

## Decision Tree implement assignment

2013011800 구장희

# 순서

1. data structure - tree class , node class
2. summary of algorithm for crucial method
3. summary of rest algorithm
4. instruction for compile

# data structure

[tree class]

```
14 /**
15  * Tree class
16  *
17  * List<Map<List<String>, String>> data      : initial list of tuples by reading training data
18  * List<Map<List<String>, String>> tdata    : initial list of tuples by reading test data
19  * List<Map<List<String>, String>> rdata    : list of tuples for result data
20  *
21  * Map<String, List<String>> attributes    : pair of each (attribute , its possible value)
22  *
23  * List<String> label                      : list of class-label
24  * List<String> attribute                  : list of attribute name
25  *
26  * String trainingset, testset, resultset  : path to each files
27  * String classname                       : class name
28  * int numattr                             : number of attributes
29  * Node root                              : root node of decision tree
30  *
31  */
32 public class Tree {
33     List<Map<List<String>, String>> data, tdata, rdata;
34     Map<String, List<String>> attributes;
35     List<String> label;
36     List<String> attribute;
37     String trainingset, testset, resultset, classname = null;
38     int numattr = 0;
39     Node root;
40
41 }
```

[node class]

```
603- /**
604-  * Node class
605-  *
606-  * String label      : node's class-label , null at default
607-  * String decision   : node's split attribute , null at default
608-  * String value      : value of parent node's decision attribute
609-  * Node child        : its child node
610-  * List<Map<List<String>, String>> list : list of tuples belong to node
611-  *
612-  */
613- public class Node {
614-     String label, decision, value;
615-     List<Node> child;
616-     List<Map<List<String>, String>> list;
617-
618-     /* constructor */
619-     public Node(List<Map<List<String>, String>> _list) {
620-         this.list = _list;
621-         this.child = new ArrayList<Node>();
622-     }
623-
624-     /* add child node */
625-     void addBranch(Node _child) {
626-         this.child.add(_child);
627-     }
628- }
629- }
```

## # summary of algorithm for crucial method

[main method]

```
579 /**
580  * main method
581  *
582  * store all parameter as path to files
583  * and call work method after create decision-tree instance
584  *
585  * @param    path to training-dataset file
586  * @param    path to test-dataset file
587  * @param    path to result-dataset file
588  * @return   void
589  *
590  */
591 public static void main(String[] args) {
592     String trainingset, testset, result;
593     if (args.length != 3) {
594         System.out.println("args error!!!");
595     } else {
596         trainingset = args[0];
597         testset = args[1];
598         result = args[2];
599         new Tree().work(trainingset, testset, result);
600     }
601 }
602 }
```

### [work() in tree class]

- read training data
- construct decision tree
- read test data
- tree test from test data
- write result data

```
137 /**
138  * main work in DecisionTree algorithm
139  *
140  * read training-data and store all of them(all data-label pairs)
141  * construct all tree recursively and store root node
142  * read test-data and and store all of them(all data)
143  * test decision tree by test-data
144  * and write result to path of result-data
145  *
146  * @param    path to training-dataset.txt
147  * @param    path to test-dataset.txt
148  * @param    path to result-dataset.txt
149  * @return   void
150  *
151  */
152 void work(String _trainingset, String _testset, String _result) {
153     trainingset = _trainingset;
154     testset = _testset;
155     resultset = _result;
156     readTrainingData(trainingset);
157     root = id3(data, attributes, attribute);
158     readTestData(testset);
159     TreeTest();
160     writeResult(resultset);
161 }
162
```

## [id3() in tree class]

```
295  /**
296   * crucial method for recursively constructing decision-tree(specially each node)
297   *
298   * create new node from list of tuples
299   *
300   * if # of attribute is zero, then return node after majority-vote and store its label.
301   * if list of tuples are uniformly homogeneous, then return node after store its label.
302   *
303   * choose best attribute for split from (information gain or gain ratio or gini index).
304   * it is done simply by removing comment!!!
305   *
306   * after choosing best attribute, store it as decision attribute for node.
307   *
308   * for each value of best attribute,
309   * create sub list of tuples which has same value at best attribute
310   * create set pair of (attribute-its possible value) by removing best attribute at all tuples in sub list
311   * create list of attribute by removing best attribute
312   *
313   * finally create child node by calling same method recursively
314   * and store child node's value same as each value of best attribute
315   * and make pair parent-child node relationship
316   *
317   * @param    list of tuples
318   * @param    set pair of (attribute-its possible value)
319   * @param    list of attribute
320   *
321   * @return    node which is constructed
322   */
323
324  Node id3(List<Map<List<String>, String>> tuples, Map<String, List<String>> _attributes, List<String> _attribute) {
325      List<Map<List<String>, String>> _tuples = new ArrayList<Map<List<String>, String>>(tuples);
326      Node node = new Node(_tuples);
327
328      if (_attribute.size() == 0) {
329          node.label = majorityVote(_tuples);
330          return node;
331      }
332
333      Map<String, Integer> dictionary = summarizeExamples(_tuples);
334      for (String key : dictionary.keySet()) {
335          if (dictionary.get(key) == _tuples.size()) {
336              node.label = key;
337              return node;
338          }
339      }
340
341      //String bestAttr = getBestAttr(getInfoGain(_tuples, _attributes, _attribute));
342      //String bestAttr = getBestAttr(getGainRatio(_tuples, _attributes, _attribute));
343      String bestAttr = getBestAttr(getGiniIndex(_tuples, _attributes, _attribute));
344
345
346      int idx = _attribute.indexOf(bestAttr);
347      node.decision = bestAttr;
348
349      for (String value : _attributes.get(bestAttr)) {
350          int size = 0;
351
352          for (Map<List<String>, String> tuple : _tuples) {
353              for (List<String> key : tuple.keySet()) {
354                  size = Math.max(size, key.size());
355              }
356          }
357          if (size == 0) {
358              node.label = majorityVote(_tuples);
359              return node;
360          }
361      }
```

```

362 List<Map<List<String>, String>> subexamples = new ArrayList<>();
363 for (Map<List<String>, String> tuple : _tuples) {
364     for (List<String> line : tuple.keySet()) {
365         if (line.size() > idx) {
366             if (line.size() == size && line.get(idx).equalsIgnoreCase(value)) {
367                 subexamples.add(tuple);
368             }
369         }
370     }
371 }
372 if (subexamples.size() != 0) {
373     Map<String, List<String>> subattributes = new HashMap<String, List<String>>(_attributes);
374     subattributes.remove(bestattr);
375
376     List<String> subattribute = new ArrayList<String>(_attribute);
377     subattribute.remove(idx);
378
379     for (Map<List<String>, String> tuple : subexamples) {
380         for (List<String> line : tuple.keySet()) {
381             line.remove(idx);
382         }
383     }
384     Node child = id3(subexamples, subattributes, subattribute);
385     child.value = value;
386     node.addBranch(child);
387 }
388 }
389 return node;
390 }
391

```

## # summary of rest algorithm

[readTrainingData & readTestData]

```
42  /**
43   * read training-dataset file
44   * store all tuples(all data-label pair)
45   * store classes & all possible class-label
46   * store all attributes & all possible value of each attribute
47   *
48   * @param    path to training-dataset.txt
49   * @return    void
50   * @exception FileNotFoundException e
51   * @exception IOException e
52   */
53  void readTrainingData(String _trainingset) {}
54
55  /**
56   * read test-dataset file
57   * store all tuples(all data)
58   *
59   * @param    path to test-dataset.txt
60   * @return    void
61   * @exception FileNotFoundException e
62   * @exception IOException e
63   */
64  void readTestData(String _testset) {}
65
66  /**
```

## [writeResult & TreeTest]

```
163  /**
164   * write result to path of result-data
165   *
166   * add induced class-label at each tuples of test-data
167   *
168   * @param    path to result-dataset.txt
169   * @return    void
170   * @exception FileNotFoundException e
171   */
172  void writeResult(String resultset) {}
173
174  /**
175   * construct result-data by adding class-label at each tuples
176   * in test-data.
177   * each class-label is induced by calling findLabel method
178   *
179   * @param    void
180   * @return    void
181   */
182  void TreeTest() {
183      rdata = new ArrayList<>();
184      for (Map<List<String>, String> line : tdata) {
185          for (List<String> tuple : line.keySet()) {
186              String tlabel = findLabel(tuple, root);
187              Map<List<String>, String> temp = new HashMap<>();
188              temp.put(tuple, tlabel);
189              rdata.add(temp);
190          }
191      }
192  }
```



## [findLabel]

```
214  /**
215   * inducing class-label at each tuples
216   *
217   * if the node has no label
218   *
219   * find childnode which has value as its parent node's decision attribute.
220   * and call same method recursively simply change node to childnode.
221   * if there is no childnode which has same value then, do majority vote.
222   *
223   * @param    a tuple which has no class label
224   * @param    a node at decision tree
225   * @return   induced class label
226   *
227   */
228  String findLabel(List<String> tuple, Node node) {
229      boolean flag = false;
230      if (node.label == null) {
231          for (Node childnode : node.child) {
232              if (childnode.value.equalsIgnoreCase(tuple.get(attribute.indexOf(node.decision)))) {
233                  flag = true;
234                  return findLabel(tuple, childnode);
235              }
236          }
237          if (!flag) {
238              return majorityVote(node.list);
239          }
240      }
241      return node.label;
242  }
243  }
```

## [summarizeExamples]

```
244  /**
245   * construct pair of (class-label , its count)
246   * by searching all tuples and count numbers of class-label
247   * for checking whether tuples is homogeneous or not
248   *
249   * @param    list of tuples
250   * @return   pair of (class-label , count)
251   *
252   */
253  Map<String, Integer> summarizeExamples(List<Map<List<String>, String>> examples) {
254      List<Map<List<String>, String>> temp = new ArrayList<Map<List<String>, String>>(examples);
255      Map<String, Integer> ret = new HashMap<>();
256      for (String key : label) {
257          ret.put(key, 0);
258      }
259      for (int i = 0; i < temp.size(); i++) {
260          String fkey = (String) temp.get(i).values().toArray()[0];
261          Integer count = ret.get(fkey);
262          ret.put(fkey, (count + 1));
263      }
264      return ret;
265  }
266
267
```

## [majority-vote]

```
268  /**
269   * search all tuples and count numbers of class-label
270   * and return major class-label
271   *
272   * @param    List of tuples
273   * @return    class-label which is major in the tuples
274   *
275   */
276  String majorityVote(List<Map<List<String>, String>> _tuples) {
277      List<Map<List<String>, String>> tmp = new ArrayList<Map<List<String>, String>>(_tuples);
278
279      String ret = null;
280      int[] temp = new int[label.size()];
281      for (int i = 0; i < tmp.size(); i++) {
282          String fkey = (String) tmp.get(i).values().toArray()[0];
283          temp[label.indexOf(fkey)]++;
284      }
285      int max = -1;
286      for (int i = 0; i < label.size(); i++) {
287          if (temp[i] > max) {
288              max = temp[i];
289              ret = label.get(i);
290          }
291      }
292      return ret;
293  }
294  }
```

## [getBestAttr]

```
392  /**
393   *
394   * @param    pair of (attribute - its gain for any metric)
395   * @return    best attribute for split node
396   *
397   */
398  String getBestAttr(Map<String, Double> infogain) {
399      String ret = "";
400      double min = 100.0;
401      for (String key : infogain.keySet()) {
402          if (infogain.get(key) < min) {
403              min = infogain.get(key);
404              ret = key;
405          }
406      }
407      return ret;
408  }
409  }
```

## [getGiniIndex]

```
410  /**
411   * calculating gini-index
412   *
413   * @param    list of tuples
414   * @param    pair of (attribute and its possible value)
415   * @param    list of attributes
416   * @return   pair of (attribute - its gain) for gini-index
417   *
418   */
419  Map<String, Double> getGiniIndex(List<Map<List<String>, String>> _tuples, Map<String, List<String>> _attributes,
420    List<String> _attribute) {
421    Map<String, Double> gainstore = new HashMap<>();
422    int arr[] = new int[_attribute.size()];
423    for (int i = 0; i < _tuples.size(); i++) {
424      String fkey = (String) _tuples.get(i).values().toArray()[0];
425      arr[_attribute.indexOf(fkey)]++;
426    }
427    for (String attr : _attributes.keySet()) {
428      int idx = _attribute.indexOf(attr);
429      double global = _tuples.size();
430      double gain = 0;
431      double info_a = 0;
432      for (String value : _attributes.get(attr)) {
433        double local = 0;
434        int[] Arr = new int[_attribute.size()];
435        for (int i = 0; i < _tuples.size(); i++) {
436          for (List<String> key : _tuples.get(i).keySet()) {
437            if (key.get(idx).equalsIgnoreCase(value)) {
438              local++;
439              String fkey = (String) _tuples.get(i).values().toArray()[0];
440              Arr[_attribute.indexOf(fkey)]++;
441            }
442          }
443        }
444        info_a += local / global * gini(Arr);
445      }
446      gain = gini(arr) - info_a;
447      gain = Math.round(gain * 1000) / 1000.0;
448      gainstore.put(attr, gain);
449    }
450    return gainstore;
451  }
```

## [gini]

```
453  /*  
454   * calculating gini-index value at each integer array  
455   *  
456   * @param    integer array  
457   * @return    its value of gini-index  
458   *  
459   */  
460  double gini(int[] arr) {  
461      double ret = 1;  
462      double sum = 0;  
463      for (int i = 0; i < arr.length; i++) {  
464          sum += arr[i];  
465      }  
466      for (int i = 0; i < arr.length; i++) {  
467          double p = arr[i] / sum;  
468          ret -= Math.pow(p, 2);  
469      }  
470      return Math.round(ret * 1000) / 1000.0;  
471  }  
472  
473
```

## [getGainRatio]

```
474  /*  
475   * calculating gain-ratio  
476   *  
477   * @param    list of tuples  
478   * @param    pair of (attribute and its possible value)  
479   * @param    list of attributes  
480   * @return    pair of (attribute - its gain) for gain-ratio  
481   *  
482   */  
483  Map<String, Double> getGainRatio(List<Map<List<String>, String>> _tuples, Map<String, List<String>> _attributes,  
484                                   List<String> _attribute) {  
485      Map<String, Double> gainstore = new HashMap<>();  
486      gainstore = getInfoGain(_tuples, _attributes, _attribute);  
487  
488      for (String attr : _attributes.keySet()) {  
489          int idx = _attribute.indexOf(attr);  
490          int[] arr = new int[_attributes.get(attr).size()];  
491          int index = 0;  
492          for (String value : _attributes.get(attr)) {  
493              int local = 0;  
494              for (int i = 0; i < _tuples.size(); i++) {  
495                  for (List<String> key : _tuples.get(i).keySet()) {  
496                      if (key.get(idx).equalsIgnoreCase(value)) {  
497                          local++;  
498                      }  
499                  }  
500              }  
501              arr[index]=local;  
502              index++;  
503          }  
504          double gain = gainstore.get(attr);  
505          gainstore.put(attr, gain/entropy(arr));  
506      }  
507      return gainstore;  
508  }  
509
```

## [getInfoGain]

```
511 * calculating info-gain
512 *
513 * @param    list of tuples
514 * @param    pair of (attribute and its possible value)
515 * @param    list of attributes
516 * @return   pair of (attribute - its gain) for info-gain
517 *
518 */
519 Map<String, Double> getInfoGain(List<Map<List<String>, String>> _tuples, Map<String, List<String>> _attributes,
520 List<String> _attribute) {
521
522     Map<String, Double> gainstore = new HashMap<>();
523     int infoArray[] = new int[_attribute.size()];
524     for (int i = 0; i < _tuples.size(); i++) {
525         String fkey = (String) _tuples.get(i).values().toArray()[0];
526         infoArray[_attribute.indexOf(fkey)]++;
527     }
528     for (String attr : _attributes.keySet()) {
529         int idx = _attribute.indexOf(attr);
530         double global = _tuples.size();
531         double gain = 0;
532         double info_a = 0;
533         for (String value : _attributes.get(attr)) {
534             double local = 0;
535             int[] infoADArray = new int[_attribute.size()];
536             for (int i = 0; i < _tuples.size(); i++) {
537                 for (List<String> key : _tuples.get(i).keySet()) {
538                     if (key.get(idx).equalsIgnoreCase(value)) {
539                         local++;
540                         String fkey = (String) _tuples.get(i).values().toArray()[0];
541                         infoADArray[_attribute.indexOf(fkey)]++;
542                     }
543                 }
544             }
545             info_a += local / global * entropy(infoADArray);
546         }
547         gain = entropy(infoArray) - info_a;
548         gain = Math.round(gain * 1000) / 1000.0;
549         gainstore.put(attr, gain);
550     }
551     return gainstore;
552 }
```

## [entropy]

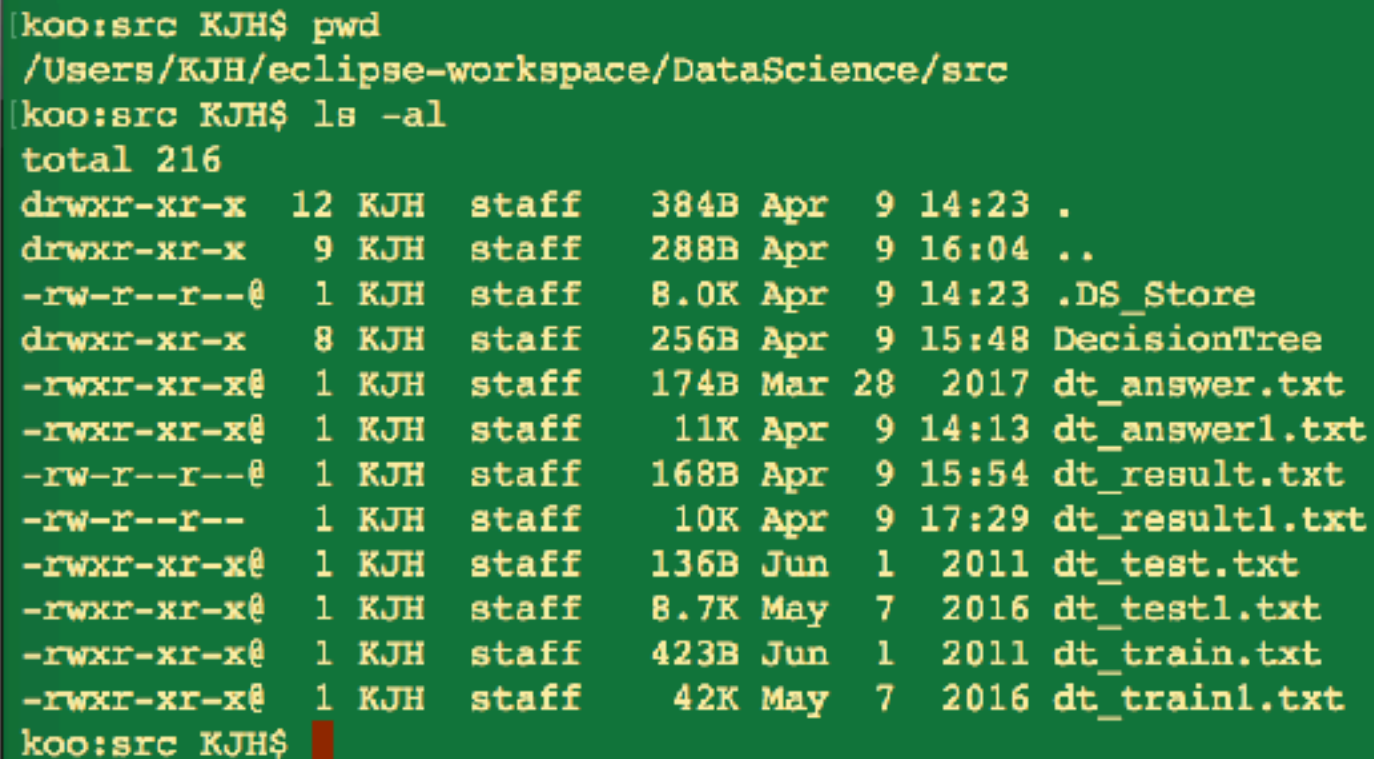
```
554  /**  
555   * calculating entropy value of each integer array  
556   *  
557   * @param    integer array  
558   * @return    its value of info gain  
559   */  
560  
561  double entropy(int[] arr) {  
562      double ret = 0;  
563      double sum = 0;  
564      for (int i = 0; i < arr.length; i++) {  
565          sum += arr[i];  
566      }  
567      for (int i = 0; i < arr.length; i++) {  
568          double p = arr[i] / sum;  
569          if (p != 0)  
570              ret -= p * Math.log(p) / Math.log(2);  
571      }  
572      return Math.round(ret * 1000) / 1000.0;  
573  }  
574
```

## # instruction for compile

### [Enviornment]

- OS : Mac OS
- Language : Java

### [Screenshot - 실행 전]

A screenshot of a macOS terminal window with a green background. The window title bar shows three colored buttons (red, yellow, green) on the left. The terminal text shows the user 'koo' in the directory 'src' of user 'KJH' running 'pwd' and 'ls -al'. The output of 'ls -al' shows a directory listing with permissions, owner, group, size, date, and filename. The files listed are '.', '..', '.DS\_Store', 'DecisionTree', 'dt\_answer.txt', 'dt\_answer1.txt', 'dt\_result.txt', 'dt\_result1.txt', 'dt\_test.txt', 'dt\_test1.txt', 'dt\_train.txt', and 'dt\_train1.txt'.

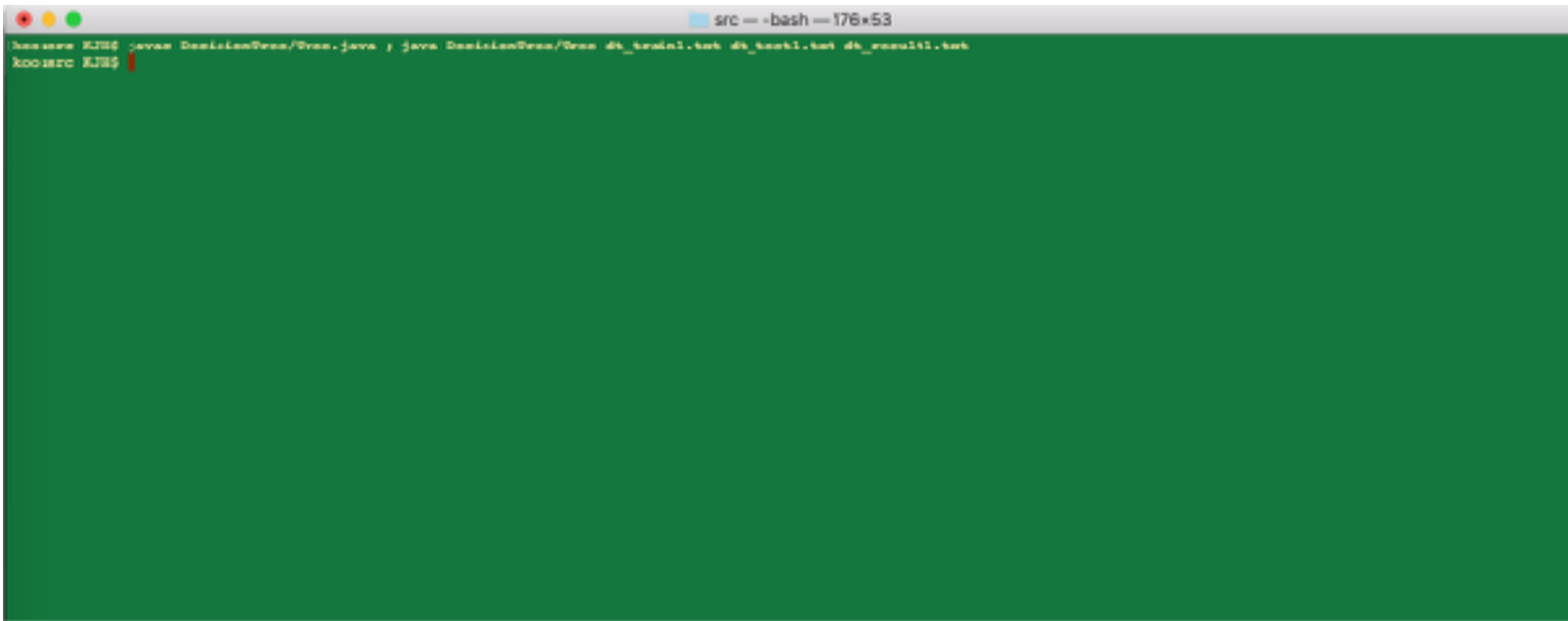
```
koo:src KJH$ pwd
/Users/KJH/eclipse-workspace/DataScience/src
koo:src KJH$ ls -al
total 216
drwxr-xr-x  12 KJH  staff   384B Apr  9 14:23 .
drwxr-xr-x   9 KJH  staff   288B Apr  9 16:04 ..
-rw-r--r--@   1 KJH  staff   8.0K Apr  9 14:23 .DS_Store
drwxr-xr-x   8 KJH  staff   256B Apr  9 15:48 DecisionTree
-rwxr-xr-x@   1 KJH  staff   174B Mar 28  2017 dt_answer.txt
-rwxr-xr-x@   1 KJH  staff    11K Apr  9 14:13 dt_answer1.txt
-rw-r--r--@   1 KJH  staff   168B Apr  9 15:54 dt_result.txt
-rw-r--r--   1 KJH  staff    10K Apr  9 17:29 dt_result1.txt
-rwxr-xr-x@   1 KJH  staff   136B Jun  1  2011 dt_test.txt
-rwxr-xr-x@   1 KJH  staff   8.7K May  7  2016 dt_test1.txt
-rwxr-xr-x@   1 KJH  staff   423B Jun  1  2011 dt_train.txt
-rwxr-xr-x@   1 KJH  staff    42K May  7  2016 dt_train1.txt
koo:src KJH$
```



## [Screenshot - 실행 방법]

compile: `$javac DecisionTree/Tree.java`

execute : `$java DecisionTree/Tree dt_train1.txt dt_test1.txt dt_result1.txt`

A screenshot of a terminal window with a dark green background. The window title bar at the top shows three colored window control buttons (red, yellow, green) on the left and the text 'src -- bash -- 176x53' on the right. The terminal content shows two lines of text: the first line is 'hanserc KJH\$ javac DecisionTree/Tree.java ; java DecisionTree/Tree dt\_train1.txt dt\_test1.txt dt\_result1.txt' and the second line is 'hanserc KJH\$' followed by a red cursor. The rest of the terminal area is empty.