Introduction to Artificial Intelligence Prog. 4 Report
0516003 李智嘉

1. Introduction

- 這次作業我將實做CART based的Decision Tree，並以其實做Random Forest
- 實驗的部份，我將針對三種Classification Dataset做測試，分別是iris、wine和ionosphere，詳細資訊如下方列表

Dataset Name	# Data	# Feature	# Class
iris	150	4	3
wine	178	12	3
ionosphere	351	34	2

- 每次會跑一次或多次，每次都將training與testing data以8:2的比例隨機切割。將model fit到training data後，inference在testing data上，最後算平均testing的accuracy
- 以下我將針對下列幾點做實驗
  i. Data Boostrap的影響
  ii. Random Forest tree number的影響
  iii. 限制Tree不同深度的影響
  iv. Tree每次split時，所考慮的feature數量

2. Experiment

- Data Boostrap
  - setting
    1. tree_num: 30
    2. max sample feature number: None
    3. tree max depth: 2
    4. experiment times: 1

	without Boostrap	with Boostrap
iris	0.920	0.931
wine	0.840	0.923
ionosphere	0.817	0.900

可以看到，在三種dataset上面，有Boostrap的表現普遍比沒有做還要高上不少

而剛好其增長的幅度與data的複雜程度（data數量、feature數量）成正相關，ionosphere成長最多，iris成長最少。原因可能也是因為data越複雜，做Boostrap後能讓decision tree吃到更多樣的training data，使樹的diversity更高，能表達的範圍更廣

- Randome Forest tree number
  - setting
    1. max sample feature number: None
    2. tree max depth: None
    3. tree data bootstrap: True
    4. experiment times: 3

	tree_num = 1	tree_num = 10	tree_num = 20
iris	0.932	0.932	0.924

可以看到，tree number對成功率的影響竟然並沒有多大差別，但應該是因為沒有限制tree深度的情況，於是下面實驗，我設定每棵數最高只能長到深度2

	tree_num = 1	tree_num = 5	tree_num = 10
iris	0.932	0.932	0.919
wine	0.878	0.946	0.940
ionosphere	0.903	0.895	0.893

當深度限制在2的時候，對於iris dataset，不同tree number還是沒有太大的改變，可能是因為iris相對太簡單了
像是wine dataset，tree_num=1時表現只有0.878，到tree_num=5時就進步到0.946，而對於這份dataset，大概樹的數量到這邊也就飽和了，tree_num再往上提可以看到沒有太大的差別
最後是inosphere，也可以看到tree_num對這份dataset的影響並不是很大。有可能是因為這個task雖然feature多，但是是binary classification，深度限制為2的tree也已經夠用了

- Limit Tree Depth
  - setting
    1. tree_num: 30
    2. max sample feature number: None
    3. tree data bootstrap: True
    4. experiment times: 1

	depth=1	depth=2	depth=3	depth=4
iris	0.548	0.917	0.960	0.875

wine	0.735	0.971	0.968	1.0
ionosphere	0.847	0.932	0.899	0.907

從上面數據可以看到三種data，將深度增加，都可以增加accuracy的表現。
對於iris dataset，樹的深度到3都還可以讓accuracy增加，但到深度4看起來就因為overfitting，讓testing的accuracy從0.96變成0.875了
對於wine dataset，深度看起來2~4都是差不多的，depth=4時accuracy變成1可能是sample上的巧合
但對於ionosphere，深度是2時表現最好，3和4變得差一點，但表現也是差不多
這邊比較可惜的是因為執行時間關係，沒有再測試深度限制很深的時候accuracy如何

- Limit Feature Number
  - setting
    1. tree_num: 30
    2. max_depth: 3
    3. tree data bootstrap: True
    4. experiment times: 1

	# feature=1	# feature=2	# feature=3	# feature=4
iris	0.923	1.0	0.979	0.973

	# feature=1	# feature=4	# feature=8	# feature=12
wine	0.897	0.970	0.952	0.942

	# feature=1	# feature=10	# feature=22	# feature=34
ionosphere	0.882	0.960	0.960	0.906

可以看到，在三種data上面，sample一定數量的feature去切node都表現的比用全部的feature還要好，這也驗證了增加tree的diversity是有助於整體model的performance的
比較有趣的是，只取一個feature去切（Extremely random forest），表現其實不會說太差，在iris的data甚至還有0.92的accuracy，可能也是因為這些task相對來說都算簡單，且random forest的樹有一定數量，如果再更多顆的話應該會表現的更好（但由於硬體設備限制，30顆就跑相當久的時間了）

3. Conclusion

- 這次報告中，我實做了decision tree與random forest，並在三種不同大小的data上做了些實驗

- 實驗著重在一些Random Forest的特色，包括Boostrap, tree number, tree depth, feature bagging，這些參數對於accuracy的影響情況
- 比較可惜的是，要跑實驗的話，效能實在是太重要了。可能是因為我是用pandas來寫，所以執行速度可能會慢相當多，也就不能跑太複雜的實驗。如果有機會的話，我會把pandas換掉，只用list和dict來做，讓效率提高，才可以跑更多實驗
- 另外一點也是實驗上的遺憾，這次報告所使用的三種data不管在data數量還是feature數量上甚至是任務本身，都是相當簡單的，這也造成random forest的樹的數量就算只有一顆（退化成decision tree），表現也不會說太差。如果有機會再做更複雜的實驗的話，我會嘗試找更複雜，更難的data來做

main.py

```
import model
import pandas as pd
import numpy as np
from tqdm import tqdm

############################
tree_num = 30
max_feature = None
max_depth = 2
boostrap = True
criterion = 'gini'

experiment_num = 1
data = 'ionosphere'    ## iris wine ionosphere
############################

def read_data(data='iris'):
    return pd.read_csv(f'{data}.csv')

print('====================================')
print(f'Data {data}')
print(f'tree_num {tree_num}')
print(f'max_feature {max_feature}')
print(f'max_depth {max_depth}')
print(f'boostrap {boostrap}')
print(f'criterion {criterion}')
print(f'experiment_num {experiment_num}')
print('====================================')
data = read_data(data)

avg_acc = 0.
for _ in tqdm(range(experiment_num)):
    msk = np.random.rand(len(data)) < 0.8
    train_data, test_data = data[msk], data[~msk]
    random_forest = model.RandomForest(tree_num, max_feature, boostrap,
criterion, max_depth)
    random_forest.fit(train_data)
    acc = random_forest.score(test_data)
    # print(f'Accuracy: {acc}')
```

```
    avg_acc += acc
print(f'Average accuracy: {avg_acc/experiment_num}')
```

model.py

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from collections import Counter
import matplotlib.pyplot as plt
import random
import utils

class DecisionTree():
    def __init__(self, criterion='gini', max_depth=None,
max_features=None):
        if criterion != 'gini' and criterion != 'entropy':
            raise Exception('DecisionTree key error: criterion myst be
"gini" \
                            or "entropy"')

        self.criterion = criterion
        self.max_depth = max_depth
        self.tree = []  # each element would be
                        # [split_feature, split_value, larger_goto_idx,
                        #  others_goto_idx, category]
        self.max_features = max_features
        if self.max_features is not None:
            self.max_features = int(self.max_features)

    def check_terminate(self, y):
        first = None
        for i in y:
            if first is None:
                first = i
            elif first != i:
                return False
        return True

    def fit(self, x, y):
        if type(y) == pd.core.frame.DataFrame:
            y = y['0']
```

```
        self.tree.append([])
        self.split(x, y, 1, 0)

    def pickup_feature_names(self, keys):
        if self.max_features is None or len(keys) < self.max_features:
            return keys
        else:
            return random.sample(list(keys.to_numpy()),
self.max_features)

    def split(self, x, y, depth, tree_idx):
        if (self.max_depth is not None and depth > self.max_depth) or
self.check_terminate(y.to_numpy()):
            cnt = Counter(y.to_numpy())
            self.tree[tree_idx] = \
                [None, None, None, None, cnt.most_common(1)[0][0]]
            return

        while True:
            feature_names = self.pickup_feature_names(x.keys())

            best_split_feature = None
            best_split_value = None
            best_criteria = None

            for feature_name in feature_names:
                feature = x[feature_name].sort_values().to_numpy()
                for idx in range(1, len(feature)):
                    split_value = (feature[idx-1]+feature[idx])/2
                    larger_y, others_y = y[x[feature_name] >
split_value], y[x[feature_name] <= split_value]
                    larger_n, others_n = len(larger_y), len(others_y)
                    if self.criterion == 'gini':
                        new_gini =
larger_n*utils.gini(larger_y.to_numpy()) +
others_n*utils.gini(others_y.to_numpy())
                        new_gini /= (larger_n+others_n)
                        if best_criteria is None or new_gini <
best_criteria:
```

```
                            best_criteria = new_gini
                            best_split_feature = feature_name
                            best_split_value = split_value
                    elif self.criterion == 'entropy':
                        after_entropy =
larger_n*utils.entropy(larger_y.to_numpy()) +
others_n*utils.entropy(others_y.to_numpy())
                        after_entropy /= (larger_n+others_n)
                        if best_criteria is None or after_entropy <
best_criteria:
                            best_criteria = after_entropy
                            best_split_feature = feature_name
                            best_split_value = split_value

            larger_y, others_y = y[x[best_split_feature] >
best_split_value], y[x[best_split_feature] <= best_split_value]
            larger_n, others_n = len(larger_y), len(others_y)
            if self.criterion == 'gini':
                init_criteria = utils.gini(y.to_numpy())
                new_gini = larger_n*utils.gini(larger_y.to_numpy()) +
others_n*utils.gini(others_y.to_numpy())
                new_gini /= (larger_n+others_n)
            elif self.criterion == 'entropy':
                init_criteria = utils.entropy(y.to_numpy())
                after_entropy =
larger_n*utils.entropy(larger_y.to_numpy()) +
others_n*utils.entropy(others_y.to_numpy())
                after_entropy /= (larger_n+others_n)

            if (x[best_split_feature] > best_split_value).sum() == 0 or
\
                    (x[best_split_feature] <= best_split_value).sum() ==
0:
                continue
            else:
                break

        greater_idx = len(self.tree)
        others_idx = greater_idx+1
```

```
        self.tree[tree_idx] = [best_split_feature, best_split_value,
greater_idx, others_idx, None]
        self.tree.append([])
        self.tree.append([])
        self.split(x[x[best_split_feature] > best_split_value],
larger_y, depth+1, greater_idx)
        self.split(x[x[best_split_feature] <= best_split_value],
others_y, depth+1, others_idx)

    def predict_single(self, x):
        # self.tree: [split_feature, split_value, larger_goto_idx,
others_goto_idx, category]
        cur_tree_idx = 0
        while self.tree[cur_tree_idx][4] is None:
            if x[self.tree[cur_tree_idx][0]] >
self.tree[cur_tree_idx][1]:
                cur_tree_idx = self.tree[cur_tree_idx][2]
            else:
                cur_tree_idx = self.tree[cur_tree_idx][3]
        return self.tree[cur_tree_idx][4]

    def predict(self, x):
        pred_y = []
        for _, row in x.iterrows():
            pred_y.append(self.predict_single(row))
        return pred_y

    def score(self, x, y, return_pred_y=False):
        pred_y = self.predict(x)
        tot_n, acc_n = len(pred_y), 0
        y = y.to_numpy()
        for i in range(tot_n):
            if y[i] == pred_y[i]:
                acc_n += 1
        if return_pred_y:
            return acc_n/tot_n, pred_y
        else:
            return acc_n/tot_n

class RandomForest():
```

```
    def __init__(self, n_estimators, max_features, boostrap=True,
criterion='gini', max_depth=None):
        self.n_tree = n_estimators
        self.boostrap = boostrap
        self.trees = []
        for _ in range(n_estimators):
            self.trees.append(DecisionTree(criterion=criterion,
max_depth=max_depth, max_features=max_features))

    def fit(self, base_x):
        for tree_idx in range(self.n_tree):
            if self.boostrap:
                new_x = base_x.sample(n=len(base_x), replace=True)
            else:
                new_x = base_x
            new_y = new_x['y']
            new_x = new_x.loc[:, new_x.columns != 'y']

            self.trees[tree_idx].fit(new_x, new_y)

    def predict(self, x):
        res = None
        for i in range(self.n_tree):
            pred = self.trees[i].predict(x)
            pred = np.array(pred).reshape(-1, 1)
            if res is None:
                res = pred
            else:
                res = np.concatenate((res, pred), axis=1)
        return [Counter(res[i]).most_common(1)[0][0] for i in
range(len(x))]

    def score(self, data, return_pred_y=False):
        y = data['y']
        x = data.loc[:, data.columns != 'y']
        pred_y = self.predict(x)
        tot_n, acc_n = len(pred_y), 0
        y = y.to_numpy()
        for i in range(tot_n):
            if y[i] == pred_y[i]:
```

```
                acc_n += 1
        if return_pred_y:
            return acc_n/tot_n, pred_y
        else:
            return acc_n/tot_n
```

model.py

```
import numpy as np

def gini(sequence):
    categories = set(sequence)
    res, num = 1., len(sequence)
    if num == 0:
        return 0
    for category in categories:
        res -= (len(sequence[sequence == category])/num) ** 2
    return res


def entropy(sequence):
    categories = set(sequence)
    res, num = 0, len(sequence)
    if num == 0:
        return 0
    for category in categories:
        prob = len(sequence[sequence == category])/num
        res += -prob*np.log2(prob)
    return res
```