大数据导论lab2 实验报告

220110519 邢瑞龙

数据集以及预处理

数据集：Adult Data Set

训练集数量：26904

测试集数量：14130

样本特征：14种

样本类别：2种（个人收入大于50K or 小于等于50K）

预处理：

1. 去除缺失值，异常值，重复行：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import collections
import random

columns = ['age', 'workclass', 'fnlwgt', 'education', 'educationNum',
'maritalStatus', 'occupation', 'relationship', 'race', 'sex',
          'capitalGain', 'capitalLoss', 'hoursPerWeek', 'nativeCountry',
'income']
df_train_set = pd.read_csv('./adult.data', names=columns)
df_test_set = pd.read_csv('./adult.test', names=columns, skiprows=1) #第一行是
非法数据

print(len(df_train_set))
print(len(df_test_set))
df_train_set.to_csv('./train_adult.csv', index=False)
df_test_set.to_csv('./test_adult.csv', index=False)

df_train_set = pd.read_csv('./train_adult.csv')
df_test_set = pd.read_csv('./test_adult.csv')

df_train_set.drop(['fnlwgt', 'educationNum'], axis=1, inplace=True) # fnlwgt
列用处不大，educationNum与education类似
df_test_set.drop(['fnlwgt', 'educationNum'], axis=1, inplace=True)
df_train_set.drop_duplicates(inplace=True) # 去除重复行
df_test_set.drop_duplicates(inplace=True)
df_train_set.dropna(inplace=True) # 去除空行
df_test_set.dropna(inplace=True)

# 去除含有'?'的行
new_columns = ['workclass', 'education', 'maritalStatus', 'occupation',
'relationship', 'race', 'sex',
               'nativeCountry', 'income']
```

```
for col in new_columns:
    df_train_set = df_train_set[~df_train_set[col].str.contains(r'\?', 
regex=True)]
    df_test_set = df_test_set[~df_test_set[col].str.contains(r'\?', 
regex=True)]
#save to csv
```

2. 连续变量离散化（为了缩小决策树的最优切分点的搜索范围，这里将连续变量离散化）

```
#连续变量离散化
continuous_column = ['age', 'capitalGain', 'capitalLoss', 'hoursPerWeek']
allbins = [[0, 20, 40, 60, 80,100],
        [0, 10000, 50000, 100000],
        [0,1,5000],
        [0, 20, 40, 60, 80, 100]]
for col, bins in zip(continuous_column, allbins):
    df_train_set[col] = pd.cut(df_train_set[col], bins, right=False, 
labels=False)
    df_test_set[col] = pd.cut(df_test_set[col], bins, right=False, 
labels=False)

print(df_train_set.head())
print(df_test_set.head())
```

3. 离散变量index化（为了方便程序处理，这里将单个属性下的不同类别映射到不同的数字）

```
# #离散变量index化
discrete_column = ['workclass', 'education', 'maritalStatus', 'occupation', 
'relationship', 'race', 'sex', 'nativeCountry', 'income']
workclass_mapping = {' Private': 0, ' Self-emp-not-inc': 1, ' Self-emp-inc': 
1, ' Local-gov': 2,
                     ' State-gov': 2, ' Federal-gov': 2, ' Without-pay': 3, ' 
Never-worked': 3}
education_mapping = {
    ' 5th-6th': 0,
    ' 7th-8th': 0,
    ' 9th': 0,
    ' 10th': 0,
    ' 11th': 0,
    ' 12th': 0,
    ' HS-grad': 0,
    ' Preschool': 1,
    ' 1st-4th': 1,
    ' Assoc-acdm': 2,
    ' Assoc-voc': 2,
    ' Some-college': 3,
    ' Bachelors': 3,
    ' Doctorate': 4,
    ' Prof-school': 4,
    ' Masters': 4
}
```

```python
income_mapping = {' <=50K': 0, ' <=50K.':0, ' >50K': 1, ' >50K.': 1}
special_mapping_name = ['workclass', 'education' 'income']
df_test_set['workclass'] = df_test_set['workclass'].map(workclass_mapping)
df_train_set['workclass'] = df_train_set['workclass'].map(workclass_mapping)
df_test_set['education'] = df_test_set['education'].map(education_mapping)
df_train_set['education'] = df_train_set['education'].map(education_mapping)
df_test_set['income'] = df_test_set['income'].map(income_mapping)
df_train_set['income'] = df_train_set['income'].map(income_mapping)
print(df_train_set.head())
print(df_test_set.head())
for col in discrete_column:
    if(col in special_mapping_name):
      continue
    else:
        res1 = df_test_set[col].value_counts().keys()
        res2 = df_train_set[col].value_counts().keys()
        res = list(set(res1).union(set(res2)))
        mapping = dict(zip(res, range(len(res))))
        print(mapping)
        df_train_set[col] = df_train_set[col].map(mapping)
        df_test_set[col] = df_test_set[col].map(mapping)
print(df_train_set.head())
print(df_test_set.head())
#save to csv
df_train_set.to_csv('../train_adult_pro.csv', index=False)
df_test_set.to_csv('../test_adult_pro.csv', index=False)
```

- 特殊地，对于workclass：我们将其分为四类

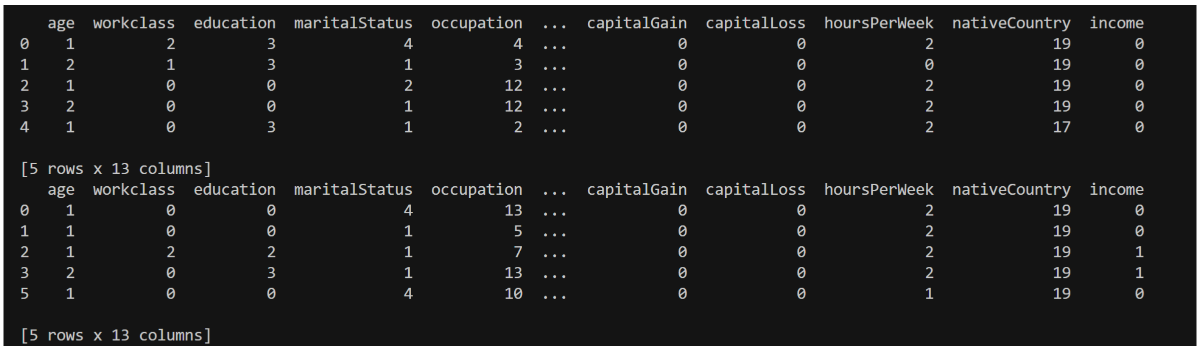
```python
workclass_mapping = {' Private': 0, ' Self-emp-not-inc': 1, ' Self-emp-inc': 1, ' Local-gov': 2, ' State-gov': 2, ' Federal-gov': 2, ' Without-pay': 3, ' Never-worked': 3}
```

- 对于education,这里划分为四类：幼稚园阶段，中小学阶段，大学阶段和更高学历阶段

```python
education_mapping = {
    ' 5th-6th': 0,
    ' 7th-8th': 0,
    ' 9th': 0,
    ' 10th': 0,
    ' 11th': 0,
    ' 12th': 0,
    ' HS-grad': 0,
    ' Preschool': 1,
    ' 1st-4th': 1,
    ' Assoc-acdm': 2,
    ' Assoc-voc': 2,
    ' Some-college': 3,
    ' Bachelors': 3,
    ' Doctorate': 4,
    ' Prof-school': 4,
    ' Masters': 4
}
```

- 其余属性不做特殊处理

预处理结果（部分）：



```
   age  workclass  education  maritalStatus  occupation  ...  capitalGain  capitalLoss  hoursPerWeek  nativeCountry  income
0    1          2          3              4           4  ...            0            0             2             19       0
1    2          1          3              1           3  ...            0            0             0             19       0
2    1          0          0              2          12  ...            0            0             2             19       0
3    2          0          0              1          12  ...            0            0             2             19       0
4    1          0          3              1           2  ...            0            0             2             17       0

[5 rows x 13 columns]
   age  workclass  education  maritalStatus  occupation  ...  capitalGain  capitalLoss  hoursPerWeek  nativeCountry  income
0    1          0          0              4          13  ...            0            0             2             19       0
1    1          0          0              1           5  ...            0            0             2             19       0
2    1          2          2              1           7  ...            0            0             2             19       1
3    2          0          3              1          13  ...            0            0             2             19       1
5    1          0          0              4          10  ...            0            0             1             19       0

[5 rows x 13 columns]
```

决策树算法

决策树算法选择：**CART**

1. **问题假设**：

   假设数据集 D 中有 n 个样本，样本属于 k 个属性 $\{C_1, C_2, ..., C_k\}$，每个属性取值个数为 $N_1, N_2......N_k$。对于每一个属性 和其取值C_i，V_j，我们希望计算其基尼系数 $Gini(C_i)$，然后选择最大的属性和值作为当前节点的判别依据,其输出类别$Y \in \{y_1, y_2\}, y_1, y_2$分别为个人收入$>= 50K$和$< 50K$

2. **总体信息熵**：
   当前节点$father$ 的基尼系数 $Gini(father)$ 为：

   $$Gini(father) = 2p(1-p)$$

   其中，p 是类别 y_1 在节点$father$ 中的样本比例。

3. **选取一个类别 C以及其对应的一个取值V**：
   每次选取一个属性C_i 来计算划分后数据的基尼系数。将当前节点 $father$ 按照属于属性 C_i 是否等于V 划分为2个子集：

   - $father_V$ 和$father_{\neg V}$

4. **划分后基尼系数计算**：
   根据类别 C_i，V 的划分计算在 $Gini(father|C_i)$，即按属性 C_i 划分后的基尼系数：

   $$Gini(father|C_i) = \frac{|father_V|}{|father|}Gini(father_V) + \frac{|father_{\neg V}|}{|father|}Gini(father_{\neg V})$$

5. **impurity reduction**：
   衡量划分优劣：

   $$\Delta G = G(father) - Gini(father|C_i)$$

6. **选取impurity reduction最大的类别以及取值**：
   计算所有类别 $C_1, C_2, ..., C_k$ 的所有取值V下对应的划分优劣，选择ΔG最大的类别 $C_{\text{best}}, V_{best}$ 作为当前节点的判别依据：

   $$C_{\text{best}}, V_{best} = \arg\max_i \Delta(C_i, V_j)$$

7. **继续递归**：
   根据选中的 $C_{\text{best}}, V_{best}$ 将数据集划分为子集，并递归执行以上步骤，直到满足终止条件。

```
import numpy as np
```

```python
import pandas as pd
import random
cart_config = {
    'max_depth': 10,
    'min_samples_split': 2,
    'min_gini': 0.02
}
unused_splits = set()  # 记录已经使用过的切分点，避免重复使用

class Node:
    def __init__(self, feature_index, feature_value, left, right, label):
        self.feature_index = feature_index
        self.feature_value = feature_value
        self.left = left
        self.right = right
        self.label = label

    def predict(self, x):
        if self.label is not None:
            return self.label
        if x[self.feature_index] == self.feature_value:
            return self.left.predict(x)
        else:
            return self.right.predict(x)

    def __str__(self):
        return 'feature_index: %d, feature_value: %d, label: %d' %
(self.feature_index, self.feature_value, self.label)

def calc_gini(y):
    """
    计算数据集的基尼指数
    :param X: 当前节点所包含数据
    :return: 基尼指数
    """
    n = len(y)
    if n == 0:
        return 0
    m = y.sum()
    prob = m / n
    gini = 2 * prob * (1 - prob)
    return gini

def split_dataset(X, y,feature_index, feature_value):
    """
    按照给定的列划分数据集
    :param X: 当前节点所包含数据
    :param index: 指定特征的列索引
    :param value: 指定特征的值
    :return: 切分后的数据集
    """
    left = X[X[:, feature_index] == feature_value]
    right = X[X[:, feature_index] != feature_value]
    left_labels = y[X[:, feature_index] == feature_value]
    right_labels = y[X[:, feature_index] != feature_value]
    return left, right, left_labels, right_labels
```

```
def choose_best_feature_to_split(X, y):
    """
    选择最好的特征进行分裂
    :param X: 当前节点数据
    :return: best_value:(分裂特征的index, 特征的值), best_df:(分裂后的左右子树数据集),
best_gain:(选择该属性分裂的最大信息增益)
    """
    best_gini = 1
    best_feature = -1
    best_split = None
    best_value = -1
    n = X.shape[0]
    for (i, j) in unused_splits:
        left, right, left_labels, right_labels = split_dataset(X, y, i, j)
        gini = left.shape[0]/n * calc_gini(left_labels) + right.shape[0]/n *
calc_gini(right_labels)
        if gini < best_gini:
            best_gini = gini
            best_feature = i
            best_value = j
            best_split = (left, right, left_labels, right_labels)
    return best_feature, best_value, best_split, best_gini


def build_decision_tree(X, y, depth, flags):
    """
    构建CART树
    :param X: 数据集
    :param y: 标签集
    :param depth: 当前深度
    :return: CART树
    """
    if(len(np.unique(y)) == 1):
        return Node(None, None, None, None,np.argmax(np.bincount(y)))
    if depth >= cart_config['max_depth']:
        return Node(None, None, None, None, np.argmax(np.bincount(y)))
    if y.shape[0] <= cart_config['min_samples_split']:
        return Node(None, None, None, None, np.argmax(np.bincount(y)))
    gini = calc_gini(y)
    if(gini <= cart_config['min_gini']):
        return Node(None, None, None, None, np.argmax(np.bincount(y)))
    if(len(unused_splits) == 0):
        return Node(None, None, None, None, np.argmax(np.bincount(y)))
    best_feature, best_value, best_split, best_gini =
choose_best_feature_to_split(X, y)
    if(best_gini >= gini):
        return Node(None, None, None, None, np.argmax(np.bincount(y)))
    node = Node(best_feature, best_value, None, None, None)
    node.left = build_decision_tree(best_split[0], best_split[2], depth + 1,
flags)
    node.right = build_decision_tree(best_split[1], best_split[3], depth + 1,
flags)
    return node
```

```python
def save_decision_tree(cart):
    """
    决策树的存储
    :param cart: 训练好的决策树
    :return: void
    """
    np.save('cart.npy', cart)


def load_decision_tree():
    """
    决策树的加载
    :return: 保存的决策树
    """

    cart = np.load('cart.npy', allow_pickle=True)
    return cart.item()
```

训练集和验证集划分

```python
def split_train_val(X, y, ratio=0.8, seed=42):
    """
    划分训练集和验证集
    :param X: 特征
    :param y: 标签
    :param ratio: 训练集比例
    :return: X_train, y_train, X_val, y_val
    """
    np.random.seed(seed)
    n = X.shape[0]
    y = y.astype(int)
    indices = np.arange(n)
    np.random.shuffle(indices)
    X, y = X[indices], y[indices]
    split = int(n * ratio)
    X_train, y_train = X[:split], y[:split]
    X_val, y_val = X[split:], y[split:]
    return X_train, y_train, X_val, y_val
```

- 本次实验采用经典的将train_data按照8:2分为训练集和验证集
- **note**:为了简化算法空间复杂度以及实现过程，这里将可能的特征分割点提前计算出来，并在构建树的时候**不会**随着树的构建一步步剔除。意思就每次计算geni系数时，都是所有的特征再算一次。由于每次都是取最好的，所以不会对算法的最优性造成影响，只是会增加算法的部分时间。

前剪枝

为了以防决策树过拟合，参数设置：

- 最大深度为10
- 节点分割最少样本数量最少为2

- 节点内基尼系数最小为0.02

```
cart_config = {
    'max_depth': 10,
    'min_samples_split': 2,
    'min_gini': 0.02
}
```

后剪枝

在训练集上训练好后，我们对决策树在验证集上做后剪枝

```
def prune_tree(node, X_val, y_val):
    """
    对树进行后剪枝
    :param node: 当前节点
    :param X_val: 验证集特征
    :param y_val: 验证集标签
    :return: 剪枝后的节点
    """
    if y_val.shape[0] == 0:
        return node
    if node.label is not None:
        return node

    # 对左右子树进行递归剪枝
    if node.left:
        node.left = prune_tree(node.left, X_val[X_val[:, node.feature_index] ==
node.feature_value], y_val[X_val[:, node.feature_index] == node.feature_value])
    if node.right:
        node.right = prune_tree(node.right, X_val[X_val[:, node.feature_index] !=
node.feature_value], y_val[X_val[:, node.feature_index] != node.feature_value])

    # 评估当前节点是否需要剪枝
    if node.left is not None and node.right is not None:
        # 如果左右子节点都是叶子节点，则考虑剪枝
        if node.left.label is not None and node.right.label is not None:
            # 剪枝前后的准确率比较
            left_right_preds = np.where(X_val[:, node.feature_index] ==
node.feature_value, node.left.label, node.right.label)
            before_prune_acc = np.mean(left_right_preds == y_val)

            # 剪枝，直接将当前节点作为叶子节点
            node_preds = np.full(X_val.shape[0], np.argmax(np.bincount(y_val)))
            after_prune_acc = np.mean(node_preds == y_val)

            if after_prune_acc >= before_prune_acc:
                # 如果剪枝后准确率没有下降，则进行剪枝
                node = Node(None, None, None, None,
np.argmax(np.bincount(y_val)))

    return node
```

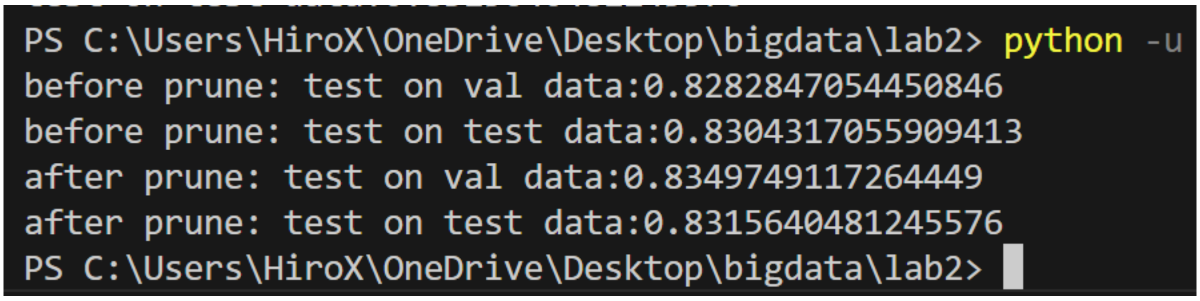
```python
cart = build_decision_tree(X_train, y_train, 0, unused_splits)
    cnt = 0
    for i in range(X_val.shape[0]):
        if(cart.predict(X_val[i]) == y_val[i]):
            cnt += 1
    print(f"before prune: test on val data:{cnt/X_val.shape[0]}")

    # 后剪枝
    cart = prune_tree(cart, X_val, y_val)

    cnt = 0
    for i in range(X_val.shape[0]):
        if(cart.predict(X_val[i]) == y_val[i]):
            cnt += 1
    print(f"before prune: test on val data:{cnt/X_val.shape[0]}")
```

测试结果以及准确率

- 不进行后剪枝在测试集准确率：0.8305024769992922
- 进行后剪枝在测试集准确率：0.8316348195329087



```
PS C:\Users\HiroX\OneDrive\Desktop\bigdata\lab2> python -u
before prune: test on val data:0.8282847054450846
before prune: test on test data:0.8304317055909413
after prune: test on val data:0.8349749117264449
after prune: test on test data:0.8315640481245576
PS C:\Users\HiroX\OneDrive\Desktop\bigdata\lab2>
```

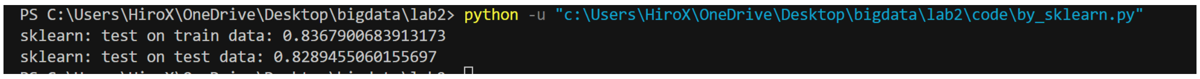
- 表明剪枝起了效果

对比sklearn结果：

参数设置

```python
clf = DecisionTreeClassifier(max_depth=10, random_state=42, criterion='gini',
splitter='best',min_samples_split=2)
```

- sklearn在测试集准确率:0.8289455060155697



```
PS C:\Users\HiroX\OneDrive\Desktop\bigdata\lab2> python -u "c:\Users\HiroX\OneDrive\Desktop\bigdata\lab2\code\by_sklearn.py"
sklearn: test on train data: 0.8367900683913173
sklearn: test on test data: 0.8289455060155697
```

> 还比sklearn更优？ :)