**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ГРОДНЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ ЯНКИ КУПАЛЫ**

*На правах рукописи*

УДК 004.8

Басинский Вадим Михайлович

**Разработка и анализ качества модификаций метаэвристических алгоритмов для задачи классификации**

Магистерская диссертация на соискание степени

магистра прикладной математики и информационных технологий

по специальности

1-31 81 12 Прикладной компьютерный анализ данных

Научный руководитель

кандидат физико-математических наук,

Цехан О. Б.

Допущена к защите

Заведующий кафедрой математического и информационного обеспечения экономических систем

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О. Б. Цехан

Гродно, 2018

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc516526877)

[СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ 5](#_Toc516526878)

[ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ 6](#_Toc516526879)

[ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ КЛАССИФИКАЦИИ И ПРИНЦИП ЕЁ РЕШЕНИЯ МЕТАЭВРИСТИЧЕСКИМИ АЛГОРИТМАМИ 8](#_Toc516526880)

[1.1 Постановка задачи классификации 8](#_Toc516526881)

[1.2 Обзор методов решения задачи классификации 8](#_Toc516526882)

[1.3 Принцип решения задачи классификации при помощи метаэвристических алгоритмов 13](#_Toc516526883)

[1.4 Дискретизация значений атрибутов 16](#_Toc516526884)

[1.5 Обзор метаэвристических алгоритмов 18](#_Toc516526885)

[1.5.1 Муравьиный алгоритм. 18](#_Toc516526886)

[1.5.2 Генетический алгоритм. 21](#_Toc516526887)

[1.6 Оценка качества классификационного правила 23](#_Toc516526888)

[1.7 Выводы по первой главе 24](#_Toc516526889)

[ГЛАВА 2 ПРОГРАММНЫЙ ИНСТРУМЕНТАРИЙ ДЛЯ ОПТИМИЗАЦИИ ПАРАМЕТРОВ МЕТАЭВРИСТИЧЕСКИХ АЛГОРИТМОВ 25](#_Toc516526890)

[2.1 Основные идеи разработанного программного продукта 25](#_Toc516526891)

[2.2 Описание входных и выходных данных 27](#_Toc516526892)

[2.3 Описание оптимизируемых параметров 28](#_Toc516526893)

[2.4 Проектирование системы классов 31](#_Toc516526894)

[2.5 Описание основных методов и процедур 39](#_Toc516526895)

[2.6 Псевдокод алгоритма муравьиной колонии 43](#_Toc516526896)

[2.7 Выводы по второй главе 43](#_Toc516526897)

[ГЛАВА 3 АНАЛИЗ И ОПТИМИЗАЦИЯ МОДИФИКАЦИЙ МЕТАЭВРИСТИЧЕСКИХ АЛГОРИТМОВ 44](#_Toc516526898)

[3.1 Анализ использованных и разработанных модификаций 44](#_Toc516526899)

[3.2 Сравнение результатов работы различных комбинаций модификаций 48](#_Toc516526900)

[3.3 Анализ результатов оптимизации 52](#_Toc516526901)

[3.4 Выводы по третьей главе 58](#_Toc516526902)

[ЗАКЛЮЧЕНИЕ 60](#_Toc516526903)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 61](#_Toc516526904)

[ПРИЛОЖЕНИЯ 64](#_Toc516526905)

# ВВЕДЕНИЕ

Важной составляющей любой модели, используемой в задачах принятия решений, является задача классификации. Типичными примерами современных приложений этой задачи являются задачи интеллектуального управления бизнесом (англ. business intelligence), анализ социальных сетей, прогнозирования спроса и персонификации предложений в Интернет-торговле, в области продвижения веб-сайтов и многие другие [1, 2]. Новый класс задач подобного рода появился в последнее время в связи развитием концепции больших данных (англ. Big Data).

Проблема классификации представляет собой задачу упорядочения объектов по некоторым классификационным признакам и отнесения каждого объекта к какому-то заранее известному классу.

Наиболее часто для решения задачи классификации используются следующие методы:

1. Нейронные сети,
2. Деревья принятия решений,
3. Линейные классификаторы,
4. Байесовский классификатор.

Одним из относительно новых подходов к решению задачи классификации является подход, основанный на идеях роевых алгоритмов или метаэвристических алгоритмах случайного поиска [15, с. 23-27]. Например, принцип использования алгоритма муравьиной колонии описан в работе [28].

Преимуществом использования метаэвристических алгоритмов для решения задачи классификации является то, что, благодаря большому количеству параметров, их можно довольно точно подстроить под требуемую задачу и даже конкретный набор данных. В тоже время, как результаты работы метаэвристических алгоритмов, так и сами параметры алгоритма, очень легко интерпретировать, в отличие от, например, нейронных сетей.

Однако, осуществлять подбор параметров вручную нецелесообразно. Поэтому был реализован метаэвристический алгоритм, решающий задачу классификации, и дополнительно обеспечена возможность подбора параметров в автоматическом режиме, с использованием эволюционных алгоритмов, например, генетического.

Таким образом, *цель диссертационного исследования –* разработать и программно реализовать модификации метаэвристических алгоритмов для повышения качества решения задачи классификации

*Задачи*, которые ставятся перед автором данной работы:

* Провести обзор и анализ самых распространенных и широко используемых методов решения задачи классификации,
* Описать процесс сведения задачи классификации к задаче поиска путей на графе,
* Рассмотреть принцип уменьшения размерности задачи путем проведения дискретизации исходных данных,
* Проанализировать параметры метаэвристических алгоритмов и дать начальные оценки каждому из параметров,
* Спроектировать систему классов приложения,
* Дать формальное описание основных процедур и методов,
* Проанализировать результаты тестирования приложения на выбранных наборах данных для определения преимуществ, недостатков и путей дальнейшего исследования.

*Объект исследования* – метаэвристические алгоритмы в задаче классификации.

*Предмет исследования* – оптимизация параметров алгоритма муравьиной колонии генетическим алгоритмом при решении задачи классификации.

*Актуальность исследования* очевидна: задача классификации возникает в различных областях человеческой деятельности: диагностирование болезней в медицине, предсказание месторождений полезных ископаемых, оптическое распознавание символов, распознавание речи, обнаружение спама, классификация документов. Разработка более эффективных методов решения задачи классификации окажет положительный экономический эффект на состояние данных отраслей [13, c.57-60].

Разрабатываемая система представляет собой десктопное приложение. В качестве языка для разработки программного модуля использовался C#, средой разработки была выбрана Visual Studio 2017.

Актуальность работы подтверждается указом Президента Республики Беларусь от 22 апреля 2015 г. № 166 «О приоритетных направлениях научно-технической деятельности в Республики Беларусь на 2016 – 2020 годы» (п. 7. Информационно-коммуникационные и авиакосмические технологии: разработка интегрированных систем автоматизации управления процессами и ресурсами организаций).

Для достижения поставленной цели в процессе написания данной работы использовались следующие методы: оптимизации, сравнительный, графический и другие.

# СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ

# ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

***Связь работы с крупными научными программами (проектами) и темами***

Диссертационное исследование проводилось в рамках темы второй половины дня «Математическое моделирование устойчивого развития социально-экономических объектов, выявление факторов повышения конкурентоспособности объектов изучения» кафедры математического и информационного обеспечения экономических систем.

***Цель и задачи исследования***

*Цель –* разработать и программно реализовать модификации метаэвристических алгоритмов для повышения качества решения задачи классификации.

*Задачи*, которые ставятся перед автором данной работы:

* Провести обзор и анализ самых распространенных и широко используемых методов решения задачи классификации,
* Описать процесс сведения задачи классификации к задаче поиска путей на графе,
* Рассмотреть принцип уменьшения размерности задачи путем проведения дискретизации исходных данных,
* Проанализировать параметры метаэвристических алгоритмов и дать начальные оценки каждому из параметров,
* Спроектировать систему классов приложения,
* Дать формальное описание основных процедур и методов,
* Проанализировать результаты тестирования приложения на выбранных наборах данных для определения преимуществ, недостатков и путей дальнейшего исследования.

*Объект исследования* – метаэвристические алгоритмы в задаче классификации.

*Предмет исследования* – оптимизация параметров алгоритма муравьиной колонии генетическим алгоритмом при решении задачи классификации.

***Положения, выносимые на защиту***

1. Обзор и анализ существующих подходов для решения задачи классификации;
2. Генетический алгоритм для подбора параметров метаэвристического алгоритма при решении задачи классификации;
3. Программный продукт, написанный на языке C#, реализующий описанный в работе алгоритм.

***Личный вклад соискателя***

Все результаты, изложенные в работе, получены соискателем. Роль научного руководителя состояла в постановке рассматриваемых в диссертации задач и анализе полученных результатов.

***Апробация результатов диссертации***

1. XII Международная заочная научная конференция студентов, магистрантов, аспирантов НИРС ФЭУ-2018 «ЭКОНОМИКА И УПРАВЛЕНИЕ XXI ВЕКА» ГрГУ Гродно
2. ХII Международная научная конференция студентов и молодых ученых «Наука и образование – 2018». Астана, Евразийский национальный университет им. Л.Н. Гумилева
3. XIХ Республиканская научно-практическая конференция молодых ученых 12 мая 2018 года, Брест, Брестский государственный университет имени А.С. Пушкина
4. XI Международная НПК "Молодежь и научно-технический прогресс", Белгород, Губкинский филиал БГТУ им. В.Г. Шухова
5. Fourth International Conference and Expo BIG DATA ADVANCED ANALYTICS May 3-4, 2018 Minsk, Belarus
6. I Международная научно-практическая конференция «Информационно-коммуникационные технологии: достижения, проблемы, инновации (ИКТ-2018)», Полоцк, Полоцкий государственный университет

***Опубликованность результатов диссертации***

1. XII Международная заочная научная конференция студентов, магистрантов, аспирантов НИРС ФЭУ-2018 «ЭКОНОМИКА И УПРАВЛЕНИЕ XXI ВЕКА» ГрГУ Гродно
2. ХII Международная научная конференция студентов и молодых ученых «Наука и образование – 2018». Астана, Евразийский национальный университет им. Л.Н. Гумилева
3. XI Международная НПК "Молодежь и научно-технический прогресс", Белгород, Губкинский филиал БГТУ им. В.Г. Шухова
4. Fourth International Conference and Expo BIG DATA ADVANCED ANALYTICS May 3-4, 2018 Minsk, Belarus

***Структура и объем диссертации.***

Работа состоит из введения, трех глав, заключения, списка использованных источников и приложений.

45+ страниц, 30+ формул, 15+ рисунков, 10+ таблиц.

# ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ КЛАССИФИКАЦИИ И ПРИНЦИП ЕЁ РЕШЕНИЯ МЕТАЭВРИСТИЧЕСКИМИ АЛГОРИТМАМИ

* 1. **Постановка задачи классификации**

Формальная постановка задачи классификации может быть представлена следующим образом. Задано множество объектов , множество допустимых ответов , и существует целевая функция (target function) , значения которой = известны только на конечном подмножестве объектов . Пары «объект-ответ» (,) называются прецедентами. Совокупность пар = называется обучающей выборкой (training sample).

*Задача обучения по прецедентам* заключается в том, чтобы по выборке восстановить зависимость , то есть построить решающую функцию (decision function) , которая приближала бы целевую функцию , причём не только на объектах обучающей выборки, но и на всём множестве [7].

Под решающей функцией в данной работе понимается классификатор, основанный на решающих классификационных правилах

Каждый объект описывается в виде набора выражений , где – атрибут, а – одно из значений домена этого атрибута. Обязательным атрибутом каждой записи обучающей выборки является *class* – ***метка класса***, т.е. один из элементов множества , по которому определяется к какому классу объект относится.

* 1. **Обзор методов решения задачи классификации**

Рассмотрим наиболее часто используемые методы, применяемые при решении задачи классификации [12, c. 25]. Среди них можно выделить следующие:

* Дерево принятия решений,
* Байесовский классификатор,
* Метод опорных векторов,
* Нейронные сети,
* Метаэвристические алгоритмы.

Рассмотрим преимущества и недостатки каждого из них.

***Деревья принятия решений***

Деревья решений – это способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение.

В процессе построения модели, алгоритм, вычисляет степень влияния каждого входного атрибута на значение выходного атрибута и использует атрибут, влияющий на выходной атрибут в наибольшей степени, для разбиения узла дерева решений [4].

Такие образом, для построения дерева на каждом внутреннем узле необходимо найти такое условие (проверку), которое бы разбивало множество, ассоциированное с этим узлом на подмножества. В качестве такой проверки должен быть выбран один из атрибутов. Общее правило для выбора атрибута можно сформулировать следующим образом: выбранный атрибут должен разбить множество так, чтобы получаемые в итоге подмножества состояли из объектов, принадлежащих к одному классу, или были максимально приближены к этому, т.е. количество объектов из других классов ("примесей") в каждом из этих множеств было как можно меньше [3].

Процесс повторяется на каждом из подмножеств до того момента, пока не будет выполнено одно из условий остановки:

* Полученное множество содержит элементы только одного класса,
* Глубина дерева достигла максимально допустимой величины,
* Новое разбиение получается на нетривиальные множества – множества, содержащие достаточно большое количество объектов.

Основные методы построения деревьев принятия решений:

1. **ID3.** В основе этого алгоритма лежит понятие информационной энтропии – то есть, меры неопределенности информации (обратной мере информационной полезности величины). Для того чтобы определить следующий атрибут, необходимо подсчитать энтропию всех неиспользованных признаков относительно тестовых образцов и выбрать тот, для которого энтропия минимальна. Этот атрибут и будет считаться наиболее целесообразным признаком классификации [30, c. 81-85].
2. **C4.5.** Этот алгоритм – усовершенствование предыдущего метода, позволяющее, в частности, «усекать» ветви дерева, если оно слишком сильно «разрастается», а также работать не только с атрибутами-категориями, но и с числовыми. В общем-то, сам алгоритм выполняется по тому же принципу, что и его предшественник; отличие состоит в возможности разбиения области значений независимой числовой переменной на несколько интервалов, каждый из которых будет являться атрибутом. В соответствии с этим исходное множество делится на подмножества. В конечном итоге, если дерево получается слишком большим, возможна обратная группировка – нескольких узлов в один лист. При этом, поскольку перед построением дерева ошибка классификации уже учтена, она не увеличивается [29, c. 187-191].
3. **CART.** Алгоритм разработан в целях построения так называемых бинарных деревьев решений – то есть тех деревьев, каждый узел которых при разбиении «дает» только двух потомков. Грубо говоря, алгоритм действует путем разделения на каждом шаге множества примеров ровно напополам – по одной ветви идут те примеры, в которых правило выполняется (правый потомок), по другой – те, в которых правило не выполняется (левый потомок). Таким образом, в процессе «роста» на каждом узле дерева алгоритм проводит перебор всех атрибутов, и выбирает для следующего разбиения тот, который максимизирует значение показателя, вычисляемого по математической формуле и зависящего от отношений числа примеров в правом и левом потомке к общему числу примеров.

***Достоинства:*** в общем случае обеспечивает высокую скорость работы. Может применяться как для задач классификации, так и для задач кластеризации.

***Недостатки:*** сложная структура. Для построения дерева решений или для внесения изменений, требуется помощь специалиста в рассматриваемой области.

***Байесовский классификатор***

Для каждого наблюдения из множества высчитывается вероятность его возникновения в соответствующем классе по формуле (1.1).

, (1.1)

где – количество элементов в категории ;

– количество наблюдений из множества , принадлежащих категории .

При вычислении вероятности требуется так же априорная вероятность возникновения категории . равна отношению числа наблюдений в . к общему числу наблюдений по формуле (1.2).

. (1.2)

Классификация набора наблюдений происходит после вычисления вероятностей по формуле (1.3).

. (1.3)

***Достоинства:*** высокая скорость работы алгоритма как на этапе обучения так и на этапе анализа новых данных.

***Недостатки:*** в случае невозможно классифицировать объект [18, c. 867].

***SVM (метод опорных векторов)***

Идею метода удобно проиллюстрировать на следующем простом примере: даны точки на плоскости, разбитые на два класса.

Однако, в пространствах высоких размерностей прямая уже не будет разделять наши классы, так как понятие «ниже прямой» или «выше прямой» теряет всякий смысл. Поэтому вместо прямых необходимо рассматривать гиперплоскости — пространства, размерность которых на единицу меньше, чем размерность исходного пространства.

С точки зрения точности классификации лучше всего выбрать прямую, расстояние от которой до каждого класса максимально. Другими словами, выберем ту прямую, которая разделяет классы наилучшим образом. Такая прямая, а в общем случае — гиперплоскость, называется оптимальной разделяющей гиперплоскостью [35].

Вектора, лежащие ближе всех к разделяющей гиперплоскости, называются *опорными векторами* (support vectors).

Метод опорных векторов строит классифицирующую функцию *F* в следующем виде в формуле (1.4):

, (1.4)

где — скалярное произведение;

*w* — нормальный вектор к разделяющей гиперплоскости;

*b* — вспомогательный параметр.

Далее, мы хотим выбрать такие и которые максимизируют расстояние до каждого класса. Проблема максимизации этой величины эквивалентна минимизации . Запишем это в виде задачи, которая решается квадратичным программированием в формуле (1.5):

(1.5)

Для более общего случая необходимо учитывать ошибки алгоритма, которые требуется минимизировать. Задача примет следующий вид из формулы (1.6):

(1.6)

Переменные характеризуют величину ошибки на примере выборки мощностью . Константа позволяет регулировать величину зазора и суммарную ошибку на тренировочной выборке.

Как и любой другой метод, метод SVM имеет свои сильные и слабые стороны, которые следует учитывать при выборе данного метода.

***Достоинства:*** для классификации методом *опорных векторов*, в отличие от большинства других методов, достаточно небольшого набора данных. При правильной работе модели, построенной на тестовом множестве, вполне возможно применение данного метода на реальных данных.

***Недостатки:*** для классификации используется не все множество образцов, а лишь их небольшая часть, которая находится на границах [33].

***Нейронные сети***

Общий подход к построению классификатора на основе нейронных сетей представлен далее.

**Работа с данными**

* 1. Составить базу данных из примеров, характерных для данной задачи
  2. Разбить всю совокупность данных на два множества: обучающее и тестовое (возможно разбиение на 3 множества: обучающее, тестовое и подтверждающее).

**Предварительная обработка**

1. Выбрать систему признаков, характерных для данной задачи, и преобразовать данные соответствующим образом для подачи на вход сети (нормировка, стандартизация и т.д.). В результате желательно получить линейно отделяемое пространство множества образцов [5].
2. Выбрать систему кодирования выходных значений (классическое кодирование, 2 на 2 кодирование и т.д.)

**Конструирование, обучение и оценка качества сети**

1. Выбрать топологию сети: количество слоев, число нейронов в слоях и т.д.
2. Выбрать функцию активации нейронов
3. Выбрать алгоритм обучения сети
4. Оценить качество работы сети на основе подтверждающего множества или другому критерию, оптимизировать архитектуру (уменьшение весов, прореживание пространства признаков)
5. Остановится на варианте сети, который обеспечивает наилучшую способность к обобщению и оценить качество работы по тестовому множеству.

**Использование и диагностика**

1. Выяснить степень влияния различных факторов на принимаемое решение (эвристический подход).
2. Убедиться, что сеть дает требуемую точность классификации (число неправильно распознанных примеров мало)
3. При необходимости вернуться на этап 2, изменив способ представления образцов или изменив базу данных.
4. Практически использовать сеть для решения задачи [9].

***Выводы.*** Невозможно однозначно определить лучший метод построения классификатора, так как каждый из них имеет ряд недостатков и преимуществ, а определение наиболее подходящего метода для конкретной задачи – сложный процесс, требующий глубоких знаний [32].

Для того, чтобы построить качественный классификатор, необходимо иметь качественные данные. Никакой из методов построения классификаторов, основанный на нейронных сетях или статистический, никогда не даст классификатор нужного качества, если имеющийся набор примеров не будет достаточно полным и представительным для той задачи, с которой придется работать системе [6].

В данной работе более подробно рассмотрен последний подход – метаэвристические алгоритмы. Преимуществом этого подхода является возможность простой настройки и подбора оптимальных параметров алгоритма, возможность комбинирования остальных подходов, например, представление правил в виде, который легко воспринимается – как в ДПР, и возможность тонко настраивать принцип поиска и построения правил во время работы алгоритма на основании ранее полученных результатов – как в нейронных сетях. Описание подхода к решению задачи классификации, основанного на метаэвристических алгоритмах, дано в следующем разделе.

* 1. **Принцип решения задачи классификации при помощи метаэвристических алгоритмов**

Как говорилось ранее, при решении задачи классификации имеется два множества: – множество описаний объектов, – множество наименований классов. Каждая запись тестовой выборки может быть представлена в следующем виде, как показывает формула (1.7).

(1.7)

С другой стороны, можно считать, что имеется набор атрибутов , каждый из которых имеет заранее известное конечное множество значений .

Тогда, выражение можно расписать следующим образом, как показывается формулой (1.8).

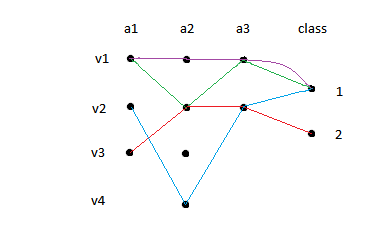
(1.8)

где – -ый атрибут во множестве ;

– -ое значение атрибута .

Таким образом можно видеть, что задачу классификации можно представить в виде графа следующим образом:

* Часть вершин графа соответствует набору выражений – эти вершины позволяют описать объекты выборки.
* Часть вершин графа соответствует элементам множества – эти вершины позволяют описать допустимые классы.
* Каждая вершина связана со всеми остальными, кроме вершин которые соответствуют другим значениям этого же атрибута.
* Граф является неориентированным – переход можно осуществить между любой допустимой парой вершин.

****Удобно представлять граф так, как показано на рисунке 1.1.

**Рисунок 1.1 – Представление задачи классификации в виде графа**

**Источник: собственная разработка автора**

Каждая допустимая запись из тренировочной выборки представляет собой один путь на таком графе. На рисунке 1.1 показано четыре пути, которые соответствуют следующим четырем записям:

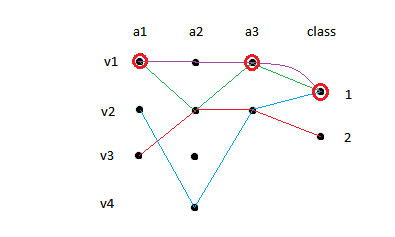
1. && ,
2. && ,
3. && ,
4. && .

Результатом работы классификатора будет являться набор *классификационных правил*, извлеченных из тренировочной выборки.

*Классификационным правилом* будем называть такой путь в графе, который позволит компактно описать целый набор путей из обучающей выборки и выделить для них общие вершины.

Примером классификационного правила для описанного случая может являться выражение в формуле (1.9).

(1.9)

Графически это проиллюстрировано на рисунке 1.2.

**Рисунок 1.2 – Пример классификационного правила**

**Источник: собственная разработка автора**

В этом случае говорим, что классификационное правило ***покрывает*** пути 1 и 2, которые соответствуют первым двум объектам тренировочной выборки, или, с точки зрения задачи классификации, может использоваться для классификации этих объектов.

* 1. **Дискретизация значений атрибутов**

Большое влияние на сложность решения задачи классификации оказывает структура входных данных. Если атрибут имеет большое количество значений, то растет число вершин в графе для соответствующего атрибута, а среднее число путей через каждую из них уменьшается. Это значительно ухудшает качество классификации и точность извлекаемого набора правил. Для решения этой проблемы используется дискретизация атрибутов. Рассмотрим некоторые способы осуществления данной процедуры.

Согласно первому методу, для дискретизации значений атрибута можно сделать следующее:

1. Выделить требуемый атрибут и атрибут, описывающий класс,
2. Отсортировать пары «значение атрибута» – «класс»,
3. Выбрать в качестве границ разбиения атрибута среднее значений, на которых класс изменяется.

Проблема этого подхода заключается в необходимости работы со множеством допустимых классов. Существуют методы, которые могут работать без этого, например, дискретизация равными интервалами и равными частотами.

Для дискретизации *методом равных интервалов* среди значений атрибута нужно выделить максимальное и минимальное значения и определить количество требуемых категорий на выходе. После этого следующим образом определяется длина интервала, как показано в формуле (1.10):

. (1.10)

Все значения атрибута, лежащие в одном интервале относятся в одну категорию.

При дискретизации *методом равных частот* необходимо задать желаемое количество категорий атрибута и разбить на групп, каждая из которых будет содержать как минимум объектов.

Для проведения дискретизации можно использовать *метод разбиения на подмножества с минимальной энтропией*. Метод использует информационную энтропию возможных разбиений для определения границы разбиения множества на два подмножества.

Пусть имеется набор записей, атрибут и граница разбиения . Тогда энтропия разбиения может быть по формуле (1.11) определена следующим образом:

.(1.11)

Разбиение продолжается на наборах и пока не будет выполняться условие остановки, описанное формулой (1.12):

. (1.12)

Величина определяется как разность начального и полученного разбиений, а величина определяется как показано в формуле (1.13):

. (1.13)

Ввод этой величины позволяет избежать разбиения множеств на слишком маленькие по объему группы. Описанные методы дискретизации рассмотрены более подробно в [19, с. 49-51].

* 1. **Обзор метаэвристических алгоритмов**
     1. **Муравьиный алгоритм.**

Общая схема алгоритма, описанного в [17, c.156], может быть представлена следующим образом.

Случайным образом выбирается вершина, в которую помещается муравей. На каждом этапе построения пути он выбирает лучшую вершину из ещё не посещенных и переходит туда, добавляя её к правилу. Выбор осуществляется на основании уровня феромонов вершин и значениях эвристической функции. Процесс построения пути осуществляется, пока выполняются следующие условия: 1) есть вершины, атрибут которых не входит в текущий путь; 2) текущее правило соответствует достаточно большому набору путей обучающей выборки.

На втором этапе производится усечение(pruning) правила. Это необходимый этап, так как по ходу построения возможно включение несущественных вершин, которые делают соответствующее правило слишком детализированным и ведут к возникновению переобучения(overfitting). Происходит это из-за жадного выбора следующего шага при локальном поиске вершин.

После получения муравьем окончательного правила необходимо провести пересчет весов вершин – обновление феромонов, для симуляции испарения на непосещённых вершинах и увеличения количества феромона на пройденных.

Эти три этапа повторяются до выполнения одного из следующих требований: 1) количество итераций цикла превысило количество муравьев колонии; 2) последние муравьев построили одинаковый путь – популяция сошлась на лучшем из найденных путей.

Количество феромона перед началом итерации для - ого значения - ого атрибута(вершина графа) обозначим . Перед началом работы алгоритма все вершины инициализируются одинаковым количеством феромона. Это значение определяется исходя из общего числа атрибутов и их возможных значений. Это количество феромонов может вычисляться следующим образом по формуле (1.14):

, (1.14)

где – количество атрибутов;

– количество значений - ого атрибута.

Вероятность выбора следующей вершины для добавления к правилу определяется следующим образом по формуле (1.15):

, (1.15)

где – значение эвристической функции вершины,

– количество феромонов в этой вершине,

– количество атрибутов,

= 1, если правило уже содержит выражение в которое входит - ый атрибут, и 0 иначе.

– количество значений, которые может принимать - ый атрибут.

– номер значения - го атрибута.

Величины и определяются группой, к которой принадлежит данный муравей, и определяют характер его поведения при построении пути. Более подробно это будет рассмотрено далее.

В качестве эвристической используется функция, описываемая формулой (1.16):

, (1.16)

где – количество атрибутов,

= 1, если правило уже содержит выражение в которое входит - ый атрибут, и 0 иначе.

– количество значений, которые может принимать - ый атрибут.

– номер значения - ого атрибута.

Величины представляют собой условные энтропии, рассчитанные для каждой вершины графа. Определяются они следующим образом, показанным в формуле (1.17):

, (1.17)

где – класс объекта;

– количество классов;

– вероятность наблюдения у объекта класса при значении у атрибута .

Величины рассчитываются только один раз до запуска муравьиного алгоритма и остаются неизменными в процессе работы независимо от того, как изменяется множество объектов.

При добавлении к правилу новой вершины необходимо пересмотреть класс, который оно предсказывает. Для этого просматриваются все записи, которые правило покрывает, и результатом становится тот класс, который встречается у большей части записей.

После построения текущего правила, т.е. некоторого пути, который описывает какую-то часть объектов обучающей выборки, выполняется усечение правила.

Осуществляется эта процедура следующим образом.

Начинается работа с полным правилом, и делается попытка получить из него набор новых правил при помощи удаления одного из выражений. Для каждого правила в полученном наборе пересчитывается набор соответствующих ему записей, результат и качество. После этого из набора берется правило с максимальным значением качества и с ним проводится та же процедура. Это будет происходить до тех пор, пока на следующем этапе не останется допустимых для удаления выражений, или качество нового правила не станет меньше предыдущего. Только после этого правило будет добавлено к списку найденных правил текущей итерации.

После добавления правила к списку необходимо обновить веса всех вершин для запуска следующего муравья. Пересчет весов можно производить различными способами, но здесь выбран подход, основанный на качестве последнего найденного правила. Для всех вершин, которые вошли в правило количество феромонов необходимо увеличить, а для всех остальных – уменьшить.

Изменение количества феромонов осуществляется следующим образом, как показывает формула (1.18):

(1.18)

После включения правила из обучающей выборки исключаются все объекты, покрытые текущим правилом, и далее запускается следующее поколение колонии.

**1.5.2 Генетический алгоритм.**

Генетический алгоритм – это метод решения как ограниченных, так и безусловных задач оптимизации, основанный на естественном отборе, процессе, который стимулирует биологическую эволюцию. Генетический алгоритм неоднократно модифицирует популяцию индивидуальных решений. На каждом этапе генетический алгоритм выбирает решения случайным образом из текущей популяции и использует их для создания особей следующего поколения. В течение последующих поколений население «эволюционирует» к оптимальному решению. Применить генетический алгоритм можно для решения множества проблем оптимизации, которые не подходят для стандартных алгоритмов оптимизации, включая проблемы, в которых целевая функция является разрывной, недифференцируемой, стохастической или сильно нелинейной. Генетический алгоритм может решать проблемы смешанного целочисленного программирования, где некоторые компоненты ограничены целыми значениями.

Генетический алгоритм использует три основных типа правил на каждом шаге для создания следующего поколения от текущей популяции:

* правила отбора,
* правила скрещивания,
* правила мутации.

Различные принципы отбора позволяют варьировать ход эволюции решения и вести поиск оптимального решения по отличным друг от друга стратегиям. Наиболее распространенными стратегиями отбора являются следующие:

* Турнирный отбор,
* Отбор рулеткой,
* Стохастическая универсальная выборка,
* Линейное ранжирование,
* Экспоненциальное ранжирование,
* Отбор усечением.

Подробный обзор с псевдокодом и описанием основных идей каждого метода приведен в работе [31].

В работе использовалась стратегия турнирного отбора с использованием усечений.

Так как любая стратегия отбора предполагает возможность оценки каждой особи, нам необходимо ввести некоторый функционал качества. Под особью в данной работе понимается один классификатор со своими значениями параметров, набором правил и значением точности классификации. Так как точность классификации и является величиной, которую необходимо максимизировать, то можно взять в качестве функционала качества именно её значение.

Вторым важным элементом генетического алгоритма является стратегия скрещивания. Их существует огромное количество, а самыми распространенными являются следующие:

* Точечное скрещивание,
* Двухточечное скрещивание,
* Многоточечное скрещивание,
* Равномерное скрещивание,
* Инверсия,
* Кольцевое скрещивание.

Перечисленные методы используются в различных работах и экспериментах, однако использовать их в данной работе не представляется возможным. Эти методы считаются методами классического генетического алгоритма, который решает задачу оптимизации дискретных параметров. Параметры предложенного в работе алгоритма будут рассмотрены далее, а сейчас необходимо знать, что только часть из них дискретна, а остальные непрерывны.

Непрерывные генетические алгоритмы используют другие стратегии скрещивания, такие как:

* Плоский кроссовер,
* Простейший кроссовер,
* Арифметический кроссовер,
* Геометрический кроссовер,
* Смешанный кроссовер,
* Линейный кроссовер,
* Дискретный кроссовер,
* Расширенный линейный кроссовер,
* Эвристический кроссовер,
* Нечеткий кроссовер,
* SBX-кроссовер.

Более подробный обзор и описание приведено в статье [8].

В данной работе использовались линейный и арифметический методы скрещивания, реализованные согласно приведенным формулам.

Мутации используются для случайного изменения некоторых параметров случайных особей. Направлены они в основном на достаточно резкую модификацию значений для получения сильно отличающихся особей, что позволяет увеличить пространство поиска, замедлить сходимость алгоритма и уменьшить вероятность застревания в локальном экстремуме.

Ещё одним инструментом генетического алгоритма считаются «катаклизмы». Идея заключается в том, чтобы после завершения работы нескольких поколений произвести «встряску», т.е. уничтожить часть особей с меньшей приспособленностью и заменить их новыми.

## Оценка качества классификационного правила

Рассмотрим несколько классических и широко используемых методов оценки качества классификации.

Согласно первому подходу показатель качества классификационного правила может рассчитываться на основании показателей точности *(precision)* и специфичности *(specificity)*. Для расчета этих показателей применяются следующие формулы (1.19) и (1.20):

, (1.19)

, (1.20)

где – количество путей, которые покрыты правилом и имеют такой же класс, как предсказанный правилом;

– количество путей, которые покрыты правилом и имеют класс, отличающийся от предсказанного правилом;

– количество путей, не покрытых правилом и имеющих класс, отличающийся от предсказанного правилом;

– количество путей, которые не покрыты правилом, но имеют класс, предсказанный правилом.

Тогда показатель качества правила можно вычислить следующим образом по формуле (2.7):

, (1.21)

Величина принимает значения от 0 до 1.

Для оценки качества извлекаемых правил также может использоваться второй подход, отличающийся от рассмотренного ранее. Суть его состоит в использовании других мер, таких как мера покрытия (*coverage*) и мера уверенности (*confidence*). Определяются они так, как показано в формулах (1.22) и (1.23).

, (1.22)

, (1.23)

В этом случае показатель качества правила будет рассчитываться по формуле (2.14).

(1.24)

Другие подходы к оценке и расчету показателя качества классификации описаны в [16] и [10].

**1.7 Выводы по первой главе**

В первой главе была дана постановка задачи классификации, проведен анализ популярных методов, которые используются для её решения. Была описана процедура сведения задачи классификации к задаче поиска путей специального вида на графе и рассмотрены некоторые методы дискретизации, использование которых позволяет уменьшить размерность задачи. Были описаны используемые в работе метаэвристические алгоритмы – генетический и муравьиной колонии, рассмотрены способы оценки показателя качества классификационных правил для предложенного алгоритма.

# ГЛАВА 2 ПРОГРАММНЫЙ ИНСТРУМЕНТАРИЙ ДЛЯ ОПТИМИЗАЦИИ ПАРАМЕТРОВ МЕТАЭВРИСТИЧЕСКИХ АЛГОРИТМОВ

## 2.1 Основные идеи разработанного программного продукта

Из постановки задачи классификации, приведенной в первой главе, следует, что цель разработанного приложения – строить классификаторы, основанные на решающих правилах, сходных с правилами деревьев принятия решений.

В первой главе описан набор алгоритмов, реализация которых в одном программном инструменте позволит получать решение задачи классификации в виде классификатора, основанного на наборе решающих классификационных правил, с подбором параметров алгоритма муравьиной колонии при построении правил в автоматическом режиме. Такой подход позволяет пропустить часть работы по анализу данных с точки зрения важности и информативности показателей, которые описывают объекты тренировочной выборки.

В этой главе будет описан процесс разработки и проектирования программного продукта, реализующего озвученные в первой главе идеи и комбинирующего рассмотренные алгоритмы генетики и муравьиной колонии для построения единого классификатора.

Построение и поиск классификационных правил на графе будет осуществлять алгоритм муравьиной колонии. Подробно процедура будет рассмотрена далее. Алгоритм муравьиной колонии имеет набор параметров, значения которых необходимо подбирать под каждый набор индивидуально, так как на основе проведения экспериментов и анализа полученных результатов было установлено, что их оптимальные значения зависят от структуры и характеристик данных в тренировочной выборке.

Так как зависимость значений параметров от структуры данных в явном виде аналитическими методами получить невозможно, то возникает вопрос определения не только оптимальных параметров, но ещё и начальных. Интервалы изменения значений каждого из параметров были выбраны эмпирически на основании анализа результатов тестирования. Предлагаемые оценки для каждого из параметров, а также их содержательная характеристика приведены в следующих разделах.

Подбор оптимальных параметров вручную осуществлять представляется затруднительным, поэтому предлагается написать модуль, реализующий оптимизацию параметров при помощи генетического алгоритма. В этом случае необходимо лишь проанализировать структуру графа после проведения процедуры дискретизации данных и дать приближенные начальные оценки каждому из параметров.

Можно считать, что получение значений параметров близких к оптимальным состоит из двух этапов:

* Выбор начальных интервалов изменения значений каждого из параметров для генерации первого поколения,
* Запуск генетического алгоритма для поиска оптимальных значений каждого параметра.

Таким образом, все что требуется для получения классификатора под нужный набор данных – это загрузить данные в приложение, а всю работа по анализу и оценке будет выполнена автоматически.

Эмпирические интервалы под каждый из параметров хранятся в виде формул в коде приложения в отдельном блоке, поэтому при необходимости их подкорректировать можно просто поменять некоторые из формул и перезапустить процедуру обучения алгоритмов.

В общем виде ***основную идею*** разработанного приложения можно сформулировать следующим образом: *генетический алгоритм будет работать с коллекциями муравьиных колоний, которые будут генерироваться поколениями на основании результатов работы предыдущих итераций.*

Алгоритм в таком виде работает поэтапно. Однако, благодаря тому, что в процессе построения правил колонии работают независимо и не обмениваются в это время никакой информацией между собой, алгоритм в таком виде хорошо распараллеливается и может быть запущен на целом кластере ЭВМ.

Отсутствие обмена информацией и опытом между муравейниками не является критичной проблемой само по себе, но благодаря использованию генетического алгоритма для управления ходом работы алгоритма муравьиной колонии, можно считать, что эта проблема решается при проведении скрещивания и генерации колоний, следующих после начального поколений.

В то же время идея распараллеливания органично сводится к идее построения и хранения не одного лучшего набора классификационных правил, а к следующему:

* Построение множества классификаторов со своими наборами классификационных правил,
* Хранение всего множества классификаторов, их наборов правил, параметров и данных, на которых он обучался,
* Проведение при тестировании голосования между всеми полученными классификаторами, что напоминает алгоритм *леса случайных деревьев (Random Forest)* и подходы, основанные на *бустинге (boosting)* и *баггинге (bagging)*.

Особенностью является то, что в отличие от алгоритмов деревьев принятия решений, дерево не строится иерархически через добавление одной вершины к дереву за раз и разделение тренировочной выборки на части, а выделяются целые ветки сразу, что позволяет исключать из рассмотрения часть покрытых правилом записей и уменьшает общее число объектов в тренировочной выборке с ходом итераций муравьиной колонии.

Приложение реализовано с использованием технологий ООП, для написания кода использовался язык C# 5 в среде Visual Studio 2017 [14].

**2.2 Описание входных и выходных данных**

Из постановки задачи классификации, приведенной в первой главе, следует, что в качестве входных данных программному продукту необходимо дать набор записей, для каждой из которых заранее известна метка класса, к которому объект относится – *тренировочную выборку*.

Тренировочные выборки для тестирования и проверки работы разработанного приложения были взяты с сайта ***OpenML***. Решено было так поступить из-за того, что выбор наборов данных там достаточно широкий, наборы различаются, и есть возможность найти такой набор, который будет проверять слабые места тестируемого алгоритма.

Для многих наборов на сайте ***OpenML*** есть задачи, по которым можно сравнивать результаты разработанного алгоритма с тем, что использовали для классификации набора другие пользователи.

Каждый набор можно скачать в виде файла *\*.arff*, что позволяет не писать несколько различных парсеров для считывания данных, а ограничиться единственным.

Другими словами, преимущество заключается в том, что все файлы будут строго типизированной структуры, с выделенными структурными блоками:

1. Метаинформация о наборе – название, автор, количество записей в наборе, количество и тип атрибутов и содержательная характеристика каждого атрибута с точки зрения информации, которую он описывает.
2. Блок ***@relation*** – для каждого из атрибутов перечислены все его значения, если они номинальные, или указан тип – *integer/real* и др.
3. Блок ***@data*** – каждая строка представляет собой описание одного объекта рассматриваемой задачи, значения перечислены в порядке, определенном порядком атрибутов в блоке ***@relation.***

Однако не меньший интерес представляет информация о том, как изменялись параметры муравьиных колоний с ходом итераций. Анализ этих данных позволит переобучить классификатор под выбранный набор, но стартовые значения параметров будут взяты не эмпирически, а равными близким к оптимальным значениям, полученным в результате работы приложения. В таком случае мы заранее можем приблизить результат к лучшему возможному, но не будет необходимости осуществлять поиск в широкой окрестности значений атрибутов, что позволит осуществить поиск оптимального классификатора более направленно.

Также в результате работы приложения можно получить наборы классификационных правил каждого из построенных классификаторов и проанализировать распределение лучших муравьев по типам, зависимость качества классификации от включения или отключения некоторых модификаций, сходимость параметров муравьев .

Таким образом, выходными данными будут являться следующие наборы файлов:

1. История генерации поколений – список, всех построенных классификаторов и их характеристики,
2. Наборы классификационных правил, соответствующие каждому из классификаторов.

Примеры каждого файла будут приведены и проанализированы в третьей главе.

**2.3 Описание оптимизируемых параметров**

При решении задачи построения классификационных правил при помощи алгоритма муравьиной колонии используется ряд параметров, выбор значений которых оказывает влияние на порядок работы процедур, что в свою очередь, оказывает влияние на качество работы алгоритма. Рассмотрим эти параметры более подробно.

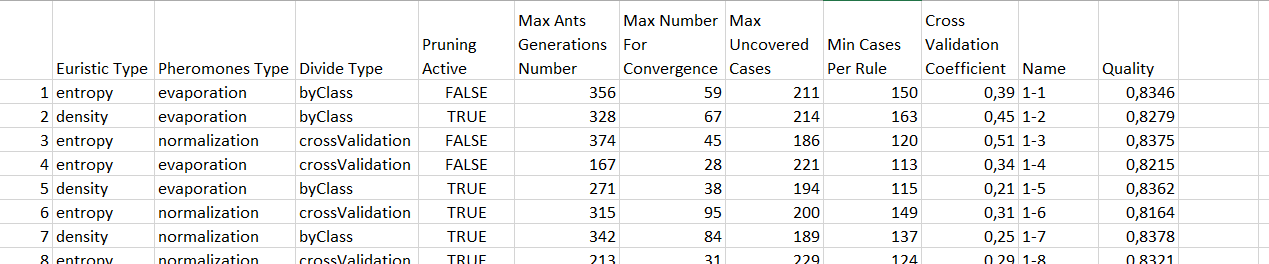
* Предельное число поколений муравьев в каждом муравейнике;
* Минимальное число найденных подряд одинаковых правил;
* Максимальное число непокрытых записей в наборе;
* Минимальное число записей, которые должно покрывать каждое правило;
* Весовые коэффициенты при расчете вероятностей переходов муравья;
* Пределы изменения количества феромонов – локальная характеристика вершин;
* Коэффициент кросс-валидации, определяющий величину каждого обучающего набора в виде доли от общего числа записей.

Как говорилось ранее, пределы изменения параметров можно оценить эмпирически, но более точный их подбор нужно осуществлять программно.

Предлагается каждому из параметров задать следующие начальные значения:

* *Предельное число поколений муравьев в каждом муравейнике* – каждое поколение муравьев является тройкой муравьев с различными весовыми параметрами, что позволяет считать их муравьями различных типов, которые в процессе движения по графу будут осуществлять переходы по-разному, получая пути, сильно отличающиеся друг от друга. Предлагается интервалом изменения значения этого параметра брать промежуток , где – число вершин полученного графа.
* *Минимальное число найденных подряд одинаковых правил* – этот параметр используется вместе с предыдущим в качестве ограничения на количество итераций, которые осуществит каждый муравейник в процессе работы. Считаем, что как только несколько муравьев одного типа нашли одинаковых правил подряд, то дальше продолжать текущую итерацию бессмысленно, так как вероятность получения нового пути очень низкая, а популяция сошлась на одном пути. Выбирается в интервале ,
* *Максимальное число непокрытых записей в наборе* – допускаем, что некоторые записи в обучающих выборках могут быть ошибочны. В таком случае часть из них будет проигнорирована и не повлияет на качество работы алгоритма. Предположим, что значение этого параметра не должно превышать 3% от количества всех записей в выборке.
* *Минимальное число записей, которые должно покрывать каждое правило* – для того, чтобы регулировать насколько сильно классификационные правила будут обобщать исходную выборку. Берется примерно равным предыдущему параметру .
* *Весовые коэффициенты при расчете вероятностей переходов муравья* – как было указано выше, используются для разделения муравьев на различные типы. Исходя их принципа расчета вероятности перехода в вершину, изменение значений параметров ведет к тому, что изменяется влияние на вероятность или локальной характеристики вершины – количества феромонов, или глобальной характеристики – эвристической функции вершины. Для разных типов муравьев берем различные интервалы, но общий принцип состоит в следующем: первая группа отдает предпочтение вершинам с большим количеством феромона, другая – с большим значением эвристической функции, третья – сбалансирована.
* *Пределы изменения количества феромонов* – использование этих параметров позволяет остановить как слишком быстрое накопление феромонов в небольшом множестве вершин после прохождения первых муравьев, так и испарение феромона на редко посещаемых вершинах, чтобы не исключать их из рассмотрения совсем.
* Ещё одним параметром является *количество деревьев*, т.е. обучающих подмножеств, на которые будет проводиться разбиение. Предлагается брать в пределах 50 – 100.

На выходе алгоритма можно получить файл следующей структуры, который показан на рисунке 2.1.



**Рисунок 2.1 – Пример выходных данных о классификаторах**

**Источник: собственная разработка**

Каждая строка, являющаяся геномом одной особи, представляет собой список пар «параметр – значение».

Параметр ***Euristic Type*** определяет, каким образом будет рассчитываться значение эвристической функции.

Параметр ***Pheromones Type*** определяет вариант, по которому будет происходить пересчет оценки вершины – количество феромонов в ней.

Параметр ***Divide Type*** влияет на принцип разбиения тренировочной выборки в процессе обучения: можно использовать кросс-валидацию или обучать каждый из муравейников на определение записей только одного из имеющихся классов.

Флаг ***Pruning Active*** включает или отключает усечение правил после их построения: муравейники для которых параметр принимает значение *false* строят более подробные правила, которые соответствуют меньшему количеству исходных записей, а в другом случае – правила, которые описывают большее множество записей, но обычно обладают меньшей точностью классификации.

Параметры ***Мах Ants Generations Number***, ***Max Number For Convergence***, ***Max Uncovered Cases***, ***Min Cases Per Rule*** соответствуют следующим параметрам:

* Предельное число поколений муравьев в каждом муравейнике;
* Минимальное число найденных подряд одинаковых правил;
* Максимальное число непокрытых записей в наборе;
* Минимальное число записей, которые должно покрывать каждое правило;

***Name*** используется для идентификации муравейника и состоит из пары чисел: номер поколения – номер муравейника в этом поколении.

***Quality*** – это и есть качество полученного муравейником набора классификационных правил, протестированного на выборке.

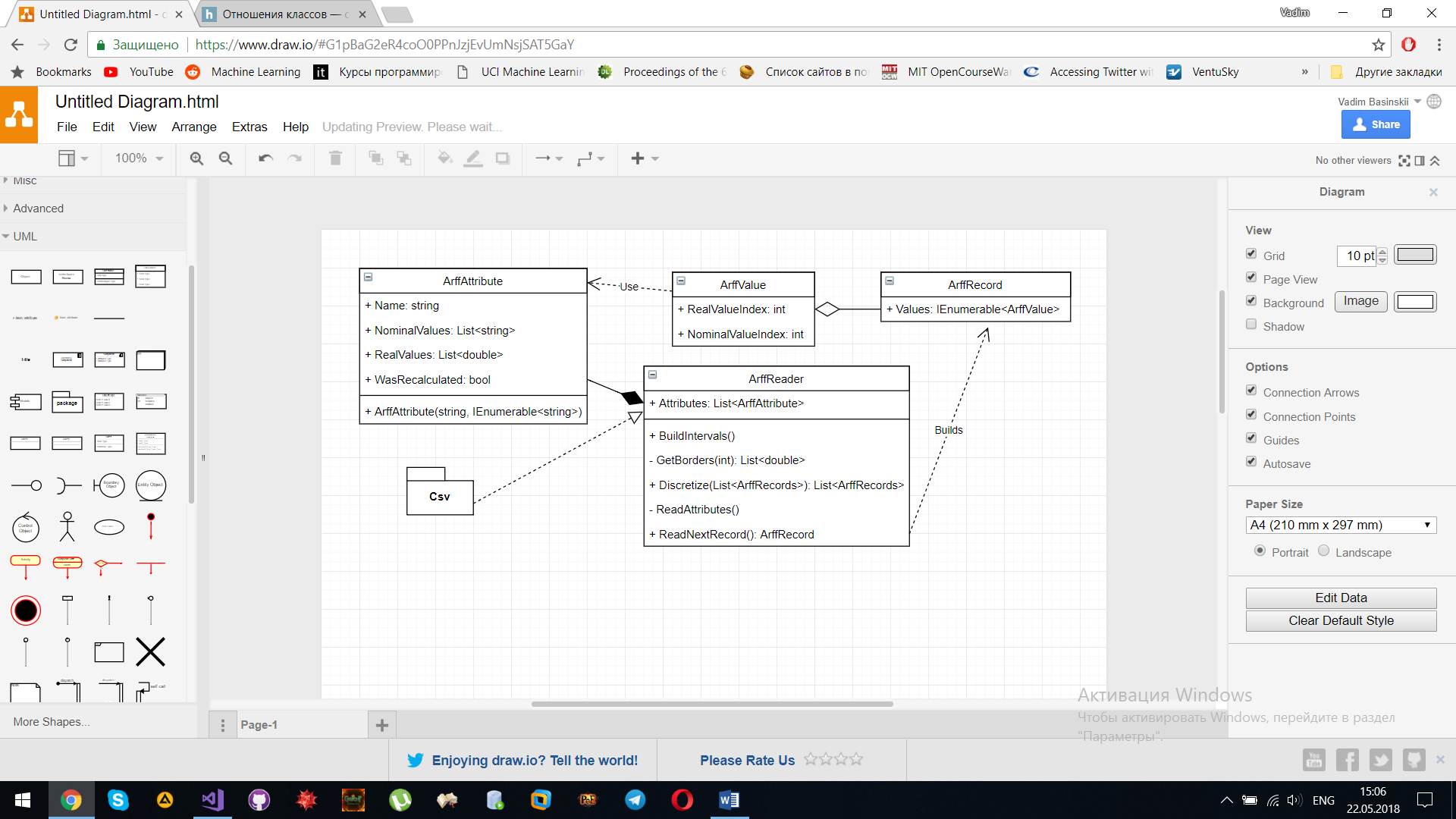
Параметр Сross Validation Coefficient имеет влияние только в случае, если значение параметра ***Divide Type*** установлено в *crossValidation*. Он определяет насколько сильно будет дробиться исходная тренировочная выборка при обучении.

Как говорилось ранее, в работе использовались как операторы классического, так и непрерывного генетического алгоритма. Связано это как раз с тем, что значения параметров последней группы – непрерывные величины.

**2.4 Проектирование системы классов**

Структуру разработанной программы представим при помощи UML диаграмм классов [11]. Решение состоит из двух проектов: **ArffSharp** и **MAClassification**.

Проект **ArffSharp** отвечает за работу с наборами данных – осуществляет считывание, парсинг и разбор тренировочной выборки данных.



**Рисунок 2.2 – Диаграмма классов проекта ArffReader**

**Источник: собственная разработка**

В проекте выделяются следующие модели:

1. ***ArffAttribute*** соответствует одному из признаков, при помощи которых описываются объекты набора данных. Поле *Name* – название признака, флаг *WasRecalculated* используется при проверке проведения дискретизации перед маппингом атрибутов на сущности ***DataAttribute*** из **MAClassification**. Массив *NominalValues* для номинальных изначально атрибутов заполняется при считывании блока ***@relation*** во время первичного разбора файла. Для атрибутов, которые не являются дискретными в исходном наборе сначала заполняется массив *RealValues*. После полного считывания всех записей и получения всех встреченных значений атрибута проводится их дискретизация методом *Discretize()* класса ***ArffReader*** одним из описанных в первой главе методов.
2. ***ArffValue*** хранит одно значение одного атрибута как индекс позиции в массиве *RealValues* или *NominalValues* соответствующего атрибута ***ArffAttribute.***
3. ***ArffRecord*** – массив объектов ***ArffValue***. Каждый объект класса соответствует одной записи исходных данных и является описанием одного объекта из исходной задачи.

Класс ***ArffReader*** – основной в описываемом проекте. Весь функционал по парсингу, разбору исходного файла заложен сюда, так же, как и все создание объектов ***ArffRecord*** и ***ArffAttribute***.

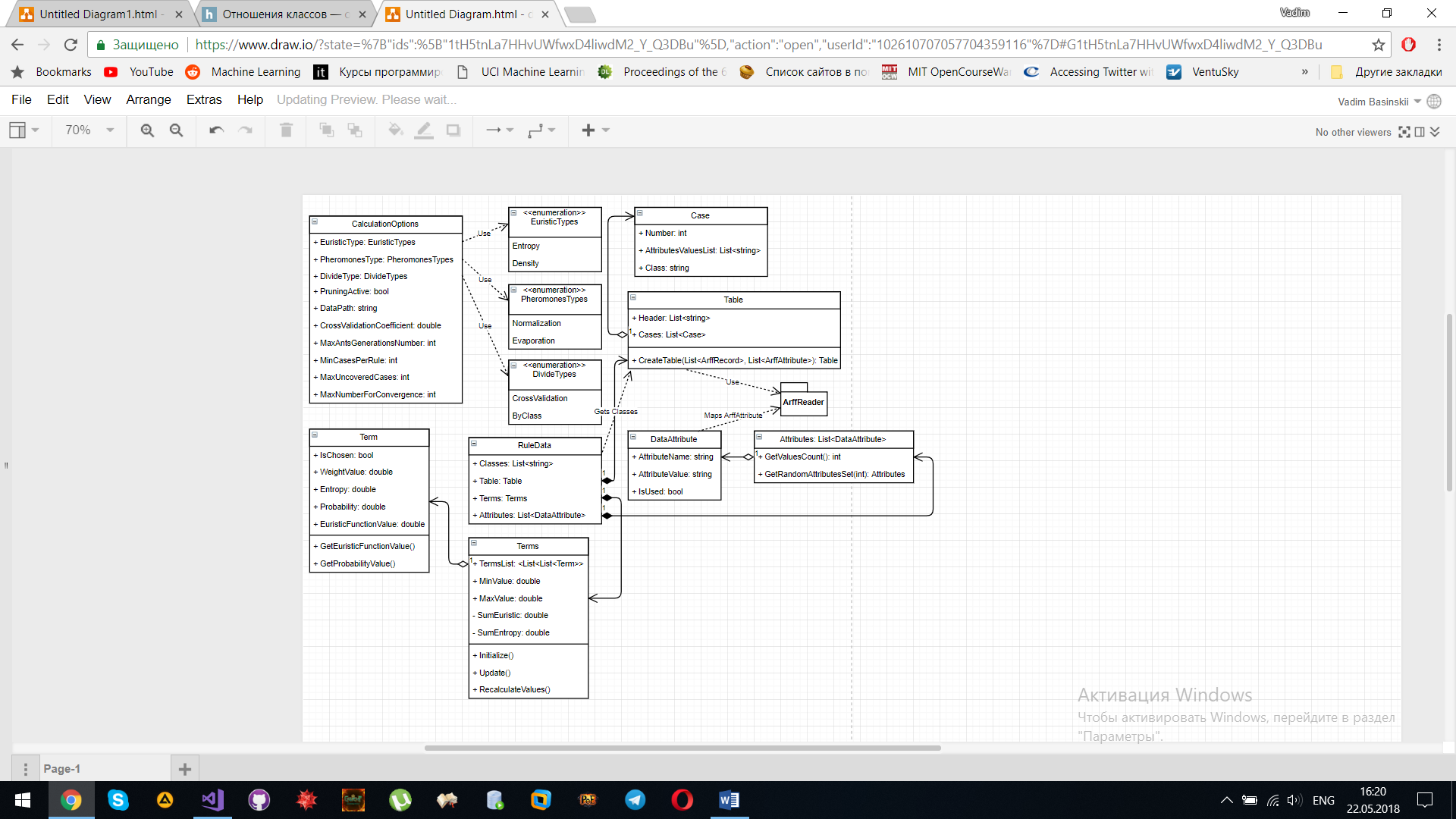
Метод *BuildIntervals()* осуществляет оценку разброса значений каждого атрибута, к которому необходимо применить процедуру дискретизации, т.е. для всех атрибутов, у которых массив *NominalValues* не содержит элементов. После оценки разброса значений определяется число интервалов разбиения и рассчитываются значения границ разбиения методом *GetBorders()*. По полученным границам разбиения строятся номинальные значения атрибутов вида **«x <value»**. Метод *Discretize()* ставит в соответствие исходному значению атрибута из массива *RealValues* новое значение из массива *NominalValues*.

Методы *ReadAttributes()* и *ReadNextRecord()* из исходного файла данных читают весь список атрибутов и следующую по порядку запись соответственно.

Проект **MAClassification** осуществляет реализацию описываемого в работе алгоритма. Включает в себя:

* классы моделей, которые являются основными сущностями,
* классы, которые отвечают за реализацию основных функций и методов, через обработку и взаимодействие на уровне объектов первой группы,

Основные модели представлены на UML диаграмме классов, показанной на рисунке 2.3.



**Рисунок 2.3– Диаграмма классов моделей данных проекта**

**Источник: собственная разработка**

На диаграмме классов выше описано как в приложении обеспечивается представление данных.

Записи выборки вносятся в объект класса *Table*. Классы *Terms* и *Attributes* используются для описания структуры графа и определения порядка построения правила.

Класс *Table* используется для хранения исходных записей тренировочной выборки.

*Header* хранит информацию о названиях атрибутов, которые описывают исходные данные.

*Cases* описывает все записи в выборке. Содержит набор объектов *Case*. Каждый объект типа *Case* несет в себе следующую информацию:

* *Number* – порядковый номер записи в наборе,
* *AttributesValuesList* – набор значений атрибутов, которые описывают соответствующий объект. Задаются в том же порядке, как упорядочены атрибуты в *Header*,
* *Result* – метка класса, к которому относится объект. Метка является одним из значений набора *Results*, который формируется при считывании данных.

Процесс инициализации исходных данных осуществляется следующим образом:

* Считываются все записи для формирования объекта *Table*.
* Выделяются множество атрибутов и множество значений каждого из них.
* Формируется объект *initialTerms*, который является представлением задачи классификации в виде графа, как было показано ранее.
* Для каждого элемента *Term*, из которых состоит объект *initialTerms* рассчитываются его характеристики: значение эвристической функции и начальное количество феромона.

*CalculationOptions* – набор значений параметров алгоритма. Каждой колонии передается свой экземпляр с отличными от других параметрами. Список полей соответствует параметрам, описанным в предыдущих разделах работы.

В качестве полей *CalculationOptions* также используются значения перечислений:

* *DivideType* – значения *CrossValidation* и *ByClass*. Определяет каким образом будет производиться разбиение тренировочной выборки для осуществления обучения алгоритма.
* *EuristicType –* значения *Entropy* и *Density*. Определяет какая функция будет использоваться для расчета глобальной характеристики вершин графа – эвристическая на основании энтропии значений атрибутов или на основании плотности путей через каждую из вершин.
* *PheromonesType –* значения *Normalization* и *Evaporation*. Определяет каким образом будет вестись пересчет локальной характеристики вершин графа – количества феромонов.

После инициализации и расчета всех необходимых значений объект *initialTerm* хранит следующую информацию, которая показана на рисунке 2.4.



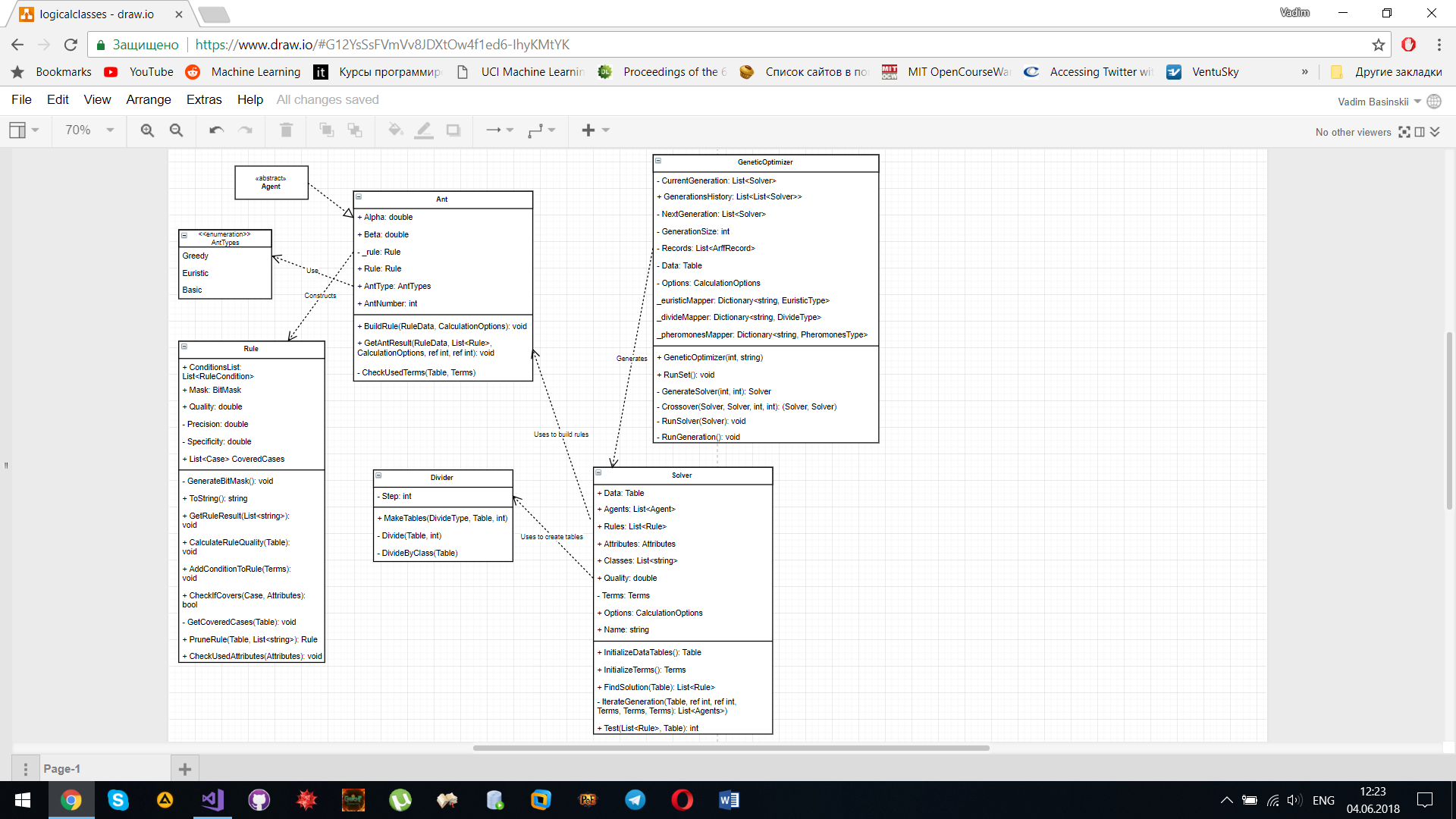
**Рисунок 2.4 – Пример описания вершин графа для задачи классификации**

**Источник: собственная разработка автора**

Объект типа *Term* является образом одной вершины графа. Для него определены следующие параметры:

* *AttributeName* – название атрибута.
* *AttributeValue* – значение указанного атрибута.
* *IsChosen* – флаг, который используется для исключения вершины из рассмотрения при переходах по графу во время построения одного пути.
* *WeightValue* – количество феромона в вершине.
* *EuristicFunctionValue* – значение эвристической функции вершины.
* *Probability* – вероятность перехода в вершину. Эта величина рассчитывается при построении пути конкретным муравьем, так как при расчете необходима информация о характеристиках муравья для учета важности параметров.
* *Entropy* – рассчитывается в случае, если в качестве эвристической функции используется функция на основании значений условной энтропии выражений.

Рассмотрим классы с основным функционалом и посмотрим на их взаимосвязь. Диаграмма классов, реализующих основные методы и процедуры, приведена на рисунке 2.5.



**Рисунок 2.5– Диаграмма классов проекта MAClassification**

**Источник: собственная разработка**

*Agent* – абстрактный класс, который определяет контракт для классов, которые могут использоваться для построения правил. В работе от него наследуется класс Ant, для которого реализованы требуемые методы: *BuildRule()* и *GetResult()*.

Для класса *Ant* предусмотрено свойство *AntType*, которое определяет в каких интервалах и каком отношении будут выбираться параметры , определяющие характер муравья при построении правил. Значения параметров хранят свойства *Alpha* и *Beta*. Допустимыми значениями свойства *AntType* являются **Basic, Greedy, Euristic**, которые находятся в связанном перечислении: *enum AntTypes*. Подробный анализ влияния значений параметров на получаемые результаты приведен в следующей главе в обзоре модификаций алгоритма.

Метод *BuildRule()* осуществляет построение одного классификационного правила согласно процедуре, рассмотренной в следующем разделе. Метод *GetResult()* используется для дополнительной обработки перед построением правила, а после того как правило было построено – для осуществления попытки усечения правила, если опция включена, и сброса состояния графа вершин в исходное состояние – разрешение для посещения всех вершин. Делается это в связи с тем, что при описании требований к структуре графа и правил было указано, что в правило не должны входить вершины, соответствующие разным значениям одного атрибута. Осуществляет эту проверку метод *CheckUsedTerms().*

Одним из основных результатов работы реализованного алгоритма являются наборы классификационных правил: коллекции *List<Rule>,* полученные в результате работы различных муравейников. Класс *Rule* используется для представления объектов этого класса. Рассмотрим его структуру подробнее.

Каждое правило хранит список *List<RuleCondition> ConditionsList*, каждый элемент этого списка описывает вершину графа, а весь список целиком и является путем на этом графе, который позволяет компактно описать одну группу схожих путей.

Список *List<Case> CoveredCases* хранит набор путей, которые правило покрывает – это те объекты обучающей выборки, которые правило классифицирует с высокой точностью.

Поля *Specificity* и *Precision* – меры, рассчитываемые для вычисления значения качества правила – *Quality*, которое позволяет определить наиболее подходящие правила для добавления в итоговый набор.

Как упоминалось ранее, в приложении заложена опция включения и отключения процедуры *усечения правил*. Рекурсивный характер процедуры и необходимость довольно «тяжелых» пересчетов характеристик правила при изменении набора *ConditionsList* приводят к тому, что требуется оптимизировать время работы данной процедуры. Было решено осуществить это следующим образом: каждому правилу ставится в соответствие битовая маска – *BitMask*. Это набор из нулей и единиц, по которой можно быстро определить, через какие вершины правило проходит. В свою очередь при построении правил и определении какие пути оно покрывает появляется зависимость для некоторого набора вершин и некоторого набора путей. Информация об этой зависимости с ходом итераций не теряется, в записывается в единый глобальный *Dictionary<BitMask, List<Case>>*. Это позволяет для каждого правила в ходе работы алгоритма осуществлять поиск подходящих под него объектов быстрее, что соответственно положительно сказывается на времени выполнения процедуры усечения правила. Сама процедура усечения правил *PruneRule()* описана в следующем разделе.

Для каждого правила необходимо определить класс, который оно предсказывает и его вероятность. В работе исследовались наборы данных, для которых требовалось решить задачу бинарной классификации. Поэтому вместо хранения массива в виде зависимости – «класс – вероятность» для каждого правила хранится метка класса, к которому относится большая часть записей, которые оно покрывает, а вероятностью класса и будет мера точности – *Quality*.

На этапе тестирования точности набора классификационных правил требуется проверять какие правила среди найденных покрывают полученную запись. Эту проверку осуществляет метод *CheckIfCovers()*, который на вход принимает правило и запись и проверяет соответствие вершин, через которые проходит правило и сам путь, определяемый характеристиками объекта.

Для проведения обучения необходимо разбить исходную тренировочную выборку на несколько частей. Осуществляет это объект класса *Divider.* Он имеет три метода: *Divide(), DivideByClass()* и *MakeTables().* Метод *MakeTables()* на вход принимает объект типа перечисление *DivideType* и в зависимости от его значения вызывает один из методов. Поле *Step* используется в случае, если разделение данных идет по принципу кросс-валидации, и необходимо знать, сколько записей нужно выбрать для построения тренировочной и тестовой таблиц.

Класс *Solver* реализует логику экземпляра колонии. В нем создаются поколения муравьев, инициализируются необходимые для расчетов объекты и строятся наборы классификационных правил.

Инициализация данных осуществляется методами *InitializeDataTables()* и *InitialzeTerms()*. Метод *IterateGeneration()* осуществляет построение правил одним поколением популяции колонии – запускает *MaxAntsGenerationNumber* муравьев каждого типа *AntType* и возвращает результат в метод *FindSolution().* Метод *FindSolution()* выполняет итерации по текущему набору данных. После получения списка муравьев из метода *IterateGeneration()* выбирается самое качественное, а из текущего набора данных исключаются записи, покрытые выбранным правилом. Метод *FindSolution()* запускает поколения муравьев до тех пор пока не покрытыми останется число записей меньшее, чем допустимое – *MaxUncoveredCases.*

Класс *GeneticOptimizer* осуществляет генерацию муравейников, т.е. экземпляров класса *Solver*. Получение решения для набора данных выполняется вызовом метода *RunSet()*.

Поля *\_euristicMapper*, *\_pheromonesMapper* и *\_divideMapper* обеспечивают возможность случайного выбора соответствующих полей *CalculationOptions*.

При создании экземпляра этого класса инициализируются поля *CurrentGeneration*, *NextGeneration*, *GenerationSize*, *GenerationHistory* и часть полей *CalculationOptions* – *DataPath* и *CrossValidationCoefficient*.

Вызов метода *RunSet()* запускает работу проекта *ArffReader* для считывания файла по пути *DataPath* с последующим вызовом его функций для дискретизации данных. После этого проводится маппинг объекта *List<ArffRecord>* на объект *Table*.

Запускается цикл по поколениям колоний. На первом поколении вызывается метод *GenerateSolver()*, который создает экземпляры класса Solver с начальными параметрами, выбранными в начальных интервалах, рассмотренных ранее. На последующих поколениях вызывается метод *Croosover(Solver, Solver)*, который на основании характеристик колоний предыдущего поколения строит новые колонии, которые будут запущены в текущей итерации. Генерация новых колоний осуществляется при помощи арифметического кроссовера.

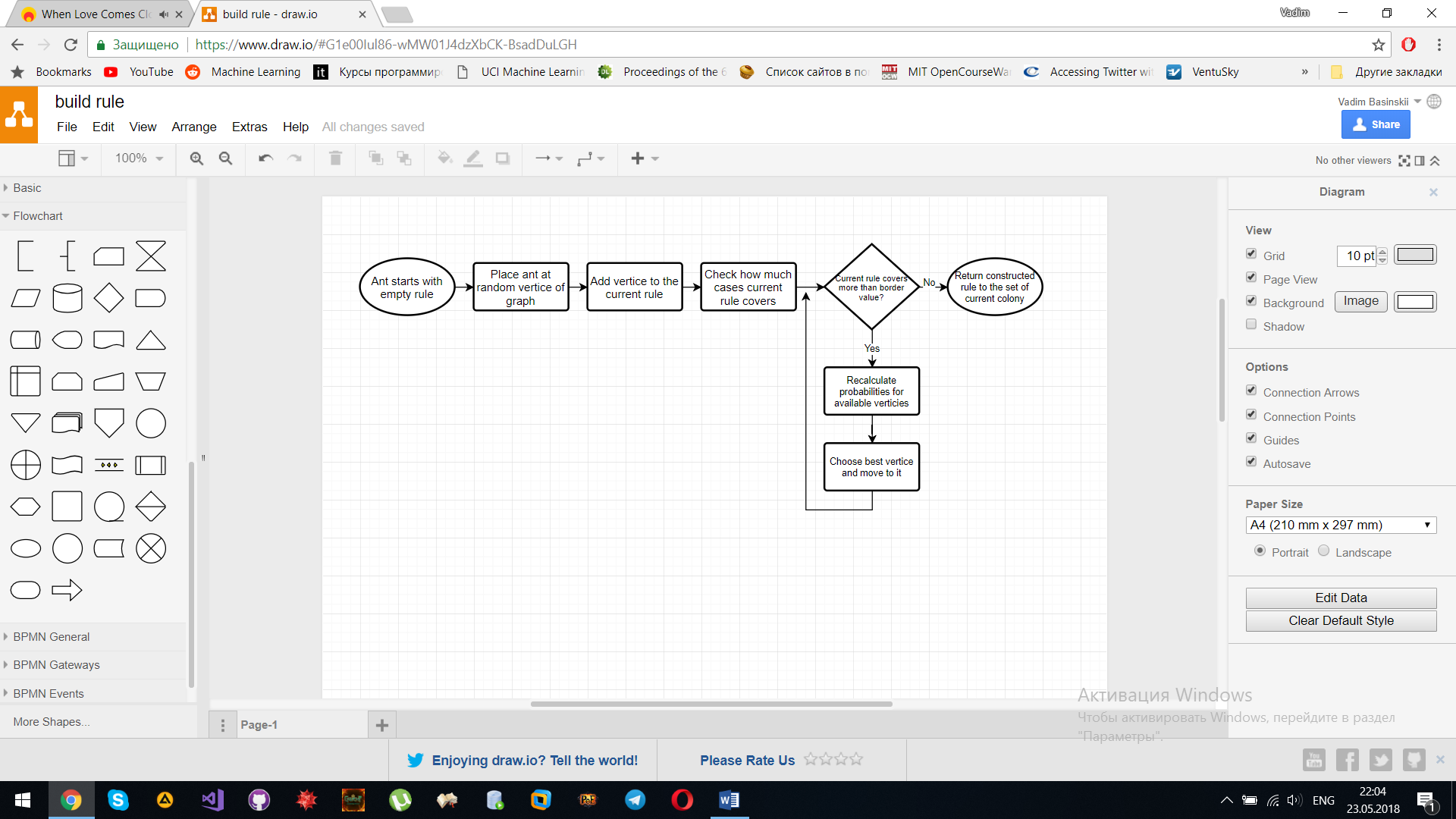
Запуск текущего поколения муравейников выполняется методом *RunGeneration()*, который запускает параллельно в несколько потоков метод *RunSolver(Solver)* для каждой колонии в наборе *CurrentGeneration*.

История запусков поколений *GenerationsHistory* записывается в файл для последующего анализа.

**2.5 Описание основных методов и процедур**

Основные самые важные с точки зрения логики работы процедуры отобразим при помощи UML диаграмм деятельности [36].

Построение правила осуществляется как показано на рисунке 2.6. Каждое правило соответствует пути по графу одного муравья.



**Рисунок 2.6 – Общий вид процедуры построения правила**

**Источник: собственная разработка автора**

Как было описано ранее? характер муравья влияет на его стратегию построения пути и выбора вершины для перехода в процессе. Определяется это при помощи параметров *Alpha* и *Beta*.

Рассмотрим процесс построения правила подробно. Считаем, что муравей начинает с пустым правилом, причем, пустое правило соответствует всем записям выборки.

С учетом характеристик муравья рассчитываются вероятности его попадания в допустимые вершины. Процесс построения одного правила заканчивается в тот момент, когда оно соответствует числу записей меньшему, чем допустимый порог.

К правилу добавляется одна вершина. Считаем, что муравей находится в ней и дальнейшее построение пути будет вестись оттуда.

Из списка допустимых для перехода вершин исключаются все вершины, имеющие в поле *AttributeName* значение, вершина с которым уже имеется в правиле.

Определяется, какому набору записей соответствует новое правило и переопределяется класс, предсказываемый правилом. Считаем, что правило предсказывает тот класс, который имеет большая часть записей, которым правило соответствует.

После определения описанных выше характеристик правила производится расчет качества правила, по которому дальше будет происходить выбор лучшего для добавления к финальному набору.

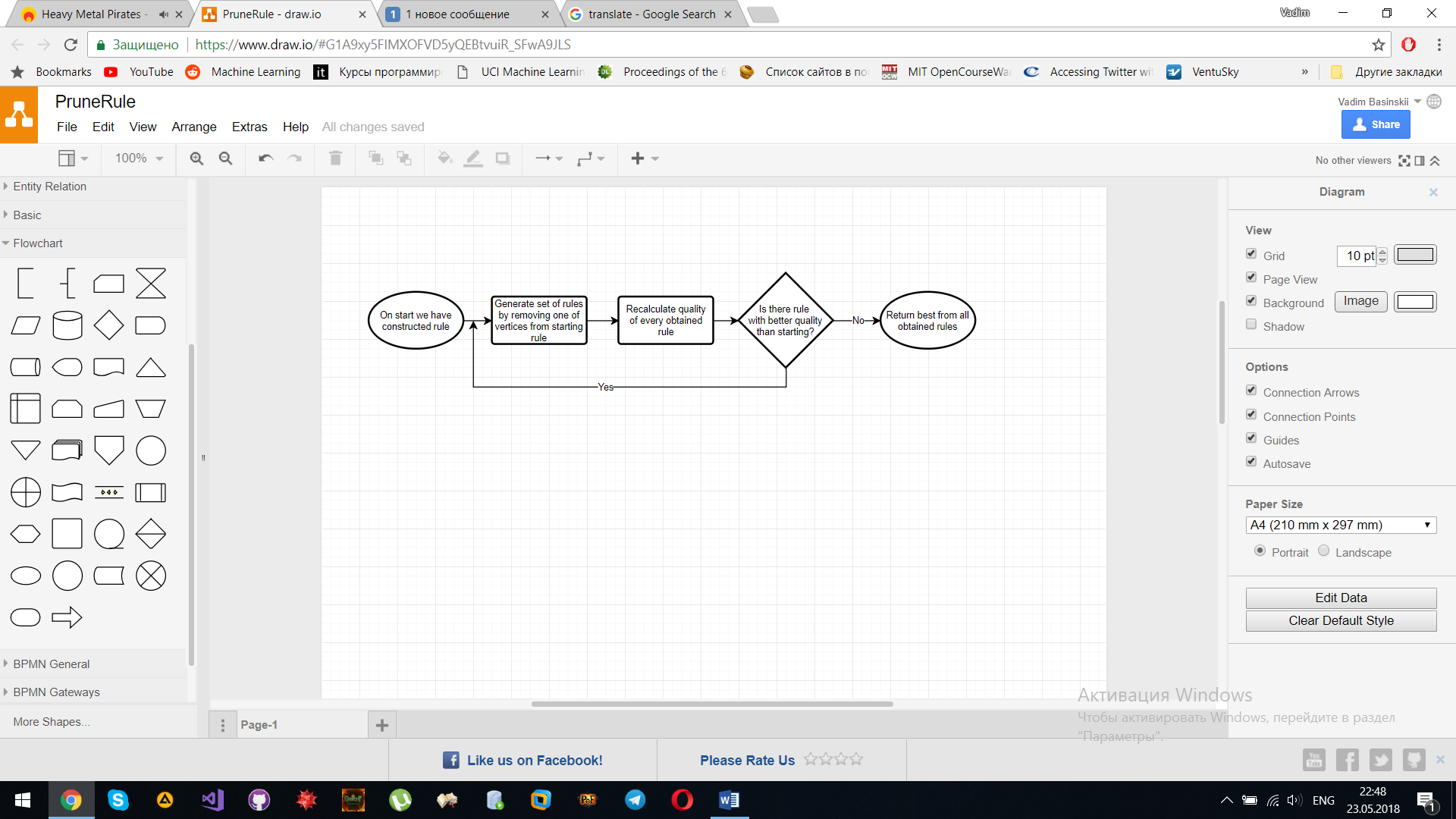
Усечение правил позволяет получать наборы, правила в которых лучше обобщают исходные множества записей. Принцип построения правил, описанный ранее, приводит к тому, что каждое полученное правило будет покрывать малое количество записей, что негативно сказывается на точности последующего тестирования.

Использование усечения правил позволяет получать более качественные правила, что повышает общую точность набора извлеченных классификационных правил при тестировании и позволяет решить проблему переобучения на тренировочной выборке.

Процедура рекурсивная, производится следующим образом:

* Из правила последовательно исключается, каждое из входящих в него условий.
* Получается набор новых правил , где – количество условий в правиле .
* Для каждого пересчитывается множество покрытых объектов, результат и качество.
* Среди всех выбирается правило с максимальным качеством .
* Если качество больше качества исходного правила – рекурсивный запуск с , иначе выход.

Процедура усечения правила осуществляется следующим образом, как показано на рисунке 2.7.

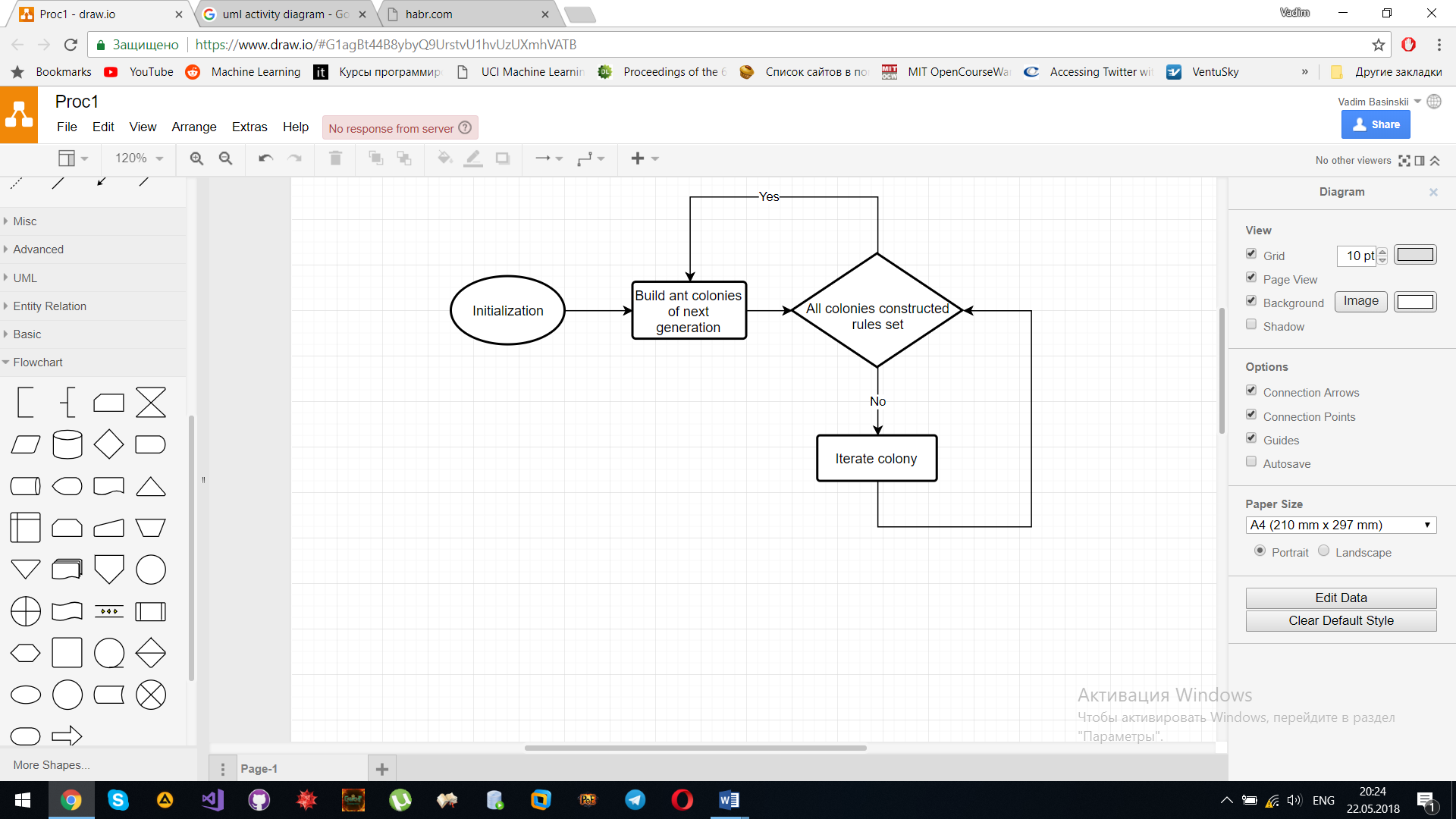


**Рисунок 2.7 – Общий вид функции, реализующей усечение правила**

**Источник: собственная разработка автора**

После загрузки набора данных пользователем управление отдается модулю ArffReader, описанному ранее в главе. Набор данных считывается и проводится маппинг данных по необходимым сущностям. После инициализации всех необходимых объектов запускается генерация первого поколения муравьиных колоний. Количество поколений, которые будут построены определяется пользователем до запуска алгоритма.

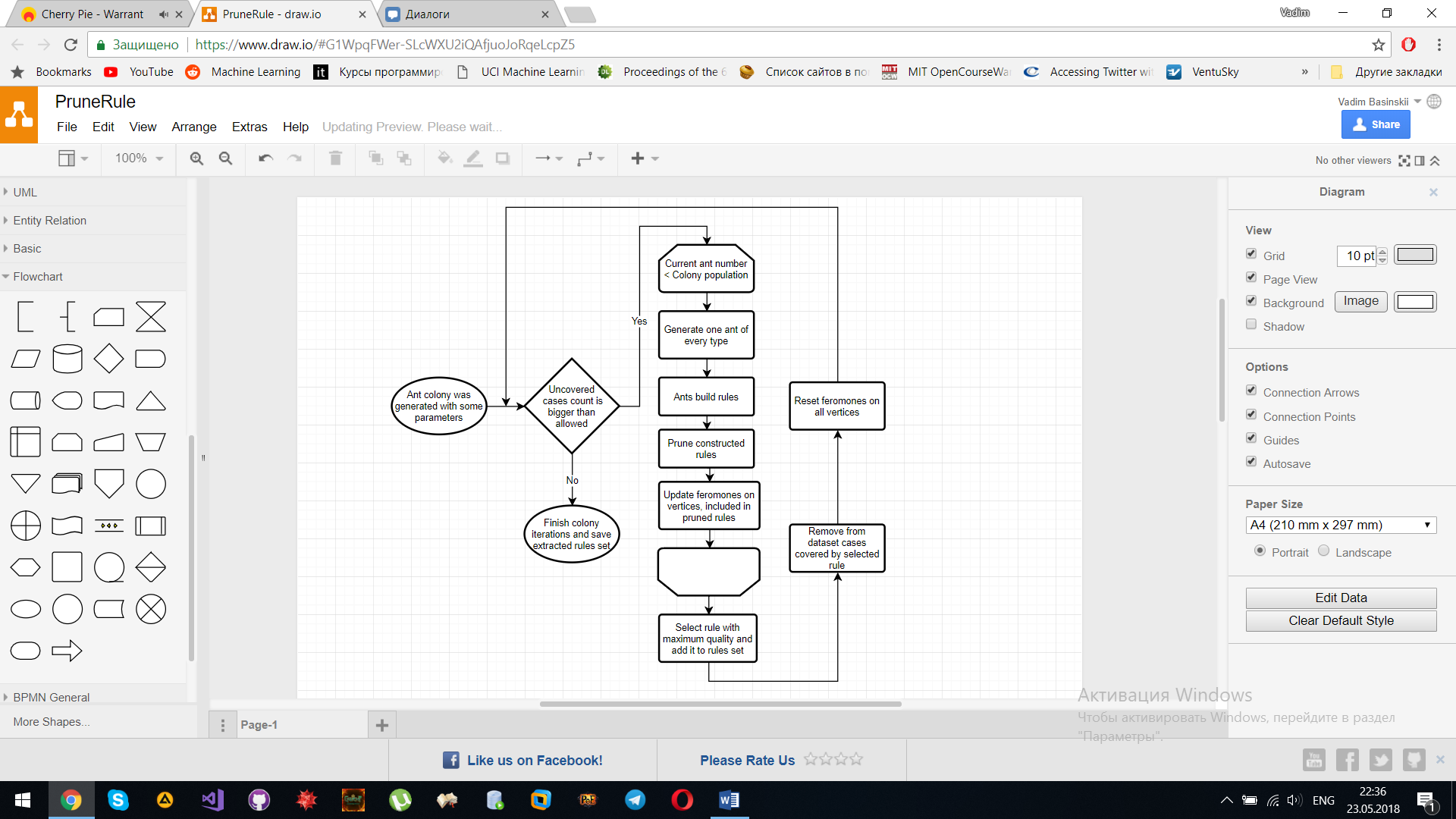
Каждая из колоний первого поколения создается с параметрами, значения которых выбраны в соответствующих интервалах, описанных ранее. В рамках одного поколения построение наборов классификационных правил выполняется параллельно. В каждой из колоний запускается своя популяция муравьев, которые могут взаимодействовать посредством обмена феромоном исключительно в пределах собственной популяции. Общий вид процедуры генетического алгоритма представлен на рисунке 2.8.



**Рисунок 2.8 ‑ Основной цикл по поколениям генетики**

**Источник: собственная разработка автора**

Процедуры, которые осуществляет класс *Solver* можно описать следующей схемой, приведенной на рисунке 2.9.



**Рисунок 2.9 ‑ Работа одной колонии**

**Источник: собственная разработка автора**

**2.6 Псевдокод алгоритма муравьиной колонии**

1. Инициализация:
   1. Считывается обучающая выборка.
   2. Выделяются атрибуты: имя и набор значений.
   3. Выделяются классы.
   4. Рассчитывается энтропия для каждой пары «атрибут – значение».
   5. Для каждой пары «атрибут – значение» рассчитываются веса феромонов, значения эвристической функции по соответствующим формулам.
2. Список выделенных правил – пустой.
3. While (количество непокрытых записей больше порогового значения)
   1. Определяются допустимые для перехода вершины.
   2. Количество муравьев = 0, количество одинаковых путей подряд = 0.
   3. Список правил текущей популяции – пустой.
   4. While (есть муравьи, не построившие путь или популяция не сошлась)
      1. Запускается 3 муравья – по одному из каждой группы – на отдельных наборах вершин. Для каждого из них правило строится и усекается как описано ранее.
   5. Из всех правил текущей популяции выбирается максимально качественное и добавляется к списку выделенных правил.
   6. Из выборки всех объектов исключаются покрытые найденным правилом.

**2.7 Выводы по второй главе**

Описаны основные идеи, которые легли в основу приложения. Продемонстрирован принцип связи генетического алгоритма и алгоритма муравьиной колонии. Проанализированы параметры алгоритмов, даны начальные эмпирические оценки интервалов значений каждого из них. Рассмотрена логика приложения, описаны основные процедуры и методы. При помощи UML диаграмм классов показана структура приложения и связи объектов в нем. Основные методы и процедуры отражены на UML диаграммах деятельности. Показано, что принимается на вход приложения, приведены требования к данным и их структуре, пояснено, что получается в результате работы программы.

# ГЛАВА 3 АНАЛИЗ И ОПТИМИЗАЦИЯ МОДИФИКАЦИЙ МЕТАЭВРИСТИЧЕСКИХ АЛГОРИТМОВ

## 3.1 Анализ использованных и разработанных модификаций

Модификация параметров муравьиного алгоритма позволяет изменять процедуру. Среди модификаций выделим следующие группы:

* Модификации пересчета оценок,
* Модификации принципа выбора вершин,
* Модификации типов муравьев,
* Модификации метода построения графа.

Многие из модификаций алгоритма муравьиной колонии описаны другими авторами, что говорит о достаточно большой популярности метода, а некоторые были предложены непосредственно автором данной магистерской диссертации в процессе исследования.

К модификациям *пересчета оценок* отнесем следующие:

***Пересчет количества феромонов с использованием коэффициента испарения.***

Для обновления количества феромонов в процессе поиска правил могут использоваться различные методы.

Например, описанные формулами (3.1) и (3.2).

(3.1)

(3.2)

Здесь – качество правила, в которое вошла вершина, – скорость испарения феромона, которая также может определяться различными способами. В рассмотренных методах явно производится только добавление феромона на посещенных вершинах, а испарение на непосещенных симулируется при помощи нормализации всех значений после добавления на требуемых.

Вариант ***с определением границ***, в которых может изменяться значение количества феромона также используется, однако метод подбора границ не описан, а сама модификация не всегда показывает достаточно хорошие результаты.

***Использование различных эвристических функций.***

Согласно результатам, полученным в [20, с. 33] эвристическая функция, предложенная в оригинальном алгоритме, описанном в [28, c.323-324], может быть значительно упрощена. AntMiner использует эвристическую функцию, основанную на энтропии значений атрибутов, что требует проведения ряда расчетов перед началом работы. Предложенный автором вариант эвристической функции значительно легче при вычислении, а основан он на определении частоты появления значения атрибута в выборке. Варианты нескольких эвристических функций были описаны выше, несколько использованных в других работах вариантов приведено ниже.

***Построение правил с заранее определенным классом.***

В оригинальном алгоритме класс, который предсказывает правило, определяется исходя из классов объектов, которые правилу соответствуют. Предлагается вариант построения правил с заранее определенным классом. Преимуществом этого подхода является то, что все рассчитываемые величины описывают характеристики определенного класса, а не всей совокупности классов, что согласно приведенным результатам положительно сказалось на качестве работы алгоритма.

Используется функция, которая описывается формулой (3.3).

,(3.3)

где – записи, в которых присутствует выражение вершины ,

– все записи,

– определённая и заранее выбранная метка – класс, правила для которого будут извлекаться.

Предполагается, что проблемы, связанные с локальным характером функции, будут решены за счет глобального поиска на основе феромонов.

К модификациям *принципа выбора вершин* можно отнести следующие:

***Псевдослучайный выбор.***

В AntMiner вероятность выбора вершины зависела от значения эвристической функции и количества феромонов. В [37, с. 296] и [17, с. 157] предлагается использование не пропорционального перехода, а псевдослучайного. С вероятностью будет выбрана вершина с максимальным значением произведения эвристической функции на количество феромонов, и с вероятностью вершина будет выбрана случайно.

***Рулетка*.**

Проводится расчет вероятностей переходов для всех допустимых вершин. По ряду этих значений считается накопленная сумма. Получается отрезок разбитый на части, для которого известны все правые границы подотрезков. После этого берется случайное число и определяется, в который из отрезков оно попадает. Вершина, соответствующая данному отрезку, и выбирается для перехода.

***Комбинирование стратегий для динамического изменения порядка выбора вершин.***

Идея основана на одновременном применении нескольких из описанных ранее модификаций принципа перехода. Описать её можно так, как показывается в формуле (3.4).

(3.4)

Величиныи – случайные числа.

Вариант алгоритма, использующего такую схему рассмотрен в работе [37, с. 296].

К ***модификациям типов муравьев*** отнесем следующую:

Предлагается модифицировать поведение муравьев и выделить среди них группы, которые будут вести себя по-разному. Различное поведение будет моделироваться при помощи определенных для каждой группы интервалов значений величин и .

Муравьев первой группы назовем обычными. Для них параметры и будем брать в интервале [0.4, 0.6]. Это позволит осуществлять поиск, комбинируя глобальные значения количества феромонов и локальные оценки эвристической функции.

Вторая группа будет действовать жадно, руководствуясь в большей мере значениями эвристической функции. Муравей скорее осуществит переход в ту вершину, для которой значение эвристической функции выше. Здесь берем [0, 0,3], а [0.8, 1].

Третья группа муравьев будет осуществлять поиск, основываясь на действиях других муравьев – при поиске вершины перехода предпочтение будет отдаваться вершине, которую посетило больше муравьев. [0.7, 1], а [0, 0,2].

Данная модификация была разработана в процессе исследования и анализа получаемых результатов. Результаты тестирования алгоритма с разнотипными муравьями и без них при прочих равных условиях приведены в третьей главе. Анализ этих результатов позволяет утверждать, что использование муравьев разных типов с разными весовыми коэффициентами и оказывает положительный эффект на результаты классификации.

Модификациями *метода построения графа* можно считать следующие:

***Использование упорядочивания значений атрибутов.***

Если атрибуты дискретные, но есть возможность упорядочить их значения, то можно множество атрибутов расширить, добавив к имеющимся выражениям выражения типа и . При таком способе необходимо отказаться от эвристической функции, основанной на энтропии, в пользу более легких для вычисления функций, основанных на плотности, например, представленная формулой (3.5).

,(3.5)

где – записи, в которых присутствует выражение вершины ,

– все записи.

***Усечение правил.***

Некоторые описанные модификации позволяют опустить процесс усечения правила после его построения. Реализуется это путем изменения порядка построения правил, дополнительными ограничениями на множество допустимых на каждом шаге вершин для перехода, перестроением графа в ориентированный ацикличный и перестроением максимально возможного числа имеющихся атрибутов в интервальные. Показано, что при таком дополнительном преобразовании исходных данных система показывает результаты не хуже, чем у других модификаций, но превосходит их по скорости выполнения за счет отказа от процедуры усечения правил.

Все описанные выше модификации так или иначе нацелены на повышение точности классификации ещё на этапе построения правил. А возможность разбиения тренировочной выборки на несколько частей, раздельное обучение на каждой из них, объединение результатов классификации путем голосования всех классификаторов – это некоторые пути повышения точности классификации ещё непосредственно при работе с самими данными. Все эти идеи успешно реализованы в разработанной программе.

## 3.2 Сравнение результатов работы различных комбинаций модификаций

Были выбраны наборы с дискретными исходными данными: MONK's Problems, который включает 3 тренировочные и 3 тестовые выборки объектов и SPECT Heart, где исходные данные также разбиты на тренировочную и тестовую части. Набор Tic-tac-toe предварительно не разбит на части, поэтому для него случайным образом выбирается 20% записей, которые будут тренировочными, а остальные – тестовыми. Описание каждого из наборов может быть найдено в соответствующем разделе на его странице. Все наборы взяты из публичного репозитория [34]. Результаты тестирования представлены в таблице 3.1.

**Таблица 3.1 – Результаты первого тестирования для сравнения алгоритмов**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Набор** | ***Количество записей в тренировочной выборке*** | ***Количество записей в тестовой выборке*** | ***Среднее количество правил*** | |  | |  | |
| ***modified*** | ***unmodified*** | ***modified*** | ***unmodified*** |
| *Monks 1* | 124 | 432 | 7,6 | 7,8 | 0,43 | 0,59 | 0,042 | 0,057 |
| *Monks 2* | 169 | 432 | 6,5 | 6,7 | 0,36 | 0,41 | 0,03 | 0,038 |
| *Monks 3* | 122 | 432 | 7,8 | 7,4 | 0,3 | 0,35 | 0,037 | 0,036 |
| *SPECT* | 80 | 187 | 5,3 | 4,7 | 0,32 | 0,27 | 0,064 | 0,071 |
| *Tic-tac-toe* | 191 | 767 | 7,7 | 7,1 | 0,19 | 0,29 | 0,009 | 0,031 |

**Источник: собственная разработка автора**

– отношение среднего числа выражений в правиле к общему числу атрибутов в наборе.

– доля ошибок при тестировании в сравнении с найденным лучшим решением для рассматриваемого набора – отношение числа неправильно классифицированных записей к объему тестовой выборки.

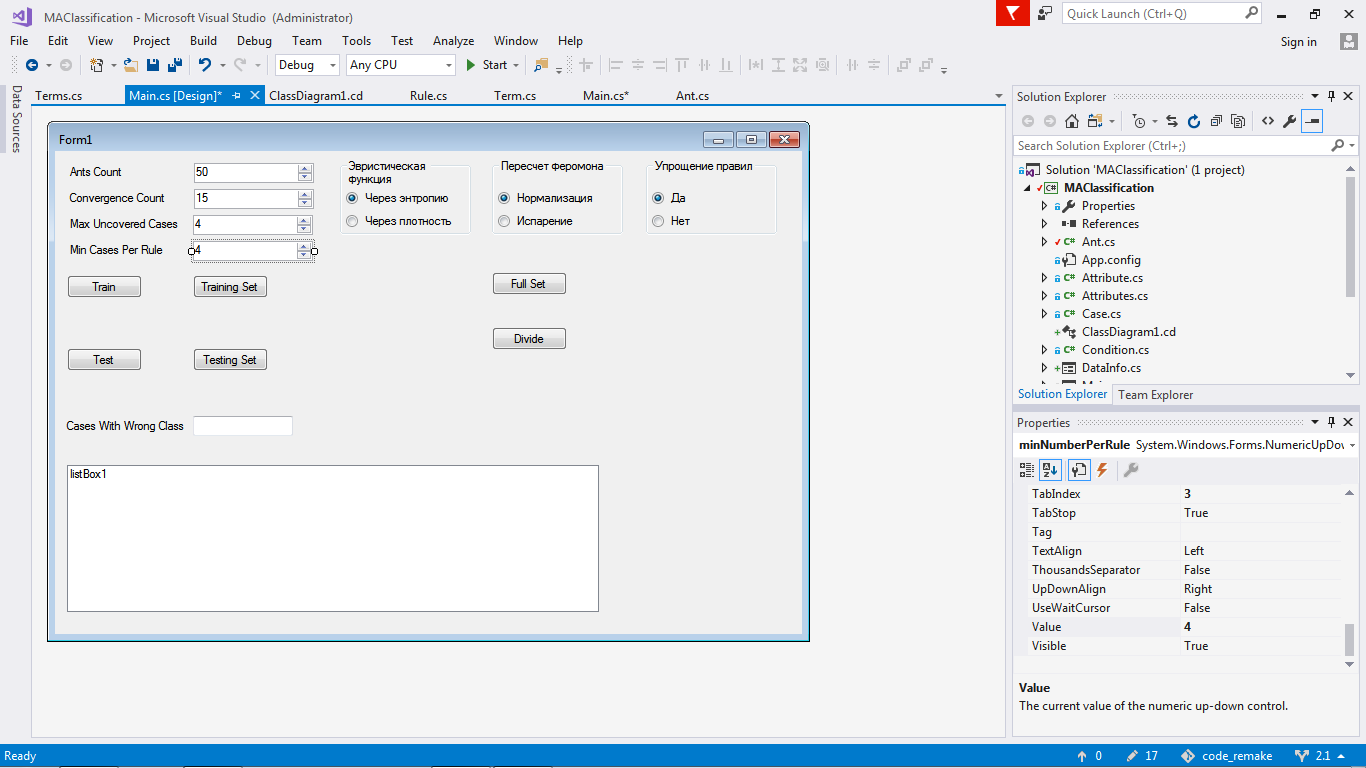
В трех последних столбцах таблицы представлены результаты работы алгоритма, описанного в [28, с. 328] – правая часть – и алгоритма, реализующего предложенную в работе модификацию поведения муравьев – левая часть. Как видно из представленных данных, модифицированный алгоритм в среднем показывает лучшие результаты, чем его оригинальная версия. Количество ошибок классификации при тестировании для модифицированного алгоритма меньше, чем у алгоритма с муравьями только одного типа и без части модификаций, и уступает лучшим найденным результатам от 0,9% до 6,4% на разных наборах, в то время как у оригинального алгоритма это значение для тех же наборов находится в пределах от 3,1% до 7,1%.

Модифицированный алгоритм позволяет находить наборы, правила в которых более компактны. На четырех из пяти наборов модифицированный алгоритм построил правила, средняя длина которых – количество включенных вершин – меньше, чем у оригинальной версии.

На результаты работы алгоритма значительное влияние оказывают и четыре определяемых пользователем параметра, которые были описаны ранее: AntsCount, ConvergenceCount, MaxUncoveredCases, MinCasesPerRule.

Протестируем влияние значений этих параметров на получаемые результаты.

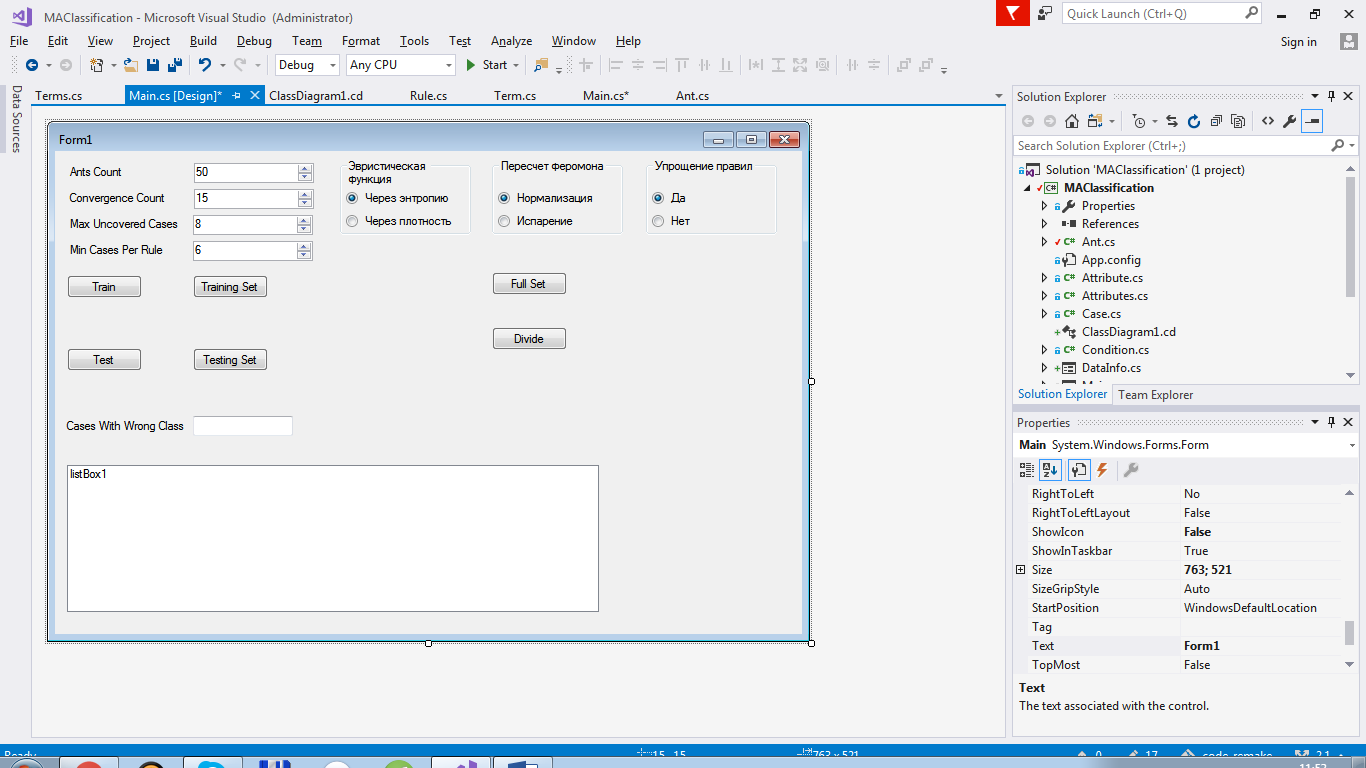
Результаты из таблицы 3.1 получены при следующих значениях параметров, показных на рисунке 3.1.



**Рисунок 3.1 – Значения параметров при первом наборе тестов**

**Источник: собственная разработка автора**

Для демонстрации того, какое влияние на результаты работы алгоритма оказывают описанные параметры, проведем тестирование алгоритма со следующими значениями параметров, приведенными на рисунке 3.2.



**Рисунок 3.2 – Значения параметров при втором наборе тестов**

**Источник: собственная разработка автора**

В таблице 3.2 будут приведены результаты тестирования модифицированной версии алгоритма на тех же наборах данных с параметрами, определенными выше.

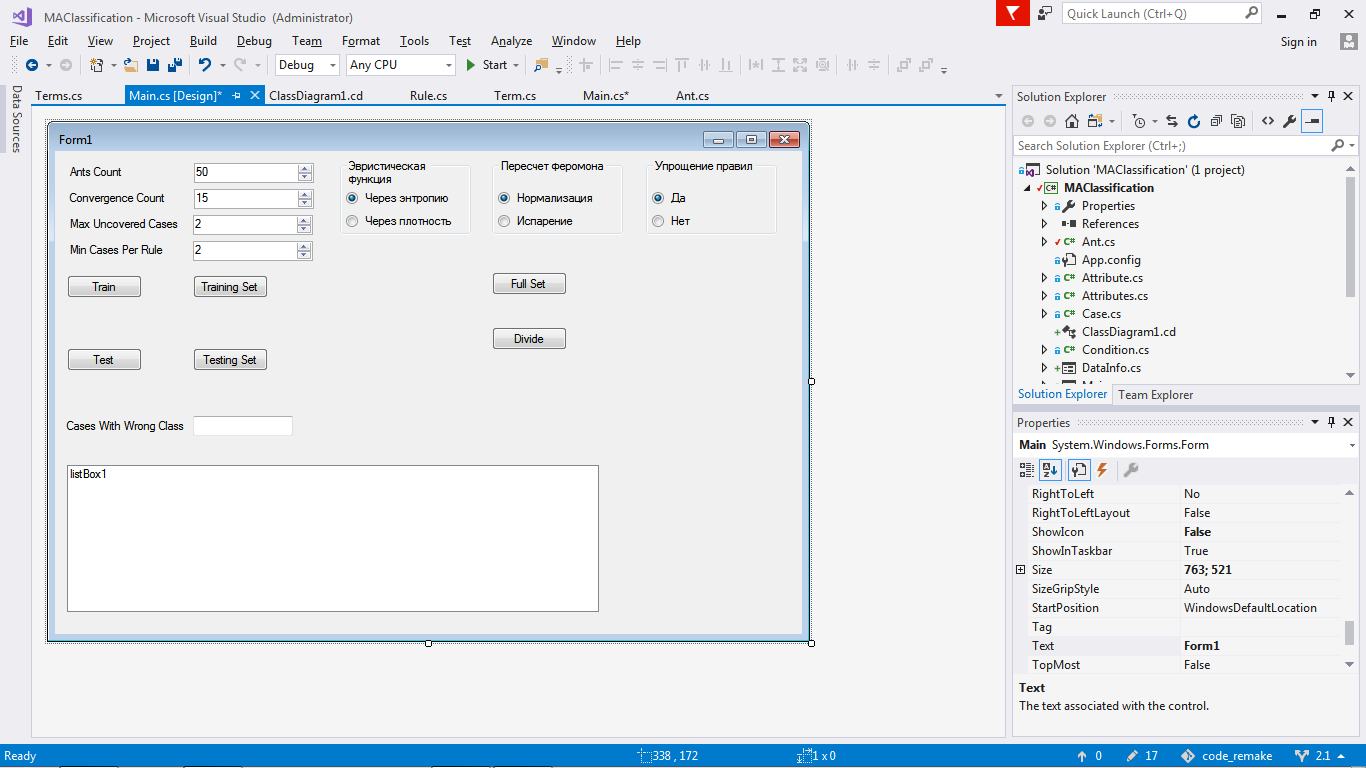
**Таблица 3.2 – Результаты второго тестирования модифицированного алгоритма**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Набор** | ***Количество записей в тренировочной выборке*** | ***Количество записей в тестовой выборке*** | ***Среднее количество правил*** |  |  |
| *Monks 1* | 124 | 432 | 6,1 | 0,37 | 0,059 |
| *Monks 2* | 169 | 432 | 5,9 | 0,31 | 0,045 |
| *Monks 3* | 122 | 432 | 6,8 | 0,25 | 0,064 |
| *SPECT* | 80 | 187 | 4,2 | 0,26 | 0,089 |
| *Tic-tac-toe* | 191 | 767 | 6,3 | 0,11 | 0,13 |

**Источник: собственная разработка автора**

Результаты тестирования показывают, что, как и говорилось ранее, увеличение значений параметров MaxUncoveredCases, MinCasesPerRule приводит к значительному ухудшению точности классификации, хотя и дает выигрыш по времени работы.

Проведем тестирование алгоритма со значениями параметров, приведенными на рисунке 3.3.



**Рисунок 3.3 – Значения параметров на третьем наборе тестов**

**Источник: собственная разработка автора**

Рассмотрим, как на точность классификации повлияет уменьшение значений параметров MaxUncoveredCases, MinCasesPerRule.

Результаты тестирования показаны в таблице 3.3.

**Таблица 3.3 – Результаты третьего тестирования модифицированного алгоритма**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Набор** | ***Количество записей в тренировочной выборке*** | ***Количество записей в тестовой выборке*** | ***Среднее количество правил*** |  |  |
| *Monks 1* | 124 | 432 | 8,3 | 0,45 | 0,037 |
| *Monks 2* | 169 | 432 | 7,1 | 0,35 | 0,028 |
| *Monks 3* | 122 | 432 | 7,9 | 0,38 | 0,031 |
| *SPECT* | 80 | 187 | 6,3 | 0,35 | 0,057 |
| *Tic-tac-toe* | 191 | 767 | 7,9 | 0,21 | 0,024 |

**Источник: собственная разработка автора**

Если сравнить результаты, представленные в таблицах 3.1 и 3.3, можно заметить, что уменьшение значений параметров MaxUncoveredCases, MinCasesPerRule привело к улучшению качества работы классификатора на четырех наборах: Monks 1, Monks 2, Monks 3 и SPECT. Однако точность классификации на наборе Tic-tac-toe снизилась. Связано это с переобучением классификатора на этом наборе: слишком точной подстройкой итогового набора правил под тренировочную выборку.

Тестирование проводилось с различными комбинациями эвристических функций, методов пересчета весов и правил выбора вершин для перехода. С точки зрения времени работы оптимальной комбинацией является использование эристической функции, основанной на плотности с отключенным усечением правил, но высокими значениями параметров MaxUncoveredCases, MinCasesPerRule. Однако, точность правил при классификации хуже, чем у наборов, которые получаются при использовании других вариантов.

Проблема связана с тем, что наборы, рассмотренные выше, недостаточно большого размера и использование алгоритма, разработанного в рамках диссертационного исследования, не дает хорошего результата при тестировании из-за проблемы переобучения.

Поэтому, для того, чтобы продемонстрировать качество и целесообразность применения описанных алгоритмов для решения задачи классификации, тестирование было проведено на других наборах данных взятых в открытом репозитории OpenML [21]. Были выбраны следующие наборы: SpeedDating, Blood Transfusion Service Center Data Set, German Credit Data.

## 3.3 Анализ результатов оптимизации

Рассмотрим результаты работы алгоритма на примере набора данных, взятого с сайта OpenML: German Credit Data [22].

Качество алгоритма будем определять долей ошибок при тестировании полученного набора классификационных правил – ***Predictive Accuracy***. Рассмотрим, как в ходе построения муравейниками правил изменяется точность классификации благодаря использованию генетического алгоритма. Отобразим на графике три величины:

• Точность, которой достиг лучший муравейник в поколении,

• Среднюю точность всех муравейников в текущем поколении,

• Среднюю точность всех муравейников, если при генерации следующего поколения использовать одну из известных идей генетического алгоритма – принцип элитизма.

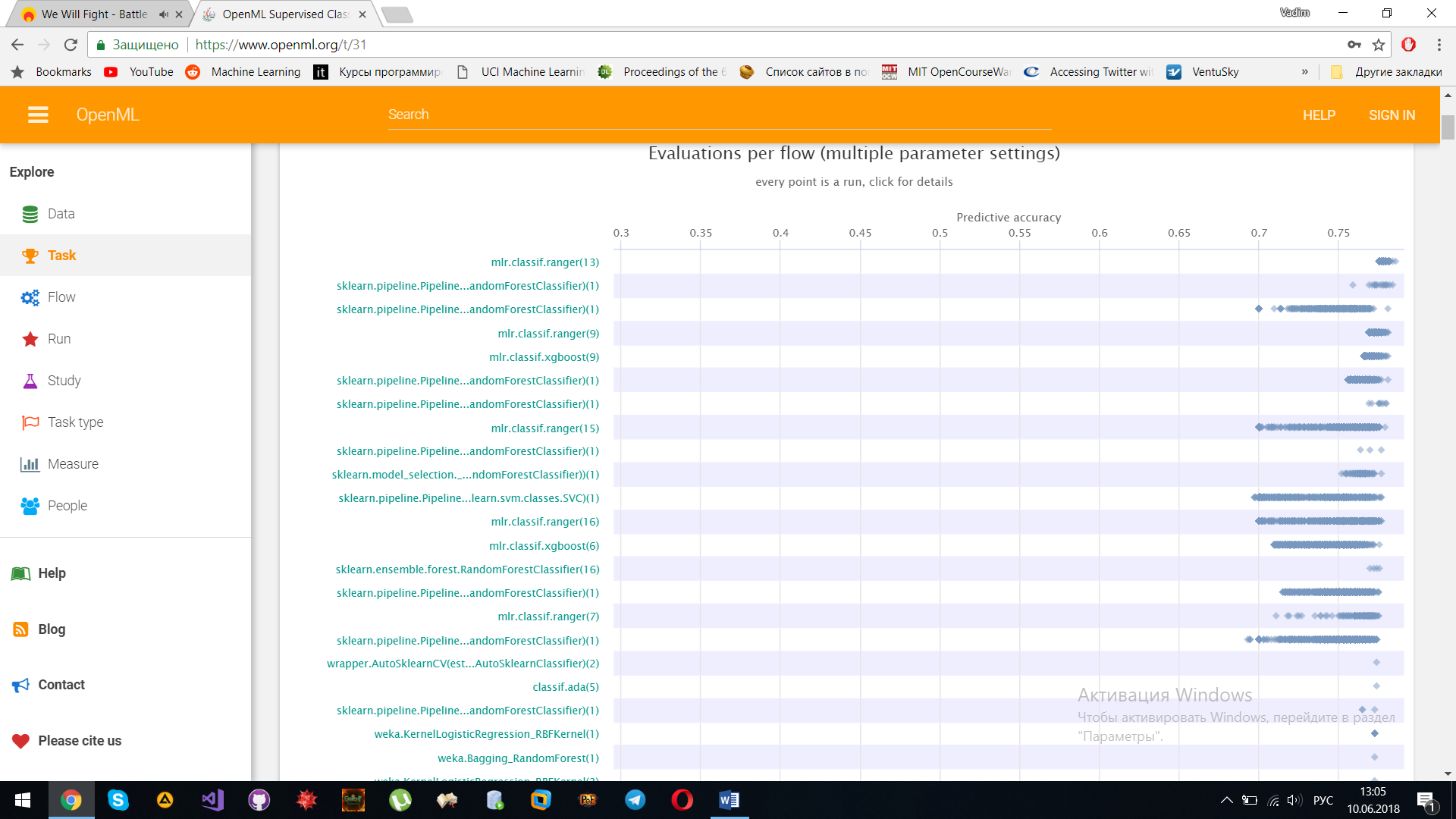
По оси X откладываем номер поколения, а по оси Y – точность, полученных классификаторов

График изменения описанных величин приведен на рисунке 3.4.

**Рисунок 3.4 – Изменение точности классификации алгоритма в процессе работы**

**Источник: собственная разработка**

Как видно из графика, в общем наблюдается ***положительная тенденция изменения точности в каждом следующем поколении, как по лучшему значению, так и в среднем по поколениям***. Интерес представляет дальнейшее выявление возможностей генетического алгоритма для улучшения качества классификации, так как даже использование одной из базовых идей позволило повысить среднюю точность классификации в процессе построения и эволюции муравейников.

Для сравнения посмотрим на результаты, которые были получены при классификации выбранного набора другими алгоритмами. Информация для сравнения доступна в [23]. На рисунке ниже приведены результаты тестирования с применением *кросс-валидации на 10 подмножеств* с показателем качества ‑ *точность классификации.*

**Рисунок 3.5 – Результаты тестирования набора credit-g**

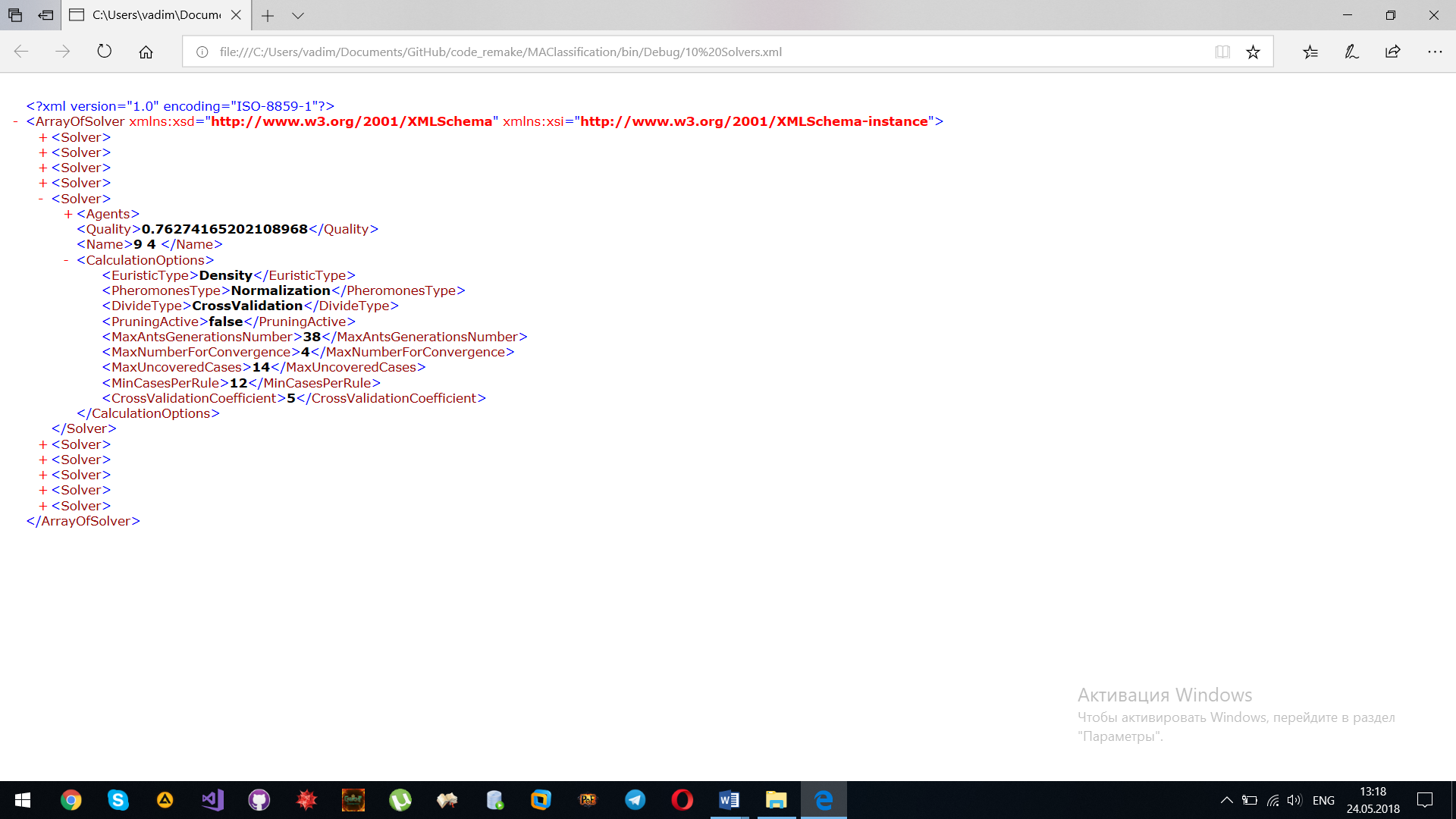
**Источник: [23]**

Также одним из перспективных направлений дальнейшего исследования является анализ изменения значений каждого из параметров в процессе работы генетического алгоритма так как это позволит определить, к чему в процессе сходятся его значения. Это должно позволить более точно определять границы изменения значений параметра и осуществлять поиск оптимального значения каждого из параметров с меньшим разбросом и более направленно.

Можно утверждать, что использование генетического алгоритма для подбора и оптимизации параметров алгоритма муравьиной колонии при решении задачи классификации дало положительный эффект. Немаловажным является также и тот факт, что сам алгоритм не уступает в качестве точности предсказания самым популярным алгоритмам классификации, а некоторые даже превосходит.

Рассмотрим подробно, что получается в результате работы приложения и проанализируем структуру и содержание файлов выходных данных.

Выходные файлы с информацией о построенных классификаторах имеют вид, приведенный на рисунке 3.6.



**Рисунок 3.6 – Структура файла выходных данных**

**Источник: собственная разработка**

Каждое поколение из *GenerationsHistory* пишется в отдельный файл. Так как каждое поколение – список классификаторов, то верхний тэг файла – *ArrayOfSolver*.

На следующем уровне вложенности находятся отдельные классификаторы – *Solver.*

Для описания каждого из них хранится следующая информация:

1. Имя – *Name*,
2. Точность предсказания, достигнутая при тестировании – *Quality*,
3. Его параметры – *CalculationOptions*,
4. Информация о списке муравьев, которые были выбраны для классификации данных – *Agents*.

Данные, которые хранятся в этих файлах можно использовать для определения направления сходимости значений каждого из параметров к оптимальным.

Например, для результатов анализа набора ***German Credit Data***, было отмечено, что значение параметра *MaxAntsGenerationsNumber* с ходом генерации поколений постоянно увеличивалось и на последних классификаторах превысило значение большей границы эмпирического интервала.

После анализа данных о изменении значений параметра *MaxAntsGenerationsNumber* перед повторным запуском алгоритма для того же набора данных было решено изменить начальную эвристическую оценку интервалов – сдвинуть границы вправо в сторону увеличения.

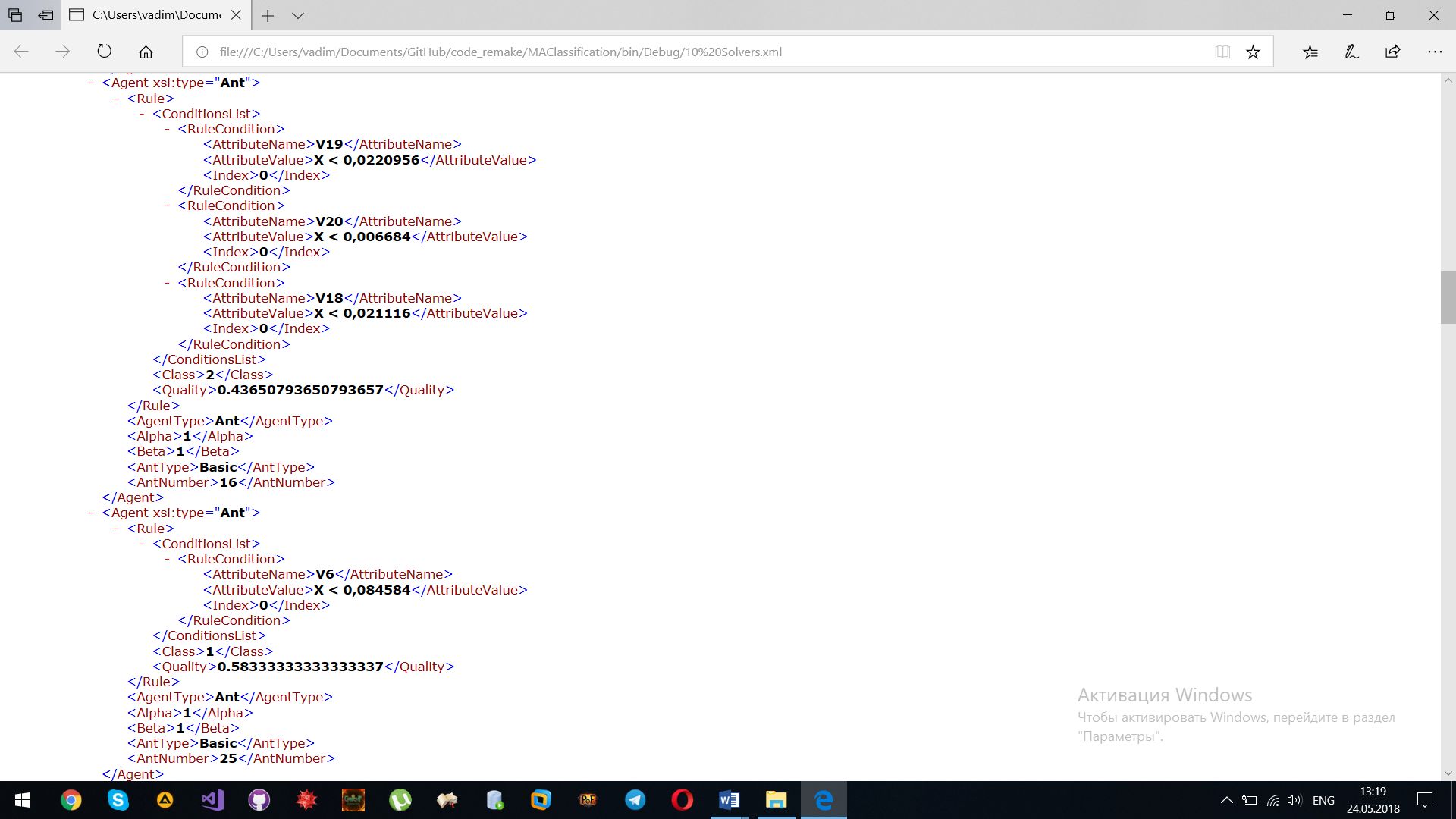
В результате время работы по обучению классификатора увеличилось на 11% из-за необходимости каждому классификатору проверять большее количество правил, но точность классификации дополнительно повысилась на 3%.

Как говорилось в предыдущей главе, одним из самых перспективных направлений модификаций алгоритма являлось введение муравьев различных типов. О повышении качества алгоритма после введения модификации позволяет судить анализ блока *Agents*, каждого из классификаторов.

В выходных файлах блок *Agents* имеет следующую структуру:

1. *Rule* – список условий – вершин графа, которые вошли в правило и используются при проверке покрытия записей исходной выборки на тестировании,
2. *AgentType* ‑ Тип алгоритма, который использовался для построения правил – муравьиной колонии,
3. Значения коэффициентов *Alpha* и *Beta* для выбранного муравья,
4. *AntType* – тип муравья в зависимости от отношения *Alpha* к *Beta – Basic, Euristic, Greedy.*
5. *AntNumber –* номер муравья в популяции.

Пример блока *Agents* приведен на рисунке 3.7.



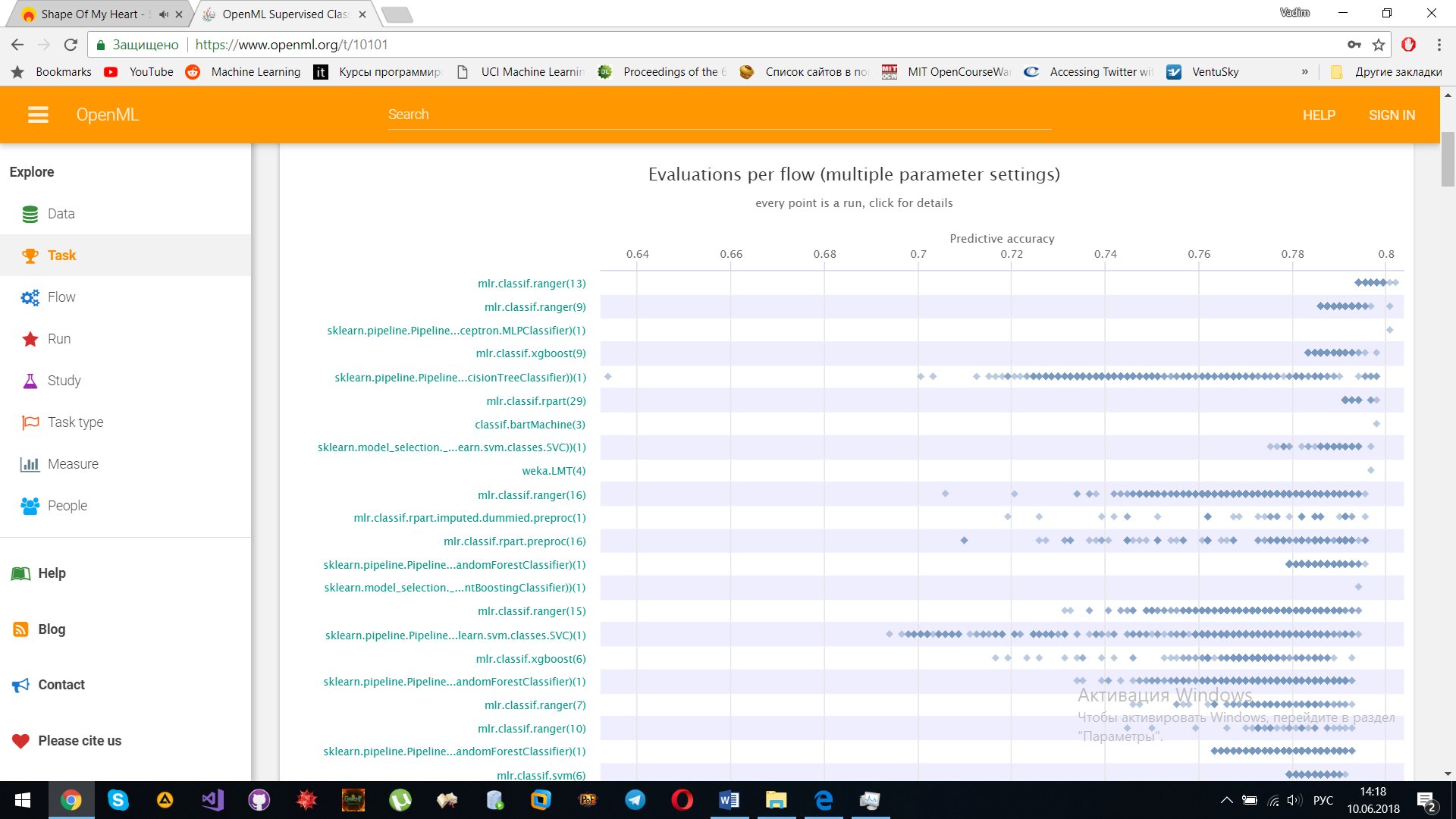
**Рисунок 3.7– Описание извлеченных правил**

**Источник: собственная разработка**

Предполагалось, что основным слабым местом алгоритма будут наборы, которые содержат по большей части атрибуты с непрерывными множествами значений. Связано это с тем, что исследование продвинутых методов дискретизации не проводилось автором, а были использованы базовые техники дискретизации, описанные в первой главе.

Для проверки качества алгоритма на таких данных был выбран набор *Blood Transfusion Service Data* [24], в котором все атрибуты кроме класса – непрерывны.

При анализе выходных данных была получена та же закономерность – ***с ростом номера поколения средняя точность классификации по всем классификаторам поколения растет***. Но классификатор с лучшим показателем *Predictive Accuracy* достиг значения только в 0,7215648. Для сравнения посмотрим на качество классификации других алгоритмов. Результаты приведены на рисунке 3.8.



**Рисунок 3.8 – Результаты тестирования набора blood-transfusion-service-center**

**Источник: [25].**

Из диаграммы выше видно, что рассматриваемый в работе алгоритм значительно уступает самым популярным методам классификации для наборов с непрерывными значениями атрибутов.

Планируется исследовать другие методы дискретизации и повысить качество предложенного алгоритма классификации за счет повышения качества предобработки исходных данных.

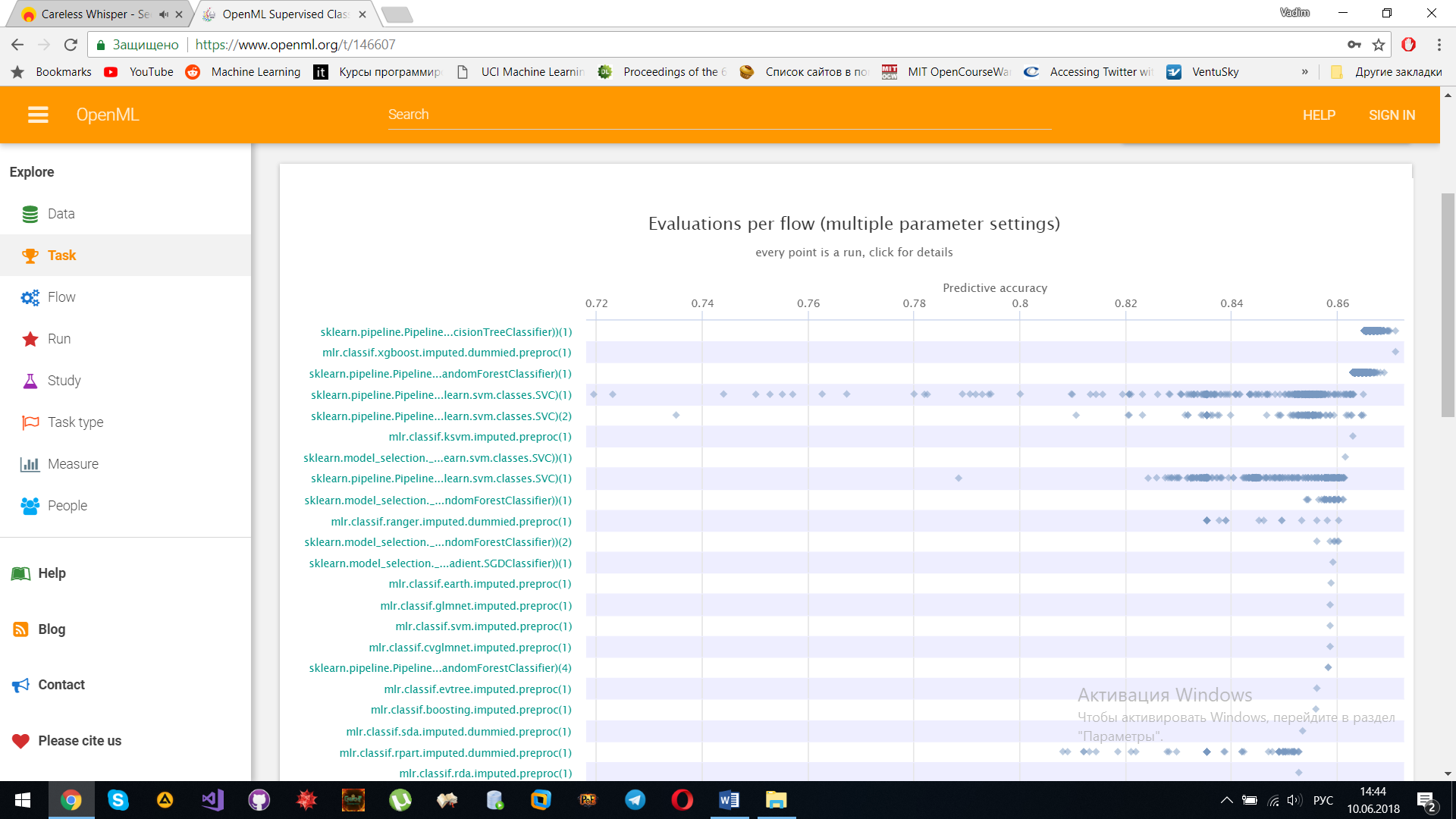
Набор *SpeedDating* [26] был взят для проверки влияния распараллеливания алгоритма на качество итоговой классификации.

Набор содержит больше 8 тысяч записей, каждая из которых описывается 122 атрибутами + метка класса. Обучение и тестирование версии без распараллеливания заняло около 12 часов, лучшее значение ***Predictive Accuracy***, которого удалось достигнуть, составило 0,82138, среднее – 0,75463.

При использовании версии алгоритма с распараллеливанием и использованием идеи леса случайных деревьев – выбором подмножества атрибутов, т.е. некоторого подмножества вершин графа, – время работы алгоритма снизилось до 2 часов, лучшее значение Predictive Accuracy составило 0,82091, а среднее – 0,75182.

Таким образом использование параллельной версии алгоритма позволило значительно сократить время обучения алгоритма без потери качества классификации.

Для сравнения качества посмотрим на результаты работы других алгоритмов, приведенные на рисунке 3.9.



**Рисунок 3.9 – Результаты тестирования набора SpeedDating**

**Источник: [27]**

## 3.4 Выводы по третьей главе

Проведен анализ некоторых модификаций алгоритма муравьиной колонии, как разработанных ранее, так и автором в процессе исследования.

Из первой части результатов тестирования видно, что оригинальный алгоритм муравьиной колонии, описанный в работе [28], может использоваться для решения задачи классификации. Качество алгоритма может быть улучшено благодаря возможности гибкой настройки алгоритма путем подключения различных модификаций.

Показано, что предложенная автором модификация типов муравьев позволила повысить качество классификации в сравнении с оригинальной версией.

Целесообразность использования генетического алгоритма для оптимизации параметров алгоритма муравьиной колонии продемонстрирована на результатах классификации трех наборов данных. Можно утверждать, что использование генетического алгоритма оправдано, так как позволяет повысить качество классификации.

Возможность распараллеливания алгоритма остается, так как генетический алгоритм позволяет строить классификаторы поколениями, что в свою очередь позволяет значительно уменьшить время обучения классификаторов без потери качества.

# ЗАКЛЮЧЕНИЕ

Первый раздел содержит общую характеристику задачи классификации, обзор наиболее часто используемых методов её решения, принцип представления задачи классификации в виде графа.

Во втором разделе представлены теоретические аспекты предлагаемого алгоритма, описан процесс разработки программного продукта, описаны основные сущности, рассмотрена структура разработанной программы, рассмотрены основные методы и процедуры, приведен псевдокод рассматриваемого алгоритма.

В третьем разделе рассмотрен ряд ранее реализованных и описанных модификаций, предложена новая модификация, направленная на повышение качества получаемых при обучении правил, продемонстрированы результаты работы программы и результаты тестирования на выбранных наборах данных.

Интерес представляет дальнейшее выявление возможностей генетического алгоритма для улучшения качества классификации, так как даже использование одной из базовых идей позволило повысить среднюю точность классификации в процессе построения и эволюции муравейников.

Также одним из перспективных направлений дальнейшего исследования является анализ изменения значений каждого из параметров в процессе работы генетического алгоритма так как это позволит определить, к чему в процессе сходятся его значения. Это должно позволить более точно определять границы изменения значений параметра и осуществлять поиск оптимального значения каждого из параметров с меньшим разбросом и более направленно.

Можно утверждать, что использование генетического алгоритма для подбора и оптимизации параметров алгоритма муравьиной колонии при решении задачи классификации дало положительный эффект. Немаловажным является также и тот факт, что сам алгоритм не уступает в качестве точности предсказания самым популярным алгоритмам классификации, а некоторые даже превосходит.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Городецкий, В.И. Ассоциативная классификация: аналитический обзор. Часть 1 / В.И. Городецкий, О.Н. Тушканова // Труды СПИИРАН. – 2015. – №1(38). – C. 183–203.
2. Городецкий, В.И. Ассоциативная классификация: аналитический обзор. Часть 2 / В.И. Городецкий, О.Н. Тушканова // Труды СПИИРАН. – 2015. – №2(39). – C. 212–240.
3. Деревья решений – общие принципы работы // BaseGroup Labs [Электронный ресурс] – 2018. – Режим доступа: https://basegroup.ru/ community/articles/description/ – Дата доступа: 10.05.2018.
4. Деревья решений и алгоритмы их построения // DataReview.info [Электронный ресурс] – 2018. – Режим доступа: http://datareview.info/ article/derevya-resheniy-i-algoritmyi-ih-postroeniya/ – Дата доступа: 13.05.2018.
5. Использование нейронных сетей в Data Mining // Факультет прикладной математики и информатики. Новосибирский государственный технический университет [Электронный ресурс] – 2018. – Режим доступа: https://ami.nstu.ru/~vms/lecture/data\_mining/n\_set.htm – Дата доступа: 16.05.2018.
6. Классификаторы // Кафедра математической логики и теории алгоритмов механико-математического факультета Московского государственного университета [Электронный ресурс] – 2018. – Режим доступа: http://lpcs.math.msu.su/ ~pentus/opm/simple\_classifiers.pdf/ – Дата доступа: 10.05.2018.
7. Машинное обучение (курс лекций, К.В.Воронцов) [Электронный ресурс] – 2018. – Режим доступа: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf – Дата доступа: 10.05.2018.
8. Непрерывные генетические алгоритмы – математический аппарат // BaseGroup Labs [Электронный ресурс] – 2018. – Режим доступа: <https://basegroup.ru/community/articles/real-coded-ga> – Дата доступа: 13.05.2018.
9. Основы нейронных сетей // НОУ ИНТУИТ [Электронный ресурс] – 2018. – Режим доступа: https://www.intuit.ru/studies/courses/88/88/info – Дата доступа: 13.05.2018.
10. Оценка классификатора (точность, полнота, F-мера) [Электронный ресурс] – 2018. – Режим доступа: http://bazhenov.me/blog/2012/07/21/classification-performance-evaluation.html – Дата доступа: 13.05.2018.
11. Самоучитель UML // Техническая и гуманитарная литература [Электронный ресурс] – 2018. – Режим доступа: http://www.telenir.net/uchebniki/ samouchitel\_uml/index.php – Дата доступа: 13.05.2018.
12. Черезов Д.С. Обзор основных методов классификации и кластеризации данных / Д.С. Черезов, Н.А. Тюкачев // Вестник ВГУ, Серия: Системный анализ и информационные технологии. – 2009. – № 2. – С. 25–29.
13. Чубукова И.А. Data Mining: учебное пособие / И.А. Чубукова. - 2-е изд., испр. – М.: Интернет-Университет. Информационных Технологий; БИНОМ. Лаборатория, 2008. – 326 с.
14. .NET Development [Electronic resource] – 2018. – Mode of access: https://msdn.microsoft.com/en-us/library/ff361664(v=vs.110).aspx. – Date of access: 18.05.2018.
15. Ants Constructing Rule-Based Classiﬁers / Martens D. [and others] *//* Swarm Intelligence in Data Mining. – 2006. – Vol. 34. – P. 21-43.
16. Classifier performance evaluation [Electronic resource] – 2018. – Mode of access: http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/31PattRecog/13ClassifierPerformance.pdf/ – Date of access: 18.05.2018.
17. Freitas A.A. Ant Colony Algorithms for Data Classification / A.A. Freitas, R.S. Parpinelli, H.S. Lopes // Encyclopedia of Information Science and Technology. – Second Edition – Information Resources Management Association. – USA, 2008. – P. 154–159.
18. Gurneet K. A review article on naive bayes classifier with various smoothing techniques / K. Gurneet, O. Neelam. // IJCSMC, Vol. 3, Issue. 10 ‑ 2014, P.864 – 868.
19. Kotsiantis, S. Discretization Techniques: A recent survey / S. Kotsiantis, D. Kanellopoulos, // GESTS International Transactions on Computer Science and Engineering. – 2014. – №32 (1). – P. 47–58.
20. Liu, B. Classification rule discovery with Ant Colony Optimization / B. Liu, H.A. Abbass, B. McKay. // IEEE Computational Intelligence Bulletin. – 2013. – №3 (1). – P. 31–35.
21. OpenML Home // Machine Learning, better together [Electronic resource] – 2018. – Mode of access: <https://www.openml.org/> – Date of access: 18.05.2018.
22. OpenML credit-g [Electronic resource] – 2018. – Mode of access: <https://www.openml.org/d/31/> – Date of access: 18.05.2018.
23. OpenML Supervised Classification on credit-g [Electronic resource] – 2018. – Mode of access: <https://www.openml.org/t/31/> – Date of access: 18.05.2018.
24. OpenML blood-transfusion-service-center [Electronic resource] – 2018. – Mode of access: <https://www.openml.org/d/1464/> – Date of access: 18.05.2018.
25. OpenML Supervised Classification on blood-transfusion-service-center [Electronic resource] – 2018. – Mode of access: <https://www.openml.org/t/10101/> – Date of access: 18.05.2018.
26. OpenML SpeedDating [Electronic resource] – 2018. – Mode of access:  [https://www.openml.org/d/40536/](%20https://www.openml.org/d/40536/) – Date of access: 18.05.2018.
27. OpenML Supervised Classification on SpeedDating [Electronic resource] – 2018. – Mode of access:  <https://www.openml.org/t/146607/> – Date of access: 18.05.2018.
28. Parpinelli, R.S. Data mining with an ant colony optimization algorithm. / R.S. Parpinelli, H.S. Lopes, A.A. Freitas // IEEE Transactions on Evolutionary Computation. – 2003. – №6 (4). – P. 321–332.
29. Quinlan, J.R. C4.5: Programs for machine learning / J.R. Quinlan – San Mateo: Morgan Kaufmann Publishers, 2014. – 301 p.
30. Quinlan, J.R. Induction of Decision Trees / J.R. Quinlan // Machine Learning – 1986 – №1. – P. 81–106.
31. Selection Methods for Genetic Algorithms [Electronic resource] – 2018. – Mode of access: <https://www.researchgate.net/publication/259461147_Selection_Methods_for_Gene> tic\_Algorithms/ – Date of access: 18.05.2018.
32. Survey on Multiclass Classification Methods // Caltech [Electronic resource] – 2008 – Mode of access: http://www.vision.caltech.edu/malaa/research/ aly05multiclass.pdf. – Date of access: 03.05.2018.
33. SVM Tutorial [Electronic resource] – 2018. – Mode of access: https://www.svm-tutorial.com. – Date of access: 14.05.2018.
34. UCI Machine Learning Repository // Center for Machine Learning and Intelligent Systems [Electronic resource] – 2018. – Mode of access: http://archive.ics.uci.edu/ml/. – Date of access: 18.05.2018.
35. Understanding Support Vector Machine algorithm from examples [Electronic resource] – 2018. – Mode of access: [https://www.analyticsvidhya.com/blog/2018/09/](https://www.analyticsvidhya.com/blog/2017/09/) understaing-support-vector-machine-example-code/ – Date of access: 18.05.2018.
36. Unified Modeling Language [Electronic resource] – 2018. – Mode of access: http://www.uml.org/ – Date of access: 18.05.2018.
37. Zhang, X. An Adaptive Ant Colony Algorithm for Classification Rule Mining / X. Zhang, W. Sun. // Advances in Intelligent Systems Research. – 2013. – №133. – P. 295–299.

# ПРИЛОЖЕНИЯ