МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Анализ и реализация алгоритмов построения дерева решений

Студент Д. Ю. Евстигнеева
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., доцент, Н.А. Сосина
(ученая степень, звание, И.О. Фамилия)

Консультант Е.В. Косс
(ученая степень, звание, И.О. Фамилия)

Тольятти 2022

Аннотация

Тема бакалаврской работы: «Анализ и реализация алгоритмов построения дерева решений».

Актуальность данной работы в том, что при решении множества практических задач, существует необходимость в классификации и регрессии.

Цель работы: анализ и реализация алгоритма ID3, алгоритма C4.5 и алгоритма CART.

Объект исследования: задачи оптимизации, решаемые на основе метода дерева решений.

Предмет исследования: следующие алгоритмы генерации дерева решений:

- алгоритм ID3,
- алгоритм C4.5,
- алгоритм CART.

Полученные результаты: реализованы три алгоритма построения дерева решений (алгоритм ID3, алгоритм C4.5, алгоритм CART) и проанализированы их результаты.

Структура выпускной квалификационной работы представлена введением, тремя главами, заключением, списком литературы.

Во введении описывается актуальность, цели и задачи данного исследования.

В первой главе работы рассматриваются история возникновения и анализ деревьев решений.

Во второй главе описываются методы и алгоритмы решения задач с помощью дерева решений.

В третьей главе описывается процесс реализации алгоритмов построения дерева решений и их тестирование. На основе результатов тестирования формулируется вывод.

Бакалаврская работа выполнена на 48 страницах, содержит 12 рисунков и 2 таблицы.

Abstract

The title of the graduation work is «Analysis and implementation of algorithms for building a decision tree».

The relevance of this work is that when solving many practical problems, there is a need for classification and regression.

The aim of this work is the analysis and implementation of the ID3 algorithm, the C4.5 algorithm and the CART algorithm.

The object of the study is the optimization problems solved on the basis of the decision tree method.

The subject of the study is the following algorithms for generating a decision tree:

- ID3 algorithm,
- Algorithm C4.5,
- CART algorithm.

The result of the graduation work is three algorithms for building a decision tree (ID3 algorithm, C4.5 algorithm, CART algorithm) are implemented and their results are analyzed.

The graduation work is divided into several logically connected parts which include theoretical information, implementation of algorithms and, comparative analysis of algorithms.

The introduction describes the relevance, goals and objectives of this study.

The first chapter of the paper discusses the history of the emergence and analysis of decision trees.

The second chapter describes the methods and algorithms for solving problems using a decision tree.

The third chapter describes the process of implementing algorithms for building a decision tree and testing them. Based on the test results, a conclusion is drawn.

The graduation work consists of an explanatory note on 48 pages, introduction, three parts including 12 illustrations, 2 tables, the list of 27 references including 5 foreign sources.

Оглавление

Введение

В выпускной квалификационной работе рассматривается процесс анализа и реализации алгоритмов построения дерева решений.

Актуальность работы заключается в том, что методы деревьев решений популярны в аналитике данных и машинном обучении с практическим применением в различных секторах, от здравоохранения до финансов и технологий. Деревья принятия решений проще, чем нейронные сети, в связи с тем, что их правила формируются на естественном языке. Эти правила генерируются за счет обобщения множества отдельных наблюдений, которые служат обучающими примерами и описывают предметную область. Поэтому их называют индуктивными правилами, а сам процесс обучения — индукцией деревьев решений.

Объектом выпускной квалификационной работы являются задачи оптимизации, решаемые на основе метода дерева решений.

Предметом выпускной квалификационной работы являются следующие алгоритмы генерации дерева решений:

- алгоритм ID3,
- алгоритм C4.5,
- алгоритм CART.

Целью работы является анализ и реализация алгоритма ID3, алгоритма C4.5 и алгоритма CART.

Для достижения поставленных целей необходимо решить следующие задачи:

- проанализировать алгоритмы построения дерева решений для исследования их эффективности,
- выбрать технологии для проектирования алгоритмов построения дерева решений,
- исследовать математические модели алгоритмов построения дерева решений,

- реализовать алгоритмы построения дерева решений,
- протестировать реализованные алгоритмы построения дерева решений,
- проанализировать эффективность алгоритмов построения дерева решений.

Результатом работы является реализация алгоритма ID3, алгоритма C4.5 и алгоритма CART, а также анализ их эффективности.

В первой главе рассматривается история возникновения метода «дерева решений», а также задачи, решаемые с помощью данного метода.

Во второй главе рассматриваются методы и алгоритмы решения задач с помощью дерева решений.

В третьей главе рассматриваются современные архитектурные решения; реализуются и тестируются алгоритмы построения дерева решений с использованием выбранной ранее технологии.

1 История возникновения и анализ дерева решений

1.1 История возникновения метода

Деревья принятия решений начали использовать много веков назад. Деревья были символически важны для большинства древних культур, им часто поклонялись, и они часто присутствовали в искусстве. Их связь с бессмертием и их разветвленная структура сделали их естественными опорами для генеалогий, показывающих, например, родословную значимых личностей или королевской власти. Пример подобного дерева представлен на рисунке 1.

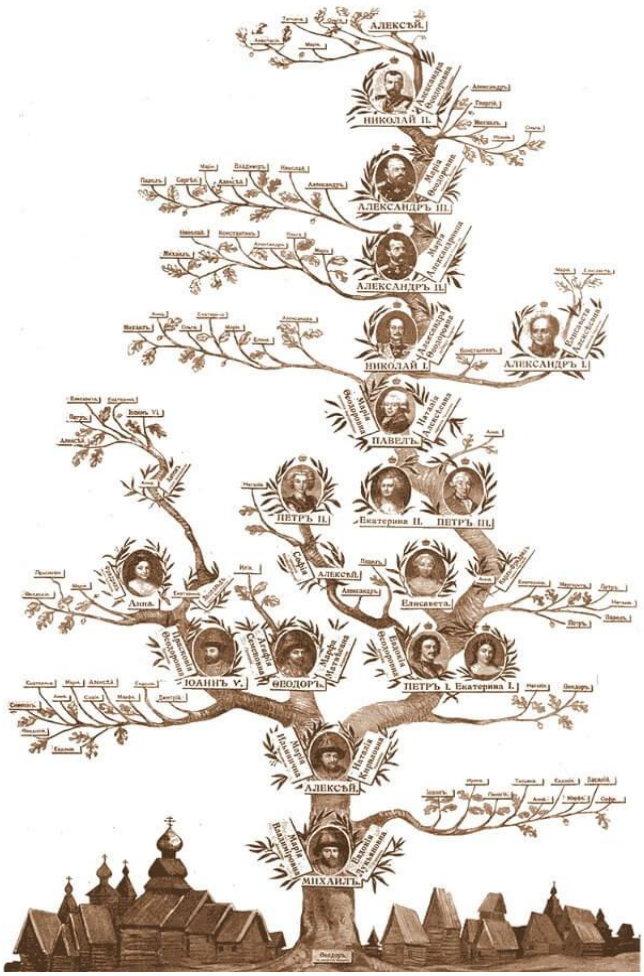

Рисунок 1 - Генеалогическое древо рода Романовых

Деревья визуально устанавливали родословную и, что не менее важно в средневековых обществах, помогали контролировать инбридинг, показывая, насколько близки люди были связаны с потенциальным супругом.

В начале 1990-х годов 14 ученых-компьютерщиков из Мэрилендского университета делили между собой 80-мегабайтный жесткий диск. Диск часто был перегружен, а неиспользуемые файлы занимали место в запущенных подкаталогах. Найти что-либо было все равно, что вслепую протянуть руку по всем ветвям разросшегося дерева.

Профессором кафедры Бен Шнайдерман был предложен алгоритм из шести строк, который визуализировал диск в виде прямоугольника. Вертикальные деления разбивают прямоугольник на более мелкие, представляющие каталоги, которые затем разделяются по горизонтали для отображения подкаталогов. Каждый из самых маленьких прямоугольников соответствовал мегабайту дискового пространства, поэтому свободное место было видно с первого взгляда.

Он назвал свое изобретение «древовидной картой», и оно было принято компьютерными лабораториями по всему миру. Пример древовидной карты изображен на рисунке 2.

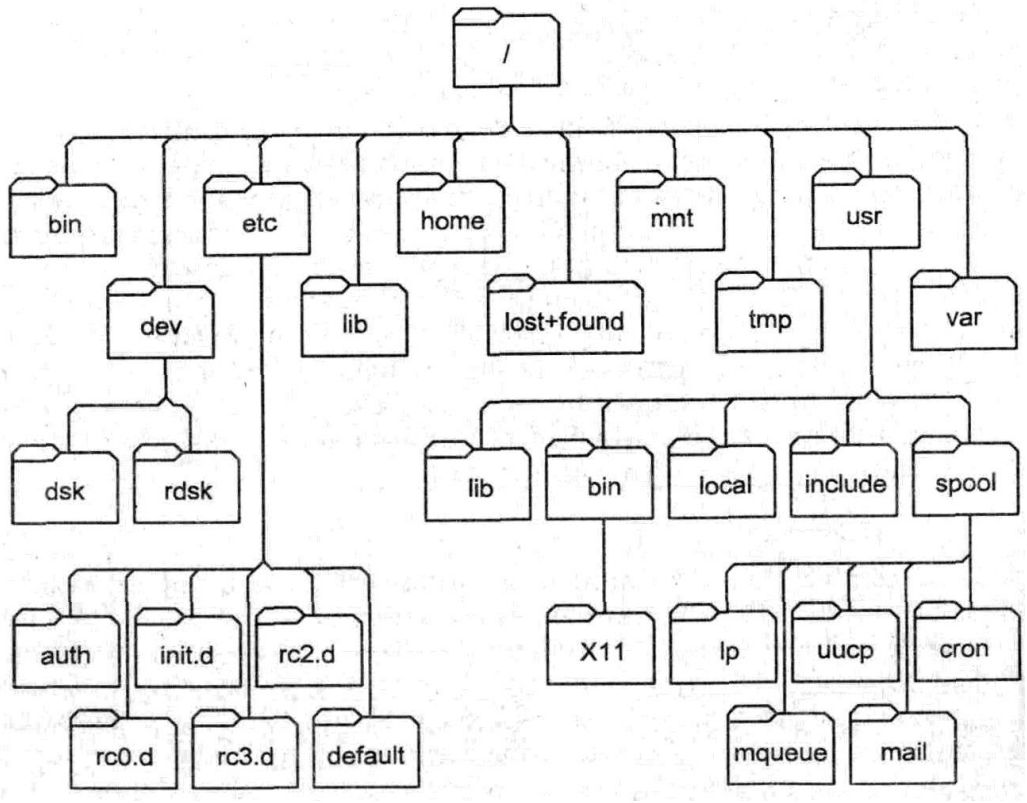

Рисунок 2 – Структура файловой системы в виде дерева

Вскоре он нашел и другие применения, например, в интерактивных графиках акций и долей, популярных и сегодня.

Эти иерархические древовидные карты «олицетворяют недавний рост визуализации информации», — пишет Мануэль Лима в книге «Книга деревьев: визуализация ветвей знаний». И по мере того, как большие данные поглощают лаборатории и жизни, потребность в таких мощных визуализациях будет только возрастать.

Деревья решений до сих являются мощным и популярным инструментом. Они обычно используются аналитиками данных для проведения прогнозного анализа (например, для разработки операционных стратегий в компаниях).

Также они являются популярным инструментом для машинного обучения и искусственного интеллекта, где они используются в качестве

обучающих алгоритмов для обучения с учителем (то есть категоризации данных на основе различных тестов, таких как классификаторы «да» или «нет»).

В широком смысле деревья решений используются в самых разных отраслях для решения многих типов проблем. Из-за своей гибкости они используются в различных секторах, от технологий и здравоохранения до финансового планирования.

1.2 Анализ дерева решений.

«Анализ дерева решений — это общий инструмент прогнозного моделирования, который имеет приложения, охватывающие ряд различных областей. Как правило, деревья решений строятся с помощью алгоритмического подхода, который определяет способы разделения набора данных на основе различных условий. Это один из наиболее широко используемых и практичных методов обучения с учителем.

Деревья решений — это непараметрический контролируемый метод обучения, используемый как для задач классификации, так и для задач регрессии. Цель состоит в том, чтобы создать модель, которая предсказывает значение целевой переменной, изучая простые правила принятия решений, выведенные из характеристик данных. Правила принятия решений в большинстве случаев имеют форму операторов if-then-else. Чем глубже дерево, тем сложнее правила и точнее модель» [2].

Дерево решений — это древовидный граф с узлами, представляющими места, где мы выбираем атрибут и задаем вопрос; ребра представляют ответы на вопрос; а листья представляют фактический вывод или метку класса. Они используются при нелинейном принятии решений с простой линейной поверхностью принятия решений.

Общие способы использования моделей дерева решений включают следующее:

- выбор переменных. Количество переменных, которые регулярно отслеживаются в клинических условиях, резко увеличилось с введением электронного хранилища данных. Многие из этих переменных имеют незначительное значение, и поэтому их, вероятно, не следует включать в упражнения по интеллектуальному анализу данных. Как и пошаговый выбор переменных в регрессионном анализе, методы дерева решений могут использоваться для выбора наиболее подходящих входных переменных, которые следует использовать для формирования моделей дерева решений, которые впоследствии можно использовать для формулирования клинических гипотез и информирования последующих исследований;
- оценка относительной важности переменных. Как только набор релевантных переменных определен, исследователи могут захотеть узнать, какие переменные играют основную роль. Как правило, важность переменной вычисляется на основе снижения точности модели (или чистоты узлов в дереве) при удалении переменной. В большинстве случаев, чем больше записей влияет на переменную, тем выше ее важность;
- обработка пропущенных значений. Обычный, но неправильный метод обработки отсутствующих данных заключается в исключении случаев с отсутствующими значениями. Это неэффективно и может привести к систематической ошибке в анализе. Анализ дерева решений может работать с отсутствующими данными двумя способами. Первый способ может классифицировать отсутствующие значения как отдельную категорию, которую можно анализировать вместе с другими категориями. Второй способ может использовать построенную модель дерева решений, которая устанавливает переменную с большим количеством отсутствующих значений в

качестве целевой переменной, чтобы сделать прогноз и заменить эти отсутствующие прогнозируемым значением;

- прогноз. Это одно из наиболее важных применений моделей дерева решений. Используя древовидную модель, полученную из исторических данных, легко предсказать результат для будущих записей;
- манипуляции с данными. В медицинских исследованиях распространено слишком много категорий одной категориальной переменной или сильно искаженных непрерывных данных. В этих обстоятельствах модели дерева решений могут помочь в принятии решения о том, как лучше всего свернуть категориальные переменные в более управляемое количество категорий или как разделить сильно искаженные переменные на диапазоны.

Основные термины, которые необходимо знать для работы с деревьями решений:

- узлы: существует три типа узлов. Корневой узел, также называемый узлом принятия решения, представляет выбор, который приведет к подразделению всех записей на два или более взаимоисключающих подмножества. Внутренние узлы, также называемые случайными узлами, представляют собой один из возможных вариантов выбора, доступных в данной точке древовидной структуры. Верхний край узла соединяется с его родительским узлом, а нижний край соединяется с его дочерними узлами или конечными узлами. Листовые узлы, также называемые конечными узлами, представляют конечный результат комбинации решений или событий;
- ветви: Ветви представляют собой случайные результаты или события, исходящие из корневых узлов и внутренних узлов. Модель дерева решений формируется с использованием иерархии ветвей. Каждый путь от корневого узла через внутренние узлы к конечному узлу представляет правило принятия решения о классификации. Эти

пути дерева решений также могут быть представлены как правила «если-то». Например, «если выполняются условие 1, условие 2, условие … и условие k, то происходит исход j»;

- разделение**:** это процесс разделения узла на два или более подузлов;
- узел принятия решения**:** когда подузел разделяется на дополнительные подузлы, он называется узлом принятия решения;
- листовой/конечный узел**:** узлы, которые не разделяются, называются конечными или конечными узлами;
- сокращение**:** когда мы удаляем подузлы узла принятия решений, этот процесс называется сокращением. Можно сказать обратный процесс расщепления;
- ветвь/поддерево: часть всего дерева называется ветвью или поддеревом;
- родительский и дочерний узел: узел, который разделен на подузлы, называется родительским узлом подузлов, тогда как подузлы являются дочерними элементами родительского узла.

Деревья решений классифицируют примеры, сортируя их вниз по дереву от корня до некоторого конечного узла, при этом конечный узел предоставляет классификацию для примера.

Каждый узел в дереве действует как тестовый пример для некоторого атрибута, и каждое ребро, спускающееся с этого узла, соответствует одному из возможных ответов на тестовый пример. Этот процесс носит рекурсивный характер и повторяется для каждого поддерева, укоренившегося в новых узлах.

Проиллюстрируем это с помощью примера. Предположим, мы хотим сыграть в бадминтон в определенный день, скажем, в субботу, — как вы решите, играть или нет. Допустим, вы выходите и проверяете, жарко или холодно, проверяете скорость ветра и влажность, какая погода, т.е. солнечно, облачно или дождливо. Вы принимаете во внимание все эти факторы, чтобы решить, хотите ли вы играть или нет.

Итак, вы вычисляете все эти факторы за последние десять дней и формируете справочную таблицу, подобную приведенной ниже таблице 1.

Таблица 1 - Наблюдения за последние десять дней

День	Погода	Температура	Влажность	Ветер	Играть?
1	Солнечная	Жарко	Высокая	Слабый	Нет
2	Облачно	Жарко	Высокая	Слабый	Да
3	Солнечная	Нейтрально	Нормальная	Сильный	Да
4	Облачно	Нейтрально	Высокая	Сильный	Да
5	Дождливая	Нейтрально	Высокая	Сильный	Нет
6	Дождливая	Холодно	Нормальная	Сильный	Нет
7	Дождливая	Нейтрально	Высокая	Слабый	Да
8	Солнечная	Жарко	Высокая	Сильный	Нет
9	Облачно	Жарко	Нормальная	Слабый	Да
10	Дождливая	Нейтрально	Высокая	Сильный	Нет

Теперь вы можете использовать эту таблицу, чтобы решить, играть вам или нет. Но что, если погода в субботу не совпадает ни с одной строкой в таблице? Это может быть проблемой. Дерево решений было бы отличным способом представления данных, поскольку оно учитывает все возможные пути, которые могут привести к окончательному решению, следуя древовидной структуре на рисунке 3.

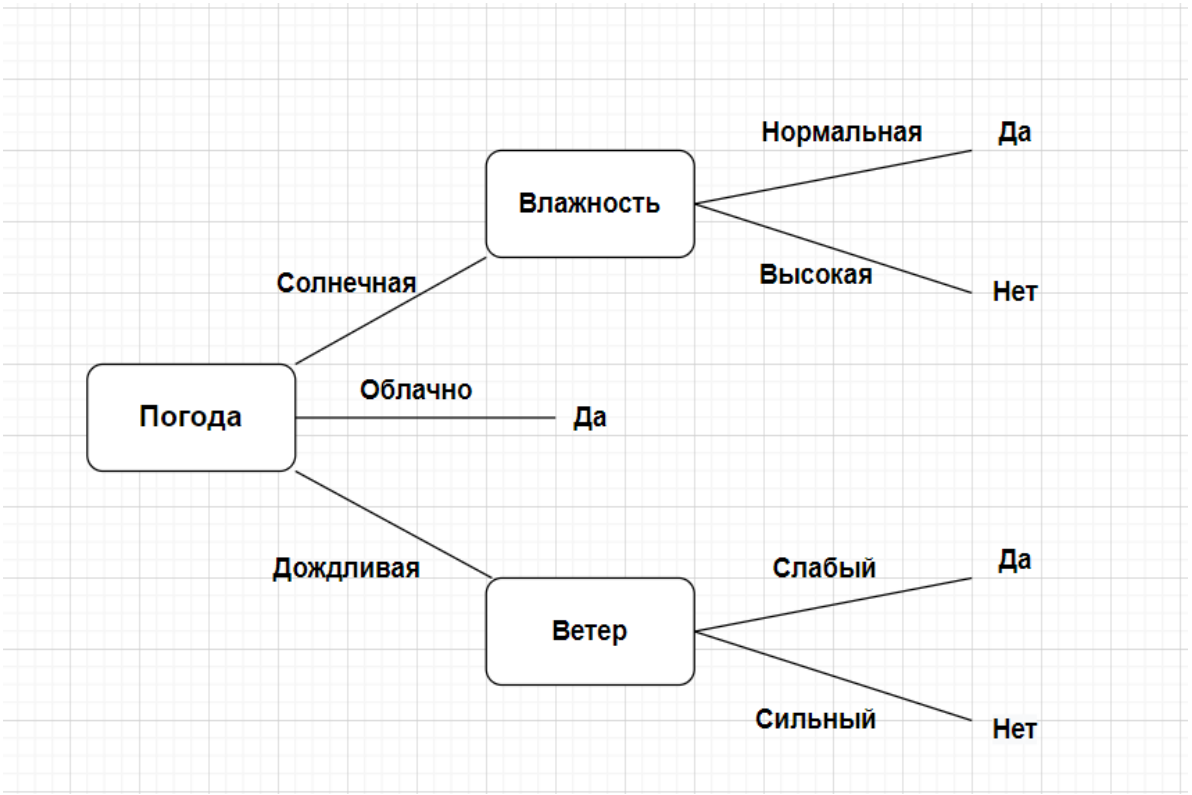

Рисунок 3 - Дерево решений для концепции «Играть в бадминтон»

Рисунок 3 иллюстрирует обученное дерево решений. Мы видим, что каждый узел представляет атрибут или функцию, а ветвь от каждого узла представляет собой результат этого узла. Наконец, именно на листьях дерева принимается окончательное решение. Если функции непрерывны,

внутренние узлы могут проверить значение функции на соответствие порогу (рис. 4).

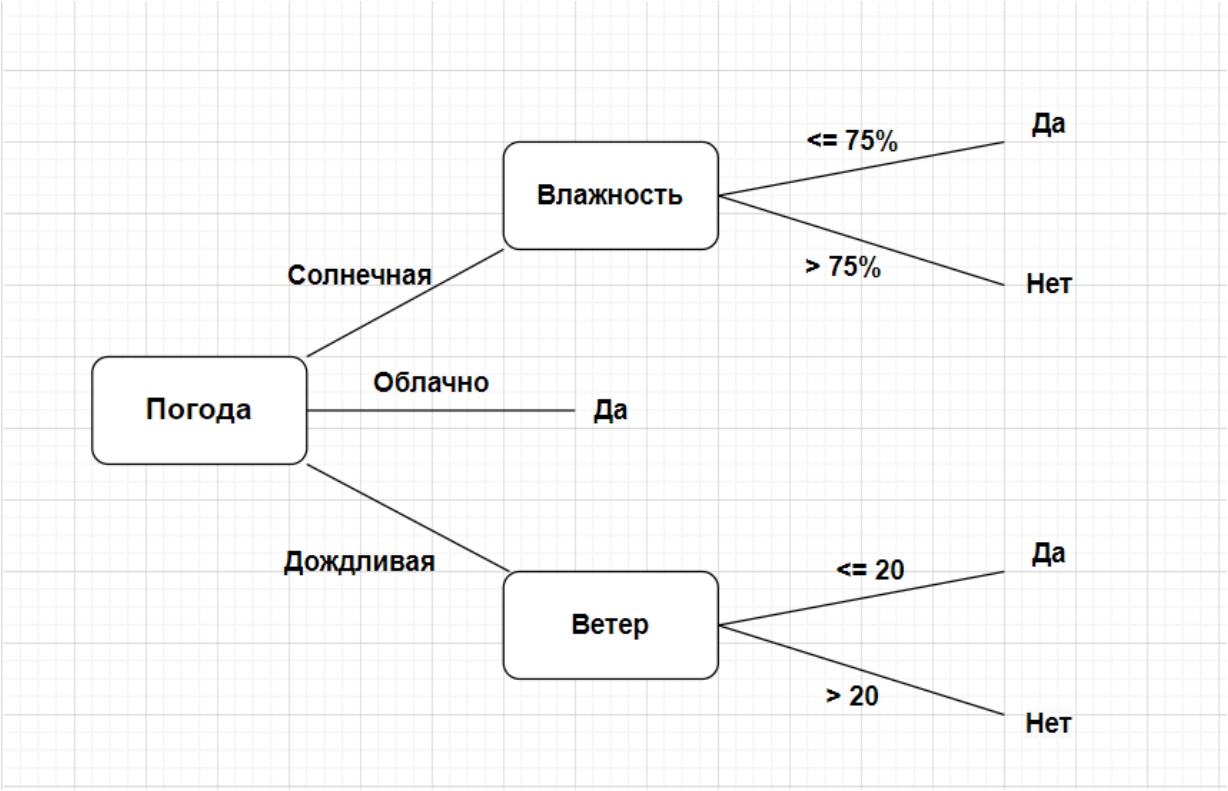

Рисунок 4 - Дерево решений для концепции «Играть в бадминтон» (когда атрибуты непрерывны)

Общий алгоритм дерева решений можно описать следующим образом:

- выберите лучший атрибут/особенность. Лучший атрибут - это тот, который лучше всего разбивает или разделяет данные;
- задайте соответствующий вопрос;
- следуйте пути ответа;
- переходите к шагу 1, пока не придете к ответу.

Лучшее разделение — это разделение двух разных этикеток на два набора.

1.3 Выразительность деревьев решений

Деревья решений могут представлять любую логическую функцию входных атрибутов. Давайте воспользуемся деревьями решений, чтобы выполнить функцию трех логических вентилей *AND*, *OR* и *XOR*.

Булевая функция *AND* изображена на рисунке 5.

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T

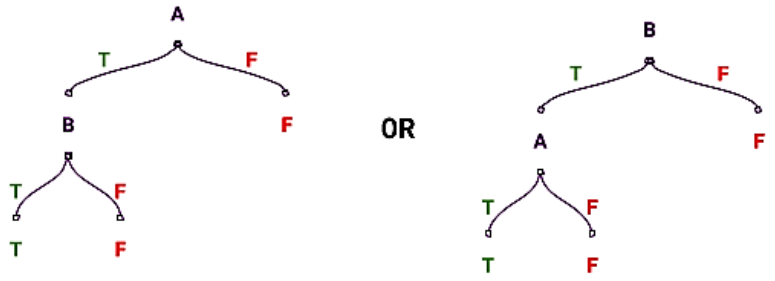

Рисунок 5 - Дерево решений для операции *AND*

На рисунке 5 мы видим, что есть две концепции-кандидата на создание дерева решений, выполняющего операцию *AND*. Точно так же мы можем создать дерево решений, которое выполняет логическую операцию *OR*.

Логическая функция: *OR* изображена на рисунке 6.

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

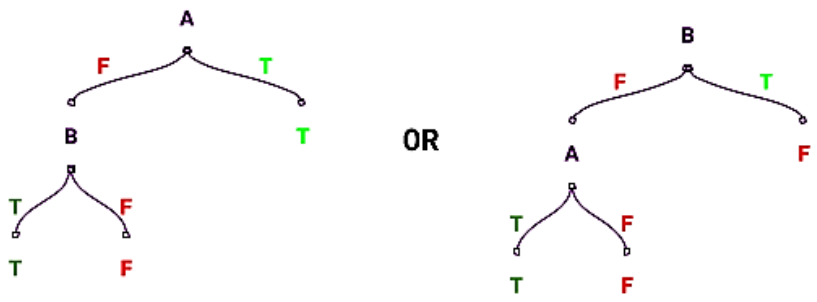

Рисунок 6 - Дерево решений для операции *OR*

Далее рассмотрим еще одну логическую функцию. Точно также, как и по двум предыдущим функциям мы можем создать дерево решений, которое будет выполнять нашу логическую операцию.

Логическая функция: *XOR* изображена на рисунке 7.

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

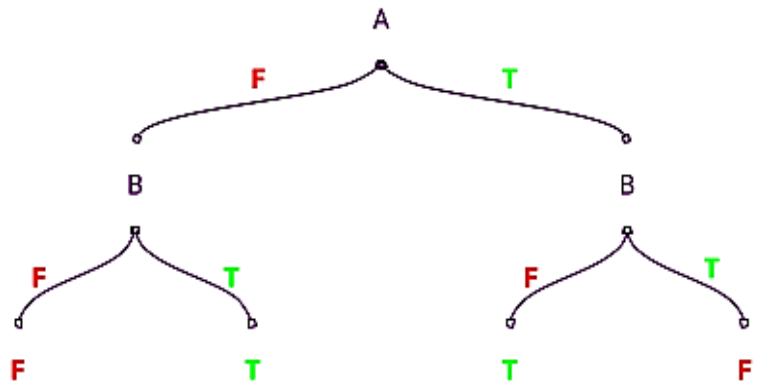

Рисунок 7 - Дерево решений для операции *XOR*

Давайте создадим дерево решений, выполняющее функцию *XOR*, используя 3 атрибута. Результат процесса изображен на рисунке 8.

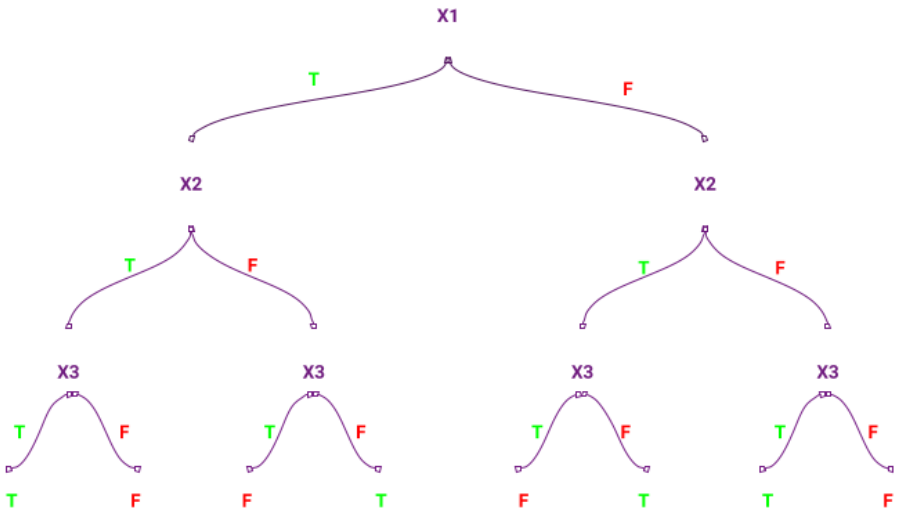

Рисунок 8 - Дерево решений для операции *XOR* с тремя операндами

В дереве решений, показанном выше, для трех признаков имеется *7* узлов в дереве, т. е. при *n = 3* количество узлов = *2^3-1*. Точно так же, если у нас есть *n* атрибутов, в дереве решений будет *2^n* узлов (приблизительно). Таким образом, дерево требует экспоненциального количества узлов в худшем случае.

Мы можем представить логические операции, используя деревья решений. Но какие еще функции мы можем представить и о скольких деревьях решений нам придется беспокоиться? Давайте ответим на этот вопрос, выяснив возможное количество деревьев решений, которые мы можем сгенерировать с заданными *N* различными атрибутами (при условии, что атрибуты булевы). Поскольку таблицу истинности можно преобразовать в дерево решений, мы сформируем таблицу истинности из *N* атрибутов в качестве входных данных, приведенных на таблице 2.

Таблица 2 – Таблица истинности из *N* атрибутов

X1	X2	X3	....	XN	OUTPUT
T	T	T	...	T	…
T	T	T	...	F	…
...	...	...	...	...	…
F	F	F	...	F	…

Приведенная выше таблица 2 содержит *2^n* строк (т. е. количество узлов в дереве решений), что представляет возможные комбинации входных атрибутов, и, поскольку каждый узел может содержать двоичное значение, количество способов заполнить значений в дереве решений *{2^{2^n}}*. Таким

образом, пространство деревьев решений, т. е. пространство гипотез дерева решений, очень выразительно, поскольку оно может представлять множество различных функций. Но это также означает, что нужно иметь хитрый способ поиска лучшего дерева среди них.

Стоит отметить разницу в вышеописанных деревьях.

Так последним мы описали бинарное дерево. Итак, хоть это и лучшее разделение, но на практике зачастую применяется достаточно редко. И для построения оных используются другие алгоритмы, чем для построения деревьев решения, в которых из узла выходит, не два, а больше ветвей. Также разные алгоритмы применяются в деревьях решений, в которых функции непрерывны. Различие в алгоритме оных будет описано в следующей главе.

Выработка решения в условиях риска.

Условия риска и неопределенности характеризуются так называемыми условиями многозначных ожиданий будущей ситуации во внешней среде. В этом случае лицо принимающее решение должно сделать выбор альтернативы (Не имея точного представления о факторах внешней среды и их влиянии на результат).

В этих условиях исход, результат каждой альтернативы представляет собой функцию условий - факторов внешней среды (функцию полезности), который не всегда способно предвидеть лицо принимающее решение. Для представления и анализа результатов выбранных альтернативных стратегий используют матрицу решений, называемую также платежной матрицей.

2 Методы и алгоритмы решения задач с помощью дерева решений

2.1 Деревья решений: общие принципы

Как уже упоминалось ранее, задачами, которые решаются с помощью данного аппарата, являются:

- классификация — отнесение объектов к одному из заранее известных классов. Целевая переменная должна иметь дискретные значения;
- регрессия (численное предсказание) — предсказание числового значения независимой переменной для заданного входного вектора;
- описание объектов — набор правил в дереве решений позволяет компактно описывать объекты. Поэтому вместо сложных структур, описывающих объекты, можно хранить деревья решений.

Процесс построения.

«Процесс построения деревьев решений заключается в последовательном, рекурсивном разбиении обучающего множества на подмножества с применением решающих правил в узлах. Процесс разбиения продолжается до тех пор, пока все узлы в конце всех ветвей не будут объявлены листьями. Объявление узла листом может произойти естественным образом (когда он будет содержать единственный объект, или объекты только одного класса), или по достижении некоторого условия остановки, задаваемого пользователем (например, минимально допустимое число примеров в узле или максимальная глубина дерева)» [5].

Алгоритмы построения деревьев решений относят к категории так называемых жадных алгоритмов. Жадными называются алгоритмы, которые допускают, что локально-оптимальные решения на каждом шаге (разбиения в узлах), приводят к оптимальному итоговому решению. В случае деревьев решений это означает, что если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться

назад и выбрать другой атрибут, который дал бы лучшее итоговое разбиение. Поэтому на этапе построения нельзя сказать обеспечит ли выбранный атрибут, в конечном итоге, оптимальное разбиение. [6]

В основе большинства популярных алгоритмов обучения деревьев решений лежит принцип «разделяй и властвуй». Алгоритмически этот принцип реализуется следующим образом. Пусть задано обучающее множество S, содержащее n примеров, для каждого из которых задана метка класса *Ci(i=1..k),* и m атрибутов *Aj(j=1..m),* которые, как предполагается, определяют принадлежность объекта к тому или иному классу. Тогда возможны три случая:

- «все примеры множества *S* имеют одинаковую метку класса *Ci* (т.е. все обучающие примеры относятся только к одному классу). Очевидно, что обучение в этом случае не имеет смысла, поскольку все примеры, предъявляемые модели, будут одного класса, который и «научится» распознавать модель. Само дерево решений в этом случае будет представлять собой лист, ассоциированный с классом *Ci*. Практическое использование такого дерева бессмысленно, поскольку любой новый объект оно будет относить только к этому классу» [7];
- «множество S вообще не содержит примеров, т.е. является пустым множеством. В этом случае для него тоже будет создан лист (применять правило, чтобы создать узел, к пустому множеству бессмысленно), класс которого будет выбран из другого множества (например, класс, который наиболее часто встречается в родительском множестве)» [8];
- множество *S* содержит обучающие примеры всех классов *Ck*. В этом случае требуется разбить множество *S* на подмножества, ассоциированные с классами. Для этого выбирается один из атрибутов *Aj* множества *S* который содержит два и более уникальных значения *(a1,a2,...,ap),* где *p* — число уникальных значений признака.

Затем множество *S* разбивается на *p* подмножеств *(S1,S2,...,Sp),* каждое из которых включает примеры, содержащие соответствующее значение атрибута. Затем выбирается следующий атрибут и разбиение повторяется. Это процедура будет рекурсивно повторяться до тех пор, пока все примеры в результирующих подмножествах не окажутся одного класса.

Описанная выше процедура лежит в основе многих современных алгоритмов построения деревьев решений. Очевидно, что при использовании данной методики, построение дерева решений будет происходить сверху вниз (от корневого узла к листьям).

На данный момент существует немалое количество алгоритмов обучающих деревья решений, но наиболее популярными считаются: ID3, CART, C4.5.

ID3 (Итеративный дихотомайзер 3) был разработан в 1986 году Россом Куинланом. Алгоритм создает многоходовое дерево, находя для каждого узла (т.е. жадным способом) категориальный признак, который даст наибольший прирост информации для категориальных целей. Деревья вырастают до максимального размера, а затем обычно применяется этап обрезки, чтобы улучшить способность дерева обобщать невидимые данные.

«C4.5 является преемником ID3 и убрал ограничение, согласно которому объекты должны быть категориальными, путем динамического определения дискретного атрибута (на основе числовых переменных), который разбивает непрерывное значение атрибута на дискретный набор интервалов. C4.5 преобразует обученные деревья (т. е. выходные данные алгоритма ID3) в наборы правил «если-то». Эта точность каждого правила затем оценивается, чтобы определить порядок, в котором они должны применяться. Сокращение выполняется путем удаления предварительного условия правила, если точность правила улучшается без него» [1].

«CART (деревья классификации и регрессии) очень похож на C4.5, но отличается тем, что поддерживает числовые целевые переменные (регрессия)

и не вычисляет наборы правил. CART строит бинарные деревья, используя функцию и порог, которые дают наибольший прирост информации в каждом узле» [9].

Основные этапы построения.

Для того чтобы построить дерево решений необходимо решить несколько задач, которые связаны с соответствующим шагом процесса обучения:

- выбор атрибута, по которому будет производиться разбиение в данном узле (атрибута разбиения);
- выбор критерия остановки обучения;
- выбор метода отсечения ветвей (упрощения);
- оценка точности построенного дерева.

Ниже рассмотрим эти этапы поподробнее.

Выбор атрибута разбиения.

«При формировании правила для разбиения в очередном узле дерева необходимо выбрать атрибут, по которому это будет сделано. Общее правило для этого можно сформулировать следующим образом: выбранный атрибут должен разбить множество наблюдений в узле так, чтобы результирующие подмножества содержали примеры с одинаковыми метками класса, или были максимально приближены к этому, т.е. количество объектов из других классов (примесей) в каждом из этих множеств было как можно меньше. Для этого были выбраны различные критерии, наиболее популярными из которых стали теоретико-информационный и статистический» [2].

Теоретико-информационный критерий

Как следует из названия, критерий основан на понятиях теории информации, а именно — информационной энтропии (формула 1).

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right) \quad (1)$$

где n — число классов в исходном подмножестве;

Ni — число примеров i-го класса;

N — общее число примеров в подмножестве.

Энтропия — это способ измерения неопределенности класса в подмножестве примеров. Предположим, что элемент принадлежит подмножеству S, имеющему два класса: положительный и отрицательный. Энтропия определяется как N битов, необходимых, чтобы сказать, является ли x положительным или отрицательным [10]-[13].

Энтропия всегда дает число от 0 до 1. Поэтому, если подмножество, сформированное после разделения с использованием атрибута, является чистым, то нам понадобятся нулевые биты, чтобы сказать, является ли оно положительным или отрицательным. Если сформированное подмножество имеет равные N положительных и отрицательных элементов, то N необходимых битов будет 1 как изображено на рисунке 9.

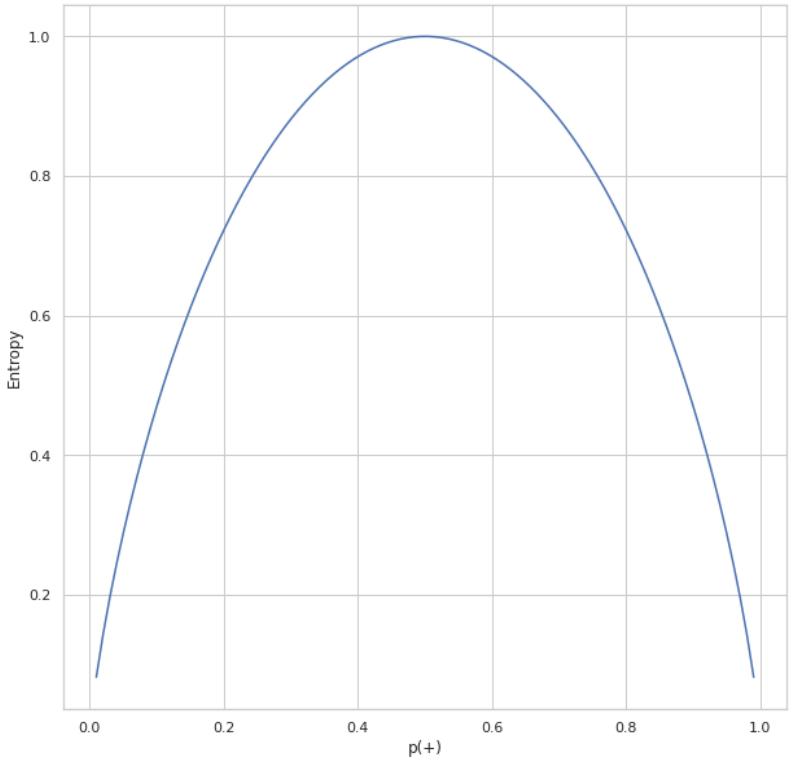

Рисунок 9 - Энтропия

Приведенный выше график показывает отношение между энтропией и. вероятностью положительного класса. Как мы видим, энтропия достигает *1*, что является максимальным значением, когда есть равные шансы для элемента быть положительным или отрицательным. Энтропия минимальна, когда *p(+)* стремится к нулю (символизируя *x* отрицательно) или *1* (символизируя *x* положительно).

Энтропия говорит нам, насколько чистым или нечистым является каждое подмножество после разделения. Что нам нужно сделать, так это агрегировать эти оценки, чтобы проверить, возможно ли разделение или нет. Это делается путем получения информации.

«Таким образом, энтропия может рассматриваться как мера неоднородности подмножества по представленным в нём классам. Когда классы представлены в равных долях и неопределённость классификации наибольшая, энтропия также максимальна. Если все примеры в узле относятся к одному классу, т.е. *N=Ni,* логарифм от единицы обращает энтропию в ноль» [14].

Лучшим атрибутом разбиения *Aj* будет тот, который обеспечит максимальное снижение энтропии результирующего подмножества относительно родительского. На практике, однако, говорят не об энтропии, а о величине, обратной ей, которая называется информацией. Тогда лучшим атрибутом разбиения будет тот, который обеспечит максимальный прирост информации результирующего узла относительно исходного (формула 2):

$$Gain(A) = Info(S) - Info(SA), \quad (2)$$

где *Info(S)* —информация, связанная с подмножеством *S* до разбиения;

Info(S_A) —информация, связанная с подмножеством, полученными при разбиении по атрибуту *A*.

Таким образом, задача выбора атрибута разбиения в узле заключается в максимизации величины *Gain(A)*, называемой приростом информации (от

англ. *gain* — прирост, увеличение). Поэтому сам теоретико-информационный подход известен как критерий прироста информации. Он впервые был применён в алгоритме ID3, а затем в C4.5 и других алгоритмах [15].

Критерий остановки алгоритма.

«Теоретически, алгоритм обучения дерева решений будет работать до тех пор, пока в результате не будут получены абсолютно «чистые» подмножества, в каждом из которых будут примеры одного класса. Правда, возможно при этом будет построено дерево, в котором для каждого примера будет создан отдельный лист. Очевидно, что такое дерево окажется бесполезным, поскольку оно будет переобученным — каждому примеру будет соответствовать свой уникальный путь в дереве, а, следовательно, и набор правил, актуальный только для данного примера» [16].

Переобучение в случае дерева решений ведёт к тем же последствиям, что и для нейронной сети — точное распознавание примеров, участвующих в обучении, и полная несостоятельность на новых данных. Кроме этого, переобученные деревья имеют очень сложную структуру, и поэтому их сложно интерпретировать.

Очевидным решением проблемы является принудительная остановка построения дерева, пока оно не стало переобученным. Для этого разработаны следующие подходы.

- «ранняя остановка — алгоритм будет остановлен, как только будет достигнуто заданное значение некоторого критерия, например, процентной доли правильно распознанных примеров. Единственным преимуществом подхода является снижение времени обучения. Главным недостатком является то, что ранняя остановка всегда делается в ущерб точности дерева, поэтому многие авторы рекомендуют отдавать предпочтение отсечению ветвей» [17];
- ограничение глубины дерева — задание максимального числа разбиений в ветвях, по достижении которого обучение

останавливается. Данный метод также ведёт к снижению точности дерева;

- задание минимально допустимого число примеров в узле — запретить алгоритму создавать узлы с числом примеров меньше заданного (например, 5). Это позволит избежать создания тривиальных разбиений и, соответственно, малозначимых правил.

«Все перечисленные подходы являются эвристическими, т.е. не гарантируют лучшего результата или вообще работают только в каких-то частных случаях. Поэтому к их использованию следует подходить с осторожностью. Каких-либо обоснованных рекомендаций по тому, какой метод лучше работает, в настоящее время тоже не существует. Поэтому аналитикам приходится использовать метод проб и ошибок» [18].

Отсечение ветвей.

Как было отмечено выше, если «рост» дерева не ограничить, то в результате будет построено сложное дерево с большим числом узлов и листьев. Как следствие оно будет трудно интерпретируемым. В то же время решающие правила в таких деревьях, создающие узлы, в которые попадают два-три примера, оказываются малозначимыми с практической точки зрения.

«Гораздо предпочтительнее иметь дерево, состоящее из малого количества узлов, которым бы соответствовало большое число примеров из обучающей выборки. Поэтому представляет интерес подход, альтернативный ранней остановке — построить все возможные деревья и выбрать то из них, которое при разумной глубине обеспечивает приемлемый уровень ошибки распознавания, т.е. найти наиболее выгодный баланс между сложностью и точностью дерева» [19].

Альтернативным подходом является так называемое отсечение ветвей (pruning). Он содержит следующие шаги:

- построить полное дерево (чтобы все листья содержали примеры одного класса);

- определить два показателя: относительную точность модели — отношение числа правильно распознанных примеров к общему числу примеров, и абсолютную ошибку — число неправильно классифицированных примеров;
- удалить из дерева листья и узлы, отсечение которых не приведёт к значимому уменьшению точности модели или увеличению ошибки.

«Отсечение ветвей, очевидно, производится в направлении, противоположном направлению роста дерева, т.е. снизу-вверх, путём последовательного преобразования узлов в листья. Преимуществом отсечения ветвей по сравнению с ранней остановкой является возможность поиска оптимального соотношения между точностью и понятностью дерева. Недостатком является большее время обучения из-за необходимости сначала построить полное дерево» [20].

Альтернативный метод предотвращения переоснащения состоит в том, чтобы попытаться остановить процесс построения дерева на ранней стадии, прежде чем он создаст листья с очень маленькими выборками. Эта эвристика известна как ранняя остановка, но иногда также известна как предварительная обрезка деревьев решений.

На каждом этапе разбиения дерева мы проверяем ошибку перекрестной проверки. Если ошибка не уменьшается достаточно значительно, мы останавливаемся. Ранняя остановка может оказаться недостаточной из-за слишком ранней остановки. Текущий сплит может принести мало пользы, но сделав его, последующие сплиты более значительно уменьшат ошибку.

Извлечение правил

Иногда даже упрощённое дерево решений все ещё является слишком сложным для визуального восприятия и интерпретации. В этом случае может оказаться полезным извлечь из дерева решающие правила и организовать их в наборы, описывающие классы.

«Для извлечения правил нужно отследить все пути от корневого узла к листьям дерева. Каждый такой путь даст правило, состоящее из множества условий, представляющих собой проверку в каждом узле пути.

Визуализация сложных деревьев решений в виде решающих правил вместо иерархической структуры из узлов и листьев может оказаться более удобной для визуального восприятия» [21]-[23].

2.2 Алгоритм CART

CART (Classification and regression trees) - представляют собой набор методов классификации и прогнозирования. Техника направлена на создание правил, которые предсказывают значение выходной (целевой) переменной на основе известных значений предикторных (объяснительных) переменных. Переменные-предикторы могут быть смесью категориальных и непрерывных переменных. Кроме того, CART может быть уместным в таких сферах услуг, как банковское дело и здравоохранение, где многие потенциальные причины отклонений и дефектов носят категорический характер (например, географическое положение, продукты, каналы связи, партнеры) [26].

Проблема с использованием регрессионных или обобщенных линейных моделей (GLM) в таких случаях заключается в том, что много фиктивных переменных затрудняют интерпретацию результатов. CART — полезный непараметрический метод, который можно использовать для объяснения непрерывной или категориальной зависимой переменной с точки зрения нескольких независимых переменных. Независимые переменные могут быть непрерывными или категориальными. CART использует подход к разделению, широко известный как «разделяй и властвуй».

Алгоритм CART работает, чтобы найти независимую переменную, которая создает лучшую однородную группу при разделении данных. Для задачи классификации, где переменная ответа является категориальной, это

решается путем вычисления информации, полученной на основе энтропии, полученной в результате разделения. Для числового ответа однородность измеряется статистическими данными, такими как стандартное отклонение или дисперсия [24].

Двумя важными параметрами метода CART являются критерий минимального разделения и параметр сложности (*Cp*). Критерий минимального разделения — это минимальное количество записей, которые должны присутствовать в узле, прежде чем можно будет попытаться выполнить разделение. Это должно быть указано в самом начале. *Cp* — это параметр сложности, позволяющий избежать разделения тех узлов, которые явно не нужны. Другой способ рассмотрения этих параметров заключается в том, что значение C p определяется после «выращивания дерева», а оптимальное значение используется для «обрезки дерева».

Алгоритм CART продолжает делить набор данных до тех пор, пока в каждом «листовом» узле не останется минимальное количество записей, как указано критерием минимального разделения. В результате получается древовидная структура. Затем значение *Cp* наносится на график относительно различных уровней дерева, и оптимальное значение используется для обрезки дерева [27].

В алгоритме CART идея неопределенности формализована в индексе Gini. Если набор данных *T* содержит данные n классов, тогда индекс *Gini* определяется следующим образом.

$$Gini(T) = 1 \sum_{i=1}^{n} p_i^2, \qquad (3)$$

где *pi* — это относительная частота (вероятность) класса *i* в *T*.

Если набор *T* разбивается на две части *T1* и *T2* с числом примеров в каждом *N1* и *N2* соответственно, тогда показатель качества разбиения будет равен:

$$Gini_{split}(T) = \frac{N_1}{N} * Gini(T_1) + \frac{N_2}{N} * Gini(T_2), \quad (4)$$

Наилучшим считается то разбиение, для которого *Ginisplit(T)* минимально. Обозначим *N* — число примеров в узле — предке, *L*, *R* — число примеров соответственно в левом и правом потомке, *li* и *ri* — число экземпляров *i*-го класса в левом/правом потомке. Тогда качество разбиения оценивается по следующей формуле:

$$Gini_{split} = \frac{L}{N} * \left(1 - \sum_{i=1}^{n} \left(\frac{l_i}{L}\right)^2\right) + \frac{R}{N} * \left(1 - \sum_{i=1}^{n} \left(\frac{r_i}{R}\right)^2\right) \rightarrow min, \quad (5)$$

Чтобы уменьшить объем вычислений формулу можно преобразовать:

$$Gini_{split} = \frac{1}{N} * \left(L * \left(\frac{1}{L^2} - \sum_{i=1}^{n} l_i^2\right)\right) + R * \left(1 - \frac{1}{R^2} * \sum_{i=1}^{n} r_i^2\right) \rightarrow min,$$

(6)

Так как умножение на константу не играет роли при минимизации:

$$Gini_{split} = L - \frac{1}{L} * \sum_{i=1}^{n} l_i^2 + R - \frac{1}{R} * \sum_{i=1}^{n} r_i^2 \rightarrow min, \quad (7)$$

$$Gini_{split} = N - \left(\frac{1}{L} + \sum_{i=1}^{n} l_i^2 + \frac{1}{R} * \sum_{i=1}^{n} r_i^2\right) \rightarrow min, \quad (8)$$

$$\tilde{G}_{split} = \frac{1}{L} * \sum_{i=1}^{n} l_i^2 + \frac{1}{R} * \sum_{i=1}^{n} r_i^2 \rightarrow max, \quad (9)$$

В итоге, лучшим будет то разбиение, для которого величина максимальна. Таким образом, при построении «дерева решений» по методу CART ищется такой вариант ветвления, при котором максимально уменьшается значение показателя *Ginisplit(T)*.

2.3 Алгоритм ID3

ID3 считается очень простым алгоритмом дерева решений. Он определяет классификацию объектов или записей путем проверки значений их атрибутов. Он строит дерево решений для заданных данных в нисходящей структуре, начиная с набора записей и набора атрибутов. В каждом узле дерева проверяется один атрибут на основе максимизации измерения прироста информации и минимизации измерения энтропии, а результат используется для разделения записей. Этот процесс рекурсивно выполняется до тех пор, пока записи, данные в поддереве, не станут однородными (все записи принадлежат одному и тому же классу). Эти однородные записи становятся конечными узлами дерева решений [25].

«Пусть атрибут X принимает три значения: *A, B* и *C*. Тогда при разбиении исходного множества T по атрибуту X алгоритм сформирует три узла-потомка *T1(A), T2(B)* и *T3(C),* в первом из которых будут содержаться все примеры, в которых атрибут X принимает значение *A*, во втором — значение B, и в третьем — C. Процесс рекурсивно повторяется до тех пор, пока не будут сформированы подмножества, содержащие примеры только одного класса. Выбор атрибута на каждом разбиении производится с помощью критерия прироста информации» [4].

$$Gain(X) = Info(T) - InfoX(T), \quad (10)$$

где *Info(T)* — информация множества до разбиения,

InfoX(T) — информация после разбиения по атрибуту X.

В качестве атрибута разбиения выбирается атрибут, который обеспечивает максимальное значение *Gain(X)*.

2.4 Алгоритм C4.5

Алгоритм C4.5 используется в интеллектуальном анализе данных в качестве классификатора дерева решений, который можно использовать для генерации решения на основе определенной выборки данных (одномерные или многомерные предикторы).

C4.5 представляет собой эволюцию ID3, представленную тем же автором. Обучающий набор данных для алгоритма C4.5 представляет собой множество примеров *S=S1,S2,...,Sn*, для которых предварительно задана метка класса. Каждый пример представляет собой p-мерный вектор значений атрибутов *x=x1,x2,...,xp*.

Алгоритм C4.5 генерирует дерево решений для заданного набора данных путем рекурсивного разделения записей. Алгоритм C4.5 учитывает категориальные и числовые атрибуты. Для каждого категориального атрибута C4.5 вычисляет прирост информации и выбирает тот, который имеет наибольшее значение, и использует атрибут для получения множества результатов в виде числа различных значений этого атрибута. Для каждого числового атрибута существует два метода расчета прироста информации, первый метод состоит в расчете коэффициента усиления, а второй метод состоит в расчете прироста информации, как мы это делали в ID3, но с некоторыми отличиями.

Данные алгоритмы решения задач с помощью дерева решений являются самыми распространенными, но это не означает, что любой из них идеально подойдет для любой задачи. Необходимо помнить, что каждый случай и каждая задачи индивидуальны и требуют предварительного анализа и персонального подбора алгоритма решения.

Процесс построения дерева будет происходить сверху вниз.

«На первом шаге мы имеем пустое дерево (имеется только корень) и исходное множество *T* (ассоциированное с корнем). Требуется разбить исходное множество на подмножества. Это можно сделать, выбрав один из атрибутов в качестве проверки. Тогда в результате разбиения получаются n(по числу значений атрибута) подмножеств и, соответственно, создаются n потомков корня, каждому из которых поставлено в соответствие свое подмножество, полученное при разбиении множества T. Затем эта процедура рекурсивно применяется ко всем подмножествам (потомкам корня) и т.д» [4].

Рассмотрим подробнее критерий выбора атрибута.

Пусть $freq(C_j, S)$ – количество примеров из некоторого множества S, относящихся к одному и тому же классу *Cj*. Тогда вероятность того, что случайно выбранный пример из множества *S* будет принадлежать к классу *Cj*. Ниже запишем формулу 11.

$$P = \frac{freq(C_i,S)}{|S|}\log_2\left(\frac{1}{P}\right), \qquad (11)$$

Согласно теории информации, количество содержащейся в сообщении информации, зависит от ее вероятности.

Поскольку мы используем логарифм с двоичным основанием, то формула 11 дает количественную оценку в битах. Далее получаем формулу 12.

$$Info(T) = -\sum_{j=1}^{k} \frac{freq(C_j, T)}{|T|} \log_2 \left(\frac{freq(C_j, T)}{|T|} \right), \quad (12)$$

дает оценку среднего количества информации, необходимого для определения класса примера из множества T. В терминологии теории информации выражение (формула 12) называется энтропией множества T.

Ту же оценку, но только уже после разбиения множества T по X, дает следующее выражение:

$$Info_x(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} * info(T_i), \quad (13)$$

Тогда критерием для выбора атрибута будет являться следующая формула:

$$Gain(A) = Info(T) - InfoA(T), \quad (14)$$

«Критерий (4) считается для всех атрибутов. Выбирается атрибут, максимизирующий данное выражение. Этот атрибут будет являться проверкой в текущем узле дерева, а затем по этому атрибуту производится дальнейшее построение дерева. Т.е. в узле будет проверяться значение по этому атрибуту и дальнейшее движение по дереву будет производиться в зависимости от полученного ответа» [4].

3 Реализация и тестирование алгоритмов построения дерева решений

Для реализации своей задачи в качестве языка программирования я выбрала python так как он как никто другой подходит для анализа данных машинного обучения.

Для хранения и работы с dataSet я выбрала библиотеку pandas, а для визуализации я выбрала graphviz.

Прежде всего загружаем dataSet. Хранить мы его будем в DataFrame. И делаем мы это с помощью команды: df = pd.read_csv("dataset/golf2.txt").

Дальше мы запускаем основной метод для построения дерева.

buildDecisionTree(df,root,prevIndex),

где df - наш DataSet,

Root - это уровень того насколько мы вошли в рекурсию,

prevIndex - вспомогательная переменная для создания графики.

Сразу после вхождения в этот метод мы находим атрибут, который разделит наш массив с наибольшим коэффициентом критерия прироста информации, нахождение которого зависит от выбранного метода.

Пример кода:

```
if algorithm == 'ID3' or algorithm == 'C4.5':
        subset_entropy = calculateEntropy(subdataset)
        gain = gain - class_probability * subset_entropy
      if algorithm == 'C4.5':
        splitinfo = splitinfo - 
class_probability*math.log(class_probability, 2)
      elif algorithm == 'CART':
        decision_list = 
subdataset['Decision'].value_counts().tolist()
        subgini = 1
        for k in range(0, len(decision_list)):
```

```
                subgini = subgini -
math.pow((decision_list[k]/subset_instances), 2)
            gini = gini + (subset_instances / instances) *
subgini
```

После нахождения самого значимого критерия мы делим исходный DataSet на количество уникальных вхождений признака в самый значимый критерий. После того как мы их разделили, для каждого из подмножеств – процесс рекурсивно начинается сначала. До того момента, когда в каждом из подмножеств не останется по одному элементу. Когда в подмножестве остается один элемент – это условие выхода из рекурсии.

Стоит отметить, что данный алгоритм правдив только для номинальных критериев, но не для числовых.

В связи с тем, что алгоритмы C4.5 и CART могут работать с числовыми критериями – был разработан следующий функционал. Все числа из критерии сортируются, после чего алгоритм циклом проходит по всем уникальным числам и в зависимости от алгоритма. Далее ищет число, деление массива на которое принесёт максимальный коэффициент прироста информации.

Пример кода из приложения А:

```
    if algorithm == 'C4.5':
          threshold_gain          =          entropy          -
subset1_probability*calculateEntropy(subset1)                  -
subset2_probability*calculateEntropy(subset2)
          subset_gains.append(threshold_gain)
          threshold_splitinfo    =    -subset1_probability    *
math.log(subset1_probability,                                 2)-
subset2_probability*math.log(subset2_probability, 2)
          gainratio = threshold_gain / threshold_splitinfo
          subset_gainratios.append(gainratio)
        elif algorithm == 'CART':
```

```
            decision_for_subset1                                    =
subset1['Decision'].value_counts().tolist()

            decision_for_subset2                                    =
subset2['Decision'].value_counts().tolist()

            gini_subset1 = 1; gini_subset2 = 1

            for j in range(0, len(decision_for_subset1)):

              gini_subset1          =          gini_subset1          -
math.pow((decision_for_subset1[j]/subset1_rows),2)

            for j in range(0, len(decision_for_subset2)):

              gini_subset2          =          gini_subset2          -
math.pow((decision_for_subset2[j]/subset2_rows),2)

            gini  =  (subset1_rows/total_instances)*gini_subset1  +
(subset2_rows/total_instances) * gini_subset2

            subset_ginis.append(gini)
```

Стоит отметить, что вышеописанный фрагмент не работает для алгоритма ID3. Основным отличием алгоритма ID3 от алгоритма C4.5 является то, что C4.5 умеет работать с числовыми критериями.

Так, например, при одинаковых входных значениях алгоритм C4.5 выдает в качестве результата дерево решений на рисунке 10.

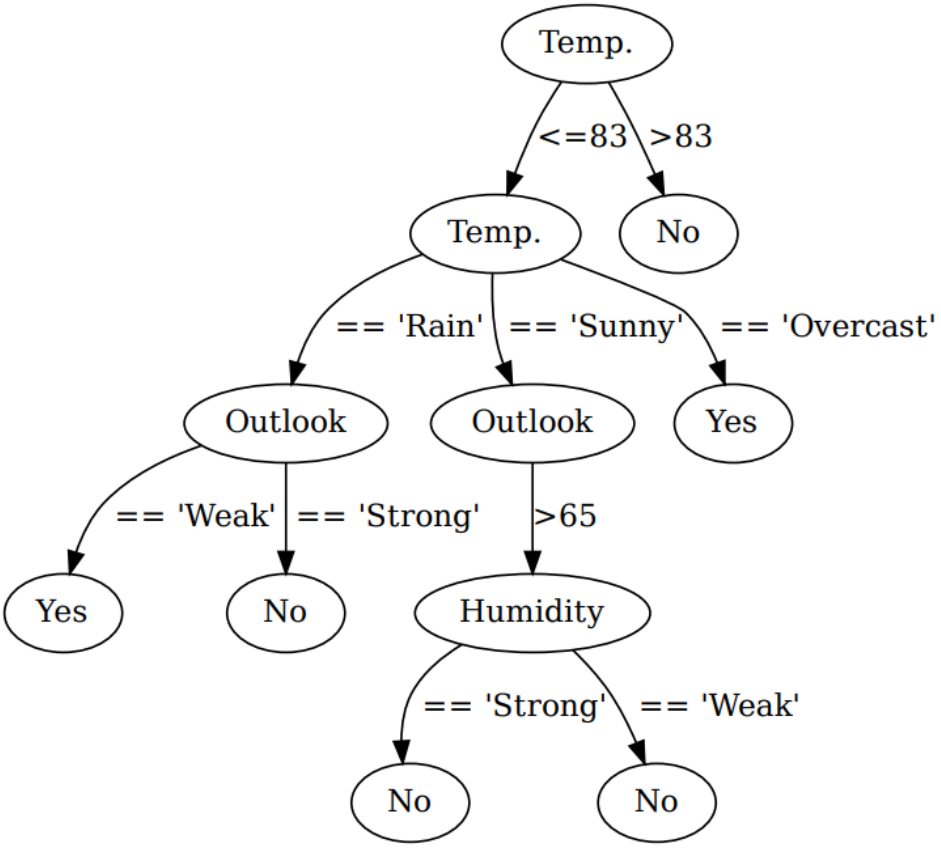

Рисунок 10 – Результат работы алгоритма C4.5

А алгоритм ID3 выдает в качестве результата дерево решений следующего вида (рис. 11):

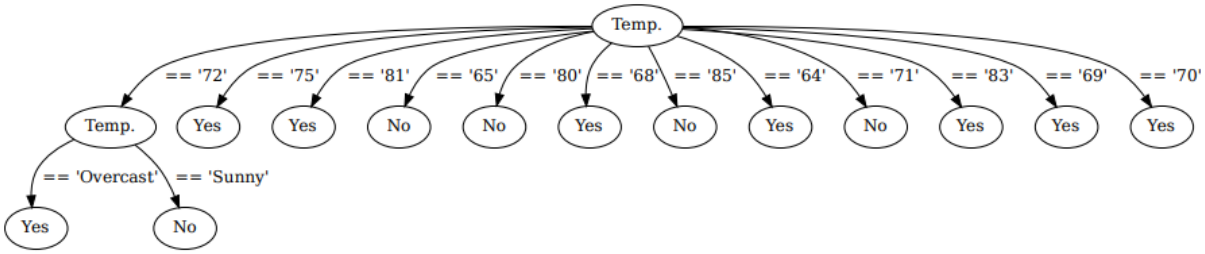

Рисунок 11 – Результат работы алгоритма ID3

Это происходит в связи с тем, что алгоритм ID3 каждое число воспринимает ни как число, а как новую лексему. В этом и заключается основная разница между методами C4.5 и ID3.

Методы C4.5 и ID3 родственные, в отличие от метода CART. Разница между CART и двумя предыдущими алгоритмами заключается в том, что, когда каждый узел выносит суждение, рассматривается только случай двух классификаций, даже если можно получить несколько значений. Например, есть три цветных шара, ID3 и C4. 5 напрямую разделены на три подкатегории, а CART может быть разделена только на определенный цвет, а затем оценена, когда она разделена один раз, на самом деле это двоичное дерево. На рисунке 12 приведены результаты работы алгоритма CART.

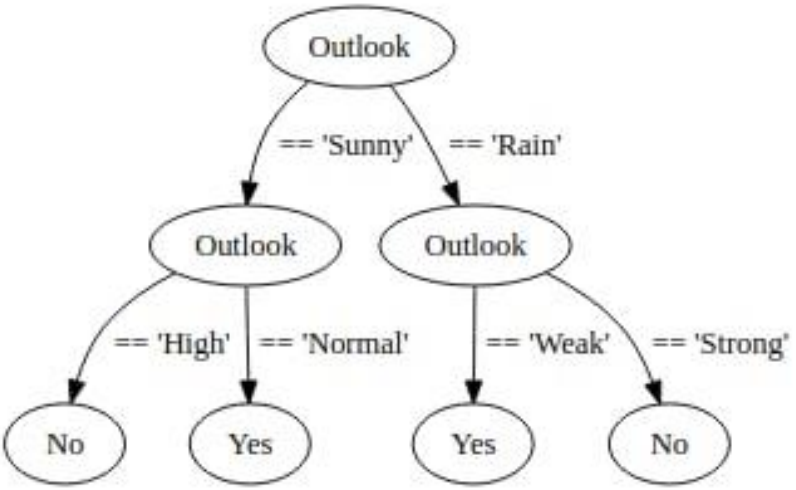

Рисунок 12 – Результат работы алгоритма CART

Стоит также отметить, что моя программа в качестве выходного результата дает не только рисунки деревьев, но также и фрагменты кода на языке программирования python:

```
def findDecision(obj):
   if obj[1]<=83:
```

```
        if obj[0] == 'Rain':
            if obj[3] == 'Weak':
                return 'Yes'
            if obj[3] == 'Strong':
                return 'No'
        if obj[0] == 'Overcast':
            return 'Yes'
        if obj[0] == 'Sunny':
            if obj[2]>65:
                if obj[3] == 'Strong':
                    return 'No'
                if obj[3] == 'Weak':
                    return 'No'
    if obj[1]>83:
        return 'No'
```

Реализовано это для того, чтобы упростить предсказание результата на основе обученной модели.

Для того чтобы сделать предсказание нам потребуется всего лишь загрузить одну строчку в метод.

Как показал анализ трёх реализованных алгоритмов построения дерева решений – каждый из рассмотренных алгоритмов походит для определенных типов задач. Так алгоритм ID3 подходит для наборов данных с категориальными признаками. Алгоритм C4.5 подходит как для наборов данных с категориальными признаками, так и для наборов данных с числовыми признаками. А алгоритм CART в свою очередь подходит как для работы с категориальными, так и с числовыми признаками, но может строить только бинарные деревья.

Цель выпускной квалификационной работы достигнута.

Заключение

В ходе выполнения дипломной работы на тему «Анализ и реализация алгоритмов построения дерева решений» был рассмотрен процесс осуществления алгоритмов построения дерева решений и их последующий анализ.

Цель работы заключалась в анализе и реализации алгоритма ID3, алгоритма C4.5, а также алгоритма CART.

Для достижения поставленной цели проанализированы основные принципы дерева решений, включающие в себя подробное изложение этапов построения.

С помощью существующих математических моделей реализованы алгоритм ID3, алгоритм C4.5, а также алгоритм CART для построения дерева решений. Алгоритмы разработаны на функциональном языке программирования Python в интегрированной свободной среде для создания приложений PyCharm.

Для работы с реализованными алгоритмами и анализа их эффективности использовалась операционная система Windows.

Как показал анализ трёх реализованных алгоритмов построения дерева решений – каждый из рассмотренных алгоритмов походит для определенных типов задач.

Так алгоритм ID3 подходит для наборов данных с категориальными признаками. Алгоритм C4.5 подходит как для наборов данных с категориальными признаками, так и для наборов данных с числовыми признаками. А алгоритм CART в свою очередь подходит как для работы с категориальными, так и с числовыми признаками, но может строить только бинарные деревья.

Список используемой литературы

1. Адельсон-Вельский Г. М., Ландис Е. М. Один алгоритм организации информации // Доклады АН СССР. - 1962. - Т. 146, № 2. - С. 263-266. URL: http://www.mathnet.ru/links/20762a931307aef02c107c3d8f6244b8/dan26964.pdf
2. Алгоритмы. Построение и анализ. Томас Кормен [и др.] - М.: «Вильямс», 2019. - 1328 с.
3. Бабенко М. А., Левин М. В. Введение в теорию алгоритмов и структур данных. — М.: МЦНМО. 2020. 144 с.
4. Вирт Н. Алгоритмы и структуры данных. - ДМК Пресс, 2011. - 272 с.
5. Вороновский Г. К., Махотило К. В., Петрашев С. Н., Сергеев С. А. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности. Х.: ОСНОВА, 1997. 112 с.
6. Гашев С. Н. Математические методы в биологии: анализ биологических данных в системе Statistica. — М.: Юрайт. 2020. 208 с.
7. Галушкин А. И. Нейрокомпьютеры. Учебное пособие. М.: Альянс, 2014. 528 с.
8. К. Шеннон. Работы по теории информации и кибернетике. М. Иностранная литература, 1963.
9. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск. - Диалектика-Вильямс, 2019. - 832 с.
10. Кравченко А. И. Анализ и обработка социологических данных. Учебник. — М.: КноРус. 2020. 498 с.
11. Левитин А. В. Алгоритмы. Введение в разработку и анализ. - М.: Вильямс, 2006. - 576 с.

12. Латыпова Р. Нейронные сети. М.: LAP Lambert Academic Publishing, 2012. 465 с.
13. Лутц М. Программирование на Python. Том 2. М.: Символ-плюс, 2013. 334 с.
14. Осовский. С. Нейронные сети для обработки информации. М.: Финансы и статистика, 2004. 343 с.
15. Форсье Д. Django. Разработка веб-приложений на Python. М.: Символ-плюс, 1979. 326 с.
16. Breiman, Leo, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.
17. Bruce Reed. The height of a random binary search tree, 2003. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.1289&rep=rep1&t ype=pdf
18. Daniel Sleator, Robert Tarjan. Self-Adjusting Binary Search Trees, 1985. URL: http://www.cs.cmu.edu/~sleator/papers/self-adjusting.pdf
19. Hovland, C. I. (1960). Computer simulation of thinking. American Psychologist, 15(11), 687-693.
20. Hunt, Earl B.; Janet Marin; Philip J. Stone (1966). Experiments in Induction. New York: Academic Press. ISBN 978-0-12-362350-8.
21. J. Ross Quinlan. C4.5: Programs for Machine learning. Morgan Kaufmann Publishers 1993.
22. Martinez Conrado, Roura Salvador. Randomized binary search trees, URL:https://www.researchgate.net/publication/220432012_Randomized_Binary_Search
23. Python 3.10.4 documentation. [Электронный ресурс] URL: https://docs.python.org/3/ (дата обращения 10.05.2022).
24. Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1):81-106.
25. Quinlan, J. Ross. C4.5: Programs for Machine learning. Morgan Kaufmann Publishers 1993.

26. Robert Tarjan. Data Structures and Networks Algorithms, 1987. URL: https://doc.lagout.
org/Others/Data%20Structures/Data%20Structures%20and%20
Network%20Algorithms%20%5BTarjan%201987-01-01%5D.pdf
27. Sabharwal N., Agrawal A. Hands-on Question Answering Systems with BERT: Applications in Neural Networks and Natural Language Processing. New York City: Apress, 2021. 184 с.

Приложение А

Программный код

```
import pandas as pd
import math
import numpy as np
import time
import imp
#------------------------


algorithm = "ID3" #ID3, C4.5, CART, Regression
from graphviz import Digraph
d = Digraph(comment=algorithm)
#------------------------
#parameters

num_of_trees = 1 #this should be a prime number
enableMultitasking = False

epochs = 10
learning_rate = 1

#------------------------
#Data set
# df = pd.read_csv("dataset/golf.txt") #nominal features and
target
df = pd.read_csv("dataset/golf2.txt") #nominal and numeric
features, nominal target
# df = pd.read_csv("dataset/golf3.txt") #nominal features and
numeric target
# df = pd.read_csv("dataset/golf4.txt") #nominal and numeric
features, numeric target
```

```
# df =
pd.read_csv("dataset/car.data",names=["buying","maint","doors","
persons","lug_boot","safety","Decision"])
```

Продолжение приложения А

```
# df = pd.read_csv("dataset/iris.data", names=["Sepal
length","Sepal width","Petal length","Petal width","Decision"])

#df = pd.read_csv("dataset/adaboost.txt")


dataset = df.copy()

#-------------------------


print(algorithm," Модель начала обучаться...")


#initialize a dictionary. this is going to be used to check
features numeric or nominal. numeric features should be
transformed to nominal values based on scales.

dataset_features = dict()

#-------------------------


def softmax(w):

    e = np.exp(np.array(w))

    dist = e / np.sum(e)

    return dist


def sign(x):

    if x > 0:

        return 1

    elif x < 0:

        return -1

    else:

        return 0
```

```
#Работает с числовыми колонками, Возвращает DataFrame, где в
калонке column_name ,будут добавлены знаки > и < в зависимости
от алгоритма и Gain

def processContinuousFeatures(df, column_name, entropy):
```

Продолжение приложения А

```
 unique_values = sorted(df[column_name].unique())

    #print(column_name,"->",unique_values)

    subset_gainratios = []; subset_gains = []; subset_ginis =
[]; subset_red_stdevs = []


    for i in range(0, len(unique_values)-1):

        threshold = unique_values[i]


        # Все элементы до threshold

        subset1 = df[df[column_name] <= threshold]

        # Все элементы после threshold

        subset2 = df[df[column_name] > threshold]


        # Кол-во элементов до threshold и после

        subset1_rows = subset1.shape[0]; subset2_rows =
subset2.shape[0]

        total_instances = df.shape[0] #subset1_rows+subset2_rows


        # Кол-во элементов в процентах До и После

        subset1_probability = subset1_rows / total_instances

        subset2_probability = subset2_rows / total_instances


        if algorithm == 'C4.5':

            threshold_gain = entropy -
subset1_probability*calculateEntropy(subset1) -
subset2_probability*calculateEntropy(subset2)

            subset_gains.append(threshold_gain)

            threshold_splitinfo = -subset1_probability *
math.log(subset1_probability, 2)-
subset2_probability*math.log(subset2_probability, 2)
```

```
            gainratio = threshold_gain / threshold_splitinfo
            subset_gainratios.append(gainratio)
```

Продолжение приложения А

```
 elif algorithm == 'CART':
            decision_for_subset1 =
subset1['Decision'].value_counts().tolist()
            decision_for_subset2 =
subset2['Decision'].value_counts().tolist()


            gini_subset1 = 1; gini_subset2 = 1


            for j in range(0, len(decision_for_subset1)):
                gini_subset1 = gini_subset1 -
math.pow((decision_for_subset1[j]/subset1_rows),2)


            for j in range(0, len(decision_for_subset2)):
                gini_subset2 = gini_subset2 -
math.pow((decision_for_subset2[j]/subset2_rows),2)


            gini = (subset1_rows/total_instances)*gini_subset1 +
(subset2_rows/total_instances) * gini_subset2


            subset_ginis.append(gini)
        #---------------------------------


    if algorithm == "C4.5":
        winner_one =
subset_gainratios.index(max(subset_gainratios))
    elif algorithm == "CART":
        winner_one = subset_ginis.index(min(subset_ginis))


    winner_threshold = unique_values[winner_one]
```

```
    #print("theshold is ",winner_threshold," for ",column_name)
    df[column_name] = np.where(df[column_name] <=
winner_threshold, "<="+str(winner_threshold),
">"+str(winner_threshold))
```

Продолжение приложения А

```
    return df

# Поиск энтропии
def calculateEntropy(df):

    # Кол-Во Строк и Столбцов
    instances = df.shape[0]; columns = df.shape[1]
    # Лист Всех Выриантов "результата" от выбранного DataFrame
    decisions = df['Decision'].value_counts().keys().tolist()

    entropy = 0

    for i in range(0, len(decisions)):
        decision = decisions[i]
        num_of_decisions =
df['Decision'].value_counts().tolist()[i]
        #print(decision,"->",num_of_decisions)

        class_probability = num_of_decisions/instances

        entropy = entropy -
class_probability*math.log(class_probability, 2)

    return entropy

def findDecision(df):
    entropy = calculateEntropy(df)
    #print("entropy: ",entropy)
```

```
    columns = df.shape[1]; instances = df.shape[0]
```

Продолжение приложения А

```
gains = []; gainratios = []; ginis = []; reducted_stdevs = []

    for i in range(0, columns-1):
        column_name = df.columns[i]
        column_type = df[column_name].dtypes

        if column_type != 'object' and algorithm !='ID3':
            df = processContinuousFeatures(df, column_name,
entropy)

        classes = df[column_name].value_counts()

        gain = entropy * 1; splitinfo = 0; gini = 0;
weighted_stdev = 0

        for j in range(0, len(classes)):
            current_class = classes.keys().tolist()[j]
            #print(column_name,"->",current_class)

            subdataset = df[df[column_name] == current_class]
            #print(subdataset)

            subset_instances = subdataset.shape[0]
            class_probability = subset_instances/instances

            if algorithm == 'ID3' or algorithm == 'C4.5':
                subset_entropy = calculateEntropy(subdataset)
```

```
                gain = gain - class_probability *
subset_entropy


            if algorithm == 'C4.5':
```

Продолжение приложения А

```
splitinfo = splitinfo -
class_probability*math.log(class_probability, 2)


            elif algorithm == 'CART':
                decision_list =
subdataset['Decision'].value_counts().tolist()
                subgini = 1


                for k in range(0, len(decision_list)):
                    subgini = subgini -
math.pow((decision_list[k]/subset_instances), 2)


                gini = gini + (subset_instances / instances) *
subgini


        #iterating over classes for loop end
        #-------------------------------


        if algorithm == "ID3":
            gains.append(gain)


        elif algorithm == "C4.5":


            if splitinfo == 0:
                splitinfo = 100 #this can be if data set
consists of 2 rows and current column consists of 1 class. still
decision can be made (decisions for these 2 rows same). set
splitinfo to very large value to make gain ratio very small. in
this way, we won't find this column as the most dominant one.
```

```
            gainratio = gain / splitinfo
            gainratios.append(gainratio)

        elif algorithm == "CART":
```

Продолжение приложения А

```
ginis.append(gini)

    #print(df)
    if algorithm == "ID3":
        winner_index = gains.index(max(gains))
    elif algorithm == "C4.5":
        winner_index = gainratios.index(max(gainratios))
    elif algorithm == "CART":
        winner_index = ginis.index(min(ginis))
    winner_name = df.columns[winner_index]

    return winner_name

def formatRule(root):
    resp = ''

    for i in range(0, root):
        resp = resp + '   '

    return resp

def storeRule(file,content):
        f = open(file, "a+")
        f.writelines(content)
        f.writelines("\n")
```

```
def createFile(file,content):
        f = open(file, "w")
        f.write(content)


def buildDecisionTree(df,root,file,prevIndex):
```

Продолжение приложения А

```
charForResp = "'"
    # Защищаем root от мутаций
    tmp_root = root * 1


    df_copy = df.copy()


    winner_name = findDecision(df)
    #find winner index, this cannot be returned by find decision
because columns dropped in previous steps
    j = 0
    for i in dataset_features:
        if i == winner_name:
            winner_index = j
        j = j + 1


    numericColumn = False
    if dataset_features[winner_name] != 'object' and algorithm
!='ID3':
        numericColumn = True


    #restoration
    columns = df.shape[1]
    for i in range(0, columns-1):
        column_name = df.columns[i]; column_type =
df[column_name].dtypes
        if column_type != 'object' and column_name !=
winner_name and algorithm !='ID3':
```

```
            df[column_name] = df_copy[column_name]

    classes = df[winner_name].value_counts().keys().tolist()

    for i in range(0,len(classes)):
```

Продолжение приложения А

```
current_class = classes[i]
        subdataset = df[df[winner_name] == current_class]
        subdataset = subdataset.drop(columns=[winner_name])

        if numericColumn == True:
            compareTo = current_class #current class might be
<=x or >x in this case
        else:
            compareTo = " == '"+str(current_class)+"'"
        #print(subdataset)

        terminateBuilding = False

        #-----------------------------------------------

        if len(subdataset['Decision'].value_counts().tolist())
== 1:
            final_decision =
subdataset['Decision'].value_counts().keys().tolist()[0] #all
items are equal in this case
            terminateBuilding = True
        elif subdataset.shape[1] == 1: #if decision cannot be
made even though all columns dropped
            final_decision =
subdataset['Decision'].value_counts().idxmax() #get the most
frequent one
            terminateBuilding = True
        #-----------------------------------------------
```

```
        #storeRule(file,(formatRule(root),"if
",winner_name,compareTo,":"))
        # ???
```

Продолжение приложения А

```
thisIndex = id(winner_index*1000  + id(winner_name) + id(root) +
id(compareTo) )
        if(prevIndex == 0):
            d.node(str((prevIndex)), str(winner_name))

        d.node(str(thisIndex), str(winner_name))

        d.edge( str(prevIndex), str(thisIndex), str(compareTo))
        storeRule(file,(formatRule(root),"if
obj[",str(winner_index),"]",compareTo,":"))

        #-----------------------------------------------

        if terminateBuilding == True: #check decision is made
            # ???
            d.node(str(thisIndex), str(final_decision))
            storeRule(file,(formatRule(root+1),"return
",charForResp+str(final_decision)+charForResp))
        else: #decision is not made, continue to create branch
and leafs
            root = root + 1 #the following rule will be included
by this rule. increase root
            buildDecisionTree(subdataset,root,file,thisIndex)

        root = tmp_root * 1

#-------------------------
```

```
if(True): #header of rules files
    header = "def findDecision("
    num_of_columns = df.shape[1]-1
    for i in range(0, num_of_columns):
```

Продолжение приложения А

```
column_name = df.columns[i]
        dataset_features[column_name] = df[column_name].dtypes

    header = header + "obj"

    header = header + "):\n"

#--------------------------
begin = time.time()

if enableMultitasking == False: #serial
    for i in range(0, num_of_trees):
        # Берем случайный кусочек из выборки равный
1\num_of_trees от всего массива
        subset = df.sample(frac=1/num_of_trees)
        root = 1

        file = "rule_"+str(i)+".py"

        createFile(file, header)

        buildDecisionTree(subset,root, file,0)
        d.render('dot.dot')

    print("завершено за ",time.time() - begin," секунд")
```

```
else: #parallel

    from multiprocessing import Pool

    subsets = []
```

Продолжение приложения А

```
    for i in range(0, num_of_trees):

        file = "rule_"+str(i)+".py"

        subset = df.sample(frac=1/num_of_trees)
        root = 1
        subsets.append((subset, root, file))

        createFile(file, header)

    if __name__ == '__main__': #windows returns expection if
this control is not applied for multitasking
        with Pool(num_of_trees) as pool:
            pool.starmap(buildDecisionTree, subsets,0)

        print("finished in ",time.time() - begin," seconds")
```