1 Psychoinformatics - Week 10 (Exercises)

by 李文婷 (b09207012@ntu.edu.tw (mailto:b09207012@ntu.edu.tw))

In [2]:
```
import warnings, numpy as np
import xgboost
from matplotlib.pyplot import *
%matplotlib inline
warnings.simplefilter('ignore', DeprecationWarning)
from sklearn import *
import copy
```
executed in 2.47s, finished 22:44:37 2023-11-10

2 1 執行並觀察以下的機器學習結果 (2分)

2.1 1.0 IRIS dataset & Ensemble model function

In [3]:
```
iris = datasets.load_iris()
X=iris.data
Y=iris.target
```
executed in 14ms, finished 22:44:37 2023-11-10

In [4]:
```
np.random.seed(0)
sss=model_selection.StratifiedShuffleSplit(n_splits=5,test_size=0.1)
def EnsembleModels(og_model, Max_n_estimators):
    accs=[] # mean cross-validation accuracies of the models w/ different n_
    for n in range(1,Max_n_estimators+1):
        print(n,end=' ') # showing progress
        acc=[] # cross-validation accuracies of the ensemble model w/ n_esti
        for train_index, test_index in sss.split(X, Y): # 5-fold cross-valic
            X_train, X_test = X[train_index], X[test_index]
            Y_train, Y_test = Y[train_index], Y[test_index]
            model=copy.deepcopy(og_model) # to avoid possible model re-trair
            model.n_estimators=n
            model.fit(X_train[:,0:2],Y_train) #training
            acc.append(model.predict(X_test[:,0:2])==Y_test)
        accs.append(np.mean(acc)) # aggregating mean cross-validation accurc
    return(accs)
```
executed in 16ms, finished 22:44:37 2023-11-10

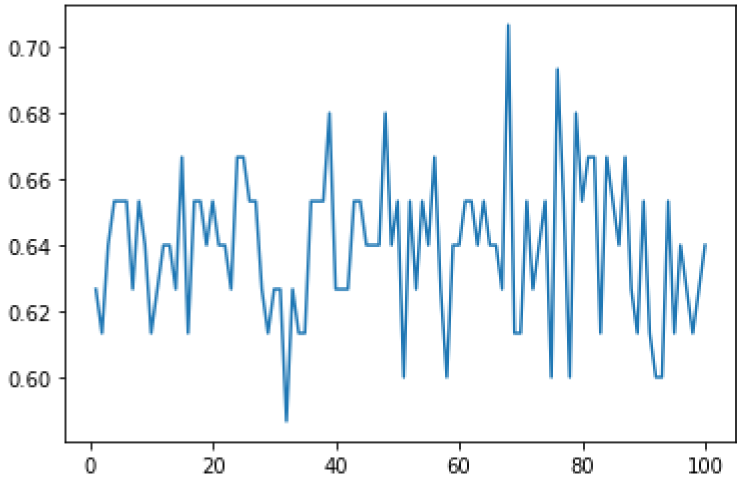
2.2 1.1 Bagging (Bootstrap Aggregating)

2.2.1 1.1.1 Tree max_depth = 1

In [15]:
```
model=ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=1))
Bagging_1 = EnsembleModels(model,100)
plot(range(1,101),Bagging_1);

print('mean accuracy:', np.mean(Bagging_1))
```
executed in 25.3s, finished 22:54:55 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.6391999999999999
```

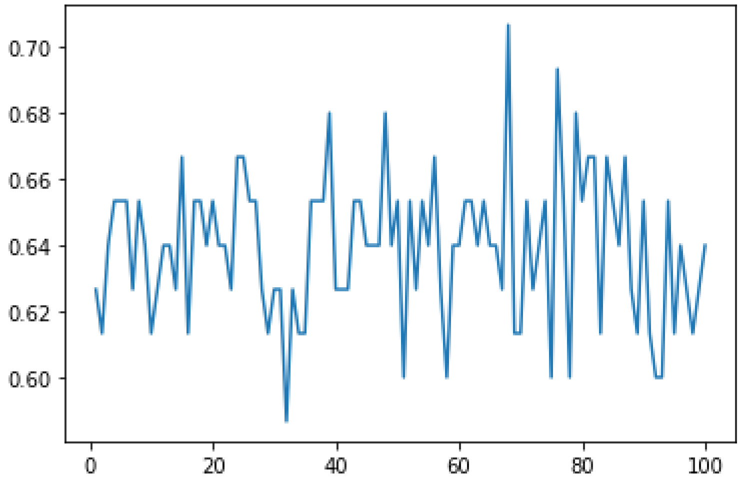


2.2.2 1.1.2 Tree max_depth = 3

In [16]:
```
model=ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=3))
Bagging_3 = EnsembleModels(model,100)
plot(range(1,101),Bagging_1);

print('mean accuracy:', np.mean(Bagging_3))
```
executed in 25.9s, finished 22:55:21 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.7758666666666667
```



In [18]:
```
# 使用t-test檢定兩者正確率是否相同
from scipy.stats import ttest_ind
ttest_ind(Bagging_1, Bagging_3, equal_var=False)
```
executed in 7ms, finished 22:55:44 2023-11-10

Out[18]:
```
Ttest_indResult(statistic=-27.092678305891035, pvalue=3.9811267298612006e-
58)
```

2.3 1.2 Boosting

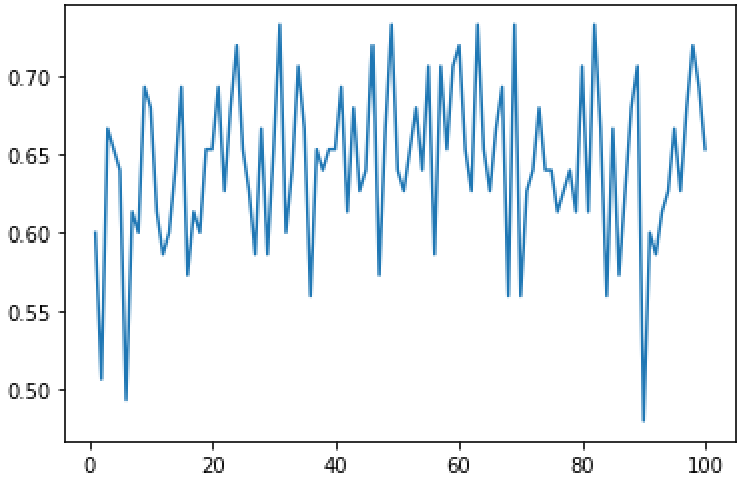
2.3.1 1.2.1 AdaBoost

2.3.1.1 1.2.1.1 Tree max_depth = 1

In [19]:
```
model=ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier(max_depth=1))
AdaBoost_1 = EnsembleModels(model,100)
plot(range(1,101),AdaBoost_1);

print('mean accuracy:', np.mean(AdaBoost_1))
```
executed in 24.0s, finished 22:57:26 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.6434666666666666
```

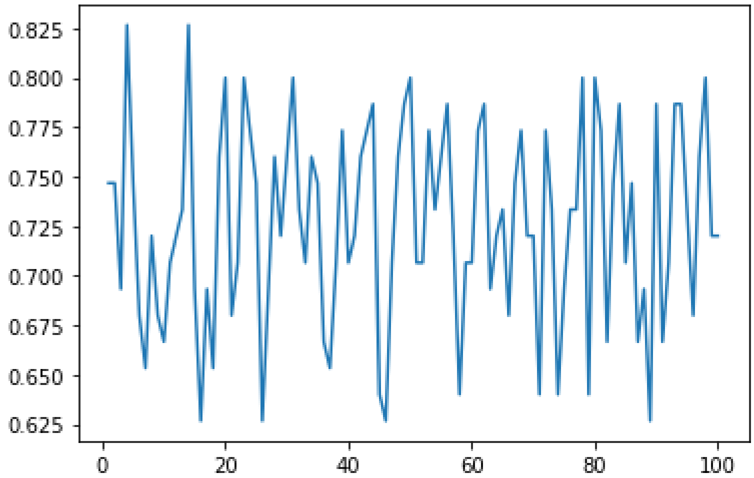


2.3.1.2 1.2.1.2 Tree max_depth = 3

In [20]:
```
model=ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier(max_depth=3))
AdaBoost_3 = EnsembleModels(model,100)
plot(range(1,101),AdaBoost_3);

print('mean accuracy:', np.mean(AdaBoost_3))
```
executed in 25.0s, finished 22:57:51 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.7258666666666668
```



In [21]:
```
# 使用t-test檢定兩者正確率是否相同
from scipy.stats import ttest_ind
ttest_ind(AdaBoost_1, AdaBoost_3, equal_var=False)
```
executed in 17ms, finished 22:57:51 2023-11-10

Out[21]:
```
Ttest_indResult(statistic=-11.391220041087026, pvalue=1.9251447921898965e-
23)
```

2.3.2 1.2.2 Gradient Boosting

The following two implementations are conceptually identical but XGBoost is more resource-efficient and can be parallelized/distributed.

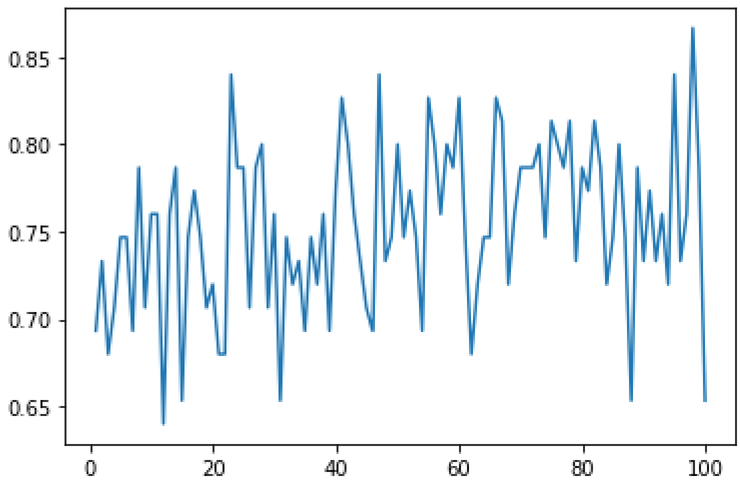
2.3.2.1 1.2.2.1 Scikit-learn's Gradient Tree Boosting

1.2.2.1.1 Tree max_depth = 1

In [22]:
```
model=ensemble.GradientBoostingClassifier(max_depth=1)
Scikit_1 = EnsembleModels(model,100)
plot(range(1,101),Scikit_1);

print('mean accuracy:', np.mean(Scikit_1))
```
executed in 22.0s, finished 22:59:23 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.7530666666666666
```

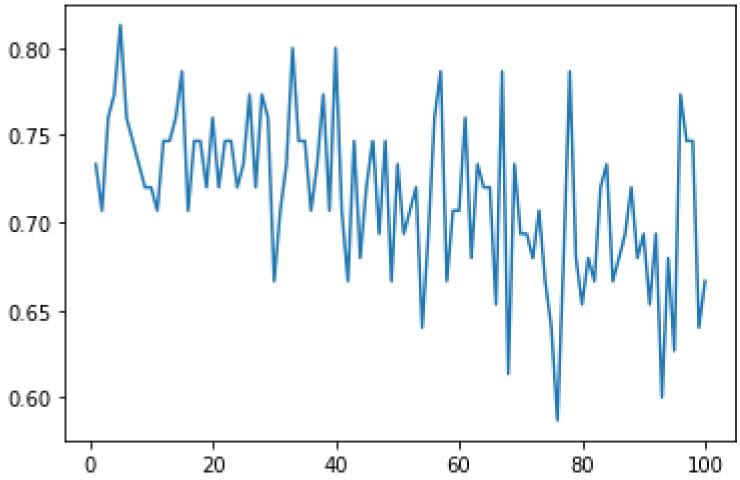


1.2.2.1.2 Tree max_depth = 3

In [23]:
```
model=ensemble.GradientBoostingClassifier(max_depth=3)
Scikit_3 = EnsembleModels(model,100)
plot(range(1,101),Scikit_3);

print('mean accuracy:', np.mean(Scikit_3))
```
executed in 32.9s, finished 22:59:56 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.7152000000000001
```



In [24]:
```
# 使用t-test檢定兩者正確率是否相同
from scipy.stats import ttest_ind
ttest_ind(Scikit_1, Scikit_3, equal_var=False)
```
executed in 15ms, finished 22:59:56 2023-11-10

Out[24]:
```
Ttest_indResult(statistic=5.739441446587949, pvalue=3.536108414414076e-08)
```

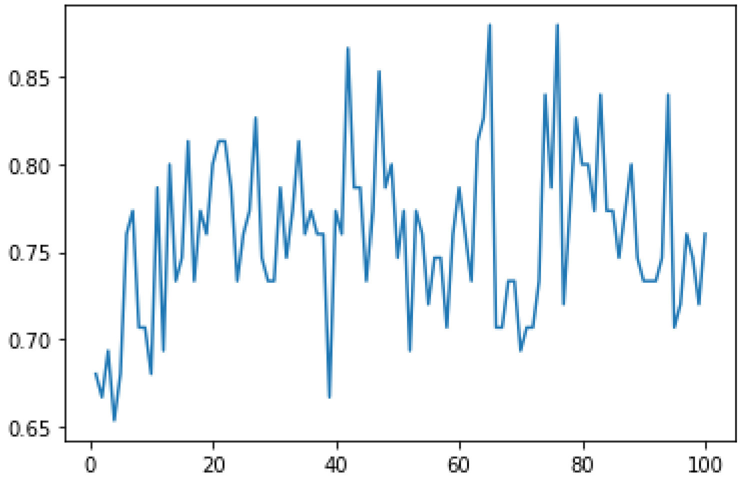
2.3.2.2 1.2.2.2 XGBoost (eXtreme Gradient Boosting)

1.2.2.2.1 Tree max_depth = 1

In [25]:
```
model=xgboost.XGBClassifier(max_depth=1)
xgboost_1 = EnsembleModels(model,100)
plot(range(1,101),xgboost_1);

print('mean accuracy:', np.mean(xgboost_1))
```
executed in 6.70s, finished 23:00:27 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.7592000000000002
```

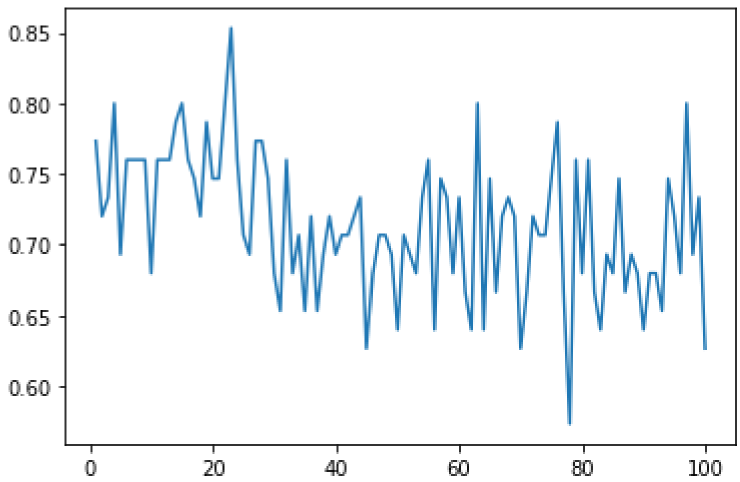


1.2.2.2.2 Tree max_depth = 3

In [26]:
```
model=xgboost.XGBClassifier(max_depth=3)
xgboost_3 = EnsembleModels(model,100)
plot(range(1,101),xgboost_3);

print('mean accuracy:', np.mean(xgboost_3))
```
executed in 9.42s, finished 23:00:37 2023-11-10

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 mean ac
curacy: 0.7141333333333333
```



In [27]:
```
# 使用t-test檢定兩者正確率是否相同
from scipy.stats import ttest_ind
ttest_ind(xgboost_1, xgboost_3, equal_var=False)
```
executed in 16ms, finished 23:00:44 2023-11-10

Out[27]:
```
Ttest_indResult(statistic=6.582498209822628, pvalue=4.0785081856641936e-1
0)
```

3 2 根據以上的觀察回答以下的問題 (6 分)

3.1 2.1 在Bagging時, 1.1.2中複雜模型的正確率是否比1.1.1簡單模型的正確率好或差? 為什麼 (2分)

在計算出兩個模型的平均正確率，並使用t-test作檢定後，發現1.1.2的複雜模型的正確率顯著地較1.1.1的簡單模型表現佳。 推測應是因為max_depth為1的decision tree的複雜度仍不足以處理這筆資料（類似老師舉的「小學生投票微積分答案」的例子），因此表現較深度為3的模型差。

3.2 2.2 在Boosting時, 1.2.1.2/1.2.2.1.2/1.2.2.2.2中複雜模型的正確率是否比1.2.1.1/1.2.2.1.1/1.2.2.2.1中相對應的簡單模型正確率好或差? 為什麼 (2分)

- 1.2.1.2 > 1.2.1.1
- 1.2.2.1.2 < 1.2.2.1.1
- 1.2.2.2.2 < 1.2.2.2.1

在計算出兩個模型的平均正確率，並使用t-test作檢定後，發現除了AdaBoost是複雜模型表現優於簡單模型外，Scikit-learn's Gradient Tree Boosting和eXtreme Gradient Boosting都是簡單模型的正確率較複雜模型高。 相較於Bagging是複雜模型表現比簡單模型好，Boosting在這筆資料中卻普遍是簡單模型表現較複雜模型好，原因可能為Boosting是根據前面的模型的錯誤之處再著重訓練，相比於每次抽樣再放回的Bagging，它的overfitting的情況會更加嚴重，因此複雜的模型反而表現較差。

3.3 2.3 為何只有Boosting在簡單模型時(1.2.1.1/1.2.2.1.1/1.2.2.2.1)，正確率大致上會隨著n_estimators數目變多而增加，但Bagging和複雜的Boosting模型卻不是如此? (2分)

使用Bagging時，會影響正確率好壞的主要是由tree max_depth決定，如同3.1題所述，若模型複雜度本身不足以處理此筆資料，則n_estimators數目變多也不會有幫助；而模型複雜度足夠時，n_estimators的數目也並不會對正確率有影響（類似「只要一兩個大學生就可以投票出微積分答案」的例子）。

複雜的Boosting模型在3.2題中的討論也有提及，它出現了overfitting的情況，模型本身的表現便不佳，因此即使n_estimators數目變多，也不會使原本就已經overfitting的模型表現得更好。