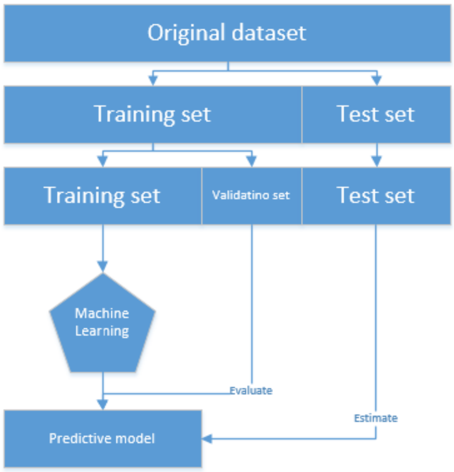
## 基本流程:

下圖為這次作業測試的主要框架，首先我們會把資料分成 Training set 與 Test set，先利用 cross validation set 方法分成 k 折，並透過 cross validation 的平均分數與變異數評估 CART algorithm 所構成的 Random Forest 的參數是否調整好，最後再利用 Test set 做最後的評估測試。




## 訓練參數說明:

| 參數 | 說明 |
| --- | --- |
| n_folds | Validation 有幾折 |
| max_depth | 每棵樹最多多深 |
| min_size | 每次樹的 node 剩下的 group 至少要有多大的 data |
| tree_num | 森林有幾棵樹 |
| bagging_ratio | 每次取多少 Sample 來訓練一棵樹(取後放回) |
| bagging_feature_num | 每次取多少 Feature 來訓練一棵樹(取後放回) |

註:

1. pruning 的方法為如果該 node 深度超過 max_depth，或剩下的 data 少於 min_size，就利用用下的 data 多數屬於哪個 label，把該 node 改為 terminal

2. Random Forest 實作的方法為多數的 Tree 進行 voting ，如果取最多樹覺得該 data 為哪個 label 來決定。

# 實驗改變 training sample 數量:

1. 主要改變 training sample 的數量，改變方法為調整 n_folds 數量，n_folds 數量從 2~10，也就是 trainging sample 與 validation 比率 50%~90%。

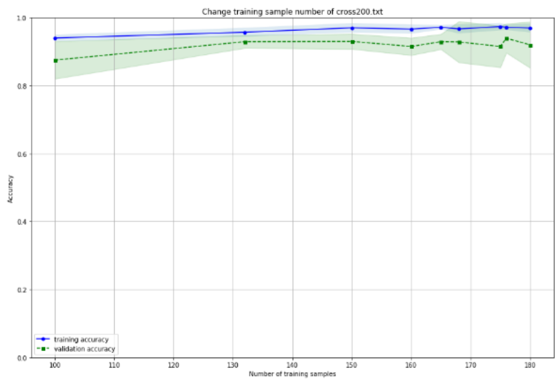
2. 其他固定參數: max_depth = 5(最深為 5 個節點)，min_size = 10 (每次 node 至少需要有多少 data)，tree_num = 10 (森林內有幾棵樹)，bagging_ratio = 0.7(每次取多少 training data)，bagging_feature_num(取最多 4 個 feature，依照 data set 決定)

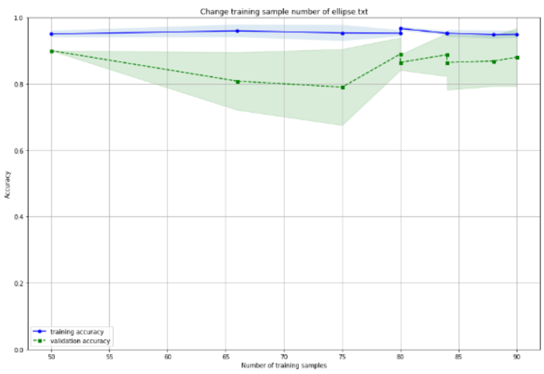
3. 下圖解釋:

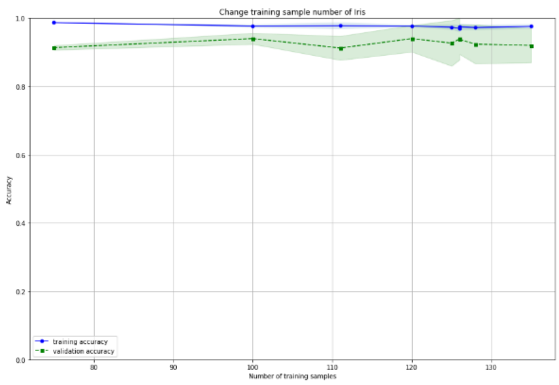
圖(一)~ 圖(五) 分別為 cross200 、 ellipse100 、 Iiris 、 wine 、breast_cancer dataset ，X 軸為 Sample 數量、Y 軸為 Accuracy 比率、藍色線為 trainging 自己的 accuracy、綠色線為 cross validation 的 mean accuracy、藍色範圍與綠色範圍為 cross validation 的 accuracy 變異數。

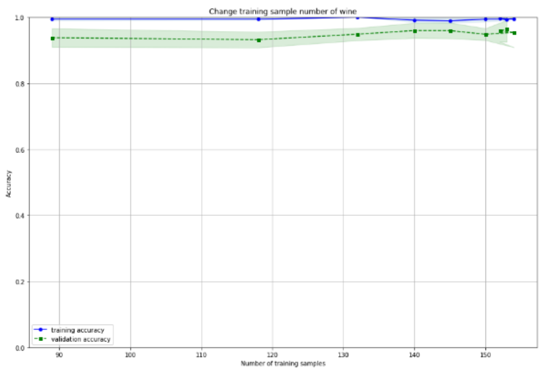
4. 觀察與分析:

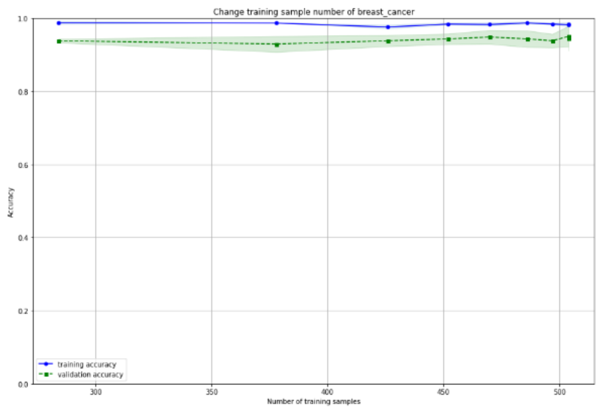
下圖的 accuracy 幾乎都大於 90%，因此可以看出這個參數的效能算是不錯的。第二個可以觀察到當 training sample 越多時，validation mean accuracy 也會越高，原本預想 training accuracy 應該會緩慢下降，因為有更多的不同的 data 進來，而且也可以降低 overfitting 的可能性，但事實上沒有特別的明顯，我覺得應該是因為有 pruning 、forest voting 的方法降低了 variance，因此 overfitting 的情況也就減少了。因此 n_folds = 10 會是比較好的選擇





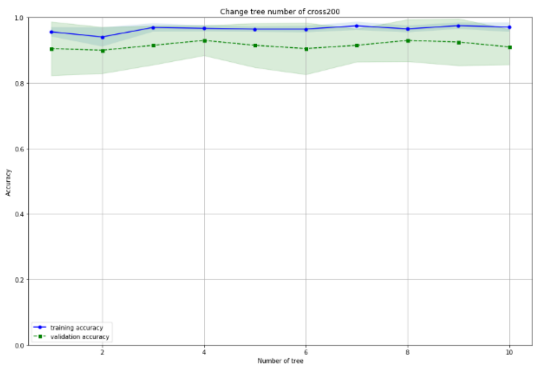

## 實驗改變 Tree number:

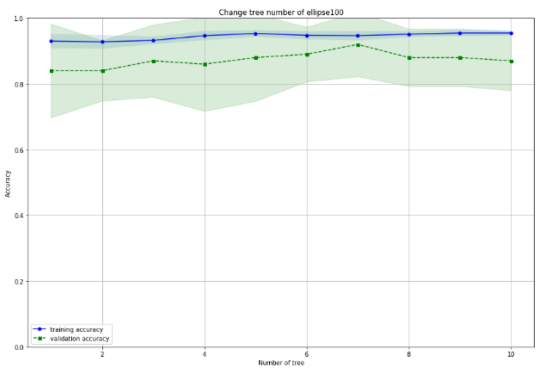
1. 主要改變 random forest 中 tree 的數量，改變方法為調整 tree_num 數量，tree_num 數量從 1~10。
2. 其他固定參數: max_depth = 5(最深為 5 個節點)，min_size = 10 (每次 node 至少需要有多少 data)，n_folds = 10，bagging_ratio = 0.7(每次取多少 training data)，bagging_feature_num(取最多 4 個 feature，依照 data set 決定)
3. 下圖解釋:

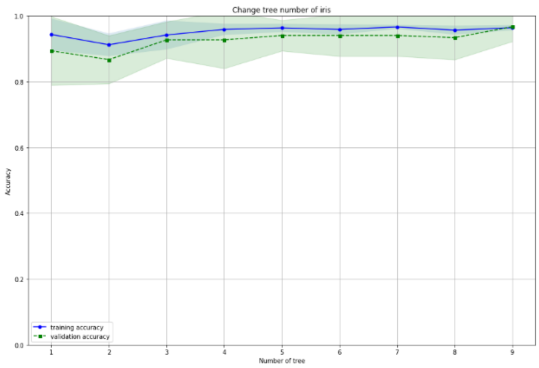
圖(一)~ 圖(五) 分別為 cross200 、 ellipse100 、 Iiris 、 wine 、breast_cancer dataset，X 軸為 tree 數量、Y 軸為 Accuracy 比率、藍色線為 trainging 自己的 accuracy、綠色線為 cross validation 的 mean accuracy、藍色範圍與綠色範圍為 cross validation 的 accuracy 變異數。

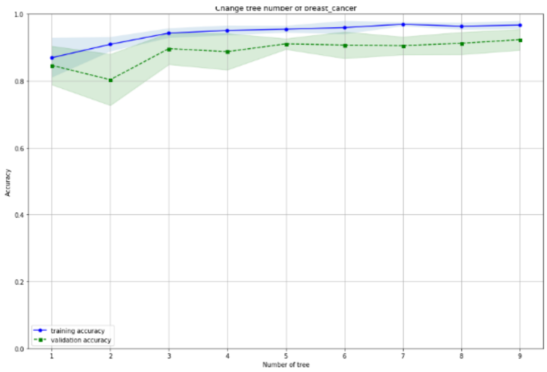
4. 觀察與分析:

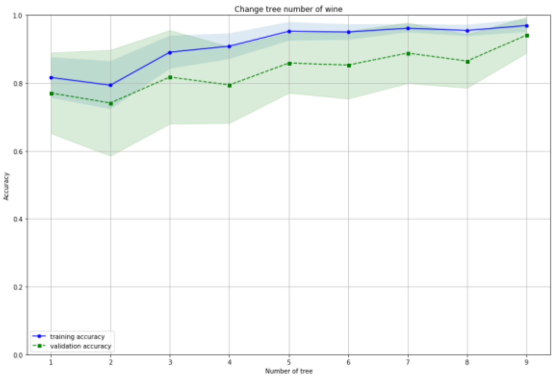
對於 cross200 dataset 來說，增加 tree 的數量效果似乎沒有那麼明顯，我想是因為它的 attribute 較少的緣故，因此 variance 自然也就不會很大。但如果對於 attribute 較多的資料，variance 會比較大，例如 wine data，attribute 有 13 個，我們能利用增加 tree 的方式來把 variance 降低，如最後圖(五)，並且提高 cross validation mean accuracy。



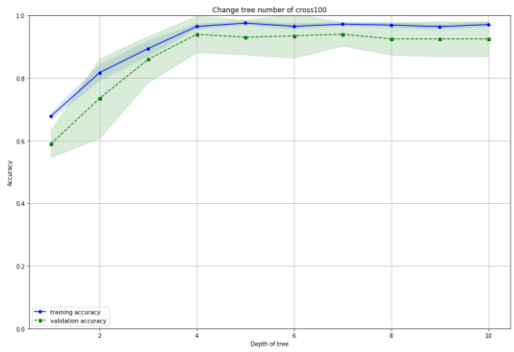


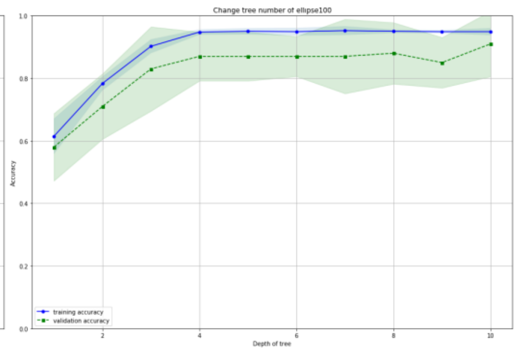



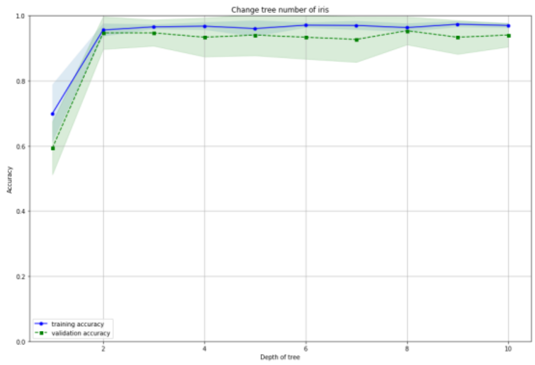



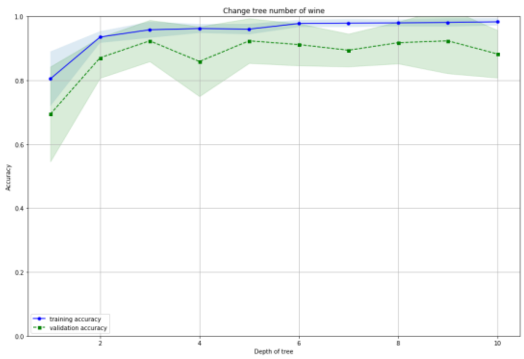

## 實驗改變 max_depth:

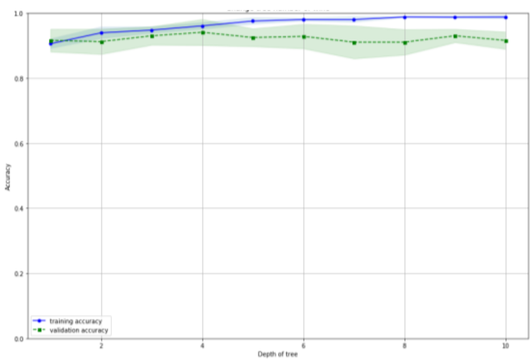
1. 主要改變 max_depth 觀察深度對 tree 的影響，max_depth 從 5 ~ 15。
2. 其他固定參數: tree_num = 10 (森林中有多少樹)，min_size = 10 (每次 node 至少需要有多少 data)，n_folds = 10，bagging_ratio = 0.7(每次取多少 training data)，bagging_feature_num(取最多 4 個 feature，依照 data set 決定)
3. 下圖解釋:
圖(一) ~ 圖(五) 分別為 cross200 、 ellipse100 、 liris 、 wine 、breast_cancer dataset，X 軸為 tree 深度、Y 軸為 Accuracy 比率、藍色線為 trainging 自己的 accuracy、綠色線為 cross validation 的 mean accuracy、藍色範圍與綠色範圍為 cross validation 的 accuracy 變異數。
4. 觀察與分析:
在 cross200 與 ellipse100 的圖中可以觀察到只要深度到達 4~5，此參數的森林樹就能夠收斂，而我想後來大於 10 的深度沒有 overfitting 的原因是 min_size = 10，後來我也有嘗試把 min_size 改小，也就是分得越深、越細，會發現當 tree 數量增加，accuracy 會緩慢減少。




結論:

這次的作業做了一些定量的分析,可以發現如果能設定樹的 min_size 加上 max_depth ，就算只有單一棵樹，如果調教成正確的數值，也能有不錯的效果。我們可以透過多次的定量檢驗，來觀察哪個參數最適合這筆資料，雖然這是很費時間的事，但效能上不僅能提升，且準確度也能夠提升。

程式碼參考:此程式碼主要介紹如何建立 CART Tree

https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

實作:

1. cross validation
2. random forest
3. bagging
4. visualization

```
# CART on the Bank Note dataset
from random import seed
from random import randrange
from csv import reader
import logging
import random
import pandas as pd
import os
import numpy as np
# Constant
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(threadName)-12.12s] [%(levelname)-5.5s]  %(message)s",
    handlers=[
        logging.FileHandler('my.log', 'w', 'utf-8'),
        logging.StreamHandler()
    ])
logger   = logging.getLogger()

# Load a CSV file
def load_csv(filename):
        if 'txt' in filename:
                dataset = []
                file = open(filename,'r')
                for line in file.readlines():
                        line = line.split()
                        dataset.append(line)
        else:
                file = open(filename, "r")
                lines = reader(file,delimiter=' ')
                dataset = list(lines)
        return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                row[column] = float(row[column].strip())

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for i in range(n_folds):
                fold = list()
                while len(fold) < fold_size:
                        index = randrange(len(dataset_copy))
                        fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
        return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
        folds = cross_validation_split(dataset, n_folds)
        logger.info('Divid dataset to '+ str(len(folds))    + ' folds')
        logger.info('Training samples have '  + str(len(folds[0])*(len(folds)-1)) +
' rows')
        validation_scores = list()
        training_scores = list()
        for fold in folds:
                # validation score
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set, [])
                test_set = list()
```

```
                for row in fold:
                        row_copy = list(row)
                        test_set.append(row_copy)
                        row_copy[-1] = None
                predicted = algorithm(train_set, test_set, *args)
                actual = [row[-1] for row in fold]
                accuracy = accuracy_metric(actual, predicted)
                validation_scores.append(accuracy)
                # training score
                test_set = list()
                for row in train_set:
                        row_copy = list(row)
                        test_set.append(row_copy)
                        row_copy[-1] = None
                predicted = algorithm(train_set, test_set, *args)
                actual = [row[-1] for row in train_set]
                training_accuracy = accuracy_metric(actual, predicted)
                training_scores.append(training_accuracy)
        return str(len(folds[0])*(len(folds)-1)),training_scores,validation_scores

# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
        left, right = list(), list()
        for row in dataset:
                if row[index] < value:
                        left.append(row)
                else:
                        right.append(row)
        return left, right

# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
        # count all samples at split point
        n_instances = float(sum([len(group) for group in groups]))
        # sum weighted Gini index for each group
        gini = 0.0
        for group in groups:
                size = float(len(group))
                # avoid divide by zero
                if size == 0:
                        continue
                score = 0.0
                # score the group based on the score for each class
                for class_val in classes:
                        p = [row[-1] for row in group].count(class_val) / size
                        score += p * p
                # weight the group score by its relative size
                gini += (1.0 - score) * (size / n_instances)
        return gini

# Select the best split point for a dataset
def get_split(dataset, sample_feature):
        class_values = list(set(row[-1] for row in dataset))
        b_index, b_value, b_score, b_groups = 999, 999, 999, None
        for index in range(len(dataset[0])-1):
                if not index in sample_feature:
                        continue
                for row in dataset:
                        groups = test_split(index, row[index], dataset)
                        gini = gini_index(groups, class_values)
                        if gini < b_score:
                                b_index, b_value, b_score, b_groups = index, row[ind
ex], gini, groups
        return {'index':b_index, 'value':b_value, 'groups':b_groups}

# Create a terminal node value
def to_terminal(group):
        outcomes = [row[-1] for row in group]
        return max(set(outcomes), key=outcomes.count)

# Create child splits for a node or make terminal
def split(node, max_depth, min_size, depth, sample_feature):
        left, right = node['groups']
```

```
        del(node['groups'])
        # check for a no split
        if not left or not right:
                node['left'] = node['right'] = to_terminal(left + right)
                return
        # check for max depth
        if depth >= max_depth:
                node['left'], node['right'] = to_terminal(left), to_terminal(right)
                return
        # process left child
        if len(left) <= min_size:
                node['left'] = to_terminal(left)
        else:
                node['left'] = get_split(left, sample_feature)
                split(node['left'], max_depth, min_size, depth+1, sample_feature)
        # process right child
        if len(right) <= min_size:
                node['right'] = to_terminal(right)
        else:
                node['right'] = get_split(right, sample_feature)
                split(node['right'], max_depth, min_size, depth+1, sample_feature)

# Build a decision tree
def build_tree(train, max_depth, min_size, sample_feature):
        root = get_split(train, sample_feature)
        root['sample_feature'] = sample_feature
        split(root, max_depth, min_size, 1, sample_feature)
        return root

# Make a prediction with a decision tree
def predict(node, row):
        if row[node['index']] < node['value']:
                if isinstance(node['left'], dict):
                        return predict(node['left'], row)
                else:
                        return node['left']
        else:
                if isinstance(node['right'], dict):
                        return predict(node['right'], row)
                else:
                        return node['right']


# Random Forest of CART Algorithm
'''
@param train, test, max_depth, min_size, tree_num, bagging_ratio, bagging_feature_nu
m
@return forest model
'''
def random_forest(train, test, max_depth, min_size, tree_num, bagging_ratio, bagging
_feature_num):
    #train random_forest
        tree_list = []
        for i in range(tree_num):
                sub_sample_num = int(len(train)*bagging_ratio)
                bagging_list   = random.sample(train, k=sub_sample_num)
                sample_feature = random.sample(range(len(train[0]) - 1), k=bagging_f
eature_num)
                tree           = build_tree(bagging_list, max_depth, min_size, sampl
e_feature)
                tree_list.append(tree)
        #boostrap aggregating
        predictions = list()
        for row in test:
                sub_predictions = list()
                for tree in tree_list:
                        tree_predict   = predict(tree, row)
                        sub_predictions.append(tree_predict)
                ans = max(set(sub_predictions), key=sub_predictions.count)
                predictions.append(ans)
        return (predictions)
```

```
def main():
        filename = 'ellipse100.txt'
        dataset = load_csv(filename)
        for i in range(len(dataset[0])):
                        str_column_to_float(dataset, i)
        n_folds = 10
        max_depth = 5
        min_size = 2
        tree_num = 10
        bagging_ratio = 0.7
        bagging_feature_num = 2
        randomForestDataFrameTraining   = pd.DataFrame(columns=['max_depth','mean_sc
ore','std_score'])
        randomForestDataFrameValidation = pd.DataFrame(columns=['max_depth','mean_sc
ore','std_score'])
        for max_depth in range(1,20):
                sampleNum, training_scores,validation_scores = evaluate_algorithm(da
taset, random_forest, n_folds, max_depth, min_size,tree_num,bagging_ratio,bagging_fe
ature_num)
                randomForestDataFrameTraining   = randomForestDataFrameTraining.appe
nd(pd.Series([max_depth, np.mean(training_scores) , np.std(training_scores)], index=
['sample_num','mean_score','std_score']), ignore_index=True)
                randomForestDataFrameValidation = randomForestDataFrameValidation.ap
pend(pd.Series([max_depth, np.mean(validation_scores) , np.std(validation_scores)],
index=['sample_num','mean_score','std_score']), ignore_index=True)
                print('traing_scores')
                print(np.mean(training_scores))
                print('validation_scores')
                print(np.mean(validation_scores))

        trainingSavePath   = os.path.join('.','training' + '.csv')
        validationSavePath = os.path.join('.','validation' + '.csv')

        randomForestDataFrameTraining.to_csv(trainingSavePath,sep=',')
        randomForestDataFrameValidation.to_csv(validationSavePath,sep=',')
if __name__ == '__main__':
        main()
```

```
# CART on the Bank Note dataset
from random import seed
from random import randrange
from csv import reader
import logging
import random
import pandas as pd
import os
import numpy as np
from sklearn import datasets

# Constant
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(threadName)-12.12s] [%(levelname)-5.5s]  %(message)s",
    handlers=[
        logging.FileHandler('my.log', 'w', 'utf-8'),
        logging.StreamHandler()
    ])
logger   = logging.getLogger()

# Load a CSV file
def load_csv(filename):
        if 'txt' in filename:
                dataset = []
                file = open(filename,'r')
                for line in file.readlines():
                        line = line.split()
                        dataset.append(line)
        else:
                file = open(filename, "r")
                lines = reader(file,delimiter=' ')
                dataset = list(lines)
        return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                row[column] = float(row[column].strip())

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for i in range(n_folds):
                fold = list()
                while len(fold) < fold_size:
                        index = randrange(len(dataset_copy))
                        fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
        return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
        folds = cross_validation_split(dataset, n_folds)
        logger.info('Divid dataset to '+ str(len(folds))    + ' folds')
        logger.info('Training samples have '  + str(len(folds[0])*(len(folds)-1)) +
' rows')
        validation_scores = list()
        training_scores = list()
        for fold in folds:
                # validation score
                train_set = list(folds)
                train_set.remove(fold)
```

```
		train_set = sum(train_set, [])
		test_set = list()
		for row in fold:
			row_copy = list(row)
			test_set.append(row_copy)
			row_copy[-1] = None
		predicted = algorithm(train_set, test_set, *args)
		actual = [row[-1] for row in fold]
		accuracy = accuracy_metric(actual, predicted)
		validation_scores.append(accuracy)
		# training score
		test_set = list()
		for row in train_set:
			row_copy = list(row)
			test_set.append(row_copy)
			row_copy[-1] = None
		predicted = algorithm(train_set, test_set, *args)
		actual = [row[-1] for row in train_set]
		training_accuracy = accuracy_metric(actual, predicted)
		training_scores.append(training_accuracy)
	return str(len(folds[0])*(len(folds)-1)),training_scores,validation_scores

# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
	left, right = list(), list()
	for row in dataset:
		if row[index] < value:
			left.append(row)
		else:
			right.append(row)
	return left, right

# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
	# count all samples at split point
	n_instances = float(sum([len(group) for group in groups]))
	# sum weighted Gini index for each group
	gini = 0.0
	for group in groups:
		size = float(len(group))
		# avoid divide by zero
		if size == 0:
			continue
		score = 0.0
		# score the group based on the score for each class
		for class_val in classes:
			p = [row[-1] for row in group].count(class_val) / size
			score += p * p
		# weight the group score by its relative size
		gini += (1.0 - score) * (size / n_instances)
	return gini

# Select the best split point for a dataset
def get_split(dataset, sample_feature):
	class_values = list(set(row[-1] for row in dataset))
	b_index, b_value, b_score, b_groups = 999, 999, 999, None
	for index in range(len(dataset[0])-1):
		if not index in sample_feature:
			continue
		for row in dataset:
			groups = test_split(index, row[index], dataset)
			gini = gini_index(groups, class_values)
			if gini < b_score:
				b_index, b_value, b_score, b_groups = index, row[ind
ex], gini, groups
	return {'index':b_index, 'value':b_value, 'groups':b_groups}

# Create a terminal node value
def to_terminal(group):
	outcomes = [row[-1] for row in group]
	return max(set(outcomes), key=outcomes.count)

# Create child splits for a node or make terminal
```

```
def split(node, max_depth, min_size, depth, sample_feature):
        left, right = node['groups']
        del(node['groups'])
        # check for a no split
        if not left or not right:
                node['left'] = node['right'] = to_terminal(left + right)
                return
        # check for max depth
        if depth >= max_depth:
                node['left'], node['right'] = to_terminal(left), to_terminal(right)
                return
        # process left child
        if len(left) <= min_size:
                node['left'] = to_terminal(left)
        else:
                node['left'] = get_split(left, sample_feature)
                split(node['left'], max_depth, min_size, depth+1, sample_feature)
        # process right child
        if len(right) <= min_size:
                node['right'] = to_terminal(right)
        else:
                node['right'] = get_split(right, sample_feature)
                split(node['right'], max_depth, min_size, depth+1, sample_feature)

# Build a decision tree
def build_tree(train, max_depth, min_size, sample_feature):
        root = get_split(train, sample_feature)
        root['sample_feature'] = sample_feature
        split(root, max_depth, min_size, 1, sample_feature)
        return root

# Make a prediction with a decision tree
def predict(node, row):
        if row[node['index']] < node['value']:
                if isinstance(node['left'], dict):
                        return predict(node['left'], row)
                else:
                        return node['left']
        else:
                if isinstance(node['right'], dict):
                        return predict(node['right'], row)
                else:
                        return node['right']


# Random Forest of CART Algorithm
'''
@param train, test, max_depth, min_size, tree_num, bagging_ratio, bagging_feature_nu
m
@return forest model
'''
def random_forest(train, test, max_depth, min_size, tree_num, bagging_ratio, bagging
_feature_num):
    #train random_forest
        tree_list = []
        for i in range(tree_num):
                sub_sample_num = int(len(train)*bagging_ratio)
                bagging_list   = random.sample(train, k=sub_sample_num)
                sample_feature = random.sample(range(len(train[0]) - 1), k=bagging_f
eature_num)
                tree           = build_tree(bagging_list, max_depth, min_size, sampl
e_feature)
                tree_list.append(tree)
        #boostrap aggregating
        predictions = list()
        for row in test:
                sub_predictions = list()
                for tree in tree_list:
                        tree_predict   = predict(tree, row)
                        sub_predictions.append(tree_predict)
                ans = max(set(sub_predictions), key=sub_predictions.count)
                predictions.append(ans)
        return (predictions)
```

```
def main():
        filename = 'cross200.txt'
        dataset = load_csv(filename)
        for i in range(len(dataset[0])):
                        str_column_to_float(dataset, i)
        n_folds = 10
        max_depth = 5
        min_size = 100
        tree_num = 10
        bagging_ratio = 0.7
        bagging_feature_num = 2
        randomForestDataFrameTraining   = pd.DataFrame(columns=['max_depth','mean_sc
ore','std_score'])
        randomForestDataFrameValidation = pd.DataFrame(columns=['max_depth','mean_sc
ore','std_score'])
        wineDataList=[]
        wine = datasets.load_wine()
        for index in range(len(wine['target'])):
                wineDataList.append(np.append(wine['data'][index],wine['target'][ind
ex]).tolist())
        dataset = wineDataList
        for max_depth in range(1,20):
                sampleNum, training_scores,validation_scores = evaluate_algorithm(da
taset, random_forest, n_folds, max_depth, min_size,tree_num,bagging_ratio,bagging_fe
ature_num)
                randomForestDataFrameTraining   = randomForestDataFrameTraining.appe
nd(pd.Series([max_depth, np.mean(training_scores) , np.std(training_scores)], index=
['sample_num','mean_score','std_score']), ignore_index=True)
                randomForestDataFrameValidation = randomForestDataFrameValidation.ap
pend(pd.Series([max_depth, np.mean(validation_scores) , np.std(validation_scores)],
index=['sample_num','mean_score','std_score']), ignore_index=True)
                print('traing_scores')
                print(np.mean(training_scores))
                print('validation_scores')
                print(np.mean(validation_scores))

        trainingSavePath   = os.path.join('.','training' + '.csv')
        validationSavePath = os.path.join('.','validation' + '.csv')

        randomForestDataFrameTraining.to_csv(trainingSavePath,sep=',')
        randomForestDataFrameValidation.to_csv(validationSavePath,sep=',')
if __name__ == '__main__':
        main()
```

```
import pandas as pd
import matplotlib.pyplot as plt
trainDataframe       = pd.read_csv('training.csv')
validationDataframe  = pd.read_csv('validation.csv')
train_mean      = trainDataframe.loc[:,'mean_score']/100
train_std       = trainDataframe.loc[:,'std_score']/100
validation_mean = validationDataframe.loc[:,'mean_score']/100
validation_std  = validationDataframe.loc[:,'std_score']/100
train_size      = trainDataframe.loc[:,'sample_num']
plt.figure(figsize=(15,10))
plt.plot(train_size, train_mean,color='blue',marker='o',markersize=5,label='training
 accuracy')
plt.fill_between(train_size,train_mean+train_std,train_mean - train_std,alpha =
0.15 )
plt.plot(train_size, validation_mean, color='green', linestyle='--', marker='s', mar
kersize=5, label='validation accuracy')
plt.fill_between(train_size, validation_mean + validation_std, validation_mean - val
idation_std,alpha = 0.15,color ='green')
plt.grid()
plt.title("Change tree number of breast")
plt.xlabel('Depth of tree')
plt.ylabel('Accuracy')
plt.legend(loc='lower left')
plt.ylim([0,1.0])
plt.show()
```