Introduction to AI

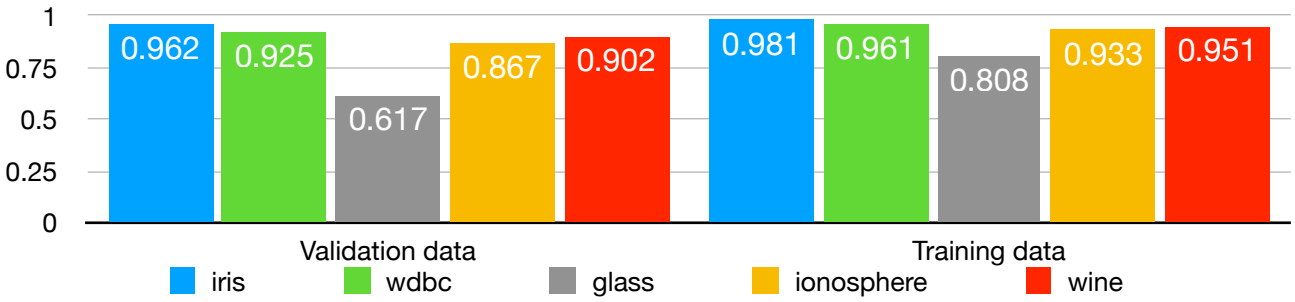
Programming Assignment #4

實驗與結果

Datasets 比較

表 1 為使用 Decision tree 對 Iris 、 Breast Cancer 、 Glass 、 Ionosphere 、 Wine 五種 Dataset 的 Validation data 和 Training data 進行測試的 Accuracy 結果。 Validation data 佔比為各項 Data 的 0.3 ，每項數據皆取 10 次測試後的平均值。

表 1



從表 1 中可以看出，除了 Glass 以外的 Datasets 都有一定水準的 Accuracy ，推測是因為 Glass 的 Target 有 7 種，而資料共有 214 筆，因此可能因 Sample 不足而受到影響。

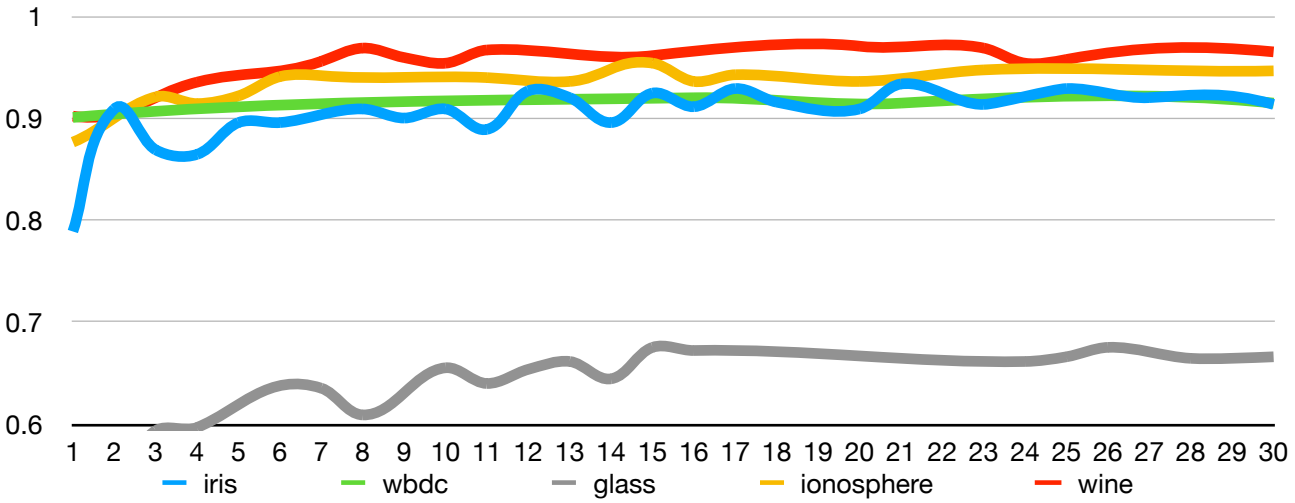
此外，Training data 進行測試的 Accuracy 結果都要較 Validation data 更高。

樹的數量比較

表 2 為使用 1 至 30 棵樹的 Random forest 對 5 種 Dataset 測試的 Accuracy 結果。每一棵樹所選用的 Attribute 數量為各項 Data 連續型 Attribute 數量的一半，Validation data 佔比為各項 Data 的 0.3 ，每項數據皆取 10 次測試後的平均值。

表 2 X 軸為每個 Random forest 中 Decision tree 的數量。

表 2



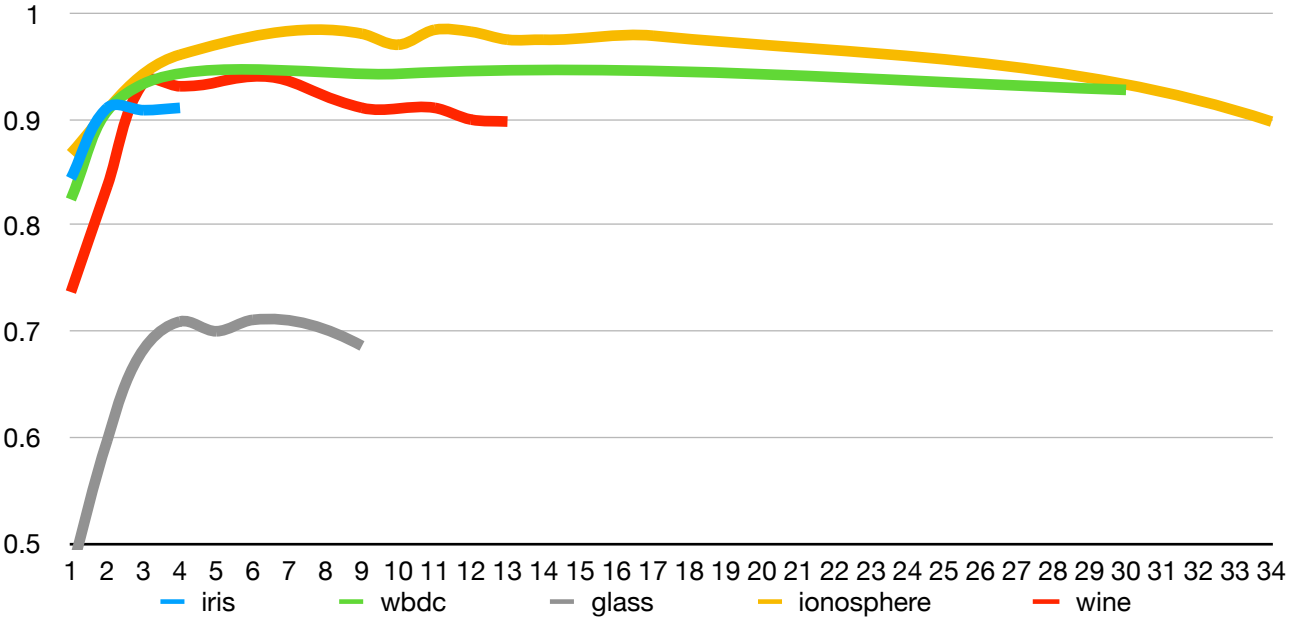
從表 2 中可以看出，當其他條件不變，當 Random forest 中的 Decision tree 數量越多，各個 Dataset 的 Accuracy 亦隨之升高，且其 Accuracy 都要比表 1 中使用 Decision tree 的結果要高。

Attributes 數量比較

表 3 比較每棵樹中選用的 Attributes 數量對 Dataset 測試的 Accuracy 結果。每次皆使用 10 棵樹的 Random Forest ， Validation data 佔比為各項 Data 的 0.3 ，每項數據皆取 10 次測試後的平均值。

表 3 X 軸為選用的 Attributes 數量。

表 3



由於每一種 Datasets 可用的 Attributes 數量不同，因此各線條投影至 X 軸的長度不一，測試中依序使用 1 個 Attribute 至該 Datasets 的所有 Attributes 。

從表 3 中可以看出，最佳的 Accuracy 約是落在選用一半的 Attribute 時，如果使用多於一半的 Attribute 時， Accuracy 的提升開始趨緩，甚至將會開始微微下降，而最差的 Accuracy 約是落在選極少數的 Attributes 時， 在 Glass 的測試中，若只選用 1 個 Attribute ，其 Accuracy 甚至不足 0.5 。

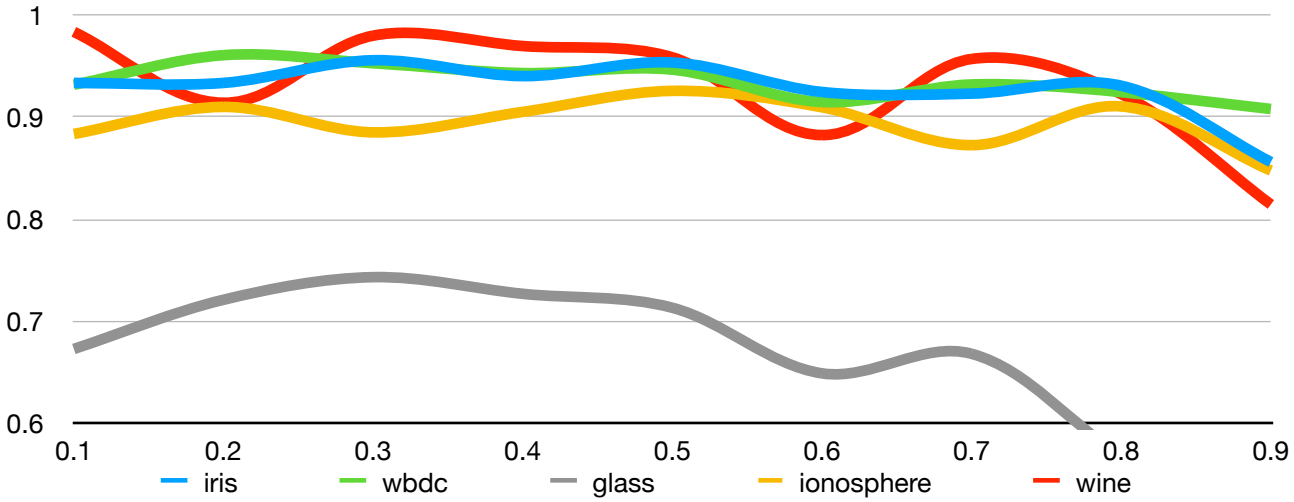
其中令我感到相當意外的是，除了 Glass 之外，其他的 Datasets 即便只有選用 1 個 Attribute ，其 Accuracy 大約也落在 0.8 以上。另外，經由這個實驗，也可以看出並非只要選擇更多的 Attributes 就能使 Accuracy 越來越高。

訓練資料與測試資料比例比較

表 4 比較各種不同的 Test data 佔比對 Dataset 測試的 Accuracy 結果。每次皆使用 10 棵樹的 Random Forest，每一棵樹所選用的 Attribute 數量為各項 Data 連續型 Attribute 數量的一半，每項數據皆取 10 次測試後的平均值。

表 4 X 軸為 Validation data 佔所有 Dataset 的比例。

表 4



從表 4 中可以看出，當 Test data 越少，亦即當 Train data 越多時，各個 Dataset 的 Accuracy 越高，我認為這樣的結果想相當合理。

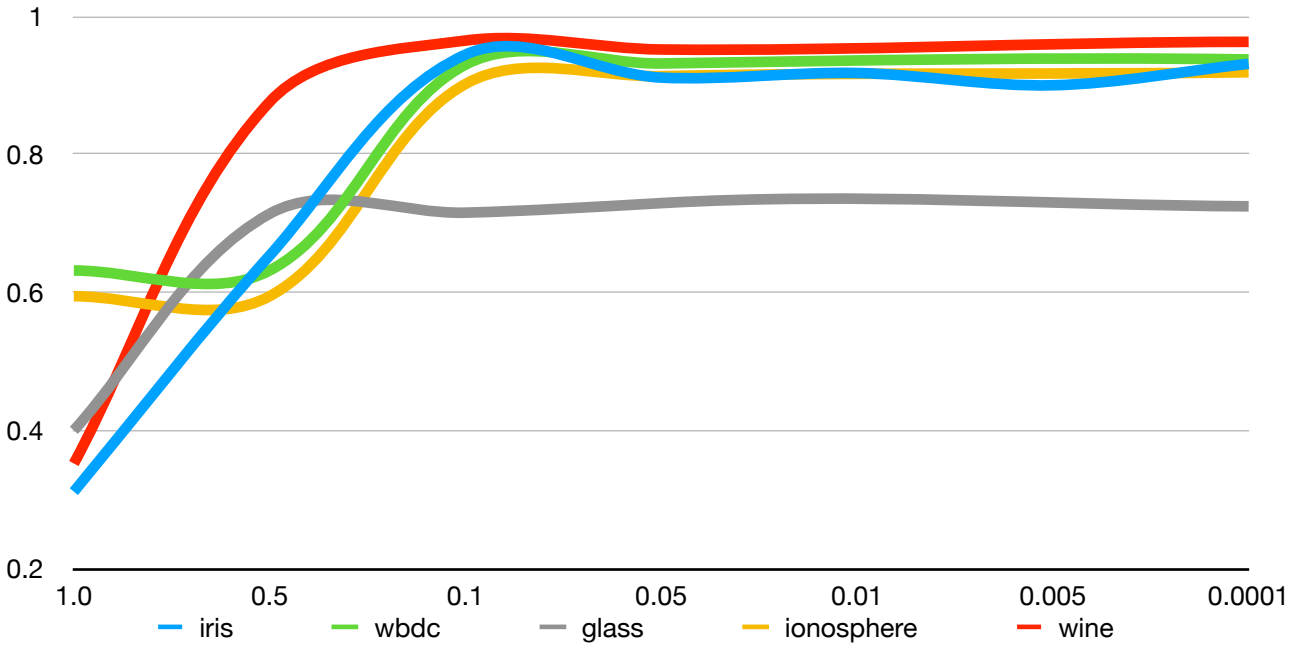
Total impurity 大小比較

在 Decision tree 中，決定一個 Node 有否為 Leaf node 的條件是判斷其 Total impurity 是否小於 Total value limit ， Total value limit 預設為 0.01 。

表 5 比較各種不同的 Total value limit 對 Dataset 測試的 Accuracy 結果。每次皆使用 10 棵樹的 Random Forest ，每一棵樹所選用的 Attribute 數量為各項 Data 連續型 Attribute 數量的一半，每項數據皆取 10 次測試後的平均值。

表 5 X 軸為 Total value limit。

表 5

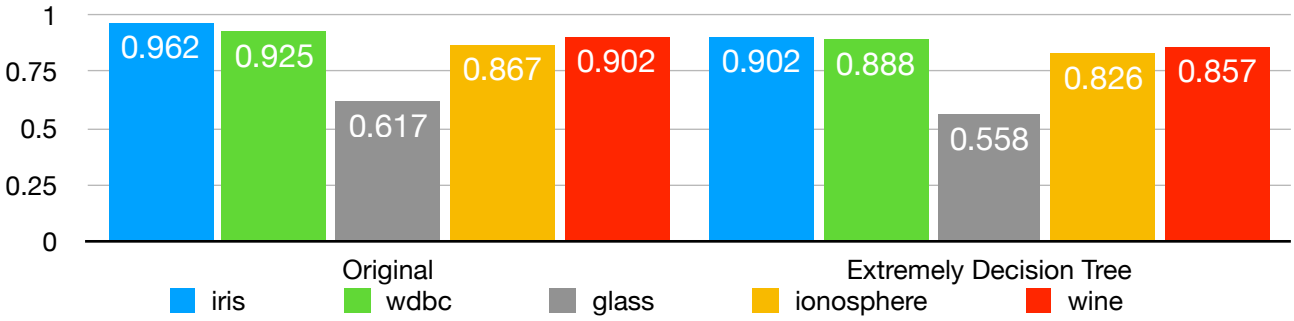


從表 5 中可以看出，當所選擇的 Total value limit 越小，各個 Dataset 的 Accuracy 越高，大約在 Total value limit 小於 0.1 後， Accuracy 的變化逐漸趨緩。

Extremely random forest

在 Extremely random forest 當中，每個 Node 隨機選擇一個 Attribute 作為 Threshold。表 6 「右方」為使用此方法的 Decision tree 對 5 種 Dataset 進行測試的 Accuracy 結果。表 6 「左方」為表 1 Validation data 數據，最為對照組參考。

表 6



從表 6 可以看出，使用此方法的 Decision tree ，其 Accuracy 結果相較正常方式稍差。

表 7 「深色粗線」為使用 1 至 30 棵樹的 Extremely Random forest 對 4 種 Dataset 測試的 Accuracy 結果。「淺色細線」為表 2 資料，最為對照組參考。

表 7

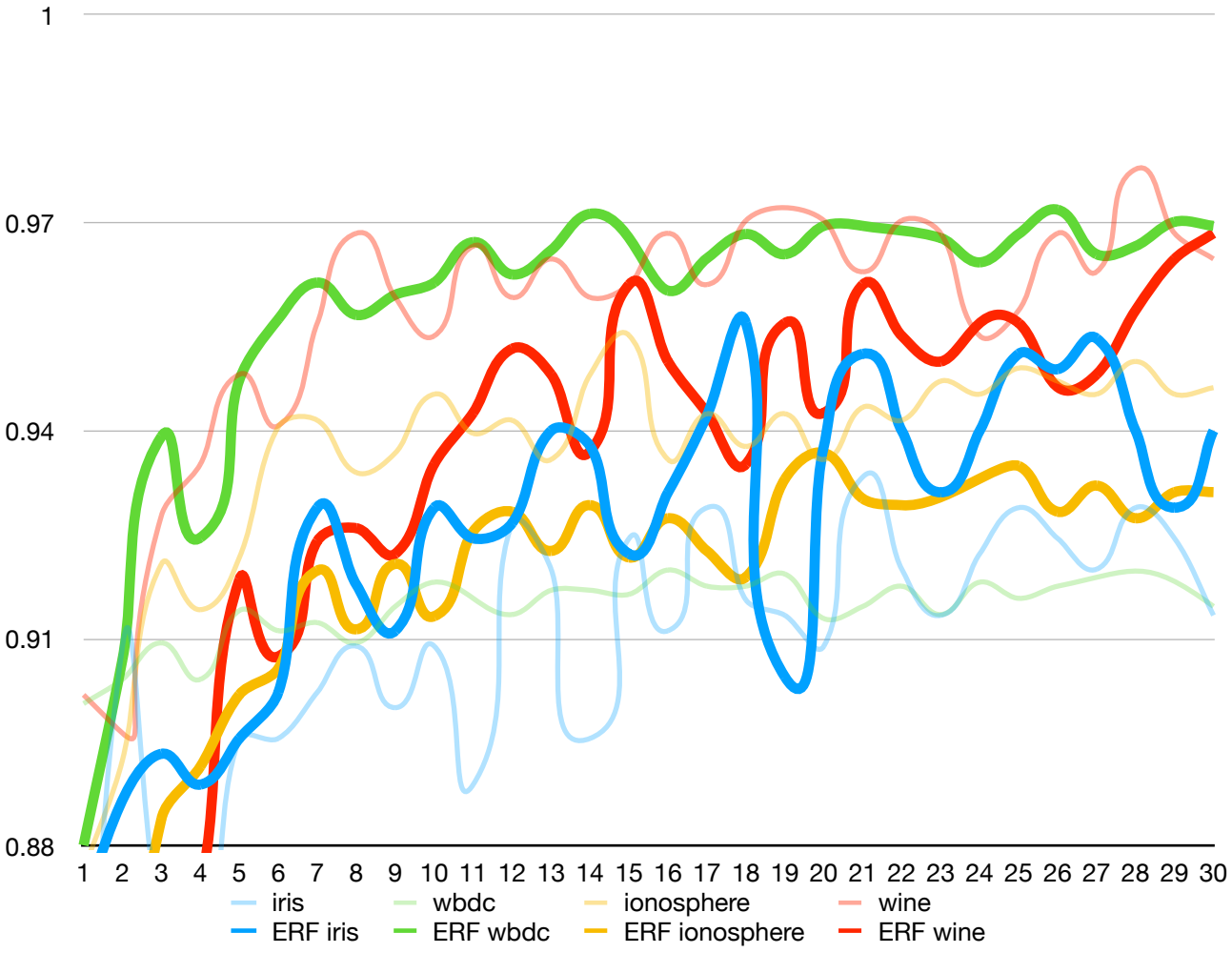


表 7 中的 ERF 代表 Extremely Random Forest 。

從表 7 可以大致看出，使用 Extremely random forest 的 Iris 和 Breast Cancer 其 Accuracy 相較原本的方式更佳，而 Ionosphere 和 Wine 則較差。其中 Breast Cancer 的效果在使用 Extremely random forest 有了最為顯著的提升。

此外，值得一提的是， Extremely random forest 建立樹時似乎要比傳統方式要更加快速，推測是因其不需遞迴檢查各種 Attributes ，而是隨機做出選擇，這個特性對於有較多 Attributes 的資料在訓練的處理速度上有著顯著的提升。

所學

透過本次實驗我所學以下幾點：

1. 利用物件導向程式設計，以 Python 實作 Random forest。
2. 了解 Decision tree 、 Random forest 、 Gini index 的運作原理、特性以及優缺點。
3. 透過比較不同的參數對於演算法效能的影響。
4. 將實驗結果整理、列表並使人易於閱讀。

疑問與探討

1. 在 Python 的實作當中，如果我將 Total value limit 設得極小，有時會出現遞迴次數過多的錯誤，後來發現是因為樹的深度過長的原因所導致，因此，比需要想辦法去限制樹的深度，我認為有以下幾種方式可以限制樹的深度：
    1. 直接限制樹的深度，當節點的深度到達深度最大上限時，即便尚未分類完成，也需要迫使其成為葉節點，做出選擇。
    2. 限制每個 Attribute 所能被選為 Threshold 的次數，如此以來，有限個 Attributes 將使樹的深度亦為有限。
    3. 增加 Total value limit。因為在任何 Datasets 中，雜訊勢必存在，Total value limit 設得過小將會導致許多節點被雜訊干擾而進行無意義的分類。
2. 從表 3 中可以看出，並不是選取的 Attributes 越多，結果就越好，而是當選取的 Attributes 數量約略在所有 Attributes 數量的一半時，其 Accuracy 最佳，我認為這是因為當不使用過多的 Attributes 時，可以有效的過濾掉部分雜訊。
3. 當 Target 為連續資料時，以目前的 Random forest 無法處理，可能需要積分方式處理。

未來發想

本次作業規定使用 CART ，因此所產生的樹皆為二元樹，亦即每個節點只選一個 Threshold 和一個 Partition value ，未來我會想要實作在一個節點中，選擇多個 Partition value 的方法，並和目前的作法比較效率及 Accuracy 上的差別。另外，spec 中有說明只會使用 real-valued attributes ，因此對於 Categorical Attributes 並不需要處理，未來我會想要加上 Categorical Attributes 的分類，觀察結果是否有能顯著的改變。我也會想要比較使用 Gini index 和 Entropy 在衡量資料分類上的差異。

Appendix

Structure

- node.py
- tree.py
- hw4.py

node.py

```
import random


class Node():
    def __init__(self, data, target_attr, selected_attrs, depth=0,
total_value_limit=0.01):
        self.action = None
        self.threshold = None
        self.threshold_value = None
        self.prediction = None
        self.data = data
        self.target_attr = target_attr
        self.selected_attrs = selected_attrs
        self.lt_value_child = None
        self.ge_value_child = None
        self.depth = depth
        self.total_value_limit = total_value_limit

        total_value = self.gini_index(data=self.data)
        if total_value < total_value_limit or depth > 100:
            # Decision node
            self.action = 'predict'
            self.prediction = self.make_prediction()
            self.data = None
        else:
            # Spilt node
            self.split(total_value)

    def __str__(self):
        ret = self.action + '\n'
        ret += str(self.depth) + '\n'
        ret += str(self.prediction) + '\n'
        ret += str(self.threshold) + '\n'
        ret += str(self.threshold_value) + '\n'
        return ret
```

```
def stats(self, data):
    # Calculate counts of each target appears in data
    stats = {}
    for row in data:
        if row[self.target_attr] in stats:
            stats[ row[self.target_attr] ] += 1
        else:
            stats[ row[self.target_attr] ] = 1
    return stats

def make_prediction(self):
    prediction = None

    # Check which target appears the most in data
    stats = self.stats(self.data)
    max_count = -1
    for target in stats:
        if stats[target] > max_count:
            prediction = target
            max_count = stats[target]

    return prediction

def gini_index(self, data):
    data_count = len(data)
    if data_count == 0:
        return 0

    stats = self.stats(data)
    probabilities = []
    for target in stats:
        probabilities.append(stats[target] / data_count)

    total_power_of_probabilities = 0
    for p in probabilities:
        total_power_of_probabilities += p * p
    gini_index = 1 - total_power_of_probabilities

    return gini_index

def remainder(self, key, value):
    lt_value_data, ge_value_data = self.split_data_by_value(key, value)
```

```
        remainder = (self.gini_index(data=lt_value_data) * len(lt_value_data)
+
                     self.gini_index(data=ge_value_data) *
len(ge_value_data)) / len(self.data)

        return remainder

    def split_data_by_value(self, key, value):
        lt_value_data = []
        ge_value_data = []
        for row in self.data:
            if row[key] < value:
                lt_value_data.append(row)
            else:
                ge_value_data.append(row)
        return lt_value_data, ge_value_data

    def random_select_threshold(self, total_value):
        info_gain = None
        threshold = None
        threshold_value = None

        # Random select a threshold
        key = random.choice(self.selected_attrs)
        threshold = key

        values = []
        for row in self.data:
            values.append(row[key])
        values.sort()
        values.remove(values[0])

        # Select the best partition value
        for value in values:
            tmp_info_gain = total_value - self.remainder(key, value)
            if info_gain is None or tmp_info_gain > info_gain:
                info_gain = tmp_info_gain
                threshold_value = value

        return threshold, threshold_value

    def select_threshold(self, total_value):
```

```
        info_gain = None
        threshold = None
        threshold_value = None

        # Select the best threshold and partition value
        for key in self.selected_attrs:
            values = []
            for row in self.data:
                values.append(row[key])
            values.sort()
            values.remove(values[0])

            for value in values:
                tmp_info_gain = total_value - self.remainder(key, value)
                if threshold is None or tmp_info_gain > info_gain:
                    info_gain = tmp_info_gain
                    threshold = key
                    threshold_value = value

        return threshold, threshold_value

    def split(self, total_value):
        # Select threshold and partition value
        threshold, threshold_value = self.select_threshold(total_value)
        # threshold, threshold_value =
self.random_select_threshold(total_value)

        # Split the data for child node
        lt_value_data, ge_value_data = self.split_data_by_value(threshold,
threshold_value)
        self.data = None

        # Define action
        self.action = 'catagorize'

        # Create child nodes
        self.threshold = threshold
        self.threshold_value = threshold_value
        self.lt_value_child = Node(lt_value_data, self.target_attr,
self.selected_attrs, self.depth+1, self.total_value_limit)
        self.ge_value_child = Node(ge_value_data, self.target_attr,
self.selected_attrs, self.depth+1, self.total_value_limit)
```

```
    def visit(self, test_data_item):
        # When test data visits
        if self.action == 'predict':
            return self.prediction
        else:
            if test_data_item[self.threshold] < self.threshold_value:
                return self.lt_value_child.visit(test_data_item)
            else:
                return self.ge_value_child.visit(test_data_item)
```

tree.py

```
from node import Node

class Tree():
    def __init__(self, train_data, target_attr, selected_attrs,
total_value_limit = 0.01):
        self.root = Node(train_data, target_attr, selected_attrs, 0,
total_value_limit)

    def predict(self, test_data_item):
        return self.root.visit(test_data_item)
```

hw4.py

```
import csv, random, statistics
from tree import Tree

datasets = {'iris':         {'target_attr': 4, 'valid_attrs': [i for i in
range(0, 4)]},
            'wdbc':         {'target_attr': 1, 'valid_attrs': [i for i in
range(2, 32)]},
            'glass':        {'target_attr': 10, 'valid_attrs': [i for i in
range(1, 10)]},
            'ionosphere':   {'target_attr': 34, 'valid_attrs': [i for i in
range(0, 34)]},
            'wine':         {'target_attr': 0, 'valid_attrs': [i for i in
range(1, 14)]},
           }

def data_reader(dataset_name):
    dataset = datasets[dataset_name]
    target_attr = dataset['target_attr']
    valid_attrs = dataset['valid_attrs']
```

```
    data = []
    data_path = 'datasets/' + dataset_name + '.data'
    with open(data_path, newline = '') as csvfile:
        file_rows = list(csv.reader(csvfile))

        for file_row in file_rows:
            if not file_row:
                continue
            row = []
            for i in range(len(file_row)):
                if i == target_attr or i not in valid_attrs:
                    row.append(file_row[i])
                else:
                    row.append(float(file_row[i]))
            data.append(row)
    return data, target_attr, valid_attrs


def data_processor(data, test_data_proportion):
    random.shuffle(data)
    train_data_count = int(len(data) * (1-test_data_proportion))
    train_data = data[:train_data_count]
    test_data = data[train_data_count:]
    return train_data, test_data


def build_forest(tree_count, selected_attrs_count, train_data, target_attr,
valid_attrs, total_value_limit, print_mode=False):
    forest = []

    for _ in range(tree_count):
        selected_attrs = random.sample(valid_attrs, selected_attrs_count)
        if print_mode:
            print('Building Tree', _+1, selected_attrs)
        forest.append( Tree(train_data, target_attr, selected_attrs,
total_value_limit) )

    return forest


def test(test_data, target_attr, forest, print_mode=False):
    if print_mode:
```

```
        print('Testing')
    correct_count = 0

    for td in test_data:
        votes = {}

        for tree in forest:
            prediction = tree.predict(td)
            if prediction in votes:
                votes[ prediction ] += 1
            else:
                votes[ prediction ] = 1

        final_decision = None
        max_vote = -1
        for prediction in votes:
            if votes[ prediction ] > max_vote:
                final_decision = prediction
                max_vote = votes[ prediction ]

        answer = td[target_attr]
        if print_mode:
            print(final_decision == answer, final_decision, answer, votes)
        if final_decision == answer:
            correct_count += 1

    if print_mode:
        print(correct_count/len(test_data))

    return correct_count / len(test_data)


def simple_test(tree_count, selected_attrs_count, train_data, test_data,
target_attr, valid_attrs, total_value_limit=0.01, print_mode=False):
    forest = build_forest(tree_count, selected_attrs_count, train_data,
target_attr, valid_attrs, total_value_limit, False)
    accuracy = test(test_data, target_attr, forest, False)
    if print_mode:
        print('{}\t{}\t{}\t{}'.format(tree_count, selected_attrs_count,
total_value_limit, accuracy))
    return accuracy
```

```
def example_test():
    dataset_name = 'wine'
    test_data_proportion = 0.3
    total_value_limit = 0.01

    dataset, target_attr, valid_attrs = data_reader(dataset_name)
    train_data, test_data = data_processor(dataset, test_data_proportion)

    print('==== Example Test ====')
    print('Dataset:\t', dataset_name)
    print()
    print('Trees\tAttrs\tLimit\tAccuracy')
    simple_test(10, 7, train_data, test_data, target_attr, valid_attrs,
total_value_limit, True)


def datasets_test(test_count=10):
    test_datasets = ['iris', 'wdbc', 'glass', 'ionosphere', 'wine']
    test_data_proportion = 0.3
    total_value_limit = 0.01

    print()
    print('==== Datasets Test ====')
    print('test_count:\t\t', test_count)
    print('test_data_proportion:\t', test_data_proportion)
    print('total_value_limit:\t', total_value_limit)
    print('trees_count:\t\t', 1)
    print()
    print('Dataset\tAccuracy')

    for dataset_name in test_datasets:
        dataset, target_attr, valid_attrs = data_reader(dataset_name)

        accuracies = []
        for _ in range(test_count):
            train_data, test_data = data_processor(dataset,
test_data_proportion)
            accuracy = simple_test(1, len(valid_attrs), train_data, test_data,
target_attr, valid_attrs, total_value_limit)
            accuracies.append(accuracy)
        print('{}\t{}'.format(dataset_name, statistics.mean(accuracies)))
        for _ in range(test_count):
```

```
            train_data, test_data = data_processor(dataset,
test_data_proportion)
            accuracy = simple_test(1, len(valid_attrs), train_data,
train_data, target_attr, valid_attrs, total_value_limit)
            accuracies.append(accuracy)
        print('{}\t{}'.format(dataset_name, statistics.mean(accuracies)))


def trees_count_test(dataset_name, max_trees_count, selected_attrs_count=None,
test_count=10):
    test_data_proportion = 0.3
    total_value_limit = 0.01

    dataset, target_attr, valid_attrs = data_reader(dataset_name)
    train_data, test_data = data_processor(dataset, test_data_proportion)
    selected_attrs_count = selected_attrs_count if selected_attrs_count else
int(len(valid_attrs)/2)

    print()
    print('==== Trees Count Test ====')
    print('Dataset:\t\t', dataset_name)
    print('test_count:\t\t', test_count)
    print('test_data_proportion:\t', test_data_proportion)
    print('total_value_limit:\t', total_value_limit)
    print('selected_attrs_count:\t', selected_attrs_count)
    print()
    print('Trees\tAccuracy')

    for trees_count in range(1, max_trees_count+1):
        accuracies = []
        for _ in range(test_count):
            accuracy = simple_test(trees_count, selected_attrs_count,
train_data, test_data, target_attr, valid_attrs, total_value_limit)
            accuracies.append(accuracy)
        print('{}\t{}'.format(trees_count, statistics.mean(accuracies)))


def attrs_count_test(dataset_name, trees_count, test_count=10):
    test_data_proportion = 0.3
    total_value_limit = 0.01

    dataset, target_attr, valid_attrs = data_reader(dataset_name)
    train_data, test_data = data_processor(dataset, test_data_proportion)
```

```
    print()
    print('==== Attrs Count Test ====')
    print('Dataset:\t\t', dataset_name)
    print('test_count:\t\t', test_count)
    print('test_data_proportion:\t', test_data_proportion)
    print('total_value_limit:\t', total_value_limit)
    print('trees_count:\t\t', trees_count)
    print()
    print('Attrs\tAccuracy')

    for selected_attrs_count in range(1, len(valid_attrs)+1):
        accuracies = []
        for _ in range(test_count):
            accuracy = simple_test(trees_count, selected_attrs_count,
train_data, test_data, target_attr, valid_attrs, total_value_limit)
            accuracies.append(accuracy)
        print('{}\t{}'.format(selected_attrs_count,
statistics.mean(accuracies)))


def test_data_propotion_test(dataset_name, trees_count,
selected_attrs_count=None, test_count=10):
    total_value_limit = 0.01

    dataset, target_attr, valid_attrs = data_reader(dataset_name)
    selected_attrs_count = selected_attrs_count if selected_attrs_count else
int(len(valid_attrs)/2)

    print()
    print('==== Test Data Propotion Test ====')
    print('Dataset:\t\t', dataset_name)
    print('test_count:\t\t', test_count)
    print('total_value_limit:\t', total_value_limit)
    print('trees_count:\t\t', trees_count)
    print('selected_attrs_count:\t', selected_attrs_count)
    print()
    print('test/all\tAccuracy')

    for i in range(1, 10):
        test_data_proportion = i / 10
        train_data, test_data = data_processor(dataset, test_data_proportion)
```

```
        accuracies = []
        for _ in range(test_count):
            accuracy = simple_test(trees_count, selected_attrs_count,
train_data, test_data, target_attr, valid_attrs, total_value_limit)
            accuracies.append(accuracy)
        print('{}\t{}'.format(test_data_proportion,
statistics.mean(accuracies)))


def total_value_limit_test(dataset_name, trees_count,
selected_attrs_count=None, test_count=10):
    total_value_limits = [1.0, 0.5, 0.1, 0.05, 0.01, 0.005, 0.0001]
    test_data_proportion = 0.3

    dataset, target_attr, valid_attrs = data_reader(dataset_name)
    train_data, test_data = data_processor(dataset, test_data_proportion)
    selected_attrs_count = selected_attrs_count if selected_attrs_count else
int(len(valid_attrs)/2)

    print()
    print('==== Total Value Limit Test ====')
    print('Dataset:\t\t', dataset_name)
    print('test_count:\t\t', test_count)
    print('test_data_proportion:\t', test_data_proportion)
    print('trees_count:\t\t', trees_count)
    print('selected_attrs_count:\t', selected_attrs_count)
    print()
    print('Limit\tAccuracy')

    for total_value_limit in total_value_limits:
        accuracies = []
        for _ in range(test_count):
            accuracy = simple_test(trees_count, selected_attrs_count,
train_data, test_data, target_attr, valid_attrs, total_value_limit)
            accuracies.append(accuracy)
        print('{}\t{}'.format(total_value_limit, statistics.mean(accuracies)))


def main():
    example_test()
    print('\n++++++++++++++++++++++++++++\n')

    datasets_test()
```

```
    print('\n++++++++++++++++++++++++++++\n')

    test_datasets = ['iris', 'wdbc', 'glass', 'ionosphere', 'wine']

    for dataset in test_datasets:
        trees_count_test(dataset, 30)
        print()
    print('\n++++++++++++++++++++++++++++\n')


    for dataset in test_datasets:
        attrs_count_test(dataset, 10)
        print()
    print('\n++++++++++++++++++++++++++++\n')

    for dataset in test_datasets:
        test_data_propotion_test(dataset, 10)
        print()
    print('\n++++++++++++++++++++++++++++\n')

    for dataset in test_datasets:
        total_value_limit_test(dataset, 10)
        print()
    print('\n++++++++++++++++++++++++++++\n')


if __name__ == '__main__':
    main()
```