

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра _____
вычислительной техники
(полное название кафедры)

Утверждаю

Зав. кафедрой Якименко А. А.

(подпись, инициалы, фамилия)

«____» _____ 201__ г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
по направлению высшего образования

09.04.01 Информатика и вычислительная техника
(код и наименование направления подготовки магистра)

Автоматики и вычислительной техники
(факультет)

Батыгин Роман Игоревич
(фамилия, имя, отчество студента – автора работы)

Программная система классификации разнотипных данных на основе ансамбля алго-
(полное название темы магистерской диссертации)
ритмов

Руководитель
от НГТУ

Автор выпускной
квалификационной работы

Альсова О. К.

(фамилия, имя, отчество)

К.Т.Н. доцент

(ученая степень, ученое звание)

(подпись, дата)

Батыгин Р. И.

(фамилия, имя, отчество)

АВТФ АММ-15

(факультет, группа)

(подпись, дата)

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра _____
вычислительной техники
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Якименко А. А.
(фамилия, имя, отчество)

(подпись, дата)

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

студенту _____
Батыгину Роману Игоревичу
(фамилия, имя, отчество)

факультета _____
Автоматики и вычислительной техники
(полное название факультета)

Направление подготовки 09.04.01 Информатика и вычислительная техника
(код и наименование направления подготовки магистра)

Магистерская программа Прикладные информационные системы и технологии
(наименование магистерской программы)

Тема Программная система классификации разнотипных данных на основе ансамбля алгоритмов
(полное название темы)

Цели работы Исследование и разработка алгоритмического и программного обеспечения системы классификации разнотипных данных на основе ансамбля алгоритмов

Реферат

Пояснительная записка к магистерской диссертации Батыгина Романа Игоревича «Программная система классификации разнотипных данных на основе ансамбля алгоритмов» представлена на 174 страницах и включает 93 рисунка и 41 таблицу. Список литературы содержит 32 библиографических источников.

Ключевые слова

Одиночный (базовый) алгоритм классификации, ансамблевый алгоритм, неоднородный ансамбль, *bagging*, *random forests*, бутстреп–выборка, *boosting*, *stacking*, деревья решений, логистическая регрессия, нейронные сети, метод k -ближайших соседей.

Данная работа посвящена разработке алгоритмического и программного обеспечения системы классификации разнотипных данных на основе ансамбля алгоритмов.

В рамках магистерской диссертация было разработано два оригинальных ансамблевых алгоритма классификации: неоднородный ансамблевый алгоритм и модифицированный неоднородный ансамблевый алгоритм. Была создана действующая программная система *ECA* (*Ensemble Classification Algorithms*), в которой реализованы как одиночные, так и ансамблевые алгоритмы классификации, включая разработанные автором, ансамблевые алгоритмы. Также в *ECA* реализован модуль *Data Miner*, предназначением которого является автоматический подбор оптимальных параметров для ансамблевых алгоритмов на основе проведения серии экспериментов над классификаторами с различными входными параметрами.

В заключительной главе данной работы приведены результаты тестирования разработанных алгоритмов и программных средств на данных из репозитория данных по машинному обучению и реальных медицинских данных.

Содержание

Введение.....	7
1 Аналитический обзор.....	12
1.1 Основные определения и понятия	12
1.2 Логистическая регрессия	13
1.3 Деревья решений	15
1.3.1 Критерии выбора атрибута для расщепления	16
1.3.2 Критерий остановки расщепления	16
1.3.3 Алгоритмы ID3 и C4.5	17
1.3.4 Алгоритм CART	19
1.3.5 Алгоритм CHAID	19
1.4 Алгоритм k – ближайших соседей.....	20
1.5 Нейронные сети	22
1.5.1 Модель нейрона.....	22
1.5.2 Многослойный персептрон	24
1.5.3 Выбор числа нейронов в многослойном персептроне.....	24
1.5.4 Обучение нейронной сети	25
1.6 Ансамбли классификаторов	26
1.6.1 Основные подходы к построению ансамблей	26
1.6.2 Bagging	26
1.6.3 Алгоритм AdaBoost.....	28
1.6.4 Алгоритм Random Forests.....	30
1.6.5 Алгоритм Stacking	30
1.7 Ансамлевые алгоритмы в стандартном статистическом ПО	31
1.7.1 Weka.....	31
1.7.2 Statistica	34
1.8 Выводы по главе	36
2 Разработка алгоритмического обеспечения системы	37
2.1 Алгоритмы вычисления оценок качества классификаторов	37
2.1.1 Методы оценки точности классификации	37
2.1.2 Ошибки классификации	37
2.1.3 ROC - анализ.....	38
2.1.4. Значимые атрибуты.....	41
2.2 Описание неоднородного ансамлевого алгоритма.....	41
2.3 Описание модифицированного неоднородного ансамлевого алгоритма	46

2.4 Выводы по главе	48
3 Разработка программной системы ЕСА	49
3.1 Основные функции системы	49
3.2 Требования к программе.....	51
3.2.1 Требования к входным данным	51
3.2.2 Требования к программному обеспечению	52
3.2.3 Нефункциональные требования.....	53
3.3 Структура системы.....	53
3.4 Реализация программной системы	56
3.4.1 Предобработка входных данных	56
3.4.2 Реализация модуля «деревья решений».....	56
3.4.3 Реализация модуля «нейронные сети».....	61
3.4.4 Реализация алгоритма k – взвешенных ближайших соседей.....	70
3.4.5 Реализация модуля «ансамблевые алгоритмы»	72
3.4.6 Реализация подсистемы построения модели классификатора	85
3.4.7 Реализация подсистем загрузки и сохранения моделей классификатора.....	86
3.4.8 Реализация подсистемы загрузки исходных данных.....	87
3.5 Выводы по главе	92
4 Описание пользовательского интерфейса ЕСА.....	93
4.1 Установка программы	93
4.2 Подготовка данных	95
4.2.1 Загрузка данных из файла	95
4.2.2 Загрузка данных из сети	95
4.2.3 Загрузка данных из базы данных.....	96
4.2.4 Работа с таблицами данных	98
4.2.5 Пропущенные значения.....	102
4.3 Настройка метода оценки точности классификации	102
4.4 Настройка формата чисел	103
4.5 Результаты классификации.....	104
4.5.1 Описание интерфейса системы отображения результатов	104
4.5.2 Описание результатов классификации	108
4.5.3 ROC - анализ.....	108
4.5.4 Классификация нового примера	111
4.6 Индивидуальные алгоритмы. Деревья решений	112
4.6.1 Алгоритм CHAID	112
4.6.2 Визуализация деревьев решений	113

4.7 Индивидуальные алгоритмы. Нейронные сети	116
4.7.1 Настройка нейронной сети.....	116
4.7.2 Визуализация нейронных сетей	118
4.8 Индивидуальные алгоритмы. Алгоритм k – взвешенных ближайших соседей	121
4.9 Индивидуальные алгоритмы. Логистическая регрессия	122
4.9.1 Настройка параметров	122
4.9.2 Значимые атрибуты.....	123
4.10 Ансамблевые алгоритмы	124
4.10.1 Неоднородный ансамблевый алгоритм и его модификация.....	124
4.10.2 Алгоритм «Случайные леса»	128
4.10.3 Алгоритм AdaBoost.....	129
4.10.4 Алгоритм Stacking.....	129
4.10.5 Просмотр структуры ансамбля	131
4.11 Выводы по главе	133
5 Разработка модуля Data Miner	134
5.1 Автоматический подбор параметров для неоднородного ансамблевого алгоритма	134
5.2 Автоматический подбор параметров для нейронной сети	137
5.3 Реализация модуля Data Miner	137
5.4 Описание пользовательского интерфейса модуля Data Miner.....	143
5.4.1 Описание основных частей интерфейса	143
5.4.2 Автоматическое построение нейронных сетей	147
5.4.3 Автоматическое построение ансамблевых алгоритмов	149
5.5 Выводы по главе	150
6 Исследование разработанных алгоритмов и программных средств	151
6.1 Тестирование программной системы ЕСА	151
6.2 Исследование реальных медицинских данных	158
6.2.1 Описание исходных данных	158
6.2.2 План исследования.....	160
6.2.3 Проведение исследования	161
6.3 Анализ результатов исследования	166
6.4 Выводы по главе	169
Заключение.....	170
Список литературы.....	172

Введение

В настоящее время во многих предметных областях находят все более широкое применение информационные технологии, при этом ставятся задачи, связанные с анализом и обработкой все больших объемов накопленной разнотипной информации. Существующие методы и средства анализа данных перестают удовлетворять современным требованиям. В результате возникает необходимость разработки более совершенных методов, технологий и программных средств анализа и обработки разнотипных данных. Одно из современных направлений в области обработки данных связано с применением технологий *Data Mining* и использованием методов машинного обучения. Под *Data Mining* (эквивалент русскоязычного понятия «интеллектуальный анализ данных») понимается процесс исследования больших объемов данных с целью выявления закономерностей и скрытых знаний, которые впоследствии применяются к новым массивам данных [1].

Одной из основных задач *Data Mining* является задача классификации, которая заключается в отнесении объектов к одному из заранее выделенных классов и построении правил отнесения объектов к тому или иному классу [2]. Разработано множество математических методов, позволяющих решать задачу классификации. Наиболее распространеными из них являются: деревья решений, логистическая регрессия, нейронные сети и другие [1]. Однако, по-прежнему, остается актуальной задача повышения качества (точности, устойчивости) классификационных решений. Одним из вариантов решения этой проблемы является разработка и использование ансамблевых алгоритмов классификации.

Ансамбль классификаторов представляет собой множество одиночных классификаторов, решения которых комбинируются некоторым образом для получения окончательного классификационного решения. Согласно литературным источникам [3,4], использование ансамблей классификаторов позволяет повысить точность классификации при решении практических задач.

Подавляющая часть работ в области разработки ансамблей классификаторов посвящена построению однородного ансамбля с использованием различных принципов формирования исходной обучающей выборки (*bagging*, *boosting*, *stacking* и другие) [5,6]. Учитывая, что необходимым и достаточным условием точности ансамбля является различность составляющих его классификаторов (ошибочная классификации разных объектов из исходной выборки) [7,8], разработка неоднородных ансамблевых алгоритмов представляется перспективным направлением.

В современном статистическом программном обеспечении ансамблевые алгоритмы представлены ограничено. Например, в *Statistica 10.0* реализован только один ансамбле-

вый алгоритм классификации *Random Forests* (случайные леса), включающий в качестве одиночных классификаторов модели деревьев решений. В *SPSS* реализован однородный ансамблевый алгоритм, основанный на линейных регрессионных моделях, для решения задачи прогнозирования. Неоднородные ансамблевые алгоритмы в профессиональных статистических пакетах, таких как *Statistica*, *SPSS*, *SAS*, на настоящий момент не реализованы.

Кроме того, профессиональные пакеты сложны в использовании и слабо ориентированы на конечного пользователя (специалиста предметной области).

На основании вышеизложенного, разработка программной системы, реализующей различные ансамблевые алгоритмы (однородные и неоднородные), как классические, так и оригинальные, является актуальной задачей. Программная система, обладающая подобным функционалом, может быть использована как в научно-исследовательской работе при решении задач, основанных на классификации данных, так и в учебном процессе при изучении прикладных математических дисциплин, связанных с анализом данных.

Цели и задачи исследования

Целью работы является исследование и разработка алгоритмического и программного обеспечения системы классификации разнотипных данных на основе ансамбля алгоритмов.

В соответствие с целью можно выделить следующие основные задачи:

1. Обзор методов и средств решения задачи классификации разнотипных данных. Постановка задачи исследования.
2. Разработка ансамблевых алгоритмов классификации разнотипных данных.
3. Разработка программной системы, реализующей классические и оригинальные ансамблевые алгоритмы.
4. Разработка модуля *Data Miner* в рамках программной системы, предназначенного для автоматического подбора оптимальных параметров ансамблевых алгоритмов на основе проведения серии экспериментов.
5. Исследование разработанных алгоритмов и программных средств на основе выборок данных из репозитория данных по машинному обучению* и реальных медицинских данных (данные о хирургическом лечении больных с патологией аорты)**.

*<http://archive.ics.uci.edu/ml/datasets>.

** Данные предоставлены Сибирским федеральным биомедицинским исследовательским центром имени академика Е.Н. Мешалкина.

Методы исследования

Для решения поставленных задач используются следующие методы: методы первичного статистического анализа данных, графического анализа данных; методы решения задач классификации данных (деревья решений, логистическая регрессия, нейронные сети, метод k -ближайших соседей); ансамблевые алгоритмы классификации, методы объектно-ориентированного программирования.

Результаты магистерской диссертации

1. Разработанные оригинальные ансамблевые алгоритмы классификации разнотипных данных:
 - неоднородный ансамблевый алгоритм;
 - модифицированный неоднородный ансамблевый алгоритм.
2. Разработанные оригинальные алгоритмы для автоматического подбора оптимальных параметров для ансамблевых алгоритмов:
 - алгоритм автоматического подбора параметров для неоднородного ансамблевого алгоритма и его модификации;
 - алгоритм автоматического подбора параметров для ансамблевых алгоритмов *AdaBoost* и *Stacking*;
 - алгоритм автоматического подбора параметров для нейронных сетей.
3. Программная система *ECA*, реализующая предложенные алгоритмы.
4. Результаты исследования реальных медицинских данных с использованием разработанных алгоритмов и программных средств.

Теоретическая значимость

Теоретическая значимость работы заключается в решении ряда актуальных проблем в области классификации данных. В данной работе были разработаны два оригинальных ансамблевых алгоритма: неоднородный ансамблевый алгоритм и модифицированный неоднородный ансамблевый алгоритм. Отличительной особенностью этих алгоритмов является итерационное применение одиночных (базовых) классификаторов на исходной обучающей выборке и включение в ансамбль только тех классификаторов, относительная ошибка классификации которых не превосходит заданного порога.

Также в рамках магистерской диссертации были разработаны алгоритмы автоматического подбора оптимальных параметров на основе серии проведенных экспериментов над классификаторами с различными входными параметрами.

Была исследована и подтверждена эффективность применения разработанных ансамблевых алгоритмов и программных средств на данных из репозитория данных по машинному обучению и на реальных медицинских данных.

Практическая значимость

Практическая значимость работы заключается в создании действующей программной системы *ECA* (*Ensemble Classification Algorithms*), которая предназначена для классификации разнотипных данных (количественных, порядковых, номинальных) из любой предметной области на основе как одиночных, так и ансамблевых алгоритмов. В программной системе реализован набор одиночных и ансамблевых алгоритмов классификации, включая оригинальные, разработанные автором, ансамблевые алгоритмы. Также в *ECA* реализован многофункциональный модуль *Data Miner*, предназначенный для автоматического подбора оптимальных параметров для ансамблевых алгоритмов на основе проведения серии экспериментов над классификаторами с различными входными параметрами.

Апробация работы

Результаты работы и ее основные положения докладывались на городской конференции магистрантов и аспирантов *Progress through Innovations* (2016 г.), на XIII-ой международной научно-технической конференции "Актуальные проблемы электронного приборостроения" АПЭП-2016.

Публикации

По теме диссертации опубликовано 5 печатных работ, включая свидетельство о государственной регистрации программной системы *ECA* для ЭВМ:

1. Batygin R. I. Data classification based on the ensemble algorithms / R. I. Batygin ; research adviser O. K. Alsova ; language adviser N. N. Shergina //Progress through Innovations : тез. науч.-практ. конф. аспирантов и магистрантов, Новосибирск, 31 марта 2016 г. – Новосибирск : Изд-во НГТУ, 2016. – С. 13–14. - 160 copy. - ISBN 978-5-7782-2869-6.
2. Батыгин Р. И. Программная система классификации разнотипных медицинских данных на основе ансамбля алгоритмов / Р. И. Батыгин //Электротехнические комплексы и системы: тез. 54-й международной научной студенческой конференции, 16-20 апреля 2016 г. - Новосибирск :Изд-во НГТУ, 2016. – С. 66. - 155 copy. - ISBN 978-5-4437-0514-9.
3. Batygin R. I. Software system for different types of data classification based on the ensemble algorithms / R. I. Batygin, O. K. Alsova // Актуальные проблемы электронного приборостроения (АПЭП–2016) = Actual problems of electronic instrument engineering (APEIE–2016) : тр. 13 междунар. науч.-техн. конф., Новосибирск, 3–6 окт. 2016 г. : в 12 т. – Новосибирск : Изд-во НГТУ, 2016. – Т. 1, ч. 2. – С. 506-509. - 60 экз. - ISBN 978-5-7782-2991-4.

4. Батыгин Р. И. Программная система классификации разнотипных данных на основе ансамбля алгоритмов = Software system for different types of data classification based on the ensemble algorithms / Р. И. Батыгин, О. К. Альсова // Актуальные проблемы электронного приборостроения (АПЭП–2016) =Actual problems of electronic instrument engineering (APEIE–2016) : тр. 13 междунар. науч.-техн. конф., Новосибирск, 3–6 окт. 2016 г. : в 12 т. – Новосибирск : Изд-во НГТУ, 2016. – Т. 9.- С.117-121. - 40 экз. - ISBN 978-5-7782-2991-4.
5. Батыгин Р.И., Альсова О.К. Программная система классификации разнотипных данных на основе ансамбля алгоритмов (ECA - Ensemble Classification Algorithms): Свидетельство о государственной регистрации программы для ЭВМ № 2017610788. 2017.

Структура и объем диссертации

Данная работа представлена на 174 страницах и состоит из введения, заключения, 6 глав, каждая из которых завершается выводами. Работа включает 93 рисунка и 41 таблицу, список литературы содержит 32 источника.

В первой главе проведен аналитический обзор источников по исследуемой теме, обоснована актуальность выбранной темы, поставлена задача исследования.

Во второй главе проведено описание разработанных оригинальных разработанных ансамблевых алгоритмов классификации: неоднородного ансамблевого алгоритма и модифицированного неоднородного ансамблевого алгоритма.

В третьей и четвертой главе описаны все этапы проектирования, разработки и реализации программной системы *ECA*, реализующей разработанные алгоритмы. Приведена ее структура и основные функции. Также приведено подробное описание пользовательского интерфейса системы.

В пятой главе приведено описание разработанного модуля *Data Miner*. Приводится описание разработанных алгоритмов для автоматического подбора оптимальных параметров для ансамблевых алгоритмов и нейронных сетей. Рассматривается реализация этого модуля в программной системе *ECA*.

В шестой главе приводится исследование разработанных алгоритмов и программных средств на основе выборок данных из репозитория данных по машинному обучению и реальных медицинских данных. Также в этой главе содержится описание медицинских данных, плана проведения исследования, хода исследования и его результатов с последующей интерпретацией.

1 Аналитический обзор

1.1 Основные определения и понятия

Задача классификации заключается в отнесении объектов к одному из заранее выделенных классов. Математически задача классификации формулируется следующим образом. Пусть имеется множество классов $Y = \{y_1, \dots, y_k\}$, $k \geq 2$ и множество объектов $X = \{x_i\}, i = \overline{1, N}$, которые необходимо классифицировать. Для каждого объекта x_i измерено m признаков $x_i = \{x_{i1}, \dots, x_{im}\}$. Классификатором называется отображение следующего вида [9]:

$$C : x_i \rightarrow [0,1]^k, \quad (1.1)$$

где $C(x_i)$ – k – мерный вектор, у которого j -ый компонент определяет степень принадлежность объекта x_i к классу y_j , $j = 1, \dots, k$.

Ниже приведены основные определения и термины, используемые в данной работе.

Под *объектом* понимается элемент множества X . Под *атрибутом* или *признаком* понимается независимая переменная, описывающая свойство объекта.

Классом называется зависимая переменная, которая принимает любое значение из множества Y .

Все признаки делятся на *качественные (числовые)* и *качественные (категориальные)*, при этом качественные признаки делятся на *порядковые* и *номинальные*. Для количественных признаков применимы любые арифметические операции, в то время как для качественных переменных применимы далеко не все операции. Для номинальных признаков применимы лишь операции «равно» и «не равно». Для порядковых признаков, помимо операций «равно» и «не равно», также применима операция сравнения.

Под *обучающей выборкой* понимается множество $X = \{x_i\}, i = \overline{1, N}$, которое используется для построения модели классификатора.

Под *тестовой выборкой* понимается выборка, которая используется для оценки точности построенной модели классификатора.

Точность классификатора определяется как доля правильно классифицированных объектов

$$p = \frac{n}{N} \cdot 100 \%, \quad (1.2)$$

где n – число верно классифицированных объектов, N – общее число объектов.

Обычно, при построении классификатора, исходная выборка разбивается на две части – обучающую и тестовую. Модель классификатора строится по обучающей выборке, а точность классификации проверяется на тестовой выборке.

Ансамбль классификаторов представляет собой множество классификаторов, чьи решения комбинируются некоторым образом для получения окончательной классификации данных.

Однородный ансамбль представляет собой классификатор, состоящий из моделей классификаторов одного типа, например, деревьев решений.

Неоднородный ансамбль представляет собой классификатор, состоящий из моделей классификаторов различных типов, например, деревьев решений, логистической регрессии, нейронной сети и т.д.

Существует множество математических методов, позволяющих решить задачу классификации. Наиболее известными из них являются деревья решений, логистическая регрессия и нейронные сети, подробное описание которых приведено в следующих разделах.

1.2 Логистическая регрессия

Логистическая регрессия – это разновидность множественной регрессии, общее назначение которой состоит в анализе связи между несколькими независимыми переменными и зависимой переменной [10]. Особенностью логистической регрессии является то, что зависимая переменная Y является качественной. С помощью логистической регрессии оценивается вероятность принадлежности наблюдения x к одному из значений зависимой переменной Y .

Частным случаем логистической регрессии является бинарная логистическая регрессия, в которой зависимая переменная y может принимать два значения. Общий вид модели регрессионного уравнения имеет вид:

$$y = f(x_1, \dots, x_n). \quad (1.3)$$

Для множественной линейной регрессии модель имеет вид:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \varepsilon, \quad (1.4)$$

где ε - случайная составляющая.

Уравнение логистической регрессии имеет следующий вид:

$$P(x) = \frac{e^y}{1 + e^y}, \quad (1.5)$$

где $P(x)$ – вероятность того, что произойдет интересующее нас событие. Диапазон изменения $P(x)$ будет от 0 до 1.

Для категориальной зависимой переменной, имеющей более двух категорий, применяется мультиномиальная логистическая регрессия. В модели мультиномиальной логистической регрессии для каждой категории зависимой переменной строится уравнение бинарной логистической регрессии. При этом, одна из категорий зависимой переменной становится опорной, а все другие категории сравниваются с ней. Уравнение мультиномиальной логистической регрессии прогнозирует вероятность принадлежности к каждой категории зависимой переменной по значениям независимых переменных [11].

Существует несколько способов нахождения коэффициентов логистической регрессии. На практике часто используют метод максимального правдоподобия. Общий вид функции логарифмического правдоподобия для оценки коэффициентов логистической регрессии имеет вид [12]:

$$L(\beta) = - \sum_{i=1}^n \left(\sum_{j=1}^{J-1} y_{ij} \ln \rho_j(x_i) + (1 - \sum_{j=1}^{J-1} y_{ij}) \ln (1 - \sum_{j=1}^{J-1} \rho_j(x_i)) \right), \quad (1.6)$$

где n – число примеров обучающей выборки, J – число классов, y_{ij} – значения матрицы Y размера $n \times (J-1)$, в которой содержатся бинарные значения выходного атрибута класса, причем $y_{ij} = 1$, если объект x_i принадлежит к классу с номером j , в противном случае $y_{ij} = 0$. β – вектор значений коэффициентов, $\rho_j(x_i)$ – вероятность класса j для примера x_i , которая вычисляется по формуле [12]:

$$\rho_j(x_i) = \frac{e^{\sum_{k=0}^K x_{ik} \beta_{kj}}}{1 + \sum_{j=1}^{J-1} e^{\sum_{k=0}^K x_{ik} \beta_{kj}}}, \quad j < J. \quad (1.7)$$

И для последнего значения класса:

$$\rho_J(x_i) = \frac{1}{1 + \sum_{j=1}^{J-1} e^{\sum_{k=0}^K x_{ik} \beta_{kj}}}, \quad (1.8)$$

где K – число входных атрибутов.

1.3 Деревья решений

Деревья решений являются одним из часто используемых методов решения задачи классификации. Деревья решений представляют собой древовидные структуры, состоящие из правил «если ... то ...», и позволяют выполнять классификацию объектов [13]. Деревья решений состоят из двух видов объектов – узлов и листьев. Узлы представляют собой объекты, содержащие правила перехода в дочерние узлы. Листья не имеют дочерних узлов и определяют один из классов c_j и множество объектов, принадлежащих к этому классу. На рисунке 1.1 изображен пример структуры дерева решений.

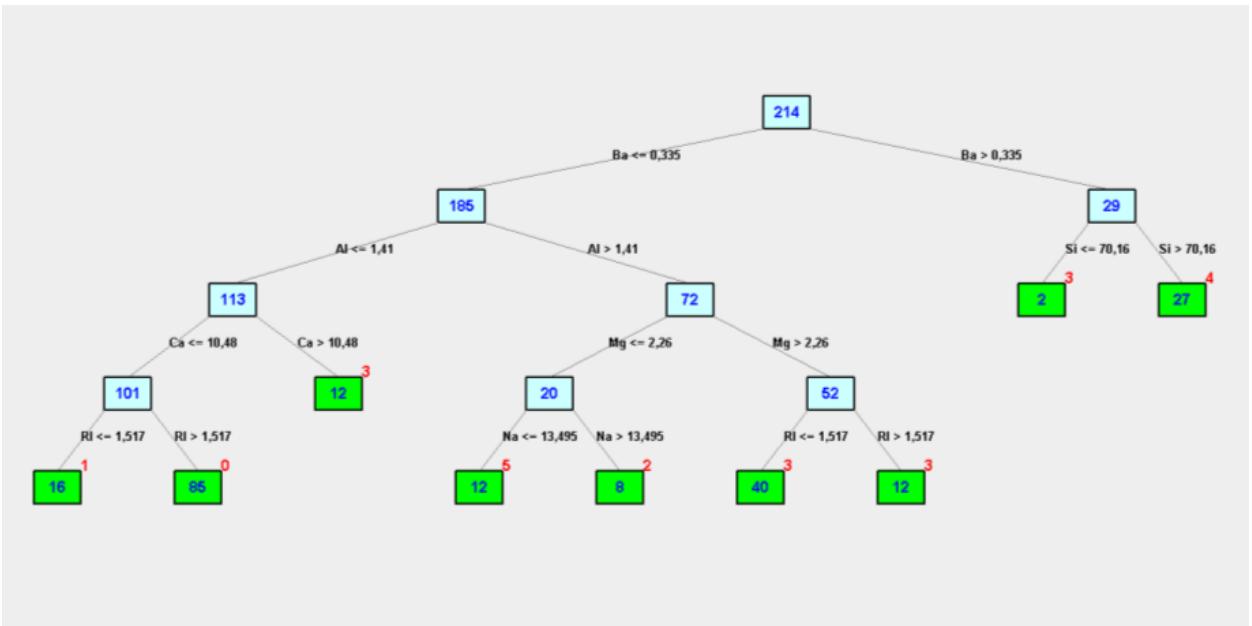


Рисунок 1.1 – Пример дерева решений

При использовании метода деревьев решений для решения задачи классификации зависимая переменная должна иметь качественный тип.

Атрибутом разбиения называют атрибут, по которому происходит проверка правила для дальнейшего ветвления. Метод, на основе которого выбирается атрибут для дальнейшего ветвления, называют *алгоритмом построения дерева решений*.

Полным деревом называется дерево решений, в котором листья являются чистыми, т.е. содержат объекты одного класса.

Деревья решений строятся на основе принципа «разделяй и властвуй», который представляет собой рекурсивную процедуру. Пусть дано множество T , в котором определены классы $Y = \{y_1, \dots, y_k\}$, $k \geq 2$. Тогда существуют три возможных варианта разбиения [13].

1. Множество T содержит два или более примера, принадлежащие одному классу, тогда дерево решений для T – это лист, определяющий класс y_i .

2. Множество T является пустым, тогда дерево решений для T – это лист, но класс, ассоциированный с листом, выбирается как наиболее часто встречающийся класс из родительского множества.
3. Множество T содержит примеры, относящиеся к разным классам. Тогда выбирается атрибут, имеющий m отличных друг от друга значений x_1, \dots, x_m , и затем исходное множество T разбивается на подмножества $\{T_1(x_1), \dots, T_m(x_m)\}$, где каждое подмножество T_i содержит примеры из множества T , в которых выбранный атрибут имеет значение x_i .

1.3.1 Критерии выбора атрибута для расщепления

Самым важным вопросом при построении дерева решений является выбор атрибута для дальнейшего ветвления (расщепления). Существует четыре основных критерия [13]:

- индекс Джини;
- критерий, основанный на энтропии;
- критерий *Gain ratio*;
- тест хи – квадрат.

Все эти критерии основаны на повышении чистоты классификации.

1.3.2 Критерий остановки расщепления

Следующим вопросом при построении дерева решений является остановка расщепления. Известно, что полное дерево решений обладает низкой точностью и описывает только обучающую выборку [13]. Существует несколько критериев остановки расщепления.

1. Минимальное число объектов n в листьях. В [1] рекомендуют задавать $n = 2$. Каждое новое подмножество при разбиении исходного множества должно содержать не менее n объектов. При невыполнении этого условия дальнейшее расщепление прекращается, и исходная вершина помечается как лист.
2. Доля неклассифицированных объектов. При выборе этого варианта ветвления продолжаются до тех пор, пока все листья не окажутся чистыми или будут содержать количество объектов, не превышающее заданную долю численности одного или нескольких классов [14].

3. Задание верхнего порога для критерия выбора атрибута расщепления. Задается некоторое значение a , и дальнейшее расщепление не происходит, если значение критерия выбора атрибута расщепления не превышает a [8].
4. Кросс проверка, подробное описание которой изложено в [14].

При использовании этих подходов возникает проблема, при которой лист содержит объекты, принадлежащие различным классам. Тогда в качестве класса ассоциированного с листом выбирается наиболее часто встречающийся класс. Если же примеров равное количество, то выбирается класс, наиболее часто встречающийся у предка листа [1].

1.3.3 Алгоритмы ID3 и C4.5

Наиболее популярными алгоритмами построения деревьев решений являются алгоритмы *ID3* и *C4.5*. Оба алгоритма строятся по принципу «разделяй и властвуй» изложеному в предыдущем разделе. Выбор атрибута для дальнейшего ветвления в алгоритме *ID3* определяется на основе энтропии разбиения, которая определяется как [13]:

$$Gain(S) = Info(T) - Info_s(T), \quad (1.9)$$

где $Info(T)$ – энтропия множества T , ассоциированного с узлом, $Info_s(T)$ – энтропия разбиения S узла. Энтропия множества T определяется по формуле [13]:

$$Info(T) = -\sum_{i=1}^k p_i \log_2 p_i, \quad (1.10)$$

где p_i – вероятность класса i в множестве T , которая вычисляется как

$$p_i = \frac{N_i}{N}, \quad (1.11)$$

где N_i – количество объектов, принадлежащих классу i , N – общее количество объектов в T .

Для разбиения S , в результате которого было образовано m потомков T_1, \dots, T_m энтропия разбиения S вычисляется по формуле [13]:

$$Info_S(T) = \sum_{i=1}^m \frac{N_i}{N} Info(T_i), \quad (1.12)$$

где – N_i - количество примеров в дочернем узле T_i .

Из свойств энтропии следует, что максимальное значение она принимает в случае, когда все классы во множестве T равновероятны. Поэтому необходимо минимизировать величину $Info_S(T)$ для того, чтобы в дочерних узлах находилось больше примеров, принадлежащих к одному классу.

Таким образом, атрибут, для которого значение $Gain(S)$ максимально выбирается для дальнейшего ветвления.

Рассмотренные формулы применимы для качественных признаков. Для количественных признаков возникает проблема выбора порогового значения z для правила ветвления. Пороговое значение выбирается по следующему алгоритму.

1. Сначала необходимо отсортировать по не убыванию обучающие примеры по выбранному количественному признаку A . Обозначим отсортированное множество как T' .
2. Из полученного множества T' выбрать медиану z этого множества, которая будет выступать в роли порогового значения для построения правила разбиения.
3. Затем для каждого значения $z_j = \frac{A_i + A_{i+1}}{2}, i = \overline{1, N-1}$ (N – число объектов во множестве T') разбить исходные обучающие примеры на два подмножества. Первое подмножество будет содержать примеры, для которых $A_i \leq z$, а второе примеры для которых $A_i > z$.
4. В конце для каждого z_j вычислить значение критерия $Gain(S)$ и из них выбрать максимальное.

Основным недостатком алгоритма *ID3* является проблема «переобучения», которая заключается в том, что непосредственное использование критерия (1.9) приводит к выбору атрибута для разбиения, имеющего большое количество значений. В результате получаются сильно ветвящиеся деревья. С точки зрения классифицирующей такой модель бесполезна. Алгоритм *C4.5* лишен этого недостатка введением нормировки, которая вычисляется по следующей формуле [13]:

$$GainRatio(S) = \frac{Gain(S)}{SplitInfo(S)}, \quad (1.13)$$

где $Split Info(S)$ вычисляется по формуле [8]:

$$SplitInfo(S) = -\sum_{i=1}^m \frac{N_i}{N} \log_2 \frac{N_i}{N}. \quad (1.14)$$

где N_i – число объектов, отнесенных в результате расщепления в i -ое подмножество. Критерий $GainRatio(S)$ устраняет проблему выбора атрибута с большим количеством значений. В алгоритме *C4.5* выбирается атрибут для ветвления, имеющий максимальное значение $GainRatio(S)$.

1.3.4 Алгоритм CART

Алгоритм *CART* строит бинарные деревья решений, т.е. деревья в котором узлы имеют двух потомков. Обычно этот алгоритм применяется для двухуровневых качественных переменных.

Алгоритм *CART* строит бинарные деревья решений. В нем выбор атрибута для расщепления определяется с помощью индекса Джини, который определяет вероятность того, что два случайным образом выбранных объекта не принадлежат к одному классу и определяется по формуле [13]:

$$Gini(T) = 1 - \sum_{i=1}^k p_i^2. \quad (1.15)$$

Для разбиения S , индекс Джини вычисляется по формуле [4]:

$$Gini_S(T) = \sum_{i=1}^m \frac{N_i}{N} Gini(T_i), \quad (1.16)$$

где N_i - количество примеров в дочернем узле T_i . Наилучшим считается то разбиение S , при котором значение $Gini_S(T)$ минимально.

Для количественных признаков применяется тот же метод выбора порогового значения z для правила ветвления, что и в алгоритмах *ID3* и *C4.5*. Алгоритм *CART* также применяется для качественных признаков, имеющих более 2-х значений. Для этого при расщеплении вершины по качественному признаку, необходимо разбить множество значений этого признака на два подмножества. Таким образом, для категориального атрибута, имеющего k значений, имеется ровно $2^{(k-1)} - 1$ вариантов разбиения множества его значений на две части.

1.3.5 Алгоритм CHAID

В алгоритме *CHAID* (*Chi-square Automatic Interaction Detector*) в каждом узле выбор атрибута для расщепления определяется с помощью критерия хи – квадрат, который обнаруживает статистические связи между переменными и вычисляется по формуле [8]:

$$\chi_p^2 = \sum_{i=1}^C \sum_{j=1}^K \frac{(n_{ij} - n_{ij}^*)^2}{n_{ij}^*}, \quad (1.17)$$

где n_{ij} – количество примеров класса i , отнесенных в результате расщепления в j -ю подвыборку, n_{ij}^* - ожидаемое количество примеров класса i в j -ой подвыборке, C – число классов, K – число значений категориального атрибута, для числового атрибута оно равно двум. Если на очередной итерации максимальное значение (по всем возможным расщеп-

лением) $\chi_p^2 < \chi_\alpha^2(C-1)(K-1)$ меньше критического значения при заданном уровне значимости α , то расщепления не происходит и вершина объявляется как лист [8].

1.4 Алгоритм k – ближайших соседей

Метод k - ближайших соседей является метрическим алгоритмом классификации. Основной принцип метода ближайших соседей основан на том, что объекту присваивается класс, наиболее часто встречающийся среди соседей данного элемента.

Алгоритм ближайших соседей использует меру сходства между объектами (функцию расстояния), которая может быть одной из следующих.

1. Евклидово расстояние:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^K (x_{ik} - x_{jk})^2}. \quad (1.18)$$

2. Квадрат Евклидова расстояния:

$$d(x_i, x_j) = \sum_{k=1}^K (x_{ik} - x_{jk})^2. \quad (1.19)$$

3. Манхэттенское расстояние:

$$d(x_i, x_j) = \sum_{k=1}^K |x_{ik} - x_{jk}|. \quad (1.20)$$

4. Расстояние Чебышева:

$$d(x_i, x_j) = \max_{1 \leq k \leq K} |x_{ik} - x_{jk}|. \quad (1.21)$$

где K – число входных атрибутов (признаков).

Существует несколько разновидностей алгоритма k – взвешенных ближайших соседей [16]:

1. *Метод ближайшего соседа.* Классифицируемый объект x относится к тому классу y_i , которому принадлежит ближайший объект обучающей выборки X .
2. *Метод k – ближайших соседей.* Классифицируемый объект x относится к тому классу y_i , которому принадлежит k – ближайших соседей обучающей выборки X .
3. *Метод k – взвешенных ближайших соседей.* В этом алгоритме i -му соседу присваивается вес ω_i , как правило, убывающий с ростом ранга соседа i . Классифицируемый объект относится к тому классу, который набирает максимальный суммарный вес среди k - ближайших соседей.

В общем случае алгоритм k – ближайших соседей состоит из последовательности следующих шагов:

Сначала для произвольного объекта x все объекты обучающей выборки располагаются в порядке возрастания расстояний до x , $d(x, x_1) \leq d(x, x_2) \leq \dots \leq d(x, x_n)$. Затем выполняется итоговая классификация объекта x по формуле [16]:

$$c(x) = \arg \max_{y \in Y} \sum_{i=1}^m [X_{i;x} = y] \omega(i, x), \quad (1.22)$$

где $X_{i;x}$ – объект обучающей выборки, который является i -ым соседом объекта x . $\omega(i, x)$ – весовая функция ближайшего соседа, которая может быть одной из следующих:

1. $\omega(i, x) = [i = 1]$ - метод ближайшего соседа;
2. $\omega(i, x) = [i \leq k]$ - метод k – ближайших соседей;
3. $\omega(i, x) = [i \leq k]q^i$ - метод k – взвешенных ближайших соседей. Здесь $0.5 \leq q \leq 1$.

В методе ближайших соседей существует проблема, когда различные атрибуты могут иметь разный диапазон представленных значений в выборке (к примеру, атрибут А представлен в диапазоне от 0 до 1, а атрибут В представлен в диапазоне от 10000 до 50000). При этом значения расстояний между объектами могут сильно зависеть от атрибутов с большими диапазонами. Поэтому данные обычно подлежат нормализации. Наиболее распространенным методом нормализации является *минимаксная нормализация*, в которой нормализованное значение для произвольного атрибута V вычисляется по формуле [17]:

$$V'(i) = \frac{V(i) - \min(V(i))}{\max(V(i)) - \min(V(i))}, \quad (1.23)$$

где $V(i)$ – значение i -го объекта для атрибута V , $\max(V(i))$ и $\min(V(i))$ – соответственно максимальное и минимальное значения для атрибута V .

1.5 Нейронные сети

1.5.1 Модель нейрона

Искусственная нейронная сеть представляет собой параллельно – распределенную систему процессорных элементов (нейронов), которые соединены между собой связями, и способны выполнять простейшую обработку данных, которая может настраивать свои параметры в ходе обучения на эмпирических данных [13]. Нейронные сети являются мощным методом моделирования и широко используются для решения задачи классификации.

Нейрон представляет собой единицу обработки информации, на основе которой строятся нейронные сети. Общая модель нейрона состоит из следующих частей [13].

1. Множество входных связей (синапсов) x_i , каждая из которых имеет числовой вес ω_i .
2. Сумматор для вычисления взвешенной суммы входных сигналов x_i .
3. Активационная функция $f(S)$, которая выполняет преобразование выходного значения сумматора.

В математическом представлении функционирование нейрона можно описать следующей парой уравнений:

$$S = \sum_{i=1}^n \omega_i x_i + \omega_0, \quad (1.24)$$

$$y = f(S). \quad (1.25)$$

где S – взвешенная сумма входных сигналов x_i , ω_0 - значение порога активации, y – выходное значение нейрона. На рисунке 1.2 приведена обобщенная модель нейрона

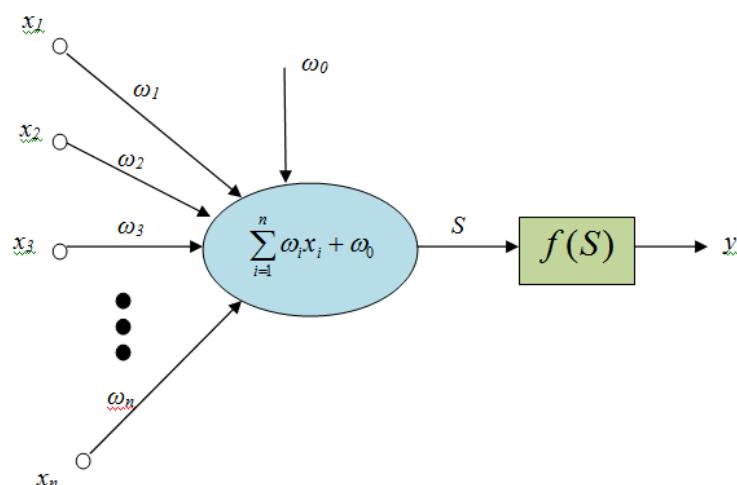


Рисунок 1.2 – Обобщенная модель нейрона

Существует три группы нейронов:

1. *Входные нейроны.* Они образуют входной слой сети. Их задача принять компоненты вектора входного сигнала $X = \{x_1, x_2, \dots, x_n\}^T$ и распределить их по другим нейронам сети.
2. *Выходные нейроны.* Они предоставляют результаты обработки нейронной сетью входного вектора X в виде выходного вектора $Y = \{y_1, y_2, \dots, y_m\}^T$. Таким образом, выходные нейроны образуют выходной слой сети.
3. *Скрытые нейроны.* Образуют скрытые слои сети и выполняют обработку данных в нейронной сети.

Очень важную роль в работе нейронной сети играет выбор активационной функции нейронов. Ниже приведены основные активационные функции нейронов, которые используются в данной работе.

1. Логистическая

$$f(S) = \frac{1}{1 + e^{-aS}}, a > 0. \quad (1.26)$$

2. Гиперболический тангенс

$$f(S) = \frac{e^{aS} - e^{-aS}}{e^{aS} + e^{-aS}}, a > 0. \quad (1.27)$$

3. Тригонометрический синус

$$f(S) = \sin aS, a > 0. \quad (1.28)$$

4. Экспоненциальная

$$f(S) = e^{\frac{S^2}{\sigma^2}}, \sigma > 0. \quad (1.29)$$

где S – взвешенная сумма входных сигналов, a и σ - соответствующие коэффициенты.

1.5.2 Многослойный персептрон

Существует множество различных архитектур нейронных сетей. Наиболее широко используемой из них является нейронная сеть типа многослойного персептрана (*multilayer perceptron, MLP*), которая является базовой для решения задачи классификации. На рисунке 1.3 приведена структура многослойного персептрана с двумя скрытыми слоями

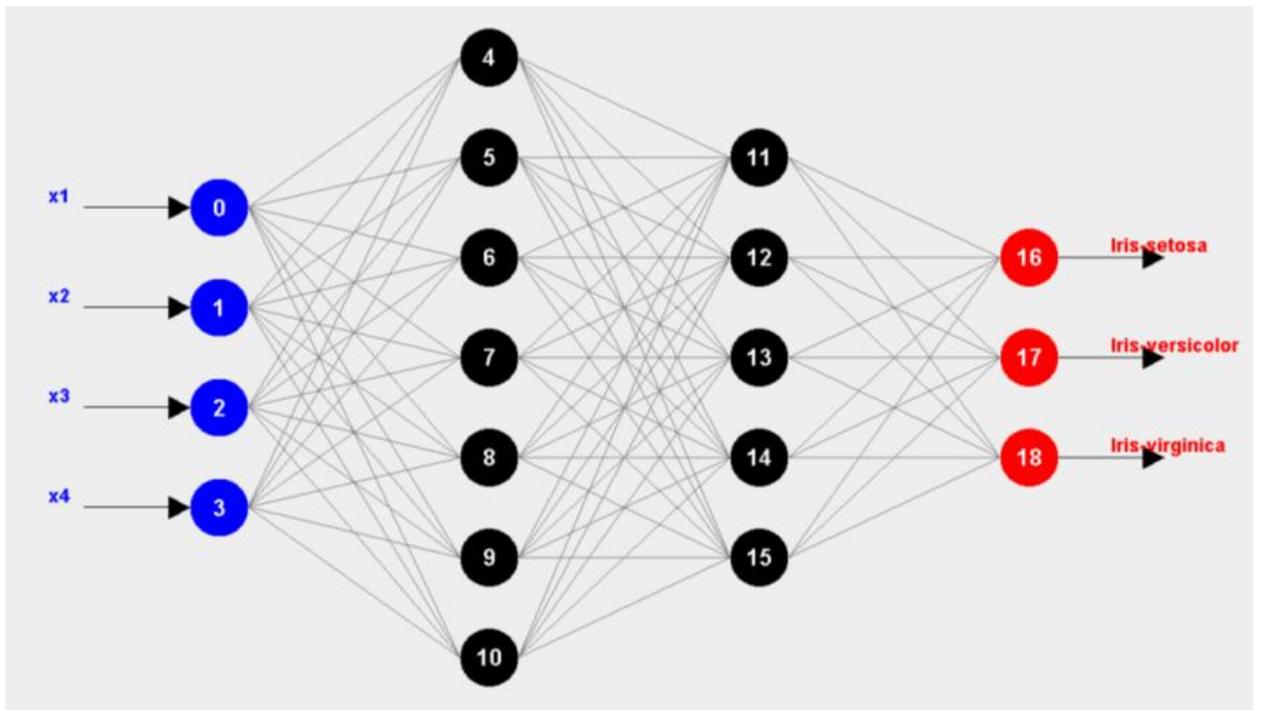


Рисунок 1.3 – Пример структуры многослойного персептрана

1.5.3 Выбор числа нейронов в многослойном персептроне

При выборе структуры нейронной сети очень важным моментом является выбор в ней оптимального числа нейронов. Для решения задачи классификации они задаются следующим образом: число нейронов входного слоя устанавливается равным числу входных атрибутов обучающего множества X , число выходных нейронов устанавливается равным числу классов. Число нейронов в скрытом слое вычисляется с помощью следующей формулы [18]:

$$\frac{W_{\min}}{N_x + N_y} \leq N = \frac{N_w}{N_x + N_y} \leq \frac{W_{\max}}{N_x + N_y}, \quad (1.30)$$

где N_x – число входов сети, N_y – число выходов сети, N_w – необходимое число синаптических весов, которое должно лежать в интервале:

$$W_{\min} = \frac{N_y N_p}{1 + \log_2 N_p} \leq N_w \leq N_y \left(\frac{N_p}{N_x} + 1 \right) (N_x + N_y + 1) + N_y = W_{\max}, \quad (1.31)$$

где N_p – число обучающих примеров.

1.5.4 Обучение нейронной сети

Для обучения нейронной сети типа многослойного персептрона обычно используется алгоритм обратного распространения ошибки [13], при этом минимизируется следующая целевая функция:

$$E(\omega) = \frac{1}{2} \sum_{i=1}^k (y_i - d_i)^2, \quad (1.32)$$

где y_i – прогнозные значения, а d_i – реальные, k – число выходов сети, ω - вектор весов связей между нейронами.

В общем случае алгоритм обучения нейронной сети можно представить в виде последовательности следующих шагов:

1. Инициализация всех весов нейронной сети начальными значениями. Обычно, случайными числами с равномерным законом распределения, лежащими в интервале от [-0.5, 0.5].
2. В цикле от $i = 1, \dots, t$, где t – число итераций для обучения. Сначала на вход сети подаем обучающий пример x_i и вычисляем выходной вектор y .
3. Затем вычисляется ошибка сети, допущенная на данном примере по формуле (1.32).
4. Если ошибка $E(\omega)$ меньше некоторого достаточно малого числа ε или $i > t$, то обучение завершается. В противном случае вычисляется коррекция всех весов нейронной сети, например с помощью *алгоритма обратного распространения ошибки*.

В заключение также отметим, что одной из особенностей функционирования нейронных сетей является необходимость в нормализации входных значений (как числовых, так и категориальных). Для этого обычно используется *минимаксная нормализация*. При этом, значения категориальных атрибутов должны быть преобразованы в уникальные числовые коды.

1.6 Ансамбли классификаторов

1.6.1 Основные подходы к построению ансамблей

Целью создания ансамбля моделей является повышение точности классификации агрегированного классификатора по сравнению с точностью классификации каждого индивидуального базового классификатора [19]. Также в работе [19] обосновано, что при построении ансамбля моделей происходит уменьшение дисперсии ошибки классификации.

Существует два основных подхода к построению ансамбля классификаторов:

- *однородный ансамбль*, состоящий из моделей классификаторов одного типа, например, деревьев решений;
- *неоднородный ансамбль*, состоящий из моделей классификаторов различных типов, например, деревьев решений, логистической регрессии, нейронной сети и т.д.

Следующим вопросом при построении ансамбля является выбор подхода для манипулирования обучающим множеством с последующим построением моделей классификаторов. Здесь существует три основных подхода:

- *bagging*;
- *boosting*;
- *stacking*.

Также очень важным вопросом является выбор метода для комбинирования решений классификаторов ансамбля. Существует несколько методов комбинирования:

1. *Голосование*. При голосовании выбирается класс, наиболее часто встречающийся у выходов классификаторов ансамбля.
2. *Взвешенное голосование*. В ансамбле одни модели могут работать точнее других.

Поэтому каждой модели классификатора назначается числовой вес. Далее для каждого класса вычисляется сумма весов классификаторов, имеющих на выходе этот класс, и из полученных результатов выбирается максимальное значение.

1.6.2 Bagging

Процедура *bagging* (бэггинг) основана на формировании из исходного обучающего множества случайных подвыборок (бутстрэп - выборок) с возвращением, с тем же числом объектов, как и у исходного обучающего множества. Затем на основе каждой подвыборки строится модель классификатора. Известно, что при использовании процедуры *bagging* получаются точные результаты, именно поэтому этот подход получил широкое применение [20].

Таким образом, алгоритм построения ансамбля классификаторов с использованием процедуры *bagging* включает следующие шаги [20]:

1. Формирование N бутстрэп – выборок из исходного обучающего множества. Где N – количество классификаторов в ансамбле.
2. Для каждой бутстрэп – выборки, сформированной на шаге 1, строится модель классификатора.
3. Выполняется итоговая классификация с помощью методов голосования или взвешенного голосования.

На рисунке 1.4 изображена схема этого алгоритма.

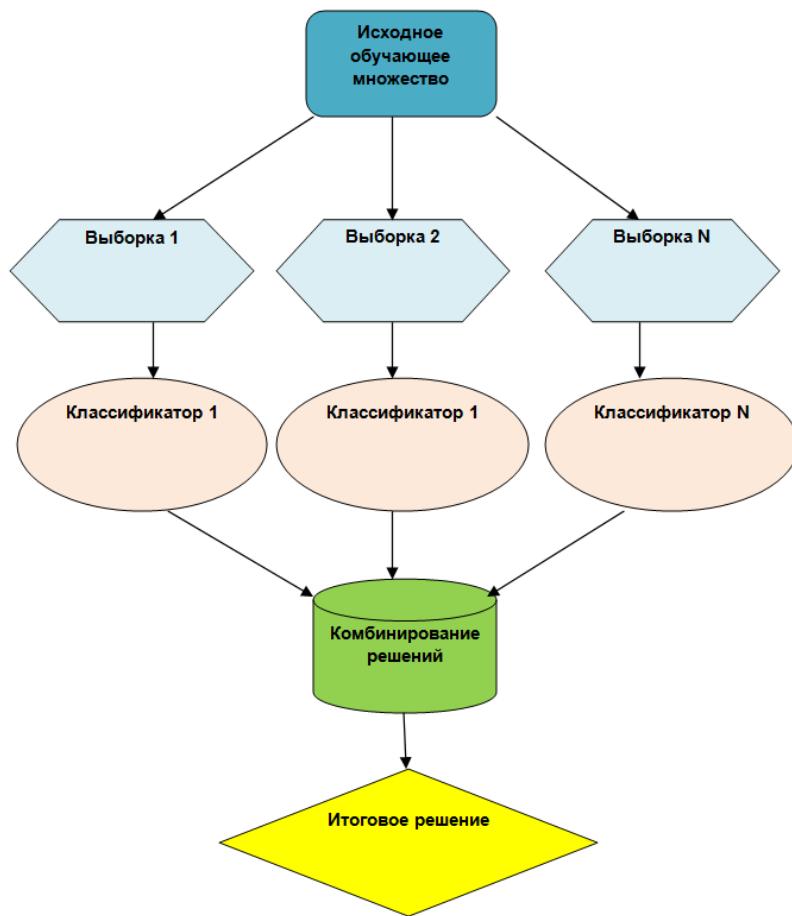


Рисунок 1.4 – Схема процедуры построения ансамбля классификаторов

Подход, основанный на бэггинге, имеет следующие достоинства:

- простота программной реализации;
- возможность распараллеливания алгоритма;
- высокая точность ансамбля.

Недостатками бэггинга являются:

1. Недетерминированность результата, так как обучающие выборки формируются случайно.

- Сложность интерпретации результатов по сравнению с индивидуальными моделями.

1.6.3 Алгоритм AdaBoost

Существует еще одна процедура построения ансамбля классификаторов, которая получила название *boosting* (бустинг). В отличие от бэггинга она использует исходное обучающее множество для всех классификаторов ансамбля. Бустинг представляет собой итеративную процедуру, в которой на каждой итерации производится перевзвешивание примеров обучающей выборки. Перевзвешивание осуществляется таким образом, чтобы последующий классификатор делал меньше ошибок, чем предыдущий [20]. При этом вес примера можно интерпретировать как вероятность его выбора для обучения следующего классификатора. Наиболее широкое применение получил алгоритм *AdaBoost*, который приведен в виде следующего псевдокода [21]

Листинг 1.1. Псевдокод алгоритма AdaBoost

Вход:

$X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки;

$Y = 1, \dots, l$ – номера классов (возможные значения классификаторов);

C – множество базовых алгоритмов классификации;

T – число итераций;

Выход:

ансамбль классификаторов $C(x)$

- $C(x) = \emptyset$
- for** $j = 1$ **to** T
- do** $\omega_0(j) = 1/n$ ► инициализация весов каждого объекта из X
- for** $j = 1$ **to** T
- do** $c^j(x) = \arg \min_{c_k^j(x) \in C} \varepsilon_k$ ► выбираем классификатор $c^j(x)$, который минимизирует взвешенную ошибку классификации, где

$$\varepsilon_k = \sum_{i=1}^n \omega_j(i)[y_i \neq c^j(x_i)]$$

- if** $\varepsilon_j \neq 0$ **and** $\varepsilon_j \leq 0.5$ ► здесь ε_j – взвешенная ошибка классификатора

$$c^j(x)$$

- then** $C(x) = C(x) \cup \{c^j(x)\}$

8.	$\beta_j = \frac{1}{2} \ln \frac{1 - \varepsilon_j}{\varepsilon_j}$	► вычисляем вес классификатора $c^j(x)$
9.	for $i = 1$ to n	► обновление весового распределения примеров
10.	do $\omega_j(i) = \omega_j(i) \cdot e^{-\beta_j y_i c^j(x_i)}$	
11.	$Z_j = \sum_{i=1}^n \omega_j(i)$	► вычислить сумму весов примеров
12.	for $i = 1$ to n	► нормировка весов всех объектов
13.	do $\omega_j(i) = \omega_j(i) / Z_j$	
14.	else break	► иначе останавливаемся
15.	return $C(x) = \arg \max_{y \in Y} \sum_{j=1}^N \beta_j$	

где N – число базовых классификаторов в ансамбле

Приведенный выше алгоритм работает следующим образом. Сначала в строках 2 – 3 происходит задание начальных весов для всех примеров обучающего множества X . Далее в цикле от $j = 1, \dots, T$ сначала выбирается классификатор $c^j(x)$ (строка 5), который минимизирует взвешенную ошибку классификации ε_k . И, если взвешенная ошибка $\varepsilon_j \neq 0$ и $\varepsilon_j \leq 0.5$, то классификатор $c^j(x)$ включается в ансамбль, и запоминается его вес (строки 7 - 8). Затем, в строках 9 – 13, происходит обновление весового распределения примеров. В конце выполняется итоговая классификация с помощью метода взвешенного голосования.

Таким образом, подход, основанный на бустинге, имеет следующие особенности:

- высокая точность ансамбля;
- ошибка всего ансамбля на каждой итерации стремится к нулю, т.к. алгоритм учитывает ошибки предыдущих классификаторов.

Недостатками бустинга являются:

1. Примеры с низкими весами не учитываются на следующей итерации, поэтому часть полезной информации может быть утеряна [13].
2. Сложность интерпретации результатов по сравнению с индивидуальными моделями.
3. В отличие от бэггинга не допускает распараллеливание.

1.6.4 Алгоритм Random Forests

В работе [8] рассматривается метод построения ансамблей классификаторов на основе случайных лесов, который был предложен Брейманом Л. Идея этого метода, заключается в построении ансамбля бинарных деревьев решений с использованием процедуры бэггинга. Алгоритм *Random Forests* (Случайные леса) включает следующие шаги

1. Формирование N случайных подвыборок на основе бэггинга. Где N – количество классификаторов в ансамбле.
2. Для каждой подвыборки, сформированной на шаге 1, строится полное (без усечения) бинарное дерево решений, где критерием остановки расщепления является минимальное количество объектов n_{\min} в листьях. Причем, расщепление очередной вершины выполняется на основе p случайно выбранных признаков (атрибутов).
3. Выполняется итоговая классификация с помощью метода голосования.

Преимуществами алгоритма *Random Forests* являются [8]:

1. Высокая точность за счет двойной инъекции случайности в алгоритм.
2. Возможность распараллеливания.
3. Простота применения, так как параметрами алгоритма являются количество деревьев в ансамбле и количество случайно отбираемых признаков при расщеплении.
4. Нет необходимости решать задачу усечения деревьев.
5. Простота реализации.

Основным недостатком алгоритма *Random Forests* является сложность интерпретации построенной модели.

1.6.5 Алгоритм Stacking

Стэкинг использует концепцию мета - обучения, т.е. пытается обучить классификатор, используя алгоритм, позволяющий обнаружить лучшую комбинацию выходов базовых моделей [13]. Существует множество вариантов алгоритма стэкинга. В общем случае для работы алгоритма стэкинг необходимо задать следующие параметры:

- $X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки, где x_i – i -ый объект обучающей выборки, n – количество объектов в выборке;
- $Y = 1, \dots, l$ – номера классов (возможные значения классификаторов);
- множество базовых алгоритмов классификации: $C = \{c_i(x)\}, i = \overline{1, r}$, где $c_i(x)$ – i -ый базовый классификатор, r – количество базовых классификаторов. Выходы базовых

классификаторов интерпретируются как новое множество признаков для *мета – данных*, на основе которых обучается *мета – классификатор*.

- *Мета – классификатор* $H(x)$. Входом мета – классификатора являются мета – данные, сформированные с помощью базовых алгоритмов, т.е. множество меток классов, к которым базовые алгоритмы отнесли описание входного объекта. Например, пусть имеется обучающее множество X , множество классов $Y = \{1, 2, 3\}$ и три базовых классификатора (произвольных). Тогда для объекта $x \in X$, с меткой класса 2, соответствующий мета – объект может выглядеть следующим образом: $\{1, 2, 1, 2\}$, где последний элемент множества соответствует номеру класса объекта x , т. е. номеру 2.

В самом простом варианте, алгоритм стэкинга выглядит следующим образом [21]:

1. Сначала обучаются все базовые классификаторы $C = \{c_i(x)\}, i = \overline{1, r}$;
2. Формируется *мета - множество* $M = \{x_i\}, i = \overline{1, n}$ на основе построенных базовых моделей. Причем M содержит r входных атрибутов и состоит из значений классов, к которым базовые классификаторы отнесли соответствующие объекты обучающего множества X ;
3. Строится мета-классификатор на основе множества M .

1.7 Ансамблевые алгоритмы в стандартном статистическом ПО

На сегодняшний день существует несколько программных продуктов, в которых реализованы ансамблевые алгоритмы классификации. Наиболее распространенными из них являются: *Weka*, *Statistica* и *SPSS*. В данном разделе проводится обзор ансамблевых алгоритмов, реализованных в *Weka* и *Statistica 10.0*.

1.7.1 Weka

Weka (Waikato Environment for Knowledge Analysis) представляет собой открытое программное обеспечение для интеллектуального анализа данных, написанное на языке программирования *Java* в университете Уайкато. Система позволяет непосредственно применять алгоритмы к выборкам данных, а также вызывать алгоритмы из программ на языке *Java* [11]. *Weka* имеет пользовательский интерфейс *Explorer*, который состоит из нескольких панелей [23]:

- панель предобработки данных *Preprocess panel*, которая позволяет загружать данные из базы, файлов и т. д.;
- панель классификации *Classify* позволяет применять алгоритмы классификации и регрессии к выборке данных, оценивать предсказательную способность алгорит-

мов, визуализировать ошибочные предсказания, *ROC*-кривые, и сам алгоритм, если это возможно;

- панель поиска ассоциативных правил *Associate* решает задачу выявления всех значимых взаимосвязей между признаками;
- панель кластеризации *Cluster* даёт доступ к различным алгоритмам кластеризации;
- панель отбора признаков *Select attributes* даёт доступ к методам отбора признаков;
- панель визуализации *Visualize* строит матрицу графиков разброса (*scatter plot matrix*), позволяет выбирать и увеличивать графики, и т. д.

В Weka реализовано несколько одиночных алгоритмов классификации и несколько ансамблевых алгоритмов, таких как однородные ансамблевые алгоритмы *Random Forests*, *AdaBoost* и *Bagging*. К примеру, для того чтобы выбрать алгоритм *Bagging* необходимо в главном окне программы перейти во вкладку *Classify*, затем нажать на кнопку «*Choose*» выбрать его из списка в пункте *meta* (рисунок 1.5)

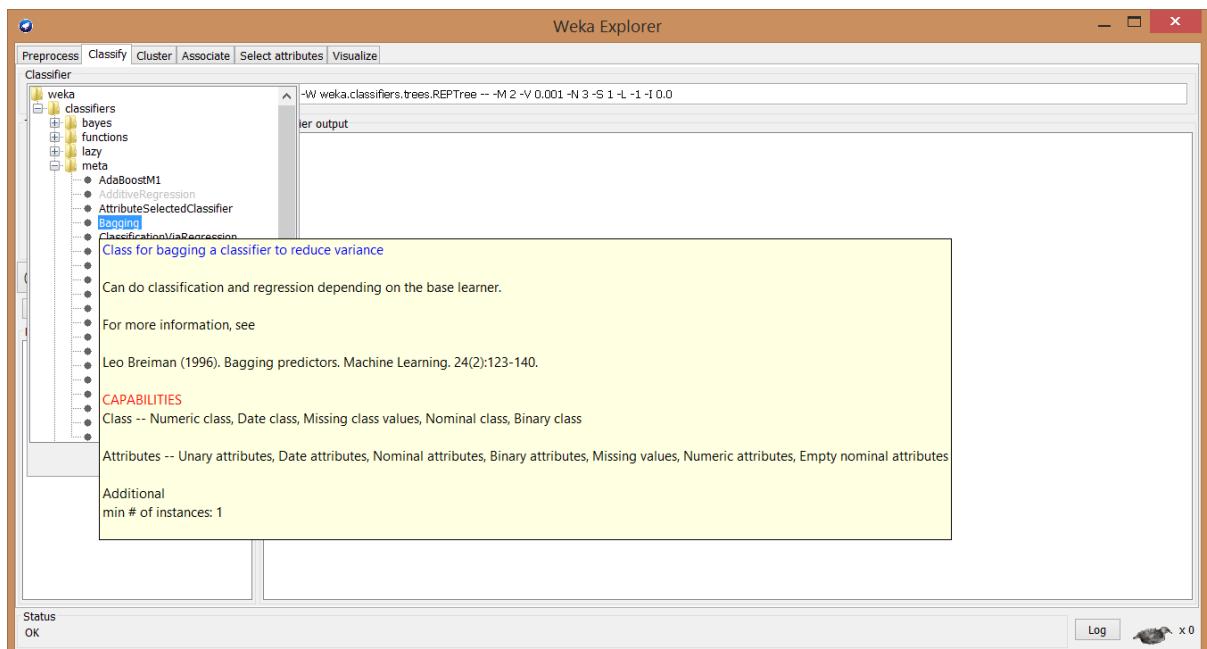


Рисунок 1.5 – Выбор алгоритма *Bagging* в Weka

Для настройки параметров алгоритма необходимо нажать на текстовое поле, которое расположено справа от кнопки «*Choose*» (рисунок 1.6)

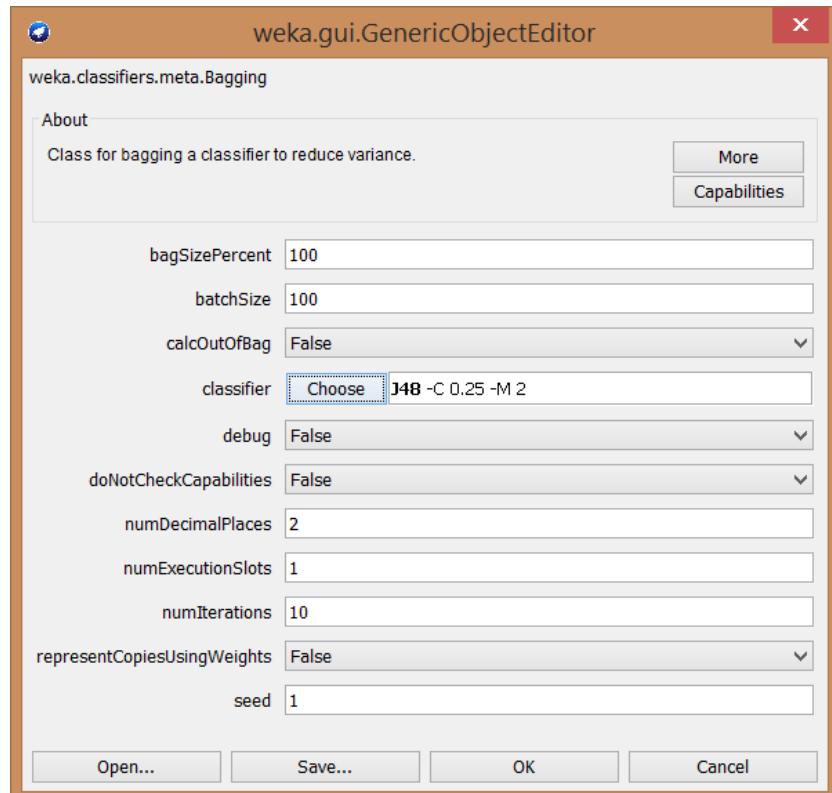


Рисунок 1.6 – Настройка параметров алгоритма *Bagging*

Здесь можно задать такие основные параметры, как размер бутстрэп - выборок (в процентах) на каждой итерации, базовый алгоритм классификации, число классификаторов в ансамбле и другие. После настройки параметров ансамблевого алгоритма и нажатия на кнопку «*OK*» на панели *Classifier output* будут отображены результаты классификации (рисунок 1.7).

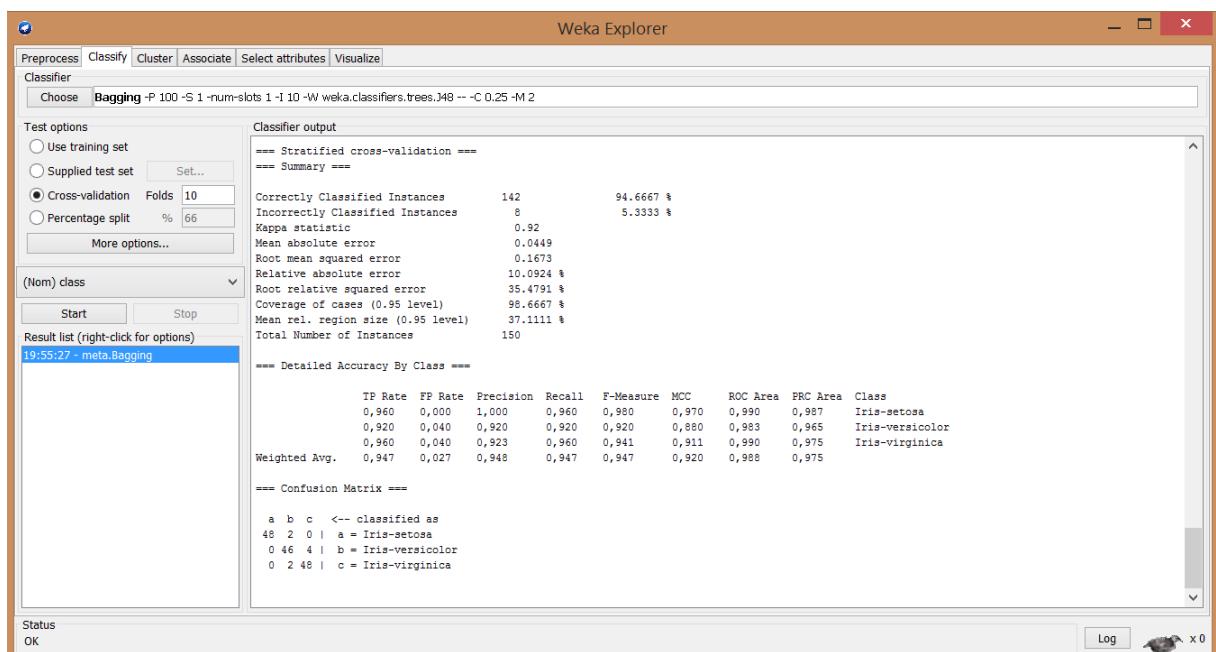


Рисунок 1.7 – Результаты классификации с помощью алгоритма *Bagging*

На этой панели собрана вся информация, касающаяся построенной модели ансамбля и результатов оценки точности. Однако интерпретировать эти результаты очень сложно, т. к. они переполнены специфичной информацией, в которой сможет разобраться только специалист в области анализа данных.

Таким образом, к основным достоинствам программной системы *Weka* можно отнести:

1. Возможность использования реализованных алгоритмов в программах, написанных на *Java*.
2. *Weka* является свободно распространяемым программным продуктом.
3. Простой графический интерфейс пользователя.
4. Подробная документация на английском языке.

Основными недостатками являются:

1. Ансамблевые алгоритмы представлены ограниченно и в системе реализованы только однородные ансамблевые алгоритмы классификации.
2. Отсутствует возможность просмотра результатов моделей классификаторов, входящих в ансамбль.
3. Отсутствует возможность параллельной работы с несколькими выборками данных.
4. Отсутствует возможность редактирования типов входных атрибутов (признаков).
5. Отсутствует возможность классификации нового примера на основе построенной модели.

1.7.2 Statistica

Statistica — это профессиональный программный пакет для статистического анализа данных. Он включает в себя реализации функций анализа, управления, добычи, визуализации данных с привлечением статистических методов [24]. В *Statistica 10* ансамблевые алгоритмы представлены еще более ограниченно чем в *Weka*. В *Statistica 10* реализован только один однородный ансамблевый алгоритм классификации *Random Forests* (случайные леса), включающий в качестве одиночных классификаторов модели деревьев решений. Для того чтобы открыть диалоговое окно с настройками параметров этого алгоритма, необходимо перейти в пункт меню *Data Miner->Random Forests for Regression and Classification*. Затем в появившемся диалоговом окне выбрать пункт *Classification Analysis* и нажать *OK*. В результате появится диалоговое окно с настройками параметров алгоритма *Random Forests* (рисунок 1.8)

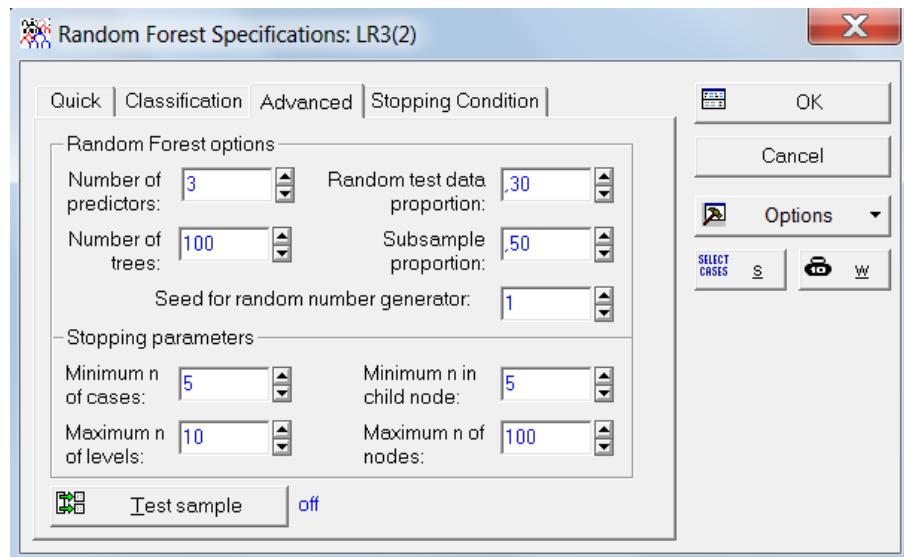


Рисунок 1.8 – Диалоговое окно с настройками параметров алгоритма *Random Forests*

Здесь задаются такие основные параметры как: число деревьев (*Number of trees*), число случайных атрибутов на каждом расщеплении вершины (*Number of predictors*), минимальное число объектов в листе (*Minimum n in child node*), максимальное число узлов в одном дереве (*Maximum n of nodes*) и другие. После настройки параметров алгоритма необходимо нажать на кнопку *OK* для построения модели случайного леса. После завершения работы алгоритма появится окно с результатами классификации (рисунок 1.9)

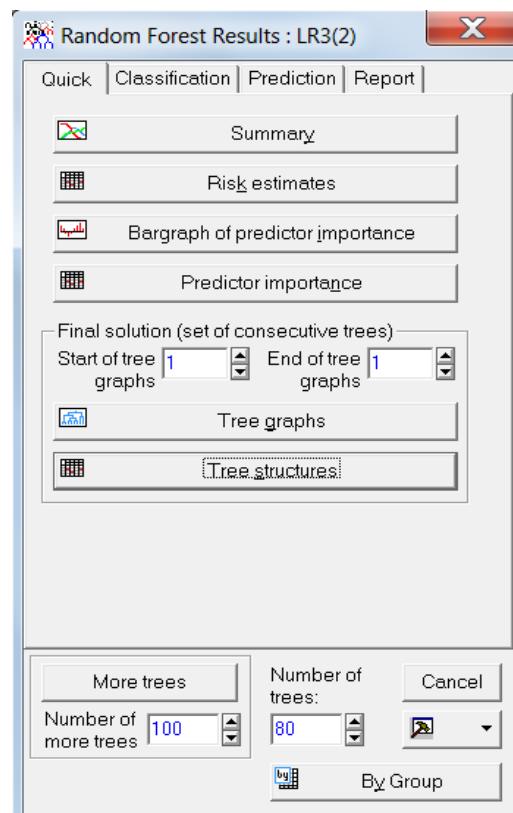


Рисунок 1.9 – Результаты классификации с помощью алгоритма *Random Forests*

Здесь можно просматривать структуры деревьев решений и результаты классификации. Во вкладке *Classification* также приводится матрица классификации. Основными недостатками здесь являются: отсутствие возможности просмотра результатов деревьев, входящих в ансамбль, отсутствие возможности классификации нового примера на основе построенной модели, также нельзя сохранить построенную модель.

1.8 Выводы по главе

В ходе аналитического обзора было выполнено следующее:

1. Проведен аналитический обзор источников по исследуемой теме.
2. Рассмотрены основные индивидуальные алгоритмы построения моделей классификаторов. Эти алгоритмы будут использованы в качестве составляющих ансамбля классификаторов.
3. Рассмотрены основные подходы к построению ансамблей классификаторов.
4. Рассмотрены базовые алгоритмы построения ансамблей классификаторов такие как: *Bagging*, *AdaBoost*, *Random Forests*. Приведены основные достоинства и недостатки этих алгоритмов.
5. Проведен обзор существующих программных средств, в которых реализованы ансамблевые алгоритмы.

В результате проведенного в этой главе обзора можно сделать следующие выводы.

1. Использование ансамблевого подхода приводит к повышению точности и устойчивости классификационных решений.
2. Большинство работ в области разработки ансамблей классификаторов посвящены построению однородного ансамбля.
3. В стандартных статистических пакетах ансамблевые алгоритмы представлены весьма ограниченно.
4. Неоднородные ансамблевые алгоритмы в профессиональных статистических пакетах на настоящий момент не реализованы.
5. Также не существует средств для автоматического подбора оптимальных параметров для ансамблевых алгоритмов.

В связи с вышеизложенным, а также учитывая, что необходимым и достаточным условием точности ансамбля является различность составляющих его классификаторов, можно сделать вывод о том, что разработка неоднородных ансамблевых алгоритмов и программных средств, реализующих ансамблевые алгоритмы классификации, представляется перспективным направлением.

2 Разработка алгоритмического обеспечения системы

2.1 Алгоритмы вычисления оценок качества классификаторов

В данном разделе рассматриваются различные оценки качества моделей классификаторов, которые используются в настоящей работе. Одной из основных характеристик классификатора является его точность в процентах, которая определяется по формуле (1.1). Существует несколько методов для оценки точности классификационных решений. Наиболее широкое применение получил метод $k \times V$ – блочной кросс – проверки на тестовой выборке, подробное описание которого приводится в следующем разделе.

2.1.1 Методы оценки точности классификации

В работе используются два метода оценки точности классификации: использование всей обучающей выборки и $k \times V$ – блочная кросс – проверка на тестовой выборке ($k \times V$ – folds cross validation). Оба метода реализованы в программной системе *ECA*. В последнем методе исходная выборка разбивается на V блоков примерно одинаковой длины, случайным образом, но со стратификацией классов. Каждый из V блоков поочерёдно объявляется тестовой подвыборкой, остальные $V - 1$ блоков объединяются в обучающую подвыборку. Далее классификатор строится по обучающей подвыборке, затем классифицирует объекты тестовой подвыборки. Процедура обучения повторяется V раз, в результате все объекты оказываются классифицированными как тестовые ровно по одному разу, и как обучающие по $V - 1$ раз [25].

Процедура $k \times V$ – блочной кросс – проверки на тестовой выборке является k - кратным выполнением описанной выше процедуры при k различных случайных разбиениях на V блоков, со стратификацией классов. В результате $k \times V$ – блочной кросс – проверки все объекты оказываются классифицированными как тестовые ровно по k раз, и как обучающие по $k(V-1)$ раз [25].

2.1.2 Ошибки классификации

В работе приводятся значения двух ошибок классификации: средняя абсолютная ошибка (*mean absolute error*) и квадратичная средняя ошибка (*root mean squared error*). Средняя абсолютная ошибка и среднеквадратическая ошибка вычисляются по формулам (2.2) и (2.3) соответственно:

$$E = \frac{\text{SumAbsErr}}{N}. \quad (2.2)$$

$$E_q = \sqrt{\frac{SumSqrErr}{N}}, \quad (2.3)$$

где $SumAbsErr$ - суммарная вероятность ошибки классификации, которая вычисляется по формуле:

$$SumAbsErr = \sum_{i=1}^N \frac{AbsErr_i}{c}. \quad (2.4)$$

а $SumSqrErr$ - квадратичная вероятность ошибочной классификации:

$$SumSqrErr = \sum_{i=1}^N \frac{AbsErr_i^2}{c}. \quad (2.5)$$

В этих формулах $AbsErr_i$ - вероятность ошибочной классификации данного примера, c – число классов.

Если в качестве метода оценки точности используется $k \times V$ – блочная кросс – проверка на тестовой выборке, то обычно вычисляются еще дополнительные показатели, такие как дисперсия ошибки классификатора:

$$s^2 = \frac{1}{kV} \sum_{i=1}^{kV} (\varepsilon_i - \bar{\varepsilon})^2, \quad (2.6)$$

где $\bar{\varepsilon}$ - средняя ошибка классификатора, ε_i - ошибка классификатора на итерации i . Также строится доверительный интервал ошибки классификатора

$$\bar{\varepsilon} - t_\alpha \frac{s}{\sqrt{kV}} \leq a \leq \bar{\varepsilon} + t_\alpha \frac{s}{\sqrt{kV}}, \quad (2.7)$$

где $\bar{\varepsilon}$ - средняя ошибка классификатора, t_α - квантиль распределения Стьюдента уровня α , s - выборочное среднеквадратическое отклонение (СКО) ошибки классификатора.

2.1.3 ROC - анализ

ROC-кривая (*Receiver Operator Characteristic*) – кривая, которая наиболее часто используется для представления результатов бинарной классификации. *ROC* - кривая показывает зависимость количества верно классифицированных положительных примеров от количества неверно классифицированных отрицательных примеров. При этом предполагается, что у классификатора имеется некоторый параметр, варьируя который, можно получать то или иное разбиение на два класса. Этот параметр часто называют порогом, или точкой отсечения. В зависимости от него будут получаться различные величины ошибок I и II рода [10]. В таблице 2.1 приведена таблица сопряженности ошибок I и II рода [10].

Таблица 2.1 – Таблица сопряженности ошибок I и II рода

Модель	Фактически	
	Положительно	Отрицательно
Положительно	TP	FP (<i>ошибка I рода</i>)
Отрицательно	FN (<i>ошибка II рода</i>)	TN

В этой таблице используются следующие обозначения [10]:

- TP (*True Positives*) – верно классифицированные положительные примеры (истинно положительные случаи);
- TN (*True Negatives*) – верно классифицированные отрицательные примеры (истинно отрицательные случаи);
- FN (*False Negatives*) – положительные примеры, классифицированные как отрицательные (*ошибка II рода* - ложноотрицательные примеры);
- FP (*False Positives*) – отрицательные примеры, классифицированные как положительные (*ошибка I рода* - ложноположительные случаи).

Это количественные показатели, которые показывают, сколько примеров как классифицировано.

Для идеального классификатора график *ROC* - кривой проходит через верхний левый угол. Поэтому, чем ближе кривая к верхнему левому углу, тем выше предсказательная способность модели. И, наоборот, чем меньше изгиб кривой, и чем ближе она расположена к диагональной прямой, тем менее эффективна модель. Диагональная линия соответствует плохому классификатору, то есть полной неразличимости двух классов [10].

В *ROC* – анализе существуют различные оценки качества модели. Одними из них являются TPR , FPR , TNR , FNR , *Precision*, *Recall* и *F-Measure*. Рассмотрим эти оценки более подробно:

Доля истинно положительных примеров:

$$TPR = \frac{TP}{TP + FN}. \quad (2.8)$$

Доля ложноположительных примеров:

$$FPR = \frac{FP}{FP + TN}. \quad (2.9)$$

Доля истинно отрицательных примеров:

$$TNR = \frac{TN}{FP + TN}. \quad (2.10)$$

Доля ложноотрицательных примеров:

$$FNR = \frac{FN}{TP + FN}. \quad (2.11)$$

Precision – точность, показывающая, сколько из классифицированных как правильные являются правильными:

$$Precision = \frac{TP}{TP + FP}. \quad (2.12)$$

Recall (Полнота) – это доля истинно положительных примеров, тоже что и *TPR*. Она позволяет характеризовать способность модели классифицировать как можно большее число положительных примеров из ожидаемых.

F – мера - характеристика качества построенной модели; изменяется от 0 до 1. Она является средней гармонической величин *Precision* и *Recall* и вычисляется по формуле:

$$F = \frac{2PR}{P + R}, \quad (2.13)$$

где $P = Precision$, $R = Recall$.

Чувствительностью (*Sensitivity*) называют долю истинно положительных случаев, то есть чувствительность это есть *TPR*.

Специфичностью (*Specificity*) называют долю истинно отрицательных случаев, то есть специфичность это есть не что иное как *TNR*.

Также существует еще один численный показатель – площадь под кривой *AUC*, которая изменяется от 0 до 1 («идеальный» классификатор). Чем больше показатель *AUC*, тем лучше предсказательная способность модели. В таблице 2.2 показана градация качества модели от значения этого параметра [10].

Таблица 2.2 – Определение качества модели от показателя *AUC*

Интервал <i>AUC</i>	Качество модели
0.9 – 1.0	Отличное
0.8 – 0.9	Очень хорошее
0.7 – 0.8	Хорошее
0.6 – 0.7	Среднее
0.5 – 0.6	Неудовлетворительное

2.1.4. Значимые атрибуты

ROC – анализ также применяется для определения значимых атрибутов с помощью логистической регрессии. При этом используется метод на основе результатов построения *ROC*-кривых и вычисления *AUC*. Сначала для каждого атрибута j строится модель логистической регрессии и затем вычисляется среднее арифметическое $AUC_{cp.,j}$ по формуле:

$$AUC_{cp.,j} = \frac{1}{c} \sum_{i=1}^c AUC_{ij}, \quad (2.14)$$

где c – количество классов, AUC_{ij} - площадь под кривой для j -го атрибута для i -го класса. Если это значение превышает 0,6 (средняя значимость), то данный атрибут j можно считать значимым, иначе – нет.

2.2 Описание неоднородного ансамблевого алгоритма

В данном разделе приводится описание разработанного неоднородного ансамблевого алгоритма классификации и его модификации. Основная идея этих алгоритмов заключается в итерационном применении одиночных классификаторов на обучающей выборке и учете в итоговом решении при построении ансамбля вклада только тех классификаторов, ошибка классификации которых не превосходит заданный порог.

Основными отличительными особенностями этих алгоритмов являются:

1. Задание любой комбинации базовых алгоритмов классификации, а также настройка их параметров. В качестве базовых классификаторов могут выступать любые методы классификации, такие как деревья решений, нейронные сети, логистическая регрессия и другие.
2. Выбор метода формирования обучающих выборок на каждой итерации.
3. Выбор критерия, по которому выбирается классификатор на очередной итерации.
4. Выбор метода голосования для определения окончательного результата классификации.

Таким образом, для работы неоднородного ансамблевого алгоритма и модифицированного неоднородного ансамблевого алгоритма необходимо задать следующие **основные параметры**:

- $X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки, где x_i – i -ый объект обучающей выборки, n – количество объектов в выборке;
- $Y = 1, \dots, l$ – номера классов (возможные значения классификаторов);
- Число итераций – T ;

- Минимальный порог ошибки для включения базовых классификаторов в ансамбль – ε_{\min} ;
 - Максимальный порог ошибки для включения базовых классификаторов в ансамбль – ε_{\max} ;
 - Множество базовых алгоритмов классификации: $C = \{c_i(x)\}, i = \overline{1, r}$, где $c_i(x)$ – i -ый базовый классификатор, r – количество базовых классификаторов.
- и **дополнительные параметры**, такие как:
1. процедуру формирования обучающих выборок на каждой итерации: $S \in \{\text{INITIAL}, \text{BOOTSTRAP}, \text{RANDOM_SUBSAMPLE}, \text{RANDOM_BOOTSTRAP}\}$:
 - *использование исходного обучающего множества* (параметр *INITIAL*). На каждой итерации для обучения индивидуального классификатора будет использовано все обучающее множество X ;
 - *бутстрэп – выборки* (параметр *BOOTSTRAP*). На каждой итерации для обучения индивидуального классификатора будет использована бутстрэп – выборка, сформированная из исходной обучающей выборки X ;
 - *случайные подвыборки* (параметр *RANDOM_SUBSAMPLE*). На каждой итерации для обучения классификатора будет использована подвыборка случайного размера $d \sim [1, N]$, но в отличие от бутстрэп – выборки, каждый объект может отбираться в выборку только один раз;
 - *бутстрэп-выборки случайного размера* (параметр *RANDOM_BOOTSTRAP*). На каждой итерации для обучения классификатора будет использована бутстрэп – выборка случайного размера $d \sim [1, N]$.
 2. Выбор индивидуального классификатора на каждой итерации: $I \in \{\text{OPTIMAL_CLASSIFIER}, \text{RANDOM_CLASSIFIER}\}$:
 - *случайный классификатор* (параметр *RANDOM_CLASSIFIER*). На каждой итерации индивидуальный классификатор выбирается случайным образом (равновероятно);
 - *оптимальный классификатор* (параметр *OPTIMAL_CLASSIFIER*). На каждой итерации выбирается индивидуальный классификатор, который минимизирует ошибку классификации на соответствующей выборке, сформированной на j -ой итерации.
 3. Метод голосования: $A \in \{\text{MAJORITY_VOTES}, \text{WEIGHTED_VOTES}\}$:
 - *метод большинства голосов* (параметр *MAJORITY_VOTES*);

- метод взвешенного голосования (параметр *WEIGHTED_VOTES*), где вес индивидуального классификатора j вычисляется по формуле:

$$\omega_j = \frac{1}{2} \ln \frac{1 - \varepsilon_j}{\varepsilon_j}, \quad (2.15)$$

где ε_j – ошибка классификатора на выборке, сформированной на итерации j .

Разработанный неоднородный ансамблевый алгоритм можно представить в виде последовательности выполнения следующих шагов на каждой итерации $j = 1, \dots, T$:

1. Формирование обучающей подвыборки из исходной обучающей выборки X^b на основе заданного метода S .
2. Выбор классификатора $c^j(x)$ из множества C с помощью метода выбора классификатора I .
3. Обучение классификатора $c^j(x)$ на выборке X^b .
4. Вычисление ошибки ε (относительное количество неверно классифицированных объектов), построенной модели классификатора.
5. Включение в ансамбль классификатора $c^j(x)$ в случае, если ошибка ε классификации не превосходит заданного порога: $\varepsilon > \varepsilon_{\min}$ и $\varepsilon < \varepsilon_{\max}$; в противном случае классификатор не включается в ансамбль. Также, здесь вычисляется и запоминается вес классификатора $c^j(x)$ в случае, если в качестве метода итоговой классификации используется взвешенное голосование.

На последнем шаге выполняется итоговая классификация на основе использования построенного ансамбля с применением заданного метода голосования A . Также следует отметить, что при задании параметра $S = INITIAL$, произойдет последовательное обучение всех классификаторов из множества C . Так как нет смысла создавать множество дубликатов моделей. В листинге 2.1 приведено описание разработанного неоднородного ансамблевого алгоритма в виде псевдокода

Листинг 2.1 Псевдокод неоднородного ансамблевого алгоритма

Вход:

$X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки;

$Y = I, \dots, l$ – номера классов (возможные значения классификаторов);

ε_{\min} – минимальный порог ошибки для включения базовых классификаторов в ансамбль;

ε_{\max} – максимальный порог ошибки для включения базовых классификаторов в ансамбль;

$C = \{c_i(x)\}, i = \overline{1, r}$ - множество базовых алгоритмов классификации;

$S \in \{\text{INITIAL}, \text{BOOTSTRAP}, \text{RANDOM_SUBSAMPLE}, \text{RANDOM_BOOTSTRAP}\}$ - метод формирования обучающих подвыборок на каждой итерации;

$I \in \{\text{RANDOM_CLASSIFIER}, \text{OPTIMAL_CLASSIFIER}\}$ - метод выбора базового классификатора на каждой итерации;

$A \in \{\text{MAJORITY_VOTES}, \text{WEIGHTED_VOTES}\}$ - метод голосования;

$\text{Sample}(X, S)$ – функция формирования обучающей подвыборки из исходной выборки X с помощью метода S ;

$\text{SelectClassifier}(C, I)$ – функция выбора базового классификатора из множества C с помощью метода I ;

1. **if** $I = \text{OPTIMAL_CLASSIFIER}$
2. **then return** $c(x) = \arg \min_{c_k(x) \in C} \varepsilon_k$ ► ε_k - ошибка классификатора $c_k(x)$
3. **else return** $c(x) = \text{rnd}_{1 \leq i \leq r}(c_i(x))$

Выход:

ансамбль классификаторов $C(x)$

1. $C(x) = \emptyset$
2. $\omega = \emptyset$
3. **if** $S = \text{INITIAL}$
4. **then** $T = r$ ► устанавливаем T равным числу базовых алгоритмов классификации
5. **for** $j = 1$ **to** T
6. **do** $X^b = \text{Sample}(X, S)$
7. **if** $S = \text{INITIAL}$
8. **then** $c^j(x) = c_j(x)$
9. **else** $c^j(x) = \text{SelectClassifier}(C, I)$

```

10.    Обучить классификатор  $c^j(x)$  на выборке  $X^b$ 


$$\varepsilon = \frac{\sum_{i=1}^{|X^b|} [y_i \neq c^j(x_i)]}{|X^b|}$$

11.     $\varepsilon = \frac{\sum_{i=1}^{|X^b|} [y_i \neq c^j(x_i)]}{|X^b|}$  ► ошибка классификатора  $c^j(x)$  на выборке  $X^b$ 

12.    if  $\varepsilon > \varepsilon_{\min}$  and  $\varepsilon < \varepsilon_{\max}$ 
13.        then  $C(x) = C(x) \cup \{c^j(x)\}$ 
14.        if  $A = WEIGHTED\_VOTES$ 
15.            then  $\omega_j = \frac{1}{2} \ln \frac{1 - \varepsilon_j}{\varepsilon_j}$ 
16.             $\omega = \omega \cup \{\omega_j\}$ 
17.    end
18.    if  $C(x) = \emptyset$ 
19.        then return "Не удалось построить модель"
20.    if  $A = WEIGHTED\_VOTES$ 
21.        then return  $C(x) = \arg \max_{y \in Y} \sum_{j=1}^N \omega_j$ 
22.        else return  $C(x) = \arg \max_{y \in Y} \sum_{j=1}^N [c^j(x) = y]$ 
где  $N$  – число базовых классификаторов в ансамбле.

```

Рассмотрим приведенный выше псевдокод более подробно. На начальном этапе создаются пустое множество для ансамбля классификаторов $C(x)$ и множество ω для хранения их весов. Далее, если параметр $S = INITIAL$, то число итераций T устанавливается равным числу базовых классификаторов r (строки 3 - 4). Далее в цикле от $j = 1, \dots, T$ выполняются основные шаги алгоритма. Сначала в строке 6 формируется обучающее подмножество на основе заданного метода S . В строках 7 – 9 происходит выбор очередного классификатора $c^j(x)$ на основе метода выбора I . Затем вычисляется ошибка ε классификатора $c^j(x)$ на выборке X^b . Далее, если ошибка $\varepsilon_{\min} < \varepsilon < \varepsilon_{\max}$, то классификатор $c^j(x)$ включается в ансамбль и запоминается его вес ω_j в случае, если параметр $A = WEIGHTED_VOTES$. В конце выполняются следующие действия. Если не удалось включить ни одну модель в ансамбль, то возвращается ошибка (строки 18 - 19). В противном случае выполняется итоговая классификация с помощью метода голосования A (строки 21 - 22).

Таким образом, в разработанном неоднородном ансамблевом алгоритме можно различным образом варьировать входные параметры, что в очередной раз подчеркивает его уникальность и гибкость.

2.3 Описание модифицированного неоднородного ансамблевого алгоритма

В этом алгоритме используется точно такая же идея, как и в неоднородном ансамблевом алгоритме. Основное отличие заключается в том, что в процессе построения ансамбля формируются обучающие подмножества с различным числом атрибутов, т. е. на каждой итерации j выбирается случайное число атрибутов $s \sim [1, k]$ (k – число атрибутов), которые также отбираются случайным образом (равновероятно). Также, если задать параметр $S = INITIAL$, то алгоритм будет вести себя точно также как и в остальных случаях, в отличие от базовой версии неоднородного ансамблевого алгоритма. Более детально модифицированный неоднородный ансамблевый рассматривается в листинге 2.2

Листинг 2.2 Псевдокод модифицированного неоднородного ансамблевого алгоритма

Вход:

Точно такие же входные параметры, как и у неоднородного ансамблевого алгоритма (см. листинг 2.1).

Выход:

ансамбль классификаторов $C(x)$

1. $C(x) = \emptyset$
2. $\omega = \emptyset$
3. **for** $j = 1$ **to** T
4. **do** $s = rnd(i)$ ► генерируем случайное число атрибутов
 $1 \leq i \leq k$
5. $X_s^b = Sample(X, S)$ ► формируем выборку с s случайными атрибутами
6. $c^j(x) = SelectClassifier(C, I)$
7. Обучить классификатор $c^j(x)$ на выборке X_s^b
8. $\varepsilon = \frac{\sum_{i=1}^{|X_s^b|} [y_i \neq c^j(x_i)]}{|X_s^b|}$ ► ошибка классификатора $c^j(x)$ на выборке X_s^b
9. **if** $\varepsilon > \varepsilon_{\min}$ **and** $\varepsilon < \varepsilon_{\max}$
10. **then** $C(x) = C(x) \cup \{c^j(x)\}$

```

11.      if  $A = WEIGHTED\_VOTES$ 
12.
13.      then  $\omega_j = \frac{1}{2} \ln \frac{1 - \varepsilon_j}{\varepsilon_j}$ 
14.       $\omega = \omega \cup \{\omega_j\}$ 
15. end
16. if  $C(x) = \emptyset$ 
17.   then return “Не удалось построить модель”
18. if  $A = WEIGHTED\_VOTES$ 
19.   then return  $C(x) = \arg \max_{y \in Y} \sum_{j=1}^N \omega_j$ 
20.   else return  $C(x) = \arg \max_{y \in Y} \sum_{j=1}^N [c^j(x) = y]$ 
где  $N$  – число базовых классификаторов в ансамбле.

```

Приведенный выше псевдокод отличается от базовой версии данного алгоритма только строками 4 – 5, где на каждой итерации происходит формирование обучающей подвыборки с s случайными атрибутами с помощью метода формирования обучающей подвыборки S .

2.4 Выводы по главе

В данной главе диссертационной работы сделано следующее:

1. Рассмотрены основные методы для оценки точности классификации, такие как использование обучающей выборки и метод $k \times V$ – блочной кросс проверки на тестовой выборке.
2. Выбраны основные характеристики качества классификации, такие как точность классификации, дисперсия ошибки классификации, доверительный интервал средней ошибки классификации, AUC .
3. Рассмотрен алгоритм для определения значимых признаков с использованием логистической регрессии и ROC – анализа.
4. Разработано два оригинальных ансамблевых алгоритма: неоднородный ансамблевый алгоритм и модифицированный неоднородный ансамблевый алгоритм. Основными отличительными особенностями этих алгоритмов являются возможность задания любой комбинации базовых классификаторов, которые впоследствии будут использованы при построении ансамбля и возможность варьирования таких параметров как S, I, A .

3 Разработка программной системы ECA

3.1 Основные функции системы

Программная система *ECA* предназначена для классификации разнотипных данных (количественных, порядковых, номинальных) из любой предметной области на основе ансамбля алгоритмов. В данной версии программной системы реализованы индивидуальные алгоритмы классификации, такие как: деревья решений (алгоритмы *CART*, *ID3*, *C4.5*, *CHAID*), логистическая регрессия, нейронные сети (многослойный персептрон) и алгоритм k – взвешенных ближайших соседей, а также пять ансамблевых алгоритмов классификации:

- неоднородный ансамблевый алгоритм;
- модифицированный неоднородный ансамблевый алгоритм;
- алгоритм случайные леса;
- алгоритм *AdaBoost*;
- алгоритм *Stacking*.

В программной системе реализован большой модуль *Data Miner*, функции которого заключаются в выдаче рекомендаций относительно выбора оптимальных параметров для ансамблевых алгоритмов и нейронных сетей на основе проведенного эксперимента.

Также в программной системе реализованы следующие функциональные возможности [31]:

1. Ввод, редактирование, сохранение исходных данных:

- загрузка исходных данных из файлов с расширениями *.xls*, *.xlsx*, *.csv*, *.arff*;
- загрузка исходных данных из сети по протоколам *http* и *ftp*;
- возможность удаленного подключения к базе данных и выборка из нее данных для последующей классификации;
- сохранение набора данных в файл с расширениями *.xls*, *.xlsx*, *.csv*, *.arff*;
- возможность работы с несколькими исходными выборками данных;
- возможность редактирования таблицы с исходными данными;
- возможность добавления и удаления объектов из таблицы с данными;
- возможность очистки таблицы с данными;
- возможность удаления объектов из таблицы, имеющих пропуски;
- возможность изменения имен признаков.

2. Настройка параметров классификации:

- возможность выбора исходных признаков для классификации и их типов (количественный и/или качественный);

- возможность выбора выходного признака (класса);
- возможность выбора любой комбинации базовых классификаторов, которые будут использоваться при построении ансамбля, а также настройка их параметров;
- выбор метода для оценки точности классификатора;
- настройка формата чисел (количество знаков после запятой).

3. Классификация, как представление результатов:

- выдача результатов классификации данных в табличном виде;
- классификация новых данных на основе построенной модели;
- сохранение результатов классификации в файл с расширениями *.xls*, *.xlsx*;
- визуализация деревьев решений и нейронных сетей, а также настройка параметров этих изображений;
- сохранение изображений деревьев решений и нейронных сетей в файл с расширением *.png*;
- сохранение изображений деревьев решений и нейронных сетей в системный буфер обмена;
- сохранение модели классификатора в файл с расширением *.txt*;
- загрузка модели классификатора из файла с расширением *.txt*;
- модуль для анализа данных: расчет основных статистических характеристик для атрибутов;
- модуль *Data Miner*;
- визуализация результатов классификации – построение *ROC* – кривых.

4. Настройка графического интерфейса:

- возможность изменения типа и размера шрифта в таблицах;
- возможность изменения цвета фона.

В последующих разделах будет приведено подробное описание структуры программной системы в целом и ее реализации.

3.2 Требования к программе

3.2.1 Требования к входным данным

Система должна обеспечивать загрузку исходных данных из файлов форматов *.xls*, *.xlsx*, *.csv*, *.arff*.

XLS – это документы *Microsoft Office Excel*. Данные должны храниться на первом листе. В первой строке должны быть записаны имена атрибутов без повторений. В последующих строках должны быть записаны данные в соответствии с перечисленными атрибутами (допускаются пропущенные значения). Посторонних записей на странице не должно быть.

CSV (*Comma - Separated Values* - значения, разделённые запятыми) – текстовый формат, предназначенный для представления табличных данных. Каждая строка файла — это одна строка таблицы. В первой строке должны быть записаны имена атрибутов через запятую, без пробелов. Данные записываются в соответствии с перечисленными атрибутами, разделяются запятыми без пробелов. Значения, содержащие пробелы должны быть заключены в кавычки.

ARFF (*Attribute-Relation File Format* – Файл формата атрибут – отношение) – файл является текстовым файлом *ASCII*, который описывает список экземпляров, пользующихся одним набором атрибутов. *ARFF* файлы имеют две отдельные секции. В первом разделе – заголовок – представлена информация об атрибуках (имя и значения), во втором – сами данные. Заголовок файла *ARFF* содержит имя отношения, список атрибутов (столбцов данных), и их типы.

Типы значений атрибутов могут быть следующие:

- *numeric* – числовой;
- *<nominal-specification>* – категориальный.

Числовые атрибуты могут быть вещественными или целыми числами. Категориальные атрибуты определяются путем предоставления их значений списком: значения списка заключены в фигурные скобки, а сами значения отделяются запятыми. Значения, содержащие пробелы должны быть заключены в кавычки.

Форматы *.csv* и *.arff* взяты из библиотеки *weka*. Эти форматы используются для представления данных в удобном для построения моделей классификаторов.

Также, система должна предоставлять возможность загрузки исходных данных из других источников:

- Из баз данных, таких как *MySQL*, *Oracle*, *SQL Server*, *Microsoft Access*, *PostgreSQL*, *SQLite*. Должна существовать возможность подключения к любой базе данных с последующим формированием выборок из нее с помощью *SQL* запросов.
- Из сети по протоколам *http* и *ftp*. Причем допускается загрузка файлов тех же форматов *.xls*, *.xlsx*, *.csv*, *.arff*.

3.2.2 Требования к программному обеспечению

Программная система должна представлять собой обычное настольное *java* – приложение, которое собирается с помощью *maven*. Также должны быть установлены следующие основные библиотеки:

- *weka-dev-3.9.1.jar* – библиотека машинного обучения и интеллектуального анализа данных (версии 3.9.1). В ней реализованы классы для построения модели логистической регрессии, построение *ROC* – кривых, классы для представления набора исходных данных и их фильтрацию;
- *poi-3.16-beta2.jar* – библиотека для работы с документами формата *Microsoft Office Excel*;
- *poi-ooxml-3.16-beta2.jar* – библиотека для работы с документами формата *Microsoft Office Excel 2007*;
- *jcommon-1.0.23.jar* и *jfreechart-1.0.19.jar* - библиотеки для построения и отображения графиков;
- *mysql-connector-java-5.1.40-bin.jar* – *jdbc* драйвер, который предоставляет возможность подключения к СУБД *MySQL*;
- *mssql-jdbc-6.1.0.jre8.jar* - *jdbc* драйвер, который предоставляет возможность подключения к СУБД *SQL Server*
- *ojdbc7-12.1.0.2.jar* - *jdbc* драйвер, который предоставляет возможность подключения к СУБД *Oracle*.
- *postgresql-9.4.1212.jar* - *jdbc* драйвер, который предоставляет возможность подключения к СУБД *PostgreSQL*.
- *icanaccess-4.0.1.jar* - *jdbc* драйвер, который предоставляет возможность подключения к СУБД *Microsoft Access*.
- *sqlite-jdbc-3.16.1.jar* - *jdbc* драйвер, который предоставляет возможность подключения к СУБД *SQLite*.

3.2.3 Нефункциональные требования

- *Надежность.* Обработка ошибок ввода, таких как некорректное задание параметров алгоритмов;
- *Удобство использования.* Навигация по всем разделам;
- *Простота использования.* Система должна быть ориентирована на конечного пользователя не математика.

3.3 Структура системы

Структура программной системы, представленной на рисунке 3.1, состоит из следующих частей [31]:

1. Система окон графического интерфейса, которая предназначена для взаимодействия пользователя с системой. Она, в свою очередь, состоит из следующих подсистем:
 - подсистема построения таблиц с данными, которая отвечает за построение таблицы с исходными данными, таблицы с выбором исходных признаков и их типов (количественные и/или качественные), а также выбор результирующего признака – класса;
 - подсистема настройки метода для оценки точности классификации. На данный момент реализовано два способа: использование обучающего множества, $k \times V$ – блочная кросс – проверка на тестовой выборке ($k \times V$ – *folds cross validation*);
 - подсистема настройки формата чисел (количество десятичных знаков поле запятой);
 - подсистема выбора алгоритмов классификации, как одиночных, так и ансамблевых;
 - подсистема настройки параметров алгоритма классификации;
 - подсистема построения модели классификатора. Выполняет также функцию отображения текущего прогресса построения модели, т.к. некоторые алгоритмы требуют длительного времени для обучения;
 - подсистема результатов классификации, которая отвечает за отображение, визуализацию и сохранение результатов в файл, а также сохранение модели классификатора в файл и классификацию нового объекта на основе построенной модели.

2. Система загрузки исходных данных из файлов с расширениями *.xls*, *.xlsx*, *.csv*, *.arff*, из сети по протоколам *http* и *ftp*, из базы данных (*MySQL*, *Oracle*, *Microsoft Access*, *SQLite*, *PostgreSQL*, *SQL Server*).
3. Система индивидуальных алгоритмов классификации, таких как деревья решений (*CART*, *C4.5*, *ID3*, *CHAID*), нейронная сеть, логистическая регрессия, алгоритм k – взвешенных ближайших соседей.
4. Система ансамблевых алгоритмов классификации (алгоритм *Random Forests*, неоднородный ансамблевый алгоритм, модифицированный неоднородный ансамблевый алгоритм, алгоритм *AdaBoost*, алгоритм *Stacking*).
5. Система генерации алгоритма, которая отвечает за формирование множества индивидуальных алгоритмов классификации, которые впоследствии будут использоваться ансамблевым алгоритмом.
6. Справочная система, которая содержит руководство пользователя, по использованию программного продукта.
7. Система загрузки модели классификатора из файла.
8. Модуль *Data Miner*, функции которого заключаются в выборе оптимальных параметров алгоритмов на основе проведенного эксперимента для нейронных сетей или ансамблевых алгоритмов.
9. Сервер БД. В качестве него может выступать любой сервер (компьютер), на котором развернута СУБД (*MySQL*, *Oracle*, *Microsoft Access*, *SQLite*, *PostgreSQL*, *SQL Server*).
10. *JDBC* – драйвера, написанные на *java*. С помощью них осуществляется взаимодействие системы *ECA* с базами данных.
11. На веб – серверах и *ftp* – серверах хранятся файлы с исходными данными. Доступ к ним осуществляется по протоколам *http* и *ftp* соответственно.

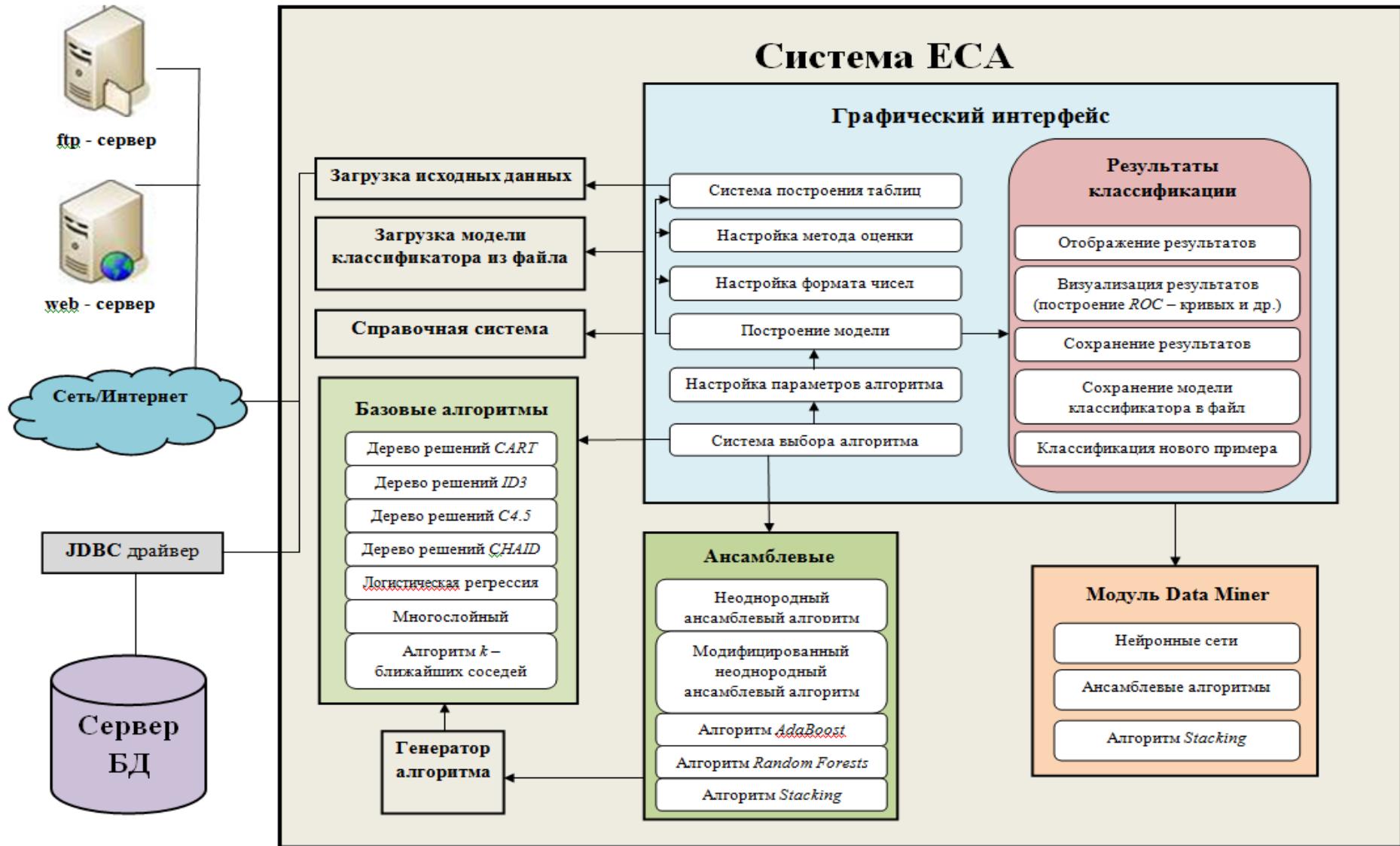


Рисунок 3.1 – Структура программной системы ЕСА

3.4 Реализация программной системы

3.4.1 Предобработка входных данных

Так как исходные данные могут содержать пропущенные значения, во всех алгоритмах, реализованных в программной системе, они фильтруются с использованием следующего принципа:

1. Сначала удаляются все объекты с пропущенным значением класса.
2. Затем для каждого числового атрибута пропущенные заменяются средним арифметическим его значений, а для категориальных атрибутов пропущенные значения заменяются значением с кодом 0. Отметим, что данный подход используется в библиотеке анализа данных *weka*.

3.4.2 Реализация модуля «деревья решений»

В программной системе реализовано четыре алгоритма построения дерева решений, такие как *CART*, *ID3*, *C45*, *CHAID*. Все они могут работать как с числовыми, так и с категориальными атрибутами. При этом выходная переменная класса должна иметь категориальный тип. Отметим также, что в данном модуле реализована возможность настройки случайного дерева решений. Это значит, что при построении дерева решений на каждой итерации выбор наилучшего расщепления будет происходить на основе N атрибутов, отбираемых случайным образом (равновероятно). Число 0 соответствует выбору всех атрибутов.

Далее мы рассмотрим все классы этого модуля. На рисунках 3.2 – 3.3 приведена диаграмма классов модуля деревья решений.

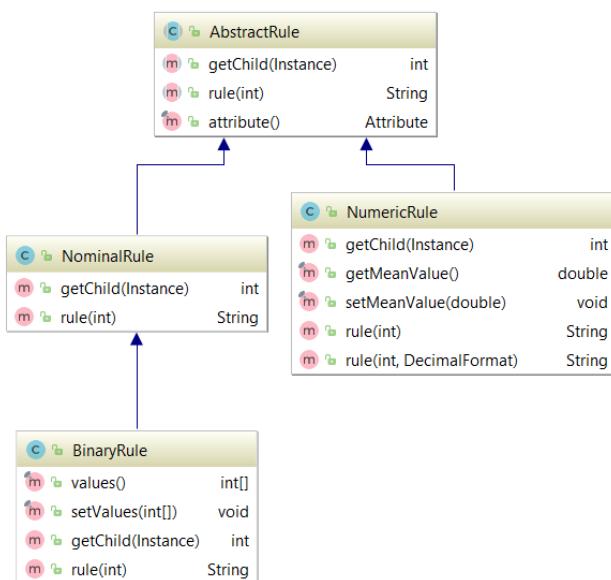


Рисунок 3.2 – Диаграмма классов правил для расщепления вершины

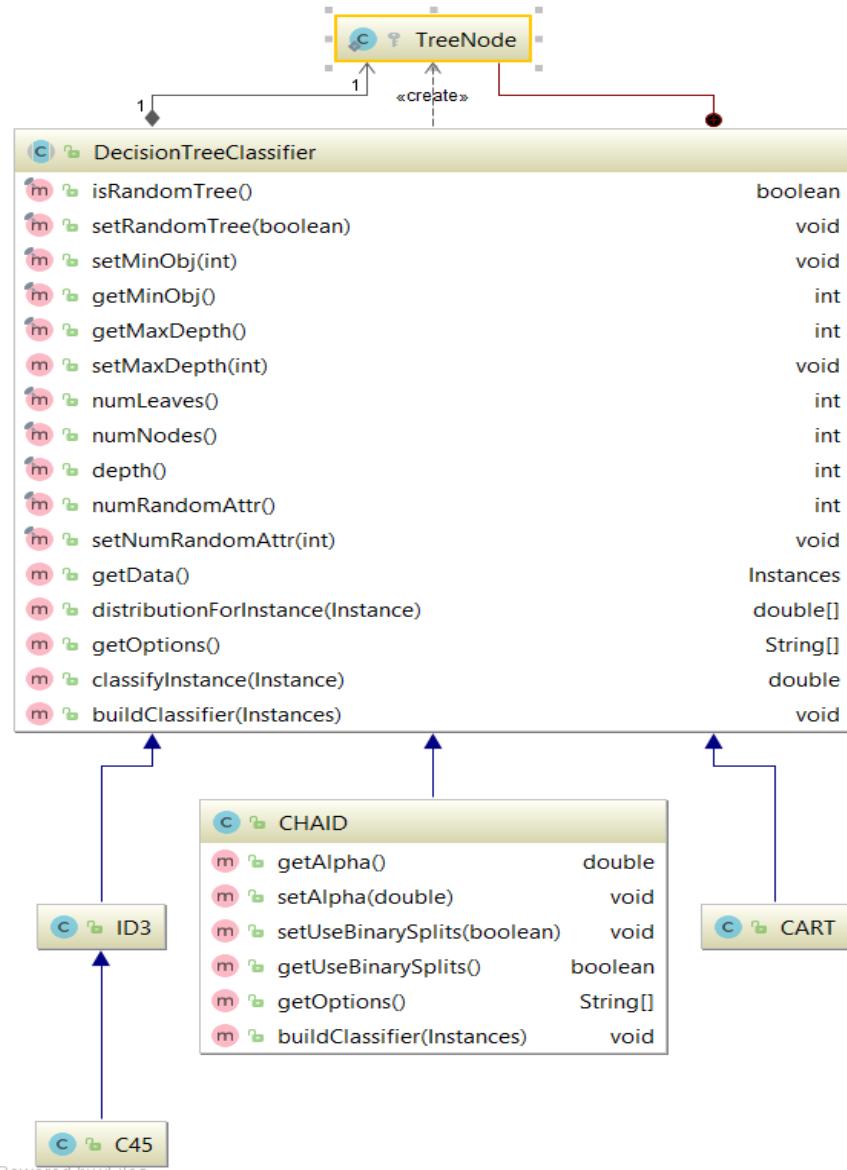


Рисунок 3.3 – Диаграмма классов деревьев решений

Абстрактный класс *AbstractRule* реализует модель правила для расщепления вершины. В таблице 3.1 приведены основные методы этого класса.

Таблица 3.1 – Описание методов класса *AbstractRule*

Название метода	Описание метода
<i>int getChild(Instance obj)</i>	Возвращает индекс дочернего узла, к которому будет отнесен пример <i>obj</i> .
<i>String rule(int i)</i>	Возвращает текстовое представление правила, по которому осуществляется переход к <i>i</i> -ому дочернему узлу дерева.
<i>Attribute attribute()</i>	Возвращает ссылку на атрибут расщепления.

Производный класс *NominalRule* реализует модель расщепления вершины для категориального атрибута.

Производный класс *NumericRule* реализует модель расщепления вершины для числового атрибута. В таблице 3.2 приведены основные методы этого класса.

Таблица 3.2 – Описание методов класса *NumericRule*

Название метода	Описание метода
<code>void setMeanValue(double meanValue)</code>	Задает значение медианы для расщепления.
<code>double getMeanValue()</code>	Возвращает значение медианы для расщепления.
<code>String rule(int i, DecimalFormat fmt)</code>	Возвращает отформатированное текстовое представление правила, по которому осуществляется переход к <i>i</i> -ому дочернему узлу дерева.

Класс *BinaryRule* реализует модель расщепления вершины на два подмножества для категориального атрибута. В таблице 3.3 приведено описание методов этого класса.

Таблица 3.3 – Описание методов класса *BinaryRule*

Название метода	Описание метода
<code>int[] values()</code>	Возвращает массив бинарных значений категориального атрибута.
<code>void setValues(int[] values)</code>	Задает массив бинарных значений категориального атрибута. Длина массива должна соответствовать числу значений категориального атрибута. Если элемент массива с индексом <i>i</i> равен нулю, это означает, что любой объект, имеющий в качестве кодового значения атрибута значение <i>i</i> , будет отнесен в левую подвыборку (дочернюю вершину), в противном случае - в правую.

Внутренний класс ***TreeNode*** реализует модель узла дерева решений. Он содержит такие поля как:

- *Instances objects* – ссылка на объект набора данных ассоциированных с узлом;
- *TreeNode[] child* – массив дочерних узлов;
- *TreeNode parent* – ссылка на родительский узел. Для корневого узла она равна значению *null*;
- *AbstractRule rule* – ссылка на объект правила для расщепления вершины;
- *double classValue* – значение класса узла;
- *int index* – индекс узла;
- *int depth* – глубина узла.

Абстрактный класс ***DecisionTreeClassifier*** реализует модель дерева решений в виде *m*-арного дерева и содержит основные методы для настройки параметров дерева решений (таблица 3.4).

Таблица 3.4 – Описание методов класса *DecisionTreeClassifier*

Название метода	Описание метода
<i>boolean isRandomTree()</i>	Возвращает <i>TRUE</i> , если дерево является случайным, <i>FALSE</i> в противном случае.
<i>void setRandomTree(boolean isRandom)</i>	Устанавливает флаг, который отвечает за то, является ли дерево случайным.
<i>int numRandomAttr()</i>	Возвращает число случайных атрибутов, которые используются при каждом расщеплении вершины.
<i>void setNumRandomAttr(int numRandomAttr)</i>	Задает число случайных атрибутов.
<i>void setMinObj(int minObj)</i>	Задает минимальное число объектов в листе.
<i>int getMinObj()</i>	Возвращает минимальное число объектов в листе.
<i>void setMaxDepth(int maxDepth)</i>	Задает максимальную глубину дерева.
<i>int getMaxDepth()</i>	Возвращает максимальную глубину дерева.
<i>int depth()</i>	Возвращает глубину дерева. <i>Примечание:</i> Необходимо вызывать этот метод после построения дерева решений.

Окончание таблицы 3.4

Название метода	Описание метода
<i>int numLeaves()</i>	Возвращает число листов дерева. <i>Примечание:</i> Необходимо вызывать этот метод после построения дерева решений.
<i>int numNodes()</i>	Возвращает число узлов дерева. <i>Примечание:</i> Необходимо вызывать этот метод после построения дерева решений.
<i>Instances getData()</i>	Возвращает ссылку на объект исходного набора данных.
<i>void buildClassifier(Instances data)</i>	Выполняет построение модели дерева решений.
<i>double classifyInstance(Instance obj)</i>	Выполняет классификацию нового примера. Возвращает номер класса.
<i>double[] distributionForInstance(Instance obj)</i>	Возвращает массив вероятностей классов для нового примера.
<i>String[] getOptions()</i>	Возвращает перечень входных параметров классификатора в виде массива строк.

Производные классы **CART**, **ID3**, **C45** реализуют соответствующие алгоритмы построения деревьев решений. В алгоритме **C4.5** используется критерий *GainRatio*, который вычисляется только в том случае, если число значений категориального атрибута не менее чем $0,3N$, где N – число объектов в узле. В противном случае используется формула 1.12.

Класс **CHAID** реализует модель дерева решений, в котором на каждой итерации выбор наилучшего атрибута для расщепления происходит на основе теста хи-квадрат. В таблице 3.5 приведены основные методы этого класса.

Таблица 3.5 – Описание методов класса **CHAID**

Название метода	Описание метода
<i>void setAlpha(double alpha)</i>	Задает значение уровня значимости α для теста хи-квадрат.
<i>double getAlpha()</i>	Возвращает значение уровня значимости α для теста хи-квадрат.

Окончание таблицы 3.5

Название метода	Описание метода
<code>void setUseBinarySplits(Boolean flag)</code>	Задает булевское значение, которое определяет, будет ли дерево решений бинарным или нет. Если параметр <code>flag = true</code> , то дерево решений будет бинарным. Это означает, что при выборе категориального атрибута вершина будет расщепляться на два подмножества, как и в алгоритме <i>CART</i> .
<code>boolean getUseBinarySplits()</code>	Возвращает <i>TRUE</i> , если дерево является бинарным, <i>FALSE</i> в противном случае.

Таким образом, данная реализация алгоритма *CHAID* позволяет задавать значение уровня значимости α для теста хи-квадрат, а также тип дерева (бинарное/ m - арное).

3.4.3 Реализация модуля «нейронные сети»

В программной системе реализован модуль нейронных сетей типа многослойного персептрона. В качестве алгоритма обучения нейронные сети используют алгоритм обратного распространения ошибки (*back propagation*). Также, перед началом обучения все данные проходят нормализацию, используя минимаксную нормализацию.

Данный модуль поддерживает настройку активационной функции нейронов скрытого слоя, которая может быть одной из следующих:

1. Логистическая.
2. Гиперболический тангенс.
3. Тригонометрический синус.
4. Экспоненциальная.

Нейроны выходного слоя используют логистическую активационную функцию. На рисунке 3.4 приведена *UML* – диаграмма классов, соответствующих активационным функциям нейронов.

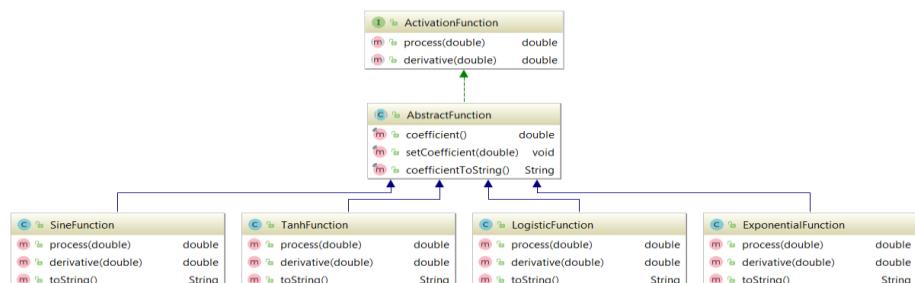


Рисунок 3.4 – Диаграмма классов активационных функций нейронов

Интерфейс ***ActivationFunction*** является базовым для активационных функций. Он, в свою очередь, предоставляет следующие методы, представленные в таблице 3.6.

Таблица 3.6 – Описание методов класса *ActivationFunction*

Название метода	Описание метода
<i>double process (double s)</i>	Вычисляет значение активационной функции от заданного аргумента <i>s</i> .
<i>double derivative(double s)</i>	Вычисляет значение производной активационной функции от заданного аргумента <i>s</i> .

Абстрактный класс ***AbstractActivationFunction*** реализует интерфейс ***ActivationFunction***. Класс представляет собой модель активационной функции с параметром. В таблице 3.7 приведены методы этого класса.

Таблица 3.7 – Описание методов класса *AbstractActivationFunction*

Название метода	Описание метода
<i>double coefficient()</i>	Возвращает значение коэффициента активационной функции.
<i>void setCoefficient(double a)</i>	Задает значение коэффициента активационной функции.
<i>String coefficientToString()</i>	Возвращает значение коэффициента активационной функции в виде строки.

Производные классы ***SineFunction***, ***LogisticFunction***, ***TanhFunction***, ***ExponentialFunction*** реализуют активационные функции нейронов, такие как тригонометрический синус, логистическая, гиперболический тангенс и экспоненциальная соответственно.

Теперь перейдем к рассмотрению реализации самой нейронной сети типа многослойного персептрона. На рисунке 3.5 приведена *UML* - диаграмма классов модуля нейронные сети.



Рисунок 3.5 – Диаграмма классов модуля нейронные сети

Абстрактный класс ***LearningAlgorithm*** предоставляет абстрактные методы для реализации алгоритма обучения нейронной сети.

Класс ***BackPropagation*** наследуется от ***LearningAlgorithm*** и реализует алгоритм обратного распространения ошибки. В таблице 3.8 приведены методы этого класса.

Таблица 3.8 – Описание методов класса *LearningAlgorithm*

Название метода	Описание метода
<code>void train(double[] d, double[] y)</code>	Выполняет корректировку весов связей нейронной сети на основе прогнозных значений сети <i>y</i> и реальных значений <i>d</i> .
<code>String[] getOptions()</code>	Возвращает список параметров алгоритма обучения в виде массива строк.
<code>void initializeWeights()</code>	Задает начальное значение веса для каждой связи, как случайное число из интервала [-0.5, 0.5].
<code>double getLearningRate()</code>	Возвращает коэффициент скорости обучения нейронной сети (аналог шага для градиентных методов поиска минимума).
<code>void setLearningRate(double learningRate)</code>	Устанавливает значение коэффициента скорости обучения, которое должно лежать в интервале (0, 1], иначе будет сгенерировано исключение типа <i>IllegalArgumentException</i> .
<code>double getMomentum()</code>	Возвращает значение коэффициента момента, который используется для повышения эффективности алгоритма.
<code>void setMomentum(double momentum)</code>	Устанавливает значение коэффициента момента, которое должно лежать в интервале [0, 1), иначе будет сгенерировано исключение типа <i>IllegalArgumentException</i> .

Класс *Neuron* представляет собой модель нейрона и предоставляет следующие методы.

Таблица 3.9 – Описание методов класса *Neuron*

Название метода	Описание метода
<i>void setActivationFunction(ActivationFunction function)</i>	Задает активационную функцию нейрона.
<i>ActivationFunction getActivationFunction()</i>	Возвращает ссылку на объект активационной функции нейрона
<i>void setOutValue(double outValue)</i>	Устанавливает выходное значение нейрона
<i>double getOutValue()</i>	Возвращает выходное значение нейрона.
<i>int index()</i>	Возвращает индекс нейрона.
<i>int getType()</i>	Возвращает тип нейрона в виде целочисленной константы, которая может принимать одно из следующих значений: <ul style="list-style-type: none"> - <i>Neuron.IN_LAYER</i> – входной слой; - <i>Neuron.HIDDEN_LAYER</i> – скрытый слой; - <i>Neuron.OUT_LAYER</i> – выходной слой.
<i>void setSumValue(double sumValue)</i>	Задает взвешенную сумму входных значений нейрона.
<i>double getSumValue()</i>	Возвращает взвешенную сумму входных значений нейрона.
<i>void setError(double error)</i>	Задает значение ошибки нейрона (используется алгоритмом обучения).
<i>double getError()</i>	Возвращает значение ошибки нейрона.
<i>double sum()</i>	Вычисляет и возвращает взвешенную сумму входных значений нейрона.
<i>double process()</i>	Вычисляет и возвращает выходное значение нейрона, используя текущее значение суммы значений входов.
<i>double derivative()</i>	Вычисляет и возвращает производную от выходного значения нейрона, используя текущее значение суммы значений входов.

Окончание таблицы 3.9

Название метода	Описание метода
<i>double process(double sum)</i>	Вычисляет и возвращает выходное значение нейрона.
<i>double derivative(double sum)</i>	Вычисляет и возвращает производную от выходного значения нейрона.
<i>void addOutLink(NeuralLink link)</i>	Добавляет объект выходной связи.
<i>void addInLink(NeuralLink link)</i>	Добавляет объект входной связи.
<i>Iterator<NeuralLink> outLinks()</i>	Возвращает объект итератора для доступа к выходным связям нейрона.
<i>Iterator<NeuralLink> inLinks()</i>	Возвращает объект итератора для доступа к входным связям нейрона.

Класс *NeuralLink* представляет собой модель связи между нейронами. Ниже приведено описание методов этого класса.

Таблица 3.10 – Описание методов класса *NeuralLink*

Название метода	Описание метода
<i>Neuron source()</i>	Возвращает объект нейрона источника.
<i>Neuron target()</i>	Возвращает целевой объект нейрона.
<i>void setWeight(double weight)</i>	Устанавливает значение веса связи.
<i>double getWeight()</i>	Возвращает значение веса связи.
<i>void setPreviousCorrect(double previousCorrect)</i>	Устанавливает значение предыдущей коррекции веса (используется алгоритмом обучения).
<i>double getPreviousCorrect()</i>	Возвращает значение предыдущей коррекции веса.

Основной класс *MultilayerPerceptron* представляет собой модель многослойного персептрона. Этот класс может быть использован для решения любых других задач, связанных с применением нейронных сетей. Отметим, что структура скрытого слоя может быть совершенно произвольной, и содержать различное число нейронов в каждом слое. Ниже приведено описание основных методов этого класса.

Таблица 3.11 – Описание методов класса *MultilayerPerceptron*

Название метода	Описание метода
<code>void setHiddenLayer(String layer)</code>	Задает структуру скрытого слоя в виде строки значений, разделенных запятыми.
<code>String getHiddenLayer()</code>	Возвращает структуру скрытого слоя.
<code>void setInLayerNeuronsNum(int inLayerNeuronsNum)</code>	Задает число нейронов во входном слое.
<code>void setOutLayerNeuronsNum(int outLayerNeuronsNum)</code>	Задает число нейронов в выходном слое.
<code>int inLayerNeuronsNum()</code>	Возвращает число нейронов во входном слое.
<code>int outLayerNeuronsNum()</code>	Возвращает число нейронов в выходном слое.
<code>int layersNum()</code>	Возвращает число слоев сети.
<code>int getLinksNum()</code>	Возвращает число связей сети.
<code>int hiddenLayersNum()</code>	Возвращает число скрытых слоев сети.
<code>void setMinError(double min_error)</code>	Задает минимальную допустимую ошибку сети.
<code>double getMinError()</code>	Возвращает минимальную допустимую ошибку сети.
<code>void setMaxIterationsNum(int max_iterations_num)</code>	Задает максимальное число итераций необходимое для обучения сети.
<code>int getMaxIterationsNum()</code>	Возвращает максимальное число итераций необходимое для обучения сети.

Окончание таблицы 3.11

Название метода	Описание метода
<i>ActivationFunction getActivationFunction()</i>	Возвращает объект активационной функции нейронов скрытого слоя.
<i>ActivationFunction getOutActivationFunction()</i>	Возвращает объект активационной функции нейронов выходного слоя.
<i>void setActivationFunction(ActivationFunction function)</i>	Задает объект активационной функции нейронов скрытого слоя.
<i>void setOutActivationFunction(ActivationFunction function)</i>	Задает объект активационной функции нейронов выходного слоя.
<i>void setLearningAlgorithm(LearningAlgorithm algorithm)</i>	Задает объект алгоритма обучения нейронной сети.
<i>LearningAlgorithm getLearningAlgorithm()</i>	Возвращает объект алгоритма обучения нейронной сети.
<i>void build()</i>	Выполняет построение структуры нейронной сети.
<i>void train(double[][] x, double[][] d)</i>	Обучает нейронную сеть, используя в качестве обучающей выборки массивы входных значений x и выходных значений d .
<i>double[] computeOutputVector(double[] x)</i>	Возвращает вектор выходных значений сети на основе вектора входных x входных значений.
<i>IterativeBuilder getIterativeBuilder(double[][] x, double[][] y)</i>	Возвращает объект, предназначенный для пошагового (итеративного) обучения нейронной сети.

NeuralNetwork представляет собой класс, реализующий модель классификатора на основе нейронной сети. Этот класс использует нейронную сеть типа **MultilayerPerceptron**. Объекты этого класса создаются с помощью конструктора, который в качестве аргумента принимает ссылку на объект типа **Instances**. При этом число нейронов во входном слое устанавливается равным числу атрибутов без атрибута класса, а число выходных нейронов устанавливается равным числу классов. В таблице 3.12 приведено описание основных методов этого класса.

Таблица 3.12 – Описание методов класса *NeuralNetwork*

Название метода	Описание метода
<i>Instances getData()</i>	Возвращает объект исходного набора данных.
<i>void buildNetwork()</i>	Выполняет построение структуры нейронной сети.
<i>void buildClassifier(Instances data)</i>	Выполняет нормализацию данных. После этого строит и обучает нейронную сеть.
<i>double classifyInstance(Instance obj)</i>	Выполняет классификацию нового примера. Возвращает номер класса.
<i>double[] distributionForInstance(Instance obj)</i>	Возвращает массив вероятностей классов для нового примера.
<i>String[] getOptions()</i>	Возвращает перечень входных параметров классификатора в виде массива строк.
<i>MultilayerPerceptron network()</i>	Возвращает объект нейронной сети типа многослойного персептрона.

В заключение этого раздела хочется отметить, что данная реализация многослойного персептрона позволяет задавать совершенно любую структуру скрытого слоя.

3.4.4 Реализация алгоритма k – взвешенных ближайших соседей

Данный модуль расположен в пакете `eca.metrics` и содержит основные классы, которые реализуют модель классификатора k – взвешенных ближайших соседей. Обратим внимание на то, что перед началом обучения модели каждый категориальный атрибут, имеющий k ($k > 2$) категорий, преобразуется в $k - 1$ бинарных атрибутов, затем все данные проходят нормализацию, используя минимаксную нормализацию. Также реализована возможность настройки функции расстояния, которая может быть одной из следующих:

5. Евклидово расстояние (используется по умолчанию)
6. Квадрат Евклидова расстояния
7. Манхэттенское расстояние
8. Расстояние Чебышева

Далее подробно рассмотрим все классы этого модуля, приведенные на рис. 3.6 в качестве *UML* – диаграммы.

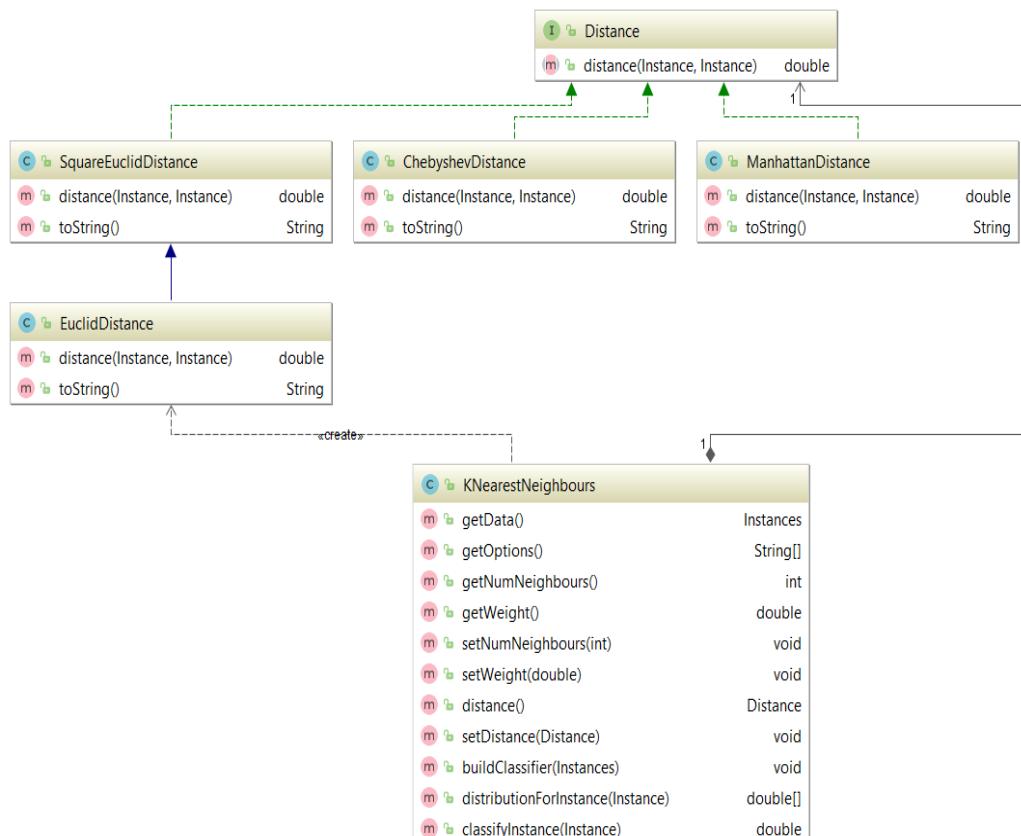


Рисунок 3.6 – Диаграмма классов модуля k – взвешенных ближайших соседей

Главный класс ***KNearestNeighbours*** представляет собой модель классификатора. В таблице 3.13 приведены основные методы этого класса.

Таблица 3.13 – Описание методов класса *KNearestNeighbours*

Название метода	Описание метода
<i>Instances getData()</i>	Возвращает объект исходного набора данных.
<i>String[] getOptions()</i>	Возвращает перечень входных параметров классификатора в виде массива строк.
<i>intgetNumNeighbours</i>	Возвращает число ближайших соседей.
<i>double getWeight()</i>	Возвращает вес ближайшего соседа.
<i>void setNumNeighbours(int numNeighbours)</i>	Задает число ближайших соседей. Генерирует исключение типа <i>IllegalArgumentException</i> , если число соседей меньше одного.
<i>void setWeight(double weight)</i>	Задает вес ближайшего соседа. Генерирует исключение типа <i>IllegalArgumentException</i> , если $weight \notin [0.5,1]$
<i>Distance distance()</i>	Возвращает объект функцию вычисления расстояния между объектами.
<i>void setDistance(Distance distance)</i>	Устанавливает функцию вычисления расстояния между объектами.
<i>void buildClassifier(Instances data)</i>	Преобразует все категориальные атрибуты в бинарные затем выполняет нормализацию данных. После этого строит модель классификатора.
<i>double classifyInstance(Instance obj)</i>	Выполняет классификацию нового примера. Возвращает номер класса.
<i>double[] distributionForInstance(Instance obj)</i>	Возвращает массив вероятностей классов для нового примера.

Интерфейс ***Distance*** представляет собой объект функцию для вычисления расстояния между объектами. Он содержит единственный метод *double distance (Instance o1, In-*

stance o2) для вычисления расстояния между объектами. Этот интерфейс реализуют следующие классы *SquareEuclidDistance*, *EuclidDistance*, *ManhattanDistance*, *ChebychevDistance*. Они реализуют функции: квадрата евклидова расстояния, евклидова расстояния, манхэттенского расстояния, расстояния Чебышева соответственно.

3.4.5 Реализация модуля «ансамблевые алгоритмы»

В данном разделе будет подробно рассмотрена вся инфраструктура модуля ансамблевые алгоритмы, который является ключевым в данной работе. Все реализованные в данной системе ансамблевые алгоритмы, за исключением алгоритма *Random Forests* являются неоднородными. Это означает, что существует возможность задания любой комбинации базовых классификаторов, которые будут использоваться при построении ансамбля. Также можно задавать любые параметры самих базовых моделей.

Все эти алгоритмы требуют длительного времени для обучения, поэтому классы были спроектированы так, чтобы была возможность построения ансамбля моделей посредством взаимодействия с графическим интерфейсом. К примеру, чтобы была возможность просмотра текущего прогресса построения модели. Для этого в реализацию была добавлена возможность пошагового построения модели ансамбля.

Теперь рассмотрим все классы этого модуля по отдельности, а также рассмотрим все особенности их реализации. На рисунке 3.7 приведена UML – диаграмма классов модуля ансамблевые алгоритмы.

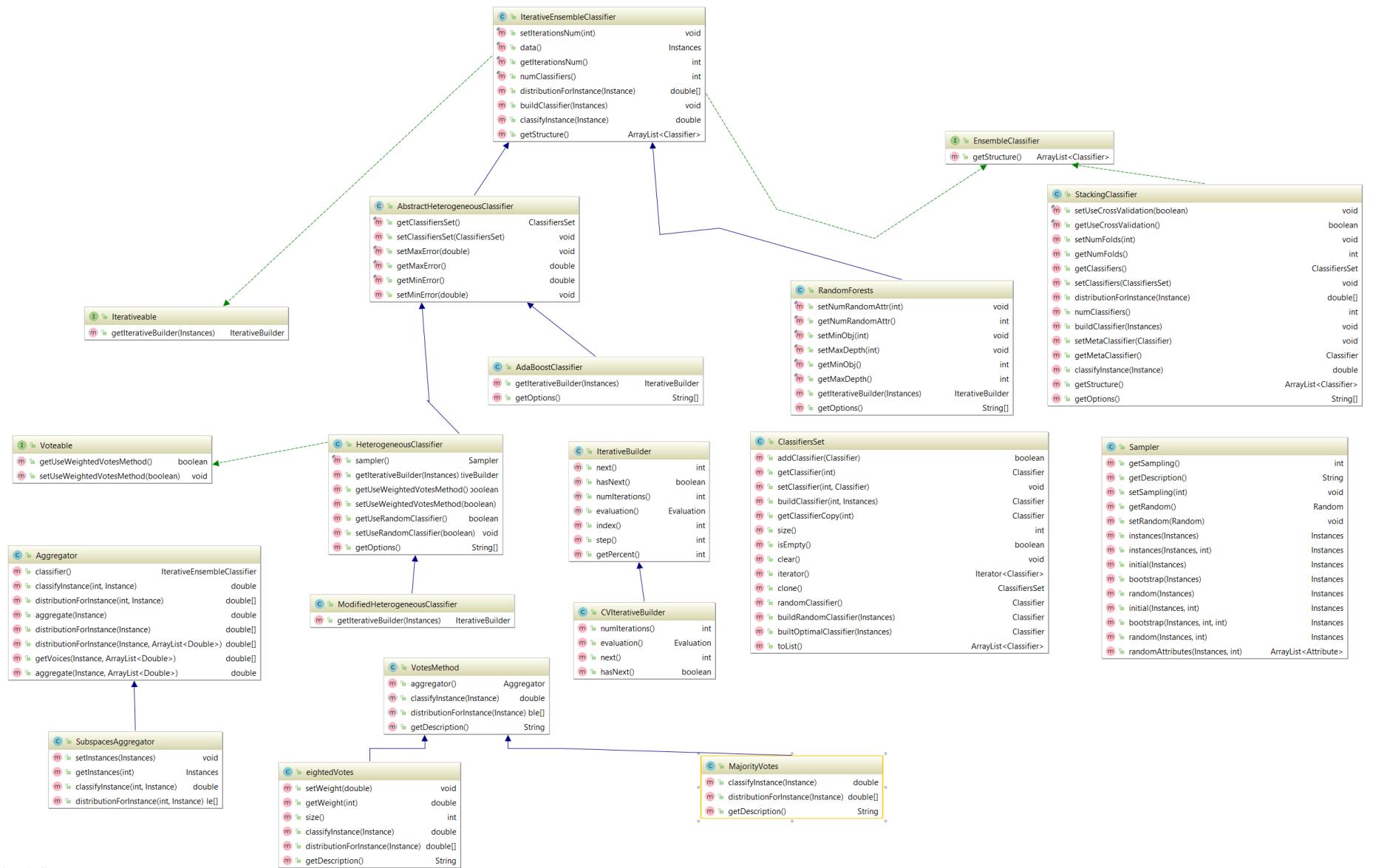


Рисунок 3.7 – Диаграмма классов модуля ансамблевые алгоритмы

Вспомогательные классы:

Главный интерфейс ***EnsembleClassifier*** содержит единственный метод, который возвращает структуру ансамбля классификаторов в виде списка типа *ArrayList<Classifier>*.

Абстрактный класс ***IterativeBuilder*** реализует логику пошагового построения модели классификатора и оценки его точности на основе метода использования всего обучающего множества. Этот класс используется в подсистеме «Построение модели классификатора», реализация которой будет рассмотрена в следующем разделе. Основные методы этого класса приведены в таблице 3.14.

Таблица 3.14 – Описание методов класса *IterativeBuilder*

Название метода	Описание метода
<i>int next()</i>	Выполняет следующий шаг алгоритма и возвращает номер следующей итерации. Генерирует исключение типа <i>NoSuchElementException</i> , если модель полностью построена.
<i>boolean hasNext()</i>	Проверяет, есть ли следующий шаг алгоритма.
<i>int numIterations()</i>	Возвращает число итераций необходимых для построения ансамбля.
<i>Evaluation evaluation()</i>	После того как модель построена, выполняется вычисление основных характеристик классификатора. И возвращается объект типа <i>Evaluation</i> , который содержит информацию о характеристиках построенной модели.
<i>int index()</i>	Возвращает текущий номер итерации.
<i>int step()</i>	Возвращает величину шага между итерациями.
<i>int getPercent()</i>	Возвращает значение, лежащее в интервале от [0, 100], которое соответствует текущему проценту построения модели.

Класс ***CVIterativeBuilder*** реализует логику пошагового построения модели классификатора и оценки его точности, используя метод $k \times V$ – блочной кросс – проверки на тестовой выборке.

Интерфейс ***Iterativeable*** предоставляет методы для создания объектов типа ***IterativeBuilder***. В таблице 3.15 приведены основные методы этого класса.

Таблица 3.15 – Описание методов класса *Iterativeable*

Название метода	Описание метода
<i>IterativeBuilder getIterativeBuilder(Instances data)</i>	Возвращает объект типа <i>IterativeBuilder</i> , который отвечает за пошаговое построение модели классификатора.

Класс коллекция *ClassifiersSet* используется для хранения базовых алгоритмов классификации, которые впоследствии будут использованы при построении ансамбля. Эта коллекция представляет собой структуру данных списка на основе надежного массива. Она, в свою очередь, содержит следующие методы, представленные в таблице 3.16.

Таблица 3.16 – Описание методов класса *ClassifiersSet*

Название метода	Описание метода
<i>boolean addClassifier(Classifier model)</i>	Добавляет классификатор в конец списка. Возвращает <i>false</i> , если <i>model ≠ null</i> , <i>true</i> в противном случае.
<i>Classifier getClassifier(int i)</i>	Возвращает объект классификатора из позиции <i>i</i> .
<i>void setClassifier(int i, Classifier c)</i>	Заменяет классификатор из позиции <i>i</i> на новый классификатор <i>c</i> .
<i>Classifier buildClassifier(int i, Instances data)</i>	Строит и возвращает модель классификатора из позиции <i>i</i> на наборе данных <i>data</i> .
<i>Classifier getClassifierCopy(int i)</i>	Возвращает копию классификатора из позиции <i>i</i>
<i>int size()</i>	Возвращает число классификаторов в коллекции.
<i>boolean isEmpty()</i>	Проверяет коллекцию на пустоту.
<i>void clear()</i>	Выполняет очистку коллекции.
<i>Iterator<Classifier> iterator()</i>	Возвращает ссылку на объект итератора, для обхода коллекции.

Окончание таблицы 3.16

Название метода	Описание метода
<i>ClassifiersSet clone()</i>	Выполняет клонирование объекта
<i>Classifier randomClassifier()</i>	Возвращает ссылку на объект классификатора, который выбирается случайным образом.
<i>Classifier buildRandomClassifier()</i>	Строит и возвращает ссылку на объект классификатора, который выбирается случайным образом.
<i>ArrayList<Classifier> toList()</i>	Возвращает представление коллекции классификаторов в виде объект типа <i>ArrayList<Classifier></i> .

Отметим также, что этот класс есть не что иное, как реализация подсистемы «Генератор алгоритма» (рисунок 3.1).

Классы, реализующие алгоритмы голосования:

Объединение результатов базовых моделей, а также итоговую классификацию примеров реализует базовый класс *Aggregator*, который содержит основные методы, представленные в таблице 3.17.

Таблица 3.17 – Описание методов класса *Aggregator*

Название метода	Описание метода
<i>IterativeEnsembleClassifier classifier()</i>	Возвращает ссылку на объект типа <i>IterativeEnsembleClassifier</i> .
<i>double classifyInstance(int i, Instance obj)</i>	Выполняет классификацию примера <i>obj</i> с использованием классификатора из позиции <i>i</i> . Возвращает номер класса.
<i>double[] distributionForInstance(int i, Instance obj)</i>	Возвращает массив вероятностей классов для нового примера с использованием классификатора из позиции <i>i</i> .

Окончание таблицы 3.17

Название метода	Описание метода
<code>double[] getVoices(Instance obj, ArrayList<Double> weights)</code>	<p>Возвращает массив голосов классификаторов для примера <i>obj</i>.</p> <p><i>Примечание:</i></p> <p>Если <i>weights</i> ≠ null, то используется метод взвешенного голосования, в противном случае метод большинства голосов.</p>
<code>double aggregate(Instance obj)</code>	Выполняет классификацию нового примера с помощью метода большинства голосов. Возвращает номер класса.
<code>double aggregate(Instance obj, ArrayList<Double> weights)</code>	Выполняет классификацию нового примера с помощью метода взвешенного голосования. Возвращает номер класса.
<code>double[] distributionForInstance(Instance obj)</code>	Возвращает массив вероятностей классов для нового примера с использованием метода большинства голосов.
<code>double[] distributionForInstance(Instance obj, ArrayList<Double> weights)</code>	Возвращает массив вероятностей классов для нового примера с использованием метода взвешенного голосования.

Класс-наследник ***SubspacesAggregator*** используется в модифицированном неоднородном ансамблевом алгоритме и реализует логику объединения результатов базовых моделей, а также итоговую классификацию примеров.

В модифицированном неоднородном ансамблевом алгоритме на каждой итерации формируются обучающие подмножества различного признакового описания, и, поэтому, необходимо запоминать эти подмножества в массиве объектов типа *Instances*, где эти объекты соответствуют базовым классификаторам. А при классификации нового примера нужно преобразовывать этот пример, учитывая структуру признакового описания обучающих подмножеств. В таблице 3.18 приведены основные методы этого класса.

Таблица 3.18 – Описание методов класса *SubspacesAggregator*

Название метода	Описание метода
<i>void setInstances(Instances data)</i>	Добавляет объект обучающего подмножества.
<i>Instances getInstances(int i)</i>	Возвращает ссылку на объект обучающего подмножества из позиции <i>i</i> .
<i>double classifyInstance(int i, Instance obj)</i>	Преобразует пример <i>obj</i> с учетом признакового описания обучающего подмножества из позиции <i>i</i> . Затем выполняет классификацию преобразованного примера с использованием классификатора из позиции <i>i</i> . Возвращает номер класса.
<i>double[] distributionForInstance(int i, Instance obj)</i>	Преобразует пример <i>obj</i> с учетом признакового описания обучающего подмножества из позиции <i>i</i> . И затем возвращает массив вероятностей классов для преобразованного примера с использованием классификатора из позиции <i>i</i> .

Абстрактный класс ***VotesMethod*** реализует логику классификации новых примеров для ансамблевых алгоритмов. В таблице 3.19 приведено описание основных методов этого класса.

Таблица 3.19 – Описание методов класса *SubspacesAggregator*

Название метода	Описание метода
<i>Aggregator aggregator()</i>	Возвращает ссылку на объект типа <i>Aggregator</i> .
<i>double classifyInstance(Instance obj)</i>	Выполняет классификацию примера <i>obj</i> . Возвращает номер класса.
<i>double[] distributionForInstance(Instance obj)</i>	Возвращает массив вероятностей классов для нового примера.

Производные классы ***MajorityVotes*** и ***WeightedVotes*** реализуют логику итоговой классификации новых примеров с помощью метода большинства голосов и метода взвешенного голосования соответственно. Отметим, что класс ***WeightedVotes*** также хранит массив весовых коэффициентов для базовых классификаторов, которые впоследствии используются при итоговой классификации.

Классы, реализующие ансамблевые алгоритмы классификации:

Начнем с рассмотрения абстрактного базового класса ***IterativeEnsembleClassifier***. Этот класс реализует логику построения итеративного ансамблевого алгоритма и содержит основные поля, такие как число итераций и массив построенных базовых классификаторов. В таблице 3.20 приведены основные методы этого класса.

Таблица 3.20 – Описание методов класса *IterativeEnsembleClassifier*

Название метода	Описание метода
<i>void setIterationsNum(int numIterations)</i>	Задает число итераций. Генерирует исключение типа <i>IllegalArgumentException</i> , если число итераций меньше одной.
<i>int getIterationsNum()</i>	Возвращает число итераций.
<i>int numClassifiers()</i>	Возвращает число классификаторов в ансамбле.
<i>Instances data()</i>	Возвращает ссылку на исходный набор обучающих данных.
<i>ArrayList<Classifier> getStructure()</i>	Возвращает структуру построенного ансамбля в виде списка базовых классификаторов.
<i>void buildClassifier(Instances data)</i>	Выполняет построение модели классификатора.
<i>double classifyInstance(Instance obj)</i>	Выполняет классификацию нового примера. Возвращает номер класса.
<i>double[] distributionForInstance(Instance obj)</i>	Возвращает массив вероятностей классов для нового примера.

Класс ***RandomForests*** реализует ансамблевый алгоритм «Случайные леса». В таблице 3.21 приведены основные методы этого класса.

Таблица 3.21 – Описание методов класса *RandomForests*

Название метода	Описание метода
<i>int numRandomAttr()</i>	Возвращает число случайных атрибутов, которые используются при каждом расщеплении вершины решений <i>CART</i> .
<i>void setNumRandomAttr(int numRandomAttr)</i>	Задает число случайных атрибутов решений <i>CART</i> .
<i>void setMinObj(int minObj)</i>	Задает минимальное число объектов в листе для дерева решений <i>CART</i> .
<i>int getMinObj()</i>	Возвращает минимальное число объектов в листе для дерева решений <i>CART</i> .
<i>void setMaxDepth(int maxDepth)</i>	Задает максимальную глубину дерева решений <i>CART</i> .
<i>int getMaxDepth()</i>	Возвращает максимальную глубину дерева решений <i>CART</i> .
<i>String[] getOptions()</i>	Возвращает перечень входных параметров классификатора в виде массива строк.
<i>IterativeBuilder getIterativeBuilder(Instances data)</i>	Возвращает ссылку на объект <i>IterativeBuilder</i> для пошагового построения ансамбля.

Абстрактный класс ***AbstractHeterogeneousClassifier*** реализует базовый функционал для семейства неоднородных ансамблевых алгоритмов классификации и содержит методы, представленные в таблице 3.22.

Таблица 3.22 – Описание методов класса *AbstractHeterogeneousClassifier*

Название метода	Описание метода
<i>ClassifiersSet getClassifiersSet()</i>	Возвращает коллекцию базовых классификаторов, которые будут использоваться при построении ансамбля.
<i>void setClassifiersSet(ClassifiersSet set)</i>	Задает коллекцию базовых классификаторов, которые будут использоваться при построении ансамбля.
<i>double getMaxError()</i>	Возвращает значение максимальной допустимой ошибки для базового классификатора.
<i>void setMaxError(double max_error)</i>	Задает значение максимальной допустимой ошибки для базового классификатора.
<i>double getMinError()</i>	Возвращает значение минимальной допустимой ошибки для базового классификатора.
<i>void setMinError(double min_error)</i>	Задает значение минимальной допустимой ошибки для базового классификатора.

Класс *AdaBoostClassifier* реализует стандартный ансамблевый алгоритм *AdaBoost*. Отметим, что в данной реализации перевзвешивание примеров на каждой итерации осуществляется с помощью метода *resampleWithWeights(Random random, double[] weights)* класса *Instances* из библиотеки *weka*, где *weights* – массив распределений вероятностей примеров. Обучающая выборка для следующей итерации формируется с учетом этого распределения. В таблице 3.23 приведены основные методы этого класса.

Таблица 3.23 – Описание методов класса *AdaBoostClassifier*

Название метода	Описание метода
<i>String[] getOptions()</i>	Возвращает перечень входных параметров классификатора в виде массива строк.
<i>IterativeBuilder get IterativeBuilder(Instances data)</i>	Возвращает ссылку на объект <i>IterativeBuilder</i> для пошагового построения алгоритма <i>AdaBoost</i> .

Интерфейс *Voteable* предоставляет методы, представленные в таблице 3.24, для реализации метода взвешенного голосования.

Таблица 3.24 – Описание методов класса *Voteable*

Название метода	Описание метода
<code>void setUseWeightedVotesMethod(Boolean flag)</code>	Устанавливает булевский флаг, который определяет, использовать ли метод взвешенного голосования или нет.
<code>boolean getUseWeightedVotesMethod()</code>	Возвращает признак использования метода взвешенного голосования.

Класс ***HeterogeneousClassifier*** реализует оригинальный неоднородный ансамбль алгоритм. В этом алгоритме возможен случай, когда ни один из базовых классификаторов не удается включить в ансамбль из-за недопустимой ошибки. В таком случае после построения ансамбля будет сгенерировано исключение типа *Exception*. Это относится и к реализации алгоритмов *AdaBoost* и модифицированного неоднородного ансамблевого алгоритма. В таблице 3.25 приведено описание основных методов этого класса.

Таблица 3.25 – Описание методов класса *Voteable*

Название метода	Описание метода
<code>Sampler sampler()</code>	Возвращает объект типа <i>Sampler</i> отвечающий за формирование обучающих выборок на каждой итерации.
<code>String[] getOptions()</code>	Возвращает перечень входных параметров классификатора в виде массива строк.
<code>IterativeBuilder getIterativeBuilder(Instances data)</code>	Возвращает ссылку на объект <i>IterativeBuilder</i> для пошагового построения алгоритма.
<code>void setUseRandomClassifier(boolean flag)</code>	Устанавливает флаг, отвечающий за метод выбора классификатора на каждой итерации. Если <i>flag</i> = <i>true</i> , то используется метод выбора случайного классификатора, в противном случае выбирается оптимальный классификатор минимизирующий ошибку классификации.

Окончание таблицы 3.25

Название метода	Описание метода
<code>boolean getUseRandomClassifier()</code>	Возвращает <code>true</code> , если на каждой итерации используется метод выбора случайного классификатора, <code>false</code> при использовании оптимального классификатора.
<code>void setUseWeightedVotesMethod(boolean flag)</code>	Устанавливает булевский флаг, который определяет, использовать ли метод взвешенного голосования или нет. Если <code>flag = false</code> , то будет использоваться метод большинства голосов.
<code>boolean getUseWeightedVotesMethod()</code>	Возвращает <code>true</code> , если используется метод взвешенного голосования, <code>false</code> - метод большинства голосов.

Класс ***ModifiedHeterogeneousClassifier*** наследуется от класса ***HeterogeneousClassifier*** и реализует модифицированный неоднородный ансамблевый алгоритм. Этот алгоритм в качестве настроек использует точно такие же параметры, как и у неоднородного ансамблевого алгоритма.

В завершении этого раздела приведем описание методов класса ***Sampler***, который отвечает за выбор метода формирования обучающих выборок на каждой итерации неоднородного ансамблевого алгоритма (таблица 3.26).

Таблица 3.26 – Описание методов класса *Sampler*

Название метода	Описание метода
<i>int getSampling()</i>	Возвращает метод формирования обучающей выборки в виде целочисленной константы. Она может принимать одно из следующих: <ul style="list-style-type: none"> – <i>Sampler.INITIAL</i> – используется исходная обучающая выборка; – <i>Sampler.BAGGING</i> – формируется бутстрэп-выборка из исходной обучающей выборки; – <i>Sampler.RANDOM_BAGGING</i> – формируется бутстрэп - выборка случайного размера $n \sim [1, N]$ из исходной обучающей выборки. – <i>Sampler.RANDOM</i> – формируется подвыборка случайного размера $n \sim [1, N]$ из исходной обучающей выборки.
<i>void setSampling(int sampling)</i>	Задает метод формирования обучающей выборки в виде целочисленной константы. Генерирует исключение типа <i>IllegalArgumentException</i> , если передано некорректное недопустимое целочисленное значение.
<i>void setRandom(Random random)</i>	Задает объект типа <i>Random</i> из пакета <i>java.util</i> .
<i>Random getRandom()</i>	Возвращает ссылку на объект типа <i>Random</i> из пакета <i>java.util</i> .
<i>Instances instances(Instances data)</i>	Возвращает выборку, сформированную с помощью заданного метода.
<i>Instances instances(Instances data, int numAttr)</i>	Возвращает выборку, сформированную с помощью заданного метода. Выборка содержит <i>numAttr</i> число случайных атрибутов, которые формируются случайным образом (равновероятно).
<i>Instances initial(Instances data)</i>	Возвращает исходную обучающую выборку.
<i>Instances initial(Instances data, int numAttr)</i>	Возвращает исходную обучающую выборку с <i>numAttr</i> числом случайных атрибутов.

Окончание таблицы 3.26

Название метода	Описание метода
<i>Instances bootstrap(Instances data)</i>	Возвращает бутстрэп-выборку, сформированную из исходной выборки <i>data</i> .
<i>Instances bootstrap(Instances data, int numAttr)</i>	Возвращает бутстрэп-выборку с <i>numAttr</i> числом случайных атрибутов, сформированную из исходной выборки <i>data</i> .
<i>Instances random(Instances data)</i>	Возвращает подвыборку случайного размера, сформированную из исходной выборки <i>data</i> .
<i>Instances random(Instances data, int numAttr)</i>	Возвращает подвыборку случайного размера с <i>numAttr</i> числом случайных атрибутов, сформированную из исходной выборки <i>data</i> .

Таким образом, была подробно рассмотрена вся система классов, реализующих ансамблевые алгоритмы классификации. Хочется отметить, что данный модуль является легко расширяемым.

3.4.6 Реализация подсистемы построения модели классификатора

Большинство реализованных алгоритмов классификации требуют длительного времени для построения, поэтому целесообразно запускать их построение в отдельном рабочем потоке (в фоновом режиме) для того чтобы:

1. Была возможность дальнейшей работы с приложением без блокирования пользовательского интерфейса.
2. Была возможность отображения текущего прогресса построения модели.

Для реализации этих целей используется класс *SwingWorker<T, V>*, который генерирует результат типа *T* и данные о прогрессе типа *V*. В нашем случае необходимо только отображать текущий процент построения модели классификатора, поэтому мы задаем типы параметров обобщения как *Void*. Этот класс содержит основные методы для реализации логики выполнения трудоемкой задачи в фоновом режиме:

- *doInBackground()* – метод, который выполняется в рабочем потоке. В этом методе и происходит построение модели классификатора. В случае, если модель классификатора строится пошагово, то на каждой итерации вызывается метод *setProgress* для обновления внутреннего свойства *progress*. Также необходимо назначить слушателя на событие изменения значения этого свойства с помощью метода *addPropertyChangeListener*. Таким образом, при очередном изменении значения

свойства *progress* этот слушатель будет обновлять пользовательский интерфейс. Для отображения прогресса на пользовательском интерфейсе используется класс *JProgressBar* из библиотеки *swing*. В листинге 3.1 приведена реализация метода *doInBackground()*.

Листинг 3.1. Реализация метода *doInBackground()*

```
// Полная реализация этого класса приведена в приложении

@Override
protected Void doInBackground() {
    try {
        while (!isCancelled() && builder.hasNext()) {
            //выполняем очередную итерацию алгоритма
            builder.next();
            //обновляем значение свойства progress
            setProgress(builder.procent());
        }
    } catch (Exception e) {
        //запоминаем признак успешного построения модели
        isSuccess = false;
        errorMessage = e.getMessage(); //запоминаем текст ошибки
    }
    setProgress(100); //устанавливаем значения прогресса равным 100
    if (!isCancelled()) {
        try {
            Thread.sleep(300);
        } catch(InterruptedException e) {
        }
    }
}

return null;
}
```

- *done()* – этот метод выполняет завершающее изменение пользовательского интерфейса. В нашем случае закрывает диалоговое окно построения модели.

В завершении этого раздела следует отметить, что в реализации предусмотрена возможность в любой момент отменить построение модели классификатора.

3.4.7 Реализация подсистем загрузки и сохранения моделей классификатора

В программной системе *ECA* реализована возможность загрузки и сохранения моделей классификаторов в текстовые файлы. Так как структуры некоторых моделей очень сложны, было принято решение сохранять объекты этих модели в сериализованном виде (в виде потока байтов). Для этого достаточно всего лишь реализовать интерфейс *Serializable* из пакета *java.io*. Для выполнения операций записи и чтения потока байтов из файлов используются классы потокового ввода *ObjectInputStream* и потокового вывода *ObjectOutputStream*. В листинге 3.2 приведена реализация класса *SerializedObject*, который реализует логику загрузки и сохранения моделей классификаторов в текстовые файлы.

Листинг 3.2. Реализация класса *SerializedObject*

```
public class SerializedObject implements Serializable {  
  
    public static void serialize(Object obj, String fileName)  
        throws Exception {  
        try (ObjectOutputStream strm =  
            new ObjectOutputStream(new FileOutputStream(fileName))) {  
            //сохраняет сериализованный объект в файл  
            strm.writeObject(obj);  
        }  
        catch (Exception e) {  
            throw new Exception(e);  
        }  
    }  
  
    public static Object deserialize(String fileName)  
        throws Exception {  
        Object obj = null;  
        try (ObjectInputStream strm =  
            new ObjectInputStream(new FileInputStream(fileName))) {  
            //выполняет чтение объекта из файла  
            obj = strm.readObject();  
        }  
        catch (Exception e) {  
            throw new Exception(e);  
        }  
        return obj;  
    }  
}
```

Метод *serialize* выполняет сохранение сериализованного объекта *obj* в файл с именем *fileName*, а метод *deserialize* выполняет чтение потока байтов из файла с именем *fileName*.

3.4.8 Реализация подсистемы загрузки исходных данных

Как уже отмечалось ранее, программная система *ECA* предоставляет возможность загрузки исходных данных из различных источников таких как:

- файлы форматов *.xls*, *.xlsx*, *.csv*, *.arff*;
- сеть/интернет с использованием протоколов *http* и *ftp*;
- базы данных (*MySQL*, *PostgreSQL*, *SQL Server*, *Microsoft Access*, *Oracle*, *SQLite*).

Для загрузки данных из файлов форматов *.xls*, *.xlsx*, *.csv*, *.arff* реализован класс *DataLoader*, расположенный в пакете *eca.core.converters*. Он предоставляет единственный метод *getDataSet*, который возвращает объект данных типа *Instances*, которые были загружены из файла *file*. Для реализации логики загрузки исходных данных из файлов форматов *.xls* и *.xlsx* был разработан класс *XLSLoader*. В реализации этого класса к входным данным предъявляются следующие требования:

- допускаются ячейки следующих форматов: числовые форматы, дата, текстовый, булевский;

- данные не должны содержать посторонние записи;
- данные не должны содержать пустые столбцы;
- каждый столбец должен содержать данные одного типа.

При этом в исходных данных могут присутствовать пропущенные значения. В этом случае, объекты, содержащие пропуски, не будут удаляться из исходных данных. Для загрузки исходных данных из файлов форматов *.csv* и *.arff* используются классы *CSVLoader* и *ArffLoader*, взятые из библиотеки *weka*.

Для реализации логики загрузки исходных данных из сети был разработан класс *DataLoaderImpl*, который расположен в пакете *eca.net*. Для доступа к удаленным ресурсам используется стандартный класс *URLConnection*. Он, в свою очередь, возвращает ссылку на объект потокового ввода типа *InputStream*, который привязан к ресурсу. Этот объект используется для получения содержимого ресурса.

Для реализации логики загрузки исходных данных из базы данных был разработан большой модуль, все классы которого расположены в пакете *eca.jdbc*. В данном модуле реализована возможность подключения к любой базе данных (из списка доступных СУБД) с помощью соответствующих *jdbc* драйверов, и, впоследствии, получение из нее выборок с помощью запросов типа *SELECT*. Рассмотрим все классы этого модуля по отдельности. На рисунке 3.8 приведена диаграмма классов этого модуля.

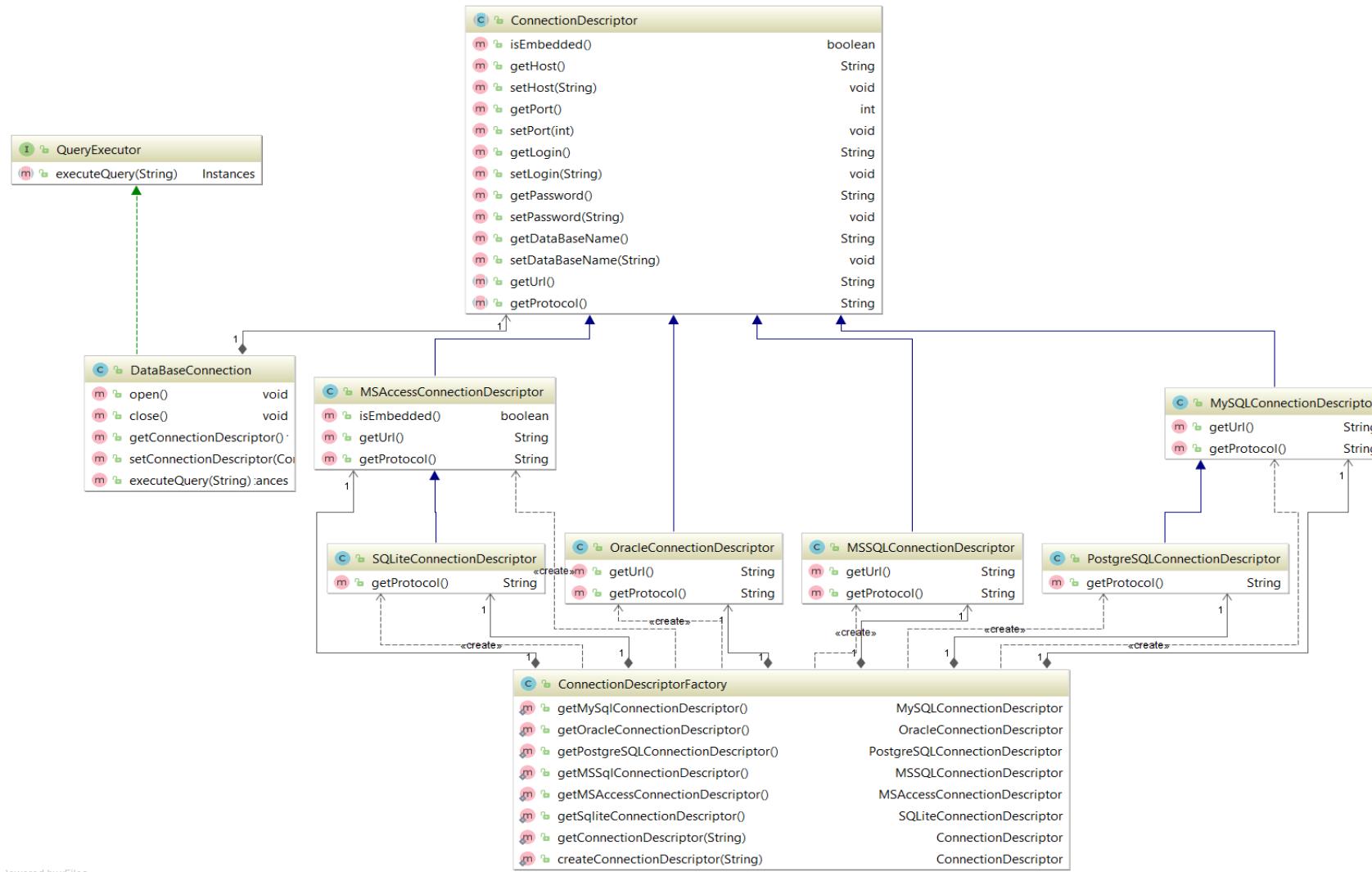


Рисунок 3.8 – Диаграмма класса модуля загрузки исходных данных из БД

Интерфейс *QueryExecutor* предназначен для реализации логики получения выборки исходных данных из БД с помощью *SELECT* запросов. Он содержит единственный метод *executeQuery*, который возвращает ссылку на объект типа *Instances*, представляющий собой результат соответствующего запроса к БД. Этот интерфейс реализует класс *DataBaseConnection*, основные методы которого приведены в таблице 3.27.

Таблица 3.27 – Описание методов класса *DataBaseConnection*

Название метода	Описание метода
<i>void open()</i>	Открывает соединение с базой данных.
<i>void close() throws SQLException</i>	Закрывает соединение с базой данных.
<i>Instances executeQuery(String query) throws SQLException</i>	Выполняет запрос к базе данных и возвращает результат запроса в виде объекта типа <i>Instances</i> .
<i>void setConnectionDescriptor(ConnectionDescriptor connectionDescriptor)</i>	Задает дескриптор соединения с базой данных.
<i>ConnectionDescriptor getConnectionDescriptor()</i>	Возвращает ссылку на объект типа <i>ConnectionDescriptor</i> , в котором описаны параметры соединения с базой данных.

Абстрактный базовый класс *ConnectionDescriptor* предназначен для хранения параметров соединения с конкретной СУБД. В таблице 3.28 приведены основные методы этого класса.

Таблица 3.28 – Описание методов класса *ConnectionDescriptor*

Название метода	Описание метода
<i>Boolean isEmbedded()</i>	Возвращает <i>true</i> , если СУБД является встроенной, <i>false</i> , если серверной.
<i>void setHost(String url)</i>	Задает имя или <i>IP</i> – адрес хоста, на котором запущена БД.

Окончание таблицы 3.28

Название метода	Описание метода
<i>String getHost()</i>	Возвращает имя или <i>IP</i> – адрес хоста, на котором запущена БД.
<i>void setPort(int port)</i>	Задает номер порта.
<i>void getPort()</i>	Возвращает номер порта.
<i>String getUrl()</i>	Возвращает <i>url</i> – строку подключения к БД.
<i>void set DataBaseName(String DataBaseName)</i>	Задает имя БД.
<i>String getDataBaseName()</i>	Возвращает имя БД
<i>String getProtocol()</i>	Возвращает префикс <i>url</i> – строки соединения (например, <i>jdbc:mysql://</i>)
<i>void setLogin(String login)</i>	Задает имя пользователя в БД.
<i>String getLogin()</i>	Возвращает имя пользователя БД.
<i>void setPassword(String password)</i>	Задает пароль пользователя в БД.
<i>String getPassword()</i>	Возвращает пароль пользователя в БД.

Для реализации хранения параметров подключения для конкретной СУБД были реализованы соответствующие подклассы класса ***ConnectionDescriptor***. Объекты этих классов создаются с помощью класса ***ConnectionDescriptorFactory***, который реализован на основе известных паттернов проектирования, таких как *factory* и *singleton*.

3.5 Выводы по главе

В данной главе подробно описано проектирование и реализация программной системы *ECA* и выполнено следующее:

1. Приведены основные функциональные возможности программной системы *ECA*.
2. Спроектирована структура системы, включающая следующие основные блоки: загрузка исходных данных, загрузка модели классификатора из файла, графический интерфейс, базовые алгоритмы классификации, ансамблевые алгоритмы классификации, модуль *Data Miner*, справочная система.
3. Выполнена и описана реализация программной системы, приведены *UML* - диаграммы классов с подробным описанием.
4. Реализованы базовые алгоритмы классификации: деревья решений *CART*, *C4.5*, *ID3*, *CHAID*, нейронная сеть, логистическая регрессия, алгоритм k – взвешенных ближайших соседей.
5. Реализованы ансамблевые алгоритмы *AdaBoost*, *Random Forests*, *Stacking*.
6. Реализованы оригинальные ансамблевые алгоритмы: неоднородный ансамблевый алгоритм и модифицированный неоднородный ансамблевый алгоритм.

4 Описание пользовательского интерфейса ECA

4.1 Установка программы

Программа реализована на языке программирования *java* и поэтому для ее установки и корректной работы требуется установленная на компьютере версия *jre* не ниже 1.8.0 (рекомендуемая версия *jre1.8.0_91*). После этого необходимо запустить инсталлятор *ECASetup.exe* (рисунок 4.1).

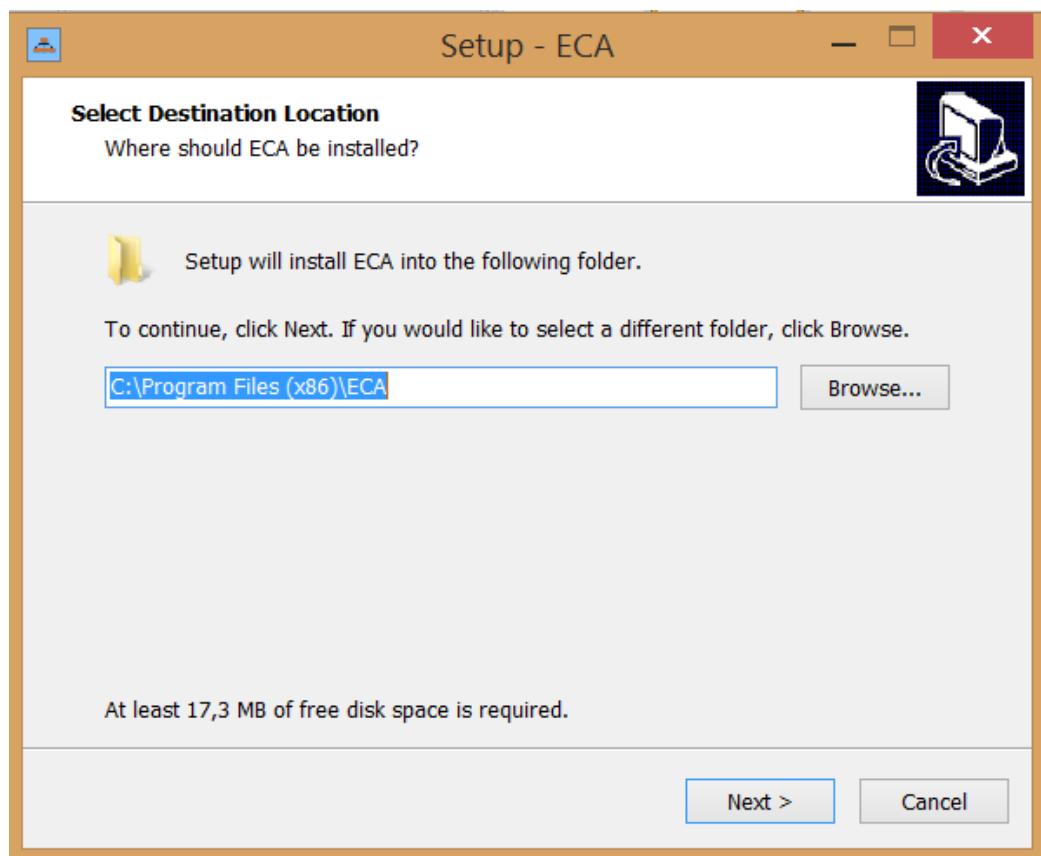


Рисунок 4.1 – Установка программы ECA

Здесь необходимо выбрать папку для установки программы и затем нажать на кнопку «*Next*». В результате появится следующее окно (рисунок 4.2).

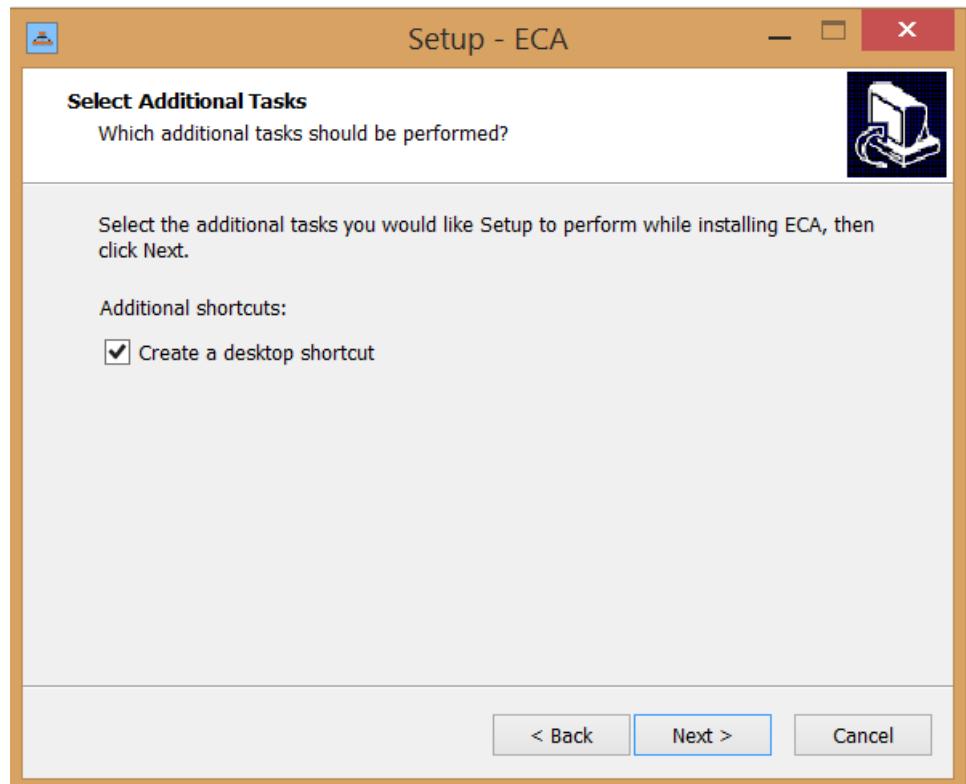


Рисунок 4.2 – Установка программы ЕСА (создание ярлыка на рабочем столе)

На следующем шаге вы можете установить галочку на пункте «*Create a desktop shortcut*», после этого необходимо нажать на кнопку «*Next*» (рисунок 4.3).

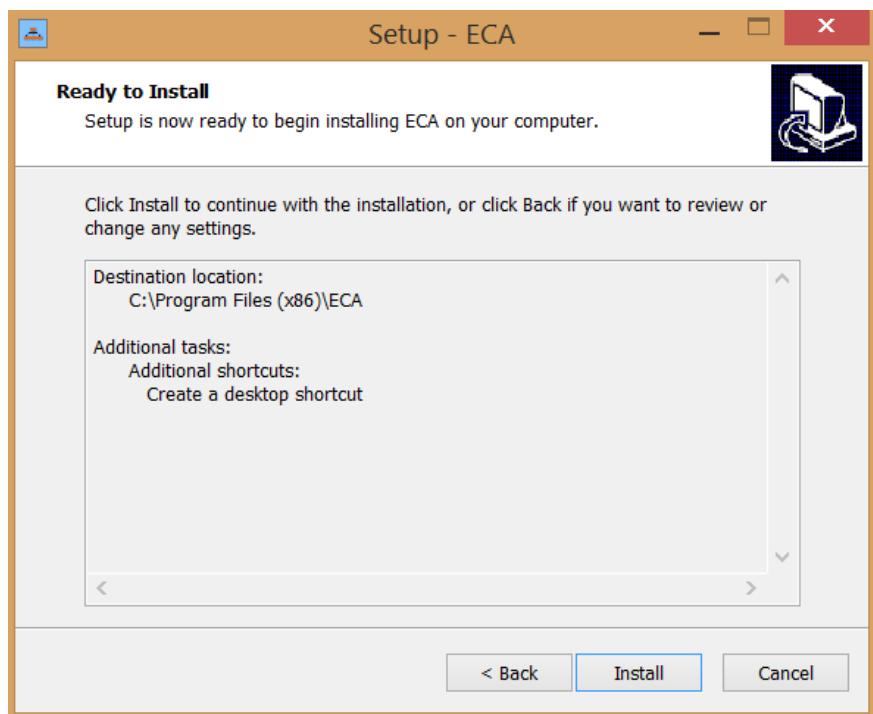


Рисунок 4.3 – Установка программы ЕСА

Теперь необходимо нажать на кнопку «*Install*», и программа со всеми необходимыми библиотеками установится на ваш компьютер.

4.2 Подготовка данных

4.2.1 Загрузка данных из файла

В системе реализована возможность загрузки данных, имеющих следующие форматы: *.xls*, *.xlsx*, *.csv*, *.arff*. Для того чтобы загрузить набор данных из файла, необходимо выбрать пункт меню *Файл>Открыть*. Затем выбрать из появившегося диалогового окна файл с данными и нажать на кнопку «*Open*» (рисунок 4.4).

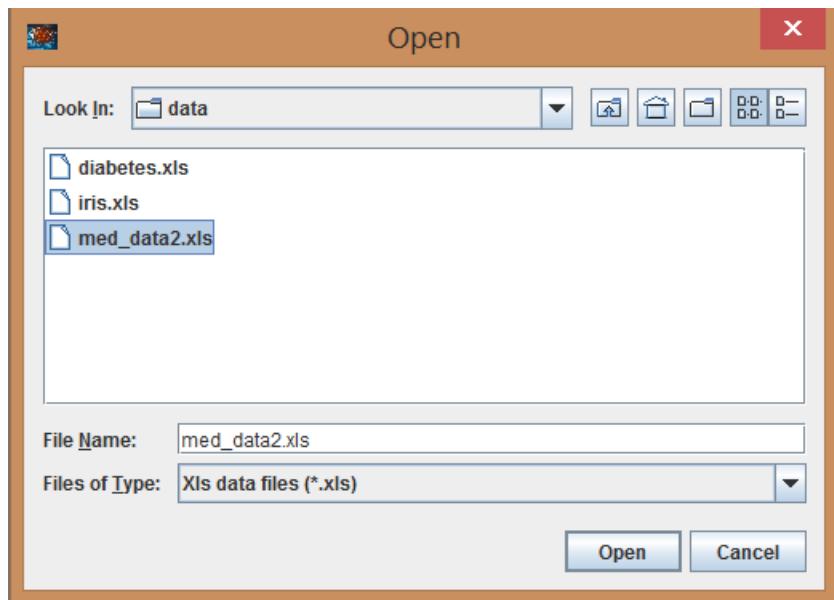


Рисунок. 4.4 - Выбор файла с данными

По умолчанию, формат файла, который возможен для открытия – *.xls*. Однако, можно ставить фильтры на отображение только файлов форматов *.xlsx*, *.csv* и *.arff*. Если в файле с данными имеются ошибки, например дублирование имен атрибутов, то будет выдано сообщение об ошибке.

4.2.2 Загрузка данных из сети

Для того чтобы загрузить набор данных из сети по протоколам *http* или *ftp* необходимо выбрать пункт меню *Файл>Загрузка данных из сети*. (рисунок 4.5)

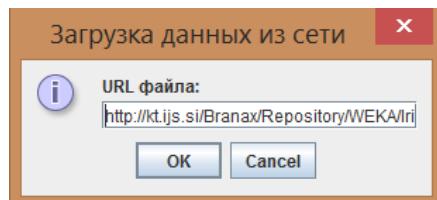


Рисунок 4.5 - Загрузка данных из сети

Затем, в появившемся окне ввести адрес файла с данными, который должен быть в формате *.csv*, *.arff*, *.xls*, *.xlsx*, и нажать на кнопку «*OK*».

4.2.3 Загрузка данных из базы данных

В программной системе также имеется возможность удаленного подключения к любой базе данных с помощью *JDBC* драйвера и загрузки из нее набора данных. Данная версия программы поддерживает работу с такими СУБД как *MySQL*, *Oracle*, *PostgreSQL*, *Microsoft Access*, *SQL Server*, *SQLite*. Допускается загрузка данных из таблиц, имеющих следующие типы столбцов: *CHAR*, *VARCHAR*, *BOOLEAN*, *BIT*, *DOUBLE*, *FLOAT*, *INT*, *BIGINT*, *SMALLINT*, *MEDIUMINT*, *TINYINT*, *NUMERIC*, *DECIMAL*, *REAL*, *DATE*, *TIME*, *TIMESTAMP*. При этом типы столбцов, такие как *CHAR*, *VARCHAR*, *BOOLEAN*, *BIT* трактуются как категориальные атрибуты, а типы столбцов типа *DATE*, *TIME*, *TIMESTAMP* трактуются как атрибуты типа даты и времени.

Для того, чтобы подключиться к базе данных необходимо выбрать пункт меню *Файл>Подключиться к базе данных*. Затем в появившемся диалоговом окне (рисунок 4.6) выбрать СУБД из выпадающего списка и ввести следующие параметры подключения:

- имя или *IP* – адрес хоста, на котором запущена база данных (в зависимости от выбранной СУБД задается по умолчанию);
- номер порта (в зависимости от выбранной СУБД задается по умолчанию);
- имя базы данных, которая уже существует (в зависимости от выбранной СУБД задается по умолчанию);
- имя пользователя;
- пароль.

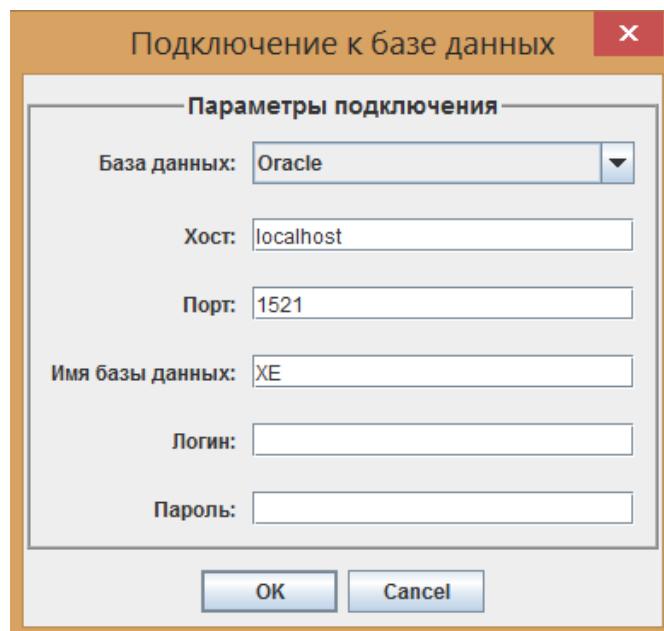


Рисунок 4.6 - Диалоговое окно подключения к БД

Далее после успешного подключения к базе данных появиться диалоговое окно для формирования наборов данных с помощью структурированного языка *SQL* – запросов, а именно запросов типа *SELECT* (рисунок 4.7).

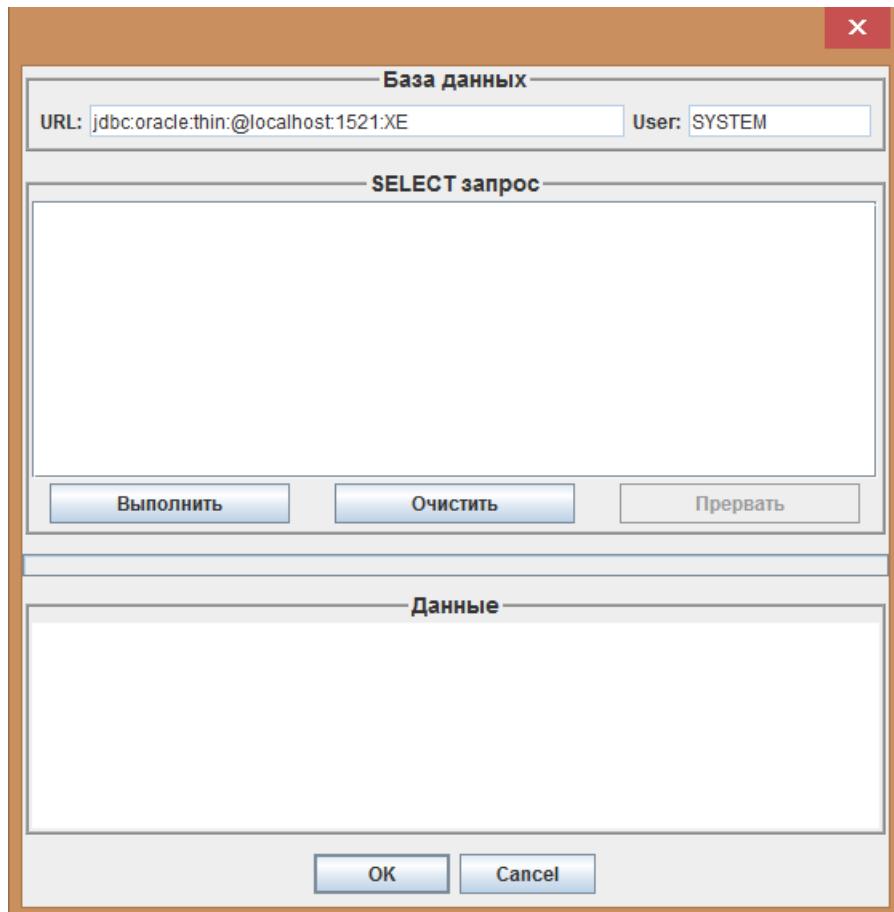


Рисунок 4.7 - Диалоговое окно для написания запросов к БД

Здесь *URL* – строка параметров подключения, в которой сначала следует *jdbc:oracle:thin:@*. Это название протокола соединения, за которым следуют хост и порт подключения, на которых запущена база данных. В нашем случае это *localhost* с портом по умолчанию *1521*. Следующая часть – это имя базы данных, которая уже существует в *Oracle*, в нашем случае это база данных с именем *XE*.

В качестве примера, введем в текстовое поле следующий запрос «*SELECT *FROM IRIS*» и нажмем на кнопку «*Выполнить*». При успешном выполнении запроса результирующий набор данных «*Iris*» добавиться в список с названием «*Данные*» (рисунок 4.8). Затем, чтобы загрузить сформированные наборы данных, необходимо нажать на кнопку «*OK*».

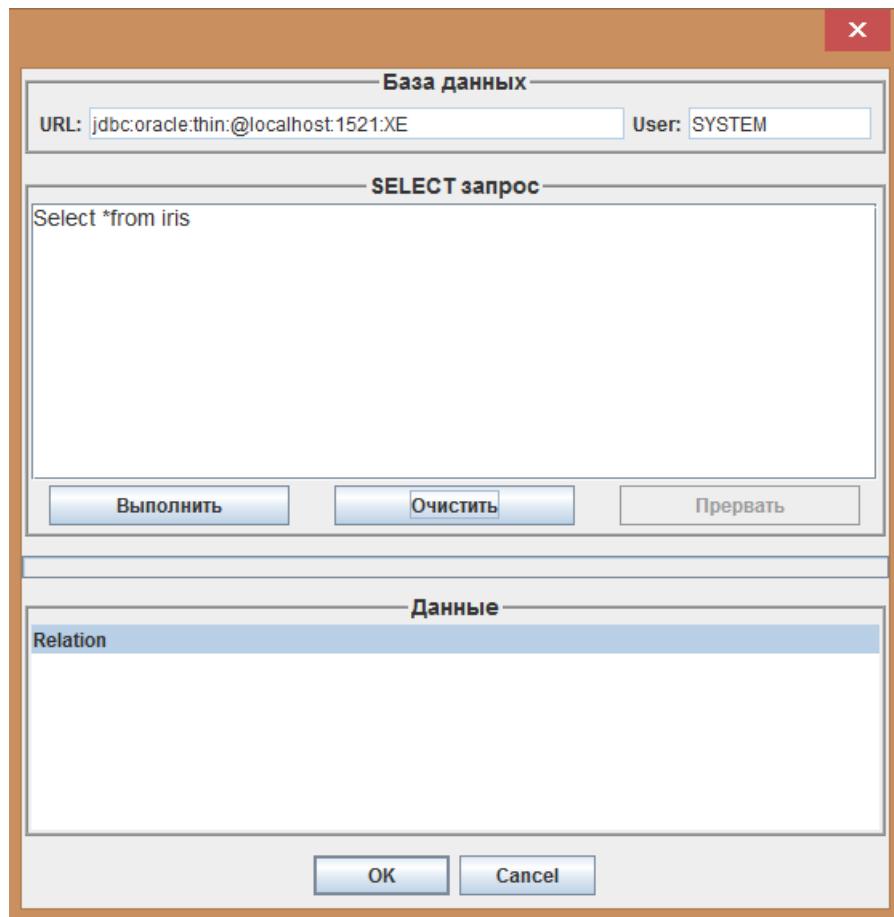


Рисунок 4.8 - Результат запроса *SELECT*

Также отметим, что присутствует возможность в любой момент прервать выполнение запроса к БД. Для этого служит кнопка «*Прервать*». Если вы хотите очистить поле с текстом запроса, то необходимо нажать на кнопку «*Очистить*».

4.2.4 Работа с таблицами данных

После загрузки набора данных, в главном окне программы появится внутренний фрейм, содержащий основную информацию о данных (название, количество объектов, количество атрибутов), таблицу с данными, таблицу с выбором атрибутов и их типов, а также выпадающий список для выбора результирующего атрибута класса. Отметим, что в программе присутствует возможность одновременной работы с несколькими листами данных. Для того чтобы переключиться на другой фрейм с данными, необходимо выбрать пункт меню *Окна*, затем в появившемся подменю выбрать интересующий набор данных (рисунок 4.9).

Информация о данных

Название: soybean Число объектов: 683 Число атрибутов: 36

Выбранные атрибуты

№	Атрибут	Тип
1	date	Категориальный
2	plant-stand	Категориальный
3	precip	Категориальный
4	temp	Категориальный
5	hail	Категориальный
6	crop-hist	Категориальный
7	area-damaged	Категориальный
8	severity	Категориальный
9	seed-tmt	Категориальный
10	germination	Категориальный
11	plant-growth	Категориальный
12	leaves	Категориальный
13	leaf-spots-halo	Категориальный
14	leaf-spots-marg	Категориальный
15	leaf-spot-size	Категориальный
16	leaf-shred	Категориальный
17	leaf-malf	Категориальный
18	leaf-mild	Категориальный
19	stem	Категориальный
20	lodging	Категориальный
21	stem-cankers	Категориальный
22	canker-lesion	Категориальный
23	frutting-bodies	Категориальный
24	external-decay	Категориальный
25	mycelium	Категориальный
26	int-discolor	Категориальный
27	sclerotia	Категориальный
28	fruit-pods	Категориальный
29	fruit-spots	Категориальный
30	seed	Категориальный
31	mold-growth	Категориальный
32	seed-discolor	Категориальный
33	seed-size	Категориальный

Выбранный класс

class

Рисунок 4.9 - Вид фрейма с данными

Над таблицей с данными можно совершать различные операции редактирования такие как:

- изменение значения ячейки;
- удаление выбранного объекта;
- удаление выбранных объектов;
- очистка данных;
- удаление объектов с пропусками;
- изменение значений атрибута;
- добавление нового объекта;
- изменение типа и размера шрифта таблицы;
- копирование данных в системных буфера обмена.

Для этого необходимо щелкнуть по таблице с данными правой кнопкой мыши, а затем из появившегося всплывающего меню выбрать интересующую операцию (рисунок 4.10).

The screenshot shows the ECA software interface. At the top, there's a menu bar with 'Файл', 'Классификаторы', 'Data Miner', 'Настройки', 'Сервис', 'Окна', and 'Справка'. Below the menu is a title bar 'Информация о данных' with 'Название: soybean', 'Число объектов: 683', and 'Число атрибутов: 36'. A table titled 'Таблица с данными' displays 37 rows of data with columns like '№', 'date', 'plant-stand', etc. Overlaid on the table is a context menu with options: 'Выбор шрифта', 'Автомасштабирование', 'Удалить выбранный объект', 'Добавить объект', 'Очистка', 'Удалить объекты с пропусками', and 'Замена значений атрибута'. To the right of the table is a 'Выбранные атрибуты' panel listing 33 attributes with checkboxes and dropdowns for their types. At the bottom right is a 'Выбранный класс' section with a dropdown set to 'class'.

Рисунок 4.10 - Всплывающее меню с операциями редактирования данных

Для того чтобы изменить значения какого либо атрибута (например все значения *A* на *B*) необходимо выбрать любую ячейку, соответствующую этому атрибуту в таблице с данными, затем во всплывающем меню выбрать пункт «Замена значений атрибута» и в появившемся диалоговом окне ввести старое и новое значения атрибута (рисунок 4.11).

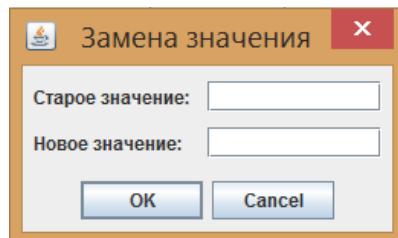


Рисунок 4.11 - Замена значения атрибута

Далее отметим, что также присутствует возможность изменения имени атрибута, для этого необходимо щелкнуть по таблице с атрибутами правой кнопкой мыши, затем из появившегося всплывающего меню выбрать пункт «Переименовать атрибут», и в появившемся диалоговом окне ввести новое уникальное имя атрибута (рисунок 4.12).

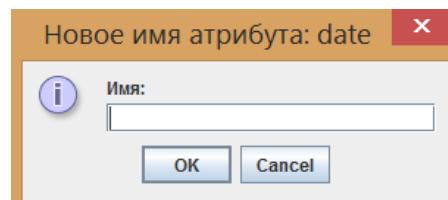


Рисунок 4.12 - Переименование атрибута

На панели «*Выбранные атрибуты*» вы можете выбирать атрибуты, которые будут участвовать при построении модели классификатора, а так же их типы, такие как **числовой, дата и время, категориальный**. При этом, тип дата и время обрабатывается в системе как числовой (дата в миллисекундах). Он вынесен в отдельную подкатегорию для того, чтобы пользователь имел возможность ввода значений даты в удобном формате. Кнопка с названием «*Выбранные атрибуты*» предназначена для выбора всех атрибутов, а кнопка «*Сброс*» для сброса всех изменений, произошедших с таблицей атрибутов, таких как выбор атрибутов и выбор их типов.

При щелчке правой кнопки мыши по фрейму с данными, появится всплывающее меню со следующими операциями (рисунок 4.13):

- изменение названия данных;
- выбор цвета фона;
- сохранение текущего набора данных в файл с расширениями *.xls*, *.csv*, *.xlsx*, *arff*.

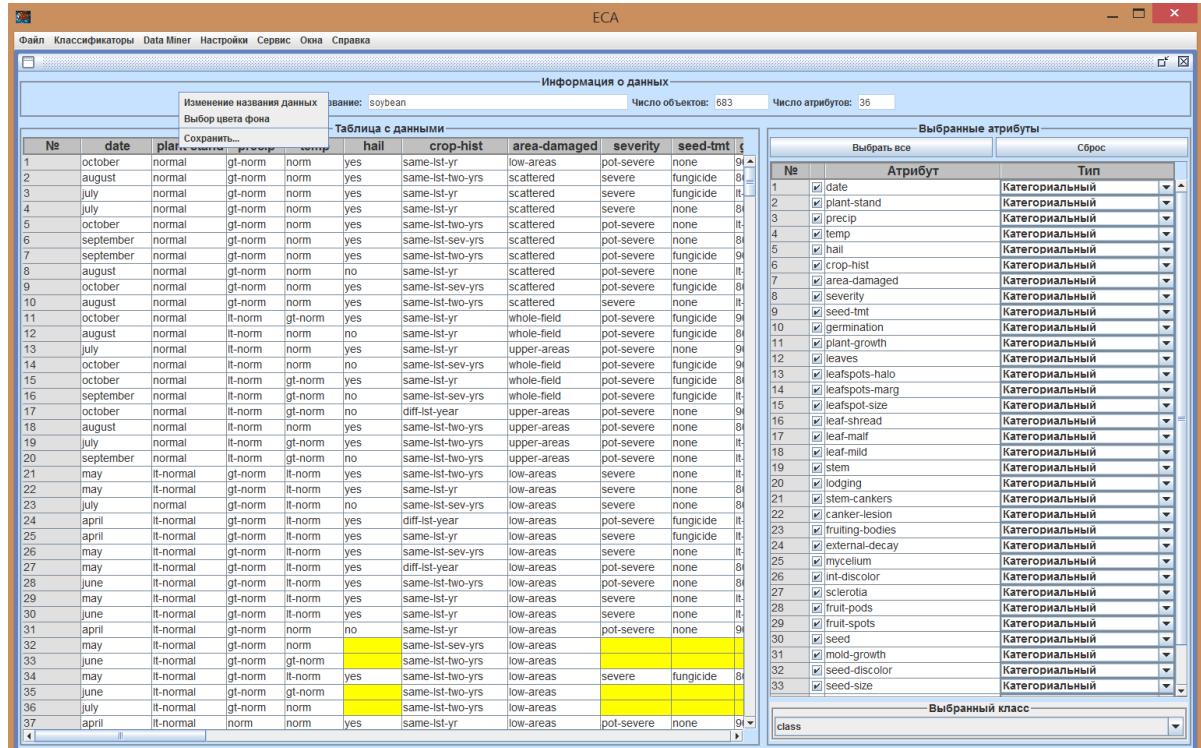


Рисунок 4.13 - Всплывающее меню с операциями

Сохранить текущий набор данных также можно с помощью пункта меню *Файл>Сохранить*.

4.2.5 Пропущенные значения

Исходные данные могут содержать пропущенные значения. При желании вы можете удалить все объекты с пропусками, щелкнув правой кнопкой мыши по таблице с данными и выбрав пункт меню «Удаление объектов с пропусками» (рисунок 4.14).

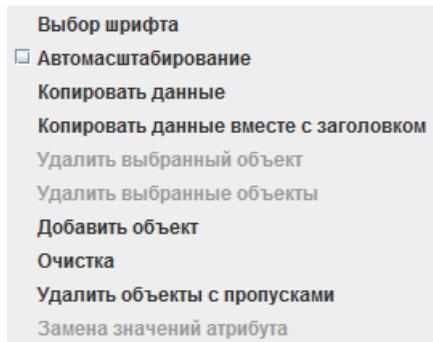


Рисунок 4.14 - Удаление объектов с пропусками

Здесь также присутствует возможность удаление одного или нескольких выбранных объектов.

4.3 Настройка метода оценки точности классификации

В программной системе *ECA* реализовано два метода оценки точности классификации: использование всей обучающей выборки и $k \times V$ – блочная кросс – проверка на тестовой выборке ($k \times V$ – *folds cross validation*), описание которых приведено в разделе 2.1. По умолчанию в программе используется первый метод использования всей обучающей выборки. Для того, чтобы выбрать другой метод оценки точности классификации необходимо перейти в пункт меню *Настройки*>*Настройка метода оценки точности* и в появившемся диалоговом окне выбрать метод оценки точности. На рисунке 4.15 приведено окно с выбором метода оценки точности классификации.

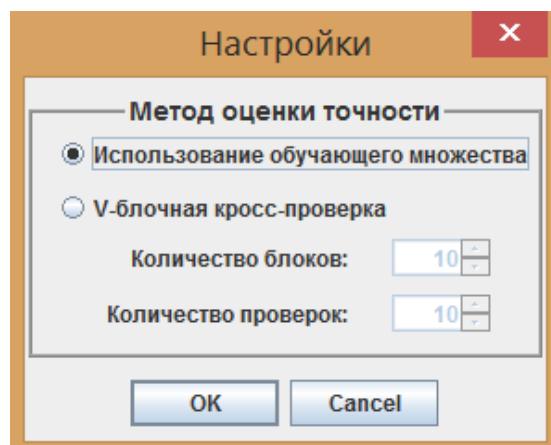


Рисунок 4.15 - Диалоговое окно выбора метода оценки точности

Для метода $k \times V$ – блочной кросс – проверки необходимо также ввести количество блоков V (по умолчанию - 10) и количество проверок k (по умолчанию - 10).

4.4 Настройка формата чисел

В программе можно настраивать формат чисел (количество десятичных знаков после запятой). Для этого необходимо перейти в пункт меню *Настройки>Настройка формата чисел* и в появившемся диалоговом окне выбрать количество знаков после запятой от одного до семи (рисунок 4.16).

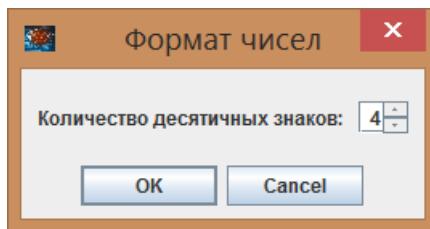


Рисунок 4.16 - Диалоговое окно настроек формата чисел

По умолчанию в программе количество десятичных знаков после запятой равно четырем.

4.5 Результаты классификации

4.5.1 Описание интерфейса системы отображения результатов

В данном разделе будут рассмотрены все основные показатели классификаторов, которые вычисляются в программной системе. Все они представляются в удобном для пользователя табличном виде. На рисунке 4.17 приведен пример окна с результатами классификации с помощью неоднородного ансамблевого алгоритма.

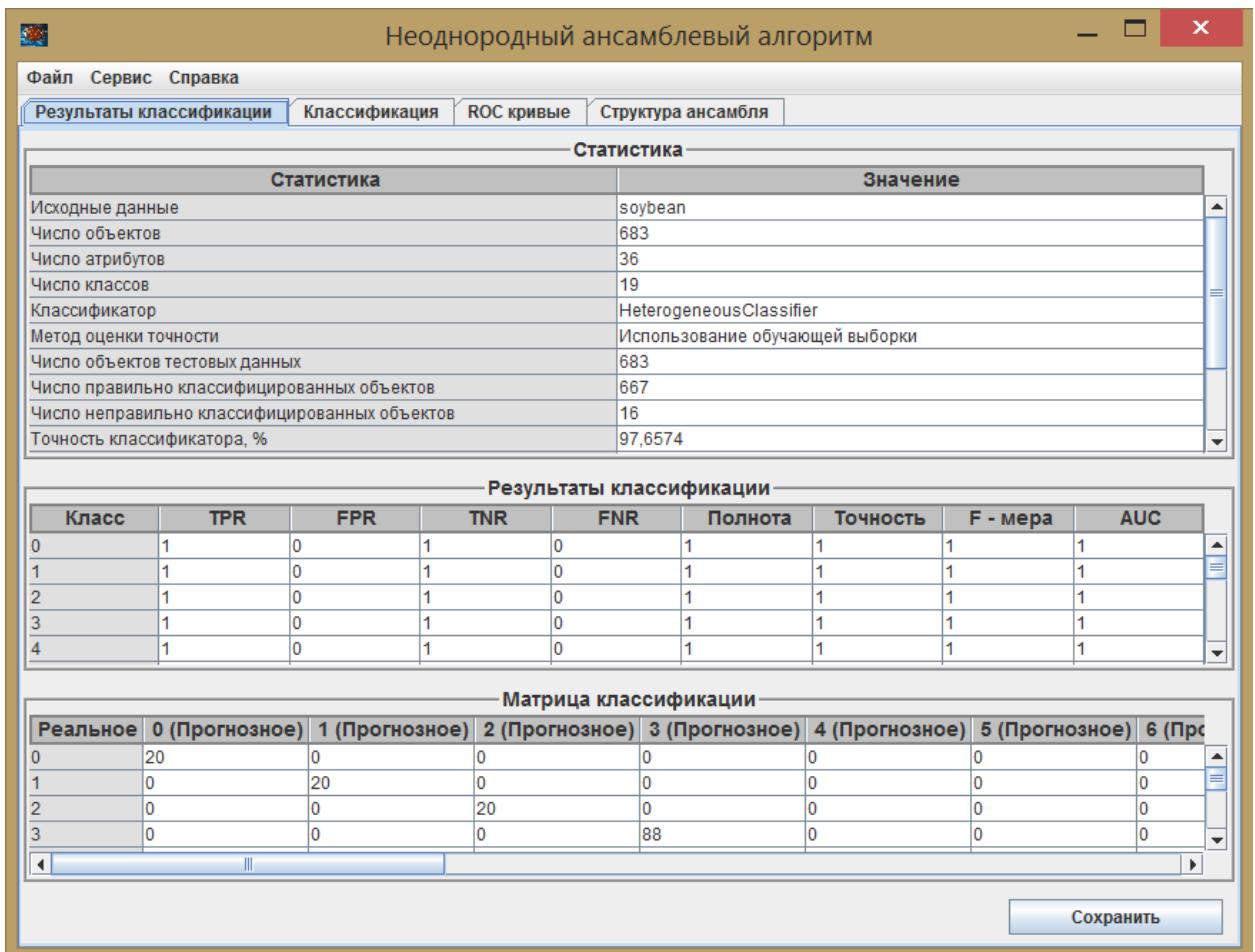


Рисунок 4.17 - Окно с результатами классификации

Здесь присутствует главное меню с тремя основными разделами, такими как файл, сервис и справка. С помощью пункта меню *Файл>Сохранить модель* вы можете сохранить модель классификатора в текстовый файл. По умолчанию создается имя файла в формате *[Название классификатора и дата построения]* (рисунок 4.18), но вы можете ввести любое другое имя файла. Также, впоследствии, вы можете загрузить сохраненную модель классификатора с помощью пункта меню *Файл>Загрузить модель в главном окне программы*.

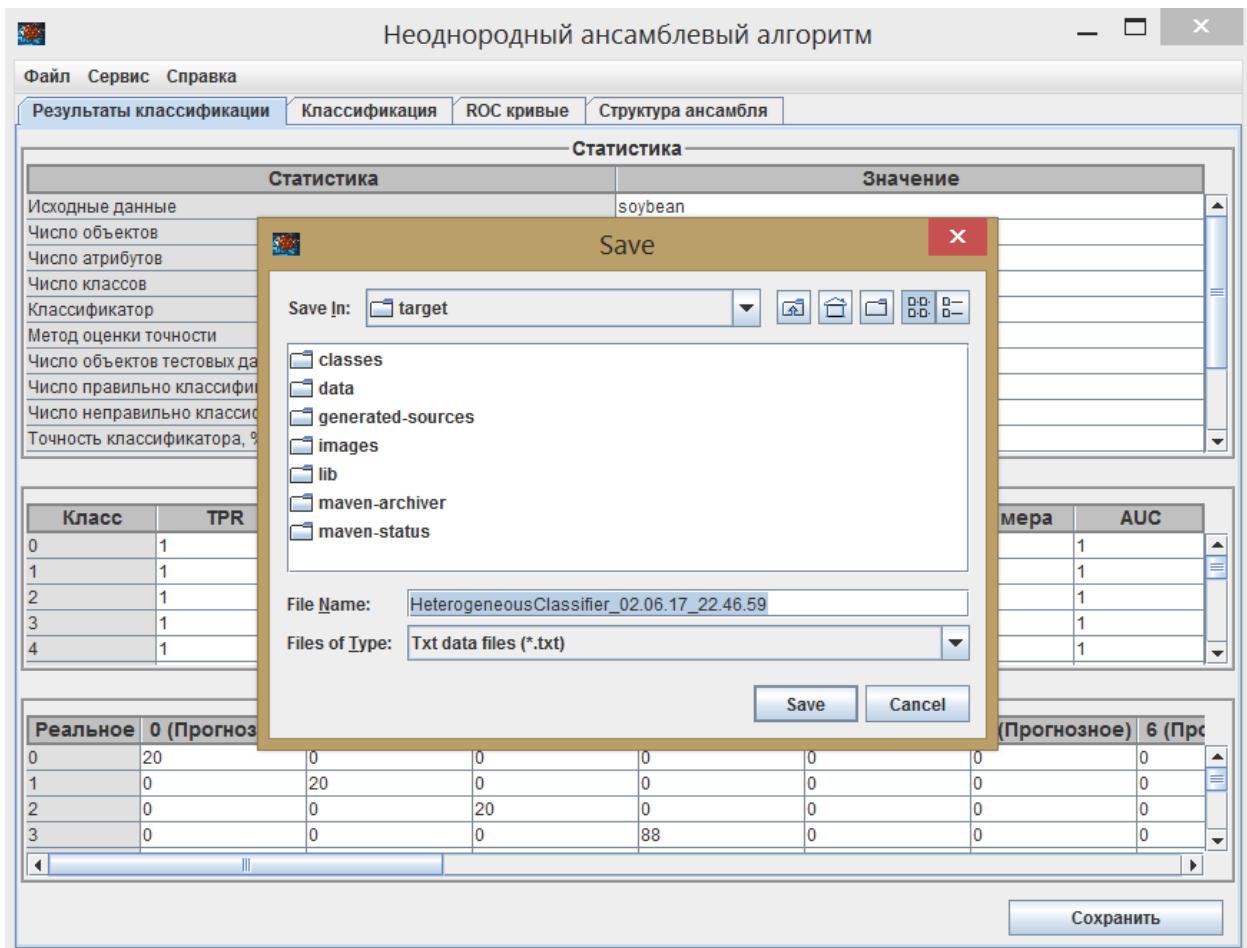


Рисунок 4.18 - Сохранение модели классификатора в файл

Пункт меню сервис состоит из трех подразделов: «*Исходные данные*», «*Входные параметры классификатора*» и «*Информация об атрибутах*», «*Статистика по атрибутам*». При щелчке на первый пункт подменю должно появиться окно, которое содержит таблицу с исходной обучающей выборкой. При щелчке на второй пункт подменю должно появиться окно с подробным описанием входных параметров классификатора, при щелчке на третий пункт должно появиться окно с информацией об атрибутах, а при щелчке на четвертый пункт должно появиться окно с основными статистическими характеристиками для атрибутов, которые представлены в удобном табличном виде. Отметим, что для числовых атрибутов также рассчитываются основные статистические характеристики, такие как: математическое ожидание, дисперсия и т.д. На рисунках 4.19 – 4.21 приведен пример соответствующих окон.

Данные: soybean

№	date	plant-stand	precip	temp	hail	crop-hist	area-dama
1	october	normal	gt-norm	norm	yes	same-lst-yr	low-areas
2	august	normal	gt-norm	norm	yes	same-lst-two-yrs	scattered
3	july	normal	gt-norm	norm	yes	same-lst-yr	scattered
4	july	normal	gt-norm	norm	yes	same-lst-yr	scattered
5	october	normal	gt-norm	norm	yes	same-lst-two-yrs	scattered
6	september	normal	gt-norm	norm	yes	same-lst-sev-yrs	scattered
7	september	normal	gt-norm	norm	yes	same-lst-two-yrs	scattered
8	august	normal	gt-norm	norm	no	same-lst-yr	scattered
9	october	normal	gt-norm	norm	yes	same-lst-sev-yrs	scattered
10	august	normal	gt-norm	norm	yes	same-lst-two-yrs	scattered
11	october	normal	lt-norm	gt-norm	yes	same-lst-yr	whole-field
12	august	normal	lt-norm	norm	no	same-lst-yr	whole-field
13	july	normal	lt-norm	norm	yes	same-lst-yr	upper-areas
14	october	normal	lt-norm	norm	no	same-lst-sev-yrs	whole-field
15	october	normal	lt-norm	gt-norm	yes	same-lst-yr	whole-field
16	september	normal	lt-norm	gt-norm	no	same-lst-sev-yrs	whole-field
17	october	normal	lt-norm	gt-norm	no	diff-lst-year	upper-areas
18	august	normal	lt-norm	norm	yes	same-lst-two-yrs	upper-areas
19	july	normal	lt-norm	gt-norm	yes	same-lst-two-yrs	upper-areas
20	september	normal	lt-norm	gt-norm	no	same-lst-two-yrs	upper-areas
21	may	lt-normal	gt-norm	lt-norm	yes	same-lst-two-yrs	low-areas
22	may	lt-normal	gt-norm	lt-norm	yes	same-lst-yr	low-areas

OK

Рисунок 4.19 - Таблица с исходной обучающей выборкой

Статистика по атрибутам

Информация о данных

Статистика	Значение
Данные	soybean
Число объектов	683
Число атрибутов	36
Число классов	19

Выбранный атрибут

Атрибут: date

Тип: Категориальный

Статистика

Код	Значение	Число объектов
0	october	90
1	august	131
2	july	118
3	september	149
4	may	75
5	april	26
6	june	93

OK

Рисунок 4.20 - Окно со статистикой по атрибутам

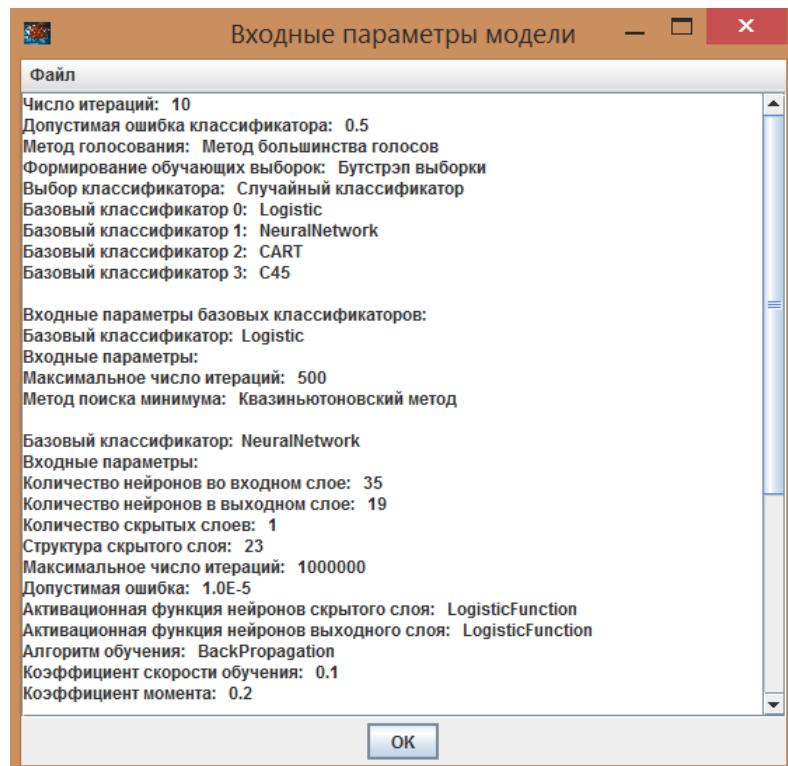


Рисунок 4.21 - Окно с информацией о входных параметрах классификатора

С помощью меню *Файл>Сохранить* вы можете сохранить соответствующую информацию в текстовый файл. В заключение этого пункта отметим, что информацию о входных параметрах классификатора можно также посмотреть, наведя на верхнюю таблицу курсором мыши (рисунок 4.22).

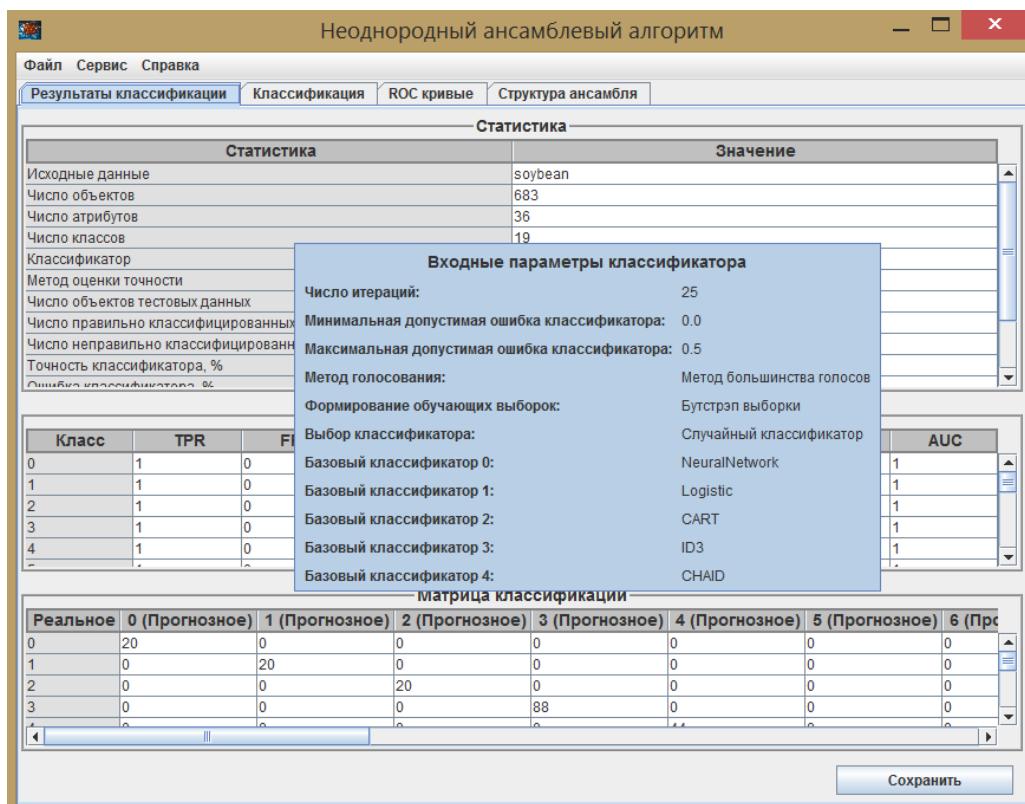


Рисунок 4.22 - Информация о входных параметрах классификатора

При этом должна появиться всплывающая подсказка с информацией о входных параметрах классификатора.

4.5.2 Описание результатов классификации

Теперь перейдем к описанию основного содержимого окна, которое состоит из трех основных вкладок: статистика, классификация, *ROC* – кривые. Основная вкладка «Статистика» состоит из трех таблиц с различными характеристиками классификации:

- основная статистика;
- результаты классификации с учетом издержек;
- матрица классификации.

В первой таблице содержится основная информация, а именно: параметры классификатора (для каждого свой, в зависимости от алгоритма), информация о данных и основные характеристики классификации, такие как точность, средняя абсолютная ошибка классификации и среднеквадратическая ошибка классификации.

Во второй таблице содержится информация о классификации с учетом издержек. Подробное описание этих показателей будет рассмотрено в следующем разделе.

В третьей таблице содержится структура матрицы классификации, в которой строки соответствуют наблюдаемым (реальным) значениям соответствующего класса, а столбцы - прогнозным значениям соответствующего класса. В каждой ячейке таблицы содержится число объектов класса i , спрогнозированных моделью как класс j .

С помощью кнопки «Сохранить» вы можете сохранить результаты классификации, приведенные в данной вкладке, в файл с расширением *.xls*, *.xlsx*.

4.5.3 ROC - анализ

Теперь перейдем к рассмотрению графиков *ROC* – кривых в программной системе *ECA*. Графики *ROC* – кривых расположены во вкладке *ROC* – кривые. На рисунке 4.23 приведен результат построения *ROC* – кривых для неоднородного ансамблевого алгоритма.

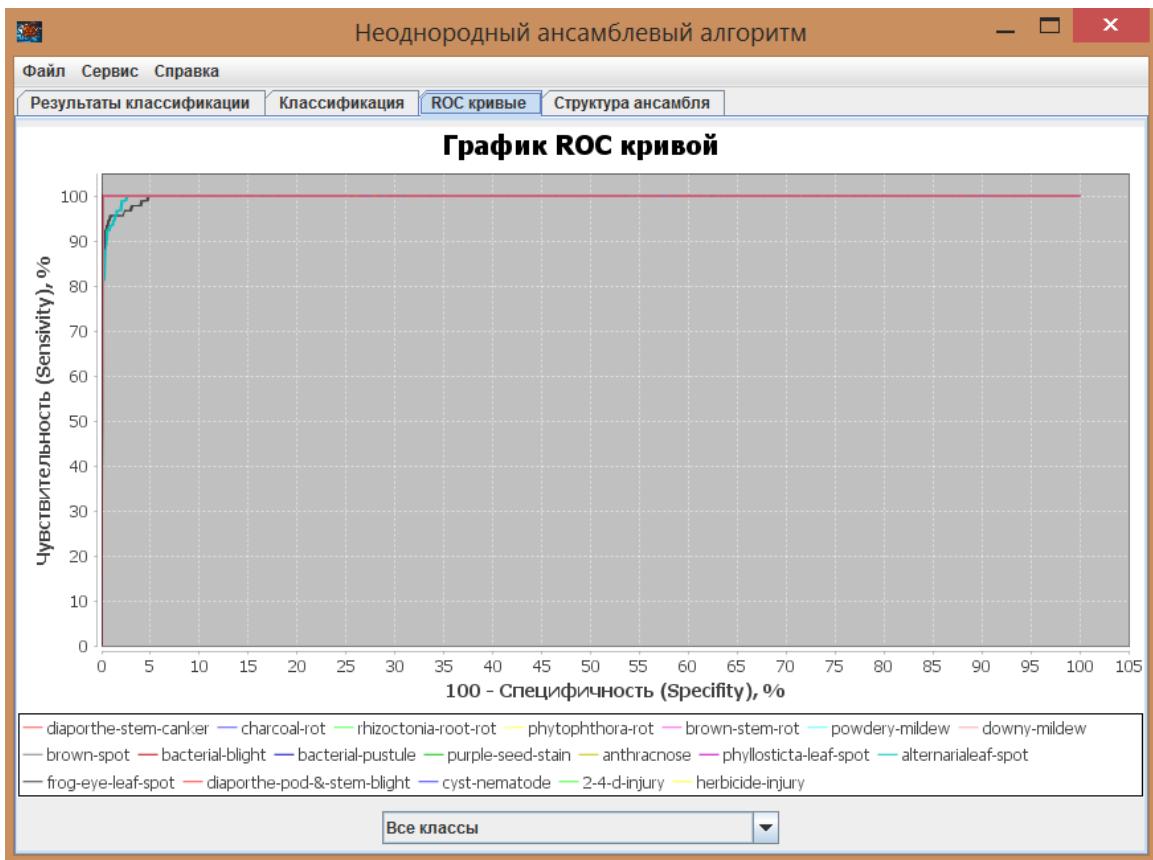


Рисунок 4.23 - Графики ROC - кривых

Если число классов больше двух, то *ROC* – кривая строится для каждого класса отдельно. По умолчанию отображаются *ROC* – кривые для всех классов, совмещенные на одной координатной плоскости. Если вы хотите посмотреть результат построения *ROC* – кривой для конкретного класса, необходимо выбрать этот класс в выпадающем списке, расположеннем под панелью с графиками. Для просмотра данных *ROC* – кривой необходимо щелкнуть правой кнопкой мыши по графику и в появившемся выпадающем меню выбрать пункт «Показать данные» (рисунки 4.24, 4.25).

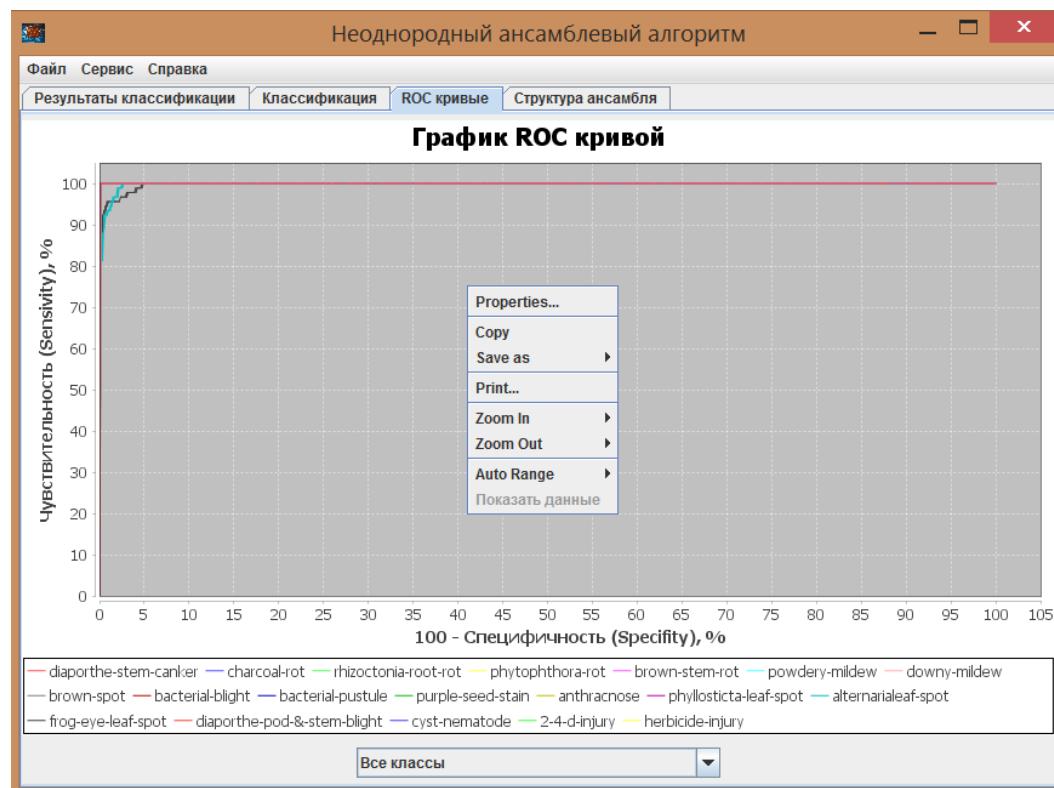


Рисунок 4.24 - Графики ROC – кривых (редактирование графика)

Данные ROC - кривой		
Порог для класса (phyllosticta-leaf-spot)	100 - Специфичность	Чувствительность
0	100	100
0	99,8492	100
0	99,6983	100
0	99,5475	100
0	99,3967	100
0	99,2459	100
0	99,095	100
0	98,9442	100
0	98,7934	100
0	98,6425	100
0	98,4917	100
0	98,3409	100
0	98,19	100
0	98,0392	100
0	97,8884	100
0	97,5867	100
0	97,4359	100
0	97,2851	100
0	97,1342	100
0	96,9834	100
0	96,8326	100
0	96,6817	100

Рисунок 4.25 - Данные ROC – кривой

Отметим, что данные для *ROC* – кривых вычисляются с помощью библиотеки *weka*. Также в этом меню расположены и другие пункты, предназначенные для редактирования параметров графика.

4.5.4 Классификация нового примера

На основе построенной модели можно классифицировать новые примеры. Для этого необходимо перейти во вкладку «Классификация» затем ввести в таблицу значения нового объекта подлежащего классификации и нажать на кнопку «Классифицировать». Результат классификации в формате [код класса, значение класса, вероятность класса] появится в нижнем текстовом поле (рисунок 4.26).

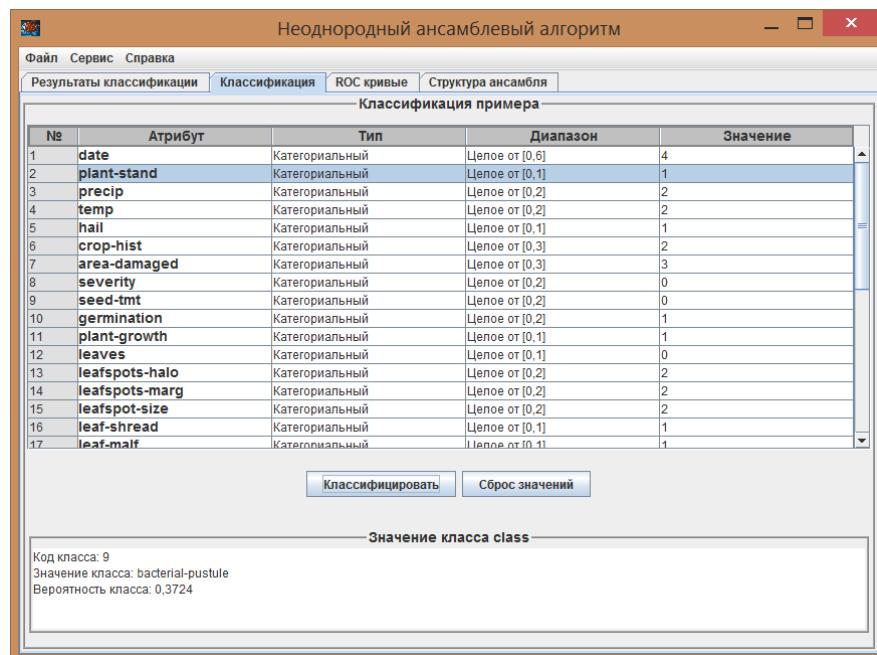


Рисунок 4.26 - Классификация нового примера

Также, при наведении мышью на соответствующий атрибут, будет высвеченна всплывающая подсказка с информацией об атрибуте (рисунок 4.27).

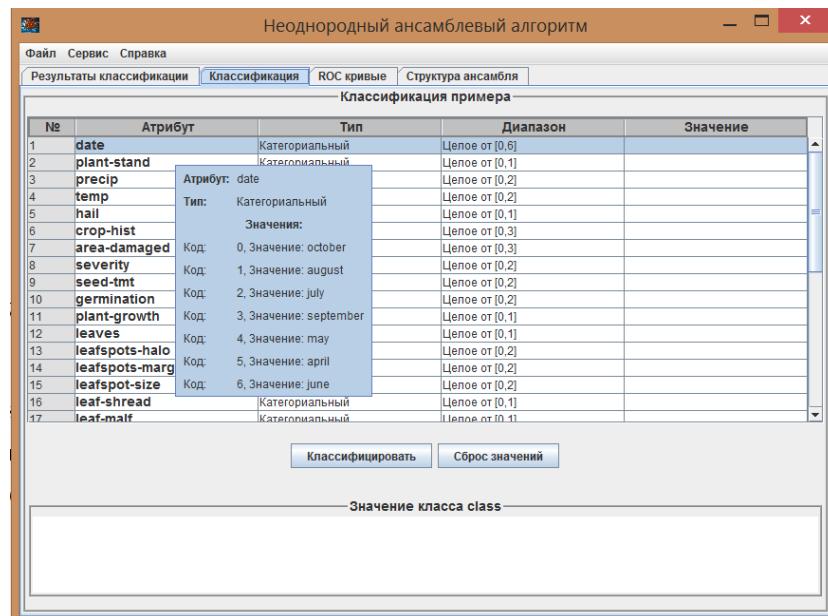


Рисунок 4.27 - Классификация нового примера (информация об атрибуте)

При желании вы можете сбросить все значения, нажав на кнопку «Сброс значений».

4.6 Индивидуальные алгоритмы. Деревья решений

В программе реализовано четыре алгоритма построения деревьев решений, такие как: *CART*, *ID3*, *C4.5*, *CHAID*. Для настройки параметров алгоритма перейдите в пункт меню *Классификаторы*>*Индивидуальные классификаторы*>*Деревья решений* и затем выберите алгоритм построения дерева решений (рисунок 4.28).

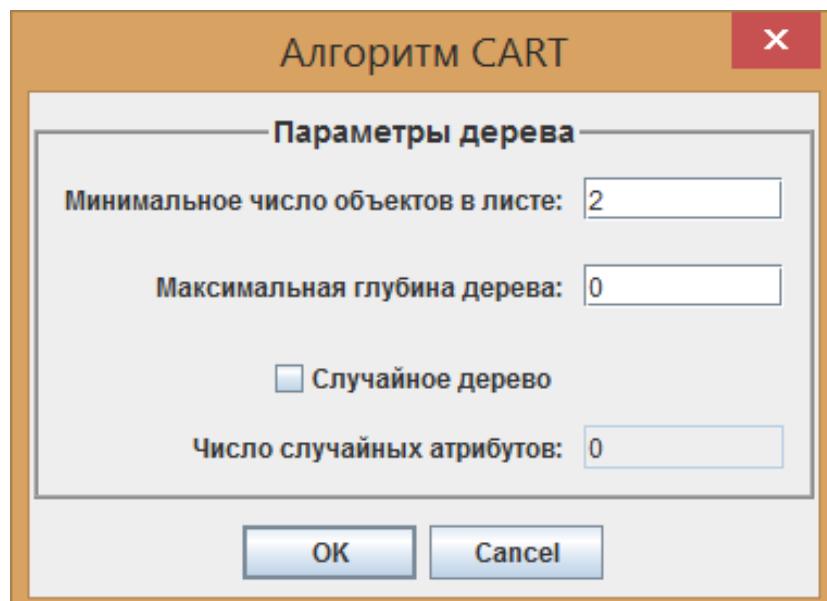


Рисунок 4.28 - Диалоговое окно настройки параметров дерева решений

За остановку обучения дерева решений отвечают два параметра:

- минимальное число объектов в листе (по умолчанию - 2);
- максимальная глубина дерева (по умолчанию – 0, где 0 соответствует бесконечности).

Если вы выберете пункт «Случайное дерево», то при построении дерева решений на каждой итерации выбор наилучшего расщепления будет происходить на основе N атрибутов, отбираемых случайным образом (равновероятно). Число 0 соответствует выбору всех атрибутов.

4.6.1 Алгоритм CHAID

Алгоритм *CHAID* расположен в пункте меню *Классификаторы*>*Индивидуальные классификаторы*>*Деревья решений*>*CHAID*. На рисунке 4.29 приведено диалоговое окно с настройками параметров алгоритма *CHAID*.

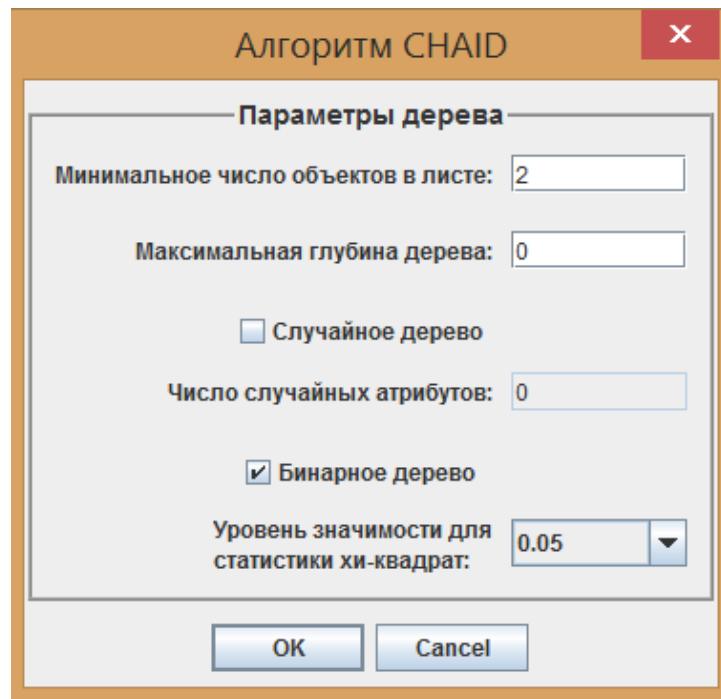


Рисунок 4.29 - Диалоговое окно с настройками параметров алгоритма *CHAID*

Для алгоритма *CHAID* необходимо задать два дополнительных параметра, таких как уровень значимости α для теста хи-квадрат и тип дерева (бинарное, *m*-арное).

4.6.2 Визуализация деревьев решений

После настройки параметров дерева решений и последующего его построения появится окно с результатами классификации, среди которых присутствует вкладка со структурой дерева решений (рисунок 4.30).

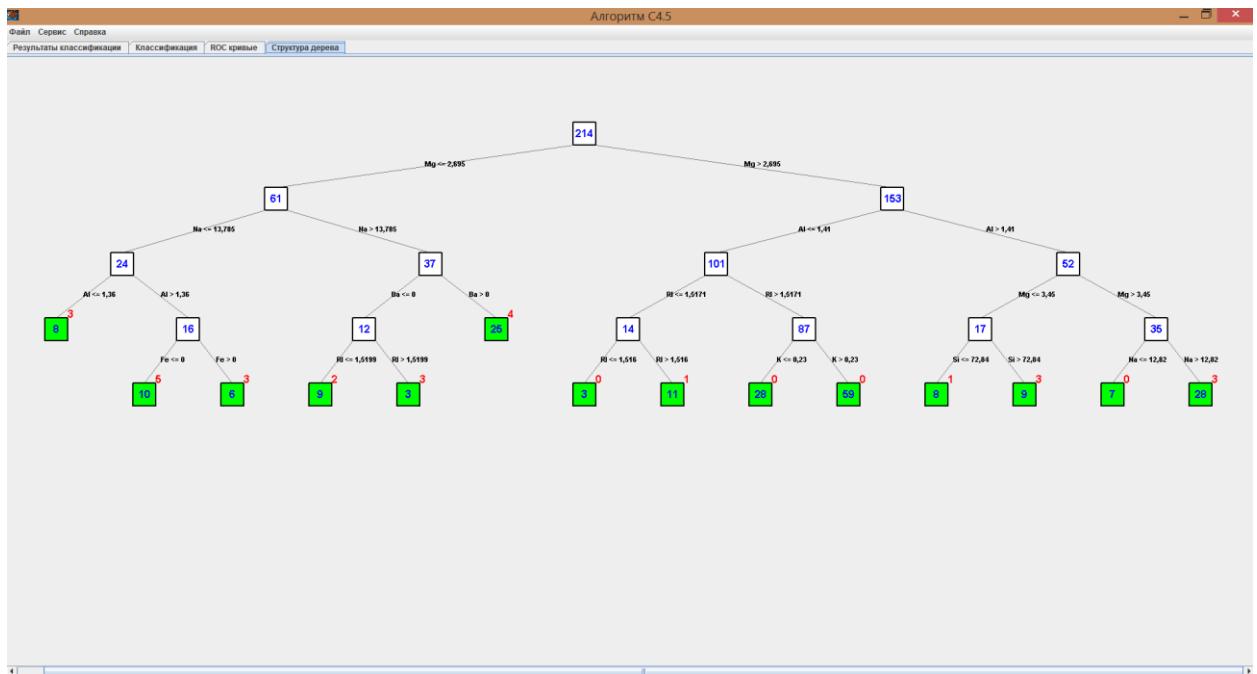


Рисунок 4.30 - Вкладка со структурой дерева решений *C4.5*

Вы можете просматривать информацию о каждом узле при наведении на него курсором мыши, при этом должна появиться всплывающая подсказка с информацией об узле (рисунок 4.31).

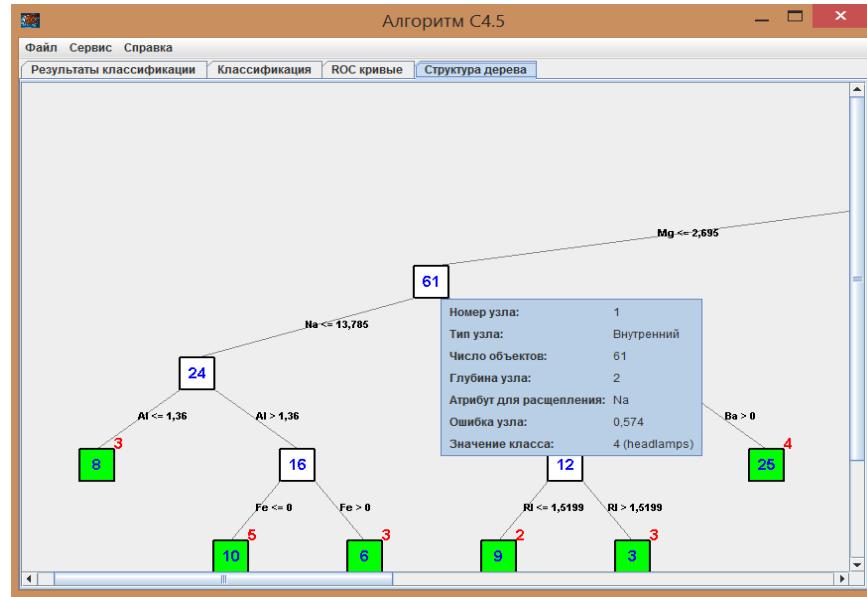


Рисунок 4.31 - Вкладка со структурой дерева решений *C4.5* (информация об узле)

При щелчке правой кнопкой мыши по панели со структурой дерева решений, появиться всплывающее меню с такими операциями как (рисунок 4.32):

- настройка параметров изображения;
- увеличение изображения;
- уменьшение изображения;
- копирование изображения;
- сохранение изображения в файл с расширением *.png*.

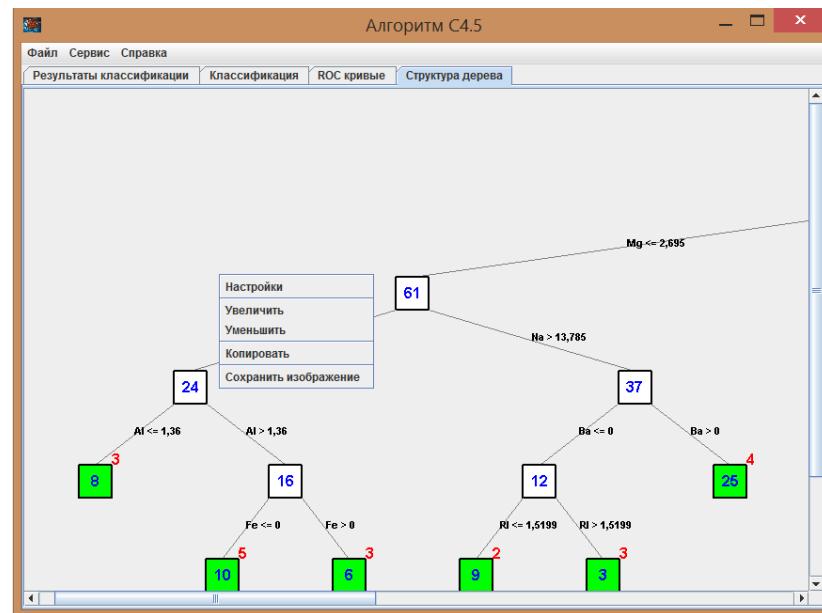


Рисунок 4.32 - Меню с основными операциями над изображением дерева

Пункт меню «Настройки» предназначен для настройки параметров картинки так, как вам удобно (рисунок 4.33).

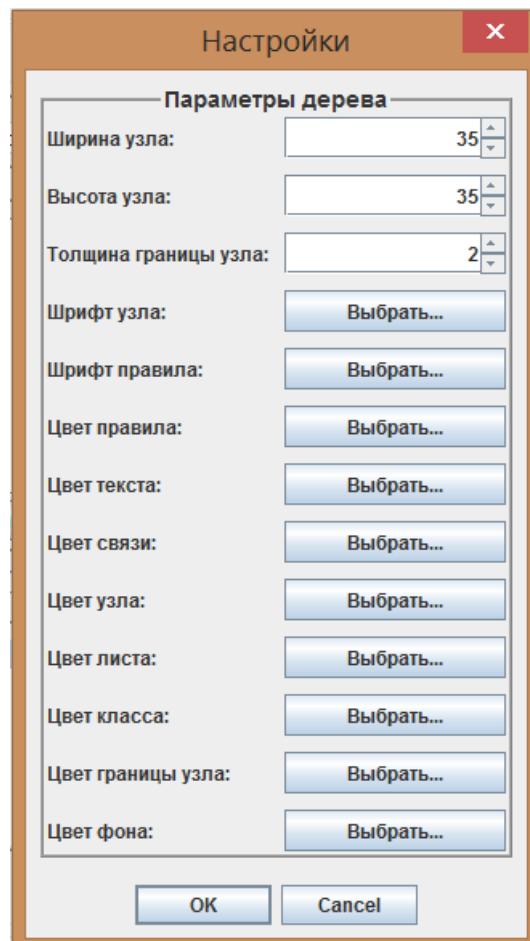


Рисунок 4.33 - Диалоговое окно с настройками параметров изображения дерева решений

Здесь присутствует возможность настройки размеров узла, толщины его границы, а также различных цветов и шрифтов.

4.7 Индивидуальные алгоритмы. Нейронные сети

4.7.1 Настройка нейронной сети

Модуль для построения нейронных сетей расположен в пункте меню *Классификаторы>Индивидуальные классификаторы>Нейронная сеть (Многослойных персептрон)*. В программе реализованы нейронные сети типа многослойного персептрана, которые используют в качестве обучения алгоритм обратного распространения ошибки (*Back Propagation*). Перед началом обучения все данные проходят нормализацию, используя минимаксную нормализацию.

На рисунке 4.34 изображен вид диалогового окна для настройки параметров нейронной сети.

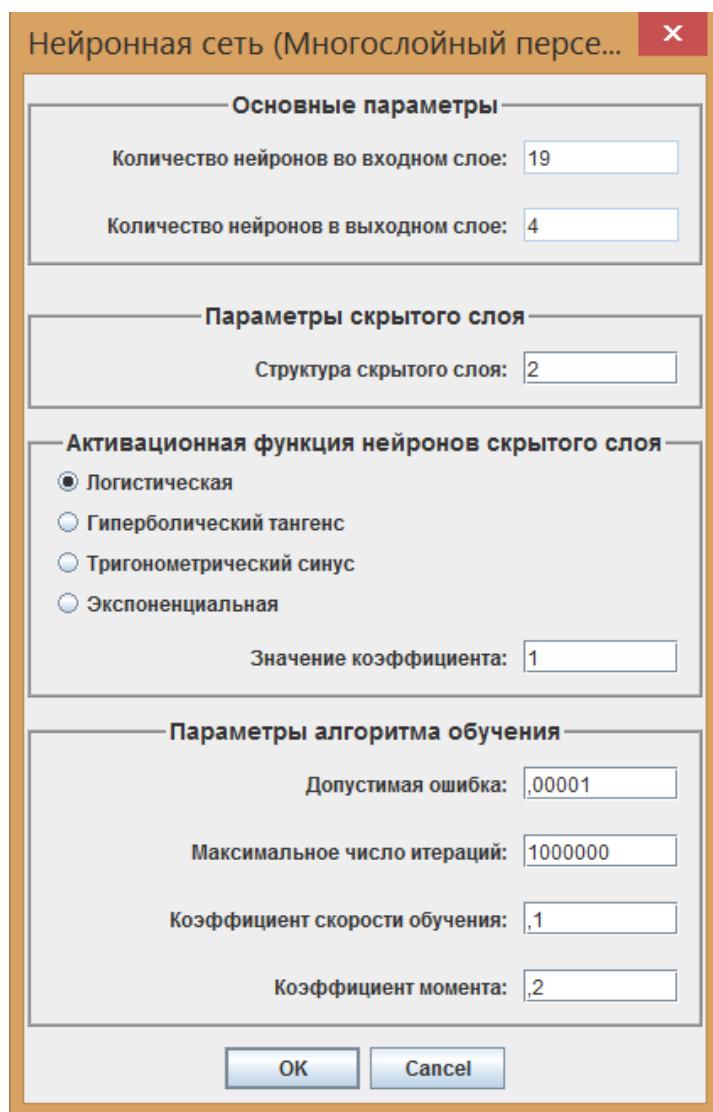


Рисунок 4.34 - Настройка параметров нейронной сети

Далее приведем описание всех параметров настройки нейронной сети:

- число нейронов входного слоя устанавливается равным числу входных атрибутов;
- число нейронов выходного слоя устанавливается равным числу классов;
- структура скрытого слоя задается списком значений через запятую, например, список 5,8,12,4 соответствует четырем скрытым слоям с соответствующим числом нейронов в каждом слое. Число нейронов в скрытом слое вычисляется с помощью формулы 1.25. По умолчанию в программе устанавливается число нейронов в скрытом слое, соответствующее нижней границе (формула 1.30). При наведении курсором мыши на панель настроек скрытого слоя будет высвеченна подсказка по настройке числа нейронов в скрытом слое.
- активационная функция нейронов скрытого слоя: логистическая, гиперболический тангенс, тригонометрический синус, экспоненциальная. При этом нейроны выходного слоя всегда используют логистическую активационную функцию.

Как уже упоминалось выше, нейронная сеть обучается с помощью алгоритма обратного распространения ошибки. Ниже приведено описание параметров алгоритма обучения:

- число итераций – число итераций необходимое для обучения сети;
- допустимая ошибка – параметр ε , задающий критерий остановки обучения, если на очередной итерации значение (формула 1.32) становится меньше чем ε , то процедура обучения заканчивается;
- скорость обучения является одним из основных параметров градиентного метода поиска минимума целевой функции, он задает шаг;
- коэффициент момента – дополнительный параметр для повышений эффективности алгоритма.

4.7.2 Визуализация нейронных сетей

После обучения нейронной сети появиться окно с результатами классификации, среди которых присутствует вкладка со структурой нейронной сети (рисунок 4.35).

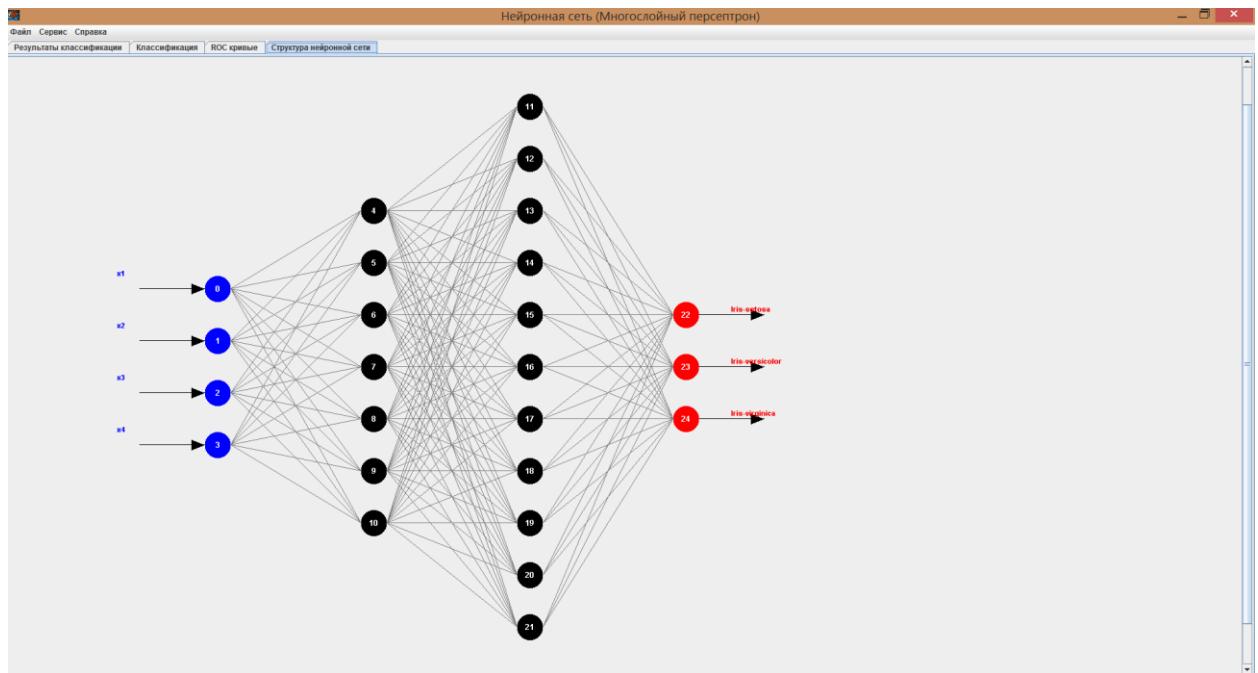


Рисунок 4.35 - Структура нейронной сети

Вы можете просматривать информацию о каждом узле при щелчке по нему левой кнопкой мыши (рисунок 4.36).

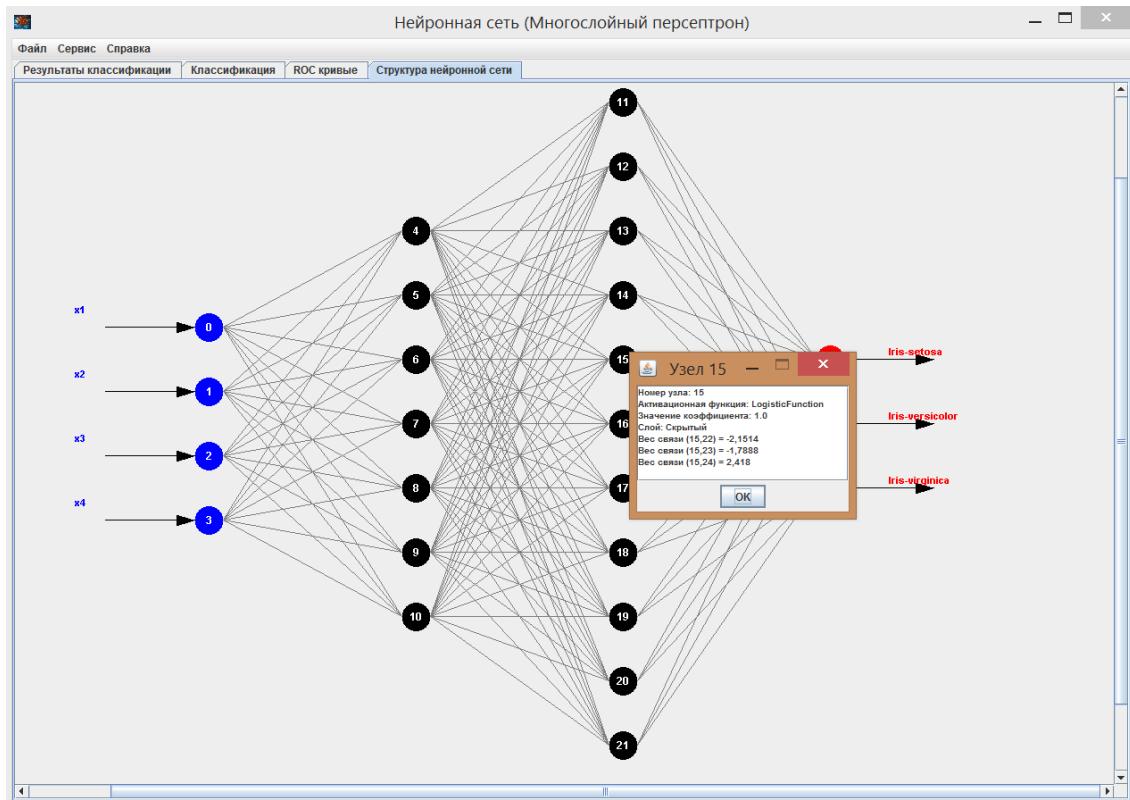


Рисунок 4.36. - Структура нейронной сети (информация об узле)

При щелчке правой кнопкой мыши по панели появиться всплывающее меню с такими операциями, как (рисунок 4.37):

- настройка параметров изображения;
- увеличение изображения;
- уменьшение изображения;
- текстовое представление модели;
- копирование изображения;
- сохранение изображения в файл с расширением .png.

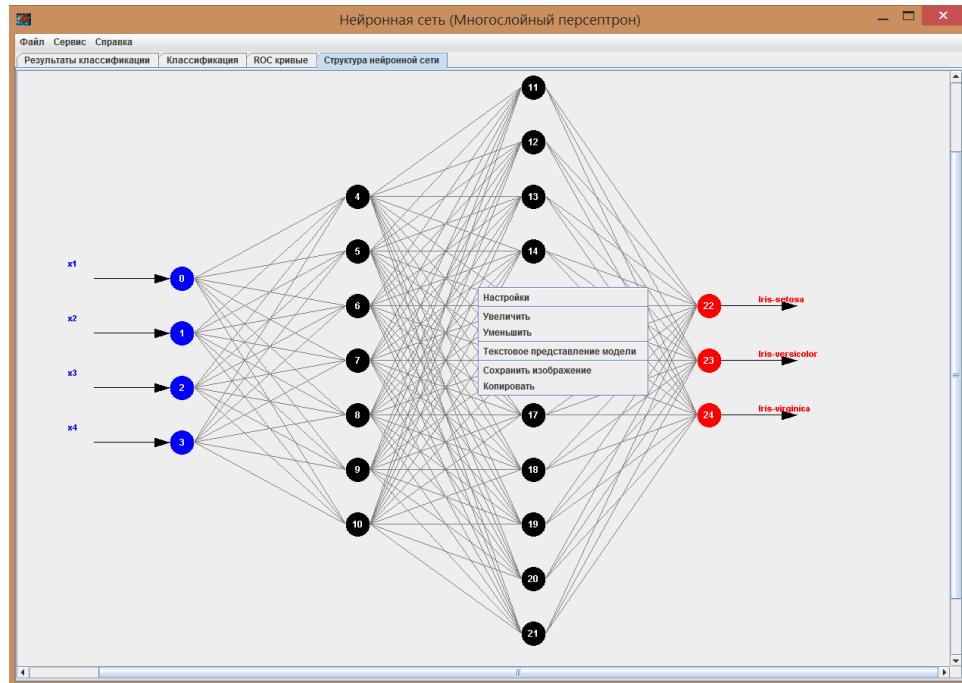


Рисунок 4.37 - Меню с операциями над изображением нейронной сети

Здесь присутствует возможность просмотра текстового представления модели нейронной сети, в котором также присутствуют значения весов всех связей (рисунок 4.38).

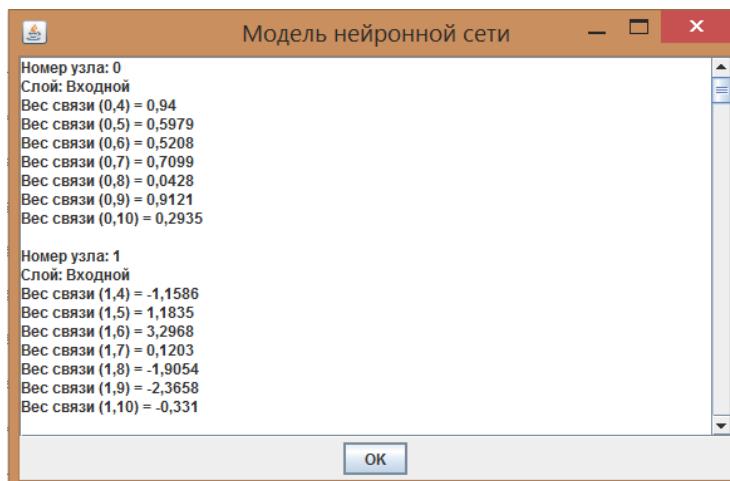


Рисунок 4.38 - Текстовое представление модели нейронной сети

При нажатии на пункт меню «Настройки» вы можете настраивать параметры картинки так, как вам удобно (рисунок 4.39).

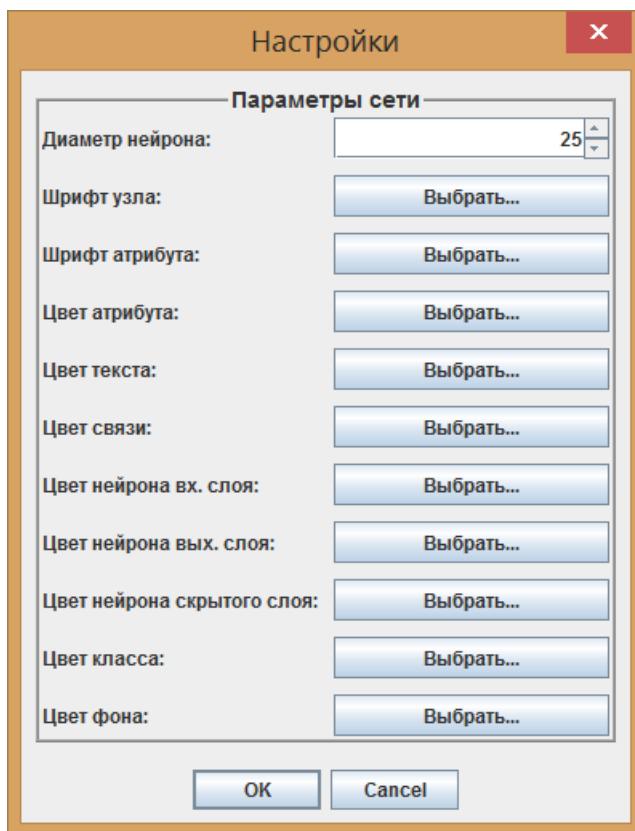


Рисунок 4.39 - Диалоговое окно с настройками параметров изображения нейронной сети

Здесь задается диаметр нейрона, а также основные цвета и шрифты составляющих изображения.

4.8 Индивидуальные алгоритмы. Алгоритм k – взвешенных ближайших соседей

Алгоритм расположен в пункте меню *Классификаторы>Индивидуальные классификаторы>Алгоритм k – взвешенных ближайших соседей*. Перед началом обучения каждый категориальный атрибут, имеющий k ($k > 2$) категорий, преобразуется в $k - 1$ бинарных атрибутов, затем все данные проходят нормализацию, используя минимаксную нормализацию. На рисунке 4.40 приведено диалоговое окно с настройками параметров алгоритма.

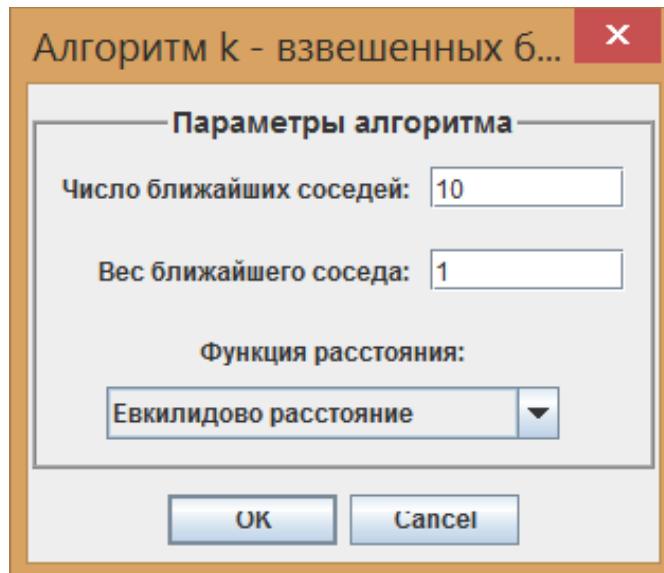


Рисунок 4.40 - Диалоговое окно с настройками параметров алгоритма

Здесь вы можете выбрать число ближайших соседей m (по умолчанию – 10), вес ближайшего соседа $\omega \in [0.5, 1]$ (по умолчанию 1) и функцию расстояния, которая может быть одной из следующих:

1. Евклидово расстояние (используется по умолчанию).
2. Квадрат Евклидова расстояния.
3. Манхэттенское расстояние.
4. Расстояние Чебышева.

4.9 Индивидуальные алгоритмы. Логистическая регрессия

4.9.1 Настройка параметров

Алгоритм мультиномиальной логистической регрессии взят из библиотеки анализа данных *weka* [8]. Он расположен в пункте меню *Классификаторы>Индивидуальные классификаторы>Логистическая регрессия*. В данном алгоритме каждый категориальный атрибут, имеющий k ($k > 2$) категорий, преобразуется в $k - 1$ бинарных атрибутов. На рисунке 4.41 приведено диалоговое окно с настройками параметров алгоритма логистической регрессии.

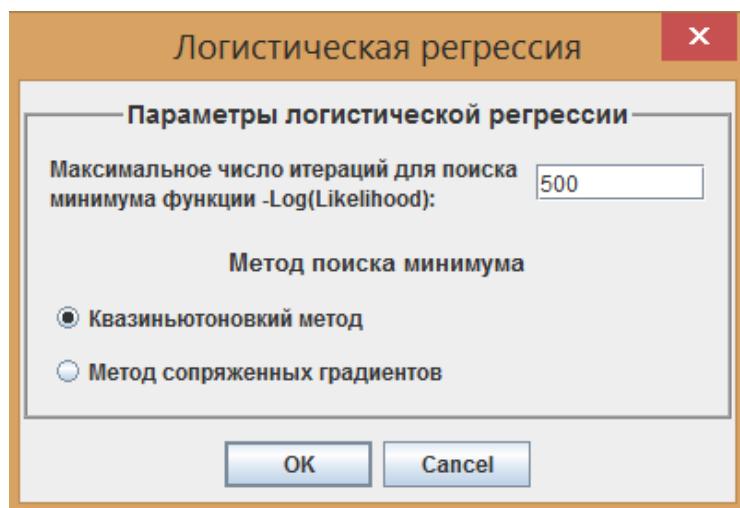


Рисунок 4.41 - Диалоговое окно с настройками параметров логистической регрессии
Здесь необходимо задать такие параметры как:

- число итераций (по умолчанию – 500) для поиска минимума функции логарифмического правдоподобия;
- метод поиска минимума: квазиньютоновский метод (используется по умолчанию), либо метод сопряженных градиентов.

После настройки параметров алгоритма и последующего его построения появится окно с результатами логистической регрессии, среди которых присутствует информация о коэффициентах модели и таблица со значимыми атрибутами (рисунок 4.42).

Логистическая регрессия								
Файл Сервис Справка	Результаты классификации	Классификация	ROC кривые	Оценки коэффициентов	Значимые атрибуты			
Атрибут	Класс 0	Класс 1	Класс 2	Класс 3	Класс 4	Класс 5	Класс 6	Класс 7
Intercept	10,5121	3,3944	2,7364	8,3659	11,5331	9,4078	5,1355	-12,235
date=october	-0,5898	-0,3185	0,8474	0,1381	1,7888	-1,1429	1,865	-9,4453
date=august	2,6375	0,0938	0,1039	0,3199	-0,861	-0,6397	0,3559	-11,4461
date=july	-1,8334	2,8082	0,5636	-0,0669	0,206	2,2307	-2,0957	-1,5498
date=september	2,2103	-0,4492	0,0733	0,3159	1,2712	0,7776	0,2435	-11,2131
date=may	-2,1275	-1,5094	-0,8236	0,4203	-1,2829	0,1056	1,1832	19,0672
date=april	-4,3384	-0,2148	3,183	-1,8906	-1,8451	-1,8552	-2,6375	25,2277
date=june	-0,7764	-1,25	-2,0593	-0,6955	-1,0671	-1,3989	-0,2537	18,7935
plant-stand=lt-normal	-3,3141	-1,3115	-0,4395	2,5243	-0,3515	-0,2393	-1,0673	-7,7719
precip=gt-norm	1,2752	-1,1449	0,0253	-1,7405	-2,8065	-2,001	0,1313	7,7628
precip=l-norm	-0,5116	3,448	0,9111	0,2227	4,3953	2,9448	0,2392	-20,8621
precip=norm	-1,4828	-0,7749	-0,6787	2,3589	0,9595	0,8172	-0,3584	3,4801
temp=norm	1,637	1,4945	-0,1784	1,7875	-0,7551	0,5018	0,0297	3,9013
temp=gt-norm	-0,1109	0,4366	1,111	1,2946	1,7118	0,3171	-0,3178	-3,1866
temp=lt-norm	-3,6035	-4,3637	-1,8014	-6,7614	-1,6534	-1,8057	0,5649	-2,7532
hail=no	-1,9835	0,1225	0,8429	3,4187	-0,5213	0,9015	2,3141	11,4413
crop-hist=same-lst-yr	-0,399	-0,0225	-1,9264	-0,2419	-0,3492	-0,2859	0,3702	-1,3891
crop-hist=same-lst-two-yrs	0,2383	1,7091	-0,4423	1,2307	2,1296	-0,135	0,2622	2,5523
crop-hist=same-lst-sev-yrs	2,2143	0,0638	0,2711	0,8757	-0,5802	0,7211	0,1822	1,3039
crop-hist=diff-lst-year	-5,3634	-4,5924	0,0314	-4,9207	-3,3743	-0,8574	-1,9347	-7,0241
area-damaged=low-areas	-0,002	0,3631	1,534	2,1064	-0,1784	0,0749	0,2725	0,8494
area-damaged=scattered	1,473	-1,9441	-1,7144	-1,2757	-1,2281	-1,1935	-1,8215	-6,0626
area-damaged=whole-field	-1,6086	0,2026	-1,1356	-0,8026	0,7464	1,1002	-0,791	6,5411
area-damaged=upper-areas	0,6146	0,9929	0,824	-0,7205	0,4342	-0,3538	2,1863	-3,5538
severity=pot-severe	0,1406	-0,3822	-0,5984	0,442	-0,8274	-0,5821	-0,4444	-0,4575
severity=severe	-2,3974	0,23	1,2425	-0,2654	3,3772	0,2708	0,2267	7,7882
severity=minor	0,5662	0,3576	0,2938	-0,4138	-0,0945	0,5687	0,4281	-1,8385
seed-tmt=None	-1,4797	-0,177	0,3422	-0,3174	-0,5275	-2,4113	-0,4644	2,7789
seed-tmt-fungicide	1,0724	0,1270	0,5516	0,2207	0,5404	1,2629	0,4472	2,6024

Рисунок 4.42 - Таблица значений коэффициентов логистической регрессии

Значение коэффициента напротив атрибута с именем *Intercept* соответствует свободному коэффициенту.

4.9.2 Значимые атрибуты

Для определения значимых атрибутов с помощью логистической регрессии используется метод на основе результатов построения ROC-кривых и вычисления AUC, описание которого приведено в разделе 2.1. На рисунке 4.43 приведен пример таблицы со значимыми атрибутами.

Логистическая регрессия									
Файл Сервис Справка	Результаты классификации	Классификация	ROC кривые	Оценки коэффициентов	Значимые атрибуты				
Атрибут	AUC (Класс 0)	AUC (Класс 1)	AUC (Класс 2)	AUC (Класс 3)	AUC (Класс 4)	AUC (Класс 5)	AUC (Класс 6)	AUC (Класс 7)	AUC (Класс 8)
date	0,6827	0,7076	0,8662	0,8016	0,7335	0,5971	0,5738	0,7787	
plant-stand	0,721	0,721	0,7426	0,8277	0,5957	0,5623	0,5623	0,5281	
precip	0,6403	0,9593	0,6403	0,6425	0,9468	0,8662	0,6403	0,6014	
temp	0,7104	0,7513	0,9548	0,5196	0,6407	0,7722	0,6675	0,6806	
hail	0,57	0,6875	0,5443	0,5676	0,5099	0,636	0,636	0,5384	
crop-hist	0,5689	0,5428	0,6061	0,5274	0,5833	0,6061	0,5387	0,586	
area-damaged	0,8826	0,7805	0,8431	0,8784	0,8893	0,5673	0,5673	0,6895	
severity	0,7235	0,681	0,8155	0,6866	0,6781	0,6275	0,5367	0,6025	
seed-tmt	0,5789	0,6033	0,5961	0,6541	0,6072	0,5893	0,6033	0,6282	
germination	0,5797	0,5649	0,6742	0,7195	0,5825	0,5784	0,6742	0,5683	
plant-growth	0,8446	0,8446	0,8446	0,884	0,5661	0,6704	0,6704	0,6347	
leaves	0,5581	0,5581	0,9313	0,5647	0,5612	0,5581	0,5581	0,6561	
leafspots-halo	0,8484	0,8484	0,8484	0,5484	0,7579	0,8484	0,8137	0,7174	
leafspots-marg	0,8484	0,8484	0,8484	0,5405	0,7555	0,8484	0,6825	0,7047	
leafspot-size	0,8484	0,8484	0,8484	0,5563	0,7603	0,8484	0,7051	0,7301	
leaf-shread	0,5724	0,5724	0,5724	0,5807	0,5751	0,5724	0,5724	0,6951	
leaf-malf	0,5339	0,5339	0,5339	0,5378	0,5352	0,5339	0,6206	0,5381	
leaf-mild	0,5302	0,5302	0,5302	0,5336	0,5313	1	1	0,5338	
stem	0,7232	0,7232	0,7232	0,7487	0,7316	0,7919	0,7919	0,5887	
lodging	0,6229	0,5456	0,5198	0,5222	0,6615	0,5317	0,5317	0,5355	
stem-cankers	0,871	0,7006	0,9857	0,8817	0,7081	0,7006	0,7006	0,5864	
canker-lesion	0,7862	0,9661	0,9525	0,9252	0,803	0,7451	0,7451	0,7489	
fruiting-bodies	0,9367	0,5784	0,5784	0,5874	0,5814	0,5784	0,5784	0,6381	
external-decay	0,9133	0,6116	0,9133	0,6322	0,6158	0,6116	0,6116	0,5944	
mycelium	0,5045	0,5045	0,65	0,505	0,5047	0,5045	0,5045	0,5051	
int-discolor	0,5483	1	0,5483	0,5538	1	0,5483	0,5483	0,5541	
sclerotia	0,5151	1	0,5151	0,5168	0,5156	0,5151	0,5151	0,5169	
fruit-pods	0,6448	0,6448	0,9789	0,6836	0,8502	0,6448	0,6448	0,651	
fruit-seeds	0,0207	0,0207	0,0207	0,6224	0,724	0,675	0,675	0,675	

Рисунок 4.43 - Таблица со значимыми атрибутами

Здесь красным цветом выделены значимые атрибуты, вычисленные с помощью метода на основе результатов построения ROC-кривых и вычисления AUC.

4.10 Ансамблевые алгоритмы

Как уже отмечалось ранее, в программной системе реализовано несколько ансамблевых алгоритмов классификации, среди которых присутствуют два оригинальных алгоритма, а именно неоднородный ансамблевый алгоритм и его модификация. Модуль с ансамблевыми алгоритмами классификации расположен в пункте меню *Классификаторы>Ансамблевые алгоритмы* (рисунок 4.44).

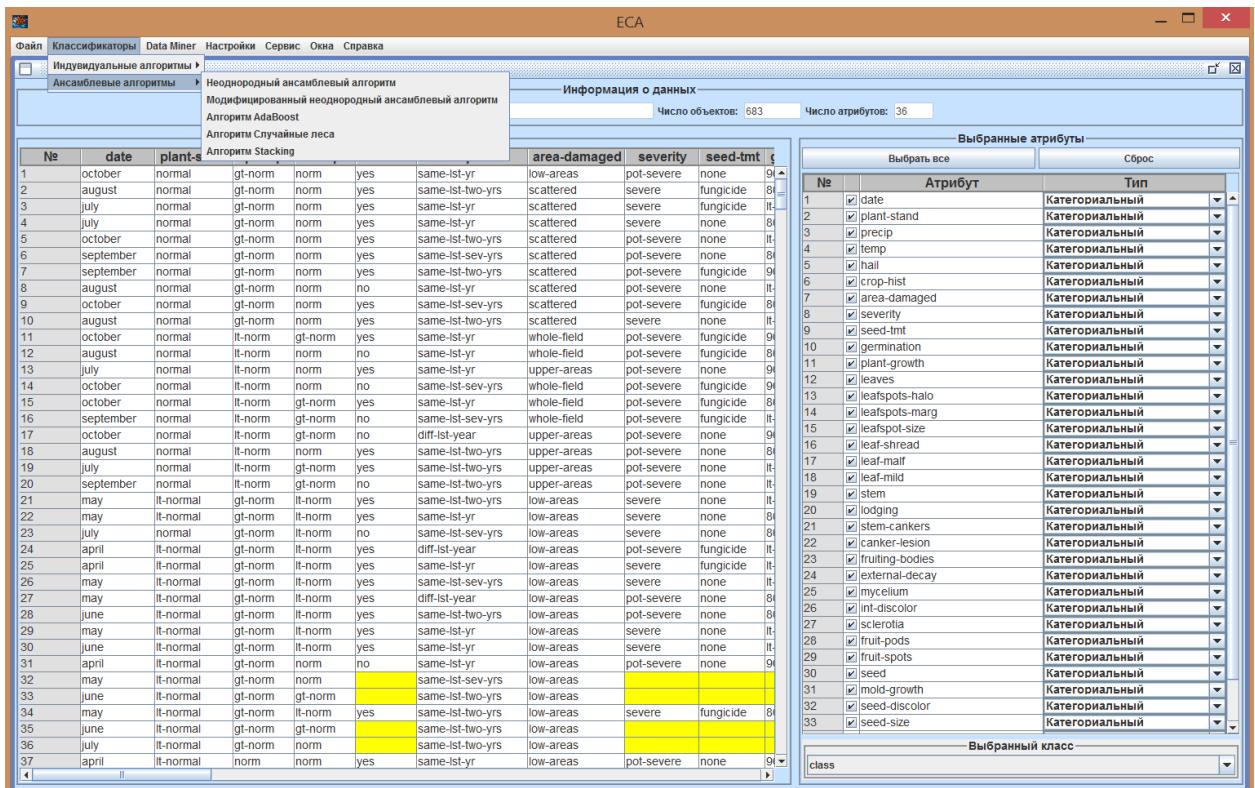


Рисунок 4.44 – Меню с ансамблевыми алгоритмами классификации

Также отметим, что в этих алгоритмах имеется возможность выбора любой комбинации индивидуальных алгоритмов, а также настройка их параметров.

4.10.1 Неоднородный ансамблевый алгоритм и его модификация

Для настройки параметров этих алгоритмов необходимо перейти в пункт меню *Классификаторы>Ансамблевые алгоритмы>Неоднородный ансамблевый алгоритм* или *Классификаторы>Ансамблевые алгоритмы>Модифицированный неоднородный ансамблевый алгоритм* (рисунок 4.45).

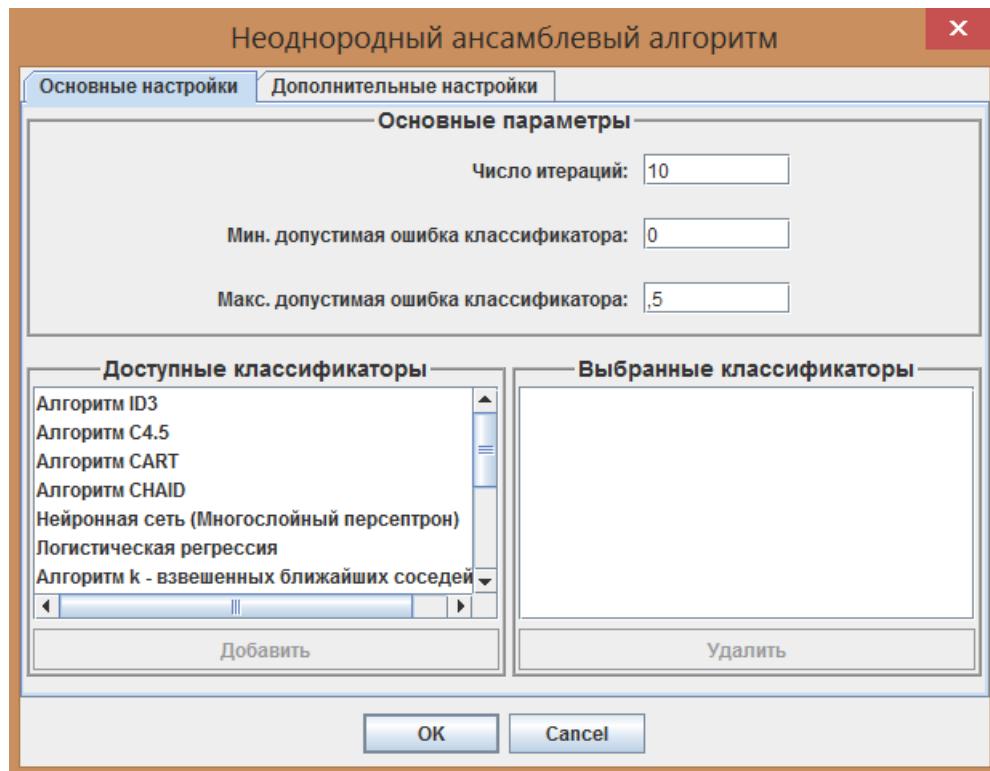


Рисунок 4.45 – Диалоговое окно с настройками параметров неоднородного ансамблевого алгоритма

Панель настроек алгоритмов состоит из двух вкладок. В первой вкладке содержатся основные параметры алгоритма, а во второй дополнительные. На рисунок 4.46 приведено изображение дополнительной вкладки.

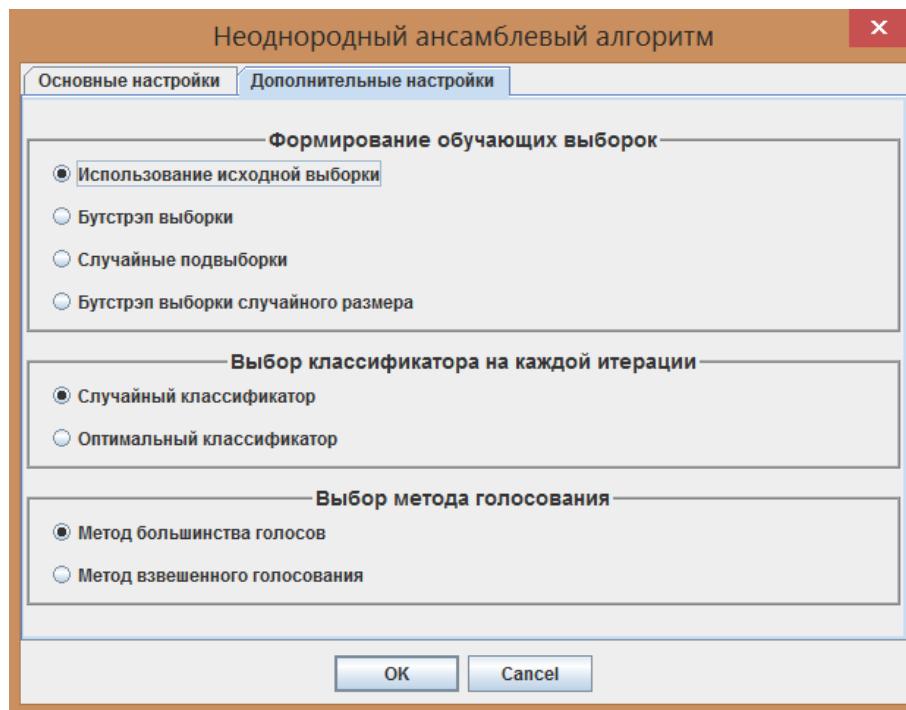


Рисунок 4.46 – Диалоговое окно с настройками параметров неоднородного ансамблевого алгоритма (дополнительные параметры)

Подробное описание входных параметров неоднородного ансамблевого алгоритма приведено в разделе 2.2.

Для того чтобы выбрать очередной индивидуальный алгоритм, который будет использоваться при построении ансамбля, необходимо выбрать его в списке «*Доступные классификаторы*» и затем нажать на кнопку «*Добавить*». После этого, выбранный алгоритм будет добавлен в список «*Выбранные классификаторы*» (рисунок 4.47). При желании вы можете удалить любой из выбранных классификаторов; для этого необходимо выбрать его в соответствующем списке и затем нажать на кнопку «*Удалить*».

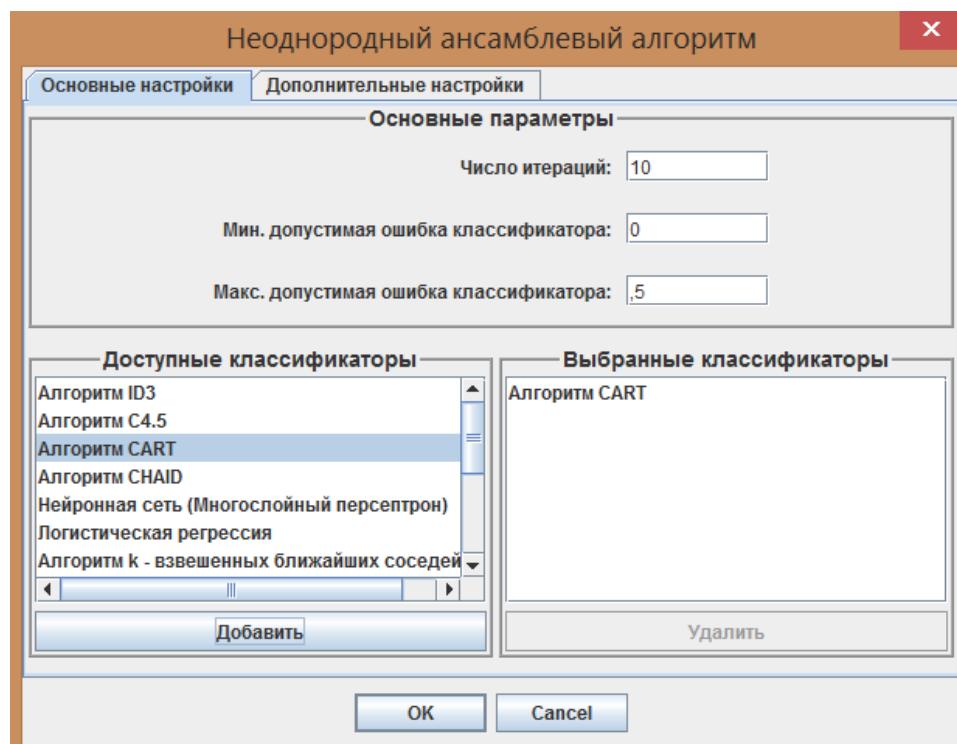


Рисунок 4.47 – Диалоговое окно с настройками параметров неоднородного ансамблевого алгоритма (выбор базовых алгоритмов)

Для того чтобы настроить параметры какого-либо выбранного индивидуального алгоритма классификации, необходимо выполнить двойной щелчок левой кнопкой мыши по названию этого алгоритма в списке «*Выбранные классификаторы*» и, затем, в появившемся диалоговом окне настроить его параметры (рисунок 4.48).

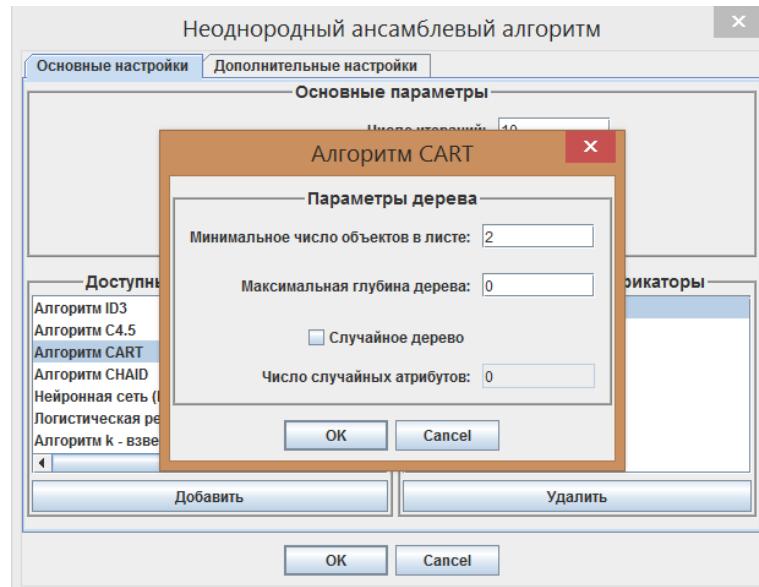


Рисунок 4.48 – Диалоговое окно с настройками параметров неоднородного ансамблевого алгоритма (настройка параметров базового алгоритма)

В конце отметим, что модифицированный неоднородный ансамблевый алгоритм отличается от базовой версии тем, что на каждой итерации формируется выборка со случаем числом r входных атрибутов, которые также отбираются случайным образом (равновероятно). На рисунке 4.49 приведено изображение диалогового окна с настройками параметров этого алгоритма.

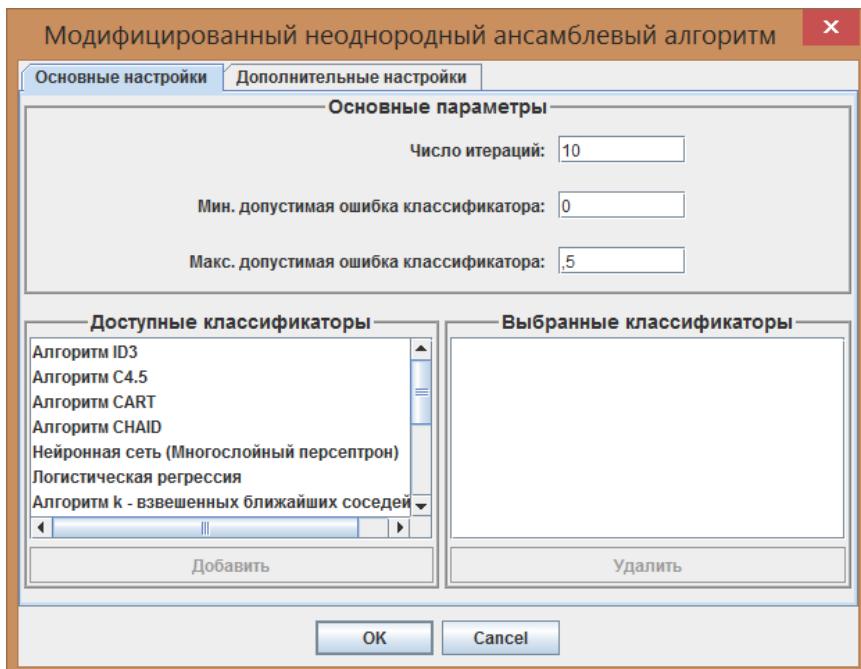


Рисунок 4.49 – Диалоговое окно с настройками параметров модифицированного неоднородного ансамблевого алгоритма

Здесь задаются точно такие же параметры, как и у неоднородного ансамблевого алгоритма.

4.10.2 Алгоритм «Случайные леса»

Данный алгоритм расположен в пункте меню *Классификаторы>Ансамблевые алгоритмы>Алгоритм Случайные леса* (рисунок 4.50).

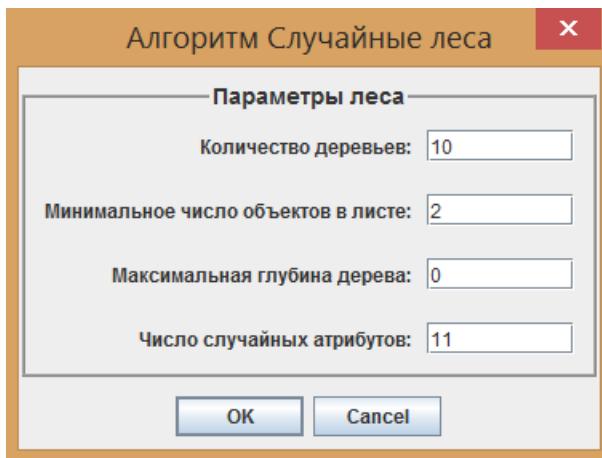


Рисунок 4.50 – Диалоговое окно с настройками параметров алгоритма «Случайные леса»

Для работы этого алгоритма необходимо задать следующие параметры:

- количество деревьев – T (по умолчанию – 10);
- минимальное количество объектов в листе (по умолчанию – 2);
- максимальная глубина каждого дерева (по умолчанию – 0, где 0 соответствует бесконечности);
- количество случайных атрибутов m (по умолчанию $k / 3$, где k – число входных атрибутов).

4.10.3 Алгоритм AdaBoost

Этот алгоритм расположен в пункте меню *Классификаторы>Ансамблевые алгоритмы>Алгоритм AdaBoost* (рисунок 4.51).

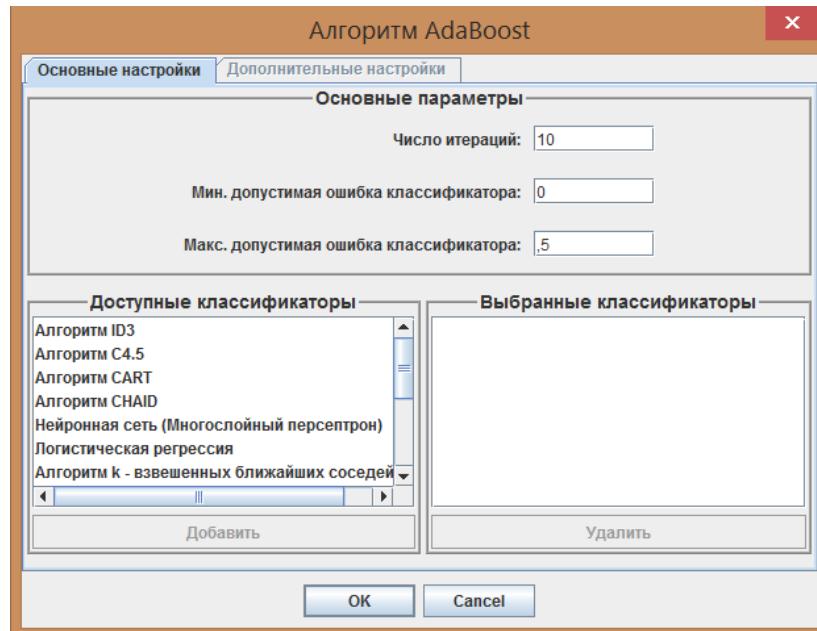


Рисунок 4.51 – Диалоговое окно с настройками параметров алгоритма *AdaBoost*

Для работы данного алгоритма нужно задать точно такие же основные параметры, как в неоднородном ансамблевом алгоритме.

4.10.4 Алгоритм Stacking

Алгоритм *Stacking* расположен в пункте меню *Классификаторы>Ансамблевые алгоритмы>Алгоритм Stacking* (рисунок 4.52).

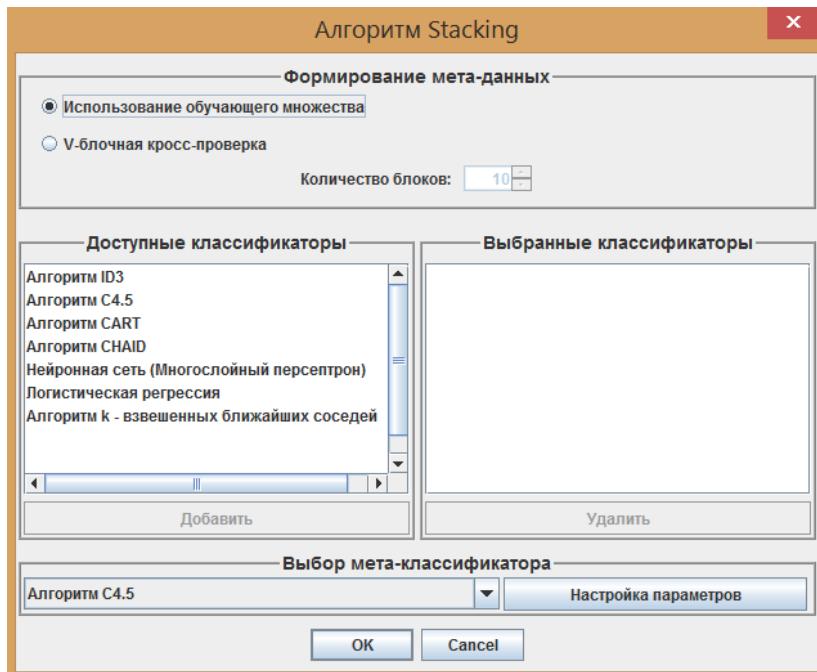


Рисунок 4.52 – Диалоговое окно с настройками параметров алгоритма *Stacking*

В алгоритме реализовано два метода формирования мета – множества:

- *использование обучающего множества* (используется по умолчанию). Мета – множество формируется на основе выходов базовых моделей, обученных на множестве X .
- V – *блочная кросс – проверка*. Здесь также необходимо задать количество блоков V (по умолчанию - 10). В этом методе мета – множество формируется на основе тестовых выборок с помощью V – *блочной кросс проверки*. Затем базовые классификаторы заново переобучаются на исходной обучающей выборке X .

Выбор и настройка параметров базовых алгоритмов осуществляется точно также, как у неоднородного ансамблевого алгоритма (раздел 4.10.2). Для настройки параметров мета – классификатора необходимо сначала выбрать его в списке, расположенным на панели «Выбор мета - классификатора», затем нажать на кнопку «Настройка параметров» и в появившемся диалоговом окне настроить параметры выбранного алгоритма (рисунок 4.53).

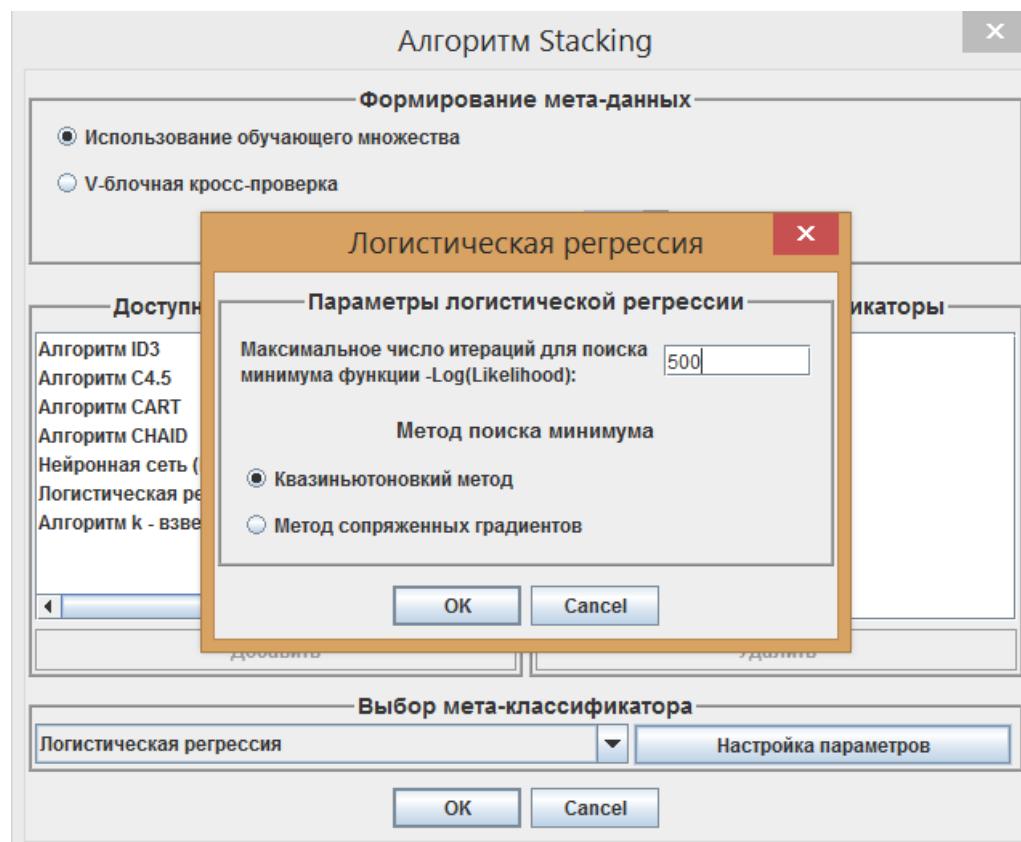


Рисунок 4.53 – Диалоговое окно с настройками параметров алгоритма *Stacking* (настройка параметров мета - классификатора)

Отметим также, что при смене названия мета – классификатора в списке, теряется информация о настройках всех предыдущих алгоритмов.

4.10.5 Просмотр структуры ансамбля

Для всех реализованных в программной системе ансамблевых алгоритмов классификации существует возможность просмотра структуры ансамбля, т.е. информацию о базовых классификаторах, входящих в ансамбль. На рисунке 4.54 приведен пример таблицы со структурой ансамбля для неоднородного ансамблевого алгоритма. Данная таблица доступна во вкладке, которая появляется в окне с результатами классификации для ансамблевых алгоритмов. Отметим, что при использовании алгоритма *Stacking*, последняя строка таблицы будет соответствовать мета – классификатору.

The screenshot shows a software interface with a title bar 'Неоднородный ансамблевый алгоритм'. Below the title bar is a menu bar with 'Файл', 'Сервис', and 'Справка'. Underneath the menu bar are four tabs: 'Результаты классификации' (selected), 'Классификация', 'ROC кривые', and 'Структура ансамбля'. The main area contains a table with two columns: '№' and 'Классификатор'. The '№' column contains integers from 0 to 9. The 'Классификатор' column lists various classifiers: C45, CART, NeuralNetwork, C45, NeuralNetwork, CART, NeuralNetwork, NeuralNetwork, Logistic, and C45. To the right of the table, there is a column labeled 'Результаты' with the text 'Посмотреть' repeated for each row. The entire table is set against a light gray background.

№	Классификатор	Результаты
0	C45	Посмотреть
1	CART	Посмотреть
2	NeuralNetwork	Посмотреть
3	C45	Посмотреть
4	NeuralNetwork	Посмотреть
5	CART	Посмотреть
6	NeuralNetwork	Посмотреть
7	NeuralNetwork	Посмотреть
8	Logistic	Посмотреть
9	C45	Посмотреть

Рисунок 4.54 – Таблица со структурой ансамбля

При наведении курсором мыши на название индивидуального классификатора, должна появиться всплывающая подсказка с входными параметрами этого классификатора (рисунок 4.55).

The screenshot shows a Windows application window titled 'Неоднородный ансамблевый алгоритм'. The menu bar includes 'Файл', 'Сервис', 'Справка', 'Результаты классификации', 'Классификация', 'ROC кривые', and 'Структура ансамбля'. The main area contains a table with columns '№' and 'Классификатор'. Row 2, labeled 'NeuralNetwork', is selected and expanded in a detailed view. This expanded view shows 'Входные параметры классификатора' (Input parameters of the classifier) with the following details:

Количество нейронов во входном слое:	35
Количество нейронов в выходном слое:	19
Количество скрытых слоев:	1
Структура скрытого слоя:	23
Максимальное число итераций:	1000000
Допустимая ошибка:	1.0E-5
Активационная функция нейронов скрытого слоя:	LogisticFunction
Активационная функция нейронов выходного слоя:	LogisticFunction
Алгоритм обучения:	BackPropagation
Коэффициент скорости обучения:	0.1
Коэффициент момента:	0.2

Рисунок 4.55 – Таблица со структурой ансамбля (входные параметры индивидуальных классификаторов)

Для того чтобы посмотреть информацию о результатах конкретного индивидуального классификатора, необходимо нажать на кнопку «Посмотреть» в соответствующей строке таблицы (рисунок 4.56). После этого появится окно с результатами классификации выбранного алгоритма.

The screenshot shows the same application window as in Figure 4.55, but for a specific classifier named 'NeuralNetwork'. The main area contains several sections of information:

- Статистика** (Statistics):

Исходные данные	soybean
Число объектов	683
Число атрибутов	36
Число классов	19
Классификатор	NeuralNetwork
Метод оценки точности	Использование обучающей выборки
Число объектов тестовых данных	683
- Результаты классификации** (Classification results):

Класс	TPR	FPR	TNR	FNR	Полнота	Точность	F - мера	AUC
0	1	0	1	0	1	1	1	1
1	1	0	1	0	1	1	1	1
2	1	0	1	0	1	1	1	1
- Матрица классификации** (Confusion matrix):

Реальное	0 (Прогнозное)	1 (Прогнозное)	2 (Прогнозное)	3 (Прогнозное)	4 (Прогнозное)	5 (П
0	16	0	0	0	0	0
1	0	21	0	0	0	0

Рисунок 4.56 – Окно с результатами индивидуального классификатора

Здесь также можно посмотреть информацию о входных параметрах классификатора, информацию о структуре обучающей выборки, на основе которой была построена модель классификатора и т.д.

4.11 Выводы по главе

В данной главе диссертационной работы приведено подробное описание пользовательского интерфейса программной системы *ECA* в целом, а именно, рассмотрены:

1. Установка программы *ECA* с помощью инсталлятора.
2. Загрузка исходных данных из файла, базы данных и сети.
3. Основные операции редактирования таблиц с исходными данными.
4. Таблицы с результатами классификации.
5. Диалоговые окна с настройками параметров алгоритмов классификации.

Также следует отметить основные достоинства реализации программной системы *ECA*:

1. Возможность параллельной работы с несколькими выборками данных. При этом существует возможность переключения с одной выборки на другую.
2. Возможность загрузки исходных данных из различных СУБД, таких как *MySQL*, *Oracle*, *PostgreSQL*, *SQL Server*, *MS Access*, *SQLite*.
3. Загрузка и сохранение моделей классификаторов в файл.
4. Удобная и быстрая настройка типов атрибутов, в отличие от других программных систем, таких как *Statistica* и *SPSS*.
5. Очень удобная визуализация деревьев решений и нейронных сетей.
6. Возможность настройки любой комбинации базовых классификаторов и их параметров, которые будут использованы при построении ансамбля.
7. Классификация нового примера на основе построенной модели классификатора.
8. Реализована возможность просмотра структуры ансамбля. Можно просматривать структуры составляющих ансамбля, а также их результаты классификации.

5 Разработка модуля Data Miner

Идея модуля *Data Miner* основана на проведении серии экспериментов над классификаторами с различными входными параметрами, и дальнейшем выборе на основе истории проведенных экспериментов оптимальных параметров для классификаторов. Критерием оптимальности здесь является максимизация точности классификации (в процентах) на обучающей или тестовой выборке. В программной системе реализован автоматический подбор оптимальных параметров для следующих алгоритмов: нейронные сети, неоднородный ансамблевый алгоритм и его модификация, алгоритм *AdaBoost*, алгоритм *Stacking*. В последующих разделах приводится подробное описание разработанных алгоритмов для автоматического подбора оптимальных параметров классификаторов и их реализация.

5.1 Автоматический подбор параметров для неоднородного ансамблевого алгоритма

Для неоднородного ансамблевого алгоритма и его модификации происходит автоматический подбор следующих параметров:

1. Множество базовых классификаторов $C = \{c_i(x)\}, i = \overline{1, r}$, где $c_i(x)$ – i -ый базовый классификатор, r – количество базовых классификаторов;
2. Метод формирования обучающих выборок на каждой итерации $s \in S = \{\text{INITIAL}, \text{BOOTSTRAP}, \text{RANDOM_SUBSAMPLE}, \text{RANDOM_BOOTSTRAP}\}$;
3. Метод выбора индивидуального классификатора на каждой итерации $i \in I = \{\text{OPTIMAL_CLASSIFIER}, \text{RANDOM_CLASSIFIER}\}$;
4. Метод голосования $a \in A = \{\text{MAJORITY_VOTES}, \text{WEIGHTED_VOTES}\}$.

Для организации процесса перебора всех возможных комбинаций из p базовых алгоритмов классификации во множестве из r классификаторов используется алгоритм *next_permutation*, который берет перестановочную последовательность (с повторениями) и возвращает следующую за ней. Если следующей перестановки не существует, то алгоритм возвращает *false*, иначе – *true* [26]. В качестве перестановочной последовательности выступает массив *mark[1...r]*, закодированный нулями и единицами, где единица означает, что классификатор используется в комбинации, ноль – не используется. Общее число таких перестановок может быть определено по формуле:

$$P(p, s) = \frac{r!}{p!s!}, \quad (5.1)$$

где r – общее число классификаторов, p – число используемых классификаторов, s – число неиспользуемых соответственно. Причем $p + s = r$. Отметим также что формула (6.1) является частным случаем формулы для вычисления числа перестановок с повторениями.

Общее число всех возможных комбинаций из r базовых алгоритмов классификации может быть определено по формуле:

$$P = \sum_{t=1}^r \frac{r!}{t!(r-t)!}, \quad (5.2)$$

где t - число используемых классификаторов, r – общее число базовых классификаторов.

Учитывая также перебор таких параметров как s, i, a , получаем общее число комбинаций входных параметров неоднородного ансамблевого алгоритма

$$K = 16P. \quad (5.3)$$

Таким образом, для работы алгоритма перебора параметров неоднородного ансамблевого алгоритма необходимо задать следующие исходные данные:

- $X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки, где x_i – i -ый объект обучающей выборки, n – количество объектов в выборке;
- $Y = I, \dots, l$ – номера классов (возможные значения классификаторов);
- число итераций – T ;
- порог ошибки для включения базовых классификаторов в ансамбль – M ;
- множество базовых алгоритмов классификации: $C = \{c_i(x)\}, i = \overline{1, r}$, где $c_i(x)$ – i -ый базовый классификатор, r – количество базовых классификаторов.

В общем случае алгоритм перебора всех возможных параметров неоднородного ансамблевого алгоритма можно представить в виде последовательности следующих шагов на каждой итерации $t = I, \dots, r$:

1. Формирование начального массива $mark[1\dots r]$, в котором первые t элементов закодированы единицами, а остальные нулями.
2. Вычисление следующей перестановки в массиве $mark[1\dots r]$ с помощью процедуры *next_permutation*.
3. Формирование множества базовых алгоритмов классификации $C_t = \{c_i(x)\}, i = \overline{1, t}$ на основе закодированного массива $mark[1\dots r]$.
4. Осуществление перебора всех возможных комбинаций дополнительных параметров неоднородного ансамблевого алгоритма, таких как s, i, a . Для каждой такой комбинации строится модель классификатора с использованием множества базовых классификаторов C_t .
5. Переход к шагу 2.

В листинге 5.1 приведено описание разработанного алгоритма автоматического подбора параметров неоднородного ансамблевого алгоритма в виде псевдокода.

Листинг 5.1. Псевдокод алгоритма автоматического подбора параметров неоднородного ансамблевого алгоритма

Вход:

$X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки;

$Y = I, \dots, l$ – номера классов (возможные значения классификаторов);

T - число итераций;

M - заданный порог ошибки для включения базовых классификаторов в ансамбль;

$C = \{c_i(x)\}, i = \overline{1, r}$ - множество базовых алгоритмов классификации;

Выход:

множество ансамблевых моделей $H = \{C_i(x)\}, i = \overline{1, K}$, где K – общее число моделей, которое определяется по формуле (6.3)

1. $H = \emptyset$
2. **for** $t = 1$ **to** r
3. **do for** $j = 1$ **to** r ► формируем массив $mark$
4. **do if** $j \leq t$
5. **then** $mark[j] = 1$
6. **else** $mark[j] = 0$
7. **while** next_permutation($mark$) ► берем следующую перестановку в $mark$
8. $C_t = \emptyset$
9. **do for** $j = 1$ **to** r ► формируем множество базовых классификаторов
10. **do if** $mark[j] = 1$
11. **then** $C_t = C_t \cup c_j(x)$
12. Для всех $s \in S$ ► перебор параметров s, i, a
13. Для всех $i \in I$
14. Для всех $a \in A$
15. Обучить классификатор $h(x)$ с входными параметрами $C_t, s,$
16. i, a, T, M на выборке X
17. $H = H \cup h(x)$

Рассмотрим приведенный псевдокод более подробно. На начальном этапе формируется пустое множество H построенных моделей классификаторов (строка 1). Затем в цикле от $t = 1, \dots, r$ выполняются основные шаги алгоритма. Сначала в строках 3 - 6 формируется начальный массив $mark$. Затем в цикле *while* перебираются все возможные перестановки с повторениями в массиве $mark$. В строках 9 – 11 формируется множество базо-

вых алгоритмов классификации, которое впоследствии будет использоваться при построении ансамбля. Затем в строках 12 - 17 осуществляется перебор всех возможных комбинаций параметров s, i, a . И в строке 17 происходит добавление построенного классификатора во множество H .

5.2 Автоматический подбор параметров для нейронной сети

Известно, что настройка параметров нейронных сетей является нетривиальной задачей. Поэтому было принято решение разработать модуль, позволяющий строить различные конфигурации нейронных сетей и на их основе выбирать наилучшие по точности. В разработанном алгоритме варьируются следующие входные параметры:

1. Активационная функция нейронов скрытого слоя $f \in F = \{LOGISTIC, TANH, SINE, EXPONENTIAL\}$.
2. Структура скрытого слоя (число нейронов в одном скрытом слое).

Для работы алгоритма необходимо задать следующие исходные данные:

- $X = \{x_i\}, i = \overline{1, n}$ – множество объектов обучающей выборки, где x_i – i -ый объект обучающей выборки, n – количество объектов в выборке;
- $Y = 1, \dots, l$ – номера классов (возможные значения классификаторов);
- множество активационных функций нейронов скрытого слоя $F = \{LOGISTIC, TANH, SINE, EXPONENTIAL\}$;
- число итерации – T .

Алгоритм построения различных конфигураций нейронных сетей можно представить в виде последовательности следующих шагов на каждой итерации $t = 1, \dots, T$:

1. Случайным образом (равновероятно) выбираем активационную функцию нейронов скрытого слоя f из множества F .
2. Случайным образом выбираем число нейронов в одном скрытом слое k . Причем число k должно лежать в интервале, который вычисляется по формуле 1.5.
3. Строим нейронную сеть с одним скрытым слоем и активационной функцией нейронов скрытого слоя f . Число нейронов в скрытом слое должно быть равно k .
4. Обучаем нейронную сеть с помощью алгоритма обратного распространения ошибки.

5.3 Реализация модуля Data Miner

В данном разделе будет подробно рассмотрена реализация модуля *Data Miner*. Все классы этого модуля расположены в пакете *esa.experiment*. Отметим также, что некоторые классы реализованы таким образом, чтобы была возможность итеративного построения

эксперимента. Такая реализация позволяет отображать текущий прогресс построения эксперимента на пользовательском интерфейсе. На рисунке 5.1 приведена диаграмма классов разработанного модуля *Data Miner*.

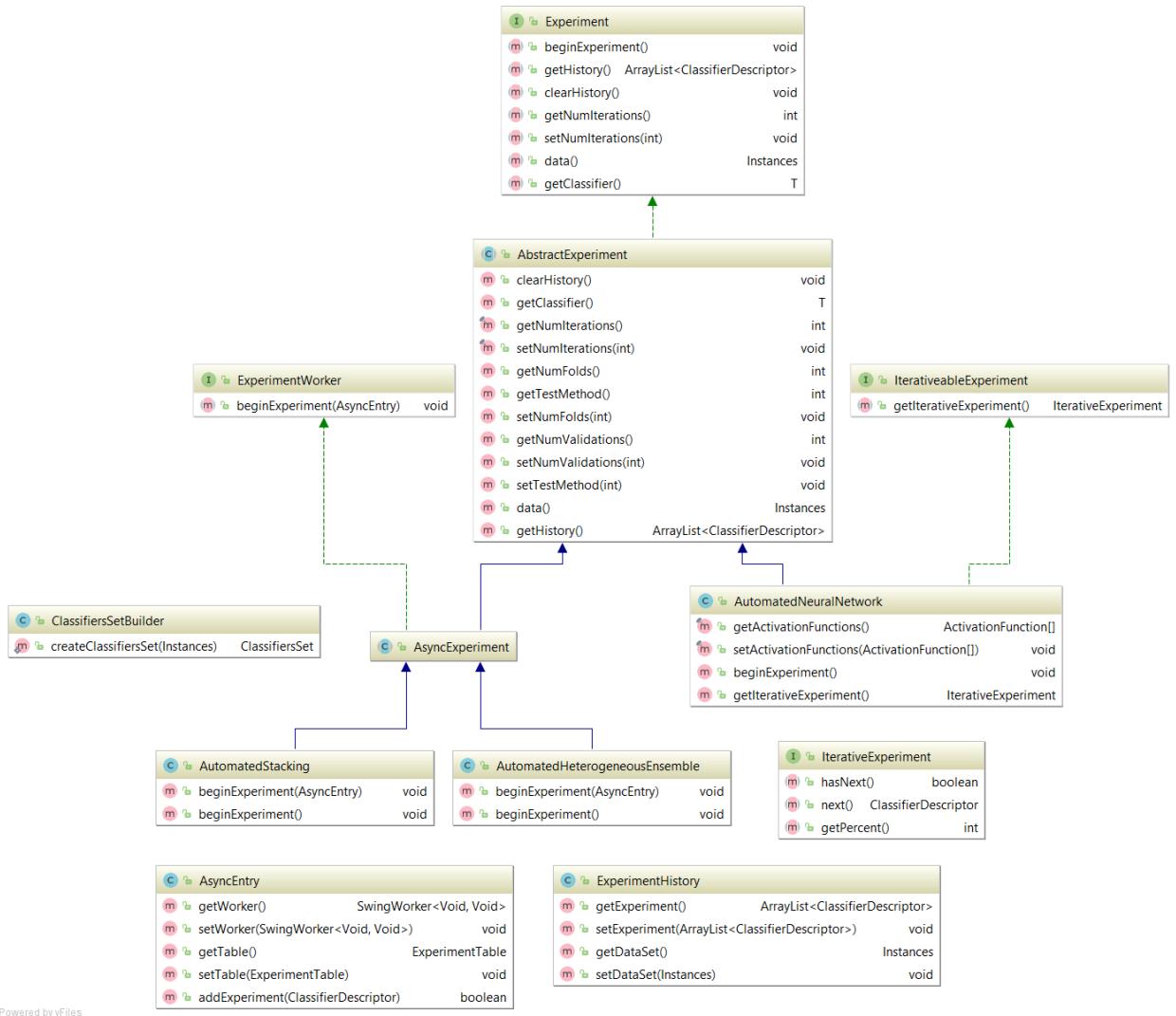


Рисунок 5.1 – Диаграмма классов модуля *Data Miner*

Далее, рассмотрим все эти классы по отдельности. Главный интерфейс *Experiment<T>* содержит основные методы, с помощью которых можно управлять экспериментом. Параметр обобщения *T* задает классификатор, который должен реализовывать интерфейс *Classifier* из библиотеки *weka*. Отметим также, что текущая реализация поддерживает автоматический подбор параметров только одного алгоритма классификации. В таблице 5.1 приведено описание всех методов этого интерфейса.

Таблица 5.1 – Описание методов класса *Experiment*

Название метода	Описание метода
<i>void beginExperiment()</i>	Выполняет построение эксперимента, при этом предыдущая история эксперимента очищается.
<i>ArrayList<ClassifierDescriptor> getHistory()</i>	Возвращает историю эксперимента в виде списка. Где класс <i>ClassifierDescriptor</i> содержит такие поля, как объект классификатора и объект типа <i>Evaluation</i> из библиотеки <i>weka</i> , в котором хранятся результаты построенной модели.
<i>void clearHistory()</i>	Выполняет очистку истории эксперимента.
<i>Instances data()</i>	Возвращает ссылку на объект классификатора типа <i>T</i> .
<i>T getClassifier()</i>	Возвращает ссылку на объект данных, которые используются при построении моделей классификаторов в ходе эксперимента.
<i>int getNumIterations()</i>	Возвращает число итераций эксперимента.
<i>void setNumIterations(int numIterations)</i>	Задает число итераций эксперимента.

Абстрактный класс *AbstractExperiment<T>* реализует интерфейс *Experiment<T>* и содержит основные методы для управления экспериментом, а также настройку его параметров. В таблице 5.2 приведены основные методы этого класса.

Таблица 5.2 – Описание методов класса *AbstractExperiment*

Название метода	Описание метода
<i>int getNumFolds()</i>	Возвращает число блоков V для $k \times V$ – блочной кросс – проверки на тестовой выборке.
<i>int getTestMethod()</i>	Возвращает целочисленную константу, которая задает метод оценки точности. Она может принимать следующие значения: <ul style="list-style-type: none"> - <i>TestMethod.TRAINING_SET</i> - <i>TestMethod.CROSS_VALIDATION</i>
<i>intgetNumValidations()</i>	Возвращает число проверок k для $k \times V$ – блочной кросс – проверки на тестовой выборке.
<i>void setNumFolds(int numFolds)</i>	Задает число блоков V для $k \times V$ – блочной кросс – проверки на тестовой выборке.
<i>void setTestMethod(int mode)</i>	Задает метод оценки точности. Генерирует исключение типа <i>IllegalArgumentException</i> в случае задания неправильного значения.
<i>void setNumValidations(int numValidations)</i>	Задает число проверок k для $k \times V$ – блочной кросс – проверки на тестовой выборке.

Интерфейс *IterativeExperiment* реализует логику пошагового построения эксперимента. В таблице 5.3 приведено описание основных методов этого интерфейса.

Таблица 5.3 – Описание методов класса *IterativeExperiment*

Название метода	Описание метода
<i>ClassifierDescriptor next()</i>	Выполняет следующий шаг эксперимента и возвращает ссылку на объект типа <i>ClassifierDescriptor</i> . Генерирует исключение <i>NoSuchElementException</i> , если эксперимент полностью проведен.
<i>boolean hasNext()</i>	Проверяет, есть ли следующий шаг алгоритма.

Окончание таблицы 5.3

Название метода	Описание метода
<code>int getPercent()</code>	Возвращает значение, лежащее в интервале от [0, 100], которое соответствует текущему проценту построения эксперимента.

Интерфейс ***IterativeableExperiment***, который предоставляет единственный метод для создания объектов типа ***IterativeExperiment***.

Класс ***AutomatedNeuralNetwork*** реализует логику автоматического подбора параметров для нейронных сетей. Отметим также, что данная реализация допускает итеративное построение нейронных сетей. В таблице 5.4 приведено описание основных методов этого класса.

Таблица 5.4 – Описание методов класса *AutomatedNeuralNetwork*

Название метода	Описание метода
<code>IterativeExperiment getIterativeExperiment()</code>	Возвращает ссылку на объект типа <i>IterativeExperiment</i> , предназначенного для итеративного построения нейронных сетей.
<code>void setActivationFunctions(ActivationFunction[] activationFunctions)</code>	Задает множество активационных функций нейронов скрытого слоя, которые будут использоваться при построении нейронных сетей.
<code>ActivationFunction[] getActivationFunctions()</code>	Возвращает массив активационных функций нейронов скрытого слоя.

Класс ***ExperimentHistory*** представляет собой структуру данных для хранения истории эксперимента. В таблице 5.5 приведены основные методы этого класса.

Таблица 5.5. – Описание методов класса *ExperimentHistory*

Название метода	Описание метода
<code>ArrayList<ClassifierDescriptor> getExperiment()</code>	Возвращает список построенных моделей классификаторов.
<code>void setExperiment(ArrayList<ClassifierDescriptor> experiment)</code>	Задает список построенных моделей классификаторов.

Окончание таблицы 5.5

Название метода	Описание метода
<i>Instances getDataSet()</i>	Возвращает ссылку на объект исходных данных, на основе которых строились модели классификаторов.
<i>void setDataSet(Instances dataset)</i>	Задает объект типа <i>Instances</i> , на основе которых строились модели классификаторов.

Класс **AsyncEntry** реализует логику асинхронного построения эксперимента, а также пополнение таблицы с уже построенными моделями классификаторов. В таблице 5.6 приведены основные методы этого класса.

Таблица 5.6 – Описание методов класса *AsyncEntry*

Название метода	Описание метода
<i>SwingWorker<Void,Void> getWorker()</i>	Возвращает ссылку на объект типа <i>SwingWorker</i> из библиотеки <i>swing</i> .
<i>ExperimentTable getTable()</i>	Возвращает ссылку на объект таблицы, в которой хранятся уже построенные модели классификаторов.
<i>void setWorker(SwingWorker<Void,Void> worker)</i>	Задает объект типа <i>SwingWorker</i> .
<i>void settable(ExperimentTable table)</i>	Задает объект таблицы, в которой хранятся уже построенные модели классификаторов.
<i>void addExperiment(ClassifierDescriptor obj)</i>	Добавляет классификатор в таблицу моделей классификаторов.

Интерфейс **ExperimentWorker** предназначен для реализации логики асинхронного построения эксперимента. Он содержит единственный метод *beginExperiment(AsyncEntry entry)*, который выполняет асинхронное построение эксперимента.

Классы **AutomatedStaking** и **AutomatedHeterogeneousEnsemble** реализуют логику автоматического подбора параметров для ансамблевых алгоритмов *Stacking* и неоднородного ансамблевого алгоритма и его модификации соответственно. Отметим также, что конструктор этого класса в качестве аргумента принимает ссылку на любой объект классификатора, являющегося подклассом класса **AbstractHeterogeneousClassifier**. Это означает, что если переданный объект классификатора не является подклассом класса

HeterogeneousClassifier, то будут осуществляться только перебор всех возможных комбинаций базовых алгоритмов классификации.

5.4 Описание пользовательского интерфейса модуля Data Miner

5.4.1 Описание основных частей интерфейса

Все алгоритмы для автоматического подбора параметров классификаторов находятся в пункте меню *Data Miner*. В общем случае главное окно модуля *Data Miner* выглядит следующим образом (рисунок 5.2).

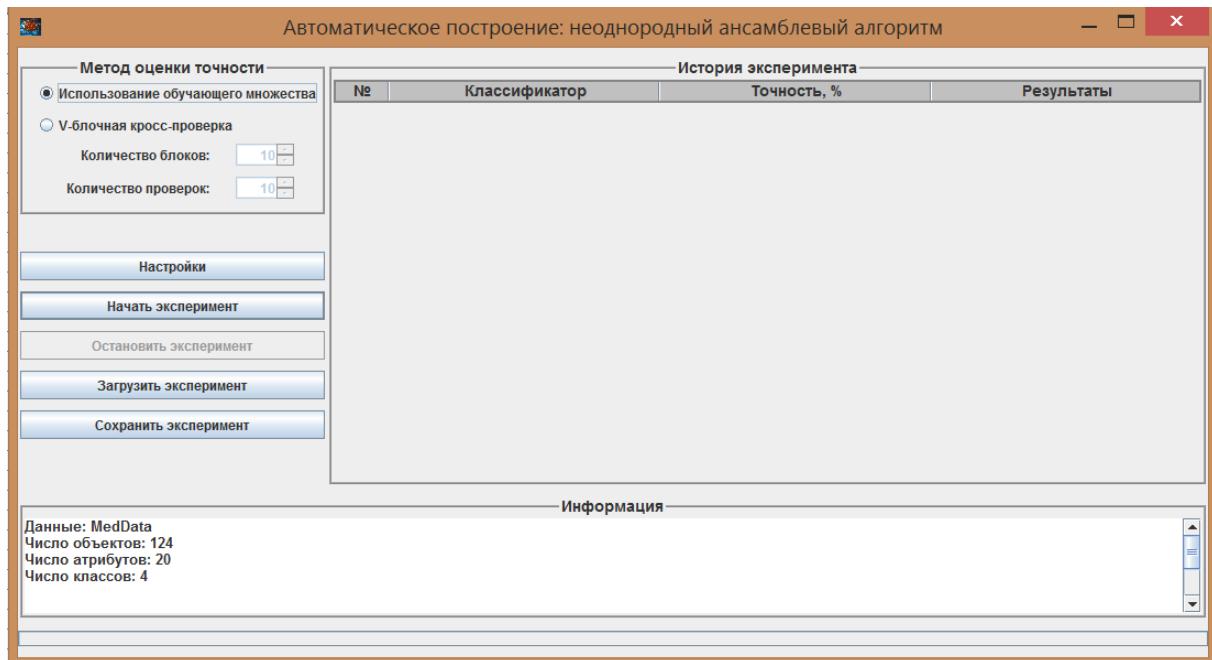


Рисунок 5.2 – Главное окно модуля Data Miner

Оно состоит из основных четырех частей:

1. Панель для выбора метода оценки точности. Здесь можно выбрать один из двух методов: использование обучающего множества или $k \times V$ – блочная кросс – проверка на тестовой выборке.
2. Панель с основными управляющими кнопками, такими как «Настройки», «Начать эксперимент», «Остановить эксперимент», «Сохранить эксперимент» и «Загрузить эксперимент».
3. Таблица с историей эксперимента. Здесь, в ходе проведения эксперимента, будут отображаться построенные модели классификаторов.
4. Нижняя текстовая панель, предназначенная для отображения прогресса эксперимента и выдачи рекомендуемых параметров для классификаторов после окончания эксперимента.

В большинстве случаев проведение эксперимента требует длительного времени, поэтому в программе присутствует возможность остановки эксперимента с помощью кнопки «*Остановить эксперимент*». При этом, в нижнем текстовом поле будут выданы рекомендуемые параметры для классификаторов на основе построенных моделей из верхней таблицы.

Все возможные варианты использования модуля *Data Miner* приведены на схеме (рисунок 5.3).

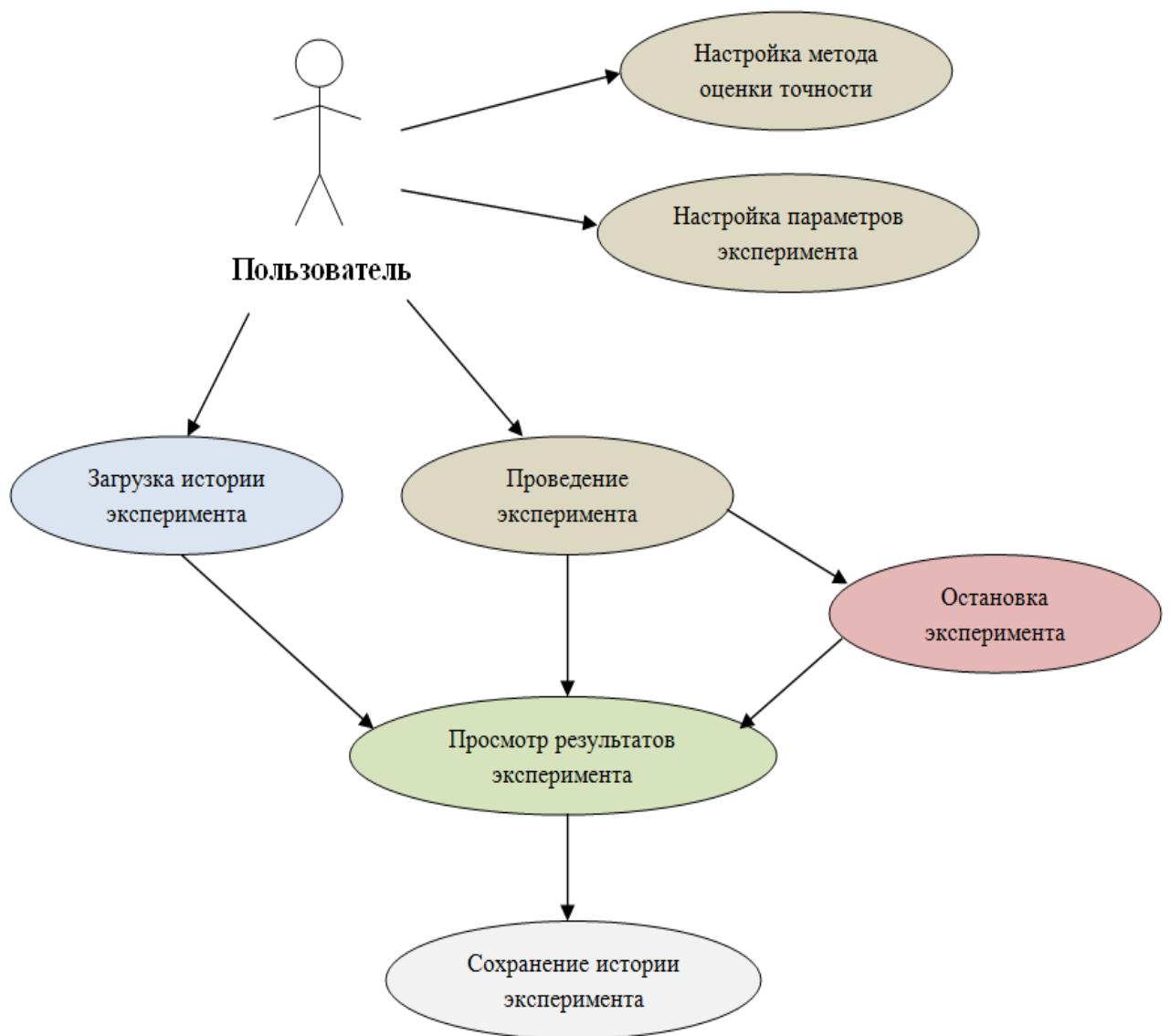


Рисунок 5.3 – Обобщенная схема вариантов использования модуля *Data Miner*

Для того чтобы сохранить историю эксперимента в текстовый файл, необходимо нажать на кнопку «Сохранить эксперимент». По умолчанию задается имя файла в формате [Классификатор Эксперимент Дата проведения] (рисунок 5.4).

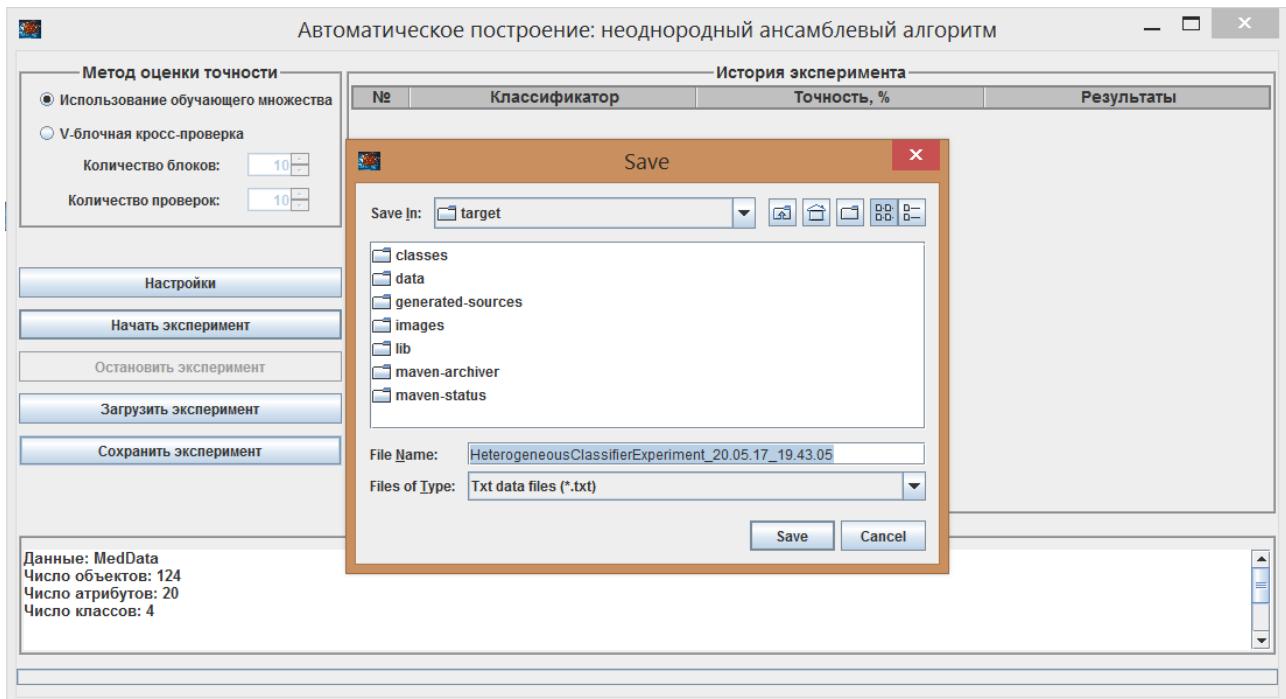


Рисунок 5.4 – Сохранение истории эксперимента в файл

Загрузить историю эксперимента можно с помощью кнопки «Загрузить эксперимент», затем выбрать файл с экспериментом в появившемся диалоговом окне и нажать на кнопку «Open» (рисунок 5.5).

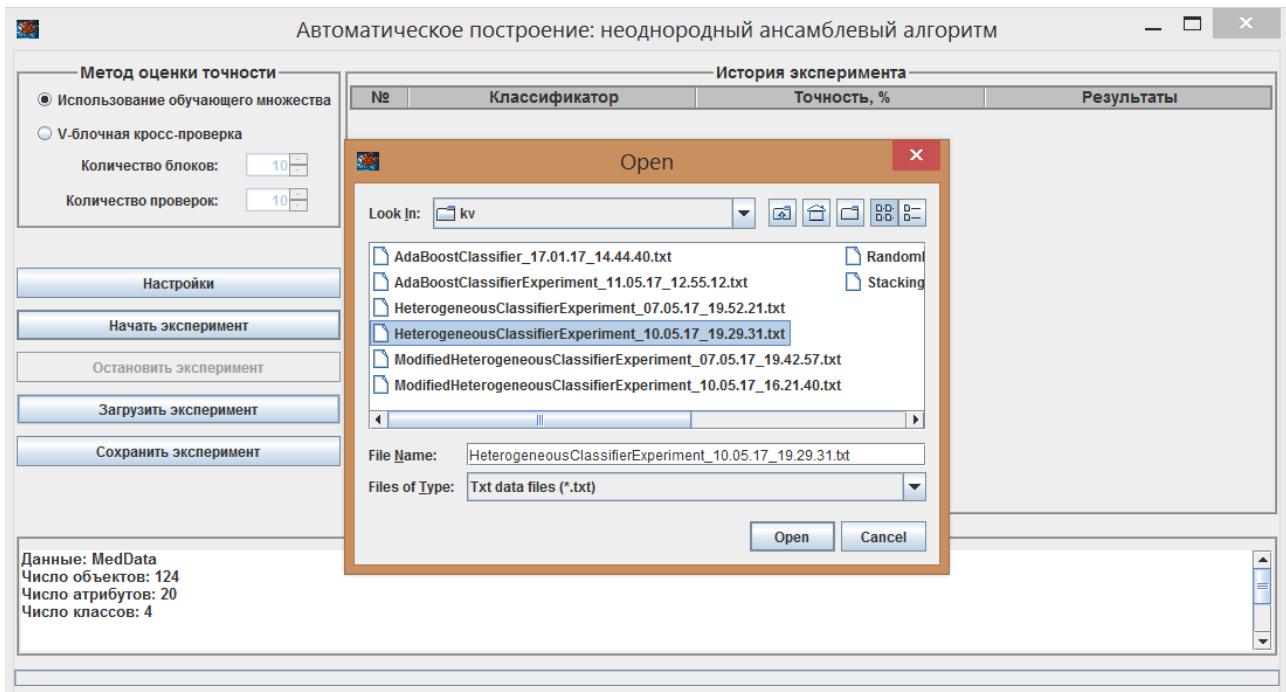


Рисунок 5.5 – Загрузка истории эксперимента из файла

После этого история эксперимента отобразится в верхней таблице (рисунок 5.6). Здесь вы можете посмотреть результаты каждой построенной модели, нажав на соответствующую кнопку «Посмотреть» (рисунок 5.7).

Автоматическое построение: неоднородный ансамблевый алгоритм

История эксперимента			
№	Классификатор	Точность, %	Результаты
0	HeterogeneousClassifier	92,7419	Посмотреть
1	HeterogeneousClassifier	92,7419	Посмотреть
2	HeterogeneousClassifier	91,9355	Посмотреть
3	HeterogeneousClassifier	91,9355	Посмотреть
4	HeterogeneousClassifier	91,129	Посмотреть
5	HeterogeneousClassifier	91,129	Посмотреть
6	HeterogeneousClassifier	91,129	Посмотреть
7	HeterogeneousClassifier	91,129	Посмотреть
8	HeterogeneousClassifier	91,129	Посмотреть
9	HeterogeneousClassifier	91,129	Посмотреть
10	HeterogeneousClassifier	91,129	Посмотреть
11	HeterogeneousClassifier	91,129	Посмотреть
12	HeterogeneousClassifier	87,0968	Посмотреть
13	HeterogeneousClassifier	87,0968	Посмотреть
14	HeterogeneousClassifier	86,2903	Посмотреть
15	HeterogeneousClassifier	86,2903	Посмотреть
16	HeterogeneousClassifier	86,2903	Посмотреть
17	HeterogeneousClassifier	86,2903	Посмотреть
18	HeterogeneousClassifier	86,2903	Посмотреть

Информация

Эксперимент завершен.
Данные: MedData
Число объектов: 124
Число атрибутов: 20
Число классов: 4
Метод оценки точности: Использование обучающего множества

Рисунок 5.6 – Таблица с историей проведенного эксперимента

Автоматическое построение: неоднородный ансамблевый алгоритм
HeterogeneousClassifier

Результаты классификации Классификация ROC кривые Структура ансамбля

Статистика

Число классов	4
Классификатор	HeterogeneousClassifier
Метод оценки точности	Использование обучающей выборки
Число объектов тестовых данных	124
Число правильно классифицированных объектов	115
Число неправильно классифицированных объектов	9
Точность классификатора, %	92,7419
Ошибка классификатора, %	7,2581

Результаты классификации

Класс	TPR	FPR	TNR	FNR	Полнота	Точность	F - мера	AUC
0	1	0,2727	0,7273	0	1	0,91	0,9529	0,9958
1	0,72	0	1	0,28	0,72	1	0,8372	0,9935
2	0,5	0	1	0,5	0,5	1	0,6667	0,9958

Матрица классификации

Реальное	0 (Прогнозное)	1 (Прогнозное)	2 (Прогнозное)	3 (Прогнозное)
0	91	0	0	0
1	7	18	0	0
2	2	0	2	0

Сохранить

Рисунок 5.7 – Просмотр результатов построенной модели

5.4.2 Автоматическое построение нейронных сетей

Модуль для построения нейронных сетей находится в пункте меню *Data Miner>Автоматическое построение: нейронные сети* (рисунок 5.8). В этом модуле строятся различные конфигурации нейронных сетей с различными активационными функциями и структурами скрытого слоя.

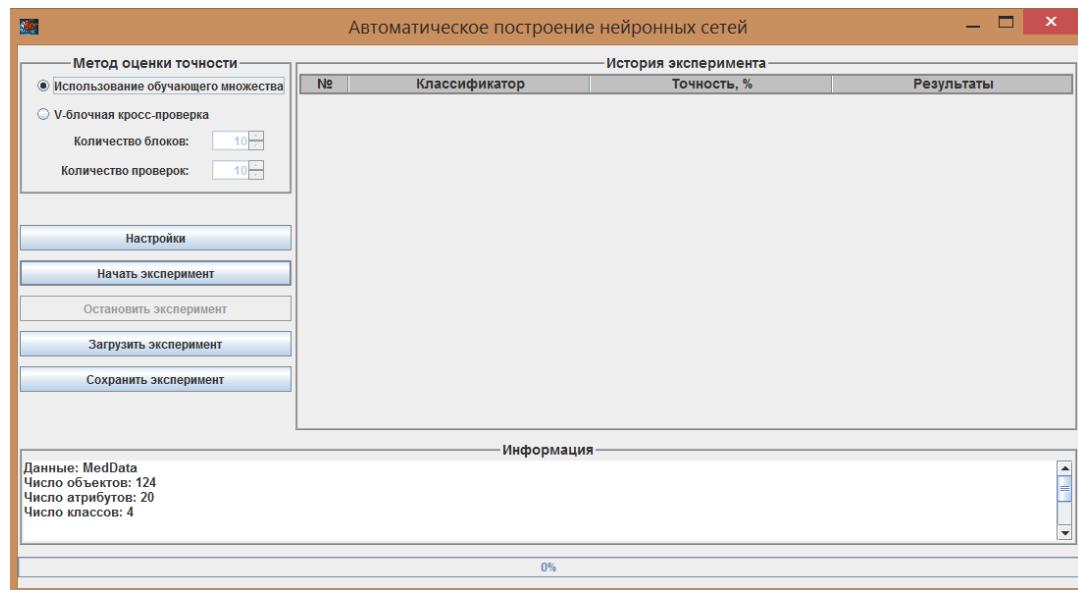


Рисунок 5.8 – Окно модуля *Data Miner* для построения нейронных сетей

При нажатии на кнопку «*Настройки*», вы можете задать число нейронных сетей (по умолчанию - 100), которые должны быть построены (рисунок 5.9). Также вы можете выбрать один из методов оценки точности классификации.

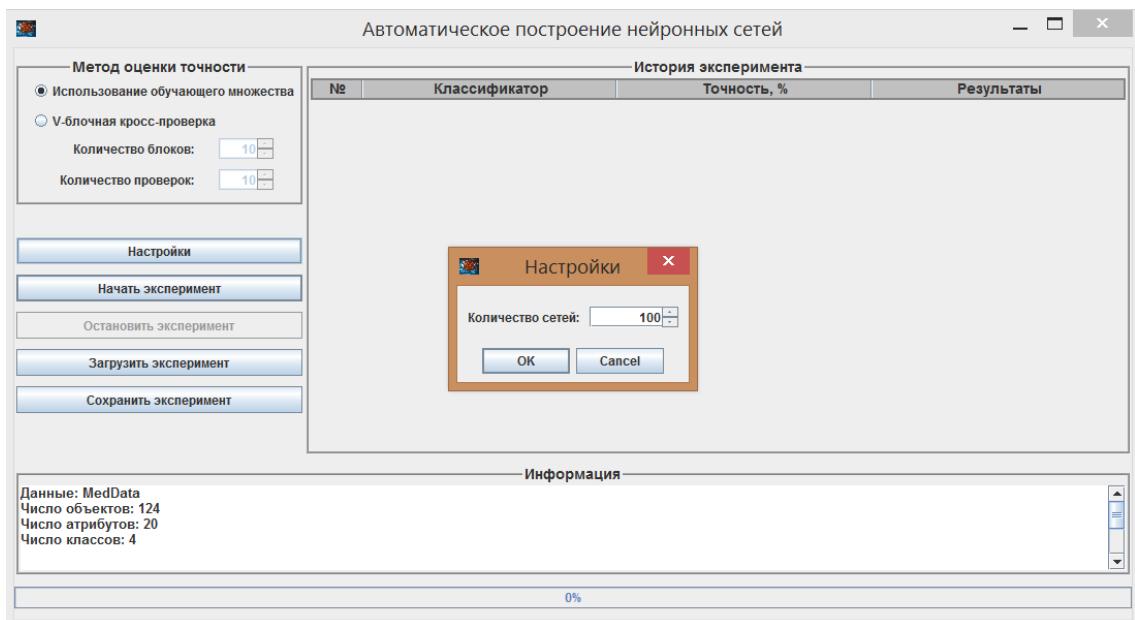


Рисунок 5.9 – Настройка числа нейронных сетей

После того как все параметры настроены, необходимо нажать на кнопку «Начать эксперимент» и ждать окончания эксперимента, или же в любой момент остановить эксперимент (рисунок 5.10). Отметим также, что вы можете просматривать результаты уже построенных нейронных сетей в ходе эксперимента.

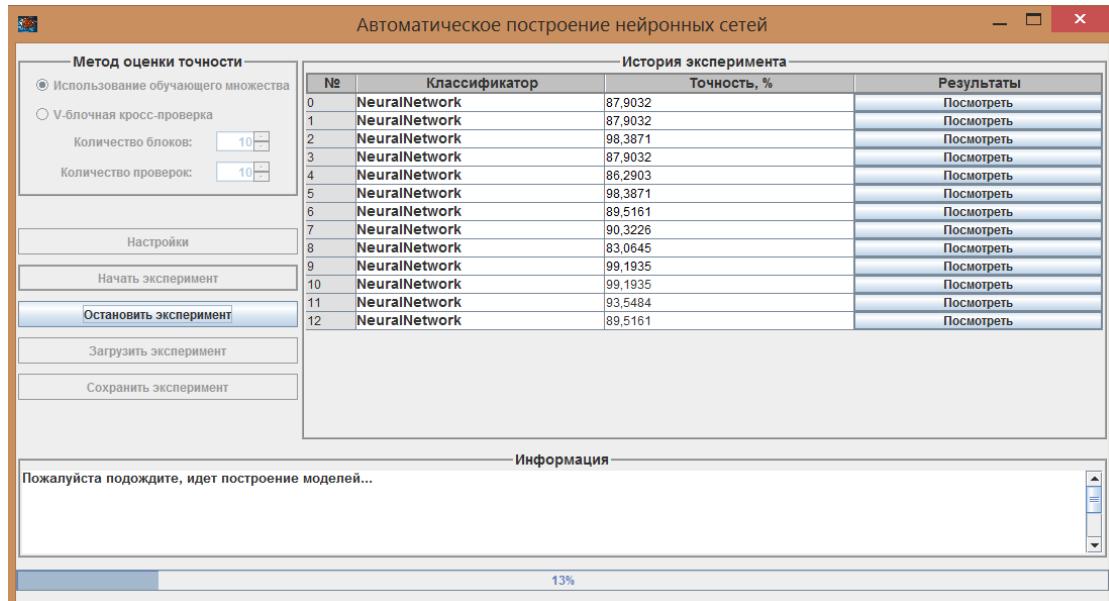


Рисунок 5.10 – Построение нейронных сетей

После окончания эксперимента все данные в таблице будут отсортированы по точности классификации и в нижнем текстовом поле будут выданы рекомендуемые параметры для классификаторов на основе построенных моделей из верхней таблицы. Также, наилучшие модели классификаторов, будут выделены красным цветом (рисунок 5.11).

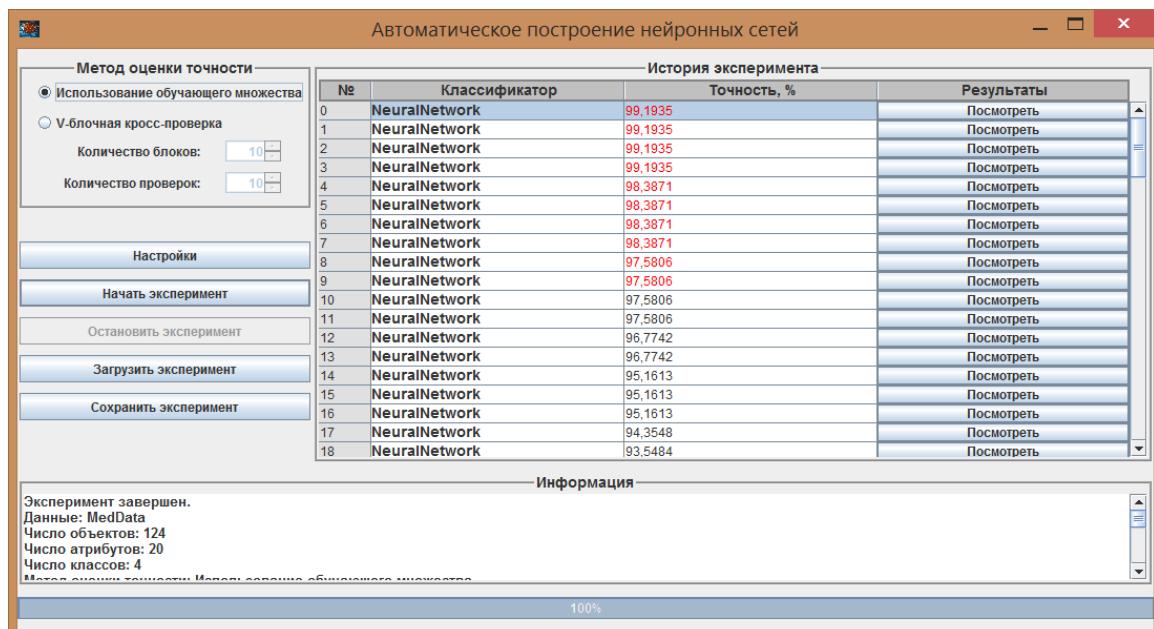


Рисунок 5.11 – Результаты эксперимента

5.4.3 Автоматическое построение ансамблевых алгоритмов

Все эти алгоритмы также находятся в пункте меню *Data Miner*. Здесь для выбранного ансамблевого алгоритма строятся различные конфигурации ансамблей с различными входными параметрами, такими как метод формирования обучающих выборок, процедура голосования и т.д., при этом осуществляется перебор всех возможных комбинаций из множества базовых классификаторов, которое также можно задать в настройках. По умолчанию это множество уже задано, но если вы хотите задать свое множество используемых базовых алгоритмов, то необходимо нажать на кнопку «Настройки» и в появившемся диалоговом окне выбрать базовые алгоритмы классификации и настроить их параметры (рисунок 5.12).

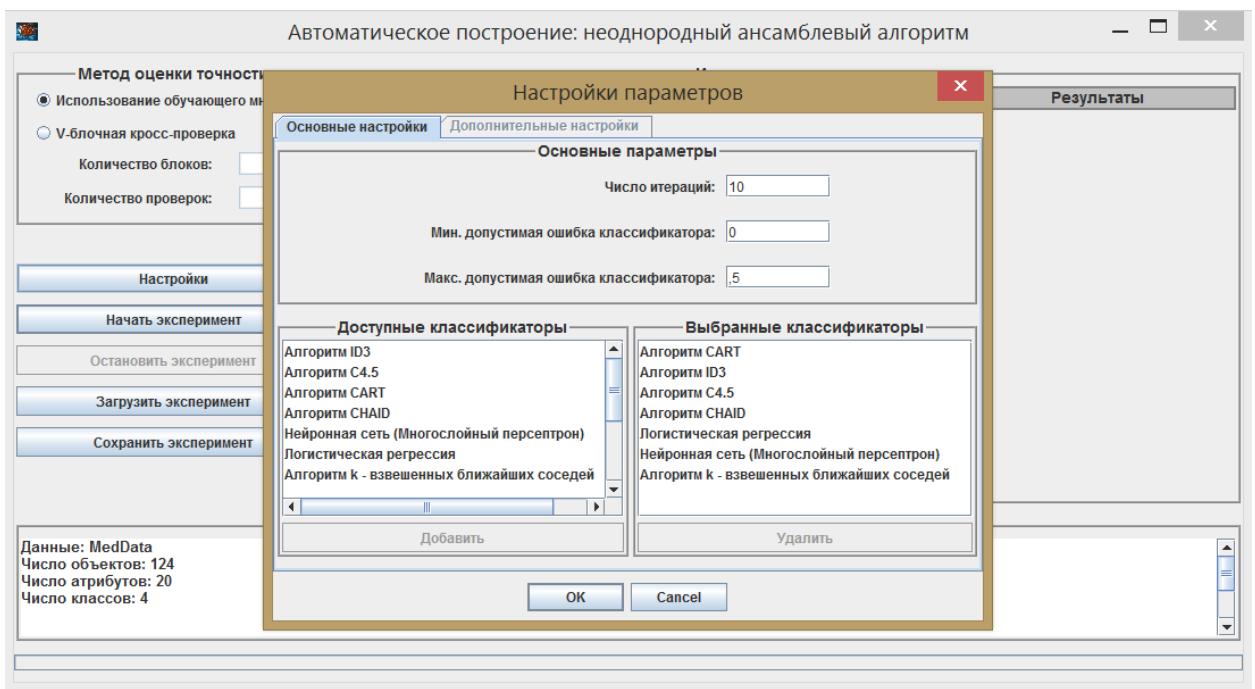


Рисунок 5.12 – Настройка параметров ансамблевого алгоритма

Здесь также можно задать число итераций для одного ансамблевого алгоритма (по умолчанию - 10) и значение допустимой ошибки (по умолчанию – 0,5) для включения классификатора в ансамбль.

Также следует отметить, что при использовании в качестве метода оценки точности $k \times V$ – блочной кросс – проверки не рекомендуется задавать слишком большое количество базовых классификаторов, т. к. при этом перебор всех возможных комбинаций параметров ансамблевого алгоритма может занимать очень много времени.

5.5 Выводы по главе

Таким образом, в данной главе были детально рассмотрены основные функции разработанного модуля *Data Miner*, а также его реализация в программной системе *ECA*. Данный модуль позволяет строить ансамблевые классификаторы с различными входными параметрами в автоматическом режиме, что является очень удобным, т.к. процесс настройки параметров неоднородных ансамблевых алгоритмов не тривиален, и занимает много времени. К отличительным особенностям этой разработки также следует отнести:

1. Оригинальный разработанный алгоритм для автоматического подбора оптимальных параметров для ансамблевых алгоритмов.
2. Возможность автоматического построения различных конфигураций для нейронных сетей.
3. Удобный пользовательский интерфейс.
4. Программная реализация этого модуля позволяет запускать процесс автоматического подбора оптимальных параметров асинхронно. При этом присутствует возможность просмотра результатов уже построенных моделей классификаторов в процессе эксперимента.
5. Возможность сохранения и загрузки истории эксперимента из файла.

6 Исследование разработанных алгоритмов и программных средств

6.1 Тестирование программной системы ECA

Данный раздел посвящен тестированию разработанных ансамблевых алгоритмов на известных данных из репозитория данных по машинному обучению. Тестирование преследовало следующие основные цели:

1. Выявление практической пригодности разработанных алгоритмов для решения задачи классификации на реальных медицинских данных.
2. Сравнение разработанных ансамблевых алгоритмов с одиночными алгоритмами, а также стандартными ансамблевыми алгоритмами.

В качестве исходных данных для тестирования использовалась выборка *ionosphere* из репозитория данных по машинному обучению [27]. Эти данные содержат информацию о 351 объекте. Данные содержат информацию о 17 импульсах, каждый из которых представляет собой комплексное число, соответствующее двум числовым атрибутам.

Для сравнения моделей используются основные характеристики качества классификаторов такие как:

- точность классификатора в процентах;
- показатели *AUC*;
- дисперсия ошибки классификатора (только для метода 10 блочной кросс – проверки).

Для тестирования использовались одиночные алгоритмы, такие как: деревья решений (*CART*, *C4.5*, *ID3*, *CHAID*), логистическая регрессия, алгоритм k – взвешенных ближайших соседей и нейронная сеть (многослойный персептрон) с логистической активационной функцией нейронов. В качестве алгоритма обучения нейронной сети использовался алгоритм обратного распространения ошибки.

В качестве ансамблевых классификаторов использовались такие алгоритмы как: неоднородный ансамблевый алгоритм и его модификация, алгоритм *AdaBoost*, *RandomForests* и *Stacking*. Все они были построены с использованием модуля *Data Miner*. Для обозначения входных параметров классификаторов использовались следующие сокращения (таблица 6.1).

Таблица 6.1 – Обозначения входных параметров классификаторов

Обозначение	Расшифровка
Деревья решений	
N_{\min}	Минимальное число объектов в листе
D_{\max}	Максимальная глубина дерева
N_{rand}	Число случайных атрибутов при расщеплении вершины
N_{trees}	Число деревьев
Логистическая регрессия (<i>Logistic</i>)	
L	Число итерации для поиска минимума логарифмической функции правдоподобия
OM	Метод поиска минимума
Нейронная сеть	
ε	Порог допустимой ошибки сети
i_{\max}	Максимальное число итерации для обучения
AF	Активационная функция нейронов скрытого слоя
η	Коэффициент скорости обучения
m	Коэффициент момента
N_h	Число нейронов в скрытом слое
Ансамблевые алгоритмы	
S	Метод формирования обучающих выборок на каждой итерации
I	Выбор индивидуального классификатора на каждой итерации
A	Метод голосования
T	Число итерация
M	Порог ошибки для включения базовых классификаторов в ансамбль
c	Множество базовых классификаторов

Продолжение таблицы 6.1

Обозначение	Расшифровка
<i>Majority_votes</i>	Метод большинства голосов
<i>Weighted_votes</i>	Метод взвешенного голосования
<i>Random_classifier</i>	Случайный классификатор
<i>Optimal_classifier</i>	Оптимальный классификатор
<i>Bootstrap_sample</i>	Бутстрэп – выборки
<i>Random_subsample</i>	Случайные подвыборки
<i>Meta_classifier</i>	Мета классификатор
<i>Random_bootstrap</i>	Бутстрэп – выборки случайного размера
Алгоритм k – взвешенных ближайших соседей (KNN)	
<i>K</i>	Число ближайших соседей
<i>Q</i>	Вес ближайшего соседа
<i>D</i>	Функция расстояния

В таблице 6.2 приведены результаты классификации с помощью одиночных и ансамблевых алгоритмов. Для оценки точности классификаторов на тестовой выборке была использована 10 блочная кросс – проверка.

Таблица 6.2 – Результаты классификации данных *ionosphere*

№	Алгоритм	Входные параметры	На обучающей выборке		На тестовой выборке		
			Точность классификатора, %	AUC для всех классов	Точность классификатора, %	AUC для всех классов	Дисперсия ошибки классификатора
1	Алгоритм <i>ID3</i>	$N_{\min} = 2$ $D_{\max} = \infty$	98,5755	AUC(0) = 0,9897 AUC(1) = 0,9897	88,8889	AUC(0) = 0,875 AUC(1) = 0,875	0,0034
2	Алгоритм <i>C4.5</i>	$N_{\min} = 2$ $D_{\max} = \infty$	98,5755	AUC(0) = 0,9897 AUC(1) = 0,9897	89,4587	AUC(0) = 0,8821 AUC(1) = 0,8821	0,001
3	Алгоритм <i>CART</i>	$N_{\min} = 2$ $D_{\max} = \infty$	97,7208	AUC(0) = 0,9872 AUC(1) = 0,9872	89,7436	AUC(0) = 0,9096 AUC(1) = 0,9096	0,0057
4	Алгоритм <i>CHAID</i>	$N_{\min} = 2$ $D_{\max} = \infty, \alpha = 0,05$	97,7208	AUC(0) = 0,9872 AUC(1) = 0,9872	88,604	AUC(0) = 0,8754 AUC(1) = 0,8754	0,0009
5	Логистическая регрессия	$L = 500$ $OM = Quasinewton$	93,7322	AUC(0) = 0,9815 AUC(1) = 0,9815	88,8889	AUC(0) = 0,8681 AUC(1) = 0,8681	0,0023
6	Нейронная сеть	$\varepsilon = 10^{-4}, i_{\max} = 10^6$ $\eta = 0,1, M = 0,2$ $AF = Logistic, N_h = 22$	90,5983	AUC(0) = 0,9428 AUC(1) = 0,9428	86,0399	AUC(0) = 0,8899 AUC(1) = 0,8899	0,0052
7	Алгоритм <i>k</i> – взвешенных ближайших соседей	$K = 10$ $Q = 1$ $D = Chebyshev$	87,7493	AUC(0) = 0,9832 AUC(1) = 0,9832	88,604	AUC(0) = 0,9338 AUC(1) = 0,9338	0,0025

Окончание таблицы 6.2

№	Алгоритм	Входные параметры	На обучающей выборке		На тестовой выборке		
			Точность классификатора, %	AUC для всех классов	Точность классификатора, %	AUC для всех классов	Дисперсия ошибки классификатора
8	Алгоритм Stacking	$c = \{C4.5, Logistic\}$ <i>Meta_classifier – C4.5</i>	98,5755	AUC(0) = 0,9912 AUC(1) = 0,9912	92,5926	AUC(0) = 0,9256 AUC(1) = 0,9256	0,0022
9	Алгоритм Random Forests	$N_{min} = 2$ $D_{max} = \infty$ $N_{rand} = 11, N_{trees} = 10$	97,15	AUC(0) = 0,9986 AUC(1) = 0,9986	92,0228	AUC(0) = 0,9641 AUC(1) = 0,9641	0,0046
10	Алгоритм AdaBoost	$T = 10, M = 0,5$ $c = \{CART, Logistic\}$	100	AUC(0) = 1 AUC(1) = 1	93,7322	AUC(0) = 0,9712 AUC(1) = 0,9712	0,0021
11	Неоднородный ансамблевый алгоритм	$T = 10, M = 0,5$ $A = Majority_votes,$ $S = Bootstrap_sample,$ $I = Random_classifier$ $c = \{C4.5, CHAID, Logistic\}$	98,5755	AUC(0) = 0,9998 AUC(1) = 0,9998	93,4473	AUC(0) = 0,9658 AUC(1) = 0,9658	0,0039
12	Модифицированный неоднородный ансамблевый алгоритм	$T = 10, M = 0,5$ $A = Weighted_votes$ $S = Random_subsample$ $I = Optimal_classifier$ $c = \{CHAID, Logistic\}$	97,4359	AUC(0) = 0,9901 AUC(1) = 0,9901	94,0171	AUC(0) = 0,9688 AUC(1) = 0,9688	0,0008

Из полученных результатов видно, что наилучший результат на тестовой выборке как по точности, так и по дисперсии ошибки классификации был получен при использовании модифицированного неоднородного ансамблевого алгоритма со следующими входными параметрами:

- $T = 10$;
- $M = 0,5$;
- $A = \text{Weighted_votes}$;
- $S = \text{Random_subsample}$;
- $I = \text{Random_subsample}$;
- $c = \{\text{CHAID}, \text{Logistic}\}$.

Точность классификации с помощью этого алгоритма на обучающей выборке составила 97,4359%, на тестовой – 94,0171%, дисперсия ошибки классификации на тестовой выборке составила 0,0008. Лучший результат по точности классификации на тестовой выборке с помощью одиночного алгоритма соответствует дереву решений *CART* – 89,7436%, что ниже точности неоднородного ансамблевого алгоритма более чем на 4%. Наилучший по точности результат с использованием стандартных ансамблевых алгоритмов соответствует алгоритму *AdaBoost* и составляет 93,7322%, что ниже точности модифицированного неоднородного ансамблевого алгоритма. Также следует отметить, что неоднородный ансамблевый алгоритм тоже показал очень хорошие результаты, которые уступают только алгоритму *AdaBoost*.

Далее проведем *ROC* – анализ, для этого построим графики *ROC* – кривых для модифицированного неоднородного ансамблевого алгоритма на обучающей и тестовой выборках (рисунки 6.1 – 6.2).

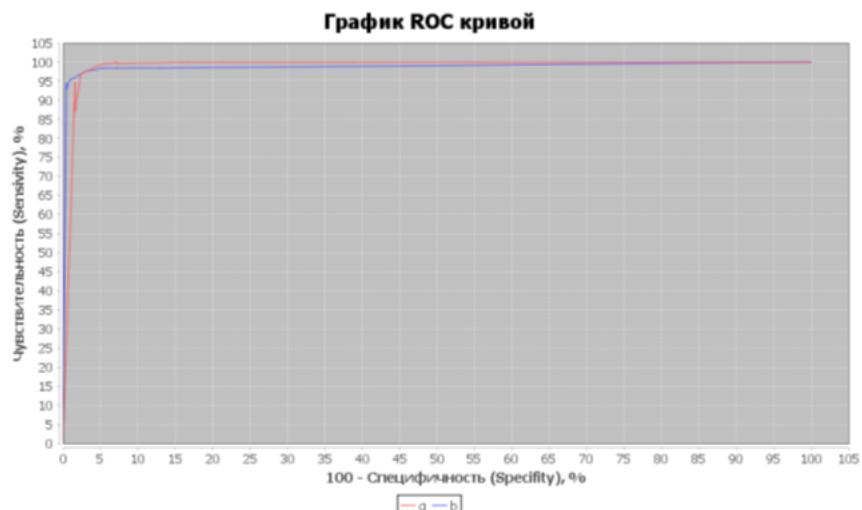


Рисунок 6.1 – Графики ROC – кривых для модифицированного неоднородного ансамблевого алгоритма на обучающей выборке

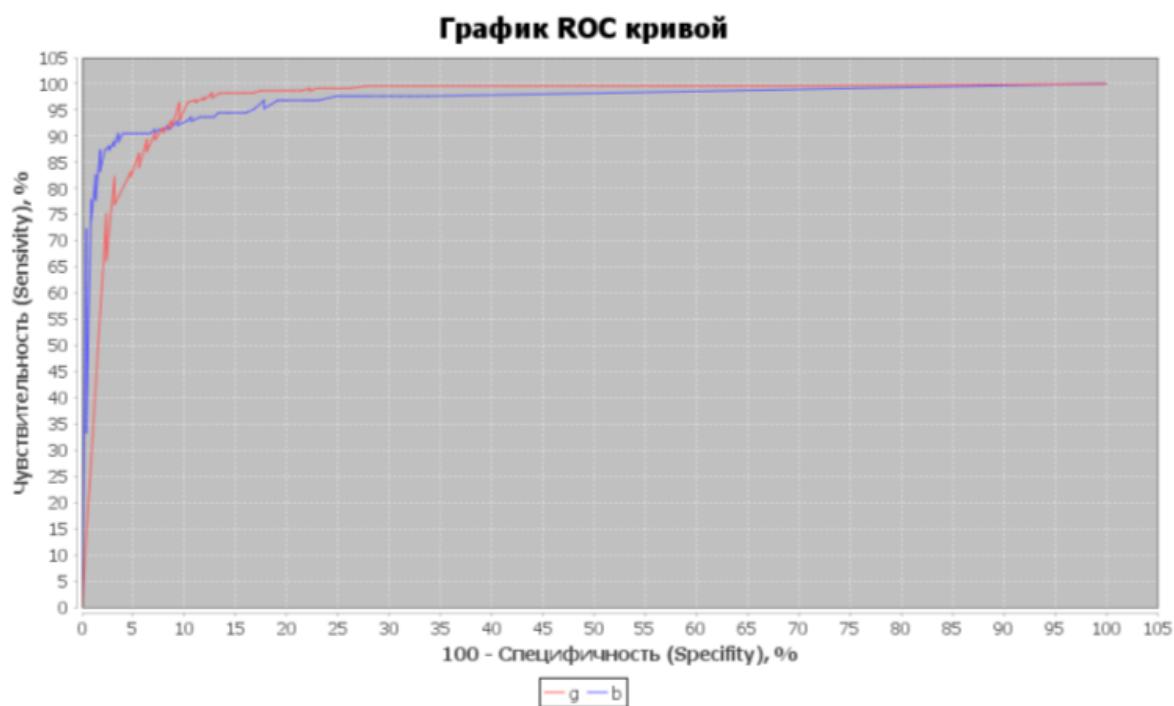


Рисунок 6.2 – Графики ROC – кривых для модифицированного неоднородного ансамблевого алгоритма на тестовой выборке

По результатам *ROC* – анализа видно, что показатели *AUC* на обучающей выборке и тестовой выборке для каждого класса близки к единице, что говорит об отличном качестве построенной модели классификатора.

Таким образом, результаты проведенного тестирования показали эффективность использования оригинальных разработанных ансамблевых алгоритмов, а также их перспективность для исследования данных.

6.2 Исследование реальных медицинских данных

6.2.1 Описание исходных данных

Программная система *ECA* тестировалась на реальных медицинских данных о хирургическом лечении больных с патологией аорты. Данные предоставлены НИИПК им. акад. Е.Н. Мешалкина Минздрава России. Эти данные содержат информацию о 124 пациентах, оперированных на восходящем отделе и дуге аорты с использованием различных хирургических технологий. Данные содержат 7 числовых, 2 порядковых и 11 номинальных признаков. Необходимо определить тип нарушения мозгового кровообращения (НМК) (результатирующий признак, класс) для каждого пациента на основе анализа исходных признаков о его состоянии. Описание всех признаков приведено в таблице 6.3.

Таблица 6.3 – Описание признаков больных с патологией аорты

№	Признак	Описание
Категориальные признаки		
1	Пол	0 – женский 1 – мужской
2	Этиология	0 – синдром Такаясу 1 – атеросклероз 2 – дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
3	Предшествующие операции на сердце	0 – да 1 – нет
4	Ишемическая болезнь сердца (ИБС)	0 – да 1 – нет
5	Артериальная гипертензия (АГ)	0 – да 1 – нет
6	Хроническая обструктивная болезнь легких (ХОБЛ)	0 – да 1 – нет
7	Патология почек	0 – да 1 – нет
8	Тип реконструкции дуги	0 – да 1 – нет

Окончание таблицы 6.3

№	Признак	Код с описанием
9	Операция Борста	0 – да 1 – нет
10	Вид перфузии головного мозга	0 – ретроградная перфузия 1 – без перфузии 2 – антеградная перфузия
11	Вмешательство на корне аорты	0 – операция Бенталла 1 – супракоронарное протезирование (операция Де Бейки, Вольфа) 2 – операция Дэвида
12	Стадия расслоения аорты	0 – острая 1 – хроническая 2 – подострая
13	Тип нарушения мозгового кровообращения (НМК)	0 – нет 1 – гипоксическая энцефалопатия 2 – инсульт 3 – транзиторные ишемические атаки
Числовые признаки		
14	Возраст	лет
15	Рост	см.
16	Вес	кг.
17	Искусственное кровообращение (ИК)	мин.
18	Окклюзия аорты (OA)	мин.
19	Время циркуляторного ареста (ЦА)	мин.
20	Количество дней в больнице	

6.2.2 План исследования

Одним из тяжелых осложнений после операций на проксимальной аорте являются нарушения мозгового кровообращения (НМК) различной степени выраженности. Для практической медицины очень важно выявить вероятность возникновения НМК в раннем послеоперационном периоде и связать степень выраженности НМК со значениями выделенных признаков риска. Для решения этой задачи план эксперимента был построен следующим образом:

1. Построение модели логистической регрессии на основе всех имеющихся признаков.
2. Определение значимых признаков на основе построенной модели логистической регрессии.
3. Построение и оценка точности моделей базовых классификаторов с использованием значимых признаков на исходном обучающем множестве и с помощью метода $k \times V$ – блочной кросс – проверки на тестовой выборке.
4. Построение и оценка точности ансамблевых моделей с различными входными параметрами с использованием значимых признаков на исходном обучающем множестве и с помощью метода $k \times V$ – блочной кросс – проверки на тестовой выборке. Причем базовые классификаторы в ансамблевых алгоритмах должны использовать точно такие же входные параметры, как в п. 3.
5. Сравнение и выбор наилучших по точности моделей.
6. Анализ и интерпретация результатов исследования.

Для сравнения моделей использовались основные характеристики качества классификаторов такие как:

- точность классификатора в процентах;
- показатели AUC ;
- дисперсия ошибки классификатора (только для метода 10×10 блочной кросс – проверки);
- 95% доверительный интервал ошибки классификатора (только для метода 10×10 блочной кросс – проверки).

6.2.3 Проведение исследования

Для проведения эксперимента в программу *ECA* были загружены медицинские данные, которые хранятся в файле *MedData.xlsx*. На рис. 6.3 приведена таблица с исходными данными и типами признаков, которые были определены автоматически.

№	Возраст	Пол	Кол-во дней в больнице	Этиология
1	38	мужской	46	синдром Тахаясу
2	57	мужской	70	атеросклероз
3	52	мужской	1	атеросклероз
4	45	мужской	50	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
5	39	мужской	46	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
6	30	мужской	32	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
7	48	мужской	34	атеросклероз
8	37	женский	21	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
9	53	мужской	27	атеросклероз
10	36	мужской	52	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
11	35	мужской	47	атеросклероз
12	52	женский	19	атеросклероз
13	47	женский	55	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
14	64	мужской	49	атеросклероз
15	50	мужской	36	атеросклероз
16	47	мужской	30	синдром Тахаясу
17	63	мужской	35	атеросклероз
18	34	женский	50	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
19	34	мужской	38	атеросклероз
20	55	мужской	39	атеросклероз
21	51	мужской	25	атеросклероз
22	49	мужской	20	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
23	57	мужской	58	атеросклероз
24	57	мужской	36	атеросклероз
25	50	мужской	34	атеросклероз
26	52	мужской	24	атеросклероз
27	40	женский	28	синдром Тахаясу
28	45	мужской	24	атеросклероз
29	55	мужской	34	атеросклероз
30	42	мужской	24	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
31	26	женский	33	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
32	25	женский	26	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
33	60	мужской	37	атеросклероз
34	41	мужской	30	дисплазии соединительной ткани (синдром Марфана, Элерса-Данло)
35	36	мужской	17	атеросклероз
36	45	женский	13	атеросклероз
37	42	женский	30	атеросклероз

Рисунок 6.3 – Таблица с медицинскими данными

Для определения значимых признаков была построена модель логистической регрессии со следующими входными параметрами:

- максимальное число итераций – 500;
- метод поиска минимума – *Quasinewton*.

Значимые признаки были определены по результатам *ROC* – анализа. На рис. 6.4 изображена таблица со значимыми признаками, которые выделены красным цветом.

Атрибут	AUC (Класс 0)	AUC (Класс 1)	AUC (Класс 2)	AUC (Класс 3)	Avg. AUC
Возраст	0,6505	0,6384	0,8073	0,5792	0,6688
Пол	0,541	0,5513	0,5042	0,5042	0,5252
Кол-во дней в больнице	0,5947	0,5834	0,5563	0,6062	0,5852
Этиология	0,5991	0,5576	0,6375	0,7104	0,6261
Стадия	0,519	0,5889	0,6417	0,7208	0,6176
Предшествующие операции на сердце	0,507	0,5246	0,5417	0,5417	0,5287
Ишемическая болезнь сердца	0,5113	0,5089	0,5125	0,5125	0,5113
Артериальная гипертензия	0,5313	0,5113	0,5042	0,6333	0,545
Хроническая обструктивная болезнь легких	0,5176	0,5109	0,5917	0,5375	0,5394
Патология почек	0,5679	0,5533	0,7042	0,5542	0,5949
Рост	0,5596	0,5521	0,6687	0,5646	0,5863
Вес	0,3756	0,476	0,6229	0,6917	0,5415
Тип реконструкции дуги	0,5651	0,5315	0,7156	0,6229	0,6088
Операция Борста	0,507	0,5004	0,5875	0,5417	0,5341
Вид перфузии головного мозга	0,6222	0,5923	0,6927	0,6625	0,6424
Вмешательство на корне аорты	0,6059	0,5776	0,5583	0,7375	0,6198
Искусственное кровообращение	0,6121	0,5915	0,5667	0,6729	0,6108
Окклюзия аорты	0,5228	0,5121	0,551	0,675	0,5652
Циркуляторный арест	0,5744	0,5259	0,699	0,6354	0,6087

Рисунок 6.4 – Таблица со значимыми признаками

По результатам ROC – анализа видно, что на значение показателя НМК в большей части влияют такие признаки как: возраст, этиология, стадия, тип реконструкции дуги, вид перфузии головного мозга, вмешательство на корне аорты, искусственное кровообращение, время циркуляторного ареста.

Затем на основе выделенных значимых признаков была решена задача классификации медицинских данных с использованием как одиночных, так и ансамблевых алгоритмов. В качестве базовых классификаторов использовались деревья решений (*CART*, *C4.5*, *ID3*, *CHAID*), логистическая регрессия, алгоритм k – взвешенных ближайших соседей и нейронная сеть (многослойный персепtron) с логистической активационной функцией нейронов. В качестве алгоритма обучения нейронной сети использовался алгоритм обратного распространения ошибки.

В качестве ансамблевых классификаторов использовались такие алгоритмы, как неоднородный ансамблевый алгоритм и его модификация, алгоритм *AdaBoost*, *RandomForests* и *Stacking*, построенные с помощью модуля *Data Miner*.

В таблице 6.4 приведены результаты классификации с использованием одиночных и ансамблевых алгоритмов. Для оценки точности классификаторов на тестовой выборке была использована 10×10 блочная кросс – проверка.

Таблица 6.4 – Результаты классификации медицинских данных

№	Алгоритм	Входные параметры	На обучающей выборке		На тестовой выборке		
			Точность классификатора, %	AUC для всех классов	Точность классификатора, %	Дисперсия ошибки классификатора	95% доверительный интервал ошибки классификатора
1	2	3	4	5	6	7	8
1	Алгоритм <i>ID3</i>	$N_{\min} = 2$ $D_{\max} = \infty$	81,4516	AUC(0) = 0,8786 AUC(1) = 0,8685 AUC(2) = 0,9885 AUC(3) = 0,9427	65,0806	0,0138	[0,3259; 0,3725]
2	Алгоритм <i>C4.5</i>	$N_{\min} = 2$ $D_{\max} = \infty$	79,8397	AUC(0) = 0,8626 AUC(1) = 0,862 AUC(2) = 0,9781 AUC(3) = 0,9427	65,2419	0,0135	[0,3245; 0,3707]
3	Алгоритм <i>CART</i>	$N_{\min} = 2$ $D_{\max} = \infty$	83,871	AUC(0) = 0,7957 AUC(1) = 0,8113 AUC(2) = 0,9094 AUC(3) = 0,7937	63,1452	0,0097	[0,349; 0,3881]
4	Алгоритм <i>CHAID</i>	$N_{\min} = 2$ $D_{\max} = \infty$ $\alpha = 0,05$	75	AUC(0) = 0,7261 AUC(1) = 0,7347 AUC(2) = 0,9667 AUC(3) = 0,9583	68,3871	0,006	[0,3007; 0,3315]

Продолжение таблицы 6.4

1	2	3	4	5	6	7	8
5	Логистическая регрессия	$L = 500$ $OM = Quasinewton$	81,4516	AUC(0) = 0,7822 AUC(1) = 0,7527 AUC(2) = 0,95 AUC(3) = 1	62,6613	0,0099	[0,3536; 0,3931]
6	Нейронная сеть	$\mathcal{E} = 10^{-4}$, $i_{\max} = 10^6$ $\eta = 0,1$, $M = 0,2$ $AF = Logistic$, $N_h = 7$	86,2903	AUC(0) = 0,8735 AUC(1) = 0,8392 AUC(2) = 0,8 AUC(3) = 1	67,5806	0,0082	[0,3063; 0,3421]
7	Алгоритм k – взведенных ближайших соседей	$K = 10$ $Q = 0,6$ $D = Chebyshev$	85,4839	AUC(0) = 0,9957 AUC(1) = 0,9984 AUC(2) = 1 AUC(3) = 1	68,2258	0,0074	[0,3007; 0,3348]
9	Алгоритм <i>Random Forests</i>	$N_{\min} = 2$ $D_{\max} = \infty$ $N_{rand} = 2$, $N_{trees} = 10$	83,871	AUC(0) = 0,9747 AUC(1) = 0,9697 AUC(2) = 1 AUC(3) = 0,9875	70,4839	0,0041	[0,2825; 0,3079]
10	Алгоритм <i>AdaBoost</i>	$T = 10$, $M = 0,5$ $c = \{CART, C4.5, KNN\}$	100	AUC(0) = 1 AUC(1) = 1 AUC(2) = 1 AUC(3) = 1	67,4194	0,0124	[0,3037; 0,3479]
11	Алгоритм Stacking	$c = \{ID3, C45, CHAID, KNN\}$ $Meta_classifier - KNN$	85,4839	AUC(0) = 0,8473 AUC(1) = 0,8921 AUC(2) = 0,875 AUC(3) = 0,75	66,2097	0,015	[0,3136; 0,3622]

Окончание таблицы 6.4

1	2	3	4	5	6	7	8
12	Неоднородный ансамблевый алгоритм	$T = 10, M = 0,5$ $A = Majority_votes$ $S = Random_subsample$ $I = Random_classifier$ $c = \{KNN\}$	81,4516	AUC(0) = 0,993 AUC(1) = 0,9996 AUC(2) = 1 AUC(3) = 1	74,5161	0,002	[0,246; 0,2637]
13	Неоднородный ансамблевый алгоритм	$T = 10, M = 0,5$ $A = Majority_votes$ $S = Random_subsample$ $I = Optimal_classifier$ $c = \{KNN, CHAID\}$	78,2258	AUC(0) = 0,9038 AUC(1) = 0,956 AUC(2) = 0,9896 AUC(3) = 0,9875	74,1129	0,0027	[0,2486; 0,2691]
14	Неоднородный ансамблевый алгоритм	$T = 10, M = 0,5$ $A = Majority_votes$ $S = Random_bootstrap$ $I = Random_classifier$ $c = \{ID3, CHAID\}$	79,0323	AUC(0) = 0,8528 AUC(1) = 0,8846 AUC(2) = 0,9906 AUC(3) = 0,9437	72,9032	0,0016	[0,2631; 0,2788]
15	Модифицированный неоднородный ансамблевый алгоритм	$T = 10, M = 0,5$ $A = Majority_votes$ $S = Random_bootstrap$ $I = Random_classifier$ $c = \{CART, C4.5, CHAID\}$	74,1935	AUC(0) = 0,9291 AUC(1) = 0,9297 AUC(2) = 0,9187 AUC(3) = 0,9406	73,3065	0,0016	[0,259; 0,2748]

6.3 Анализ результатов исследования

Таким образом, в ходе исследования реальных медицинских данных были построены различные модели классификаторов, как одиночных, так и ансамблевых. Для неоднородного ансамблевого алгоритма и модифицированного ансамблевого алгоритма было построено несколько моделей с различными входными параметрами, которые были получены с помощью модуля *Data Miner*.

Результаты проведенного исследования показали, что точность классификации выше при использовании неоднородного ансамблевого алгоритма по сравнению с одиночными классификаторами, а также другими ансамблевыми алгоритмами, такими как *AdaBoost*, *Random Forests* и *Stacking*.

Наилучший по точности результат был получен с использованием неоднородного ансамблевого алгоритма №12 со следующими входными параметрами:

- $T = 10$;
- $M = 0,5$;
- $A = \text{Метод большинства голосов}$;
- $S = \text{Случайные подвыборки}$;
- $I = \text{Случайный классификатор}$;
- $c = \{KNN\}$.

Точность классификации с помощью этого алгоритма на обучающей выборке составила 81,4516%, на тестовой – 74,5161%, дисперсия ошибки классификации на тестовой выборке составила 0,002, 95% доверительный интервал ошибки классификатора - [0,246; 0,2637]. Лучший результат по точности классификации на тестовой выборке с помощью одиночного алгоритма соответствует дереву решений *CHAID* – 68,3871%, что ниже точности неоднородного ансамблевого алгоритма, более чем на 6%. Наилучший по точности результат с использованием стандартных ансамблевых алгоритмов соответствует классификатору *Random Forests* – 70,4839%, что ниже точности неоднородного ансамблевого алгоритма, более чем на 4%. На рисунке 6.5 приведены результаты классификации на обучающей выборке с помощью неоднородного ансамблевого алгоритма №12.

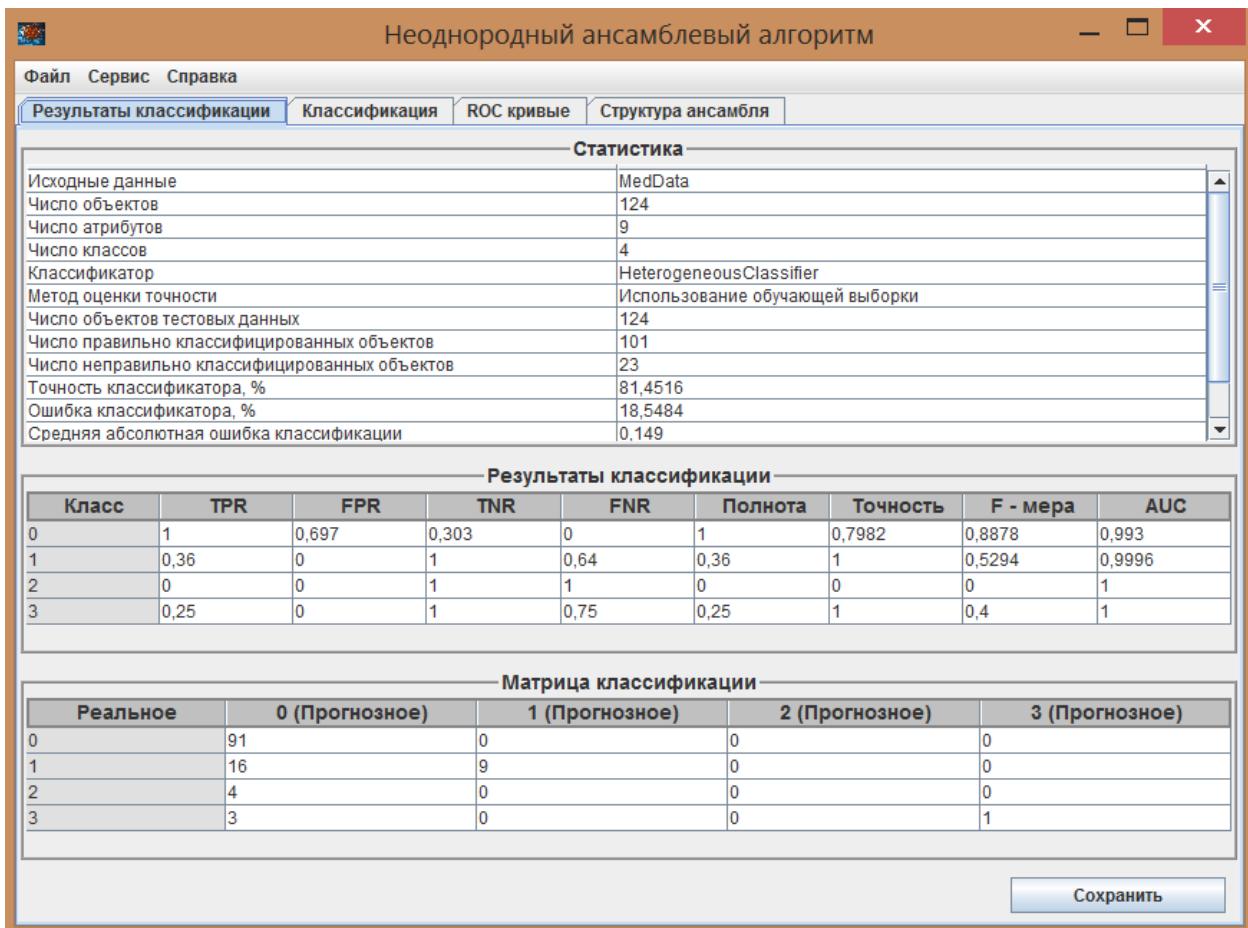


Рисунок 6.5 – Результаты классификации с помощью неоднородного ансамблевого алгоритма №12

Отметим, что модифицированный ансамблевый алгоритм также показал неплохие результаты. Наилучший по точности результат соответствует классификатору №15 со следующими входными параметрами:

- $T = 10$;
- $M = 0,5$;
- $A = \text{Метод большинства голосов}$;
- $S = \text{Бутстрэп выборки случайного размера}$;
- $I = \text{Случайный классификатор}$;
- $c = \{\text{CART, C4.5, CHAID}\}$.

Точность классификации с помощью этого алгоритма на обучающей выборке составила 74,1935%, на тестовой – 73,3065%, дисперсия ошибки классификации на тестовой выборке составила 0,0016, 95% доверительный интервал ошибки классификатора - [0,259; 0,2748].

Для наилучших конфигураций неоднородного ансамблевого алгоритма и модифицированного ансамблевого алгоритма приведем графики *ROC* – кривых для всех классов, построенных с помощью программной системы *ECA* (рисунки 6.6 – 6.7)

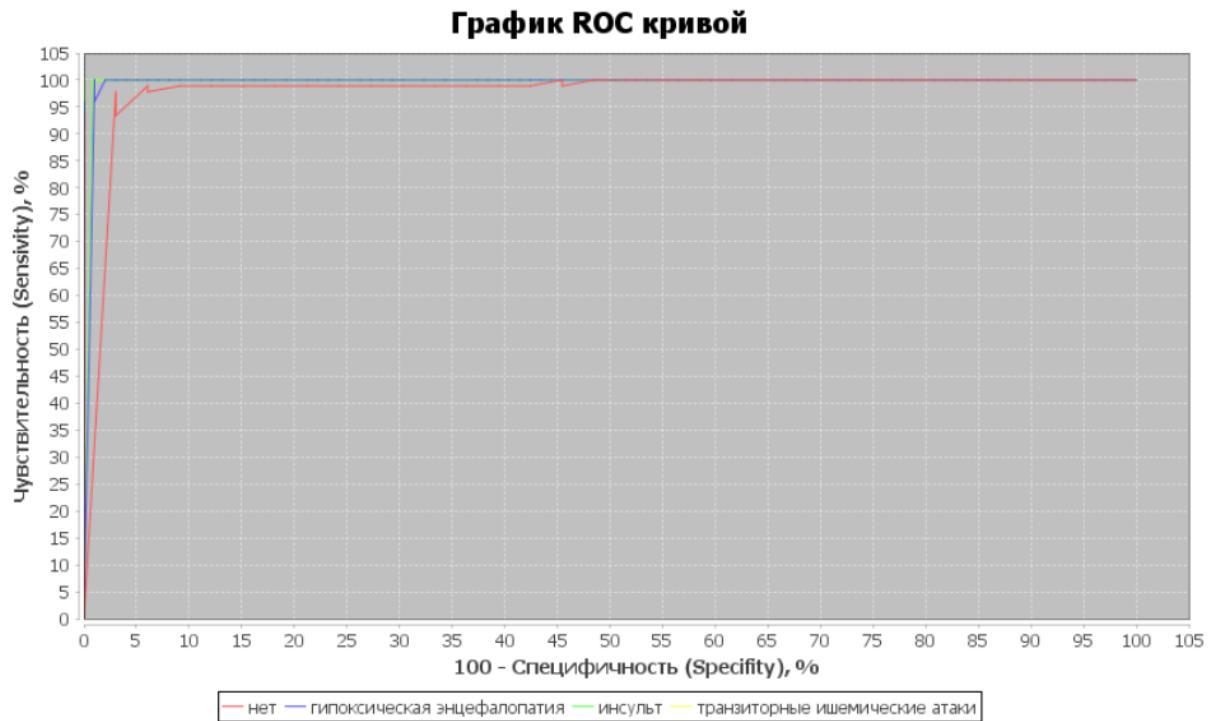


Рисунок 6.6 – Графики *ROC* – кривых для неоднородного ансамблевого алгоритма на обучающей выборке

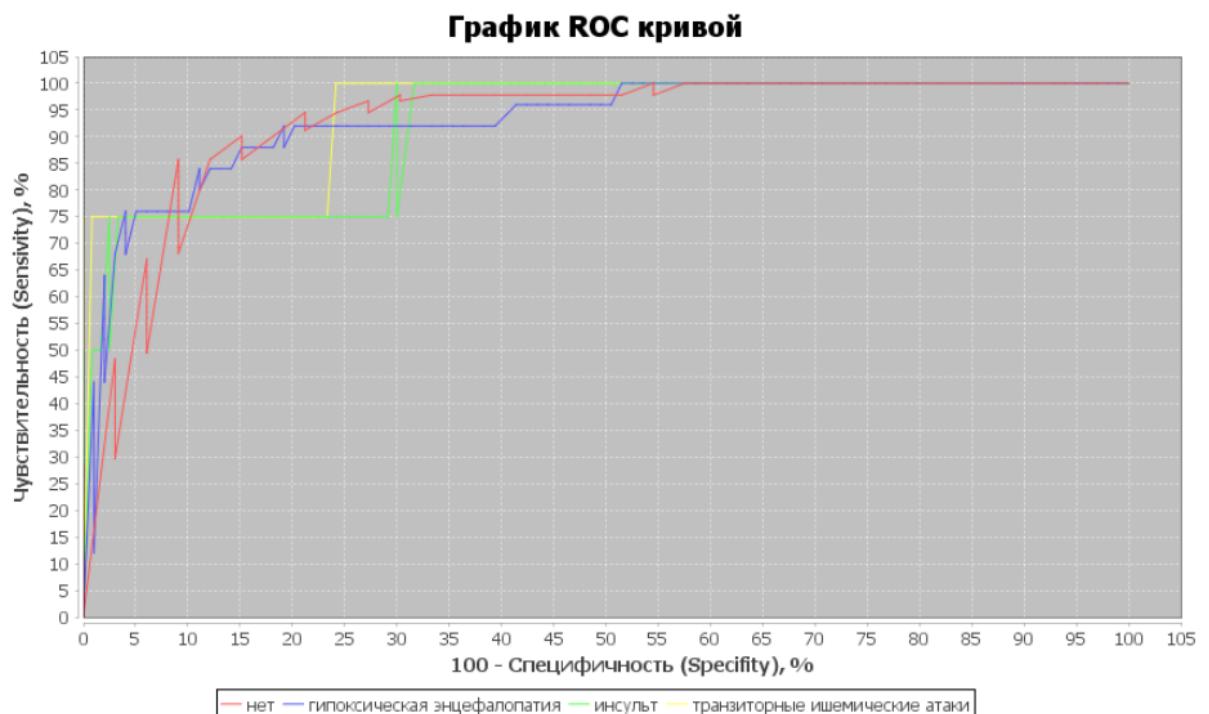


Рисунок 6.7 – Графики *ROC* – кривых для модифицированного неоднородного ансамблевого алгоритма на обучающей выборке

По результатам *ROC* – анализа видно, что показатели *AUC* на обучающей выборке для каждого класса находятся в диапазоне между 0,9 и 1, что говорит об отличном качестве построенных моделей классификаторов с помощью неоднородного и модифицированного неоднородного ансамблевых алгоритмов.

Таким образом, использование неоднородного ансамблевого алгоритма и его модификации позволяет повысить точность классификационных решений, а также уменьшить дисперсию ошибки классификации.

6.4 Выводы по главе

В данной главе диссертационной работы проводилось исследование оригинальных разработанных ансамблевых алгоритмов классификации на тестовых данных и реальных медицинских данных о хирургическом лечении больных с патологией аорты. Основные результаты проведенной работы заключались в следующем:

1. Получены результаты тестирования разработанных алгоритмов на выборке данных *ionosphere*, а также их сравнение с одиночными алгоритмами и стандартными ансамблевыми алгоритмами. Наилучший результат был получен с помощью модифицированного неоднородного ансамблевого алгоритма.
2. Получены результаты оценки точности разработанных алгоритмов на реальных медицинских данных на основе 8 – ми выделенных значимых признаков. Разработанные ансамблевые алгоритмы также показали свою эффективность.
3. Была проведена содержательная интерпретация полученных результатов.

По результатам проведенного исследования можно сделать вывод о том, что использование оригинальных ансамблевых алгоритмов приводит к повышению точности и устойчивости классификационных решений по сравнению с одиночными алгоритмами и стандартными ансамблевыми алгоритмами. Таким образом, разработанные ансамблевые алгоритмы имеют как теоретическую, так и практическую значимость.

Заключение

Данная работа посвящена разработке алгоритмического и программного обеспечения для решения задачи классификации разнотипных данных с использованием ансамбля алгоритмов. В ходе магистерского исследования были получены следующие основные результаты:

1. Был проведен аналитический обзор методов и средств для решения задачи классификации разнотипных данных. В ходе обзора были выявлены актуальные проблемы исследуемой области, а именно повышение точности (устойчивости) классификационных решений. Решением этой проблемы является использование ансамблевых алгоритмов. Также были рассмотрены основные одиночные алгоритмы классификации: деревья решений, логистическая регрессия, нейронные сети и алгоритм k – ближайших соседей и несколько существующих ансамблевых алгоритмов, таких как *Random Forests*, *AdaBoost*, *Bagging* и *Stacking*. Был проведен обзор основных программных средств, в которых реализованы ансамблевые алгоритмы классификации. Обоснована постановка задачи исследования.
2. Было разработано два оригинальных ансамблевых алгоритма классификации: неоднородный ансамблевый алгоритм и модифицированный неоднородный ансамблевый алгоритм, основная идея которых заключается в итерационном применении одиночных классификаторов на обучающей выборке и учете в итоговом решении при построении ансамбля вклада только тех классификаторов, ошибка классификации которых не превосходит заданный порог.
3. Спроектирована и реализована программная система *ECA*, которая включает реализацию разработанных ансамблевых алгоритмов классификации. Также, были рассмотрены основные функциональные возможности этой системы и детали ее реализации в целом. К основным достоинствам разработанной программной системы следует отнести:
 - возможность загрузки исходных данных из различных источников;
 - возможность одновременной работы с несколькими листами данных;
 - возможность выбора любой комбинации базовых классификаторов с настройкой их параметров, которые впоследствии будут использованы при построении ансамбля;
 - возможность просмотра структуры ансамбля, а также результатов классификации составляющих его моделей;
 - удобное представление результатов классификации в табличном виде;

- возможность классификации нового примера на основе построенной модели классификатора;
 - возможность сохранения и загрузки моделей классификаторов из файла.
4. Разработан большой модуль *Data Miner*, идея которого основана на проведении серии экспериментов над классификаторами с различными входными параметрами, и последующем выборе на основе истории проведенных экспериментов оптимальных параметров для классификаторов. В рамках этого модуля был разработан оригинальный алгоритм для автоматического подбора оптимальных параметров для неоднородного ансамблевого алгоритма и его модификации.
 5. Было проведено исследование реальных медицинских данных о хирургическом лечении больных с патологией аорты из Сибирского федерального биомедицинского исследовательского центра имени академика Е.Н. Мешалкина средствами программной системы *ECA*. Наилучшие результаты классификации были получены с использованием разработанного неоднородного ансамблевого алгоритма. Полученные результаты подтвердили эффективность использования разработанных ансамблевых алгоритмов.

Таким образом, разработанная программа система *ECA* может быть использована для решения задачи классификации разнотипных данных в любых предметных областях. Также, система является легко расширяемой и имеет перспективы для дальнейшего развития, а именно добавление новых алгоритмов классификации, расширение модуля *Data Miner*, загрузку исходных данных из других источников и другие.

Список литературы

1. Барсегян А.А., Куприянов М.С., Степаненко В.В., Холод И.И. Технологии анализа данных. Data Mining, Visual Mining, Text Mining, OLAP. – Изд-во: БХВ-Петербург, 2007. – 384 с.
2. Прикладная статистика: Классификация и снижение размерности: Справ. изд. / С. А.Айвазян, В. М. Бухштабер, И. С. Енюков, Л. Д. Мешалкин; Под ред. С. А. Айвазяна.— М.: Финансы и статистика, 1989.— 607 с.
3. Multiple Classifier Systems / J. Kittler & F. Roli (editors) // Proc. of 2nd International Workshop, MCS2001,(Cambridge, UK, 2-4 July 2001) / Lecture Notes in Computer Science. V. 2096. Springer-Verlag, Berlin.
4. Vishwath P. Fusion of multiple approximate nearest neighbor classifier for fast and efficient classification/ P. Vishwath, M.N. Murty, C. Bhatnagar// Information fusion. 2004. V. 5. pp. 239-250.
5. Quinlan J.R. Bagging, boosting and C4.5 / J.R. Quinlan // Proceedings of AAAI/IAAI. 1996. V. 1. pp. 725-730.
6. Breiman L. Bagging predictors // Machine Learning.1996.V. 24, No.2. pp. 123–140.
7. Tumer K. Decimated input ensembles for improved generalization / K. Tumer, N.C. Oza // Proceedings of the International Joint Conference on Neural Networks. – Washington, DC. 1999.
8. Чистяков С. П.Случайные Леса: Обзор // Труды Карельского научного центра РАН. 2013. №1. С. 117 – 136.
9. Новоселова Н. А., Том И. Э. Подход к построению ансамбля классификаторов с использованием генетического алгоритма //Объединенный институт проблем информатики Национальной академии наук г. Минск. 2009. с. 81 – 88.
10. Логистическая регрессия и ROC-анализ – математический аппарат [Электронный ресурс] // <https://basegroup.ru>: Н. Паклин/BaseGroup Labs – Технология анализа данных. – 2014. – URL: <http://www.basegroup.ru/library/analysis/regression/logistic> (дата обращения 27.10.15).
11. Мультиномиальная логистическая регрессия [Электронный ресурс] // <http://statmethods.ru>: Центр статистического анализа. URL: <http://statmethods.ru/konsalting/statistics-metody/137-mnozhestvennogo-vybora-regressionnaya-model.html> (дата обращения 30.10.15).
12. Czepiel S. A. Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation. 2015. pp. 10 – 12.

13. Паклин Н. Б., Орешков В. И. Бизнес аналитика: от данных к знаниям: Учебное пособие. 2-е изд., испр. - СПб.: Питер, 2013. - 704 с.
14. Деревья классификации [Электронный ресурс] // <http://www.statsoft.ru> : Электронный учебник по статистике StatSoft. URL: <http://www.statsoft.ru/home/textbook/modules/stclatree.html> (дата обращения 26.10.15).
15. CART (алгоритм) [Электронный ресурс] // <https://ru.wikipedia.org>: Википедия. URL: [https://ru.wikipedia.org/wiki/CART_\(алгоритм\)](https://ru.wikipedia.org/wiki/CART_(алгоритм)) (дата обращения 24.10.15).
16. Метод ближайших соседей [Электронный ресурс] // <http://www.machinelearning.ru>: MachineLearning.ru. URL: http://www.machinelearning.ru/wiki/index.php?title=Метод_ближайшего_соседа (дата обращения 10.05.17).
17. Алгоритм ближайшего соседа [Электронный ресурс] // <https://basegroup.ru> : Base Group Labs Технологии анализа данных. URL: <https://basegroup.ru/community/articles/knn> (дата обращения 18.05.2017).
18. Widrow B., Lehr M.A. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation // Proceedings of the IEEE, vol. 78, No. 9, September, 1990, p. 1415-1442.
19. Dietterich, T. G. **Ensemble Methods in Machine Learning** // *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*. 2000.s pp. 1-15.
20. Eric Bauer, Ron Kohavi, “An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants”, 1998
21. Yoav Freund, Robert E. Schapire. A decision-theoretic generalization of online learning and an application to boosting // Second European Conference on Computational Learning Theory. – 1995.
22. Breiman, Leo. Stacked regressions // Machine learning 24.1, 1996, pp. 49-64.
23. WEKA [Электронный ресурс] // <http://www.machinelearning.ru>: MachineLearning.ru. URL: <http://www.machinelearning.ru/wiki/index.php?title=WEKA> (дата обращения 14.05.17).
24. Statistica [Электронный ресурс] // <http://ru.wikipedia.org>: Википедия. URL: <http://ru.wikipedia.org/?oldid=68158537> (дата обращения 14.05.17).
25. Полигон алгоритмов [Электронный ресурс] // <http://www.machinelearning.ru>: MachineLearning.ru. URL: http://www.machinelearning.ru/wiki/index.php?title=Полигон_алгоритмов_классификации (дата обращения 25.12.16).

26. Next lexicographical permutation algorithm [Электронный ресурс] // <https://www.nayuki.io>: Project Nayuki. URL: <https://www.nayuki.io/page/next-lexicographical-permutation-algorithm> (дата обращения 16.05.2017).
27. Ionosphere Data Set [Электронный ресурс] // <http://archive.ics.uci.edu> : Machine Learning Repository. URL: <http://archive.ics.uci.edu/ml/datasets/Ionosphere> (дата обращения 22.05.2017).
28. * **Batygin R. I.** Data classification based on the ensemble algorithms / R. I. Batygin ; research adviser O. K. Alsova ; language adviser N. N. Shergina //Progress through Innovations : тез. науч.-практ. конф. аспирантов и магистрантов, Новосибирск, 31 марта 2016 г. – Новосибирск : Изд-во НГТУ, 2016. – С. 13–14. - 160 copy. - ISBN 978-5-7782-2869-6.
29. * **Батыгин Р. И.** Программная система классификации разнотипных медицинских данных на основе ансамбля алгоритмов / Р. И. Батыгин //Электротехнические комплексы и системы: тез. 54-й международной научной студенческой конференции, 16-20 апреля 2016 г. - Новосибирск :Изд-во НГТУ, 2016. – С. 66. - 155 copy. - ISBN 978-5-4437-0514-9.
30. * **Batygin R. I.** Software system for different types of data classification based on the ensemble algorithms / R. I. Batygin, O. K. Alsova // Актуальные проблемы электронного приборостроения (АПЭП–2016) = Actual problems of electronic instrument engineering (APEIE–2016) : тр. 13 междунар. науч.-техн. конф., Новосибирск, 3–6 окт. 2016 г. : в 12 т. – Новосибирск : Изд-во НГТУ, 2016. – Т. 1, ч. 2. – С. 506-509. - 60 экз. - ISBN 978-5-7782-2991-4.
31. * **Батыгин Р. И.** Программная система классификации разнотипных данных на основе ансамбля алгоритмов = Software system for different types of data classification based on the ensemble algorithms / Р. И. Батыгин, О. К. Альсова // Актуальные проблемы электронного приборостроения (АПЭП–2016) =Actual problems of electronic instrument engineering (APEIE–2016) : тр. 13 междунар. науч.-техн. конф., Новосибирск, 3–6 окт. 2016 г. : в 12 т. – Новосибирск : Изд-во НГТУ, 2016. – Т. 9.- С.117-121. - 40 экз. - ISBN 978-5-7782-2991-4.
32. * **Батыгин Р.И., Альсова О.К.** Программная система классификации разнотипных данных на основе ансамбля алгоритмов (ECA - Ensemble Classification Algorithms): Свидетельство о государственной регистрации программы для ЭВМ № 2017610788. 2017.