HW4 Report

0516034 楊翔鈞

#1 作業介紹

本次作業要實作 decision tree 與 random forest，並比較不同情況下（用於訓練的 attribute 數量、不同比例的 training sample、decision tree 的深度等 ）的 random forest 之間的表現差異，選用的 dataset 為以下三組：

- 2-class breast cancer
- 3-class iris
- 3-class wine

為了方便讀取檔案，我有修改 dataset 的 feature 順序，讓每行的最後一個值是該筆資料的 label（修改過後的檔案在這裡）。

#2 實驗

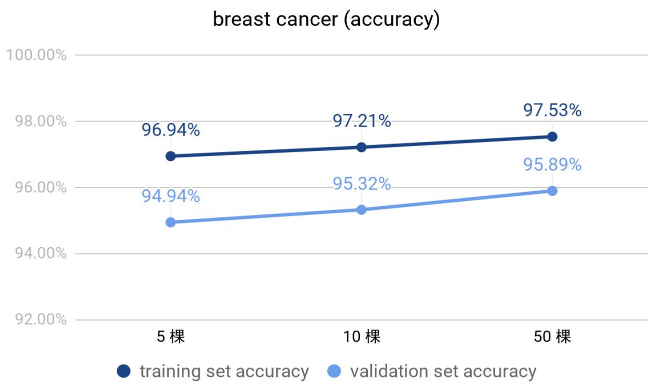
- decision tree 的最大深度：一層、三層、五層、無限制
- random forest 中樹的數量：五棵、十棵、五十棵
- 用於訓練的 attribute 數量（n 為原始資料的 attribute 數量）：sqrt(n)、log2(n)、0.1*n、0.3*n、0.5*n、0.7*n、n

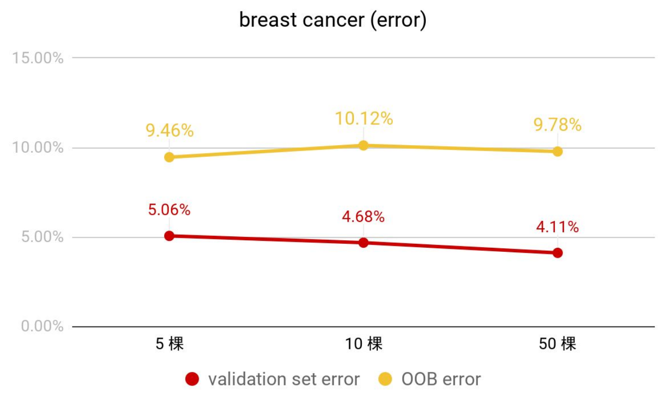
以上三個變數總共可產生 84 種組合，三組 dataset 都會跑這 84 種組合，下面會分別以這三種變數去觀察結果。（所有實驗結果 - Google Spreadsheet 連結）

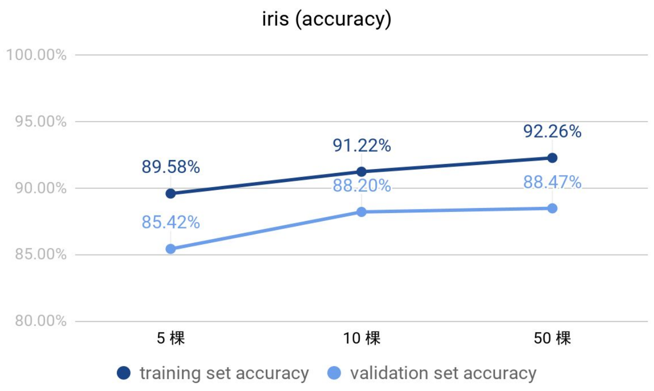
#2.1 random forest 的大小（幾棵 decision tree）

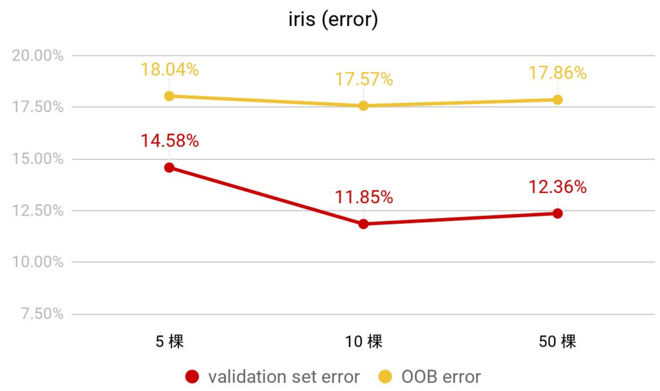
因為這邊只討論不同的 forest 大小，圖表中的數據都是把另外兩種變數跑出來的結果加總平均得到的，往後的幾組比較也是一樣，討論其中一個變數，將其餘兩個變數跑出來的結果加總平均。

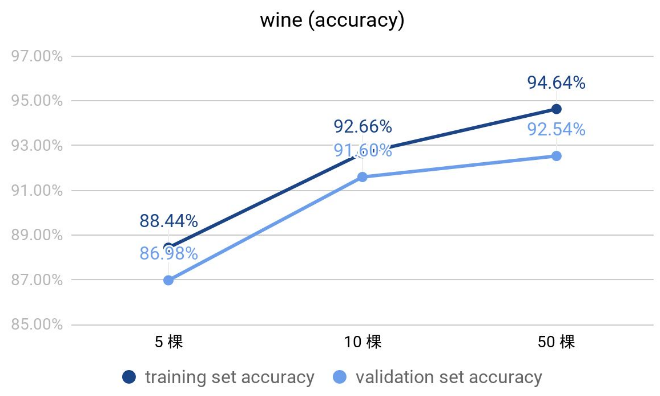
左側的折線圖是 training set 與 validation set 的 correct classification rate；右側則是 validation set error 與 out of bag error，由上而下依序為 breast cancer、iris、wine 三組 dataset。

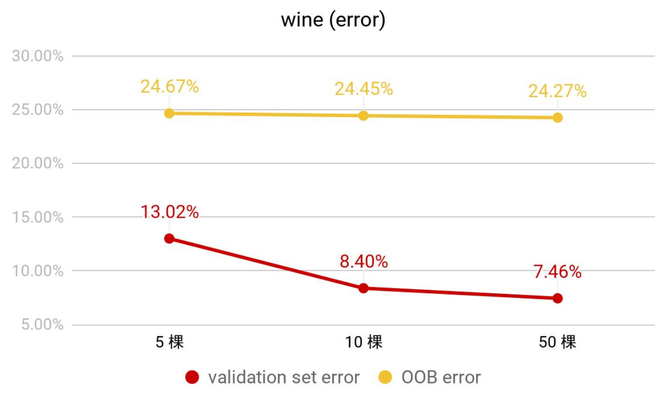






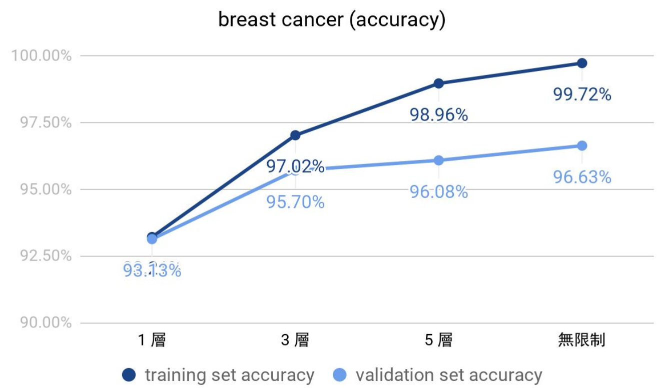


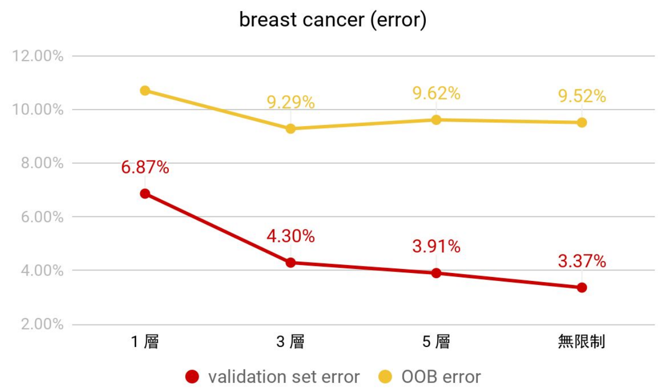


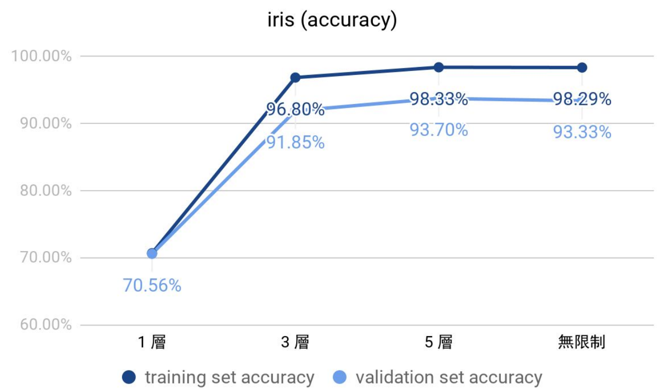


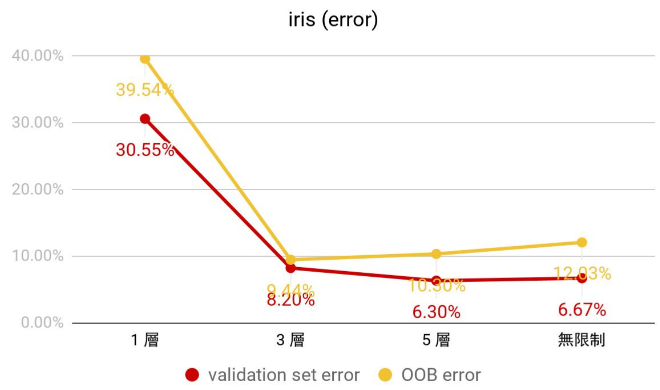
#2.2 decision tree 的最大深度

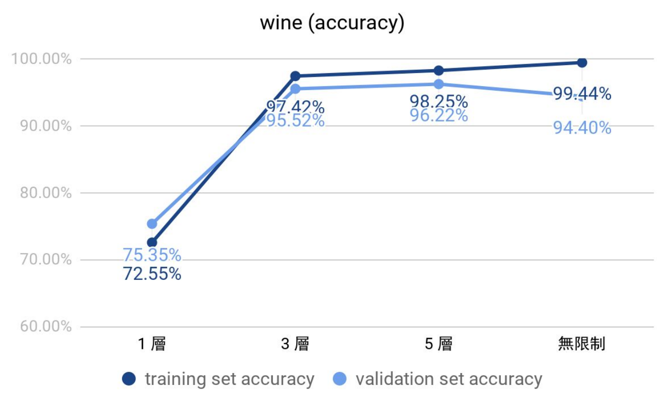
圖表編排方式同上。

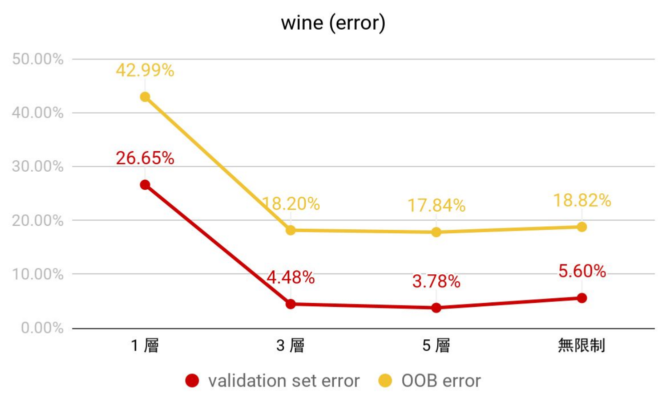






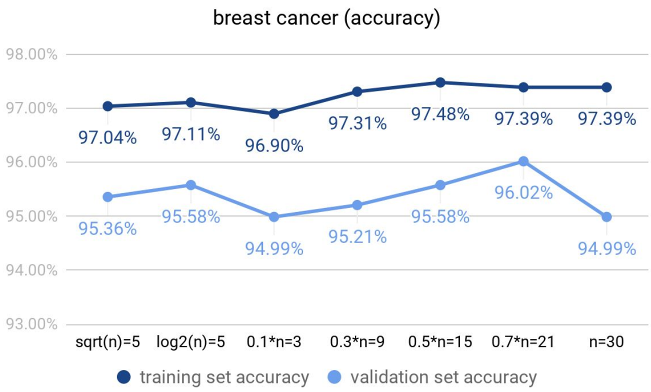


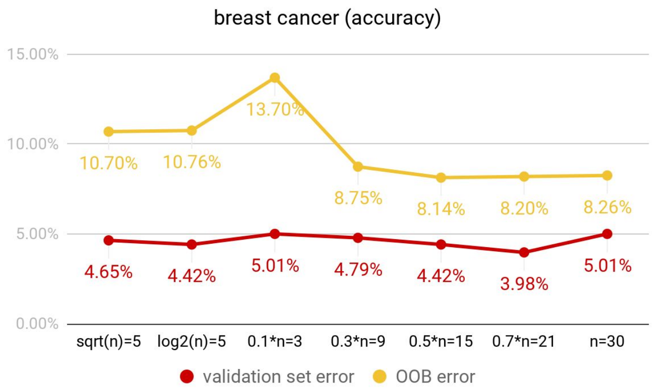


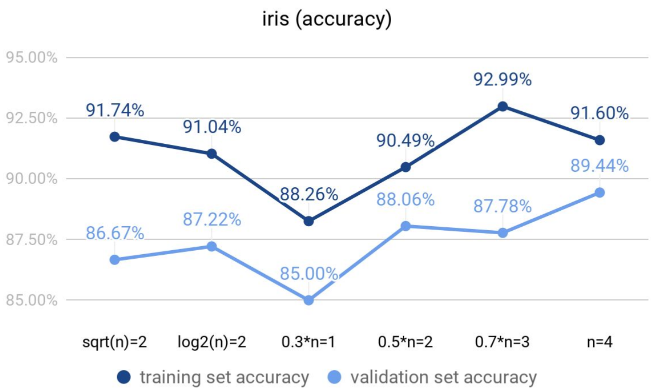


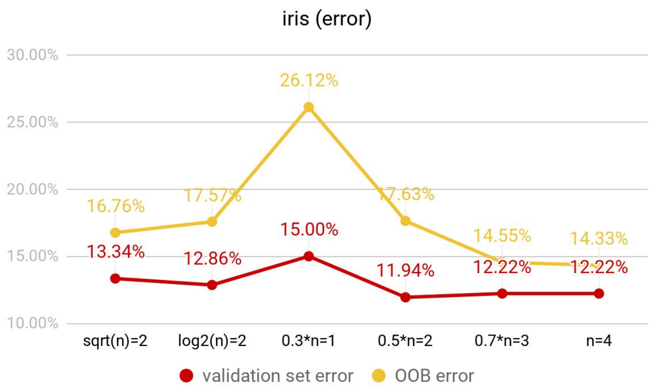
#2.3 用於訓練的 attribute 數量（n 為 attribute 總數）

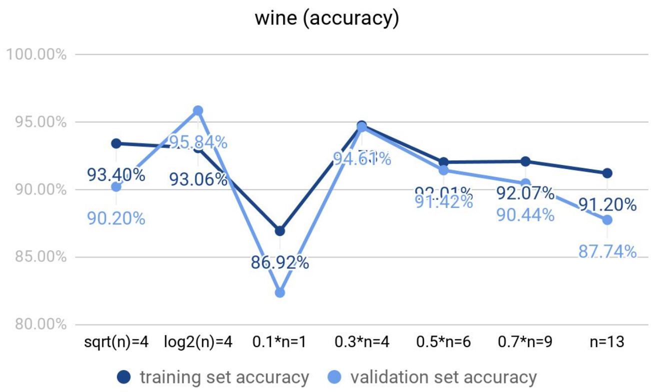
圖表編排方式同上。

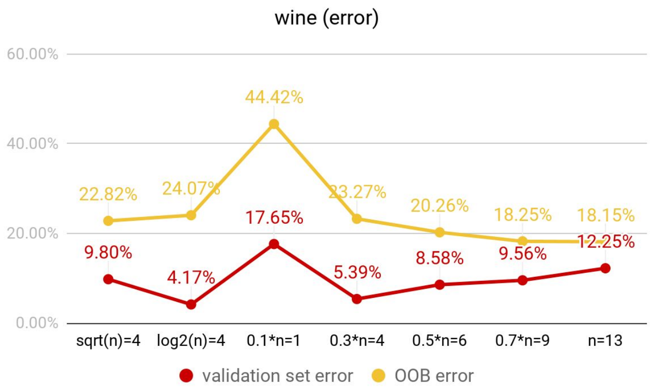












#3 觀察與想法

- forest 內的 decision tree 越多，模型的 correct classification rate 就越高，因為 prediction 是根據每棵 decision tree 的判斷結果去投票，越多 decision tree 就代表越多不同的意見，較多的意見對於決策也有幫助。
- decision tree 的深度越深，training set 的 correct classification rate 就越高，因為深度越深代表 training data 會被分得越細，decision tree 就會盡量讓所有 training data 都被正確歸類，但深度越深同時也會有 overfitting 的風險。
- 當用於訓練的 attribute 數量占原 attribute 數量的 10%~30% 時，模型的表現明顯下降，但是當用於訓練的 attribute 數量超過 sqrt(n) 的時候，模型的表現就基本上維持在一個水準，而沒有太大的起伏。同時我發現網路上很多 random forest 的套件都將用於訓練的 attribute 數量預設為 sqrt(n)，從實驗結果中我們可以觀察到，其實 sqrt(n) 就已經很足夠了。
- random forest 的目的是要從不同的角度（attribute）去分類 data，然後統計每棵 decision tree 的分類結，如果用於訓練的 attribute 數量太多，那每次用於 split 的 attribute 就都會是固定那幾個，所以比較好的方式應該是把用於訓練的 attribute 數量壓低，讓 decision tree 從有限的 attribute 中挑出能夠讓 data impurity 最小的。

- 如果要讓模型的表現有明顯的成長，就 #2 討論的三種變數來看，最快的方法應該是調整 decision tree 的最大深度，深度越大代表 data 會經過越多層的判斷，但同時又不能無止盡的加大，以免 overfitting 的情況發生。其次應該是增加 forest 內的 decision tree 數量。
- out of bag error 總是比 validation set error 大，out of bag error 的計算方式沒有經過投票，而 validation set error 是利用 forest 內的 decision tree 的投票結果去計算錯誤判斷率，所以 out of bag error 基本上一定會比 validation set error 大，這也呼應到 #2.1 的實驗，forest 內 decision tree 數量的比較。

```python
# coding: utf-8
import random
import numpy as np


def load_data(path):
    label = {}
    encode_y = 0
    X, y = [], []
    with open(path, 'r') as file:
        for line in file:
            if not line.strip():
                break
            t = line.strip().split(',')
            if label.get(t[-1]) is None:
                label[t[-1]] = encode_y
                encode_y += 1
            X.append(np.asarray(t[:-1]).astype(np.float64))
            y.append(label[t[-1]])
    X = np.asarray(X)
    y = np.asarray(y).reshape(-1, 1)
    return X, y, label


def split_dataset(X, y, train_ratio):
    def split_one_class(X, y, label):
        X_class = X[y.reshape(-1)==label]
        y_class = y[y.reshape(-1)==label]
        train_size = int(np.ceil(X_class.shape[0] * train_ratio))
        idx = np.arange(X_class.shape[0])
        train_idx_class = np.random.choice(idx, train_size,
replace=False)
        train_idx_class = np.sort(train_idx_class)
        mask = np.ma.array(idx, mask=False)
        mask.mask[train_idx_class] = True
        test_idx_class = mask.compressed()
        X_train = X_class[train_idx_class]
        X_test = X_class[test_idx_class]
        y_train = y_class[train_idx_class]
        y_test = y_class[test_idx_class]
        return X_train, X_test, y_train, y_test
    res = np.unique(y)
    X_train, y_train = None, None
    X_test, y_test = None, None
    for c in res:
        X_classC_train, X_classC_test, y_classC_train, y_classC_test =
split_one_class(X, y, c)
        if X_train is None:
            X_train = np.array(X_classC_train)
            y_train = np.array(y_classC_train)
            X_test = np.array(X_classC_test)
```

```
            y_test = np.array(y_classC_test)
        else:
            X_train = np.vstack((X_train, X_classC_train))
            y_train = np.vstack((y_train, y_classC_train))
            X_test = np.vstack((X_test, X_classC_test))
            y_test = np.vstack((y_test, y_classC_test))

    return X_train, y_train, X_test, y_test


# Gini's impurity
def gini(sequence):
    _, cnt = np.unique(sequence, return_counts=True)
    prob = cnt / sequence.shape[0]
    g = 1 - np.sum([p**2 for p in prob])
    return g


class DecisionTree():
    def __init__(self, max_depth=None):
        self.measure_func = gini
        self.max_depth = max_depth
        self.root = None
        return None

    class Node():
        def __init__(self):
            self.feature = None
            self.thres = None
            self.impurity = None
            self.data_num = None
            self.left = None
            self.right = None
            self.predict_class = None

    def get_thres(self, data):
        thres = None
        feature = None
        min_impurity = 1e10
        (n, dim) = data.shape
        dim -= 1
        for i in range(dim):
            data_sorted = np.asarray(sorted(data, key=lambda t: t[i]))
            for j in range(1, n):
                t = (data_sorted[j-1, i]+data_sorted[j, i]) / 2
                left_data = data_sorted[data_sorted[:, i]<t]
                right_data = data_sorted[data_sorted[:, i]>=t]
                left_impurity = self.measure_func(left_data[:,
-1].astype(np.int32))
                right_impurity = self.measure_func(right_data[:,
-1].astype(np.int32))
```

```python
            impurity = left_data.shape[0] * left_impurity
            impurity += right_data.shape[0] * right_impurity
            impurity /= data_sorted.shape[0]
            if impurity <= min_impurity:
                min_impurity = impurity
                thres = t
                feature = i
        return feature, thres, min_impurity

    def build_tree(self, data, depth=None):
        node = self.Node()
        if self.measure_func(data[:, -1].astype(np.int32)) == 0:
            node.predict_class = [int(data[0, -1])]
        elif depth == 0:
            label, cnt = np.unique(
                data[:, -1].astype(np.int32), return_counts=True)
            node.predict_class = list(label[cnt==np.max(cnt)])
        else:
            feature, thres, impurity = self.get_thres(data)
            left_data = data[data[:, feature]<thres]
            right_data = data[data[:, feature]>=thres]
            if left_data.shape[0]==0 or right_data.shape[0]==0:
                label, cnt = np.unique(
                    data[:, -1].astype(np.int32), return_counts=True)
                node.predict_class = list(label[cnt==np.max(cnt)])
            else:
                node.feature = feature
                node.thres = thres
                node.impurity = impurity
                node.data_num = data.shape[0]
                if depth is None:
                    node.left = self.build_tree(left_data)
                    node.right = self.build_tree(right_data)
                else:
                    node.left = self.build_tree(left_data, depth-1)
                    node.right = self.build_tree(right_data, depth-1)
        return node

    def train(self, X, y):
        data = np.hstack((X, y))
        self.root = self.build_tree(data, self.max_depth)

    def traverse(self, node, X):
        if node.predict_class is not None:
            if len(node.predict_class) > 1:
                return random.choice(node.predict_class)
            else:
                return node.predict_class[0]
        else:
            if X[node.feature] < node.thres:
                return self.traverse(node.left, X)
```

```
            else:
                return self.traverse(node.right, X)

    def print_acc(self, acc):
        print(f'max depth = {self.max_depth}')
        print(f'acc       = {acc}')
        print('====================')

    def predict(self, X, y=None):
        pred = np.zeros(X.shape[0]).astype(np.int32)
        correct = 0
        for i in range(X.shape[0]):
            pred[i] = self.traverse(self.root, X[i])
            if y is not None:
                if pred[i] == y[i, 0]:
                    correct += 1
        acc = correct / X.shape[0] if y is not None else None
        if y is not None:
            self.print_acc(acc)
        return pred, acc


class RandomForest():
    def __init__(
            self, n_estimators, max_features, max_depth=None):
        self.n_estimators = n_estimators
        self.max_features = int(np.round(max_features))
        self.max_depth = max_depth
        self.clfs = []
        for i in range(self.n_estimators):
            self.clfs.append(DecisionTree(self.max_depth))
        self.random_vecs = []
        self.oob = []
        self.oob_error = []
        return None

    def train(self, X, y):
        for i in range(self.n_estimators):
            random_vec = random.sample(range(X.shape[1]),
self.max_features)
            self.random_vecs.append(random_vec)
            sample_num = int(np.round(X.shape[0]*2/3))
            subset_idx = random.sample(range(X.shape[0]), sample_num)
            mask = np.ma.array(np.arange(X.shape[0]), mask=False)
            mask.mask[subset_idx] = True
            oob_idx = mask.compressed()
            self.oob.append(oob_idx)
            self.clfs[i].train(X[subset_idx][:, random_vec],
y[subset_idx])
            pred, _ = self.clfs[i].predict(X[oob_idx][:, random_vec])
            self.oob_error.append((np.sum(pred!=y[oob_idx, 0]),
```

```
pred.shape[0]))
            # print(f'{i+1} trees completed')

    def print_acc(self, acc):
        print(f'n estimators = {self.n_estimators}')
        print(f'max features = {self.max_features}')
        print(f'max depth    = {self.max_depth}')
        print(f'acc          = {acc}')
        print('--------------------')

    def predict(self, X, y=None):
        pred = np.zeros(X.shape[0]).astype(np.int32)
        correct = 0
        for i in range(X.shape[0]):
            vote = []
            for j in range(self.n_estimators):
                vote.append(self.clfs[j].traverse(self.clfs[j].root,
X[i, self.random_vecs[j]]))
            label, cnt = np.unique(vote, return_counts=True)
            pred[i] = label[np.argmax(cnt)]
            if y is not None:
                if pred[i] == y[i, 0]:
                    correct += 1
        acc = correct / X.shape[0] if y is not None else None
        self.print_acc(acc)
        return pred, acc


def run_experiment(X_train, y_train, X_val, y_val, n_estimators,
max_features, max_depth):
    if max_features >= 0.5:
        forest = RandomForest(
            n_estimators=n_estimators, max_features=max_features,
max_depth=max_depth)
        forest.train(X_train, y_train)
        print('performance on train')
        y_pred, acc = forest.predict(X_train, y_train)
        print('performance on val')
        y_pred, acc = forest.predict(X_val, y_val)
        print(f'val error: {np.sum(y_pred!=y_val[:,
0])/y_pred.shape[0]}')
        oob_error = np.array(forest.oob_error)
        oob_error = np.sum(oob_error, axis=0)
        print(f'oob error: {oob_error[0]/oob_error[1]}')
    else:
        print('max_feature < 0.5')
    print('====================')


if __name__ == '__main__':
    X, y, label = load_data('./breast_cancer/data')
```

```
    X_train, y_train, X_val, y_val = split_dataset(X, y, 0.8)

    run_experiment(X_train, y_train, X_val, y_val, 10,
np.sqrt(X_train.shape[0]), 5)
```