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

Кобак Р.З.

Прийняла:

Рибчак З.Л.

**Львів 2017**

**Мета роботи:**

Вивчення алгоритму побудови дерева рішень та його застосування для розв’язування задач аналізу даних.

**Індивідуальне завдання:**

**Варіант 3**

Побудувати дерево рішень на основі таблиці (Рис.1). Ввести запит у вигляді набору обмежень на параметри «погода, температура, вологість, вітео, клас» та знайти найкращий варіант, що відповідає запиту



Рис.1 Таблиця з вхідними даними

**Порядок виконання:**

Реалізовано додаток, що обробляє статистичні дані в форматі CSV (Comma Separated Value), та перетворює їх в дерево прийняття рішень.

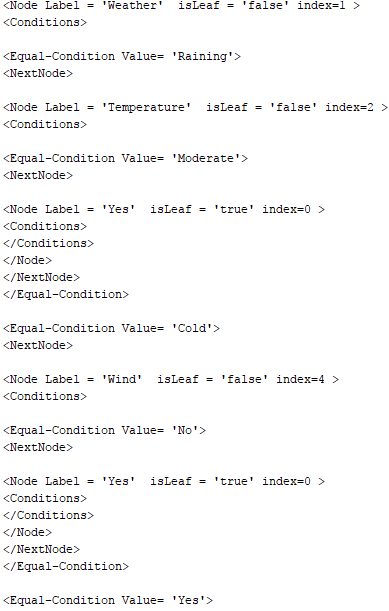
**Код програми:**

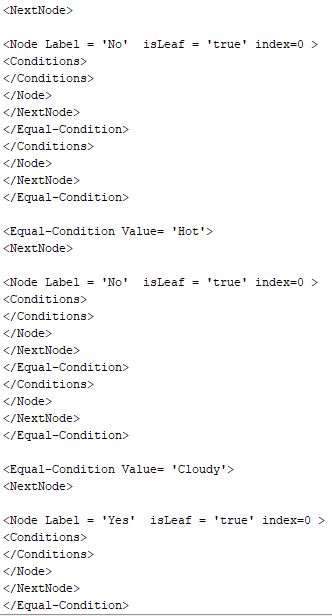
**public class** Test {  
   
 **public static void** main(String[] args) {  
 DecisionTree tree = **new** DecisionTree();  
 tree.train(**new** File(**"resources/inputData.psv"**));  
 System.***out***.println(tree.getRootNode());  
 System.***out***.println(tree.classify(**"Raining|Moderate|High|Yes"**));  
 }  
  
}

**public class** DecisionTree {  
  
 **private** BitSet **columns**;  
 **private** Set<String> **classes**;  
 **private int rowsCount**;  
 **private int columnsCount**;  
 **private** Node **rootNode**;  
 **private** File **psvFile**;  
  
 **public** DecisionTree() {  
 **this**.**columns** = **new** BitSet(**columnsCount**);  
 **classes** = **new** HashSet<String>();  
 **rootNode** = **new** Node();  
 }  
  
 **public** String classify(String row) {  
  
 String[] attrs = row.split(**"\\|"**);  
 **return** classify(attrs, **rootNode**);  
  
 }  
  
 **private** String classify(String[] attrs, Node node) {  
  
 **if** (node.isLeaf()) {  
 **return** node.getLabel();  
 }  
 String currentValue = attrs[node.getIndex() - 1];  
 **for** (Condition condition : node.getForks()) {  
 **if** (condition.test(currentValue)) {  
 **return** classify(attrs, condition.getNextNode());  
 }  
 }  
  
 **return "Cann't Find Class -- Please Learn Tree with more examples"**;  
  
 }  
  
 **public void** train(File psvFile) {  
 **try** {  
 **this**.**psvFile** = psvFile;  
 findClasses(psvFile);  
 BitSet rows = **new** BitSet(**rowsCount**);  
 **for** (**int** i = 0; i < **rowsCount**; i++) {  
 rows.set(i);  
 }  
 buildTree(**rootNode**, rows);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private void** buildTree(Node currentNode, BitSet rows) **throws** IOException {  
  
 AttributeInfo bestAttribute = findBestSplit(**psvFile**, rows);  
 **if** (bestAttribute == **null**) {  
 currentNode.setLeaf(**true**);  
 }  
 currentNode.setLabel(bestAttribute.getName());  
 currentNode.setIndex(bestAttribute.getIndex());  
 **columns**.set(bestAttribute.getIndex());  
  
 Map<String, ValueInfo> infoValues = bestAttribute.getValues();  
  
 **for** (Entry<String, ValueInfo> entry : infoValues.entrySet()) {  
 ValueInfo currentValue = entry.getValue();  
 String valueName = entry.getKey();  
 Map<String, Integer> classes = currentValue.getAttributeClasses();  
  
 **if** (currentValue.getEntropy() == 0.0) {  
 String classLabel = findClassLabel(classes);  
 Node leafNode = **new** Node();  
 leafNode.setLabel(classLabel);  
 leafNode.setLeaf(**true**);  
 EqualCondition equalCondition = **new** EqualCondition(leafNode,  
 valueName);  
 currentNode.addCondition(equalCondition);  
 } **else** {  
 Node newNode = **new** Node();  
 EqualCondition equalCondition = **new** EqualCondition(newNode,  
 valueName);  
 currentNode.addCondition(equalCondition);  
  
 buildTree(newNode, currentValue.getRows());  
  
 }  
  
 }  
  
 }  
  
 **private** String findClassLabel(Map<String, Integer> classes) {  
  
 **int** max = -1;  
 String classLabel = **""**;  
 Iterator<Entry<String, Integer>> classIterator = classes.entrySet()  
 .iterator();  
 **while** (classIterator.hasNext()) {  
 Entry<String, Integer> classEntry = (Entry<String, Integer>) classIterator  
 .next();  
 **if** (classEntry.getValue() > max) {  
 max = classEntry.getValue();  
 classLabel = classEntry.getKey();  
 }  
 }  
  
 **return** classLabel;  
  
 }  
  
 **private void** findClasses(File csvFile) **throws** IOException {  
  
 FileReader fileReader = **new** FileReader(csvFile);  
 BufferedReader reader = **new** BufferedReader(fileReader);  
 **int** counter = 0;  
 String line;  
 **while** ((line = reader.readLine()) != **null**) {  
 **if** (counter != 0) {  
 String[] cols = line.split(**"\\|"**);  
 **columnsCount** = cols.**length**;  
 **if** (cols.**length** > 2) {  
 String targetValue = cols[cols.**length** - 1];  
 **if** (!**classes**.contains(targetValue)) {  
 **classes**.add(cols[cols.**length** - 1]);  
 }  
 }  
 }  
 counter++;  
 }  
 **rowsCount** = counter;  
 reader.close();  
 fileReader.close();  
 }  
  
 **private** AttributeInfo findBestSplit(File csvFile, BitSet rows)  
 **throws** IOException {  
  
 AttributeInfo bestAttribute = **null**;  
 **double** bestEntropy = Double.***MAX\_VALUE***;  
 **for** (**int** h = 1; h < **columnsCount** - 1; h++) {  
  
 **if** (**columns**.get(h)) {  
 **continue**;  
 }  
  
 AttributeInfo data = singleAttributeInfo(csvFile, h, rows);  
 Map<String, ValueInfo> attributes = data.getValues();  
  
 **double** entropy = calculateSubTreeEntropy(attributes,  
 data.getRowCount());  
 **if** (entropy < bestEntropy) {  
 bestAttribute = data;  
 bestEntropy = entropy;  
 bestAttribute.setIndex(h);  
 }  
  
 }  
 **if** (bestAttribute == **null**) {  
 System.***out***.println(**"Input data isn`t enough for training"**);  
 System.*exit*(0);  
 }  
  
 **columns**.set(bestAttribute.getIndex());  
 **return** bestAttribute;  
  
 }  
  
 **private** AttributeInfo singleAttributeInfo(File file, **int** index, BitSet rows)  
 **throws** IOException {  
  
 AttributeInfo attributeInfo = **new** AttributeInfo();  
 attributeInfo.setIndex(index);  
 Map<String, ValueInfo> attributes = **new** HashMap<String, ValueInfo>();  
 FileReader fileReader = **new** FileReader(file);  
 BufferedReader breader = **new** BufferedReader(fileReader);  
 String line;  
 **int** counter = 0;  
 **int** rowsSize = 0;  
 **while** ((line = breader.readLine()) != **null**) {  
  
 **if** (counter == 0) {  
 String[] cols = line.split(**"\\|"**);  
 **for** (**int** i = 0; i < cols.**length**; i++) {  
 **if** (i == index) {  
 attributeInfo.setName(cols[i]);  
 }  
 }  
 } **else** {  
  
 **if** (!rows.get(counter)) {  
 counter++;  
 **continue**;  
 }  
  
 String[] cols = line.split(**"\\|"**);  
  
 **for** (**int** i = 0; i < cols.**length**; i++) {  
 **if** (i == index) {  
  
 String className = cols[cols.**length** - 1];  
 String value = cols[i];  
 **if** (!attributes.containsKey(value)) {  
  
 attributes.put(value, **new** ValueInfo(**classes**,  
 **new** BitSet(**rowsCount**)));  
 }  
  
 ValueInfo info = attributes.get(value);  
 info.increaseClass(className);  
 info.setRowAt(counter);  
 info.increaseRowCount();  
 rowsSize++;  
  
 }  
 }  
  
 }  
 counter++;  
 }  
 breader.close();  
 fileReader.close();  
 attributeInfo.setRowCount(rowsSize);  
 attributeInfo.setValues(attributes);  
 **return** attributeInfo;  
 }  
  
 **private double** calculateSubTreeEntropy(Map<String, ValueInfo> subTree, **int** count) {  
 **double** totalEntropy = 0;  
  
 Iterator<Entry<String, ValueInfo>> iterator = subTree.entrySet().iterator();  
 **while** (iterator.hasNext()) {  
 Entry<String, ValueInfo> entry = (Entry<String, ValueInfo>) iterator  
 .next();  
 ValueInfo info = entry.getValue();  
  
 **double** entropy = calculateEntropy(**new** ArrayList<Integer>(info  
 .getAttributeClasses().values()), info.getRowsCount());  
 info.setEntropy(entropy);  
 totalEntropy += ((**double**) info.getRowsCount() / count) \* entropy;  
  
 }  
  
 **return** totalEntropy;  
 }  
  
 **public double** calculateEntropy(List<Integer> classRecords, **int** total) {  
 **double** entropy = 0;  
 **for** (**int** i = 0; i < classRecords.size(); i++) {  
 **double** probability = (**double**) classRecords.get(i) / total;  
 entropy -= probability \* *logb*(probability, 2);  
 }  
 **return** entropy;  
 }  
  
 **public static double** logb(**double** a, **double** b) {  
 **if** (a == 0)  
 **return** 0;  
 **return** Math.*log*(a) / Math.*log*(b);  
 }  
  
 **public** Node getRootNode() {  
 **return rootNode**;  
 }  
  
}

**Вхідні дані:**Index|Weather|Temperature|Humidity|Wind|Class  
1|Raining|Hot|High|No|No  
2|Sunny|Hot|High|Yes|No  
3|Cloudy|Hot|High|No|Yes  
4|Raining|Moderate|High|Yes|Yes  
5|Raining|Cold|Normal|No|Yes  
6|Raining|Cold|Normal|Yes|No  
7|Cloudy|Cold|Normal|Yes|Yes  
8|Sunny|Moderate|High|No|No  
9|Sunny|Cold|Normal|No|Yes  
10|Raining|Moderate|Normal|No|Yes  
11|Sunny|Moderate|Normal|Yes|Yes  
12|Cloudy|Moderate|High|Yes|Yes  
13|Cloudy|Hot|Normal|No|Yes  
14|Raining|Moderate|High|Yes|Yes

**Результат виконання програми:**





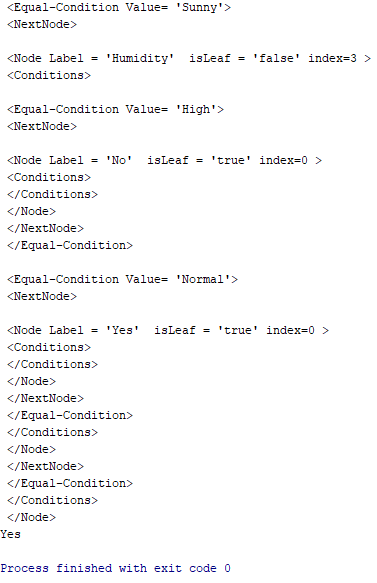


Рис.2 Результат виконання програми

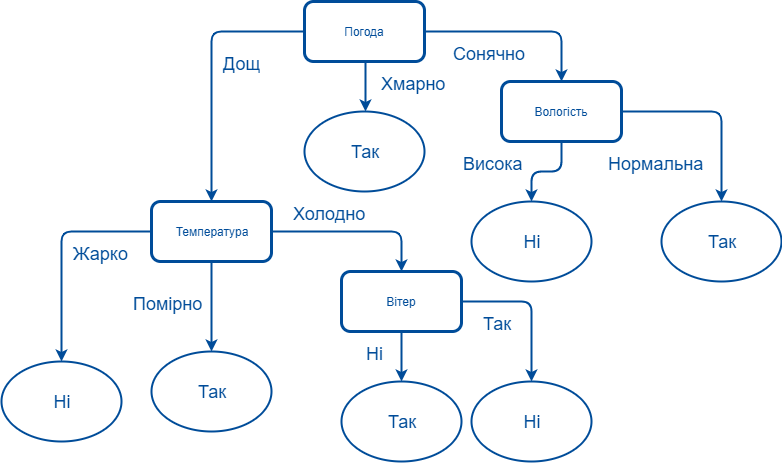


Рис.3 Згенероване дерево прийняття рішень

Результат виконання програми представлений на Рис 2. На основі результату виконання програми було згенероване дерево прийняття рішення Рис 3. Для тесту було використано наступне значення:

Погода: Дощ  
 Температура: Помірна

Вітер: Так

Вологість: Висока

Результат виконання: матч відбудеться.

**Висновок:**

При виконанні даної лабораторної роботи було реалізовано додаток для генерації дерев прийняття рішень на основі статистичних даних. На основі результатів виконання програми було згенеровано дерево прийнятті рішення та визначено результат за тестовими параметрами. Повний код програми знаходиться на репозиторії за даною URL- адресою: https://github.com/amukysh/EngineeringOfDataAndKnowledge.git/