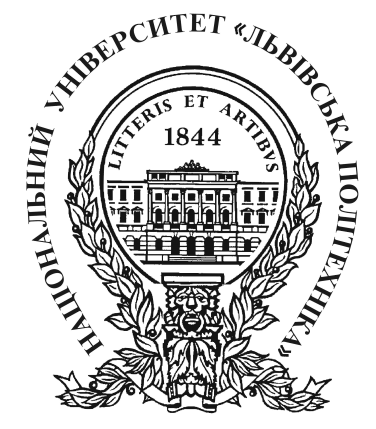
МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”



Лабораторна робота №5

з курсу «Інженерія даних та знань»

на тему: **«Індукція дерев рішень»**

Виконав:

ст. гр. ITIT-12

Галах А.Б.

Прийняла:

Рибчак З.Л.

Львів 2017

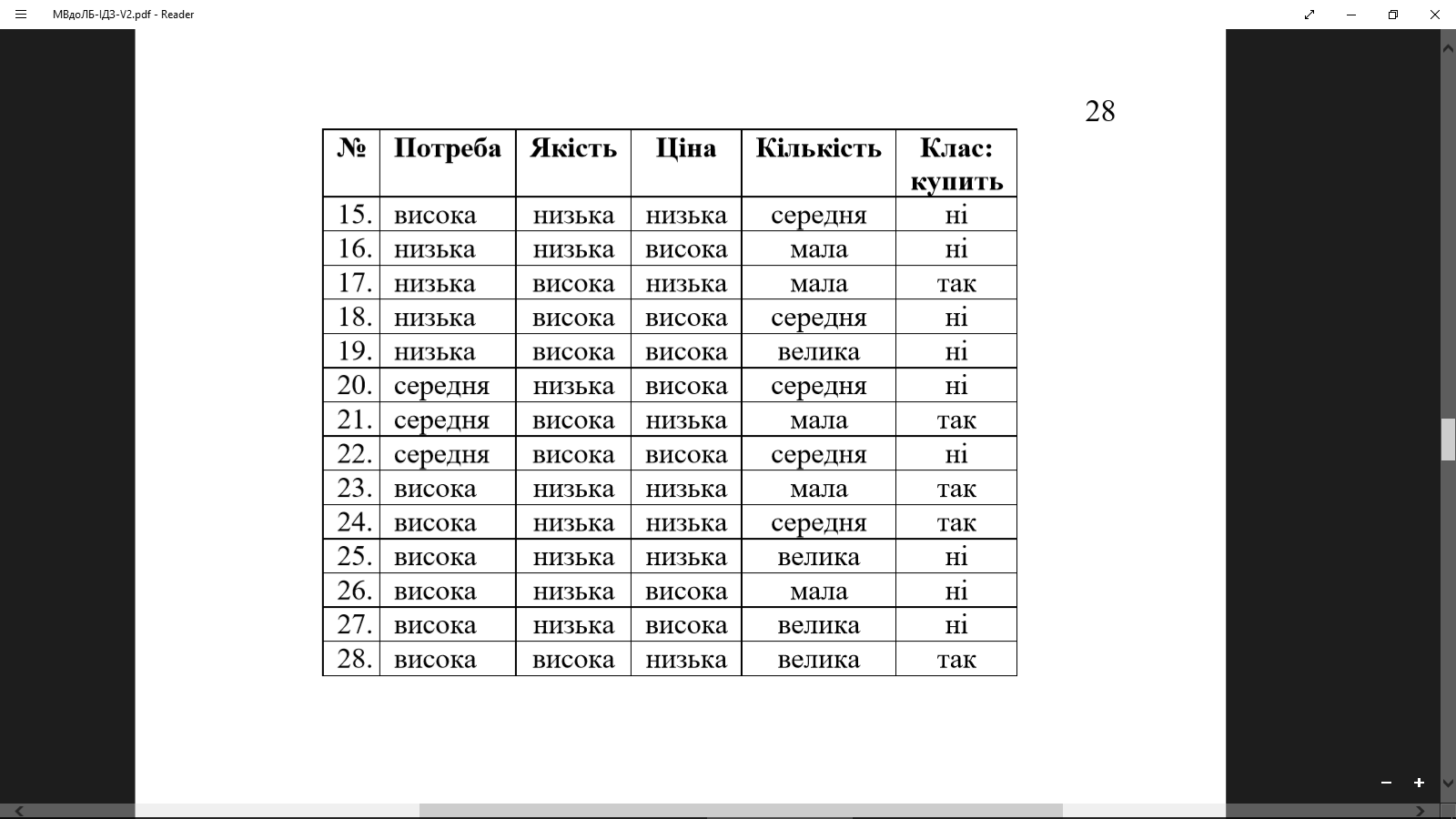
1. **Мета роботи**

Мета лабораторної роботи полягає у вивченні алгоритму побудови дерева рішень та його застосування для розв’язування задач аналізу даних.

1. **Індивідуальне завдання**

**Варіант 3**

Побудувати дерево рішень на основі таблиці. Ввести запит у вигляді набору обмежень на параметри «потреба, якість, ціна, кількість, клас» та знайти найкращий варіант, що відповідає запиту.



1. **Програмна реалізація**

Програма зчитує файл з даними формату .csv та індукує на його основі дерево рішень, яке вона відображає у консоль. після цього за заданим запитом програма видає відповідь, вказавши точність свого передбачення.

**from** \_\_future\_\_ **import** print\_function  
  
**def** get\_unique\_values(rows, col):**return** set([row[col] **for** row **in** rows])  
  
**def** get\_class\_counts(rows):counts = {}**for** row **in** rows:label = row[-1]  
 **if** label **not in** counts:  
 counts[label] = 0  
 counts[label] += 1  
 **return** counts  
  
**def** is\_numeric(value):  
 **return** isinstance(value, int) **or** isinstance(value, float)  
  
**def** partition(rows, question):true\_rows, false\_rows = [], []  
 **for** row **in** rows:  
 **if** question.match(row):  
 true\_rows.append(row)  
 **else**:  
 false\_rows.append(row)  
 **return** true\_rows, false\_rows  
  
**def** gini(rows):counts = get\_class\_counts(rows)  
 impurity = 1  
 **for** lbl **in** counts:  
 prob\_of\_lbl = counts[lbl] / float(len(rows))  
 impurity -= prob\_of\_lbl\*\*2  
 **return** impurity  
  
**def** get\_info\_gain(left, right, current\_uncertainty):p = float(len(left)) / (len(left) + len(right))  
 **return** current\_uncertainty - p \* gini(left) - (1 - p) \* gini(right)  
  
**def** find\_best\_split(rows):best\_gain = 0 best\_question = **None** current\_uncertainty = gini(rows)  
 n\_features = len(rows[0]) - 1 **for** col **in** range(n\_features): values = set([row[col] **for** row **in** rows]) **for** val **in** values: question = Question(col, val)true\_rows, false\_rows = partition(rows, question)**if** len(true\_rows) == 0 **or** len(false\_rows) == 0:  
 **continue**gain = get\_info\_gain(true\_rows, false\_rows, current\_uncertainty)**if** gain >= best\_gain:  
 best\_gain, best\_question = gain, question  
 **return** best\_gain, best\_question  
  
**class** Leaf:**def** \_\_init\_\_(self, rows):  
 self.predictions = get\_class\_counts(rows)  
  
  
**class** Decision\_Node:**def** \_\_init\_\_(self,  
 question,  
 true\_branch,  
 false\_branch):  
 self.question = question  
 self.true\_branch = true\_branch  
 self.false\_branch = false\_branch  
  
**def** build\_tree(rows):gain, question = find\_best\_split(rows)**if** gain == 0:  
 **return** Leaf(rows)true\_rows, false\_rows = partition(rows, question)true\_branch = build\_tree(true\_rows)false\_branch = build\_tree(false\_rows)**return** Decision\_Node(question, true\_branch, false\_branch)  
  
**def** print\_tree(node, spacing=**""**):**if** isinstance(node, Leaf):  
 print (spacing + **"Predict"**, node.predictions)  
 **return**print (spacing + str(node.question))print (spacing + **'--> True:'**)  
 print\_tree(node.true\_branch, spacing + **" "**)print (spacing + **'--> False:'**)  
 print\_tree(node.false\_branch, spacing + **" "**)  
  
**def** classify(row, node):**if** node.question.match(row):  
 **return** classify(row, node.true\_branch)  
 **else**:  
 **return** classify(row, node.false\_branch)  
  
**def** print\_leaf(counts):total = sum(counts.values()) \* 1.0  
 probs = {}  
 **for** lbl **in** counts.keys():  
 probs[lbl] = str(int(counts[lbl] / total \* 100)) + **"%"  
 return** probs  
  
header = [**"Потреба"**, **"Якість"**, **"Ціна"**, **"Кількість"**, **"Клас"**]  
  
**class** Question:**def** \_\_init\_\_(self, column, value):  
 self.column = column  
 self.value = value  
 **def** match(self, example):val = example[self.column]  
 **if** is\_numeric(val):  
 **return** val >= self.value  
 **else**:  
 **return** val == self.value  
 **def** \_\_repr\_\_(self):  
 condition = **"=="  
 if** is\_numeric(self.value):  
 condition = **">="  
 return "Is %s %s %s?"** % (  
 header[self.column], condition, str(self.value))

1. **Результат виконання програми**

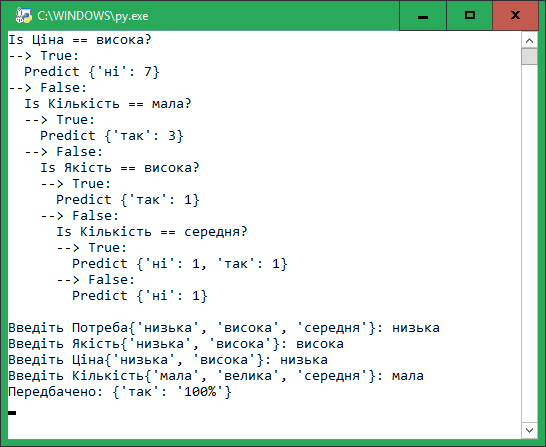


Рис.1 Програма побудувала дерево рішень та дала відповідь на запит

1. **Висновок**

У цій лабораторній роботі я вивчив алгоритм побудови дерева рішень та його застосування для розв’язування задач аналізу даних. Написав програму для реалізації поставленого завдання.