Programming Assignment #4 Report

0516007 高誌佑

1. Experiments results

Below are the environments in my experiments:

- Machine (CPU): AMD Ryzen 7-3700X 3.6GHz
- Operating System: Ubuntu 18.04
- Language: **Python 3.7.7**

I try to compare the performance with the below three kinds of experiments

- **Relative sizes of the training and validation subsets**
- **Number of trees in the forest**
- **Extremely Random forest**
- **Attribute Bagging**

Next two pages show the two tables about the results of Iris and Wine datasets.

For each setting in one row, I run the experiment 10 times and calculate the average accuracy.

- **Iris** Dataset (Number of validation data: 30)

#Trees	#Training Data	Attribute Bagging	Extremely Random forest	Accuracy
1	10	Yes	No	78.00%
1	10	No	No	70.99%
1	10	No	Yes	68.33%
5	10	Yes	No	93.33%
5	10	No	No	80.33%
5	10	No	Yes	77.67%
10	10	Yes	No	85.00%
10	10	No	No	83.67%
10	10	No	Yes	79.67%
1	30	Yes	No	89.00%
1	30	No	No	83.67%
1	30	No	Yes	80.00%
5	30	Yes	No	94.00%
5	30	No	No	91.33%
5	30	No	Yes	83.00%
10	30	Yes	No	94.67%
10	30	No	No	92.67%
10	30	No	Yes	88.33%
1	60	Yes	No	95.00%
1	60	No	No	93.00%
1	60	No	Yes	90.33%
5	60	Yes	No	97.33%
5	60	No	No	96.33%
5	60	No	Yes	91.00%
10	60	Yes	No	97.33%
10	60	No	No	96.67%
10	60	No	Yes	92.33%

- **Wine** Dataset (Number of validation data: 30)

#Trees	#Training Data	Attribute Bagging	Extremely Random forest	Accuracy
1	10	Yes	No	67.33%
1	10	No	No	64.67%
1	10	No	Yes	60.33%
5	10	Yes	No	73.33%
5	10	No	No	76.33%
5	10	No	Yes	71.00%
10	10	Yes	No	81.67%
10	10	No	No	81.00%
10	10	No	Yes	69.33%
1	30	Yes	No	81.00%
1	30	No	No	78.67%
1	30	No	Yes	71.33%
5	30	Yes	No	87.67%
5	30	No	No	83.67%
5	30	No	Yes	74.00%
10	30	Yes	No	89.00%
10	30	No	No	88.67%
10	30	No	Yes	78.00%
1	60	Yes	No	85.33%
1	60	No	No	82.67%
1	60	No	Yes	74.33%
5	60	Yes	No	89.67%
5	60	No	No	86.33%
5	60	No	Yes	79.33%
10	60	Yes	No	90.33%
10	60	No	No	89.67%
10	60	No	Yes	81.00%

2. Observations, and Interpretations

a. Relative sizes of the training and validation subsets

In both two datasets, the size of validation data is fixed. As the size of training data grows, the accuracy also grows. This is reasonable since with more training data, the tree will consider more situations and can classify the validation data better. The two datasets are very simple, so the performance are not very bad even with only 10 training data. It is expected that the increase of accuracy will slow down if I use more training data.

b. Number of trees in the forest

The number of trees affect the performance of classification. If there is only one tree, the performance may be unstable since training data is randomly sampled from the data pool. If the trees in the forest are more enough, it is less likely that all the trees has poor performance. Through the mechanism of voting, the predicted label will be more accurate and stable.

c. Bagging

In my experiment setups, I always apply tree bagging and I test the performance of attribute bagging. From the two tables, we can observe the accuracy is higher with attribute bagging. I think the reason is that attribute bagging can be viewed as one kind of regularization. If we always consider all the attributes during node split, the tree may overfit the training data and cannot classify validation set very well. Randomly select only some attributes makes the forest more diverse.

d. Extremely Random forest

The performance of extremely random forest is worse than the setting with and without attribute bagging. I guess randomly pick one attribute may often select a bad attribute to split although there is some randomness in extremely random forest. Attribute bagging has enough attributes to select, but extremely random forest only has one, which is apparently not enough.

3. Things I have learned

I have implemented decision tree before, but the implementation is ID3 version. It is the first time that I implement CART and random forest. I have learned the difference between these decision tree versions and I also learn the influence of the settings in the experiments.

4. Remaining Questions and Future Investigations

Regarding the experiments, the below things I did not try in this assignment.

- Parameters used during tree induction, such as how many attributes to consider at each node splitting.
- Methods that limit a tree's size. Examples include the minimum number of samples per node, or an upper bound on the tree's depth.
- Comparisons of out-of-bag errors and validation-set errors.

In the future, I will try to implement these settings and analyze the influence.

Appendix: Source code

```
import sys
import random
import math

class Node:
    def __init__(self):
        self.attribute_index = None
        self.threshold = None
        self.label = None
        self.parent = None
        self.left = None
        self.right = None


def gini(data):
    if len(data) == 0:
        return 0

    tmp = sorted(data, key=lambda d: d[-1])
    pre_index = 0
    g = 1.0

    for i in range(len(tmp) - 1):
        if tmp[i][-1] != tmp[i + 1][-1]:
            g -= ((i - pre_index + 1) / len(tmp)) ** 2
            pre_index = i + 1

    g -= ((len(tmp) - pre_index) / len(tmp)) ** 2
    return g


def build_decision_tree(node, data, attribute_bagging, extremely_random_forest=False):
    # check if all data has the same label
    same_label_flag = True
    for d in data[1:]:
        if data[0][-1] != d[-1]:
            same_label_flag = False
            break

    if same_label_flag:
        node.label = data[0][-1]
        return

    best_attribute_index = None
    best_threshold = None
    best_impurity = float("inf")

    attributes = [i for i in range(len(data[0]) - 1)]
    if attribute_bagging:
        num_attribute = round(math.sqrt(len(data[0]) - 1))
```

```
    attributes = random.sample(attributes, num_attribute)

if extremely_random_forest:
    attributes = random.sample(attributes, 1)

# select attribute
for i in attributes:
    # print(f'attritube {i}')
    data.sort(key=lambda d: d[i])
    same_attribute_flag = True

    # calculate the best threshold for the minimum impurity
    for j in range(len(data) - 1):
        if data[j][i] != data[j + 1][i]:

            same_attribute_flag = False
            arr1 = data[:j+1]
            arr2 = data[j+1:]
            gini1 = gini(arr1)
            gini2 = gini(arr2)
            impurity = len(arr1) * gini1 + len(arr2) * gini2

            # print(f'index {j} : {impurity}')

            if impurity < best_impurity:
                best_attribute_index = i
                best_threshold = (data[j][i] + data[j + 1][i]) / 2
                best_impurity = impurity

    if same_attribute_flag:
        gini1 = gini(data)
        impurity = len(data) * gini1

        if impurity < best_impurity:
            best_attribute_index = i
            best_threshold = data[-1][i]
            best_impurity = impurity

node.attribute_index = best_attribute_index
node.threshold = best_threshold
sub_data1 = [d for d in data if d[best_attribute_index] <= best_threshold]
sub_data2 = [d for d in data if d[best_attribute_index] > best_threshold]

# build the subtree
if len(sub_data1) > 0:
    left_child = Node()
    left_child.parent = node
    node.left = left_child
    build_decision_tree(left_child, sub_data1, attribute_bagging)

if len(sub_data2) > 0:
    right_child = Node()
```

```
        right_child.parent = node
        node.right = right_child
        build_decision_tree(right_child, sub_data2, attribute_bagging)


def classify(node, data):
    # if len(data) == 0:
    #     return 0

    # # leaf node
    # if node.label != None:
    #     correct = 0
    #     for d in data:
    #         correct += int(node.label == d[-1])
    #     return correct

    # sub_data1 = [d for d in data if d[node.attribute_index] <= node.threshold]
    # sub_data2 = [d for d in data if d[node.attribute_index] > node.threshold]
    # return classify(node.left, sub_data1) + classify(node.right, sub_data2)

    if node.label != None:
        return node.label
    if data[node.attribute_index] <= node.threshold:
        return classify(node.left, data)
    else:
        return classify(node.right, data)


def random_forest(data, tree_bagging, attribute_bagging):
    total_accuracy = 0.0

    for _ in range(10):
        training_data_pool = data[:-30]
        validation_data = data[-30:]

        num_tree = 5
        roots = []
        for i in range(num_tree):
            training_data = random.sample(training_data_pool, 60) if tree_bagging else data[:-
30]
            root = Node()
            build_decision_tree(root, training_data, attribute_bagging)
            roots.append(root)

        correct = 0
        for d in validation_data:
            # pred = {
            #     'Iris-setosa': 0,
            #     'Iris-versicolor': 0,
            #     'Iris-virginica': 0
            # }
            pred = {
```

```
                '1': 0,
                '2': 0,
                '3': 0
            }
            for i in range(num_tree):
                pred[classify(roots[0], d)] += 1

            pred_label = ''
            best_value = -1
            for k, v in pred.items():
                # print(k, v)
                if v > best_value:
                    pred_label = k
                    best_value = v

            # print(pred_label, classify(roots[0], d))
            true_label = d[-1]
            correct += int(pred_label == true_label)

        # print(correct / len(validation_data))
        total_accuracy += correct / len(validation_data)
    print(f'Total Accuracy: {total_accuracy / 10}')


if __name__ == '__main__':
    if len(sys.argv) != 2:
        print('Usage: python main.py [filename]')
        sys.exit()

    data = []
    filename = sys.argv[1]
    with open(filename, 'r') as fp:
        lines = fp.readlines()
        for line in lines:
            arr = line[:-1].split(',')
            arr.append(arr[0])
            arr = arr[1:]
            for i in range(len(arr) - 1):
                arr[i] = float(arr[i])
            data.append(tuple(arr))

    # print(data)
    random.shuffle(data)

    # random_forest(data, True, True)
    random_forest(data, True, False)
```