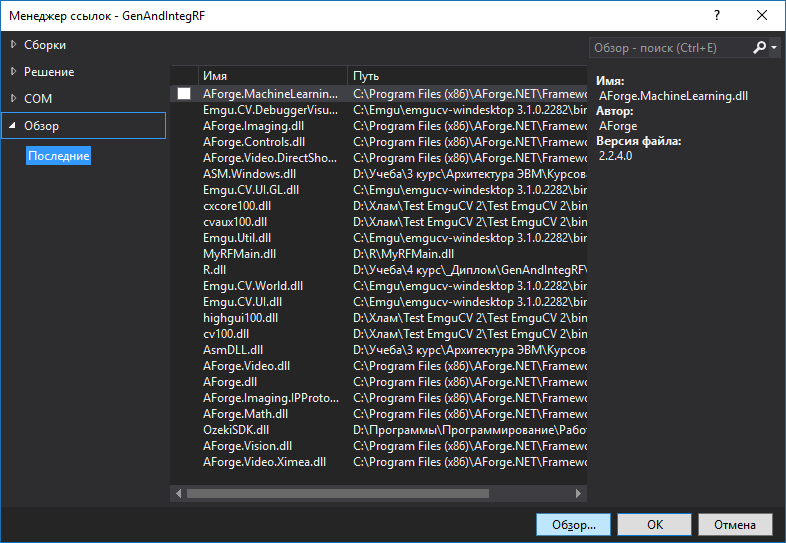


Рис. П4.20. Менеджер ссылок

1. После загрузки библиотеки необходимо убедиться, что она выбрана и нажать ОК (рис. П4.21)

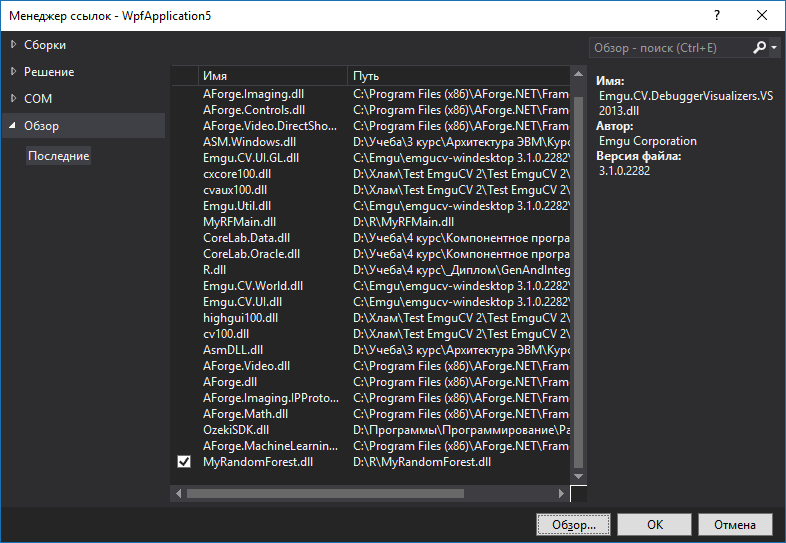


Рис. П4.21. Выбор нужной библиотеки

1. Далее в проекте необходимо подключить пространство имен данной библиотеки:

*using namespace RandomForest.*имя\_библиотеки*;*

1. Создается экземпляр класса. Имя класса совпадает и именем созданной библиотеки
2. Для использования модели необходимо вызвать функцию *CalculateAllTrees.* В качестве параметров передаются параметры, с помощью которых строилась модель Random Forest.

Использование полученного SQL кода в СУБД Oracle

Для использования полученного SQL кода, в СУБД необходимо в обозревателе для загрузки SQL скриптов. Примерный вид окно представлен на рис. П4.22, для более подробной информацией о том как запускать SQL скрипты на определенной версии Oracle следует обратиться к справке СУБД Oracle, которая поставляется вместе с базой данных.

Далее будет рассмотрено применение в программе Oracle SQL Developer.

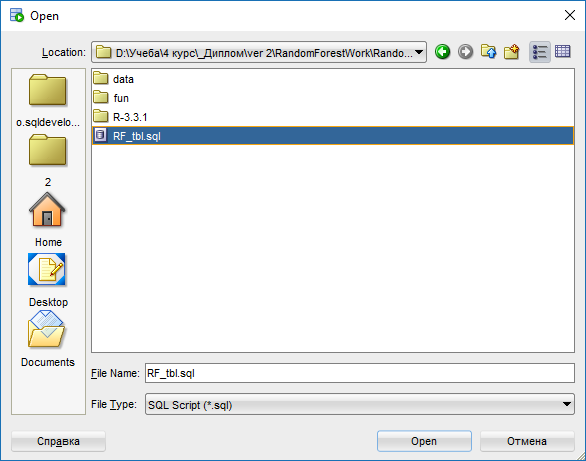


Рис. П4.22. Выбор SQL файла в Oracle SQL Developer

На рис. П4.23 представлено окно с загруженным скритом.

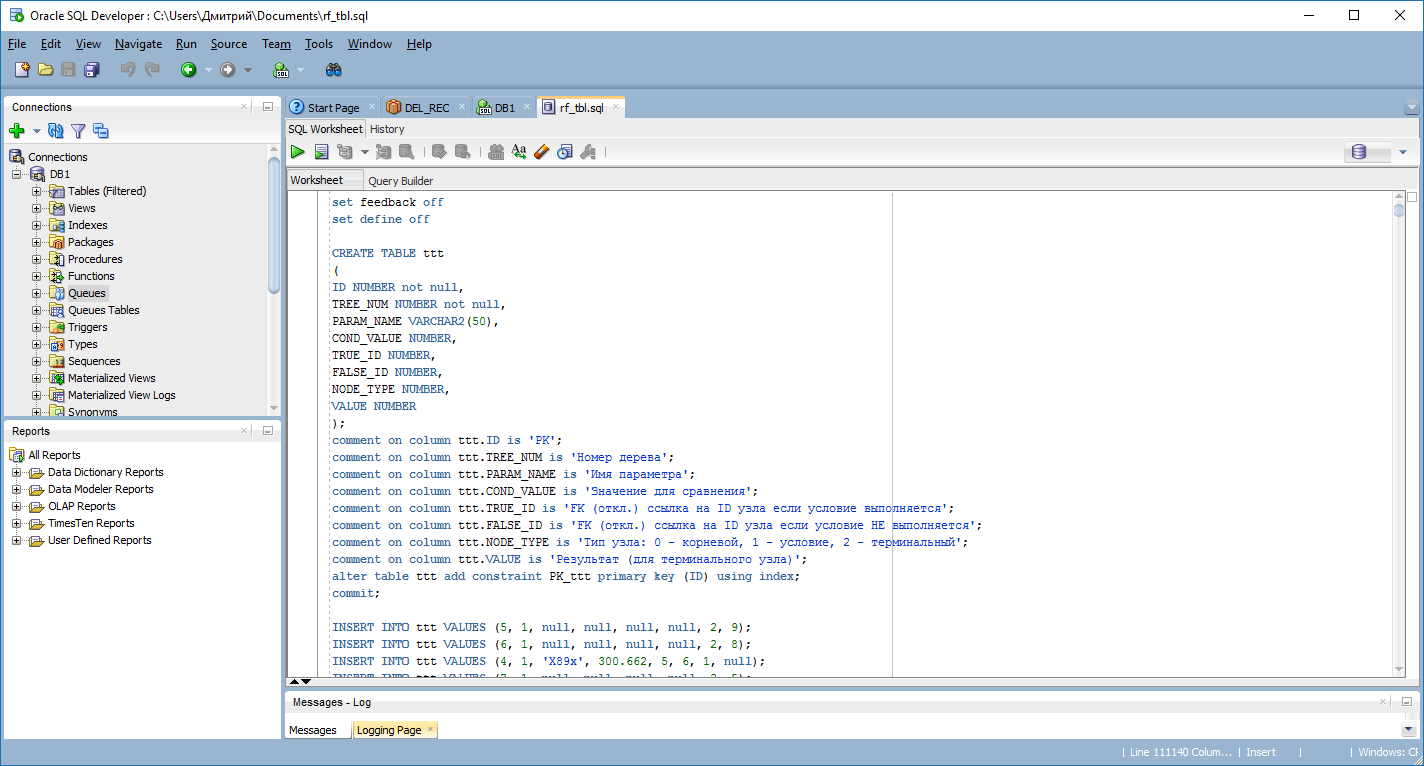


Рис. П4.23. Окно с загруженным скриптом в Oracle SQL Developer

Для его выполнения необходимо нажать кнопку «Run Script», которая расположена в верхней панели приложения.

После выполнения скрипта для добавления таблицы необходимо проделать тоже самое для скрипта с хранимой процедурой.

# Приложение 5

Набор тестовых данных

Набор данных для проведения тестирования представлен в табл. П5.1.

Таблица П5.1

Набор тестовых данных

|  |  |  |
| --- | --- | --- |
| Название теста | Значения | Комментарий |
| 1 | 2 | 3 |
| Загрузка входного файла на этапе просмотра данных | Путь к файлу: \*.csv, \*.xls, \*.xlsx  Пример:  «D:\R\Data\test.csv»  «D:\R\Data\train.xls»  «D:\data.xlsx» | Допустимые данные |
| Путь к файлу: пустая строка, некорректный путь к файлу с правильным расширением, путь к файлу с неправильным расширением (\*.txt, \*.jpg, \*.doc и т.д.)  Пример:  «D:\data.doc» | Недопустимые данные |
| Загрузка входного файла на этапе построения модели Random Forest | Путь к файлу: \*.csv  Пример:  «D:\R\Data\test.csv» | Допустимые данные |
| Путь к файлу: пустая строка, некорректный путь к файлу с правильным расширением, путь к файлу с неправильным расширением (\*.xls, \*.xlsx, \*.txt, \*.jpg, \*.doc и т.д.)  Пример: «D:\R\Data\train.xls» | Недопустимые данные |

Продолжение табл. П5.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Ввод количество деревьев принятий решений | Количество решающих деревьев: 1-1000  Пример:  «1»  «300»  «1000» | Допустимые данные |
| Количество решающих деревьев: меньше 1, больше 1000, текстовое значение  Пример:  «0»  «-300»  «1001»  «100abc» | Недопустимые данные |
| Загрузка списка возможных переменных | Список возможных вариантов: перечисление строковых значений, целочисленное числовое перечисление, комбинация этих двух вариантов  Пример:  «age, job, day, y»  «1, 2, 3, 4, y» | Допустимые данные |
| Список неверных вариантов: пустой список, все переменные в одном значении  Пример:  « »  «agejobdayy» | Недопустимые данные |

Продолжение табл. П5.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Загрузка входного файла на этапе трансляции модели Random Forest | Путь к файлу: \*.RData  Пример:  «D:\R\rf.RData » | Допустимые данные |
| Путь к файлу: пустая строка, некорректный путь к файлу с правильным расширением, путь к файлу с неправильным расширением (\*.xls, \*.xlsx, \*.txt, \*.csv, \*.doc и т.д.)  Пример:  «»  «D:\data.csv»  «D:\R\Data\train.xls» | Недопустимые данные |
| Ввод дополнительной информации на этапе трансляции | Название таблицы, класса, библиотеки: название латинскими буквами без пробелов  Пример:  «tableName»  «class» | Допустимые данные |
| Название таблицы, класса, библиотеки: название кириллицей, название с содержанием цифр, название с содержанием пробелов  Пример:  «table name»  «имя»  «12345»  «%&#!» | Недопустимые данные |

Для системного тестирования, а именно проверки построения и интеграции модели Random Forest использовалась выборка данных из португальского банка. Данная выборка служит для указания будет клиент брать дипозит или нет. Обучающая выборка была построена на звонках в данный банк. Данная задача является бинарной классификацией.

Данная выборка содержит 17 атрибутов: 16 входных и один выходной параметр. Всего имеется 45211 записей в данной выборке.

В табл. П5.2 представлена часть выбрки.

Входные параметры:

1. age – возраст (числовой параметр);
2. job – работа (категориальный параметр);
3. marital – семейное пололжение (категориальный параметр);
4. education – образование (категориальный параметр);
5. default – имеется ли кредит по умолчанию (бинарный параметр);
6. balance – средний годовой доход в евро (числовой параметр);
7. housing – имеется жилищный кредит (бинарный параметр);
8. loan – имеется личный кредит (бинарный параметр);
9. contact – способ связи с клиентом (категориальный параметр);
10. day – день последней связи с клиентом (числовой параметр);
11. month – месяц последней связи с клиентом (категориальный параметр);
12. duration – длительность общения с клиентом в сек (числовой параметр);
13. campaign – количество связей с клиетом (числовой параметр);
14. pdays – количество дней после последней связи (числовой параметр);
15. previous – количество связей в предыдущей компании (числовой параметр);
16. poutcome – предыдущий результат (категориальный параметр).

Выходной параметр:

1. y – клиент согласился на срочный вклад (бинарный параметр).

