

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

ДЕРЕВЬЯ РЕШЕНИЙ. АЛГОРИТМ С4.5
КУРСОВАЯ РАБОТА

Студентки 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Ахмановой Элины Дамировны

Научный руководитель
к. ф.-м. н.

С. В. Миронов

Заведующий кафедрой
к. ф.-м. н.

С. В. Миронов

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Решающие деревья	4
1.1 Определение и построение решающих деревьев	4
1.2 Алгоритмы деревьев решений	10
1.3 Алгоритм C4.5	11
1.4 Оценка сложности алгоритма C4.5	12
2 Реализация алгоритма C4.5	14
2.1 Инструменты и технологии	14
2.2 Реализация	14
2.3 Задача определения победителя	20
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	24
Приложение А Листинг программы	26

ВВЕДЕНИЕ

Выполненная курсовая работа посвящена решающим деревьям и алгоритму C4.5. Решающие деревья - это семейство моделей, позволяющих восстанавливать нелинейные зависимости разной сложности [1]. Эти деревья относят к классу логических методов [2]. Основной идеей можно считать объединение определенного количества более простых решающих правил. Именно благодаря этому реализации можно считать интерпретируемым.

В одной из наиболее действенных реализаций, алгоритме C4.5, для достижения максимальной обучаемости используется способ объединения нескольких вершин в одну. При этом должен использоваться критерий целесообразности объединения двух вершин.

Целью данной работы является реализация алгоритма C4.5 и изучение решающих деревьев, аналогов алгоритма C4.5. Для реализации используется язык программирования Python, так как именно он является одним из наиболее оптимальных средств решения математических задач.

Первые работы по использованию решающих деревьев берут начало с 60-х годов прошлого века. Алгоритм C4.5 разработан Джоном Квинланом [3] как модификация другого алгоритма того же автора. Актуальность представленной работы заключается в большом количестве составных задач классификации или регрессии, для которых алгоритм может быть использован в составе композиций - решающих лесов.

1 Решающие деревья

1.1 Определение и построение решающих деревьев

Работа посвящена важному классу методов машинного обучения, решающим деревьям. Эти методы созданы для решения задач классификации (когда предсказываемый результат является классом), а в специальных реализациях - и для задач регрессии (когда предсказываемый результат можно рассматривать как вещественное число: цена на дом, продолжительность пребывания пациента в больнице).

Особенность решающих деревьев заключается в том, что они решают задачу получения коэффициента важности всех используемых признаков.

Область применения дерева решений в настоящее время широка, но все задачи, решаемые этим аппаратом могут быть объединены в следующие три класса [4]:

- Описание данных: Деревья решений позволяют хранить информацию о данных в компактной форме, вместо них мы можем хранить дерево решений, которое содержит точное описание объектов.
- Классификация: Деревья решений отлично справляются с задачами классификации, т.е. отнесения объектов к одному из заранее известных классов. Целевая переменная должна иметь дискретные значения.
- Регрессия: Если целевая переменная имеет непрерывные значения, деревья решений позволяют установить зависимость целевой переменной от независимых(входных) переменных. Например, к этому классу относятся задачи численного прогнозирования (предсказания значений целевой переменной).

Решающие деревья создавались с целью попытки формализовать способ мышления, который обычно люди используют для принятия решений. Это хорошо описывает логику принятия решений банком для выдачи кредита:

- возраст клиента;
- заработная плата;
- рабочий стаж;
- наличие других кредитов и кредитная история.

Также это отлично иллюстрирует процесс работы врача, когда тот говорит с пациентом, задает ему один за другим уточняющие вопросы, может

дать необходимые советы.

Дерево решений [5] – это способ представления правил в иерархической, последовательной структуре [6]. Основа такой структуры – ответы “Да” или “Нет” в случае бинарных деревьев (алгоритм CART [7]), или на ряд вопросов, определяемых динамически в случае деревьев с произвольным числом потомков (алгоритм C4-5 [2]).

Каждой вершине дерева за исключением конечной листовой соответствует вопрос с несколькими вариантами ответа. Ответы будут являться выходящими ребрами. В зависимости от ответа осуществляется переход к одной из нижних вершин. Концевая вершина, лист, соответствует одному из классов. При положительном ответе на вопрос осуществляется переход к левой части дерева, называемой левой ветвью, при отрицательном – к правой части дерева. Таким образом, внутренний узел дерева является узлом проверки определенного условия. Для каждой конкретной задачи существуют свои типы конечного узла в количестве $n + 1$.

Решающие деревья являются ациклическими графами и воспроизводят логические схемы, позволяющие получить окончательное решение о классификации объекта с помощью ответов на иерархически организованную систему вопросов [8].

Обратим внимание на то, что заданный на последнем иерархическом уровне вопрос зависит от полученного на предыдущем уровне ответа. Подобные модели свойственны для использования в:

- медицине;
- ботанике;
- зоологии;
- социологии;
- экономике.

Решающее дерево называется бинарным, если каждая внутренняя или корневая вершина поставлена в соответствие только двум выходящим рёбрам. Бинарные деревья удобно использовать в моделях машинного обучения [8].

Модель, представленная в виде дерева решений, является интуитивной и упрощает понимание решаемой задачи. Результат работы алгоритмов конструирования деревьев решений, в отличие, например, от нейронных сетей,

представляющих собой “черные ящики”, легко интерпретируется пользователем. Это свойство деревьев решений не только важно при отнесении к определенному классу нового объекта, но и полезно при интерпретации модели классификации в целом [6].

Дерево решений позволяет понять и объяснить, почему конкретный объект относится к тому или иному классу. Деревья решений дают возможность извлекать правила из базы данных на естественном языке [6].

Распознавание с помощью решающих деревьев Предположим, что бинарное дерево T используется для распознавания объектов, описываемых набором признаков X_1, X_2, \dots, X_n [8].

Каждой вершине ν дерева T ставится в соответствие предикат, касающийся значения одного из признаков. Непрерывному признаку X_j соответствует предикат вида " $X_j \geq \delta_j^x$ где j - некоторый пороговый параметр ??..

Категориальному признаку X_{j^1} принимающему значения из множества $M_{j^1} = a_1^{j^1}, \dots, a_{r(j^1)}^{j^1}$ ставится в соответствие предикат вида " $X_{j^1} \in M_{j^1}^{v1}$ где $M_{j^1}^{v1}$ является элементом дихотомического разбиения $M_{j^1}^{v1}, M_{j^1}^{v2}$ множества M_{j^1} . Выбор одного из двух, выходящих из вершины ν ребер производится в зависимости от значения предиката.

Процесс распознавания заканчивается при достижении концевой вершины (листа). Объект относится классу согласно метке, поставленной в соответствие данному листу.

Процесс распознавания заканчивается при достижении концевой вершины (листа). Объект относится классу согласно метке, поставленной в соответствие данному листу.

Обучение решающих деревьев Рассмотрим задачу распознавания с классами K_1, \dots, K_L . Обучение производится по обучающей выборке \tilde{S}_t и включает в себя поиск оптимальных пороговых параметров или оптимальных дихотомических разбиений для признаков X_1, \dots, X_n [9].

При этом поиск производится исходя из требования снижения среднего индекса неоднородности в выборках, порождаемых искомым дихотомическим разбиением обучающей выборки \tilde{S}_t .

Индекс неоднородности вычисляется для произвольной выборки \tilde{S} , содержащей объекты из классов K_1, \dots, K_L . При этом используется несколько видов индексов, включая:

- энтропийный индекс неоднородности;
- индекс Джини [10];
- индекс ошибочной классификации.

Энтропийный индекс неоднородности вычисляется по формуле [11]:

$$\gamma_e(\tilde{S}) = - \sum P_i \ln P_i, \quad (1)$$

где P_i - доля объектов класса K_i в выборке \tilde{S} . При этом принимается, что $0 \ln(0) = 0$. Наибольшее значение $\gamma_e(\tilde{S})$ принимает при равенстве долей классов. Наименьшее значение $\gamma_e(\tilde{S})$ достигается при принадлежности всех объектов одному классу.

Индекс Джини вычисляется по формуле

$$\gamma_e(\tilde{S}) = - \sum P_i^2. \quad (2)$$

Индекс ошибочной классификации вычисляется по формуле

$$\gamma_e(\tilde{S}) = 1 - \max(P_i). \quad (3)$$

Нетрудно понять, что индексы (2) и (3) также достигают минимального значения при принадлежности всех объектов обучающей выборке одному классу.

Предположим, что в методе обучения используется индекс неоднородности γ_* . Для оценки эффективности разбиения обучающей выборки \tilde{S}_t на непересекающиеся подвыборки \tilde{S}_t^l и \tilde{S}_t^r используется уменьшение среднего индекса неоднородности в \tilde{S}_t^l и \tilde{S}_t^r по отношению к \tilde{S}_t .

Данное уменьшение вычисляется по формуле:

$$\Delta(\gamma_* \tilde{S}_t) = \gamma_*(\tilde{S}_t) - P_l \gamma_*(\tilde{S}_t^l) - P_r \gamma_*(\tilde{S}_t^r). \quad (4)$$

где P_l , P_r являются долями \tilde{S}_t^l и \tilde{S}_t^r в выборке \tilde{S}_t . На первом этапе обучения бинарного решающего дерева ищется оптимальный предикат соответствующий корневой вершине. С этой целью оптимальные разбиения строятся для каждого из признаков из набора X_1, \dots, X_n . Выбирается признак $X_{i_{\max}}$ с максимальным значением индекса $\Delta(\gamma_* \tilde{S}_t)$. Подвыборки \tilde{S}_t^l и \tilde{S}_t^r , задаваемые оптимальным предикатом для $X_{i_{\max}}$ оцениваются с помощью критерия остановки.

В качестве критерия останковки может быть использован простейший критерий достижения полной однородности по одному из классов. В случае, если какая-нибудь из выборок \widetilde{S}_t^* удовлетворяет критерию останковки, то соответствующая вершина дерева объявляется концевой и для неё вычисляется метка класса. В случае, если выборка \widetilde{S}_t^* удовлетворяет критерию останковки, то формируется новая внутренняя вершина, для которой процесс построения дерева продолжается [8].

Однако вместо обучающей выборки \widetilde{S}_t используется соответствующая вновь образованной внутренней вершине ν выборка \widetilde{S}_t , которая равна \widetilde{S}_t^* .

Для данной выборки производятся те же самые построения, которые на начальном этапе проводились для обучающей выборки \widetilde{S}_t .

Обучение может проводиться до тех пор, пока все вновь построенные вершины не окажутся однородными по классам. Такое дерево может быть построено всегда, когда обучающая выборка не содержит объектов с одним и тем же значениям каждого из признаков, принадлежащих разным классам. Однако абсолютная точность на обучающей выборке не всегда приводит к высокой обобщающей способности в результате эффекта переобучения.

Одним из способов достижения более высокой обобщающей способности является использования критериев останковки, позволяющих остановить процесс построения дерева до того, как будет достигнута полная однородность концевых вершин.

Рассмотрим несколько таких критериев [12].

1. Критерий останковки по минимальному допустимому числу объектов в выборках, соответствующих концевым вершинам.
2. Критерий останковки по минимально допустимой величине индекса $\Delta(\gamma_* \widetilde{S}_t)$. Предположим, что некоторой вершине ν соответствует выборка \widetilde{S}_ν , для которой найдены оптимальный признак вместе с оптимальным предикатом, задающим разбиение $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$. Вершина ν считается внутренней, если индекс $\Delta(\gamma_* \widetilde{S}_t)$ превысил пороговое значение τ и считается концевой в противном случае.
3. Критерий останковки по точности на контрольной выборке. Исходная выборка данных случайным образом разбивается на обучающую выборку \widetilde{S}_t и контрольную выборку \widetilde{S}_c . Выборка \widetilde{S}_t используется для построения бинарного решающего дерева. Предположим, что некоторой

вершине ν соответствует выборка \widetilde{S}_ν , для которой найдены оптимальный признак вместе с оптимальным предикатом, задающим разбиение $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$.

На контрольной выборке \widetilde{S}_c производится сравнение эффективности распознающей способности деревьев T_ν и $T_n u^{++}$.

Деревья T_ν и $T_n u^{++}$ включают все вершины и рёбра, построенные до построения вершины ν . В дереве T_ν вершина ν считается концевой. В дереве $T_n u^{++}$ вершина ν считается внутренней, а концевыми считаются вершины, соответствующие подвыборкам $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$. Распознающая способность деревьев T_ν и $T_n u^{++}$ сравнивается на контрольной выборке \widetilde{S}_c . В том, случае если распознающая способность $T_n u^{++}$ превосходит распознающую способность T_ν все дальнейшие построения исходят из того, что вершина ν является концевой. В противном случае производится исследование $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$.

4. Статистический критерий [13]. Заранее фиксируется пороговый уровень значимости ($P < 0.05, p < 0.001$ или $p < 0.001$). Предположим, что нам требуется оценить, является ли концевой вершина, для которой найдены оптимальный признак вместе с оптимальным предикатом, задающим разбиение $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$.

Исследуется статистическая достоверность различий между содержанием объектов распознаваемых классов в подвыборках $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$. Для этих целей может быть использованы известные статистический критерий: Хи-квадрат и другие критерии.

По выборкам $\{\widetilde{S}_\nu^l, \widetilde{S}_\nu^r\}$ рассчитывается статистика критерия и устанавливается соответствующее р-значение. В том случае, если полученное р-значение оказывается меньше заранее фиксированного уровня значимости вершина ν считается внутренней. В противном случае вершина ν считается концевой.

Использование критериев ранней остановки не всегда позволяет адекватно оценить необходимую глубину дерева. Слишком ранняя остановка ветвления может привести к потере информативных предикатов, которые могут быть на самом деле найдены только при достаточно большой глубине ветвления.

В связи с этим нередко целесообразным оказывается построение сначала

ла полного дерева, которое затем уменьшается до оптимального с точки зрения достижения максимальной обучающей способности размера путём объединения некоторых концевых вершин. Такой процесс в литературе принято называть "pruning" ("подрезка") [14].

При подрезке дерева может быть использован критерий целесообразности объединения двух вершин, основанный на сравнении на контрольной выборке точности распознавания до и после проведения "подрезки".

Ещё один способ оптимизации обобщающей способности деревьев основан на учёте при "подрезке" дерева до некоторой внутренней вершины ν одновременно увеличения точности разделения классов на обучающей выборке и увеличения сложности, которые возникают благодаря ветвлению из ν .

1.2 Алгоритмы деревьев решений

- ID3 (Induction of Decision Tree [15]) - рекурсивная процедура, которая принимает на вход некоторую подвыборку обучающей выборки. Когда она вызывается впервые, на вход подается вся выборка. Если все объекты выборки лежат в одном классе, образуется и возвращается новая листовая вершина, которая содержит этот класс. Иначе ищется предикат, который максимально хорошо выделяет часть классов. По этому предикату строятся две новые вершины.

Этот алгоритм является наиболее старым и лаконичным способом использования решающих деревьев. К плюсам данного алгоритма можно отнести: простоту, интерпретируемость, гибкость задания множества предикатов, возможность использования данных с пропусками, линейная трудоемкость.

Однако недостатки вытекают из того, что это жадная стратегия, и решения о ветвлении оптимальны локально. Кроме того, данная процедура очень чувствительна к составу выборки и шумовым признакам.

- C4.5 [2] - улучшенная версия алгоритма ID3, где выбор атрибута происходит на основании нормализованного прироста информации, использует алгоритм усечения решающих деревьев и избавляет от проблемы переобучения, переусложнения структуры дерева, создавая контрольную выборку данных, которая обычно вдвое короче обучающей. Эта выборка используется для сокращения количества вершин в итоговом

вом дереве.

- CART (Classification and Regression Tree) - жадный рекурсивный алгоритм построения дерева решений, использующий стратегии усечения, обобщенный для решения задач регрессии.
- MARS - алгоритм, который использует расширение деревьев решений для улучшения обработки цифровых данных.

1.3 Алгоритм C4.5

C4.5 - алгоритм построения дерева решений, созданный Россом Куинланом [16], как модификация более простого алгоритма ID3. Этот алгоритм используется для задач классификации, поэтому его часто упоминают как статический классификатор.

C4.5 строит деревья решений таким же образом, как это делает ID3, используя тестовую выборку и понятие информационной энтропии [17].

Информационная энтропия - это мера неопределенности некоторой ситуации [18]. Можно также назвать ее мерой рассеяния и в этом смысле она подобна дисперсии. Но если дисперсия является адекватной мерой рассеяния лишь для специальных распределений вероятностей случайных величин (а именно – для двухмоментных распределений, в частности, для гауссова распределения), то энтропия не зависит от типа распределения.

C4.5 моделирует дерево решений с неограниченным количеством ветвей у узла. Данный алгоритм может работать только с дискретным зависимым атрибутом и поэтому может решать только задачи классификации.

Для работы алгоритма C4.5 необходимо соблюдение следующих требований:

- каждая запись набора данных должна быть ассоциирована с одним из предопределенных классов, т.е. один из атрибутов набора данных должен являться меткой класса;
- классы должны быть дискретными;
- каждый пример должен однозначно относиться к одному из классов;
- количество классов должно быть значительно меньше количества записей в исследуемом наборе данных.

Простейшая реализация алгоритма аналогична ID3:

```
1 Input: database D
2 Tree = {}
```

```

3  If D is pure OR other stopping criteria Then
4      Terminate
5  End If
6  For all attribute a ? D do
7      Entropy for root
8  End For
9
10 abest = Best attribute entropy
11 Tree = Create a node then test a best in the root
12 Dv = sud_datasets from D
13 For all Dv do
14     Tree v= C4.5(Dv)
15 End for
16 Return Tree

```

1.4 Оценка сложности алгоритма C4.5

Алгоритм решающего дерева имеет сложность, линейную по длине выборки. Если множество предикатов настолько богато, что на шаге всегда находится предикат, разбивающий выборку \tilde{S}_t на непустые подмножества \tilde{S}_t^l и \tilde{S}_t^r , то алгоритм строит бинарное решающее дерево, безошибочно классифицирующее выборку \tilde{S}_t .

Общая схема построения дерева принятия решений выглядит следующим образом:

1. Выбирается очередной атрибут, помещается в корень.
2. Для всех его значений рекурсивно строится дерево в этом потомке.

Начиная со второго шага, алгоритм может быть распараллелен.

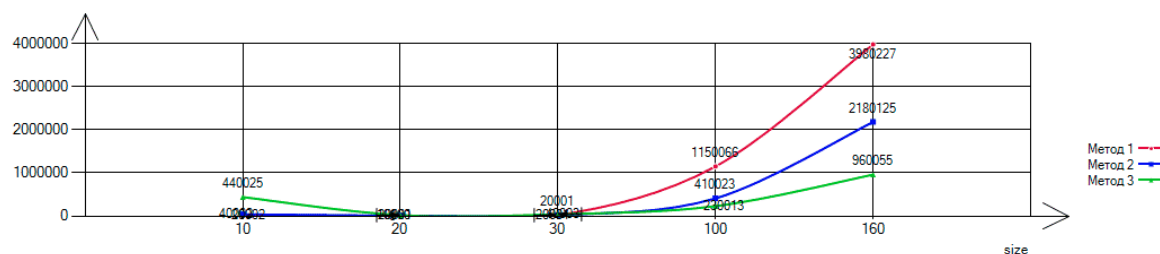


Рисунок 1 – Зависимость времени работы алгоритма от объема обучающей выборки

На рисунке 1 изображена скорость работы трех методов:

1. Линейные вычисления.
2. Распараллеливание в один поток (блокирование внешних потоков).
3. Полностью распараллеленный алгоритм на 2 потока .

Из графика 1 видно, что распараллеливание алгоритма деревьев решений имеет смысл проводить только на больших объемах данных. Для случаев, где выборка данных меньше 50 элементов распараллеливание избыточно, линейный вариант алгоритма работает быстрее.

2 Реализация алгоритма C4.5

2.1 Инструменты и технологии

Наиболее распространенными инструментами для работы с алгоритмами машинного обучения на сегодняшний день являются языки R [19] и Python [20]. В работе используется версия Python 2.7.6. Обращаем внимание на то, что мажорные версии Python 3 и 2 не являются обратно-совместимыми.

Python - мультипарадигмальный (объектно-ориентированный, рефлексивный, императивный, функциональный и др.) высокоуровневый язык программирования, предназначенный для решения разнообразных задач, в том числе - задач машинного обучения [14].

В языке уже присутствует документированная библиотека Scikit-Learn [21], в которой реализовано большое количество алгоритмов машинного обучения, в том числе и для деревьев решений. Наша реализация алгоритма C4.5 претендует на гибкое использование проверки, усечения вершин и атрибута мульти-расщепления.

2.2 Реализация

На первом этапе создадим класс `data`, который будет использоваться при загрузке данных из файлов с расширением `.csv`.

```
class data:
    def __init__(self, datafile, test=False):
        self.datafile = datafile
        self.read_data()

    def read_data(dataset, datafile, datatypes):
        print "Reading data..."
        f = open(datafile)
        original_file = f.read()
        rowsplit_data = original_file.splitlines()
        dataset.examples = [rows.split(',') for rows in rowsplit_data]

        #list attributes
        dataset.attributes = dataset.examples.pop(0)
```

```

        #create array that indicates whether each attribute is a num
        attr_type = open(datatypes)
        orig_file = attr_type.read()
        dataset.attr_types = orig_file.split(',')

```

Файлы состоят из данных и их атрибутов. Поэтому в функции read_data() мы будем считывать из разных файлов примеры, атрибуты и типы атрибутов.

```

f = open('PATH_CSV')
    original_file = f.read()
    rowsplit_data = original_file.split("\r")
    self.examples = [rows.split(',') for rows in rowsplit_data]

    self.attributes = self.examples.pop(0)

    attr_type = open('PATH_ATTR_CSV')
    orig_file = attr_type.read()
    self.attr_types = orig_file.split(',')

    for example in self.examples:
        for x in range(len(self.examples[0])):
            if self.attributes[x] == 'True':
                example[x] = float(example[x])

```

Для препроцессинга данных используется функция preprocess2() фильтрации по полю attr:

```

def preprocess2(dataset)

```

Добавим функцию entropy() для подсчета энтропии, описание которой было дано выше.

```

def entropy(instances, attributes, classifier):
    count = one_count(instances, attributes, classifier)
    total = len(instances)
    count0 = total - count
    s = (-(count/total) * math.log2(count/total)) +

```

```
(-(count0/total) * math.log2(count0/total))  
return s
```

Основным классом является класс `treeNode()`, который будет представлять дерево решений.

```
class treeNode():  
    def __init__(self, is_leaf, classification, attr_split_index,  
                attr_split_value, parent, upper_child, lower_child, height):  
        self.is_leaf = True  
        self.classification = None  
        self.attr_split = None  
        self.attr_split_index = None  
        self.attr_split_value = None  
        self.parent = parent  
        self.upper_child = None  
        self.lower_child = None  
        self.height = None
```

Для его построения используется рекурсивная функция `compute_tree()`:

```
def compute_tree(dataset, parent_node, classifier):  
    node = treeNode(True, None, None, None,  
                    parent_node, None, None, 0)  
    if (parent_node == None):  
        node.height = 0  
    else:  
        node.height = node.parent.height + 1  
  
    ones = one_count(dataset.examples, dataset.attributes,  
                    classifier)  
    if (len(dataset.examples) == ones):  
        node.classification = 1  
        node.is_leaf = True  
        return node  
    elif (ones == 0):
```



```

        node.classification = 0
        node.is_leaf = True
        return node
    else:
        node.is_leaf = False

    attr_to_split = None
    max_gain = 0
    split_val = None
    min_gain = 0.01
    dataset_entropy = calc_dataset_entropy(dataset, classifier)

    for attr_index in range(len(dataset.examples[0])):

        if (dataset.attributes[attr_index] != classifier):
            local_max_gain = 0
            local_split_val = None
            attr_value_list = [example[attr_index]
                               for example in dataset.examples]
            attr_value_list = list(set(attr_value_list))
            if(len(attr_value_list) > 100):
                attr_value_list = sorted(attr_value_list)
                total = len(attr_value_list)
                ten_percentile = int(total/10)
                new_list = []
                for x in range(1, 10):
                    new_list.append(attr_value_list[x*ten_percentile])
                attr_value_list = new_list

            for val in attr_value_list:
                local_gain = calc_gain(dataset, dataset_entropy, val,
                                       attr_index)

                if (local_gain > local_max_gain):

```

```

        local_max_gain = local_gain
        local_split_val = val

    if (local_max_gain > max_gain):
        max_gain = local_max_gain
        split_val = local_split_val
        attr_to_split = attr_index

if (split_val is None or attr_to_split is None):
    print "Something went wrong.
        Couldn't find an attribute to split on (or a split value)."
elif (max_gain <= min_gain or node.height > 20):

    node.is_leaf = True
    node.classification = classify_leaf(dataset, classifier)

    return node

node.attr_split_index = attr_to_split
node.attr_split = dataset.attributes[attr_to_split]
node.attr_split_value = split_val
#SPLIT
upper_dataset = data(classifier)
lower_dataset = data(classifier)
upper_dataset.attributes = dataset.attributes
lower_dataset.attributes = dataset.attributes
upper_dataset.attr_types = dataset.attr_types
lower_dataset.attr_types = dataset.attr_types
for example in dataset.examples:
    if (attr_to_split is not None and
        example[attr_to_split] >= split_val):
        upper_dataset.examples.append(example)
    elif (attr_to_split is not None):
        lower_dataset.examples.append(example)

```

```
node.upper_child = compute_tree(upper_dataset, node, classifier)
node.lower_child = compute_tree(lower_dataset, node, classifier)

return node
```

Для классификации сета данных используется функция:

```
def classify_leaf(dataset, classifier):
    ones = one_count(dataset.examples, dataset.attributes,
                      classifier)
    total = len(dataset.examples)
    zeroes = total - ones
    if (ones >= zeroes):
        return 1
    else:
        return 0
```

Для того, чтобы исключить влияние переобучения и утяжеления дерева, была создана функция подрезки:

```
def prune_tree(root, node, dataset, best_score):
    if (node.is_leaf == True):
        classification = node.classification
        node.parent.is_leaf = True
        node.parent.classification = node.classification
        if (node.height < 20):
            new_score = validate_tree(root, dataset)
        else:
            new_score = 0
        if (new_score >= best_score):
            return new_score
        else:
            node.parent.is_leaf = False
            node.parent.classification = None
            return best_score
```

```

else:
    new_score = prune_tree(root, node.upper_child, dataset, best_score)
    if (node.is_leaf == True):
        return new_score
    new_score = prune_tree(root, node.lower_child, dataset, new_score)
    if (node.is_leaf == True):
        return new_score
return new_score

```

Полный код приложения представлен на открытом репозитории <https://github.com> и в Приложении [А](#)

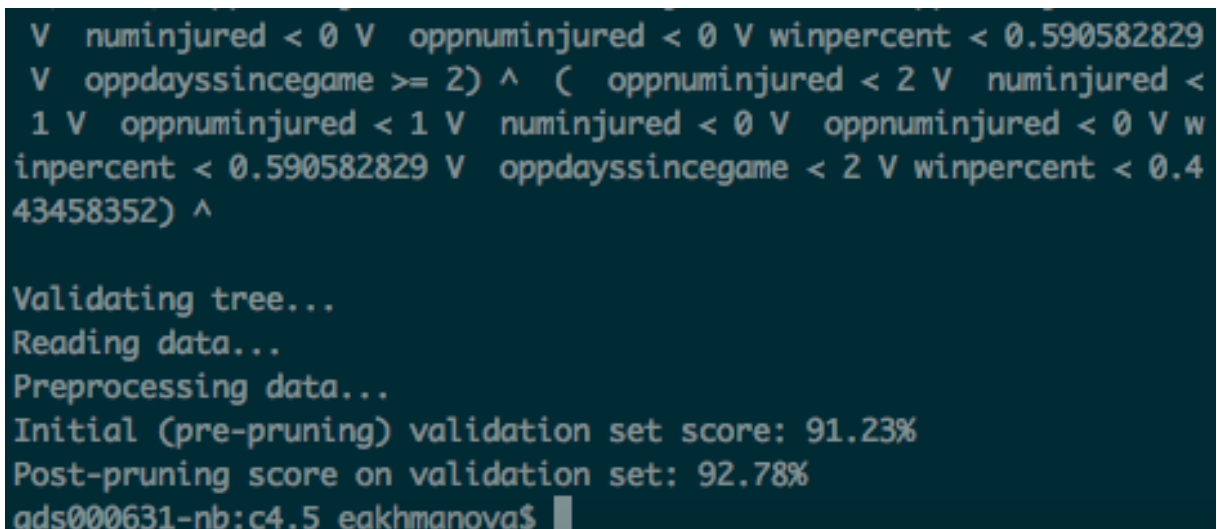
2.3 Задача определения победителя

Для того, чтобы протестировать работу алгоритма были использованы данные спортивных соревнований. По 12 дискретным признакам необходимо определить вероятность выигрыша.

Листинг 1: bash version

```
python decision-tree.py btrain.csv winner \
    -d datatypes.csv -v bvalidate.csv -p -s
```

Эта команда запускает алгоритм C4.5. Данные для обучения взяты из btrain.csv, данные для проверки – bvalidate.csv. Для классификации используется параметр "winner". Флаг -p используется для подрезки, флаг -s для печати.



```

V numinjured < 0 V oppnuminjured < 0 V winpercent < 0.590582829
V oppdayssincegame >= 2) ^ ( oppnuminjured < 2 V numinjured <
1 V oppnuminjured < 1 V numinjured < 0 V oppnuminjured < 0 V w
inpercent < 0.590582829 V oppdayssincegame < 2 V winpercent < 0.4
43458352) ^

Validating tree...
Reading data...
Preprocessing data...
Initial (pre-pruning) validation set score: 91.23%
Post-pruning score on validation set: 92.78%
gds000631-nb:c4.5 eakhmanova$

```

Рисунок 2 – Результаты выполнения программы

Winner in pre-pruning

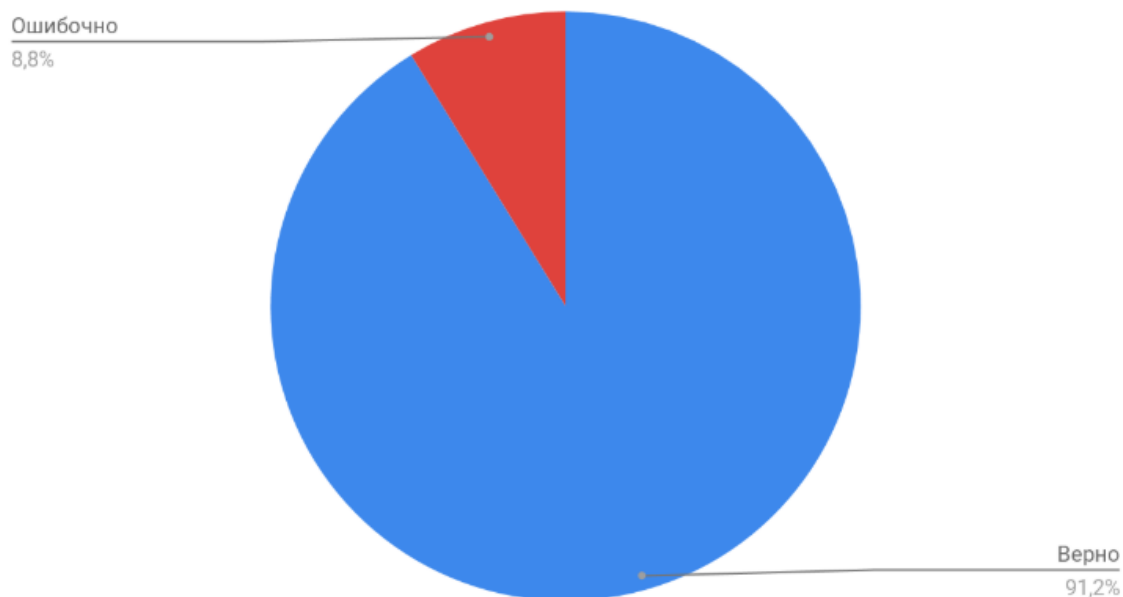


Рисунок 3 – Результаты выполнения до усечения ветвей

Winner in pruning

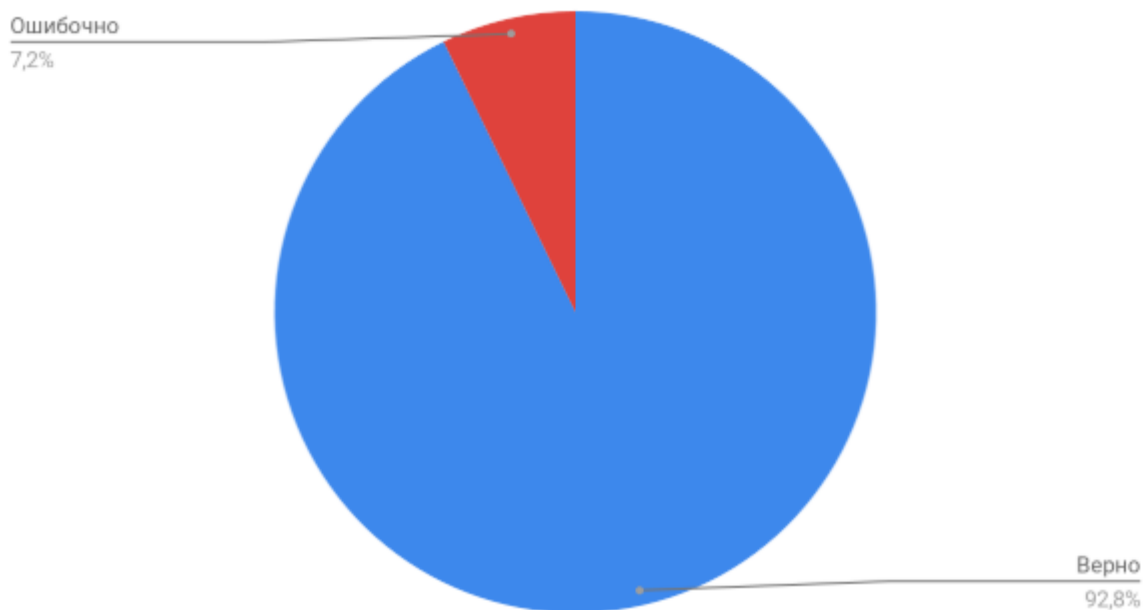


Рисунок 4 – Результаты выполнения после усечения ветвей

На рисунке 3 видно процентное соотношение верных результатов до подрезки. Точность определения победителя колеблется около 91,2%.

Модификация алгоритма позволила улучшить результаты определения победителя в среднем на 1,5%. Можно заметить на рисунке 4, что вероят-

ность определения победителя выросла до 92,7%.

Таким образом при 12 признаках благодаря усечению ветвей удалось удалить минимизировать эффект переобучения, упрощая слишком детализированные деревья, которые приносили 1,5% ошибок.

ЗАКЛЮЧЕНИЕ

В рамках выполнения данной курсовой работы нами были изучены деревья принятия решений, а также алгоритмы, связанные с ними (ID3, C4.5, CART, MARS) и задачи классификации и регрессии, для которых они используются.

Особое внимание было уделено алгоритму C4.5, его математическому аппарату и реализации на языке программирования Python. Во время реализации был создан репозиторий для хранения кода в открытом доступе. Таким образом, любой пользователь сети Интернет может воспользоваться кодом для решения задач классификации. Эти задачи могут принадлежать различным предметным областям (медицина, статистика, экономика и др.) и должны быть размечены на признаки, каждый из которых должен иметь дискретное выражение. Особенностью выбранного алгоритма служит возможность содержания в данных пропущенных признаков.

По итогам тестирования алгоритма на размеченных данных спортивных соревнований было получено дерево решений. Точность решения задачи (определение победителя в соревновании) составила 91,2% процента до усе-чения ветвей и 92,7% после. Этот результат можно считать достаточным для решения задачи классификации с помощью деревьев принятия решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Moscow Institute of Physics and Technology Специализация: Машинное обучение и анализ данных. Курс: Обучение на размеченных данных [Электронный ресурс]. — URL: <https://ru.coursera.org/lecture/supervised-learning/rieshaiushchiie-dieriev-ia-HZxD1> (Дата обращения 20.05.2018). Загл. с экр. Яз. рус.
- 2 *Quinlan, J. R.* C4.5: Programs for Machine learning / J. R. Quinlan. — San Mateo: Morgan Kaufmann Publishers, 1993. — 302 pp.
- 3 *Quinlan, J. R.* Learning logical definitions from relations. Machine Learning / J. R. Quinlan. — 1990. — 239-266 pp.
- 4 *Н. Соловьев А. Семенов, А. Ц. Е. Ч.* Интеллектуальные системы / А. Ц. Е. Ч. Н. Соловьев, А. Семенов. — Оренбург, 2013. — 236 с.
- 5 Лекции Ульяновского государственного университета [Электронный ресурс]. — URL: <https://studfiles.net/preview/6172591/page:10> (Дата обращения 20.05.2018). Загл. с экр. Яз. рус.
- 6 *Чубуков, И. Ф.* Data Mining. Учебный курс [Электронный ресурс] / И. Ф. Чубуков. — URL: <http://www.intuit.ru/department/database/datamining/> (Дата обращения 20.05.2018). Загл. с экр. Яз. рус.
- 7 *Breiman, L.* Classification and regression trees / L. Breiman, J. H. Friedman, C. J. Olshen, R. A. ands Stone. — Belmont, CA: Wadsworth International Group, 1984. — 354 pp.
- 8 *Сенько, О. В.* Курс «Математические основы теории прогнозирования» Лекция 8 Решающие деревья / О. В. Сенько. — МОТП, 2017. — 15 с.
- 9 *Сирота, А. А.* Методы и алгоритмы анализа данных и их моделирование в MATLAB / А. А. Сирота. — Петербург: БХВ-Петербург, 2017. — 384 с.
- 10 *Sen, A.* 16 diagrams On Economic Inequality / A. Sen. — Oxford: Oxford University Press, 1997. — 280 pp.
- 11 *Shannon, C. E.* A mathematical theory of communication / C. E. Shannon // *Bell System Technical Journal*. — 1948. — Pp. 379–423.
- 12 *Bies, R. R.* A genetic algorithm-based, hybrid machine learning approach to model selection / R. R. Bies, M. F. Muldoon, B. G. Pollock, S. Manuck,

- G. Smith, M. E. Sale // *Journal of Pharmacokinetics and Pharmacodynamics*. — 2006. — Pp. 196–221.
- 13 *Berger, R. L.* Statistical Inference / R. L. Berger, G. Casella. — Duxbury Press, 2001. — 374 pp.
 - 14 *Hastie, T.* The elements of statistical learning / T. Hastie, R. Tibshirani, J. Friedman // *Springer*. — 2001. — Pp. 269–272.
 - 15 *Quinlan, J. R.* Machine Learning: an artificial intelligence approach. Learning efficient classification procedures / J. R. Quinlan. — Carbonell Mitchell, 1983. — Pp. 463–482.
 - 16 *Quinlan, J. R.* Improved use of continuous attributes in c4.5 / J. R. Quinlan // *Journal of Artificial Intelligence Research*. — 1996. — Pp. 77–90.
 - 17 *Мартин, Н.* Математическая теория энтропии / Н. Мартин, Д. Ингленд. — Москва: Мир, 1988. — 350 pp.
 - 18 *Шеннон, К.* Работы по теории информации и кибернетике / К. Шеннон. — Москва, 1963. — С. 243–332.
 - 19 *Уикем, Х.* Язык R в задачах науки о данных: импорт, подготовка, обработка, визуализация и моделирование данных / Х. Уикем, Г. Гроулмунд. — Вильямс, 2017. — 592 с.
 - 20 *Коэльё, Л. П.* Построение систем машинного обучения на языке Python / Л. П. Коэльё, В. Ричерт. — Москва: ДМК Пресс, 2016. — 202 с.
 - 21 *Жерон, О.* Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем / О. Жерон. — Вильямс, 2018. — 688 с.

ПРИЛОЖЕНИЕ А

Листинг программы

Код приложения decision-tree.py.

```
from __future__ import division
import math
import copy
import sys
import ast
import csv
from collections import Counter

#####
# data class .csv btrain
#####

class data():
    def __init__(self, classifier):
        self.examples = []
        self.attributes = []
        self.attr_types = []
        self.classifier = classifier
        self.class_index = None

#####
# read data from the .csv files
#####

def read_data(dataset, datafile, datatypes):
    print "Reading data..."
    f = open(datafile)
    original_file = f.read()
    rowsplit_data = original_file.splitlines()
    dataset.examples = [rows.split(',') for rows in rowsplit_data]

    #list attributes
```

```

dataset.attributes = dataset.examples.pop(0)

#create array that indicates whether each attribute is a numerical va
attr_type = open(datatypes)
orig_file = attr_type.read()
dataset.attr_types = orig_file.split(',')

#####
# Preprocess Filtring by attr
#####
def preprocess2(dataset):
    print "Preprocessing data..."

    class_values = [example[dataset.class_index] for example in dataset.e
    class_mode = Counter(class_values)
    class_mode = class_mode.most_common(1)[0][0]

    for attr_index in range(len(dataset.attributes)):

        ex_0class = filter(lambda x: x[dataset.class_index] == '0', datas
        values_0class = [example[attr_index] for example in ex_0class]

        ex_1class = filter(lambda x: x[dataset.class_index] == '1', datas
        values_1class = [example[attr_index] for example in ex_1class]

        values = Counter(values_0class)
        value_counts = values.most_common()

        mode0 = values.most_common(1)[0][0]
        if mode0 == '?':
            mode0 = values.most_common(2)[1][0]

        values = Counter(values_1class)

```

```

model = values.most_common(1)[0][0]

if model == '?':
    model = values.most_common(2)[1][0]

mode_01 = [mode0, model]

attr_modes = [0]*len(dataset.attributes)
attr_modes[attr_index] = mode_01

for example in dataset.examples:
    if (example[attr_index] == '?'):
        if (example[dataset.class_index] == '0'):
            example[attr_index] = attr_modes[attr_index][0]
        elif (example[dataset.class_index] == '1'):
            example[attr_index] = attr_modes[attr_index][1]
        else:
            example[attr_index] = class_mode

#convert attributes that are numeric to floats
for example in dataset.examples:
    for x in range(len(dataset.examples[0])):
        if dataset.attributes[x] == 'True':
            example[x] = float(example[x])

#####
# tree node class (attr_split_idx — bool)
#####

class treeNode():
    def __init__(self, is_leaf, classification, attr_split_index, attr_sp
        self.is_leaf = True
        self.classification = None
        self.attr_split = None
        self.attr_split_index = None

```

```

        self.attr_split_value = None
        self.parent = parent
        self.upper_child = None
        self.lower_child = None
        self.height = None

#####
# compute tree
#####
{
# initialize Tree
    # if dataset is pure or stopping then stop
    # for all attributes a in dataset
        # compute information-theoretic criteria if we split on a
    # abest = best attribute according to above
    # tree = create a decision node that tests abest in the root
    # dv (v=1,2,3,...) = induced sub-datasets from D based on abest
    # for all dv
        # tree = compute_tree(dv)
        # attach tree to the corresponding branch of Tree
    # return tree
}

def compute_tree(dataset, parent_node, classifier):
    node = treeNode(True, None, None, None, parent_node, None, None, 0)
    if (parent_node == None):
        node.height = 0
    else:
        node.height = node.parent.height + 1

    ones = one_count(dataset.examples, dataset.attributes, classifier)
    if (len(dataset.examples) == ones):
        node.classification = 1
        node.is_leaf = True
    return node

```

```

elif (ones == 0):
    node.classification = 0
    node.is_leaf = True
    return node
else:
    node.is_leaf = False

attr_to_split = None
max_gain = 0
split_val = None
min_gain = 0.01
dataset_entropy = calc_dataset_entropy(dataset, classifier)

for attr_index in range(len(dataset.examples[0])):

    if (dataset.attributes[attr_index] != classifier):
        local_max_gain = 0
        local_split_val = None
        attr_value_list = [example[attr_index] for example in dataset]
        attr_value_list = list(set(attr_value_list))
        if(len(attr_value_list) > 100):
            attr_value_list = sorted(attr_value_list)
            total = len(attr_value_list)
            ten_percentile = int(total/10)
            new_list = []
            for x in range(1, 10):
                new_list.append(attr_value_list[x*ten_percentile])
            attr_value_list = new_list

        for val in attr_value_list:
            local_gain = calc_gain(dataset, dataset_entropy, val, attr_index)

            if (local_gain > local_max_gain):
                local_max_gain = local_gain

```

```

        local_split_val = val

    if (local_max_gain > max_gain):
        max_gain = local_max_gain
        split_val = local_split_val
        attr_to_split = attr_index

if (split_val is None or attr_to_split is None):
    print "Something went wrong. Couldn't find an attribute to split"
elif (max_gain <= min_gain or node.height > 20):

    node.is_leaf = True
    node.classification = classify_leaf(dataset, classifier)

    return node

node.attr_split_index = attr_to_split
node.attr_split = dataset.attributes[attr_to_split]
node.attr_split_value = split_val
#SPLIT
upper_dataset = data(classifier)
lower_dataset = data(classifier)
upper_dataset.attributes = dataset.attributes
lower_dataset.attributes = dataset.attributes
upper_dataset.attr_types = dataset.attr_types
lower_dataset.attr_types = dataset.attr_types
for example in dataset.examples:
    if (attr_to_split is not None and example[attr_to_split] >= split_val):
        upper_dataset.examples.append(example)
    elif (attr_to_split is not None):
        lower_dataset.examples.append(example)

node.upper_child = compute_tree(upper_dataset, node, classifier)
node.lower_child = compute_tree(lower_dataset, node, classifier)

```

```

    return node

#####
# Classify dataset
#####

def classify_leaf(dataset, classifier):
    ones = one_count(dataset.examples, dataset.attributes, classifier)
    total = len(dataset.examples)
    zeroes = total - ones
    if (ones >= zeroes):
        return 1
    else:
        return 0

#####
# Calculate the entropy
#####

def calc_dataset_entropy(dataset, classifier):
    ones = one_count(dataset.examples, dataset.attributes, classifier)
    total_examples = len(dataset.examples);

    entropy = 0
    p = ones / total_examples
    if (p != 0):
        entropy += p * math.log(p, 2)
    p = (total_examples - ones)/total_examples
    if (p != 0):
        entropy += p * math.log(p, 2)

    entropy = -entropy
    return entropy

#####

```


Calculate the gain of a particular attribute split

#####

```
def calc_gain(dataset, entropy, val, attr_index):
    classifier = dataset.attributes[attr_index]
    attr_entropy = 0
    total_examples = len(dataset.examples);
    gain_upper_dataset = data(classifier)
    gain_lower_dataset = data(classifier)
    gain_upper_dataset.attributes = dataset.attributes
    gain_lower_dataset.attributes = dataset.attributes
    gain_upper_dataset.attr_types = dataset.attr_types
    gain_lower_dataset.attr_types = dataset.attr_types
    for example in dataset.examples:
        if (example[attr_index] >= val):
            gain_upper_dataset.examples.append(example)
        elif (example[attr_index] < val):
            gain_lower_dataset.examples.append(example)

    if (len(gain_upper_dataset.examples) == 0 or len(gain_lower_dataset.e
        return -1

    attr_entropy += calc_dataset_entropy(gain_upper_dataset,
                                          classifier)*len(gain_upper_datas
    attr_entropy += calc_dataset_entropy(gain_lower_dataset,
                                          classifier)*len(gain_lower_datas

    return entropy - attr_entropy
```

#####

count number of examples with classification "1"

#####

```
def one_count(instances, attributes, classifier):
    count = 0
    class_index = None
```

```

#find index of classifier
for a in range(len(attributes)):
    if attributes[a] == classifier:
        class_index = a
    else:
        class_index = len(attributes) - 1
for i in instances:
    if i[class_index] == "1":
        count += 1
return count

#####
# Prune tree
#####
def prune_tree(root, node, dataset, best_score):
    if (node.is_leaf == True):
        classification = node.classification
        node.parent.is_leaf = True
        node.parent.classification = node.classification
        if (node.height < 20):
            new_score = validate_tree(root, dataset)
        else:
            new_score = 0

        if (new_score >= best_score):
            return new_score
        else:
            node.parent.is_leaf = False
            node.parent.classification = None
            return best_score
    else:
        new_score = prune_tree(root, node.upper_child, dataset, best_score)
        if (node.is_leaf == True):
            return new_score

```

```

        new_score = prune_tree(root, node.lower_child, dataset, new_score)
    if (node.is_leaf == True):
        return new_score

    return new_score

#####
# Validate tree
#####

def validate_tree(node, dataset):
    total = len(dataset.examples)
    correct = 0
    for example in dataset.examples:
        correct += validate_example(node, example)
    return correct/total

#####
# Validate example
#####

def validate_example(node, example):
    if (node.is_leaf == True):
        projected = node.classification
        actual = int(example[-1])
        if (projected == actual):
            return 1
        else:
            return 0
    value = example[node.attr_split_index]
    if (value >= node.attr_split_value):
        return validate_example(node.upper_child, example)
    else:
        return validate_example(node.lower_child, example)

#####

```

```

# Test example
#####

def test_example(example, node, class_index):
    if (node.is_leaf == True):
        return node.classification
    else:
        if (example[node.attr_split_index] >= node.attr_split_value):
            return test_example(example, node.upper_child, class_index)
        else:
            return test_example(example, node.lower_child, class_index)

#####

# Print tree
#####

def print_tree(node):
    if (node.is_leaf == True):
        for x in range(node.height):
            print "\t",
        print "Classification: " + str(node.classification)
        return
    for x in range(node.height):
        print "\t",
    print "Split index: " + str(node.attr_split)
    for x in range(node.height):
        print "\t",
    print "Split value: " + str(node.attr_split_value)
    print_tree(node.upper_child)
    print_tree(node.lower_child)

#####

# Print tree in disjunctive normal form
#####

def print_disjunctive(node, dataset, dnf_string):
    if (node.parent == None):

```

```

        dnf_string = "( "
    if (node.is_leaf == True):
        if (node.classification == 1):
            dnf_string = dnf_string[:-3]
            dnf_string += ") ^ "
            print dnf_string,
        else:
            return
    else:
        upper = dnf_string + str(dataset.attributes[node.attr_split_index])
        print_disjunctive(node.upper_child, dataset, upper)
        lower = dnf_string + str(dataset.attributes[node.attr_split_index])
        print_disjunctive(node.lower_child, dataset, lower)
    return

#####
# main function
#####

def main():
    args = str(sys.argv)
    args = ast.literal_eval(args)
    if (len(args) < 2):
        print "You have input less than the minimum number of argument!"
    elif (args[1][-4:] != ".csv"):
        print "Your training file must be .csv!"
    else:
        datafile = args[1]
        dataset = data("")
        if ("-d" in args):
            datatypes = args[args.index("-d") + 1]
        else:
            datatypes = 'datatypes.csv'
        read_data(dataset, datafile, datatypes)

```

```

arg3 = args[2]
if (arg3 in dataset.attributes):
    classifier = arg3
else:
    classifier = dataset.attributes[-1]

dataset.classifier = classifier

for a in range(len(dataset.attributes)):
    if dataset.attributes[a] == dataset.classifier:
        dataset.class_index = a
    else:
        dataset.class_index = range(len(dataset.attributes))[-1]

unprocessed = copy.deepcopy(dataset)
preprocess2(dataset)

print "Computing tree..."
root = compute_tree(dataset, None, classifier)
if ("-s" in args):
    print_disjunctive(root, dataset, "")
    print "\n"
if ("-v" in args):
    datavalidate = args[args.index("-v") + 1]
    print "Validating tree..."

    validateset = data(classifier)
    read_data(validateset, datavalidate, datatypes)
    for a in range(len(dataset.attributes)):
        if validateset.attributes[a] == validateset.classifier:
            validateset.class_index = a
        else:
            validateset.class_index = range(len(validateset.attributes))[-1]
    preprocess2(validateset)

```

```

best_score = validate_tree(root, validateset)
all_ex_score = copy.deepcopy(best_score)
print "Initial (pre-pruning) validation set score: " + str(100*all_ex_score)
if ("-p" in args):
    if ("-v" not in args): # must be with each other
        print "Error: You must validate if you want to prune"
    else:
        post_prune_accuracy = 100*prune_tree(root, root, validateset)
        print "Post-pruning score on validation set: " + str(100*post_prune_accuracy)
if ("-t" in args):
    datatest = args[args.index("-t") + 1]
    testset = data(classifier)
    read_data(testset, datatest, datatypes)
    for a in range(len(dataset.attributes)):
        if testset.attributes[a] == testset.classifier:
            testset.class_index = a
        else:
            testset.class_index = range(len(testset.attributes))[-1]
    print "Testing model on " + str(datatest)
    for example in testset.examples:
        example[testset.class_index] = '0'
    testset.examples[0][testset.class_index] = '1'
    testset.examples[1][testset.class_index] = '1'
    testset.examples[2][testset.class_index] = '?'
    preprocess2(testset)
    b = open('results.csv', 'w')
    a = csv.writer(b)
    for example in testset.examples:
        example[testset.class_index] = test_example(example, root)
    saveset = testset
    saveset.examples = [saveset.attributes] + saveset.examples
    a.writerow(saveset.examples)
    b.close()
    print "Testing complete. Results outputted to results.csv"

```

```
if __name__ == "__main__":  
    main()
```
