МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Допущен к защите

И.о. заведующего кафедрой ПМИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Разова

**классификатор на основе деревьев решений**

Курсовой проект по дисциплине  
«Проектная и научно-исследовательская деятельность»

Выполнил студент группы ФИб-2301-51-00     / А.С. Примаченко /

Руководитель к.т.н., доцент кафедры ПМИ     / А.В. Котельникова /

Работа защищена с оценкой     \_\_\_\_.\_\_\_\_.2020 г.

Члены комиссии:     /     /

    /     /

Киров 2020

Оглавление

[Введение 2](#_Toc43464111)

[Глава 1. Интеллектуальный анализ данных 4](#_Toc43464112)

[1.1 Основные понятия 4](#_Toc43464113)

[1.2 Постановка задачи 5](#_Toc43464114)

[1.3 Алгоритм ID3 6](#_Toc43464115)

[1.4 Выводы по главе 11](#_Toc43464116)

[Глава 2. Разработка программы 12](#_Toc43464117)

[2.1 Описание программы 12](#_Toc43464118)

[2.2 Выводы по главе 37](#_Toc43464119)

[Заключение 39](#_Toc43464120)

[Библиографический список 40](#_Toc43464121)

[Приложения 42](#_Toc43464122)

# Введение

Классификация – одна из основных задач машинного обучения. Ее целью является разделение объектов на классы согласно определенным признакам. Задача классификации – один из видов машинного обучения с учителем. Основная цель обучения с учителем состоит в том, чтобы на маркированных тренировочных данных извлечь модель, которая позволяет делать прогнозы о ранее не встречавшихся или будущих данных. В качестве обучающего набора используется множество объектов, для которых предварительно задана метка класса, представляющая собой дискретное, неупорядоченное значение. Задача классификации делится на бинарную и многоклассовую классификацию. Аналитические модели, решающие задачу классификации, называются классификаторами.

Одним из методов решения этой задачи являются деревья решений, представляющие собой иерархические древовидные структуры, состоящие из правил, которые генерируются в процессе обучения на обучающем наборе. Одним из преимуществ деревьев решений является то, что они легко интерпретируемы, так как они формулируются практически на естественном языке. Целью построения дерева решений является создание модели, с помощью которой можно классифицировать случаи и решить, какие значения может принимать целевая функция. Процесс построения заключается в последовательном, рекурсивном разбиении обучающего множества на подмножества на основе правил, лежащих в узлах дерева. Разбиение происходит до тех пор, пока возможно разбиение множества на более мелкие. Для достижения наилучшего результата необходимо разделить данные по признаку, который ведет к наибольшему приросту информации.

В настоящее время отсутствует сравнительный анализ алгоритмов классификатора на основе деревьев решений. Таким образом, проблемой исследования является недостаточная изученность алгоритмов создания деревьев.

Целью курсового проекта является программная реализация разработка алгоритма построения классификатора на основе дерева решений для разных наборов данных и оценка его точности.

Для достижения данной цели необходимо решить следующие задачи:

1. Обзор литературы по современным методам машинного обучения.
2. Выбор алгоритма и обучающих наборов.
3. Разработка классификатора на основе деревьев решений.
4. Экспериментальное исследование алгоритма.
5. Анализ полученных результатов.

Работа включает введение, две главы, заключение, список литературы и приложение. В первой главе описывается теоретическая часть и алгоритм построения дерева решений. Во второй главе содержится программная реализация алгоритма на языке C++ и анализ результатов работы программы на разных наборах данных.

# Глава 1. Интеллектуальный анализ данных

* 1. Основные понятия

Дерево решений — один из методов автоматического анализа данных. Решающие правила представляются в иерархической структуре, состоящей из элементов двух типов: узлов и листьев [13, с 145].

Атрибут – признак, характеризующий определенное свойство объекта.

Узел – внутренний узел дерева.

Корневой узел – начальный узел дерева решений.

Лист – конечный узел.

Решающее правило – условие в узле.

Пример дерева решений предоставлен на рисунке 1.1

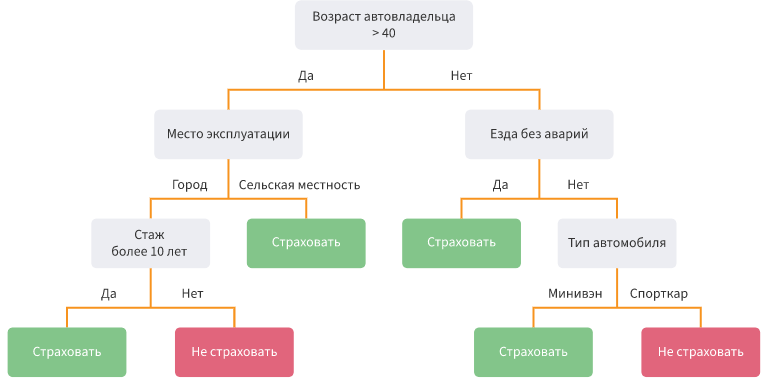


Рисунок 1.1 — Пример дерева решений

## Постановка задачи

Основными целями классификатора на основе деревьев решений являются:

1. Правильная классификация как можно большего количества данных из тестового набора.
2. Обобщение за пределами обучающей выборки так, чтобы невидимые образцы можно было классифицировать с максимально возможной точностью.
3. Легкое обновление по мере того, как становится доступным больше обучающих образцов.
4. Простая структура, насколько это возможно.

Построение классификатора может быть разбито на несколько частей [10, p 7]:

1. Выбор подходящей структуры дерева.
2. Выбор подмножеств объектов, которые будут использоваться в каждом внутреннем узле.
3. Выбор решающего правила или стратегии, которые будут использоваться на каждом внутреннем узле

При построении деревьев решений особое значение имеет выбор переменной, по которой выполняется разбитие множества на подмножества.

Для изложения формальной постановки задачи введем следующие обозначения.

– множество объектов, для каждого из которых задана метка класса и атрибутов .

Дерево строится путем расщепления множества на подмножества, основываясь на проверке значений атрибутов. Этот процесс прекращается, когда элементы подмножества в узле относятся к одному и тому же классу. При формировании решающего правила необходимо выбрать атрибут, по которому будет происходить разбиение. Выбранный атрибут должен разделить множество так, чтобы элементы подмножеств, полученных в результате расщепления, имели одинаковую метку класса или стремились к этому [11, с 112].

Различные алгоритмы используют различные способы выбора данной переменной. В данной работе будет рассмотрен алгоритм ID3, в котором используются понятия информационного выигрыша и энтропии [8, с 121].

Энтропия: (1.1)

Информационный выигрыш: (1.2)

– энтропия множества до расщепления

– энтропия после разбиения по атрибуту a.

Вычислив информационный выигрыш для каждого атрибута, алгоритм выбирает признак с наибольшим значением.

## Алгоритм ID3

### Псевдокод

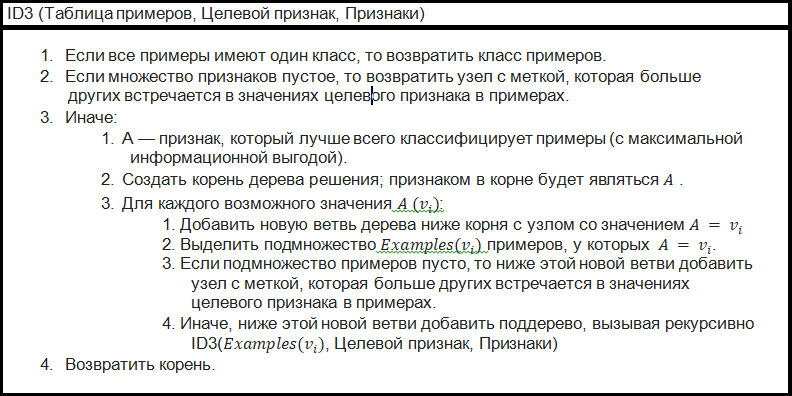


Рисунок 1.2 — алгоритм построения дерева решений ID3

### Пример работы

Алгоритм ID3 использует методику “разделяй и властвуй”, где происходит рекурсивное разбиение обучающей выборки на подмножества, содержащие объекты одного класса.

Этот алгоритм жадный, так как на каждом шаге выбирает атрибут с максимальным значением и не учитывает последующие решения [1, p 11]. Недостатком алгоритма является склонность к переобучению, особенно если разнообразие значений атрибута велико. Так же не предусмотрена возможность работы с отсутствующими данными. Позднее алгоритм был доработан, новая версия получила название C4.5.

Рассмотрим работу алгоритма на простом примере:

В таблице 1.1 приведен набор данных, где “наблюдение”, “температура”, “влажность” и “ветер” — признаки, а “игра” — метка класса.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наблюдение | Температура | Влажность | Ветер | Игра |
| Солнце | Жарко | Высокая | Нет | Нет |
| Солнце | Жарко | Высокая | Есть | Нет |
| Облачность | Жарко | Высокая | Нет | Да |
| Дождь | Норма | Высокая | Нет | Да |
| Дождь | Холодно | Норма | Нет | Да |
| Дождь | Холодно | Норма | Есть | Нет |
| Облачность | Холодно | Норма | Есть | Да |
| Солнце | Норма | Высокая | Нет | Нет |
| Солнце | Холодно | Норма | Нет | Да |
| Дождь | Норма | Норма | Нет | Да |
| Солнце | Норма | Норма | Есть | Да |
| Облачность | Норма | Высокая | Есть | Да |
| Облачность | Жарко | Норма | Нет | Да |
| Дождь | Норма | Высокая | Есть | Нет |

Таблица 1.1 — Используемый набор данных

Для выбора признака, по которому будет проходить разбиение набора данных , необходимо подсчитать значение информационного выигрыша для каждой признака по формуле (1.2). Сначала нужно вычислить энтропию набора данных, находящихся на данном узле по формуле (1.2), затем для каждого признака рассчитать энтропию подмножеств, разделенных по этому признаку.

В данном случае .

Теперь считаем энтропию для каждого признака .

По формуле (1.2) находим информационный выигрыш для каждого признака.

0.247 бит.

0.029 бит.

0.016 бит.

0.048 бит.

Максимальное значение получено для признака погода, следовательно по нему и будет происходить расщепление. Таким образом дерево будет расти до тех пор, пока в его узлах не будут находиться элементы подмножества с одинаковой меткой класса.

### Регулирование глубины дерева

На некоторых задачах возможна генерация слишком большого дерева, когда не удается найти паттерн. Это называется переобучением дерева, что может отрицательно сказаться на способности классификатора правильно работать с новыми данными [12, с 28]. Хорошая модель должна не только правильно работать на обучающем наборе, но и точно классифицировать данные, которые раньше не видела [2, p 6].

Для того чтобы предотвратить переобучение дерева решений, существуют два способа: ранняя остановка и отсечение ветвей.

Ранняя остановка привлекательная в плане экономии времени обучения, но может сказаться на точности классификации. Вместо ранней остановки рекомендуется делать отсечение ветвей [11, c 113].

Отсечение ветвей происходит снизу вверх после того, как было построено полное дерево. Каждый узел дерева является кандидатом на удаление. Если это не приведет к существенному возрастанию ошибки, он убирается и заменяется листом. Ценность правила, справедливого для 2-3 примеров, крайне низка, и в целях анализа данных такое правило практически непригодно. В большинстве случаев отсечение дает хорошие результаты, что позволяет говорить о правомерности данной методики. [11, c 113-114]

### Временная сложность

Максимально возможное количество листов в узле равно — количество примеров в тренировочном наборе. Максимальная длина пути от корня до каждого листа равна , где — количество признаков.

Так количество узлов в дереве решений всегда меньше, чем .

В корне алгоритм должен проверить значение каждого примера по каждому признаку . Временная сложность этого действия — , где — максимальное количество возможных значений признака. Временная сложность каждого узла всегда меньше, чем сложность корня.

Следовательно, в худшем случае временная сложность алгоритма ID3 — [Knowledge Acquisition from Databases]

## Выводы по главе

Таким образом, в первой главе были рассмотрены основные понятия, связанные с задачей классификации, и формальная постановка задачи,

Так же был описан алгоритм построения деревьев решений ID3 и его псевдокод. Были рассмотрены пример его работы, проблема переобучения дерева решений и временная сложность.

Реализация данного алгоритма будет осуществляться во второй главе с помощью средств языка программирования C++.

# Глава 2. Разработка программы

## Описание программы

### Входные данные

Пользователь решает, какой набор данных он хочет использовать для построения дерева решений.

Так же пользователю дается возможность сделать раннюю остановку или отсечение ветвей.

Возможными наборами являются:

1. Congressional Voting Records Data Set [4] — набор данных, содержащий 435 записей голосов Палаты представителей Конгрессменов 1984 года. Шестнадцать дискретных признаков, два класса. Определяет партию проголосовавшего на основе его голосов.
2. Breast Cancer Wisconsin (Original) Data Set [3] — содержит 699 записей результатов анализов женщин на рак груди. Девять непрерывных признаков, два класса. Делит опухоли на злокачественные и доброкачественные.
3. Ionosphere Data Set [6] — содержит 351 запись данных, полученных радаром системы, анализирующей ионосферу. 34 непрерывных признака, 2 класса. Делит собранные результаты на хорошие и плохие.
4. Mushroom Data Set [9] — содержит 8124 записей о видах грибов и их особенностях. Каждый вид был помечен как съедобный или несъедобный. Двадцать два дискретных признака, два класса. Определяет, съедобен ли гриб.
5. Iris Data Set [7] — содержит описания 150 цветков ириса, 50 цветков каждого подвида. Четыре непрерывных признака, три класса. Определяет подвид ириса.
6. Divorce Predictors Data Set [5] — содержит 170 результатов данных опроса об отношениях в семье. 54 непрерывных признака, 2 класса. Определяет, разведется ли пара.

### Программная реализация

Алгоритм программы представлен в блок-схеме на рисунке 2.1

В программе используются две структуры:

1. line, в которой содержится одна строка из входного файла. В ней хранятся значения признаков и метка класса.
2. node, содержит весь входной файл. В ней хранятся набор данных, атрибут, по которому происходит разбиение, ссылка на предка и потомков.

Обработка входных файлов происходит с помощью функции read\_data.

Используемые функции:

1. count\_att — вспомогательная функция, используется при считывании данных для разделения строки;



Рисунок 2.1

1. read\_data — обрабатывает входной файл, разбивает строку на значения признаков и класс. Записывает полученное в вектор data, состоящий из line.
2. find\_types — находит классы;
3. instances\_of\_each\_type — считает, сколько элементов каждого класса находится в подмножестве;
4. entropy — вычисляет энтропию;
5. find\_split\_point — считает информационный выигрыш для каждого признака, выбирает наилучший;
6. unite\_leaves — вспомогательная функция, используется при ранней остановке. Объединяет листья с одинаковой меткой класса;
7. split\_pre — делит множество на подмножество с учетом ранней остановки;
8. split\_nopre — делит множество на подмножество, если ранняя остановка не используется;
9. node\_print — выводит узел дерева;
10. classify — классифицирует строку из тестового набора данных;
11. prune — отсекает поддеревья;
12. leaves — используется при отсечение деревьев с дискретным набором данных.

Пользователь выбирает набор данных, на котором он хочет построить дерево. Вызывается программа, работающая с этим набором. Пользователя спрашивают, хочет ли он использовать раннюю остановку и отсечение. Если отсечение не используется, то считывается файл, содержащий тестовый набор, и файл, содержащий подтверждающий набор, который используется для отсечения ветвей.

Начинает строиться дерево.

Если используется ранняя остановка, то дерево будет расти, пока размер узла больше определенного числа. Если не используется, то до тех пор, пока в каждом узле элементы множества не будут принадлежать к одному классу. Для каждого признака вычисляется информационный выигрыш, если работа происходит с непрерывным набором данных, то вычисляется еще и значение признака, по которому будет проходить разбиение. Узел разбивается на несколько подузлов, затем каждый из них тоже разбивается.

Если отсечение не используется, то на данном этапе происходит подсчет точности дерева при помощи тестового набора. Выводится точность и полученное дерево. Если отсечение используется, то сверху вниз происходит обход дерева, пока не будет найдет узел, чьи потомки — все листья. Высчитывается точность дерева, если убрать этот узел. Для подсчета точности используется подтверждающий набор. Если точность значительно не уменьшилась, то узел заменяется на лист, меткой класса данного узла становится метка класса потомка с наибольшим количеством элементов. Если же удаление узла привело к возрастанию ошибки, данный узел остается. Программа возвращается к предыдущему узлу. Отсечение происходит, пока есть возможность. Когда больше нечего отсекать, происходит вычисление точности дерева по тестовому набору. Выводится полученная точность и построенное дерево.

Листинг основной программы приведен в приложении А.

### Условия эксперимента

Дано 6 наборов данных, каждый из которых разделен на тренировочный, подтверждающий и тестовый. Подтверждающий и тестовый наборы созданы случайным образом. Если отсечение не используется, то дерево строится на данных из тренировочного и подтверждающего наборов.

Все результаты тестов на определение времени, требующегося для построения дерева, были получены с помощью функции clock. Время, потраченное пользователем на ввод, не учитывается. Код:

double start\_time = clock();

double end\_time = clock();

double search\_time = (end\_time - start\_time) / (double)CLOCKS\_PER\_SEC;

cout << search\_time;

Тестирование алгоритма производится с помощью ноутбука на процессоре Intel Pentium n4200 с частотой 1,10 GHz. Оперативная память 8 ГБ, жесткий диск HDD объемом 256 ГБ.

### Результаты работы программы

Здесь предоставлены деревья, полученные в результате работы программы.

1. Congressional Voting Records Data Set

Листинг программы приведен в приложении Б.

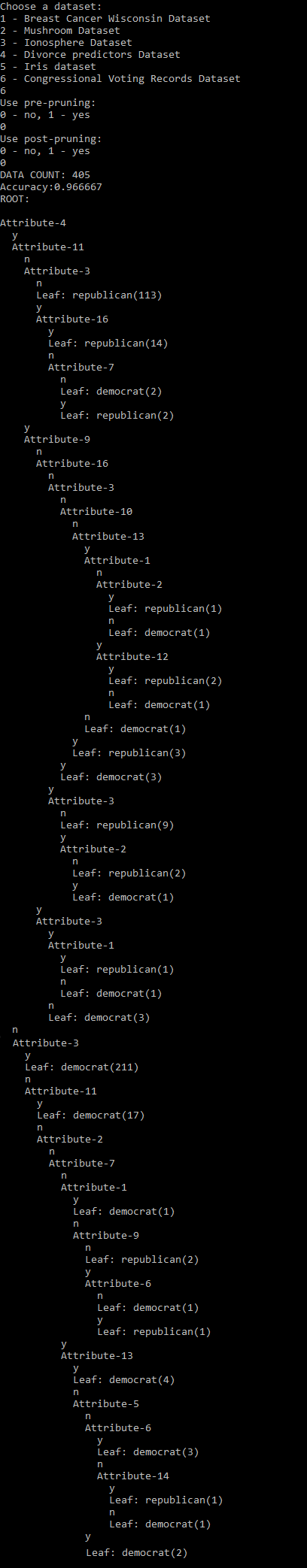


Рисунок 2.2 – результат работы программы без отсечения и остановки

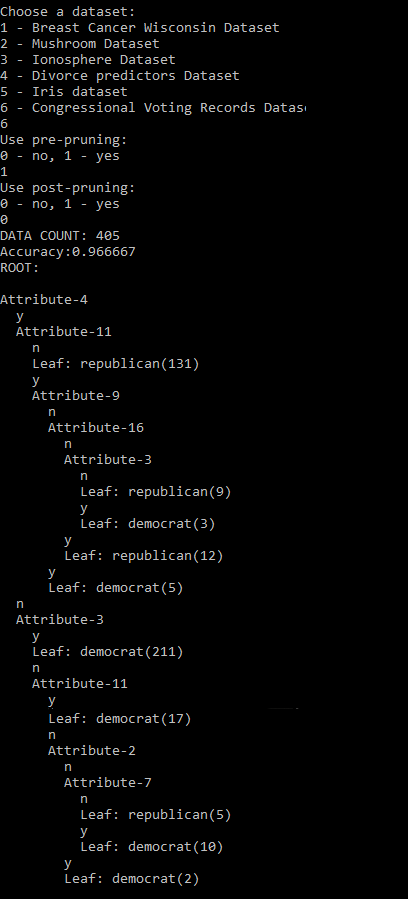


Рисунок 2.3 – Результат работы программы с ранней остановкой

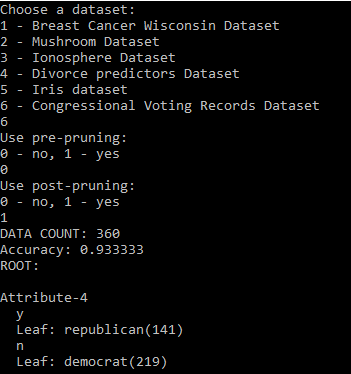


Рисунок 2.4 – результат работы программы с отсечением

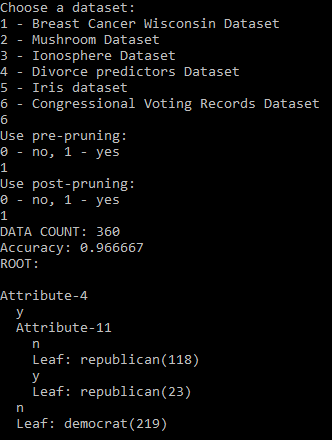


Рисунок 2.5 – результат работы программы с отсечением и остановкой

1. Breast Cancer Wisconsin (Original) Data Set

Результаты работы и листинг программы приведены в приложении В.

1. Ionosphere Data Set

Результаты работы и листинг программы приведены в приложении Г.

1. Mushroom Data Set

Результаты работы и листинг программы приведены в приложении Д.

1. Iris Data Set

Результаты работы и листинг программы приведены в приложении Е.

1. Divorce Predictors Data Set

Результаты работы и листинг программы приведены в приложении Ж.

Использование ранней остановки или отсечения уменьшает глубину дерева, что визуально его и делает более легко воспринимаемым и понимаемым.

### Экспериментальное исследование

В таблице 2.1 приведены результаты работы программы для каждого набора данных в 4 случаях: базовый без отсечения и ранней остановки, только с ранней остановкой, только с отсечением, с отсечением и остановкой.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Набор данных | | Размер обучающего набора | Размер подтверждающего набора | Размер тестового набора | Точность  (округленная) | Время работы  в секундах |
| Vote | Базовый | 360 | 45 | 30 | **0.97** | 14.90 |
| Остановка | **0.97** | 11.20 |
| Отсечение | 0.94 | 10.94 |
| Остановка + отсечение | **0.97** | 9.41 |
| Cancer | Базовый | 570 | 59 | 70 | 0.97 | 325.64 |
| Остановка | 0.99 | 324.23 |
| Отсечение | 0.99 | 273.62 |
| Остановка + отсечение | **1.00** | 265.71 |
| Ionosphere | Базовый | 270 | 30 | 43 | 0.88 | 377.07 |
| Остановка | 0.81 | 350.93 |
| Отсечение | **0.93** | 289.30 |
| Остановка + отсечение | 0.84 | 286.02 |
| Mushroom | Базовый | 7900 | 100 | 124 | **1.00** | 138.35 |
| Остановка | **1.00** | 137.99 |
| Отсечение | **1.00** | 141.16 |
| Остановка + отсечение | **1.00** | 136.16 |
| Iris | Базовый | 90 | 30 | 30 | 0.94 | 39.09 |
| Остановка | 0.94 | 33.04 |
| Отсечение | **0.97** | 19.25 |
| Остановка + отсечение | 0.94 | 17.79 |
| Divorce | Базовый | 100 | 40 | 30 | 1.00 | 67.56 |
| Остановка | 1.00 | 68.05 |
| Отсечение | **0.97** | 30.94 |
| Остановка + отсечение | **0.97** | 31.16 |

Таблица 2.1 — Результаты работы

В таблице жирным выделены результаты с наибольшей точностью. Лишь в одном случае использование отсечения привело к незначительному увеличению ошибки.

По времени работы программ можно сделать вывод, что для построения дерево по набору данных с непрерывными значениями требуется больше времени, так как необходимо не только выбрать признак, по которому будет происходить разделение, но и его значение.

Это, а также тот факт, что отсечение визуально упрощает дерево, позволяет нам сказать, что в данных случаях использование этого метода улучшило изначальный результат и решило проблему переобучения.

## Выводы по главе

Таким образом, во второй главе была разработана программа, создающая классификатор на основе деревьев решений.

Согласно полученным данным, методы регулирования глубины упрощают дерево без значительной потери точности. Можно сделать вывод, что полученные деревья работают точно не только на тренировочных данных, но и правильно классифицируют новые.

Основной причиной, по которой отсечение ветвей часто улучшает правильность классификации на будущих примерах, является удаление низкоуровневых тестов, что обычно снижает возможность переобучения.

# Заключение

В ходе данной работы был рассмотрен алгоритм ID3 и его модификации, позволяющие улучшить его точность и упростить полученное в результате работы алгоритма дерево. Так же была разработана программа на языке C++, позволяющая пользователю выбрать, какой набор данных использовать, и решить, нужно ли применять модификации.

Оригинальный алгоритм ID3 для каждого признака вычисляет информационный выигрыш и выбирает наилучший. Этот процесс происходит до тех пор, пока все элементы множества не окажутся одного класса.

Методы регулирования глубины дерева как, показали полученные результаты, улучшают не только точность классификатора, но и скорость работы алгоритма, что особенно актуально для работы с непрерывными наборами данных.

По результатам данной работы можно сделать вывод, существуют методы, позволяющие увеличить корректность классификации для различных наборов данных без потерь во времени.

Таким образом, все задачи курсового проекта были выполнены, а цель достигнута.

# Библиографический список

1. Andriy Burkov. The Hundred-Page Machine Learning Book — Andriy Burkov, 2019 — 160 p.
2. Awad M., Khanna R. Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers — Apress Media, 2015 — 264 p.
3. Breast Cancer Wisconsin (Original) Data Set — URL: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29
4. Congressional Voting Records Data Set — URL: https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records
5. Divorce Predictors Data Set — URL: https://archive.ics.uci.edu/ml/datasets/Divorce+Predictors+data+set
6. Ionosphere Data Set — URL: https://archive.ics.uci.edu/ml/datasets/Ionosphere
7. Iris Data Set — URL: https://archive.ics.uci.edu/ml/datasets/iris
8. Miroslav Kubat. An Introduction to Machine Learning — Springer International Publishing Switzerland, 2015 — 291 p.
9. Mushroom Data Set — URL: https://archive.ics.uci.edu/ml/datasets/Mushroom
10. S. Rasoul Safavian, David Landgrebe. A Survey of Decision Tree Classifier Methodology — Purdue University, 1990 — 46 p.
11. Барсегян, А. А. Анализ данных и процессов — Санкт-Петербург: БХВ-Петербург, 2009 — 512 с.
12. Кафтанников И.Л., Парасич А.В. Особенности применения деревьев решений в задачах классификации — Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника», 2015 — с 26-32.
13. Флах П. Машинное обучение, Наука и искусство построения алгоритмов, которые извлекают знания из данных – Москва: ДМК Пресс, 2015 – 400 с.

# Приложения

**Приложение А**

Main.cpp

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

double start\_time = clock();

int set = 0;

cout << "Choose a dataset:" << endl;

cout << "1 - Breast Cancer Wisconsin Dataset" << endl;

cout << "2 - Mushroom Dataset" << endl;

cout << "3 - Ionosphere Dataset" << endl;

cout << "4 - Divorce predictors Dataset" << endl;

cout << "5 - Iris dataset" << endl;

cout << "6 - Congressional Voting Records Dataset" << endl;

cin >> set;

switch (set) {

case 1: system("C:\\Users\\User\\Documents\\MVS\\курсовая\\Main\\Debug\\MainCW.exe");

break;

case 2: system("C:\\Users\\User\\Documents\\MVS\\курсовая\\Main\\Debug\\MainMushroom.exe");

break;

case 3: system("C:\\Users\\User\\Documents\\MVS\\курсовая\\Main\\Debug\\MainIonosphere.exe");

break;

case 4: system("C:\\Users\\User\\Documents\\MVS\\курсовая\\Main\\Debug\\MainDivorce.exe");

break;

case 5: system("C:\\Users\\User\\Documents\\MVS\\курсовая\\Main\\Debug\\MainIris.exe");

break;

case 6: system("C:\\Users\\User\\Documents\\MVS\\курсовая\\Main\\Debug\\MainVote.exe");

break;

default:

cout << "Choose a number from 1 to 6" << endl;

break;

}

double end\_time = clock();

cout << endl;

cout << (end\_time - start\_time) / (double)CLOCKS\_PER\_SEC;

}

Приложение Б

Vote.h

#include <iostream>

#include <string>

#include <vector>

#include <math.h>

using namespace std;

struct line {

vector <string> attributes;

string type;

line(string& t, vector<string>& att) {

this->type = t;

this->attributes = att;

}

void print() {

int i = 0;

while (i < attributes.size()) {

cout << attributes[i] << ", ";

++i;

}

cout << type << endl;

}

};

int count\_att(string str) {

int i = 0;

int count = 0;

if (str.size() == 0)

throw - 1;

while (i < str.size()) {

if (str[i] == ',') { ++count; }

++i;

}

return count;

}

bool unique(vector<string> temp, string a) {

for (int i = 0; i < temp.size(); ++i) {

if (a == temp[i]) {

return false;

break;

}

}

}

vector<string> count\_att\_types(vector<line> data, int j) {

int i = 0;

vector <string> temp;

while (i < data.size()) {

if (unique(temp, data[i].attributes[j])) temp.push\_back(data[i].attributes[j]);

++i;

}

return temp;

}

void read\_data(vector<line>& data, string& str, int& count\_comma) {

string temp\_s = "";

vector<string> temp\_a;

int r = 0;

while (str[r] != ',') {

if (str[r] == 'r' && temp\_s == "") temp\_s = "republican";

else if (str[r] == 'd' && temp\_s == "") temp\_s = "democrat";

++r;

}

int i = r + 1;

int j = 1;

while (i < str.size()) {

string temp;

while (str[i] != ',' && i != str.size()) {

temp.push\_back(str[i]);

++i;

}

if (i == str.size() || str[i] == ',') {

if (temp == "?") temp\_a.push\_back("n");

else temp\_a.push\_back(temp);

++i;

}

++j;

}

data.push\_back(line(temp\_s, temp\_a));

}

vector<string> find\_types(vector<line> data) {

if (data.size() == 0) {

return vector<string>();

}

vector<string> unique;

unique.push\_back(data[0].type);

int i = 0;

bool type2 = false;

while (i < data.size()) {

if (data[0].type != data[i].type) {

unique.push\_back(data[i].type);

break;

}

++i;

}

return unique;

}

struct node {

vector <line> data;

vector <string> unique\_types;

string decision;

string a\_label;

string label;

int data\_size;

double compare;

int attribute;

node\* parent = nullptr;

vector <node\*> children;

int level;

node(vector<line> d, vector<string> u\_t) {

this->data = d;

this->unique\_types = u\_t;

this->decision = "Root";

this->a\_label = "";

this->level = 0;

this->data\_size = 0;

}

~node() {

}

vector<int> instances\_of\_each\_type(vector<line> d) {

int type1 = 0, type2 = 0, i = 0;

while (i < d.size()) {

if (d[i].type == this->unique\_types[0]) type1++;

else if (d[i].type == this->unique\_types[1]) type2++;

++i;

}

return vector<int>{type1, type2};

}

double entropy(vector<line> data) {

double e = 0, p = 0;

vector<int> type\_count = instances\_of\_each\_type(data);

for (int i = 0; i < type\_count.size(); ++i) {

if (type\_count[i] != 0) {

p = (double)type\_count[i] / (double)data.size();

e += -p \* log2(p);

}

}

return e;

}

double find\_split\_point(vector <line> data) {

double gain = 0, cutoff = 0, attribute\_cutoff = 0;

for (int i = 0; i < data[0].attributes.size(); ++i) {

vector<string> temp\_cutoff = count\_att\_types(data, i);

vector< vector<line>>child(temp\_cutoff.size());

for (int k = 0; k < data.size(); ++k) {

for (int j = 0; j < temp\_cutoff.size(); ++j) {

if (data[k].attributes[i] == temp\_cutoff[j]) child[j].push\_back(data[k]);

}

}

double e1 = entropy(data);

vector<double>e(temp\_cutoff.size());

for (int k = 0; k < e.size(); ++k) {

e[k] = entropy(child[k]);

}

double temp\_gain = 0;

for (int k = 0; k < e.size(); ++k) {

temp\_gain = temp\_gain + (double)child[k].size() / (double)data.size() \* e[k];

}

temp\_gain = e1 - temp\_gain;

if (temp\_gain > gain) {

attribute\_cutoff = i;

gain = temp\_gain;

}

}

return attribute\_cutoff;

}

void unite\_leaves(node\* leaf) {

if (leaf->parent->children[0]->label == leaf->parent->children[1]->label) {

leaf->parent->label = leaf->parent->children[0]->label;

leaf->parent->attribute = -1;

leaf->parent->children.clear();

for (int i = 0; i < leaf->parent->data.size(); ++i) {

leaf->parent->data[i].type = leaf->label;

}

unite\_leaves(leaf->parent);

}

}

void split\_pre() {

{

if (data.size() > 10) {

vector<int> type\_count = instances\_of\_each\_type(this->data);

int f = 0;

for (int i = 0; i < type\_count.size(); ++i) if (type\_count[i] != 0) f++;

if (f <= 1) {

attribute = -1;

label = data[0].type;

return;

}

int split\_point = find\_split\_point(this->data);

vector <string> cutoff = count\_att\_types(data, split\_point);

vector< vector<line>>child(cutoff.size());

for (int i = 0; i < data.size(); ++i) {

for (int j = 0; j < cutoff.size(); ++j) {

if (data[i].attributes[split\_point] == cutoff[j]) child[j].push\_back(data[i]);

}

}

string s\_temp = ("Attribute-" + to\_string(split\_point + 1));

this->decision = s\_temp;

attribute = split\_point;

for (int i = 0; i < child.size(); ++i) {

children.resize(child.size());

children[i] = new node(child[i], this->unique\_types);

children[i]->a\_label = cutoff[i];

children[i]->parent = this;

children[i]->level = this->level + 1;

}

for (int i = 0; i < children.size(); ++i) children[i]->split\_pre();

}

else {

vector<int> type\_count = instances\_of\_each\_type(this->data);

if (type\_count[0] >= type\_count[1]) {

attribute = -1;

label = unique\_types[0];

for (int i = 0; i < data.size(); ++i) {

data[i].type = label;

}

}

else {

attribute = -1;

label = unique\_types[1];

for (int i = 0; i < data.size(); ++i) {

data[i].type = label;

}

}

unite\_leaves(this);

}

}

}

void split\_nopre() {

vector<int> type\_count = instances\_of\_each\_type(this->data);

int f = 0;

for (int i = 0; i < type\_count.size(); ++i) if (type\_count[i] != 0) f++;

if (f <= 1) {

attribute = -1;

label = data[0].type;

return;

}

int split\_point = find\_split\_point(this->data);

vector <string> cutoff = count\_att\_types(data, split\_point);

this->data\_size = data.size();

vector< vector<line>>child(cutoff.size());

for (int i = 0; i < data.size(); ++i) {

for (int j = 0; j < cutoff.size(); ++j) {

if (data[i].attributes[split\_point] == cutoff[j]) child[j].push\_back(data[i]);

}

}

string s\_temp = ("Attribute-" + to\_string(split\_point + 1));

this->decision = s\_temp;

attribute = split\_point;

for (int i = 0; i < child.size(); ++i) {

children.resize(child.size());

children[i] = new node(child[i], this->unique\_types);

children[i]->a\_label = cutoff[i];

children[i]->level = this->level + 1;

children[i]->parent = this;

}

for (int i = 0; i < children.size(); ++i) children[i]->split\_nopre();

}

void node\_print() {

string space = "";

for (int i = 0; i < this->level; ++i) {

space += " ";

}

if (children.size() == 0) {

vector <int> type\_count = instances\_of\_each\_type(data);

cout << space << this->a\_label << endl;

if (type\_count[0] != 0) {

cout << space << "Leaf: " << unique\_types[0] << "(" << type\_count[0] << ")" << endl;

}

if (type\_count[1] != 0) {

cout << space << "Leaf: " << unique\_types[1] << "(" << type\_count[1] << ")" << endl;

}

return;

}

cout << space << this->a\_label << endl;

cout << space << decision << endl;

for (int i = 0; i < children.size(); ++i) {

children[i]->node\_print();

}

}

};

bool leaves(node\* node) {

int leave = 0;

for (int i = 0; i < node->children.size(); ++i) {

if (node->children[i]->decision == "Root") leave++;

}

if (leave == node->children.size()) return true;

else return false;

}

bool classify(line data, node\* att) {

bool correct = false;

if (att->attribute != -1) {

for (int i = 0; i < att->children.size(); ++i) {

if (data.attributes[att->attribute] == att->children[i]->a\_label) correct = classify(data, att->children[i]);

}

}

else {

if (data.type == att->label) return true;

else return false;

}

return correct;

}

int prune(node\* node\_p, node\* root, vector <line> test, double accuracy) {

int change = 0;

double new\_acc = 0;

int att;

string l, d, a\_l;

int max = 0;

vector<node\*>c;

if (leaves(node\_p) && node\_p->children.size() != 0) {

string leaf;

vector<line>data = node\_p->data;

att = node\_p->attribute;

l = node\_p->label;

d = node\_p->a\_label;

vector <int> size;

c.resize(node\_p->children.size());

for (int i = 0; i < node\_p->children.size(); ++i) {

c[i] = node\_p->children[i];

if (node\_p->children[i]->data.size() >= max) {

max = node\_p->children[i]->data.size();

leaf = node\_p->children[i]->label;

}

}

node\_p->attribute = -1;

node\_p->label = leaf;

for (int i = 0; i < node\_p->data.size(); ++i) node\_p->data[i].type = leaf;

node\_p->children.clear();

int correct = 0, incorrect = 0;

for (int i = 0; i < test.size(); ++i) {

if (classify(test[i], root) == true) correct++;

else incorrect++;

}

new\_acc = (double)correct / (double)test.size();

if (accuracy - new\_acc >= 0.1) {

node\_p->children = c;

node\_p->attribute = att;

node\_p->label = l;

node\_p->data = data;

node\_p->a\_label = d;

}

else {

accuracy = new\_acc;

change++;

node\_p->decision = "Root";

return change;

}

}

else {

for (int i = 0; i < node\_p->children.size(); ++i) change += prune(node\_p->children[i], root, test, accuracy);

return change;

}

}

MainVote.cpp

#include "Vote.h"

#include <fstream>

int main()

{

ifstream file("C:\\Users\\User\\Documents\\MVS\\курсовая\\datasets\\vote.txt");

int comma\_count = 0;

vector <line> data;

vector <line> test;

vector <line> validation;

int pre = 0, post = 0;

cout << "Use pre-pruning:" << endl;

cout << "0 - no, 1 - yes" << endl;

cin >> pre;

if (pre > 1) pre = 1;

cout << "Use post-pruning:" << endl;

cout << "0 - no, 1 - yes" << endl;

cin >> post;

if (post > 1) post = 1;

if (!file.is\_open()) printf("Could not open the file\n");

else {

string line;

bool done = false;

while (getline(file, line)) {

if (line != "") {

if (!done) {

comma\_count = count\_att(line);

done = true;

}

read\_data(data, line, comma\_count);

}

}

if (post == 0) {

ifstream file1("C:\\Users\\User\\Documents\\MVS\\курсовая\\datasets\\vote\_prune.txt");

while (getline(file1, line)) {

if (line != "") {

if (!done) {

comma\_count = count\_att(line);

done = true;

}

read\_data(data, line, comma\_count);

}

}

}

cout << "DATA COUNT: " << data.size() << endl;

node\* root = new node(data, find\_types(data));

if (pre == 0) root->split\_nopre();

else root->split\_pre();

file.close();

ifstream file("C:\\Users\\User\\Documents\\MVS\\курсовая\\datasets\\vote\_test.txt");

int correct = 0;

int incorrect = 0;

if (!file.is\_open()) printf("Could not open the file\n");

else {

string line;

bool done = false;

double acc;

while (getline(file, line)) {

if (line != "") {

if (!done) {

comma\_count = count\_att(line);

done = true;

}

read\_data(test, line, comma\_count);

}

}

for (int i = 0; i < test.size(); ++i) {

if (classify(test[i], root) == true) correct++;

else incorrect++;

}

acc = (double)correct / (double)test.size();

if (post == 0) {

cout << "Accuracy:" << acc << endl;

cout << "ROOT:" << endl;

root->node\_print();

}

else {

correct = 0; incorrect = 0;

ifstream file2("C:\\Users\\User\\Documents\\MVS\\курсовая\\datasets\\vote\_prune.txt");

if (!file2.is\_open()) printf("Could not open the file\n");

else {

string line;

bool done = false;

while (getline(file2, line)) {

if (line != "") {

if (!done) {

comma\_count = count\_att(line);

done = true;

}

read\_data(validation, line, comma\_count);

}

}

}

while (prune(root, root, validation, acc) != 0) prune(root, root, validation, acc);

for (int i = 0; i < test.size(); ++i) {

if (classify(test[i], root) == true) correct++;

else incorrect++;

}

cout << "Accuracy: " << acc << endl;

cout << "ROOT:" << endl;

root->node\_print();

}

}

}

}

Приложение В

CW.h

Код во многом аналогичен коду из Приложения А, Vote.h  
Ниже преставлены функции, в которых есть отличия.

#include <iostream>

#include <string>

#include <vector>

#include <math.h>

using namespace std;

void read\_data(vector<line>& data, string& str, int& count\_comma) {

string temp\_s;

vector<double> temp\_a;

int i = 0;

int j = 0;

while (j < count\_comma) {

string temp;

while (str[i] != ',') {

temp.push\_back(str[i]);

++i;

}

if (str[i] == ',') {

if (temp == "?") temp\_a.push\_back(5);

else temp\_a.push\_back(stod(temp));

++i;

}

++j;

}

temp\_s = str.substr(i);

data.push\_back(line(temp\_a, temp\_s));

}

vector <double> find\_split\_point(vector <line> data) {

double gain = 0, cutoff = 0, attribute\_cutoff = 0;

vector <line> smaller;

vector <line> bigger;

for (int i = 1; i < data[0].attributes.size(); ++i) {

for (int j = 0; j < data.size(); ++j) {

double temp\_cutoff = data[j].attributes[i];

for (int k = 0; k < data.size(); ++k) {

if (data[k].attributes[i] <= temp\_cutoff) smaller.push\_back(data[k]);

else bigger.push\_back(data[k]);

}

double e1 = entropy(data);

double e2 = entropy(smaller);

double e3 = entropy(bigger);

double temp\_gain = e1 - (((double)smaller.size() / (double)data.size()) \* e2) - (((double)bigger.size() / (double)data.size()) \* e3);

if (temp\_gain > gain) {

cutoff = temp\_cutoff;

attribute\_cutoff = i;

gain = temp\_gain;

}

smaller.clear();

bigger.clear();

}

}

return vector<double>{cutoff, attribute\_cutoff};

}

void split\_pre() {

if (data.size() > 15) {

vector<int> type\_count = instances\_of\_each\_type(this->data);

int f = 0;

for (int i = 0; i < type\_count.size(); ++i) if (type\_count[i] != 0) f++;

if (f <= 1) {

attribute = -1;

label = data[0].type;

return;

}

vector <double> split\_point = find\_split\_point(this->data);

vector <line> smaller;

vector <line> bigger;

for (int i = 0; i < data.size(); ++i) {

double cutoff = split\_point[0];

if (data[i].attributes[split\_point[1]] <= split\_point[0]) smaller.push\_back(data[i]);

else bigger.push\_back(data[i]);

}

string s\_temp = ("Attribute-" + to\_string(int(split\_point[1] + 1)) + " <= " + to\_string(split\_point[0]));

this->decision = s\_temp;

attribute = int(split\_point[1]);

compare = split\_point[0];

left = new node(smaller, this->unique\_types);

left->level = this->level + 1;

left->parent = this;

right = new node(bigger, this->unique\_types);

right->level = this->level + 1;

right->parent = this;

left->split\_pre();

right->split\_pre();

}

else {

vector<int> type\_count = instances\_of\_each\_type(this->data);

if (type\_count[0] >= type\_count[1]) {

attribute = -1;

label = unique\_types[0];

for (int i = 0; i < data.size(); ++i) {

data[i].type = label;

}

}

else {

attribute = -1;

label = unique\_types[1];

for (int i = 0; i < data.size(); ++i) {

data[i].type = label;

}

}

unite\_leaves(this);

}

}

void split\_nopre() {

vector<int> type\_count = instances\_of\_each\_type(this->data);

int f = 0;

for (int i = 0; i < type\_count.size(); ++i) if (type\_count[i] != 0) f++;

if (f <= 1) {

attribute = -1;

label = data[0].type;

return;

}

vector <double> split\_point = find\_split\_point(this->data);

vector <line> smaller;

vector <line> bigger;

for (int i = 0; i < data.size(); ++i) {

double cutoff = split\_point[0];

if (data[i].attributes[split\_point[1]] <= split\_point[0]) smaller.push\_back(data[i]);

else bigger.push\_back(data[i]);

}

string s\_temp = ("Attribute-" + to\_string(int(split\_point[1] + 1)) + " <= " + to\_string(split\_point[0]));

this->decision = s\_temp;

attribute = int(split\_point[1]);

compare = split\_point[0];

left = new node(smaller, this->unique\_types);

left->level = this->level + 1;

right = new node(bigger, this->unique\_types);

right->level = this->level + 1;

left->parent = this;

right->parent = this;

left->split\_nopre();

right->split\_nopre();

}

void node\_print() {

string space = "";

for (int i = 0; i < this->level; ++i) {

space += " ";

}

if (left == nullptr && right == nullptr) {

vector <int> type\_count = instances\_of\_each\_type(data);

if (type\_count[0] != 0) {

cout << space << "Leaf: " << unique\_types[0] << "(" << type\_count[0] << ")" << endl;

}

if (type\_count[1] != 0) {

cout << space << "Leaf: " << unique\_types[1] << "(" << type\_count[1] << ")" << endl;

}

return;

}

cout << space << decision << endl;

left->node\_print();

right->node\_print();

}

};

void unite\_leaves(node\* leaf) {

if (leaf->parent->left->label == leaf->parent->right->label) {

leaf->parent->label = leaf->parent->left->label;

leaf->parent->attribute = -1;

for (int i = 0; i < leaf->parent->data.size(); ++i) {

leaf->parent->data[i].type = leaf->label;

}

leaf->parent->left = nullptr;

leaf->parent->right = nullptr;

unite\_leaves(leaf->parent);

}

}

int prune(node\* node\_p, node\* root, vector <line> test, double accuracy) {

int change = 0;

double new\_acc = 0;

vector<line>data;

int att;

string l, a\_l;

double compare, d;

int max = 0;

vector<node\*>c;

if (node\_p->decision == "Root" && node\_p->parent->right->decision == "Root") {

string leaf;

att = node\_p->parent->attribute;

l = node\_p->parent->label;

d = node\_p->parent->compare;

vector <int> size;

c.resize(2);

for (int i = 0; i < 2; ++i) {

if (i == 0) c[i] = node\_p->parent->left;

else c[i] = node\_p->parent->right;

if (c[i]->data.size() >= max) {

max = c[i]->data.size();

compare = c[i]->compare;

leaf = c[i]->label;

}

}

data = node\_p->data;

node\_p->parent->attribute = -1;

node\_p->parent->label = leaf;

node\_p->parent->compare = compare;

node\_p->parent->left = nullptr;

node\_p->parent->right = nullptr;

for (int i = 0; i < node\_p->parent->data.size(); ++i) node\_p->parent->data[i].type = leaf;

int correct = 0, incorrect = 0;

for (int i = 0; i < test.size(); ++i) {

if (classify(test[i], root) == true) correct++;

else incorrect++;

}

new\_acc = (double)correct / (double)test.size();

if (accuracy - new\_acc >= 0.1) {

node\_p->parent->left = c[0];

node\_p->parent->right = c[1];

node\_p->parent->attribute = att;

node\_p->parent->label = l;

node\_p->parent->compare = d;

node\_p->parent->data = data;

}

else {

accuracy = new\_acc;

change++;

return change;

}

}

else {

if (node\_p->left != nullptr) change += prune(node\_p->left, root, test, accuracy);

if (node\_p->right != nullptr) change += prune(node\_p->right, root, test, accuracy);

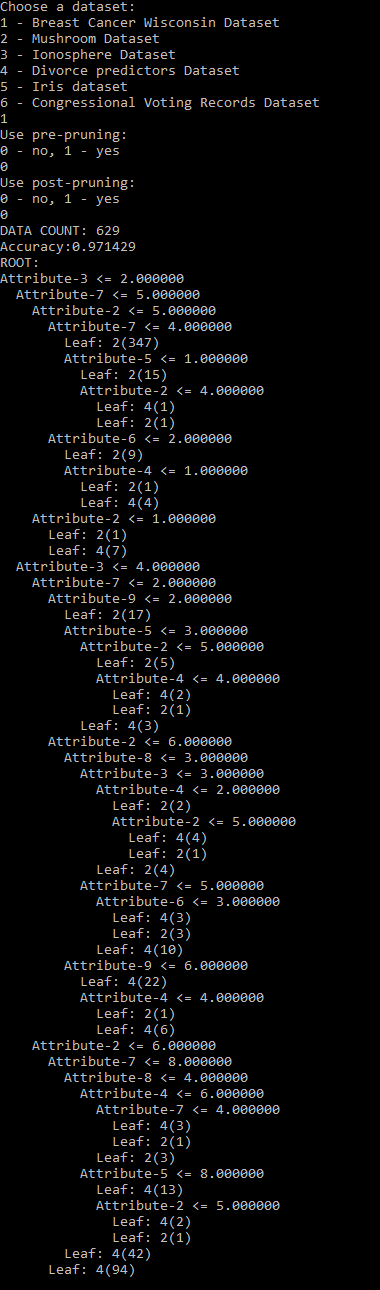
return change;

}

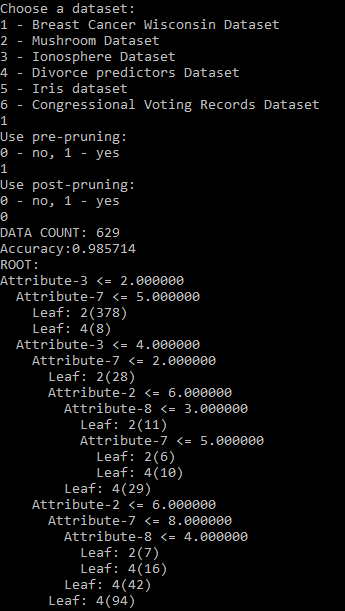
}

MainCW.cpp

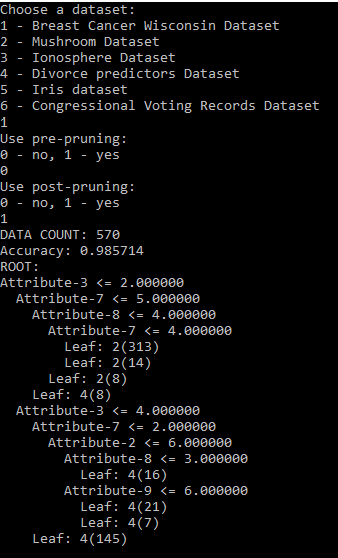
Код во многом аналогичен коду из Приложения А, Vote.cpp, отличаются лишь названия используемых файлов: cw.txt вместо vote.txt, cw\_prune.txt вместо vote\_prune.txt, cw\_test.txt вместо vote\_test.txt.



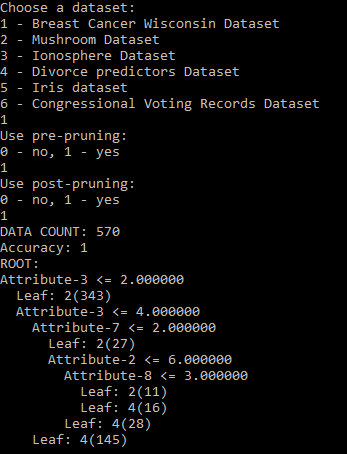
Результат работы без отсечения и ранней остановки



Результат работы с ранней остановкой



Результат работы с отсечением



Результат работы с отсечением и ранней остановкой

Приложение Г

Ionosphere.h

Код во многом аналогичен коду из Приложения Б, CW.h  
Ниже преставлены функции, в которых есть отличия.

void read\_data(vector<line>& data, string& str, int& count\_comma) {

string temp\_s;

vector<double> temp\_a;

int i = 0;

int j = 0;

while (j < count\_comma) {

string temp;

while (str[i] != ',') {

temp.push\_back(str[i]);

++i;

}

if (str[i] == ',') {

temp\_a.push\_back(stod(temp));

++i;

}

++j;

}

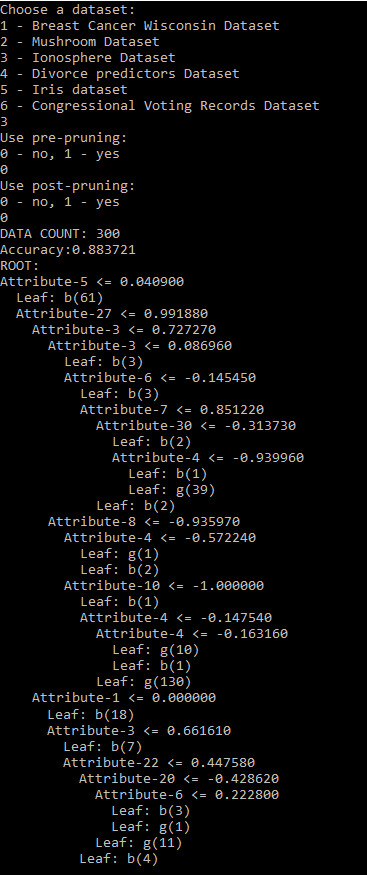
temp\_s = str.substr(i);

data.push\_back(line(temp\_a, temp\_s));

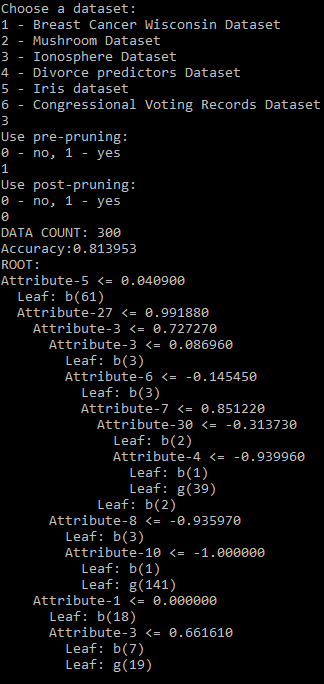
}

MainInosphere.cpp

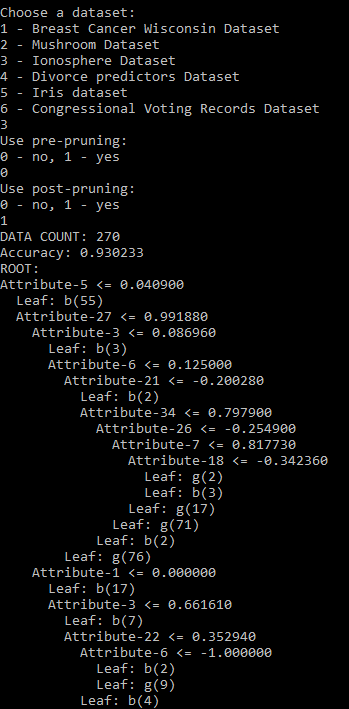
Код во многом аналогичен коду из Приложения А, Vote.cpp, отличаются лишь названия используемых файлов: ionosphere.txt вместо vote.txt, ionosphere\_prune.txt вместо vote\_prune.txt, ionosphere\_test.txt вместо vote\_test.txt.



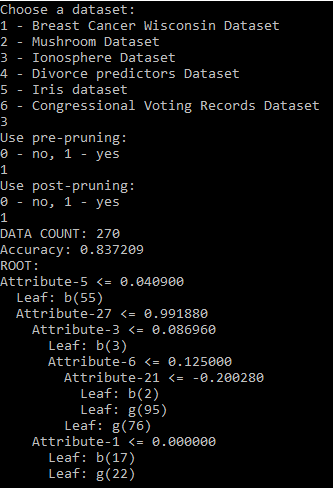
Результат работы без отсечения и ранней остановки



Результат работы с ранней остановкой



Результат работы с отсечением



Результат работы с отсечением и ранней остановкой

Приложение Д

Mushroom.h

Код во многом аналогичен коду из Приложения А, Vote.h  
Ниже преставлены функции, в которых есть отличия.

void read\_data(vector<line>& data, string& str, int& count\_comma) {

string temp\_s;

vector<string> temp\_a;

int i = 2;

int j = 0;

while (i < str.size()) {

string temp;

while (i != str.size() && str[i] != ',') {

temp.push\_back(str[i]);

++i;

}

if (i != str.size() && str[i] == ',') {

temp\_a.push\_back(temp);

++i;

}

++j;

}

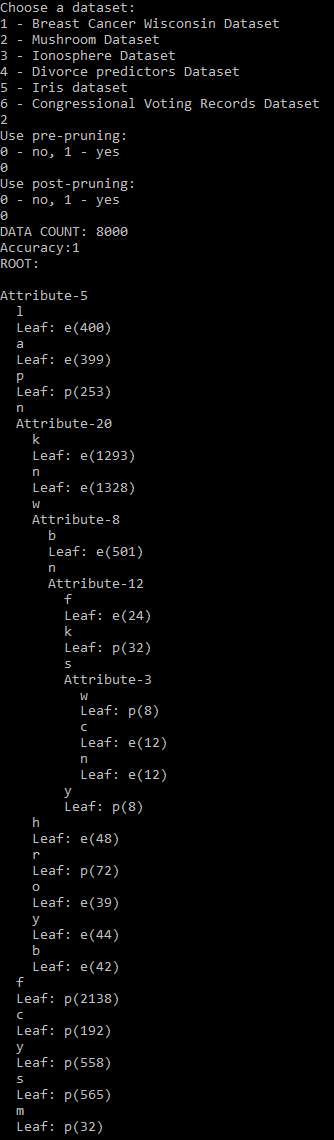
temp\_s = str.substr(0, 1);

data.push\_back(line(temp\_s, temp\_a));

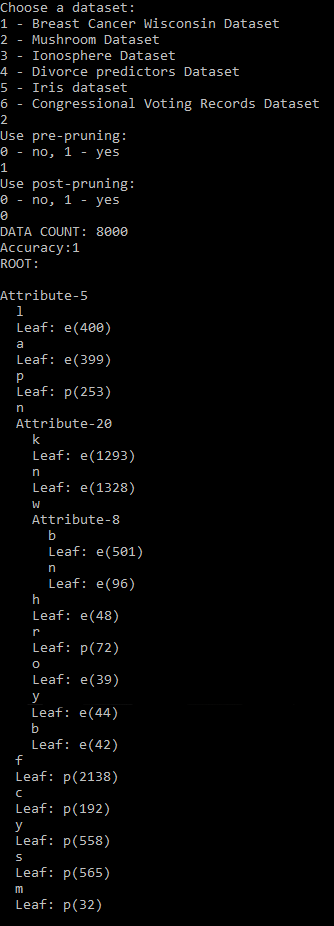
}

MainMushroom.cpp

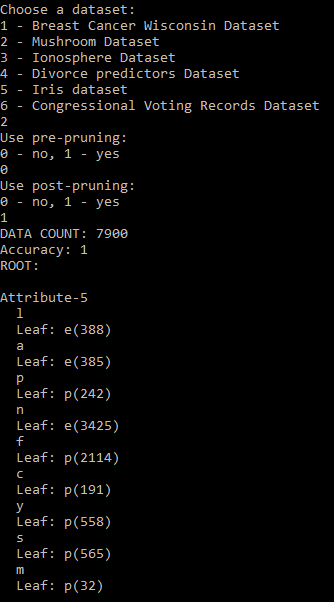
Код во многом аналогичен коду из Приложения А, Vote.cpp, отличаются лишь названия используемых файлов: mushroom.txt вместо vote.txt, mushroom\_prune.txt вместо vote\_prune.txt, mushroom\_test.txt вместо vote\_test.txt.



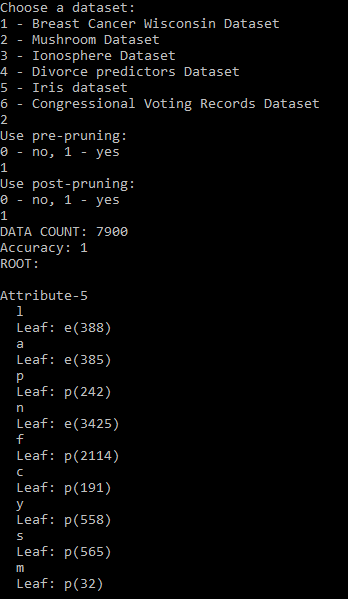
Результат работы без отсечения и ранней остановки



Результат работы с ранней остановкой



Результат работы с отсечением



Результат работы с отсечением и ранней остановкой

Приложение Е

Iris.h

Код во многом аналогичен коду из Приложения Б, CW.h  
Ниже преставлены функции, в которых есть отличия.

void read\_data(vector<line>& data, string& str, int& count\_comma) {

string temp\_s;

vector<double> temp\_a;

int i = 0;

int j = 0;

while (j < count\_comma) {

string temp;

while (str[i] != ',') {

temp.push\_back(str[i]);

++i;

}

if (str[i] == ',') {

temp\_a.push\_back(stod(temp));

++i;

}

++j;

}

temp\_s = str.substr(i);

data.push\_back(line(temp\_a, temp\_s));

}

vector<string> find\_types(vector<line> data) {

if (data.size() == 0) {

return vector<string>();

}

vector<string> unique;

unique.push\_back(data[0].type);

int i = 0;

bool type2 = false;

while (i < data.size()) {

if (data[0].type != data[i].type) {

if (!type2) { unique.push\_back(data[i].type); type2 = true; }

if (data[i].type != unique[1]) {

unique.push\_back(data[i].type);

break;

}

}

++i;

}

return unique;

}

vector<int> instances\_of\_each\_type(vector<line> d) {

int type1 = 0, type2 = 0, type3 = 0, i = 0;

while (i < d.size()) {

if (d[i].type == this->unique\_types[0]) type1++;

else if (d[i].type == this->unique\_types[1]) type2++;

else if (d[i].type == this->unique\_types[2])type3++;

++i;

}

return vector<int>{type1, type2, type3};

}

void node\_print() {

string space = "";

for (int i = 0; i < this->level; ++i) {

space += " ";

}

if (left == nullptr && right == nullptr) {

vector <int> type\_count = instances\_of\_each\_type(data);

if (type\_count[0] != 0) {

cout << space << "Leaf: " << unique\_types[0] << "(" << type\_count[0] << ")" << endl;

}

if (type\_count[1] != 0) {

cout << space << "Leaf: " << unique\_types[1] << "(" << type\_count[1] << ")" << endl;

}

if (type\_count[2] != 0) {

cout << space << "Leaf: " << unique\_types[2] << "(" << type\_count[2] << ")" << endl;

}

return;

}

cout << space << decision << endl;

left->node\_print();

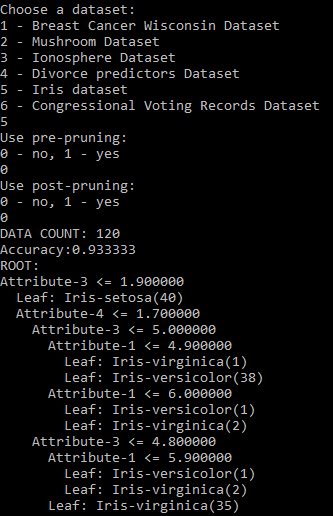
right->node\_print();

}

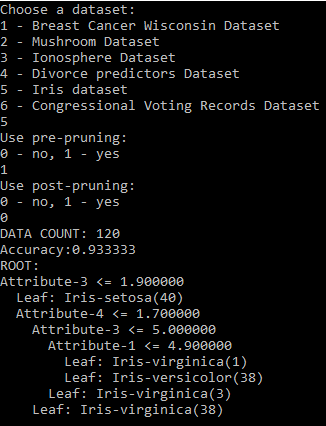
};

MainIris.cpp

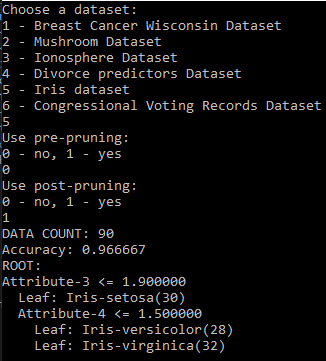
Код во многом аналогичен коду из Приложения А, Vote.cpp, отличаются лишь названия используемых файлов: iris.txt вместо vote.txt, iris\_prune.txt вместо vote\_prune.txt, iris\_test.txt вместо vote\_test.txt.



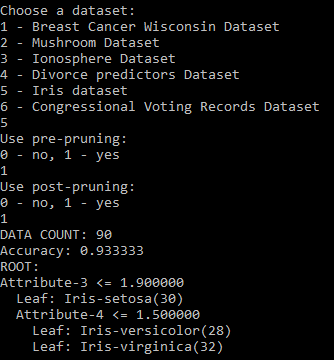
Результат работы без отсечения и ранней остановки



Результат работы с ранней остановкой



Результат работы с отсечением



Результат работы с отсечением и ранней остановкой

Приложение Ж

Divorce.h

Код во многом аналогичен коду из Приложения Б, CW.h  
Ниже преставлены функции, в которых есть отличия.

void read\_data(vector<line>& data, string& str, int& count\_comma) {

string temp\_s;

vector<double> temp\_a;

int i = 0;

int j = 0;

while (j < count\_comma) {

string temp;

while (str[i] != ' ') {

temp.push\_back(str[i]);

++i;

}

if (str[i] == ' ') {

temp\_a.push\_back(stod(temp));

++i;

}

++j;

}

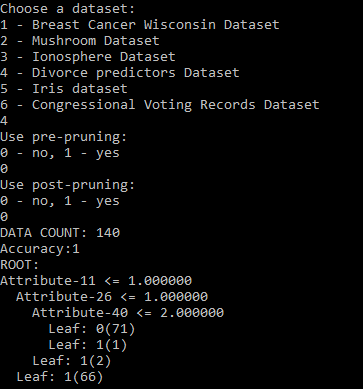
temp\_s = str.substr(i);

data.push\_back(line(temp\_a, temp\_s));

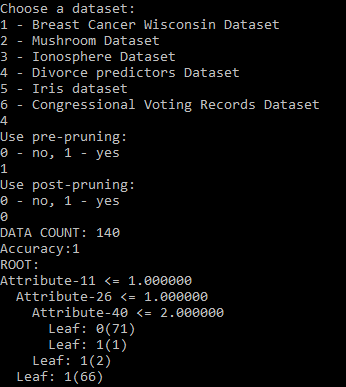
}

MainDivorce.cpp

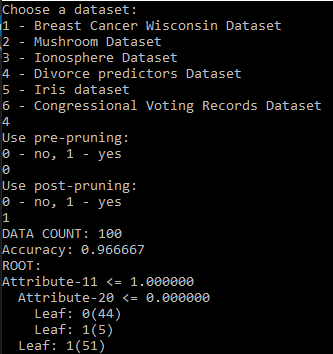
Код во многом аналогичен коду из Приложения А, Vote.cpp, отличаются лишь названия используемых файлов: divorce.txt вместо vote.txt, idivorce\_prune.txt вместо vote\_prune.txt, divorce\_test.txt вместо vote\_test.txt.



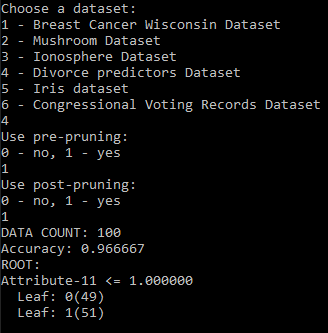
Результат работы без отсечения и ранней остановки



Результат работы с ранней остановкой



Результат работы с отсечением



Результат работы с отсечением и ранней остановкой