Decision Tree

> Pattern Recognition Homeworks
>
> Student: thu-zxs

Solutions

Problem 1

According to Decision Tree, each leaf node is assigned to the class whose samples occupied the majority.

So:

leaf node 1 of tree A is assigned to class C_1 (300>100);

leaf node 2 of tree A is assigned to class C_2 (100<300);

leaf node 1 of tree B is assigned to class C_1 (400>200);

leaf node 2 of tree B is assigned to class C_2 (0<200);

classification error of tree A :

classification error of tree B:

Also **cross-entropy** and **Gini index**:

	Node 1 of tree A	Node 2 of tree A	Node 1 of tree B	Node 2 of tree B
Cross-entropy	0.5623	0.5623	0.6365	0
Gini index	0.375	0.375	0.444	0

So **cross-entropy** of A: 0.5623+0.5623=1.1246>0.6365

And **Gini index** of A: 0.375+0.375=0.75>0.444

Programming

树结构

- 节点
  - 定义

```python
class TNode(object):
    def __init__ (self, sample_idx):
        self.feat_name_or_label = None
        self.sample_idx = sample_idx
        self.sub_node = {}
```

- 属性说明
  - `feat_name_or_label`: 当前节点用于分割的特征（非叶子结点）或当前节点的分类标签（叶子节点）
  - `sample_idx`: 当前节点包含的训练集样本
  - `sub_node`: 子节点字典:

    ```
    {'feat_value1':TNode(left_idx),'feat_value2':TNode(right_idx)}
    ```

- **树**
  - 超参数:
    - `thresh`=停止分割的阈值（不确定度减少量阈值）

```python
class DTree(object):

    def __init__ (self, thresh):

        data = sio.loadmat("Sogou_webpage.mat")
        wordMat = data['wordMat'].astype(np.int)
        doclabel = data['doclabel'].astype(np.int) - 1 # index varies from 0~8
        wordMat = self.preprocess(wordMat) # categorize

        train_idx, test_idx = self.dataSplit(np.arange(wordMat.shape[0]))
        self.train_data = wordMat[train_idx, :]
        self.test_data = wordMat[test_idx, :]
        self.train_label = doclabel[train_idx, :]
        self.test_label = doclabel[test_idx, :]

        self.thresh = thresh
```

各个主要函数被定义为树的方法

- `impurity`: 计算不纯度
  - 输入:
    - `label`=样本的分类标签
    - `select`=选择不纯度计算方法
  - 输出: `imp`=不纯度

```python
def impurity(self, label, select="gini"):

    label = np.squeeze(label)
    if len(label.shape) == 0:
        label = label[np.newaxis]
    if label.shape[0] == 0:
        return 0
    label_count = np.bincount(label)
    prob = label_count.astype(np.float32)/label.shape[0]

    if select == "entropy":
        imp = -np.sum(np.log(prob+1e-8)*prob)
    elif select == "gini":
        imp = 1-np.sum(prob**2)

    return imp
```

- `selectFeature`: 特征选择
    - 输入： `node`=当前节点
    - 输出：
        - `selected`=被选中的特征;
        - `real_idx_splitted`=此特征的分割的节点集合;
        - `max_delta`=不纯度减少量;

```python
def selectFeature(self, node):
    data, label = self.train_data[node.sample_idx],
self.train_label[node.sample_idx]
    label = np.squeeze(label)
    if len(label.shape) == 0:
        label = label[np.newaxis]
    root_imp = self.impurity(label)

    num = data.shape[0]
    max_delta = -np.inf
    selected = -1
    idx_splitted = []
    feat_set = range(0,2)
    for i in self.feat_remained:
        feat = data[:, i]
        idx_splitted_tmp = [ np.where(feat==f)[0] for f in feat_set ]
        label_splitted = [ label[idx] for idx in idx_splitted_tmp ]
        impurity_splitted = np.array([ self.impurity(l) for l in
label_splitted ])
        impurity_splitted *= np.array([ l.shape[0] for l in label_splitted
])/float(num)
        impurity_splitted = np.sum(impurity_splitted)
        if root_imp - impurity_splitted > max_delta:
```

```
                max_delta = root_imp - impurity_splitted
                selected = i
                idx_splitted = idx_splitted_tmp
    self.feat_remained = np.delete(self.feat_remained,
np.where(self.feat_remained==selected)[0][0], axis=0)
    real_idx_splitted = dict([ (feat_set[i],
node.sample_idx[idx_splitted[i]]) for i in range(len(idx_splitted)) ])

    return selected, real_idx_splitted, max_delta
```

- `splitNode`: 递归进行节点分支，分支的同时构造子节点
  - 输入： `node`=当前节点

```
def splitNode(self, node, thresh):
    ith, real_idx_splitted, max_delta = self.selectFeature(node)
    real_idx_splitted = dict([ (k, v) for k,v in
real_idx_splitted.iteritems() if len(v)!=0 ])
    if max_delta < thresh or len(real_idx_splitted.items()) == 0:
        label_cur = np.squeeze(self.train_label[node.sample_idx])
        if len(label_cur.shape) == 0:
            label_cur = label_cur[np.newaxis]
        label_count = np.bincount(label_cur)
        label_count = label_count[np.where(label_count>0)]
        node.feat_name_or_label = np.array(list(set(label_cur.tolist())))
[np.argmax(label_count)]
        return
    node.feat_name_or_label = ith
    node.sub_node = dict([ (k, TNode(v)) for k,v in
real_idx_splitted.iteritems() ])
    for n in node.sub_node.values():
      self.splitNode(n, thresh)
```

- `fit`： 即`GenerateTree`方法，命名模仿sk-learn库方式
  - 构造特征候选集合；初始化根节点；从根节点开始进行分裂（训练）

```
def fit(self):
    self.feat_remained = np.arange(self.train_data.shape[1])
    self.root = TNode(np.arange(self.train_data.shape[0]))
    self.splitNode(self.root, self.thresh)
    # self.printTree(self.root)
```

- `decision`： 决策树分类函数
  - 输入：
    - `items`： 待分类的测试样本集合
    - `record`： 是否将测试样本记录在树节点上（剪枝用到）
  - `输出`: `preds`=预测的分类结果

```python
def decision(self, items, record=False):

    preds = -np.ones(items.shape[0])
    for i, item in enumerate(items):
        node = self.root
        while not len(node.sub_node.keys()) == 0:
            if record:
                if not hasattr(node, 'val_sample_idx'):
                    node.val_sample_idx = np.array([], dtype=int)
                node.val_sample_idx = np.append(node.val_sample_idx, i)
            selected = node.feat_name_or_label
            node = node.sub_node[item[selected]]

        if record:
            if not hasattr(node, 'val_sample_idx'):
                node.val_sample_idx = np.array([], dtype=int)
            node.val_sample_idx = np.append(node.val_sample_idx, i)

        preds[i] = node.feat_name_or_label

    return preds
```

- `prune`：剪枝函数
    - 在`__init__`函数构造数据集合的基础上，将训练集平分为4份，前3份作为剪枝训练集，第4份作为验证集（validation）
    - 其中包括了一个自定义的`recurisive_prune`函数，递归进行剪枝
    - 单次剪枝实现了对所有叶子结点的剪枝，多次调用可以进行多层剪枝。

```
def prune(self):

    train_data = self.train_data.copy()
    train_label = self.train_label.copy()

    num_samples = self.train_data.shape[0]
    idxes = np.arange(num_samples)
    one_piece = num_samples/4
    pieces = np.array([ idxes[:one_piece], idxes[one_piece:2*one_piece],
idxes[2*one_piece:3*one_piece], idxes[3*one_piece:] ])

    selected_pieces = pieces[list(set(range(4)) - set([3]))]
    self.train_data = train_data[np.hstack(selected_pieces), :]
    self.train_label= train_label[np.hstack(selected_pieces), :]

    val_data = train_data[pieces[3], :]
    val_label = train_label[pieces[3], :]

    self.fit()
```

```
    preds = self.decision(val_data, record=True)
    print("Accuracy before pruned: {}".format(self.accuracy(preds, 
np.squeeze(val_label))))

    self.recursive_prune(self.root, np.squeeze(val_label))
    preds_pruned = self.decision(val_data)
    print("Accuracy after pruned: {}".format(self.accuracy(preds_pruned, 
np.squeeze(val_label))))
```

其余自定义函数

- `dataSplit`: 在树的构造函数中被调用，将数据集分成5等分，前4份训练，后1份测试。

```
def dataSplit(self, index):
    np.random.seed(2018)
    num_samples = index.shape[0]
    index_shuffled = np.random.choice(index, num_samples)
    # assert num_samples % 5 == 0
    one_piece = num_samples/5
    return index_shuffled[:4*one_piece], index_shuffled[4*one_piece:]
```

- `cross_validation`: 交叉验证函数
  - 将数据集分为4份进行4折交叉验证，打印正确率
  - 用于选取合适的超参数`thresh`

```
def cross_validation(self):

    train_data = self.train_data.copy()
    train_label = self.train_label.copy()

    num_samples = self.train_data.shape[0]
    idxes = np.arange(num_samples)
    one_piece = num_samples/4
    pieces = np.array([ idxes[:one_piece], idxes[one_piece:2*one_piece], 
idxes[2*one_piece:3*one_piece], idxes[3*one_piece:] ])

    acc = np.array([])
    for i in xrange(3, -1, -1):

        print("==== running fold {}... ====".format(4-i))
        selected_pieces = pieces[list(set(range(4)) - set([i]))]
        self.train_data = train_data[np.hstack(selected_pieces), :]
        self.train_label= train_label[np.hstack(selected_pieces), :]

        val_data = train_data[pieces[i], :]
        val_label = train_label[pieces[i], :]
```

```
        self.fit()

        preds = self.decision(val_data)
        accuracy = self.accuracy(preds, np.squeeze(val_label))
        acc = np.append(acc, accuracy)
        print("accuracy: {}".format(accuracy))

        return acc
```

- `recursive_prune`：递归进行剪枝的函数

```
def recursive_prune(self, node, val_label):

        if not hasattr(node, 'val_sample_idx'):
        return 0
    label_cur = np.squeeze(self.train_label[node.sample_idx])
    if len(label_cur.shape) == 0:
        label_cur = label_cur[np.newaxis]
    label_count = np.bincount(label_cur)
    label_count = label_count[np.where(label_count>0)]
    cls = np.array(list(set(label_cur.tolist())))[np.argmax(label_count)]
    pruned_acc = np.sum((val_label[node.val_sample_idx]==cls))

    if len(node.sub_node.keys()) == 0:
        return pruned_acc

    else:
        sub_acc = 0
        flag = 0
        for n in node.sub_node.values():
            sub_acc += self.recursive_prune(n, val_label)
            if len(n.sub_node.keys()) == 0:
                flag = 1
        if sub_acc < pruned_acc and flag == 1:
            # print("pruned!")
            node.sub_node = {}
            node.feat_name_or_label = cls
        return pruned_acc
```

- `accuracy`:准确率计算函数

```
 def accuracy(self, preds, gts):
    assert preds.shape[0] == gts.shape[0]
    return np.sum((preds==gts))/float(preds.shape[0])
```

- `printTree`:树的打印函数

```
def printTree(self, node):

    if len(node.sub_node.keys()) == 0:
        print("=== Leaf Node; Class: {}
===".format(node.feat_name_or_label))
    else:
        print("=== Branch Node ===; Feat {}
===".format(node.feat_name_or_label))
    for n in node.sub_node.values():
        self.printTree(n)
        # print("\t and its child node: {}".format(n.feat_name_or_label))
```

main函数

- 流程
  1. 交叉验证选择超参数`thresh`，并打印相应验证集准确率
  2. 选择交叉验证平均准确率最高的超参数`best_th`进行两组实验，每组实验打印训练集准确率和测试集准确率
     1. 使用全部训练集（4份）进行训练(`fit`)，并测试
     2. 使用前3份训练，后1份验证进行后剪枝(`prune`)，并测试

```
if __name__ == "__main__":

    """ select hyper params by cross-validation
    """
    thresh = np.array([0.1, 0.08, 0.05, 0.03, 0.019])
    trees = []
    acc = []
    for th in thresh:
        tree = DTree(thresh=th)
        # tree.fit()
        accuracy= tree.cross_validation()
        acc.append(accuracy.mean())
        print("Average cross-validation accuracy under threshold {}:
{}".format(th, accuracy.mean()))

    acc = np.array(acc)
    best_th = thresh[np.argmax(acc, axis=0)]

    print(" Best threshold is {}".format(best_th))



    print("--******* Under Threshold {} *******--".format(best_th))

    """ method1: use 4 fold of training set to train without pruned
```

```
    """
    best_tree = DTree(thresh=best_th)
    best_tree.fit()
    ## evaluate on training set
    preds_train = best_tree.decision(best_tree.train_data)
    accuracy_train = best_tree.accuracy(preds_train,
np.squeeze(best_tree.train_label))
    ## evaluate on test set
    preds = best_tree.decision(best_tree.test_data)
    accuracy = best_tree.accuracy(preds, np.squeeze(best_tree.test_label))
    print("--- 4 fold training ---")
    print("Training set accuracy:{}; Test set accuracy:
{}".format(accuracy_train, accuracy))

    """ method2:  use 3 fold of training set to train and prune the tree by 1
fold of validation set
    """
    print("--- 3 fold training and prune ---")
    best_tree1 = DTree(thresh=best_th)
    best_tree1.prune()
    ## evaluate on training set
    preds_train = best_tree1.decision(best_tree1.train_data)
    accuracy_train = best_tree1.accuracy(preds_train,
np.squeeze(best_tree1.train_label))
    ## evaluate on test set
    preds = best_tree1.decision(best_tree1.test_data)
    accuracy = best_tree1.accuracy(preds, np.squeeze(best_tree1.test_label))
    print("Training set accuracy:{}; Test set accuracy:
{}".format(accuracy_train, accuracy))
```

实验结果

实验中发现交叉熵不纯度性能比基尼指数差的较多，故使用基尼指数开展实验

候选超参数：`thresh=[0.1, 0.08, 0.05, 0.03, 0.019]`

`thresh`=0.019时，达到最好的结果:

- 交叉验证集平均准确率：0.7773
- `fit`训练，测试集准确率：0.7832
- `prune`训练+后剪枝，测试集准确率：0.7495