Project2 – Decision tree implementation

Python3 (numpy, pandas) 로 구현되어 있습니다.

Installation

virtualenv 위에서 실행하는 것을 추천합니다.

```
virtualenv venv -p python3
source venv/bin/activate
# virtualenv 를 사용하지 않는 다면 아래 한 줄만 실행시키세요
pip install -r requirements.txt
```

Usage

과제 명세에 나온 그대로, 2개의 입력 파일과 1개의 출력 파일을 가집니다.

```
./dt.py train_file test_file output_file
```

Summary of algorithm

본 구현에서의 Decision tree는 N*M(N: record의 개수, M: 컬럼의 개수(target column 포함)) 어레이를 참고로 Decision tree node로 이루어진 트리를 생성한다.

각 Decision tree node는

```
class DecisionTreeNode:
  def __init__(self, indices, prev_atts, gate, result):
    self.indices = indices
    self.prev_atts = prev_atts
    self.children = []
    self.gate = gate
    self.result = result
```

5개의 데이터를 포함하고 있다.

- indices: 해당 노드를 생성할 때 참고한 어레이에서 해당 노드를 거쳐가게 되는 row들의 index들
- prev_atts: 루트 노드에서 해당 노드까지 거쳐오면서 판별하게 되는 column들
- children: 자식 노드들
- gate: None이 아닐 경우, (col, value)의 길이가 2인 튜플이어야 한다. 해당 노드에 도달하려면 `rec[col] == value` 여야 한다.
- result: 해당 노드에서 더 이상 진행하지 못할 경우의 결과 class.

Tree generation

트리 생성은 루트 노드를 생성하는 것으로 시작된다.

```
def build_root(self):
    indices = np.arange(self.db.shape[0]) # 모든 레코드를 포함
    col_ans = len(self.col_sz) - 1 # target column 평가 대상에서 제외
    root = DecisionTreeNode(indices, [col_ans], None,
self.query_result(indices))
    self.build_children(root)
    return root
```

트리 노드는 모든 row들의 index들을 가리키고 있으며, 미리 target column을 prev_atts에 포함하여 target column을 평가대상으로 삼는 것을 피한다.

그 후, 재귀적으로 각 트리노드는 한번씩

```
    def build_children(self, node):
        atts = [att for att in range(len(self.col_sz)) if att not in
    node.prev_atts]
        if len(atts) == 0: return
        gains = [self.measure(node.indices, att) for att in atts]
        t_att = atts[gains.index(max(gains))]
        sub_indices = self.split_by_att(node.indices, t_att)
        curr_atts = list(node.prev_atts)
        curr_atts.append(t_att)
        node.children = [\
          DecisionTreeNode(indices, curr_atts, (t_att, value),
    self.query_result(indices)) \
            for value, indices in enumerate(sub_indices) if len(indices) >
    0]
        for child in node.children:
          self.build_children(child)
```

`build_children` 함수를 통해 남은 컬럼들을 평가하고 자식 노드를 생성한다.

prev_atts에 포함된 컬럼들을 제외한 나머지 컬럼들을 각각 `measure` 함수를 통해 평가하여, Greedy하게 가장 좋은 컬럼을 선정한다.

선정된 컬럼 `t_att` 를 기준으로 `indices` 를 분리하여 자식 노드들을 생성한다.

각 컬럼의 평가는 `information_gain_ratio` 를 통해 이루어진다.

```
    def measure(self, indices, att):
        return self.information_gain_ratio(indices, att)

    def entropy(self, indices):
        classes = self.db[indices, -1]
        uniq, counts = np.unique(classes, return_counts=True)
        probs = counts / indices.shape[0]
        return -np.sum(probs * np.log2(probs))
```

```
    def information_gain_ratio(self, indices, att):
        prev = self.entropy(indices)
        sub_indices = self.split_by_att(indices, att)
        ig = prev - sum([self.entropy(s) * len(s) for s in sub_indices if
    len(s) > 0]) / indices.shape[0]
        return ig / (self.intrinsic_value(sub_indices) + 1e-8)

    def intrinsic_value(self, sub_indices):
        pbs = np.array([i.size for i in sub_indices if i.size > 0])
        pbs = pbs / np.sum(pbs)
        return -np.sum(pbs * np.log2(pbs))
```

`information_gain_ratio` 는 해당 컬럼을 기준으로 N개의 bin으로 나누었다고 했을 때, 각 bin의 엔트로피들의 weighted sum과 기존 엔트로피와의 차이인 `information gain` 을 `intrinsic value` 로 나눈 값이다.

`intrinsic value` 는 target column이 이 아닌 `t_att` 를 기준으로 했을 때의 엔트로피 값이다.

일단 트리가 모두 형성되고 나면, 트리노드의 `gate` 를 검사하면서 `gate` 가 가진 조건과 부합되는 경우에만 트리 아래로 내려간다.

이 때, 모든 경우에 각 record는 `children` 의 노드들중 단 하나에만 진입할 수 있거나 아예 진입할 수 없음이 확실하다.

record가 어떤 트리노드에서 더 이상 `children` 이 없거나 진입할 수 있는 노드가 없다면 해당 노드의 `result` 가 판별된 class가 된다.

```
    def query(self, record):
        for child in self.children:
          if child.gate is None: # Gate가 없을 경우 바로 진행
            return child.query(record)
          gate = child.gate
          if record[gate[0]] == gate[1]: # Gate가 있을 경우 만족하는지 검사
            return child.query(record)
        return self # 어떤 child에도 진입하지 못했다면 멈춤
```

Descriptions of code

`split_by_att` 는 db를 참고로 하여 `indices` 를 `att` 컬럼을 기준으로 분리한다. 기존에 미리 `col_sz` 에 각 컬럼의 클래스 개수를 가지고 있다.

```
    def split_by_att(self, indices, att):
        sz_sub = self.col_sz[att]
        sub_indices = [list() for _ in range(sz_sub)]
        for index in indices:
          sub_indices[self.db[index, att]].append(index)
        return [np.array(l) for l in sub_indices]
```