Decision Tree Project Report

2016024893 오성준

1. Summary of Algorithm

- Build Decision Tree

1) 우선 train data 를 받아서 decision tree 를 만들었습니다. 이 때 init 함수에서 바로 build_tree_node 라는 함수를 통해서 만들어진 node 를 return 했습니다.

2) build_tree_node 에선 우선 data 의 class value 가 하나거나 더 이상 분류할 attribute 가 없으면 node 를 return 합니다. 이 때, 더 이상 분류할 attibute 가 없는 상황이라면, majority voting 을 통해서 class 를 결정 합니다.

3) 만약 2 의 경우가 아니라면, 남은 data 의 각 attibute 의 value 의 갯수가 같은지 다른지에 따라서 다른 함수를 통해서 기준이 되는 attribute 를 선정하도록 했습니다. 왜냐하면, information gain 이 attribute 의 value 의 개수에 영향을 받는다고 수업에서 배웠기 때문입니다. 개수가 다르다면 gain ratio 를 사용해서 attribute 를 사용하도록 했습니다.

4) Node 를 만든 뒤엔 해당하는 attribute 의 value 에 따라서 data 를 filtering 한 뒤에, 만약 data 가 있다면 build_tree_node 를 재귀적으로 실행한 후 child 로 append 하고 data 가 없다면, majority voting 을 통해서 node 를 만든 뒤에 child 에 append 합니다.

5) 1~4 사이의 과정을 지속해서 반복합니다.

- Test Decision Tree

1) 우선 test data 를 받은 뒤에 test_decision_tree 를 실행 했습니다. 이 함수에선 받은 data 를 순서대로 decision tree 에 적용해서 결과를 만듭니다.

2) 위의 과정에선 get_class 라는 함수를 통해서 class 를 구합니다. 계속해서 leaf node 로 내려가도록해서 만약 node 에 class 라는 value 가 none 이 아니라면 그 값을 return 해주고 while 문을 멈춥니다.

2. Detailed Description of codes (for each function)

1. Class node

[1] Value

1) Self.attribute: node 에서 기준이 되는 attribute 를 나타냅니다.
2) Self.val: 앞선 부모노드에서 나누어질 때 본인이 무슨 값이었는지 확인합니다, None 이면 root
3) Self.child: child node 를 모아둔 list 입니다.
4) Self.CLASS: leaf node 가 아니라면 None 값을 가집니다.
5) Self.zero_node: debugging 을 위한 value 로 majority voting 을 경험했는지 나타냅니다.

[2] Function: 없습니다.

2. Class decision_tree

[1] Value

1) Self.train_data: training data 를 저장합니다.
2) Self.class_name: train_data.columns[-1]로 class 이름을 저장합니다.
3) Self.hi: 전체 데이터의 class 에 따른 분포를 저장합니다.
4) Self.root: build_tree_node 를 바로 실행해서 return 받은 node 를 node 로 사용합니다.
5) Self.class_val: class value 를 모아둔 것입니다.

[2] Function

1) get_entropy(self, D):
   * data 의 entropy 를 구해서 return 합니다.
   * 수업 자료에 있는 공식을 그대로 사용했습니다. Info(D) = -sigma(p[i]*log(p[i]))
   * 역시 log 에는 1e-9 를 사용해서 log 에 0 이 안들어가도록 만들었습니다.

2) get_info_attr(self, D, attribute):
   * info<attribute>를 return 합니다. info<attribute>(D) = sigma(D[j] * info(D[j]))/len(D)의 공식을 따랐습니다.

3) get_node_attr_by_info_gain(self, D):
   * 위의 함수 2 개로 가장 큰 gain 을 구해서 해당하는 attribute 를 return 합니다.
   * gain(attribute) = info(D) - info<attribute>(D) 의 공식을 따랐습니다.
   * 이 때, 가장 큰 gain 이 0.2 보다 작다면 node_attr 을 None 으로 return 해서 바로 leaf node 를 만들도록 합니다.

4) get_split_info(self, D, attribute):
   * split info 를 구하는 함수 입니다.
   * split info a (D)=-sigma(Dj/D * log2(Dj/D))의 공식을 따릅니다.
   * 역시 log 에는 1e-9 를 사용해서 log 에 0 이 안들어가도록 만들었습니다.

5) get_node_attr_by_gain_ratio(self, D):
   * gain ratio 를 구해서 그 값을 통해서 attribute 를 return 합니다.
   * gain ratio 가 가장 큰 attribute 를 return 합니다.

6) majority_voting(self, D):
   * 여러 상황에서 class 를 결정해주는 함수입니다. 남은 data 에서 가장 많은 class value 를 return 합니다.
   * 이 때 만약, class value 에 대해서 data 의 수가 같은 value 가 여러 개라면 전체 data 에서 가장 개수가 적은 class name 을 선택했습니다.
   * 그 이유는 주어진 data set 에선 이 경우가 가장 정확도가 높고 sort_value 의 함수의 특성 때문에 지속해서 채점 값이 바뀌는 것을 막기 위해서 입니다.

7) get_type(self,D):
   * data 를 분석해서 attribute 의 value 의 개수가 서로 다르면 information gain 이 편향성을 가지기 때문에 값에 따라서 gain ratio 를 사용하기 위해서 type 을 분석해서 return 합니다.

* true 면 information gain, false 면 gain ratio 를 사용합니다.

8) build_tree_node(self, D, value):
* tree node 를 build 하는 함수로 algorithm summary 에서 설명했으므로 다른 설명을 생략합니다.

9) get_CLASS(self, d):
* test data 에 대해서 class 를 구해서 return 해줍니다.
* leaf node 를 확인하고 class 받아오면서 while 문을 정지합니다.

10) test_decision_tree(self, test_data)::
* test data 에 class column 을 추가합니다.
* 그 후엔, row 들마다 get_CLASS 를 구해서 값을 변경해줍니다.

11) tree_bfs(self):
* debugging 을 위한 함수로 tree 의 정보를 print 합니다
* 실제로 실행되진 않습니다.

3. function file_read
- Pd.read_csv 함수를 통해서 data file 에서 data 를 읽어와서 return 합니다.

3. Instructions for compiling your source codes at TA's computer (e.g screenshot)

```
seongjun-o@seongjun-oui-MacBookPro Project2_Decision Tree % python dt.py dt_train1.txt dt_test1.txt dt_result1.txt
seongjun-o@seongjun-oui-MacBookPro Project2_Decision Tree % python ch.py
322/346
```

1) dt.py: python dt.py [train name] [test name] [output name]
2) ch.py: python ch.py

4. Any other specification of your implementation and testing

* mono 로 만들어진 exe 파일이 제대로 작동하지 않아서 ch.py 를 만들어서 test 했습니다. 정확히 말하면, mac os 에서 mono 를 깔아야 하지만, package download 에 우선 1 시간이 걸렸고, 걸려서 다운로드를 받았음에도 불구하고 mono 라는 명령어가 제대로 작동하지 않았기 때문입니다.

```
seongjun-o@seongjun-oui-MacBookPro Project2_Decision Tree % mono dt_test.exe dt_answer.txt dt_result.txt
zsh: command not found: mono
seongjun-o@seongjun-oui-MacBookPro Project2_Decision Tree %
```

* 위의 이유로 실제 프로그램에서 돌아가는지는 친구한테 부탁을 받아서 확인하게 되었습니다.

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment2\test>dt_test.exe dt_answer1.txt dt_result1.txt
322 / 346
```

* 혹시 가능하다면 이 후에는 os 에 상관 없이 돌아갈 수 있는 채점 프로그램을 부탁드립니다.