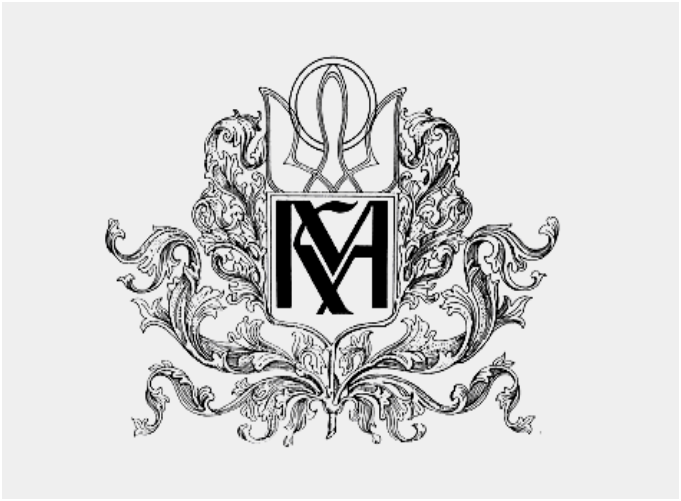
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики



Курсова робота на тему

«Дерева рішень і алгоритми їх побудови»

Керівник курсової роботи

д.т.н., доц. Глибовець А. М.

_____

(підпис)

“05” травня 2020 р.

Виконав
студент І курсу МП
факультету інформатики
Наквасюк Василь

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Декан кафедри інформатики,

к.ф-м.н., доц. Гороховський С. С.

_____

(підпис)

„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Наквасюку Василю Васильовичу факультету інформатики 1 курсу
ТЕМА Дерева рішень і алгоритми їх побудови

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Вступ
РОЗДІЛ 1: Дерева рішень
РОЗДІЛ 2: Процес побудови дерева рішень
РОЗДІЛ 3: Програмна реалізація алгоритму
Висновки
Список використаної літератури

Дата видачі „___” _____ 2020 р. Керівник _____

(підпис)

Тема: Дерева рішень і алгоритми їх побудови

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу
1.	Отримання завдання на дипломну роботу.	15.12.2019
2.	Огляд технічної літератури за темою роботи.	05.01.2020
3.	Аналіз існуючих рішень.	26.01.2020
3.	Написання вступу та плану роботи.	15.02.2020
4.	Програмна реалізація алгоритмів.	01.03.2020
6.	Написання основних розділів роботи.	20.03.2020
7.	Створення слайдів для доповіді та написання доповіді.	11.04.2020
8.	Корегування роботи відповідно до вимог щодо оформлення робіт.	20.04.2020
9.	Остаточне оформлення пояснювальної роботи та слайдів.	25.04.2020
10.	Корегування роботи згідно із зауваженнями керівника.	05.05.2020
11.	Захист курсової роботи.	22.05.2020

Студент Наквасюк В.В.

Керівник Глибовець А. М.

“_____”_____

ЗМІСТ

Анотація

У даній курсовій роботі розглянуто дерева рішень, їх побудова та декілька алгоритмів навчання дерева рішень: ID3 (Iterative Dichotomizer 3), C4.5, CART (Classification and Regression Tree) і їх переваги і недоліки в методах класифікації даних. Також досліджено реальний приклад з використанням CART алгоритму та мови програмування Python.

Ключові слова: дерева рішень, ID3, C4.5, CART.

Вступ

Актуальність теми. Задачі класифікації є досить поширеною в машинному навчанні і мають багато місць використання. Дерева рішень, як один із класифікаторів, успішно використовуються на практиці в таких областях, як:

- Банківська справа (оцінка кредитоспроможності клієнтів банку при видачі кредитів)
- Промисловість (контроль за якістю продукції (виявлення дефектів), випробування без руйнувань (наприклад, перевірка якості зварювання) і т.д.)
- Медицина (діагностика захворювань)
- Молекулярна біологія (аналіз будови амінокислот)
- Торгівля (класифікація клієнтів і товарів)

Мета дослідження. Метою є дослідження декількох методів побудови дерева рішень, їх реалізація і порівняння.

Для досягнення мети дослідження передбачаються такі кроки:

- дослідження джерел
- аналіз теоретичних знань
- реалізація алгоритмів побудови дерев рішень
- практичне застосування на реальному прикладі

Об’єкт дослідження. Дерева рішень, алгоритми ID3, C4.5, CART.

Предмет дослідження. Основні алгоритми даних методів, структури вхідних даних, основні переваги і способи використання.

Джерела дослідження. Електронні ресурси.

Розділ 1. Дерева рішень

Основи

Дерева рішень є одним з найбільш ефективних інструментів інтелектуального аналізу даних і аналітики, які дозволяють вирішувати завдання класифікації і регресії.

Вони є ієрархічними деревовидними структурами, що складаються з правил рішень виду "Якщо ..., тоді .". Правила автоматично генеруються в процесі навчання на навчальній множині і, оскільки вони формулюються практично на природній мові (наприклад, "Якщо об'єм продажів більше 1000 шт., то товар перспективний"), дерева рішень як аналітичні моделі більш зрозумілі і прозорі, чим, скажімо, нейронні мережі.

Оскільки правила в деревах рішень виходять шляхом узагальнення множини окремих спостережень (навчальних прикладів), що описують предметну область, то по аналогії з відповідним методом логічного висновку їх називають індуктивними правилами, а сам процес навчання - індукцією дерев рішень. У навчальній множині для прикладів має бути задане цільове значення, оскільки дерева рішень є моделями, що будуються на основі навчання з учителем. При цьому, якщо цільова змінна дискретна, то модель називають деревом класифікації, а якщо безперервна, то деревом регресії. Засадничі ідеї, що послужили поштовхом до появи і розвитку дерев рішень, були закладені в 1950-х роках в області досліджень моделювання людської поведінки за допомогою комп'ютерних систем. Серед них слід виділити роботи К. Ховеленда "Комп'ютерне моделювання мислення" [1] і Е. Ханта та ін. "Експерименти по індукції" [2]. Подальший розвиток дерев рішень як

самонавчальних моделей для аналізу даних пов'язано з іменами Джона Р. Куінлена [3], який розробив алгоритм ID3 і його вдосконалені модифікації C4.5 і C5.0, а також Лео Бреймана [4], який запропонував алгоритм CART і метод випадкового лісу.

Термінологія

Введемо в розгляд основні поняття, використовувані в теорії дерев рішень:

Об'єкт — приклад, шаблон, спостереження.

Атрибут — ознака, незалежна змінна, властивість.

Цільова змінна — залежна змінна, мітка класу.

Вузол — внутрішній вузол дерева, вузол перевірки.

Кореневий вузол — початковий вузол дерева рішень.

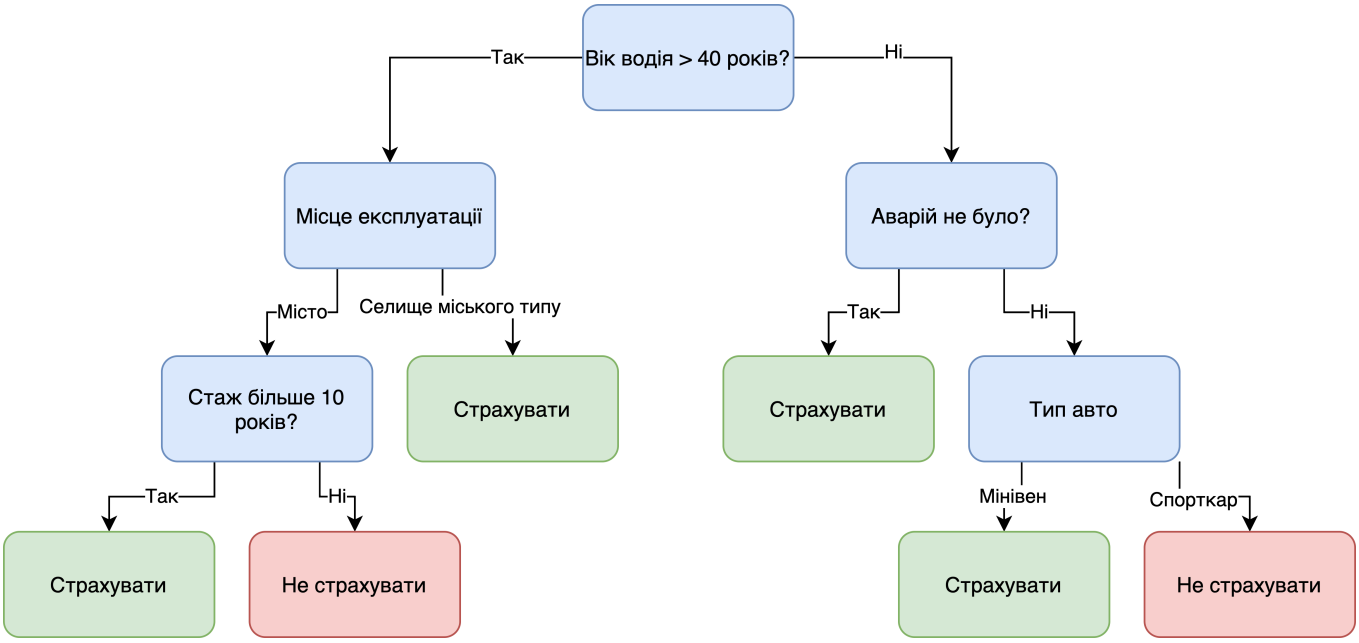
Лист — кінцевий вузол дерева, вузол рішення, термінальний вузол.

Вирішальна правило — умова у вузлі, перевірка.

Структура дерева рішень

Власне, саме дерево рішень - це метод представлення вирішальних правил в ієрархічній структурі, що складається з елементів двох типів - вузлів (node) і листя (leaf). У вузлах знаходяться вирішальні правила і робиться перевірка відповідності прикладів цьому правилу по якому-небудь атрибуту навчальної множини. У простому випадку, в результаті перевірки, безліч прикладів, що потрапили у вузол, розбивається на дві підмножини, в одне з

яких потрапляють приклади, що задовольняють правилу, а в інше — що не задовольняють.



Потім до кожної підмножини знову застосовується правило і процедура рекурсивно повторюється доки не буде досягнута деяка умова зупинки алгоритму. В результаті в останньому вузлі перевірка і розбиття не робиться і він оголошується листом. Лист визначає рішення для кожного прикладу, що потрапив в нього. Для дерева класифікації - це клас, що асоціюється з вузлом, а для дерева регресії - модальний інтервал цільової змінної, що відповідає листу.

Таким чином, на відміну від вузла, в листі міститься не правило, а підмножина об'єктів, що задовольняють усім правилам гілки, яка закінчується цим листом. Очевидно, щоб потрапити в лист, приклад повинен задовольняти усім правилам, що лежать на шляху до цього листа. Оскільки шлях в дереві до кожного листа єдиний, то і кожен приклад може потрапити тільки в один лист, що забезпечує єдиність рішення.

Застосування

Основна сфера застосування дерев рішень — підтримка процесів ухвалення управлінських рішень, використовувана в статистиці, аналізі даних і машинному навчанні. Завданнями, що вирішуються за допомогою цього апарату, є:

Класифікація - віднесення об'єктів до одного із заздалегідь відомих класів. Цільова змінна повинна мати дискретні значення.

Регресія — визначення числового значення незалежній змінній для заданого вхідного вектору.

Опис об'єктів — набір правил в дереві рішень дозволяє компактно описувати об'єкти. Тому замість складних структур, що описують об'єкти, можна зберігати дерева рішень.

Розділ 2. Процес побудови дерева рішень

Алгоритми побудови

Процес побудови дерев рішень полягає в послідовному, рекурсивному розбитті навчальної великої кількості на підмножини із застосуванням вирішальних правил у вузлах. Процес розбиття триває до тих пір, поки усі вузли у кінці усіх гілок не будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він міститиме єдиний об'єкт, або об'єкти тільки одного класу), або після досягнення деякої умови зупинки, що задається користувачем (наприклад, мінімальне допустиме число прикладів у вузлі або максимальна глибина дерева).

Алгоритми побудови дерев рішень відносять до категорії так званих жадібних алгоритмів. Жадібними називаються алгоритми, які допускають, що локально-оптимальні рішення на кожному кроці (розбиття у вузлах), призводять до оптимального вихідного рішення. У випадку дерев рішень це означає, що якщо один раз був вибраний атрибут, і по ньому було зроблено розбиття на підмножини, то алгоритм не може повернутися назад і вибрати інший атрибут, який дав би краще підсумкове розбиття. Тому на етапі побудови не можна сказати чи забезпечить вибраний атрибут, зрештою, оптимальне розбиття.

У основі більшості популярних алгоритмів навчання дерев рішень лежить принцип "розділяй і володарюй". Алгоритмічно цей принцип реалізується таким чином. Нехай задана навчальна множина S, що містить n прикладів, для кожного з яких задана мітка класу C_i (i=1..k), і m атрибутів A_j

(j=1..m), які, як передбачається, визначають приналежність об'єкту до того або іншого класу. Тоді можливі три випадки:

1. Усі приклади множин S мають однакову мітку класу C_i (тобто усі навчальні приклади відносяться тільки до одного класу). Очевидно, що навчання в цьому випадку не має сенсу, оскільки усі приклади, що пред'являються моделі, будуть одного класу, який і "навчиться" розпізнавати модель. Саме дерево рішень в цьому випадку буде листом, що асоціюється з класом C_i. Практичне використання такого дерева безглузде, оскільки будь-який новий об'єкт воно відноситиме тільки до цього класу.
2. Множина S взагалі не містить прикладів, тобто є порожньою множиною. В цьому випадку для нього теж буде створений лист (застосовувати правило, щоб створити вузол, до порожньої множини безглуздо), клас якого буде вибраний з іншої множини (наприклад, клас, який найчастіше зустрічається у батьківській множині).
3. Множина S містить навчальні приклади усіх класів C_k . В цьому випадку вимагається розбити множини S на підмножини, що асоціюються з класами. Для цього вибирається один з атрибутів A_j множини S який містить два і більше унікальних значення (a_1, a_2, ... , a_p), де p - число унікальних значень ознаки. Потім множина S розбивається на p підмножин (S_1, S_2, ... , S_p), кожне з яких включає приклади, що містять відповідне значення атрибуту. Потім вибирається наступний атрибут і розбиття повторюється. Це процедура рекурсивно повторюватиметься до тих пір, поки усі приклади в результуючих підмножинах не виявляться одного класу.

Описана вище процедура лежить в основі багатьох сучасних алгоритмів побудови дерев рішень. Очевидно, що при використанні цієї методики, побудова дерева рішень відбуватиметься зверху вниз (від кореневого вузла до листя). Нині розроблено значне число алгоритмів навчання дерева рішень : ID3, CART, C4.5, C5.0, NewId, ITrule, CHAID, CN2 і так далі. Але найбільше поширення і популярність отримали наступні:

ID3 (Iterative Dichotomizer 3) – алгоритм дозволяє працювати тільки з дискретною цільовою змінною, тому дерева рішень, побудовані за допомогою цього алгоритму, є такими, що класифікують. Число нащадків у вузлі дерева не обмежене. Не може працювати з пропущеними даними. У основі цього алгоритму лежить поняття інформаційної ентропії - тобто, міра невизначеності інформації. Для того, щоб визначити наступний атрибут, необхідно підрахувати ентропію усіх невикористаних ознак відносно тестових зразків і вибрати той, для якого ентропія мінімальна. Цей атрибут і вважатиметься найбільш доцільною ознакою класифікації.

C4.5 – вдосконалена версія алгоритму ID3, в яку додана можливість роботи з пропущеними значеннями атрибутів. Він дозволяє "усікати" гілки дерева, якщо воно занадто сильно "розростається", а також працювати не лише з атрибутами-категоріями, але і з числовими. Загалом, сам алгоритм виконується за тим же принципом, що і його попередник; відмінність полягає в можливості розбиття області значень незалежної числової змінної на декілька інтервалів, кожен з яких буде атрибутом. Відповідно до цього початкова множина ділиться на підмножини. Зрештою, якщо дерево виходить занадто великим, можливе зворотне угрупування декількох вузлів в один лист. При цьому, оскільки перед побудовою дерева помилка класифікації вже врахована, вона не збільшується.

CART (Classification and Regression Tree) – алгоритм навчання дерев рішень, що дозволяє використати як дискретну, так і безперервну цільову змінну, тобто вирішувати як завдання класифікації, так і регресії. CART розроблений в цілях побудови так званих бінарних дерев рішень - тобто тих дерев, кожен вузол яких при розбитті "дає" тільки двох нащадків. Грубо кажучи, алгоритм діє шляхом розділення на кожному кроці безлічі прикладів рівно навпіл — по одній гілці йдуть ті приклади, в яких правило виконується (правий нащадок), по іншій - ті, в яких правило не виконується (лівий нащадок). Таким чином, в процесі "зростання" на кожному вузлі дерева алгоритм проводить перебір усіх атрибутів, і вибирає для наступного розбиття той, який максимізував значення показника, що обчислюється за математичною формулою і залежного від стосунків числа прикладів в правому і лівому нащадку до загального числа прикладів.

Основні етапи побудови дерева рішень

В ході побудови дерева рішень треба вирішити декілька основних проблем, з кожною з яких пов'язаний відповідний крок процесу навчання:

- Вибір атрибуту, по якому робитиметься розбиття в цьому вузлі (атрибуту розбиття)
- Вибір критерію зупинки навчання
- Вибір методу відсікання гілок (спрощення)
- Оцінка точності побудованого дерева.

Розділ 3. Програмна реалізація CART алгоритму

Реальний приклад на основі задачі про платних клієнтів

Розглянемо реальний приклад використання дерева рішень на задачі прогнозування, чи буде користувач платним клієнтом сайту, чи ні. Так як алгоритм являється собою навчанням з учителем, то в нас має бути вхідна вибірка (данні), на якій ми будемо тренувати наше дерево рішень.

Звідки прийшов?	Країна	Читав FAQ?	Скільки проглянув сторінок?	Який пакет послуг вибрав?
Slashdot	США	Так	18	Ні
Google	Франція	Так	23	Преміум
Digg	США	Так	24	Базовий
Kiwitobes	Франція	Так	23	Базовий
Google	Великобританія	Ні	21	Преміум
direct	Нова Зеландія	Ні	12	Ні
direct	Великобританія	Ні	21	Базовий
Google	США	Ні	24	Преміум
Slashdot	Франція	Так	19	Ні
Digg	США	Ні	18	Ні
Google	Великобританія	Ні	18	Ні
…	…	…	…	…

Для написання програмного коду використаємо мову Python.

```
my_data=[['slashdot','USA','yes',18,'None'],
         ['google','France','yes',23,'Premium'],
         ['digg','USA','yes',24,'Basic'],
         ['kiwitobes','France','yes',23,'Basic'],
         ['google','UK','no',21,'Premium'],
         ['(direct)','New Zealand','no',12,'None'],
         ['(direct)','UK','no',21,'Basic'],
         ['google','USA','no',24,'Premium'],
         ['slashdot','France','yes',19,'None'],
         ['digg','USA','no',18,'None'],
         ['google','UK','no',18,'None'],
         ['kiwitobes','UK','no',19,'None'],
         ['digg','New Zealand','yes',12,'Basic'],
         ['slashdot','UK','no',21,'None'],
         ['google','UK','yes',18,'Basic'],
         ['kiwitobes','France','yes',19,'Basic']]
```

Спочатку створимо представлення дерева. Створимо новий клас *decisionnode*, за допомогою якого ми будемо формувати один вузол дерева:

```
class decisionnode:
  def __init__(self,col=-1,value=None,results=None,tb=None,fb=None):
    self.col=col
    self.value=value
    self.results=results
    self.tb=tb
    self.fb=fb
```

В кожному вузлі маємо 5 змінних екземпляра класу:

- *col* – індекс стовбця умови, яку перевіряємо.
- *value* – значення, якому має відповідати значення в стовбці, щоб результат був рівний *true*.
- *tb* і *fb* – екземпляри класу *decisionnodes*, в яких відбувається перехід у випадку, якщо результат позитивний, або негативний відповідно.
- *results* – словник результатів для цієї гілки. Значення рівне *None* для всіх вузлів, окрім листових.

Тут використаємо алгоритм CART (Classification and Regression Trees - дерева класифікації і регресії). Для побудови дерева рішень алгоритм спочатку створює кореневий вузол. Розглянувши усі спостереження в таблиці, він вибирає найкращу змінну, по якій можна розбивати дані на дві частини. Функція *dividset* розбиває множину рядків на дві підмножини виходячи з даних в одному стовбці. На вході вона приймає список рядків, номер стовпця і значення, по якому ділити стовпець. У разі прочитання FAQ можливі значення - "Так або ні", а для стовпця "Звідки прийшов?" є декілька можливостей.

```
# Divides a set on a specific column. Can handle numeric
# or nominal values
def divideset(rows,column,value):
   # Make a function that tells us if a row is in
   # the first group (true) or the second group (false)
   split_function=None
   if isinstance(value,int) or isinstance(value,float):
      split_function=lambda row:row[column]>=value
   else:
      split_function=lambda row:row[column]==value

   # Divide the rows into two sets and return them
   set1=[row for row in rows if split_function(row)]
   set2=[row for row in rows if not split_function(row)]
   return (set1,set2)
```

В цьому коді ми створюємо функцію *split_function*, яка розбиває дані на дві частини. Це робиться по різному в залежності від того, чи є множина значень неперервною (*float*) чи дискретною (*int*).

Використовуючи це розділення, наприклад по ознаці «Читав FAQ?», на нашій навчальній виборці маємо наступне:

```
$ python
>>> import treepredict
>>> treepredict.divideset(treepredict.my_data,2,'yes')
([['slashdot', 'USA', 'yes', 18, 'None'],
  ['google', 'France', 'yes', 23, 'Premium'],...]]
[['google', 'UK', 'no', 21, 'Premium'],
  ['(direct)', 'New Zealand', 'no', 12,
'None'],...])
```

True	False
Ні	Преміум
Преміум	Ні
Базовий	Базовий
Базовий	Преміум
Ні	Ні
Базовий	Ні
Базовий	Ні

Як бачимо, ми вибрали не дуже вдалу ознаку для розділення нашої множини на підмножини.

Вибір найкращого розбиття

Зроблене нами неформальне спостереження про те, що змінна вибрана не дуже добре, може бути і вірним, але для реалізації програми потрібний спосіб виміру неоднорідності множини. Нам потрібно знайти таку змінну, щоб множини якомога менше перетиналися. Перше, що нам знадобиться, — це функція для обчислення того, скільки разів кожен результат представлений у множині рядків.

```
# Create counts of possible results (the last column of
# each row is the result)
def uniquecounts(rows):
   results={}
   for row in rows:
      # The result is the last column
      r=row[len(row)-1]
      if r not in results: results[r]=0
      results[r]+=1
   return results
```

Функція *uniquecounts* повертає словник, в якому для кожного результату вказано, скільки разів він зустрівся. Вона буде використовуватись в подальшому для обчислення неоднорідності множини. Для цього існує декілька метрик, ми розглянемо дві з них: коефіцієнт Джини і ентропію.

Коефіцієнт Джині

Припустимо, що є множина зразків, що належать декільком категоріям. Коефіцієнтом Джині називається вірогідність того, що при випадковому виборі зразка і категорії виявиться, що зразок не належить до вказаної категорії. Якщо усі зразки в множині належать до однієї і тієї ж категорії, то гіпотеза завжди буде вірна, тому вірогідність помилки дорівнює 0. Якщо в групі в рівній пропорції представлені чотири можливі результати, то в 75% випадків гіпотеза виявиться невірною, тому коефіцієнт помилки дорівнює 0,75. Нижче показана функція для обчислення коефіцієнта Джині:

```
# Probability that a randomly placed item will
# be in the wrong category
def giniimpurity(rows):
  total=len(rows)
  counts=uniquecounts(rows)
  imp=0
  for k1 in counts:
    p1=float(counts[k1])/total
    for k2 in counts:
      if k1==k2: continue
      p2=float(counts[k2])/total
      imp+=p1*p2
  return imp
```

Ця функція обчислює вірогідність кожного з можливих результатів шляхом ділення лічильника появ цього результату на загальне число рядків в множині. Це дає повну вірогідність того, що для випадково вибраного рядка буде спрогнозований не той результат, який насправді має місце. Чим вище ця вірогідність, тим гірше розбиття. Вірогідність 0 - це ідеал, оскільки в цьому випадку усі рядки вже розподілені правильно.

Ентропія

У теорії інформації ентропією називають міру хаотичності множини — по суті кажучи, міру його неоднорідності.

```
# Entropy is the sum of p(x)log(p(x)) across all
# the different possible results
def entropy(rows):
    from math import log
    log2=lambda x:log(x)/log(2)
    results=uniquecounts(rows)
    # Now calculate the entropy
    ent=0.0
    for r in results.keys( ):
        p=float(results[r])/len(rows)
        ent=ent-p*log2(p)
    return ent
```

Функція *entropy* обчислює частоту входження кожного зразка (кількість його входжень, поділена на загальне число зразків — рядків).

Ця величина є мірою того, наскільки результати відрізняються один від одного. Якщо усі вони однакові (наприклад, вам повезло і усі користувачі оформили преміальну підписку), то ентропія дорівнює 0. Чим менш однорідні групи, тим вище їх ентропія. Наша мета полягає в тому, щоб розбити дані на дві нові групи, так щоб ентропія зменшилася. Протестуємо метрики за допомогою коефіцієнта Джини та ентропії:

```
>>> treepredict.giniimpurity(treepredict.my_data)
0.6328125
>>> treepredict.entropy(treepredict.my_data)
1.5052408149441479
>>> set1, set2 = treepredict.divideset(treepredict.my_data,2,'yes')
>>> treepredict.entropy(set1)
1.2987949406953985
>>> treepredict.giniimpurity(set1)
0.53125
```

Основна відмінність ентропії від коефіцієнта Джині в тому, що ентропія виходить на максимум повільніше. Тому вона штрафує неоднорідні множини трохи сильніше. Далі будемо користуватися ентропією, оскільки ця метрика частіше вживається.

Рекурсивна побудова дерева

Щоб оцінити, наскільки якісно вибраний атрибут, алгоритм спочатку обчислює ентропію усієї групи. Потім він намагається розбити групу по можливих значеннях кожного атрибуту і обчислює ентропію двох нових груп. Для визначення того, який атрибут дає найкраще розбиття, обчислюється інформаційний виграш, тобто різниця між поточною ентропією і середньозваженою ентропією двох нових груп. Він обчислюється для кожного атрибуту, після чого вибирається той, для якого інформаційний виграш максимальний. Визначивши умову для кореневого вузла, алгоритм створює дві гілки: по одній потрібно буде йти, коли умова *true*, по іншій — коли *false*.

Таким чином, обчислюючи для кожного вузла найкращий атрибут і розщеплюючи гілки, алгоритм створює дерево. Зростання гілки

припиняється, коли інформаційний виграш, отриманий від розщеплювання в цьому вузлі, виявляється менше або дорівнює нулю.

```
def buildtree(rows,scoref=entropy):
  if len(rows)==0: return decisionnode()
  current_score=scoref(rows)

  # Set up some variables to track the best criteria
  best_gain=0.0
  best_criteria=None
  best_sets=None

  column_count=len(rows[0])-1
  for col in range(0,column_count):
    # Generate the list of different values in
    # this column
    column_values={}
    for row in rows:
       column_values[row[col]]=1
    # Now try dividing the rows up for each value
    # in this column
    for value in column_values.keys():
      (set1,set2)=divideset(rows,col,value)

      # Information gain
      p=float(len(set1))/len(rows)
      gain=current_score-p*scoref(set1)-(1-p)*scoref(set2)
      if gain>best_gain and len(set1)>0 and len(set2)>0:
        best_gain=gain
        best_criteria=(col,value)
        best_sets=(set1,set2)
  # Create the sub branches
  if best_gain>0:
    trueBranch=buildtree(best_sets[0])
    falseBranch=buildtree(best_sets[1])
    return decisionnode(col=best_criteria[0],value=best_criteria[1],
                        tb=trueBranch,fb=falseBranch)
  else:
    return decisionnode(results=uniquecounts(rows))
```

При першому виклику функції *buildtree* передається увесь список рядків. Вона в циклі перебирає усі стовпці (окрім останнього, в якому зберігається результат) і по кожному значенню, присутньому в поточному стовпці, розбиває множину рядків на дві підмножини. Для кожної пари підмножин обчислюється середньозважена ентропія, для чого ентропія підмножини множиться на число рядків, що потрапили в нього. Запам'ятовується, у якої пари ця ентропія виявилася найнижчою.

Якщо середньозважена ентропія найкращої пари підмножин не менша, ніж у поточної множини, то зростання цієї гілки припиняється і зберігаються лічильники можливих результатів. Інакше для кожної підмножини знову викликається buildtree і результати виклику приєднуються до гілок *true* і *false*, що виходить з поточного вузла. У результаті буде побудовано усе дерево.

```
>>> tree = treepredict.buildtree(treepredict.my_data)
```

Зараз в змінній *tree* знаходиться навчене дерево рішень.

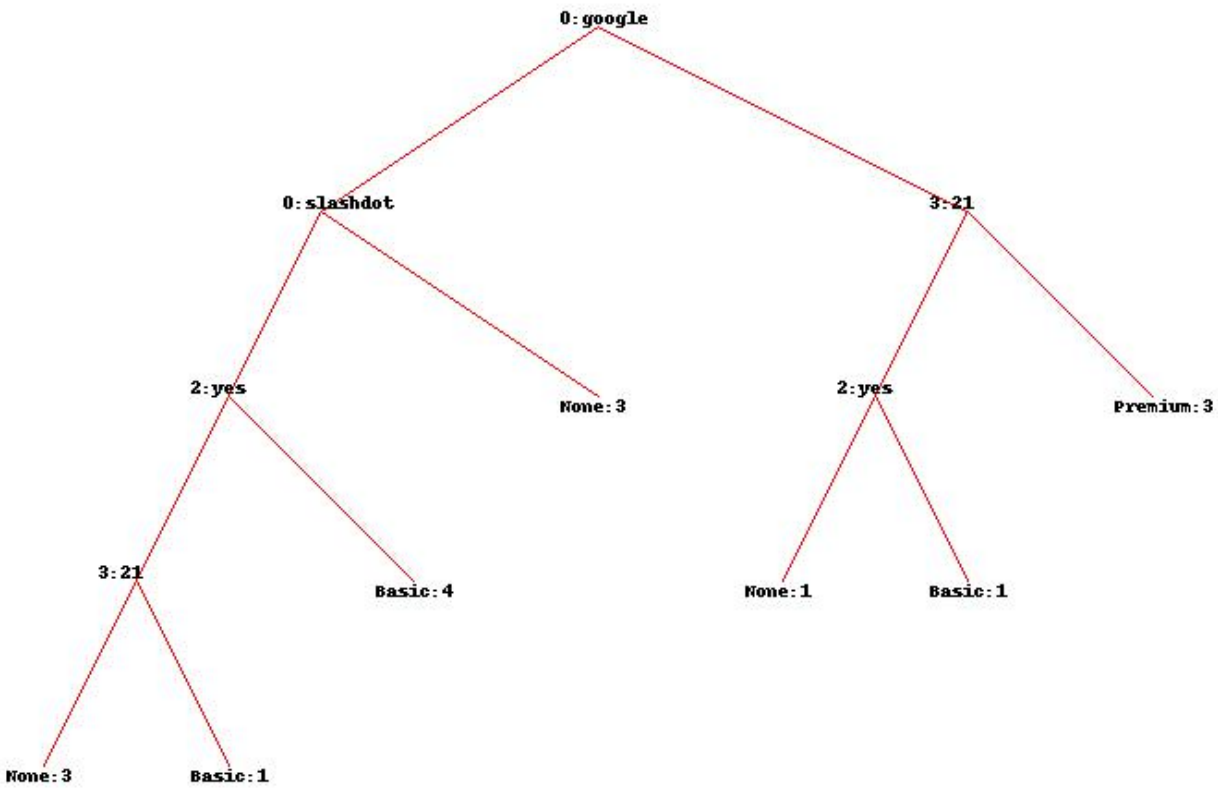
Можемо спробувати візуалізувати наше дерево рішень після розбиття використавши наступну функцію:

```
def getwidth(tree):
  if tree.tb==None and tree.fb==None: return 1
  return getwidth(tree.tb)+getwidth(tree.fb)

def getdepth(tree):
  if tree.tb==None and tree.fb==None: return 0
  return max(getdepth(tree.tb),getdepth(tree.fb))+1


from PIL import Image,ImageDraw

def drawtree(tree,jpeg='tree.jpg'):
  w=getwidth(tree)*100
  h=getdepth(tree)*100+120

  img=Image.new('RGB',(w,h),(255,255,255))
  draw=ImageDraw.Draw(img)

  drawnode(draw,tree,w/2,20)
  img.save(jpeg,'JPEG')

def drawnode(draw,tree,x,y):
  if tree.results==None:
    # Get the width of each branch
    w1=getwidth(tree.fb)*100
    w2=getwidth(tree.tb)*100

    # Determine the total space required by this node
    left=x-(w1+w2)/2
    right=x+(w1+w2)/2

    # Draw the condition string
    draw.text((x-20,y-10),str(tree.col)+':'+str(tree.value),(0,0,0))

    # Draw links to the branches
    draw.line((x,y,left+w1/2,y+100),fill=(255,0,0))
    draw.line((x,y,right-w2/2,y+100),fill=(255,0,0))

    # Draw the branch nodes
    drawnode(draw,tree.fb,left+w1/2,y+100)
    drawnode(draw,tree.tb,right-w2/2,y+100)
  else:
    txt=' \n'.join(['%s:%d'%v for v in tree.results.items()])
    draw.text((x-20,y),txt,(0,0,0))
```

Запустимо нашу функцію *drawtree*:

```
>>> treepredict.drawtree(tree,jpeg='treeview.jpg')
```

Тоді наше дерево буде мати вигляд:



Класифікація нових даних

Тепер нам потрібна функція, яка класифікує нове спостереження відповідно до дерева рішень.

```
def classify(observation,tree):
  if tree.results!=None:
    return tree.results
  else:
    v=observation[tree.col]
    branch=None
    if isinstance(v,int) or isinstance(v,float):
      if v>=tree.value: branch=tree.tb
      else: branch=tree.fb
    else:
      if v==tree.value: branch=tree.tb
      else: branch=tree.fb
    return classify(observation,branch)
```

Ця функція обходить дерево. Після кожного виклику перевіряється, чи досягнутий кінець дерева, тобто чи є у вузлі список результатів *results*. Якщо ні, то для цього спостереження перевіряється умова в поточному вузлі. Якщо воно виконане, то *classify* рекурсивно викликається для гілки *true*, інакше — для гілки *false*.

Давайте спробуємо використати нашу функцію-класифікатор для нових даних і побачимо, як наше дерево рішень спрогнозує, яким будем тип нашого користувача:

```
>>> treepredict.classify(['(direct)','USA','yes',5],tree)
{'Basic': 4}
```

Отже, ми маємо функції для створення дерева рішень по набору даних, для відображення і інтерпретації дерева і для класифікації нових спостережень. Ці функції можна застосувати до будь-якого набору даних, що складається з декількох рядків, кожна з яких містить спостереження і результат.

Висновки

Отже, під час дослідження було розглянуто різні алгоритми побудови дерева рішень, а також наведено реальний приклад з використанням мови програмування Python. В цій роботі навмисно не використовувались такі потужні бібліотеки як scikit-learn, щоб наявно показати покроково як будується дерево рішень і його основні складові.

Дерева рішень - один з основних і найбільш популярних методів класифікації. Дійсно, побудова дерев рішень дозволяє наочно продемонструвати іншим і розібратися самому в структурі даних, створити працюючу модель класифікації даних, якими б "великими" вони не були. На відміну від інших алгоритмів машинного навчання, дерева рішення можуть працювати як з числовими, так і з дискретними даними

Основний плюс використання дерев рішень - це простота інтерпретації навченої моделі. Застосувавши алгоритм до розглянутого завдання, ми отримали не лише дерево, здатне робити прогнози про поведінку нових користувачів, але і список питань, на які треба відповісти для знаходження рішення. Наприклад, видно, що люди, що приходять на цей сайт з сайту Slashdot, ніколи не стають платними передплатниками, тоді якщо знайшли його за допомогою Google і проглянули хоча б 20 сторінок, ймовірно, оформлять підписку на преміальне обслуговування. Це, у свою чергу, наводить на думку змінити рекламну стратегію, сфокусувавшись на сайтах, що дають найбільш високоякісний трафік.

Список використаної літератури:

1. Hovland, C. I. (1960). Computer simulation of thinking. American Psychologist, 15(11), 687-693.
2. Hunt, Earl B.; Janet Marin; Philip J. Stone (1966). Experiments in Induction. New York: Academic Press. ISBN 978-0-12-362350-8.
3. Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1):81-106.
4. Quinlan, J. Ross. C4.5: Programs for Machine learning. Morgan Kaufmann Publishers 1993.
5. Breiman, Leo, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.
6. Murthy, S. Automatic construction of decision trees from data: A Multi-disciplinary survey.1997.
7. Buntine, W. A theory of classification rules. 1992.
8. Machine Learning, Neural and Statistical Classification. Editors D. Mitchie et.al. 1994.