Decision Tree

0. Introduction

- Lecture: 데이터사이언스(ITE4005)
- Student: 한양대학교 컴퓨터소프트웨어학부 2017030191 이현지
- Programming Assignment #2: build a decision tree, and then classify the test set using it

1. Main Concepts

1. Summary

1. Main Idea

   > 💡 train file로부터 label과 attribute 정보를 받아 decision tree를 만든다. decision tree를 만드는 것은 가장 homogenous한 test attribute를 recursive하게 골라 트리를 만드는 과정을 거친다.

2. Entropy: 불순도를 측정하는 지표
   - 정보가 섞여있는 상태일수록 큰 값을 가지고, 분류하기 어려움
     - homogenous할수록 entropy가 낮음
   - Entropy 계산 공식

     $Entorpy(S) = \sum pi * I(Xi)$
3. Information Gain: test attribute를 골라내는 방식
   - 해당 attribute을 test attribute으로 지정했을 때 entropy가 가장 작음
   - Information Gain 계산 공식

     $Gain(A) = Info(D) - InfoA(D)$

     → Info(D)는 다 같기 때문에 InfoA(D)가 가장 작은 것 찾기

2. Main.class

1. Get data from files

```
String trainFile = args[0], testFile = args[1], outputFile = args[2];

List<String> labels = getLabels(trainFile);
String classLabel = getClassLabel(trainFile);
List<Pair<LinkedHashMap<String, String>, String>> samples = getSamples(trainFile, labels);
List<LinkedHashMap<String, String>> tests = getTests(testFile, labels);
List<String> resultList = getResultList(samples);
```

- trainFile/testFiile/outputFile: 소스 코드 실행 시 인자로 받는 train/test/output 파일의 이름
  - 경로를 명시하지 않으면 jar파일과 같은 경로에 존재
- labels: train 파일에서 각 attribute의 이름들을 담은 리스트 (classLabel은 없음)
- classLabel: train 파일의 labels에서 마지막에 존재
- samples: train 파일에서 받은 샘플들을 담은 리스트
  - 각 값은 classLabel을 제외한 attribute 값을 LinkedHashMap으로, 그리고 classLabel을 함께 Pair로 묶은 값으로 표현됨
- tests: test 파일에서 받은 샘플들을 담은 리스트

- resultList: samples에서 classLabel에 대한 result를 중복없이 담은 리스트

2. Build decision tree

```
Node tree = makeDecisionTree(samples, labels, new Node());
```

- tree: 1번에서 얻은 samples와 labels를 parameter로 하여 decision tree를 만듬
  - Node 클래스 객체로 반환하고 해당 tree 노드는 루트값을 나타냄

3. Get Answers with the above decision tree

```
List<LinkedHashMap<String, String>> answers = getAnswers(tests, tree, resultList, classLabel);
makeOutputFile(answers, labels, classLabel, outputFile);
```

- answers: 1번과 2번에서 얻은 tests와 tree, resultList, classLabel을 parameter로 하여 tests를 classify하고 그 값을 LinkedHashMap들의 리스트로 표현
- makeOutputFile: answers를 정해진 형식에 맞게 {outputFile}이름으로 생성

3. Node.class

- Tree 구조를 나타내기 위한 클래스
- isAttrNode로 attribute을 나타내는지, vertex(attribute의 값)을 나타내는지 구분
- attrNode이면 List<Node> children을 가지고, verNode이면 Node nextAttr을 가짐

```
package cse.ds;

import java.util.ArrayList;
import java.util.List;

public class Node {
    public boolean isAttrNode = true;

    private String attribute;
    private List<Node> children;
    private Node nextAttr;

    public Node(){
        this.children = new ArrayList<>();
    }

    public static Node attrNode(String attribute){
        Node newNode = new Node();
        newNode.isAttrNode = true;
        newNode.attribute = attribute;
        return newNode;
    }

    public static Node verNode(String value){
        Node newNode = new Node();
        newNode.isAttrNode = false;
        newNode.attribute = value;
        return newNode;
    }

    public void setNextAttr(Node attrNode){ this.nextAttr = attrNode; }

    public Node getNextAttr(){ return this.nextAttr; }

    public List<Node> getChildren(){ return this.children; }

    public void setAttribute(String attribute){ this.attribute = attribute; }

    public String getAttribute(){ return this.attribute; }

    public String toString(){
        ...
    }
}
```

4. Pair.class

- java에 없는 Pair 자료구조를 표현하는 클래스

```
package cse.ds;

public class Pair<T,S> {
    private T first;
    private S second;

    public Pair(){ }

    public Pair(T first, S second) {
        this.first = first;
        this.second = second;
    }

    public T getFirst(){ return this.first; }

    public S getSecond(){ return this.second; }

    public void setSecond(S second){ this.second = second; }

    public String toString(){ return "(" + this.first + ", " + this.second + ")"; }
}
```

2. Details

1. getTotalSize()

1. Specification
    - Paramter: List<List<Pair<LinkedHashMap<String, String>, String>>> splitSamples
    - Return type: Integer
2. Description

    split된 samples를 parameter로 받아 그 크기를 계산하여 반환한다.

3. Code

```
public static Integer getTotalSize(List<List<Pair<LinkedHashMap<String, String>, String>>> splitSamples){
    Integer totalSize = 0;
    for(List<Pair<LinkedHashMap<String, String>, String>> subSamples : splitSamples){
        totalSize += subSamples.size();
    }
    return totalSize;
}
```

2. getP()

1. Specification
    - Parameter: List<String> classes
    - Return type: List<Double>
2. Description
3. Code

```
public static List<Double> getP(List<String> subSampleClassLabels){
    LinkedHashMap<String, Integer> subSampleClasses = new LinkedHashMap<>();
    for(String subSampleClassLabel : subSampleClassLabels){
        Integer classesCounterNum = subSampleClasses.getOrDefault(subSampleClassLabel, 0);
        subSampleClasses.put(subSampleClassLabel, ++classesCounterNum);
    }

    Double subSampleClassSize = subSampleClassLabels.size() * 1.0;
    List<Double> P = new ArrayList<>();
    for(Entry<String, Integer> subSampleClassesEntry : subSampleClasses.entrySet()){
        Integer subSampleClassValue = subSampleClassesEntry.getValue();
        P.add(subSampleClassValue / subSampleClassSize);
```

```
    }
    return P;
}
```

3. getEntropy()

1. Specification
   - Parameter: List<Pair<LinkedHashMap<String, String>, String>> subSamples
   - Return type: Double
2. Description

> 💡 Entropy를 구하는 공식(1-1번 참고)을 사용하여 parameter로 들어온 subSamples의 entropy를 구한다.

3. Code

```
public static Double getEntropy(List<Pair<LinkedHashMap<String, String>, String>> subSamples){
    List<String> subSampleClassLabels = new ArrayList<>();
    for(Pair<LinkedHashMap<String, String>, String> subSample : subSamples){
        subSampleClassLabels.add(subSample.getSecond());
    }
    List<Double> P = getP(subSampleClassLabels);

    Double entropy = 0.0;
    for(Double P0 : P){
        if(P0 > 0){
            entropy += (-1) * P0 * (Math.log(P0) / Math.log(2));
        }
    }
    return entropy;
}
```

4. getSplitSamplesEntropy()

1. Specification
   - Parameter: List<List<Pair<LinkedHashMap<String, String>, String>>> splitSamples
   - Return type: Double
2. Description

> 💡 getEntropy(2-3번 참고) 함수를 사용하여 parameter로 들어온 splitSamples의 entropy의 합을 구한다.

3. Code

```
public static Double getSplitSamplesEntropy(List<List<Pair<LinkedHashMap<String, String>, String>>> splitSamples){
    Integer totalSize = getTotalSize(splitSamples);
    Double entropy = 0.0;
    for(List<Pair<LinkedHashMap<String, String>, String>> subSamples : splitSamples){
        entropy += getEntropy(subSamples) * subSamples.size() / totalSize;
    }
    return entropy;
}
```

5. getSplitSamples()

1. Specification
   - Parameter: List<Pair<LinkedHashMap<String, String>, String>> samples, String attribute
   - Return type: LinkedHashMap<String, List<Pair<LinkedHashMap<String, String>, String>>>
2. Description

> 💡 parameter로 받은 label을 기준으로 samples를 result로 구별하여 파티셔닝한다.

3. Code

```
public static LinkedHashMap<String, List<Pair<LinkedHashMap<String, String>, String>>> getSplitSamples(List<Pair<LinkedHashMap<Str
    LinkedHashMap<String, List<Pair<LinkedHashMap<String, String>, String>>> splitSamples = new LinkedHashMap<>();
    for (Pair<LinkedHashMap<String, String>, String> sample : samples) {
        String key = sample.getFirst().get(attribute);
        if(splitSamples.containsKey(key)){
            splitSamples.get(key).add(sample);
        }
        else{
            List<Pair<LinkedHashMap<String, String>, String>> temp = new ArrayList<>();
            temp.add(sample);
            splitSamples.put(key, temp);
        }
    }
    return splitSamples;
}
```

6. informationGain()

1. Specification
   - Parameter: List<Pair<LinkedHashMap<String, String>, String>> samples, String attribute
   - Return type: Double
2. Description

> 💡 parameter로 받은 attribute을 기준으로 samples를 split하고(2-5번 참고), split된 samples의 entropy를 구한다.

3. Code

```
public static Double informationGain(List<Pair<LinkedHashMap<String, String>, String>> samples, String attribute){
    LinkedHashMap<String, List<Pair<LinkedHashMap<String, String>, String>>> splitSamples = getSplitSamples(samples, attribute);
    List<List<Pair<LinkedHashMap<String, String>, String>>> subSamples = new ArrayList<>();
    for(Entry<String, List<Pair<LinkedHashMap<String, String>, String>>> splitSamplesEntry : splitSamples.entrySet()){
        subSamples.add(splitSamplesEntry.getValue());
    }
    return getSplitSamplesEntropy(subSamples);
}
```

7. sortMapByValue()

1. Specification
   - Parameter: LinkedHashMap<String, Integer> LinkedHashMap
   - Return type: List<Pair<String, Integer>>
2. Description

> 💡 parameter로 받은 map을 value를 기준으로 정렬하여 반환한다.

3. Code

```
public static List<Pair<String, Integer>> sortMapByValue(LinkedHashMap<String, Integer> map) {
    List<Entry<String, Integer>> entries = new LinkedList<>(map.entrySet());
    entries.sort(Entry.comparingByValue());
    Collections.reverse(entries);

    List<Pair<String, Integer>> valueList = new ArrayList<>();
    for (Entry<String, Integer> entry : entries) {
        valueList.add(new Pair<>(entry.getKey(), entry.getValue()));
    }
```

```
    return valueList;
}
```

8. buildDecisionTree()

1. Specification
   - Parameter: List<Pair<LinkedHashMap<String, String>, String>> samples, List<String> labels, Node node
   - Return type: Node
2. Description

> 💡 parameter로 받은 samples와 labels에 대하여 entropy가 가장 작은 attribute를 test attribute로 선정하여 분기하는 것을 recursive하게 작동한다.

3. Code

```
public static Node buildDecisionTree(List<Pair<LinkedHashMap<String, String>, String>> samples, List<String> labels, Node node) {
    if (labels.isEmpty()) {
        return node;
    }

    LinkedHashMap<String, Integer> classResult = new LinkedHashMap<>();
    for (Pair<LinkedHashMap<String, String>, String> sample : samples) {
        Integer classResultNum = classResult.getOrDefault(sample.getSecond(), 0);
        classResult.put(sample.getSecond(), ++classResultNum);
    }

    List<Pair<String, Integer>> classReulstList = sortMapByValue(classResult);
    String moreResult = classReulstList.get(0).getFirst();

    Double minSubSampleEntropy = 987654321.0;
    String testAttribute = "";
    for (String label : labels) {
        Double subSampleEntropy = informationGain(samples, label);
        if (subSampleEntropy < minSubSampleEntropy) {
            minSubSampleEntropy = subSampleEntropy;
            testAttribute = label;
        }
    }
    LinkedHashMap<String, List<Pair<LinkedHashMap<String, String>, String>>> splitSamples = getSplitSamples(samples, testAttribute
    List<String> labels2 = new ArrayList<>(labels);
    labels2.remove(testAttribute);

    for (Entry<String, List<Pair<LinkedHashMap<String, String>, String>>> splitSample : splitSamples.entrySet()) {
        String attribute = splitSample.getKey();
        List<Pair<LinkedHashMap<String, String>, String>> subSamples = splitSample.getValue();
        node.setAttribute(testAttribute);
        Node verNode = verNode(attribute);
        node.getChildren().add(verNode);
        Node newNode = attrNode(moreResult);
        verNode.setNextAttr(newNode);
        buildDecisionTree(subSamples, labels2, newNode);
    }
    node.getChildren().add(verNode(moreResult));
    return node;
}
```

9. classify()

1. Specification
   - Parameter: LinkedHashMap<String, String> sample, Node node, List<String> resultList
   - Return type: Node
2. Description

> 💡 buildDecisionTree 함수(2-9번 참고)로 만든 트리(node)로 test를 classify한다.

3. Code

```
public static Node classify(LinkedHashMap<String, String> test, Node node, List<String> resultList){
    if(node.isAttrNode){
        String attr = node.getAttribute();
        if(resultList.contains(attr)){
            return node;
        } else {
            boolean flag = false;
            for (int i = 0 ; i < node.getChildren().size() ; i++){
                Node child = node.getChildren().get(i);
                if(child.getAttribute().equals(test.get(attr))){
                    flag = true;
                    return classify(test, child.getNextAttr(), resultList);
                }
            }
            if(!flag){
                return node.getChildren().get(node.getChildren().size()-1);
            }
        }
    }
    return node;
}
```

3. How to Execute

1. Environment

- OS: Mac OS Catalina 10.15.7
- Runtime: JDK 15.0.1
- IDE: IntelliJ Ultimate 2020.03

2. Structure

execution - decisionTree.jar

ㄴ dt_train.txt

ㄴ dt_test.txt

ㄴ dt_train1.txt

ㄴ dt_test1.txt

project - src - cse.ds - Main.class

ㄴ DecisionTree.class

ㄴ Node.class

ㄴ Pair.class

ㄴ out - artifacts - decisionTree.jar

3. Usage

- execution 폴더의 decisionTree.jar 파일로 실행 가능

```
hyunjigonji@ihyeonjiui-MacBookPro execution % ls
decisionTree.jar        dt_test.txt             dt_test1.txt            dt_train.txt            dt_train1.txt
hyunjigonji@ihyeonjiui-MacBookPro execution % java -jar decisionTree.jar dt_train.txt dt_test.txt dt_result.txt
hyunjigonji@ihyeonjiui-MacBookPro execution % ls
decisionTree.jar        dt_test.txt             dt_train.txt
dt_result.txt           dt_test1.txt            dt_train1.txt
hyunjigonji@ihyeonjiui-MacBookPro execution % mono ../../../2.\ Decision\ Tree/test/dt_test.exe ../../../2.\ Decision\ Tree/test/dt_
answer.txt dt_result.txt
5 / 5
```

```
% java -jar decisionTree.jar {train_file} {test_file} {result_file}

ex) java -jar decisionTree.jar dt_train.txt dt_test.txt dt_result.txt
```