Psychoinformatics - Week 10 (Exercises)

by your name (your email)

In [ ]:
```
# !pip3 install xgboost
import warnings, numpy as np
import xgboost
import matplotlib.pyplot as plt
from matplotlib.pyplot import *
%matplotlib inline
warnings.simplefilter('ignore', DeprecationWarning)
from sklearn import *
import copy
```

1 執行並觀察以下的機器學習結果 (2分)

1.0 IRIS dataset & Ensemble model function

In [ ]:
```
iris = datasets.load_iris()
X=iris.data
Y=iris.target
```

In [ ]:
```
Y
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Acuracy

In [ ]:
```
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_estimator
    for n in range(1,Max_n_estimators+1):
        print(n,end=' ') # showing progress
        acc=[] # cross-validation accuracies of the ensemble model w/ n_estimators=n
        for train_index, test_index in sss.split(X, Y): # 5-fold cross-validation of
            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-training
            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 accuracies acro
    return(accs)
```

在此建立了一個函數 `EnsembleModels`，該函數的目的是在使用指定的原始模型（`og_model`）時，評估在不同的 `n_estimators` 下，形成的Ensemble model的性能。這裡使用的Ensemble model 基於 Stratified Shuffle Split 方法來交叉驗證。

1. `sss = model_selection.StratifiedShuffleSplit(n_splits=5, test_size=0.1)`：創建一個分層隨機抽樣的交叉驗證物件，將資料集分成5個子集，10% 的數據將被用作測試集，
    - 分層隨機抽樣：其中每個子集中的類別比例與整個資料集中的比例相同
2. 定義函數 `EnsembleModels(og_model, Max_n_estimators)`：
    - `og_model`：原始模型，這是在每次迭代中複製的。
    - `Max_n_estimators`：Ensemble model中樹的最大數量。Ensemble model是通過組合多個弱學習器，以形成一個更強大的模型。而 n_estimators 就是指定用於組合的弱學習器（樹）的數量。
3. `accs = []`：用於保存每個Ensemble model的平均交叉驗證準確度。
4. `for n in range(1, Max_n_estimators + 1)`，該迴圈迭代不同數量的基本模型（樹）。
5. 內部迴圈 `for train_index, test_index in sss.split(X, Y)`：在每個交叉驗證迭代中，使用分層隨機抽樣來獲取訓練集和測試集。
6. `model.fit(X_train[:, 0:2], Y_train)`：使用訓練集訓練模型，僅使用資料的前兩個特徵。
7. `acc.append(model.predict(X_test[:, 0:2]) == Y_test)`：計算模型在測試集上的預測準確度。
8. `accs.append(np.mean(acc))`：計算當前基模型數量下的平均交叉驗證準確度，並將其添加到 `accs` 列表中。
9. 返回 `accs` 列表，其中包含不同樹數量下的平均交叉驗證準確度。

Precision

In [ ]:
```python
from sklearn.metrics import precision_score

np.random.seed(0)
sss = model_selection.StratifiedShuffleSplit(n_splits=5, test_size=0.1)

def EnsembleModels_pre(og_model, Max_n_estimators):
    precisions = []  # mean cross-validation precisions of the models w/ different n_
    for n in range(1, Max_n_estimators + 1):
        print(n, end=' ')  # showing progress
        precision = []  # cross-validation precisions of the ensemble model w/ n_esti
        for train_index, test_index in sss.split(X, Y):  # 5-fold cross-validation of
            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-training
            model.n_estimators = n
            model.fit(X_train[:, 0:2], Y_train)  # training
            y_pred = model.predict(X_test[:, 0:2])
            precision.append(precision_score(Y_test, y_pred, average='weighted',zero_
        precisions.append(np.mean(precision))  # aggregating mean cross-validation pr
    return precisions
```

執行模型

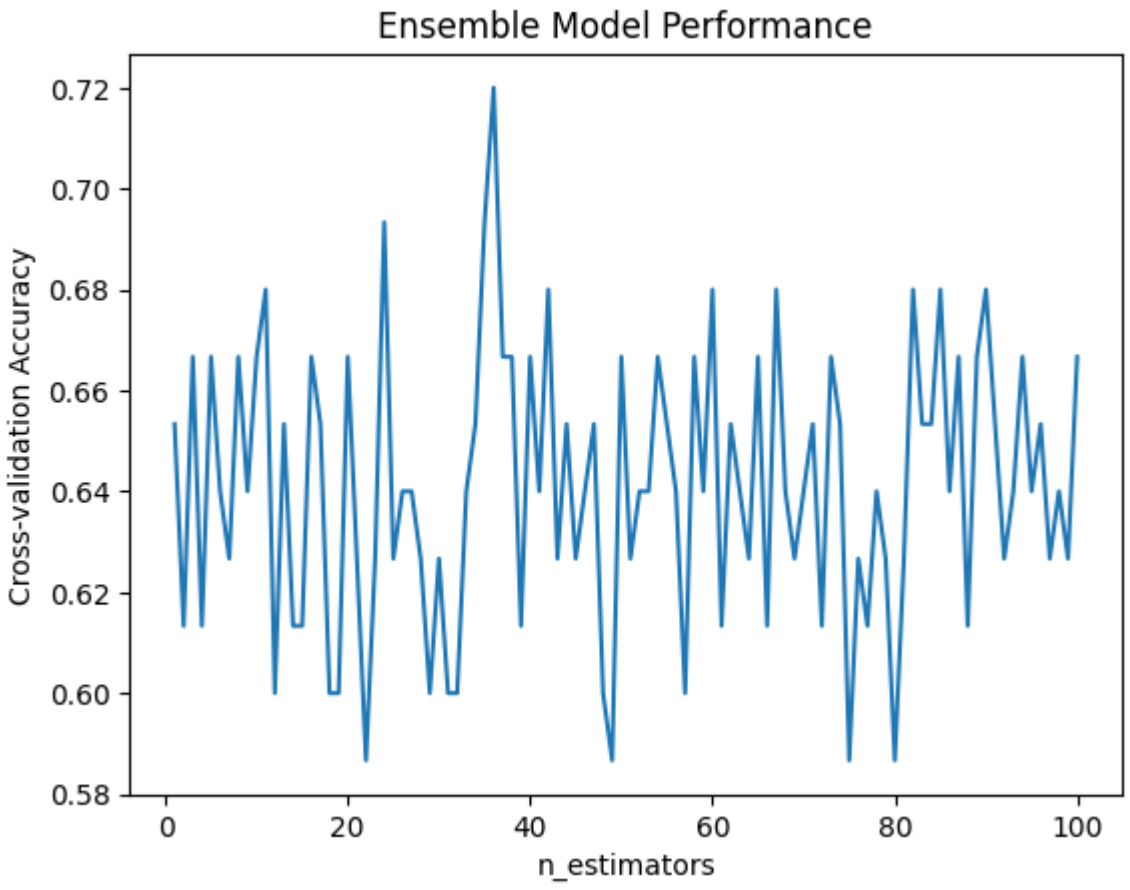
利用**平均數**、**最大值**、**最小值**、**標準差**紀錄每個模型的 Accuracy 的統計分佈

1.1 Bagging (Bootstrap Aggregating)

1.1.1 Tree max_depth = 1

In [ ]:
```
model=ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=1))
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

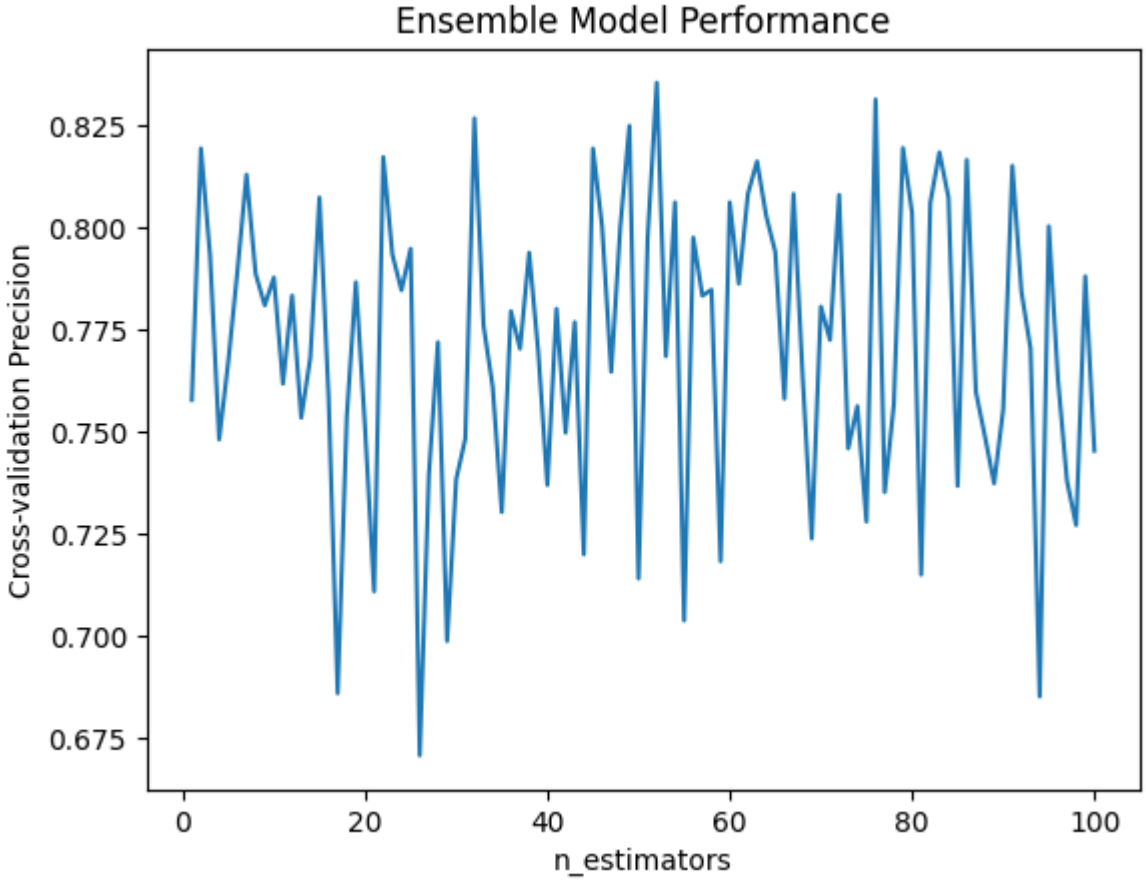
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 3
2 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Accuracy: 0.6410666666666667
Standard Deviation of Accuracy: 0.026976862514219685
Max Accuracy: 0.72
Min Accuracy: 0.5866666666666667
```

In [ ]:
```
model=ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=1))
# Assume EnsembleModels returns accs list
accs = EnsembleModels_pre(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Precision')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Precision: {mean_accuracy}')
print(f'Standard Deviation of Precision: {std_accuracy}')
print(f'Max Precision: {max_accuracy}')
print(f'Min Precision: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```



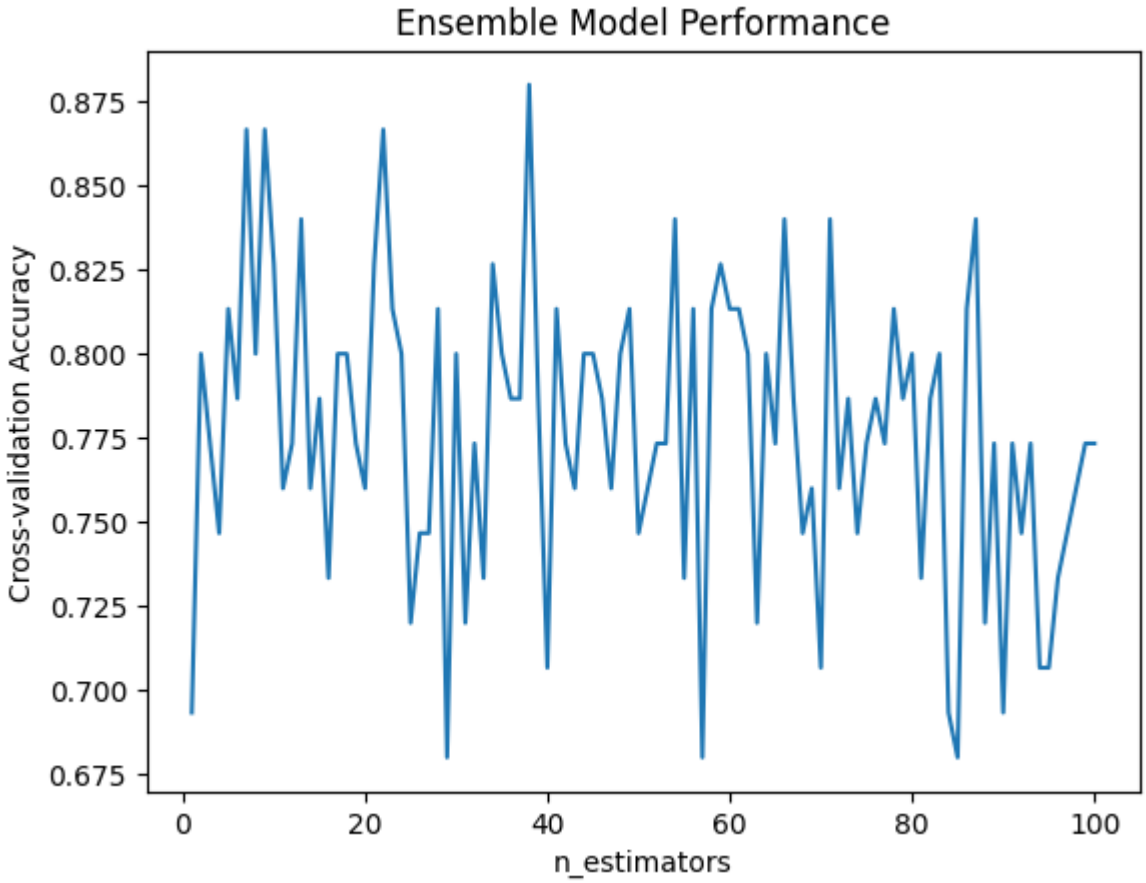

```
Mean Precision: 0.7714065175565176
Standard Deviation of Precision: 0.03559789796468128
Max Precision: 0.8355820105820106
Min Precision: 0.6706734006734006
```

1.1.2 Tree max_depth = 3

In [ ]:
```
model=ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=3))
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)
```

```
# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```

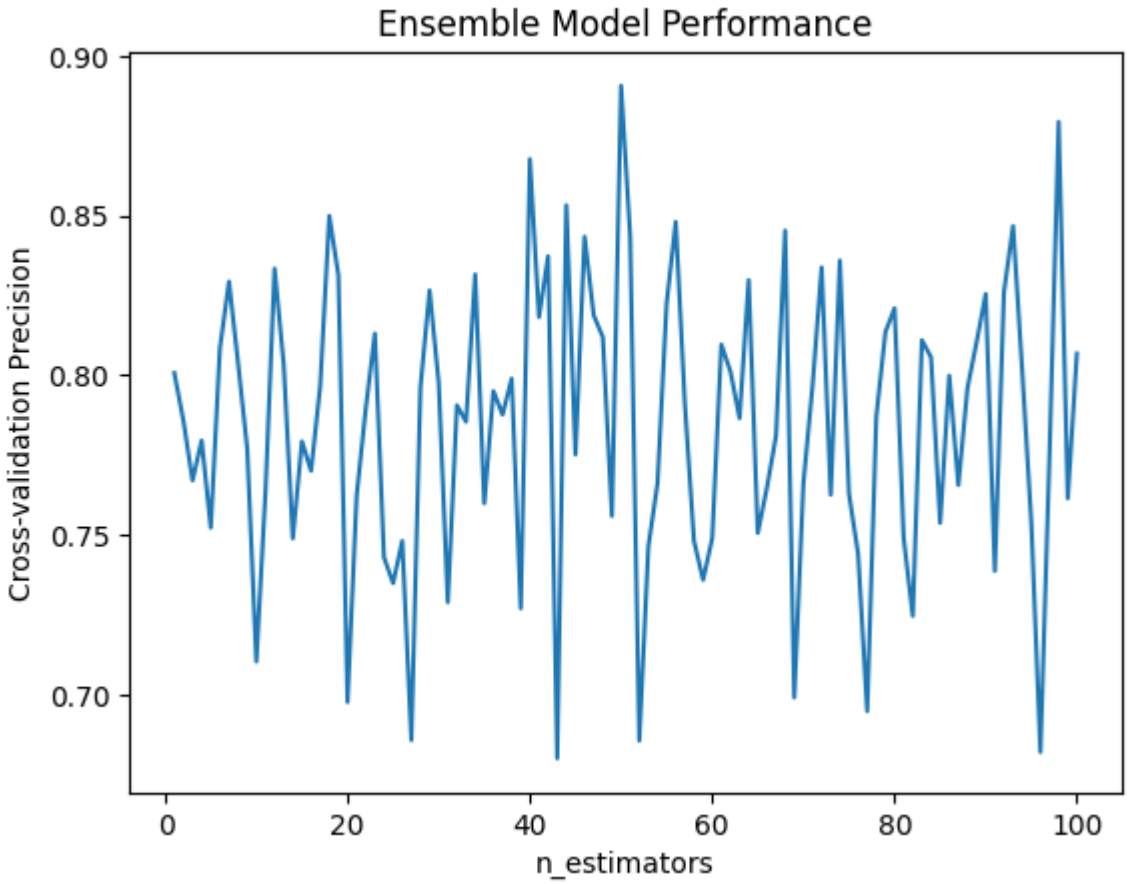



```
Mean Accuracy: 0.7769333333333331
Standard Deviation of Accuracy: 0.043486089729935486
Max Accuracy: 0.88
Min Accuracy: 0.68
```

In [ ]:
```
model=ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=3))
# Assume EnsembleModels returns accs list
accs = EnsembleModels_pre(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)
```

```
# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Precision')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Precision: {mean_accuracy}')
print(f'Standard Deviation of Precision: {std_accuracy}')
print(f'Max Precision: {max_accuracy}')
print(f'Min Precision: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Precision: 0.7847396825396825
Standard Deviation of Precision: 0.04473990051670579
Max Precision: 0.8906084656084655
Min Precision: 0.6799999999999999
```

Use HyperOpt (Bayesian Optimization) to test Bagging

In [ ]:

```
# @title
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials

# Define objective function
def objective(params):
    max_depth = int(params['max_depth'])  # Convert to integer

    model = ensemble.BaggingClassifier(tree.DecisionTreeClassifier(max_depth=max_dept

    # Call your EnsembleModels function
    accs = EnsembleModels(model, Max_n_estimators=40)
```

```python
    # Return the objective value (1 - mean accuracy) as Hyperopt minimizes
    return {'loss': 1 - np.mean(accs), 'status': STATUS_OK, 'max_depth': int(max_dept

# Define search space
space = {
    'max_depth': hp.quniform('max_depth', 1, 10, 1)  # 這裡假設 max_depth 可以在 1 到 10
}

# Create Trials object to track evaluation results
trials = Trials()


# Use Hyperopt's fmin function for optimization
best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=40, trials=trials)

depth_loss_dict = {}

for result in trials.results:
    max_depth = int(result['max_depth'])
    loss = result['loss']
    depth_loss_dict[max_depth] = loss

for max_depth in range(1, 11):
    loss = depth_loss_dict.get(max_depth, None)
    if loss is not None:
        print(f"max_depth: {max_depth}, loss: {loss}")
    else:
        print(f"max_depth: {max_depth}, loss: N/A (not evaluated)")
```

```
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
100%|██████████| 40/40 [07:55<00:00, 11.89s/trial, best loss: 0.22133333333333327]
max_depth: 1, loss: 0.3623333333333333
max_depth: 2, loss: 0.24
max_depth: 3, loss: 0.22166666666666668
max_depth: 4, loss: 0.2460000000000001
max_depth: 5, loss: 0.2663333333333333
max_depth: 6, loss: 0.2736666666666666
max_depth: 7, loss: 0.28666666666666674
max_depth: 8, loss: 0.2899999999999999
max_depth: 9, loss: 0.31966666666666677
max_depth: 10, loss: 0.29033333333333344
```

In [ ]:
```python
for max_depth in range(1, 11):
    loss = depth_loss_dict.get(max_depth, None)
    if loss is not None:
        print(f"max_depth: {max_depth}, loss: {loss}")
    else:
        print(f"max_depth: {max_depth}, loss: N/A (not evaluated)")
```

```
max_depth: 1, loss: 0.3623333333333333
max_depth: 2, loss: 0.24
max_depth: 3, loss: 0.22166666666666668
max_depth: 4, loss: 0.2460000000000001
max_depth: 5, loss: 0.2663333333333333
max_depth: 6, loss: 0.2736666666666666
max_depth: 7, loss: 0.28666666666666674
max_depth: 8, loss: 0.2899999999999999
max_depth: 9, loss: 0.31966666666666677
max_depth: 10, loss: 0.29033333333333344
```

1.2 Boosting

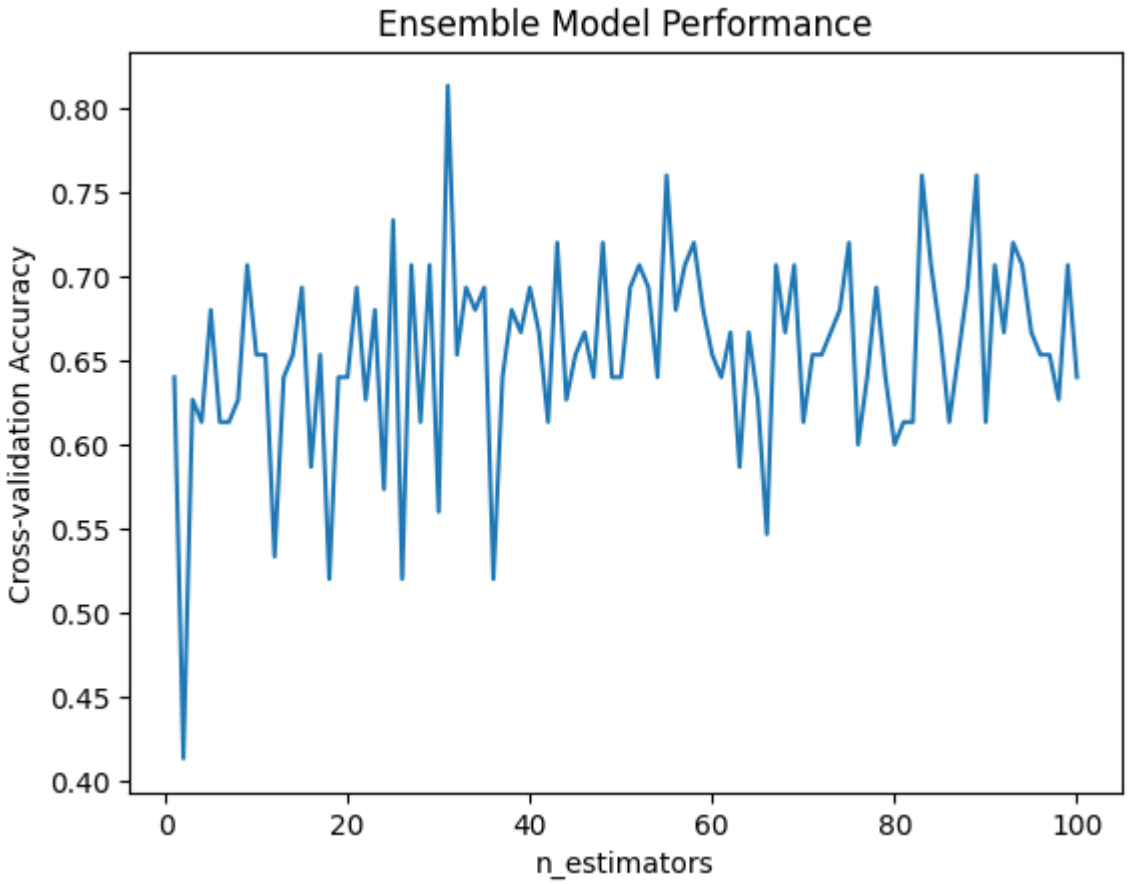
1.2.1 AdaBoost

1.2.1.1 Tree max_depth = 1

In [ ]:
```python
model=ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier(max_depth=1))
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)
```

```
# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```



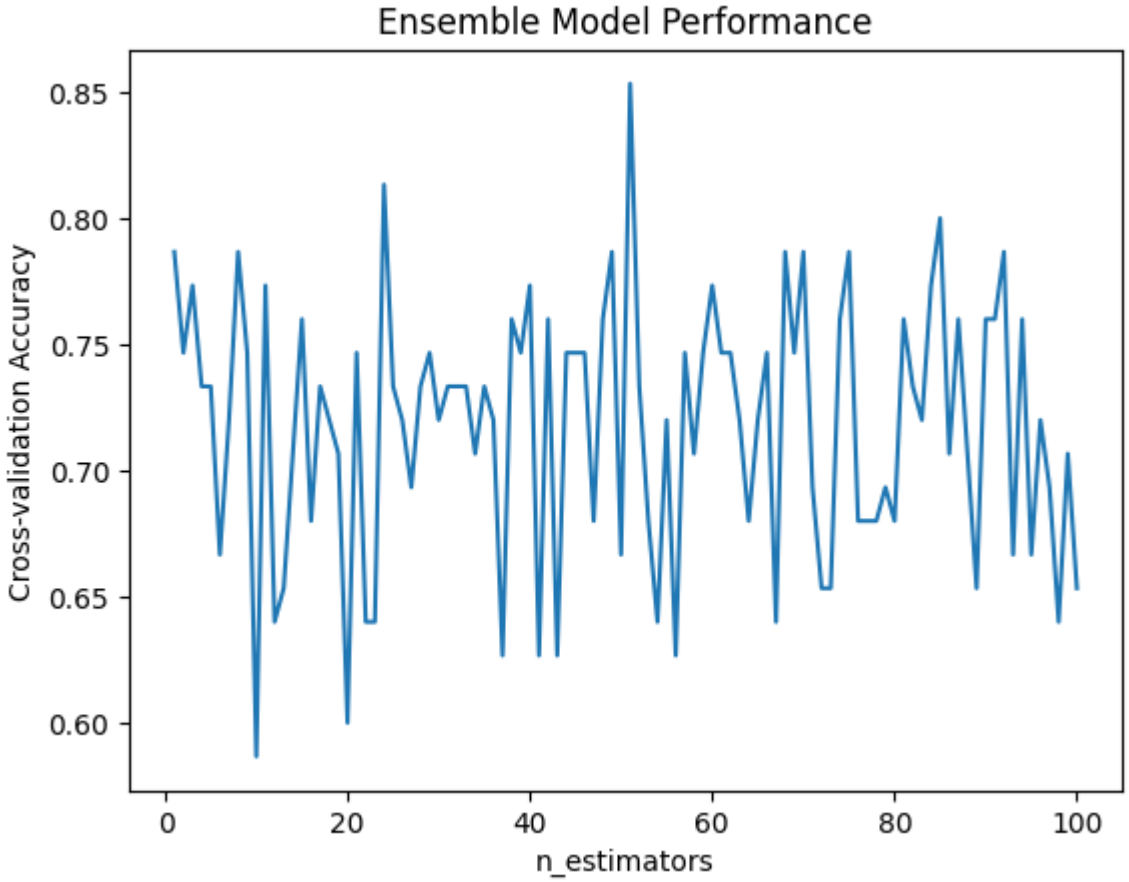

```
Mean Accuracy: 0.6548
Standard Deviation of Accuracy: 0.057407548864819746
Max Accuracy: 0.8133333333333334
Min Accuracy: 0.41333333333333333
```

1.2.1.2 Tree max_depth = 3

In [ ]:
```
model=ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier(max_depth=3))
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
```

```
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Accuracy: 0.7178666666666667
Standard Deviation of Accuracy: 0.05114580465036535
Max Accuracy: 0.8533333333333334
Min Accuracy: 0.5866666666666667
```

Use HyperOpt (Bayesian Optimization) to test Adaboost

In [ ]:

```
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials

# Define objective function
def objective(params):
    max_depth = int(params['max_depth'])  # Convert to integer

    model = ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier(max_depth=max_dep

    # Call your EnsembleModels function
    accs = EnsembleModels(model, Max_n_estimators=40)

    # Return the objective value (1 - mean accuracy) as Hyperopt minimizes
    return {'loss': 1 - np.mean(accs), 'status': STATUS_OK, 'max_depth': int(max_dept

# Define search space
space = {
```

```
    'max_depth': hp.quniform('max_depth', 1, 10, 1)  # 這裡假設 max_depth 可以在 1 到 10
}

# Create Trials object to track evaluation results
trials = Trials()


# Use Hyperopt's fmin function for optimization
best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=40, trials=trials)

depth_loss_dict = {}

for result in trials.results:
    max_depth = int(result['max_depth'])
    loss = result['loss']
    depth_loss_dict[max_depth] = loss

for max_depth in range(1, 11):
    loss = depth_loss_dict.get(max_depth, None)
    if loss is not None:
        print(f"max_depth: {max_depth}, loss: {loss}")
    else:
        print(f"max_depth: {max_depth}, loss: N/A (not evaluated)")
```

```
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
100%|██████████| 40/40 [06:28<00:00,  9.72s/trial, best loss: 0.2743333333333333]
max_depth: 1, loss: 0.3550000000000001
max_depth: 2, loss: 0.29133333333333344
max_depth: 3, loss: 0.28200000000000003
max_depth: 4, loss: 0.2886666666666666
max_depth: 5, loss: 0.29800000000000004
max_depth: 6, loss: 0.29766666666666663
max_depth: 7, loss: 0.29466666666666663
max_depth: 8, loss: 0.30499999999999994
max_depth: 9, loss: 0.31866666666666676
max_depth: 10, loss: 0.31766666666666665
```

In [ ]:
```python
for max_depth in range(1, 11):
    loss = depth_loss_dict.get(max_depth, None)
    if loss is not None:
        print(f"max_depth: {max_depth}, loss: {loss}")
    else:
        print(f"max_depth: {max_depth}, loss: N/A (not evaluated)")
```

```
max_depth: 1, loss: 0.3550000000000001
max_depth: 2, loss: 0.29133333333333344
max_depth: 3, loss: 0.28200000000000003
max_depth: 4, loss: 0.2886666666666666
max_depth: 5, loss: 0.29800000000000004
max_depth: 6, loss: 0.29766666666666663
max_depth: 7, loss: 0.29466666666666663
max_depth: 8, loss: 0.30499999999999994
max_depth: 9, loss: 0.31866666666666676
max_depth: 10, loss: 0.31766666666666665
```

1.2.2 Gradient Boosting

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

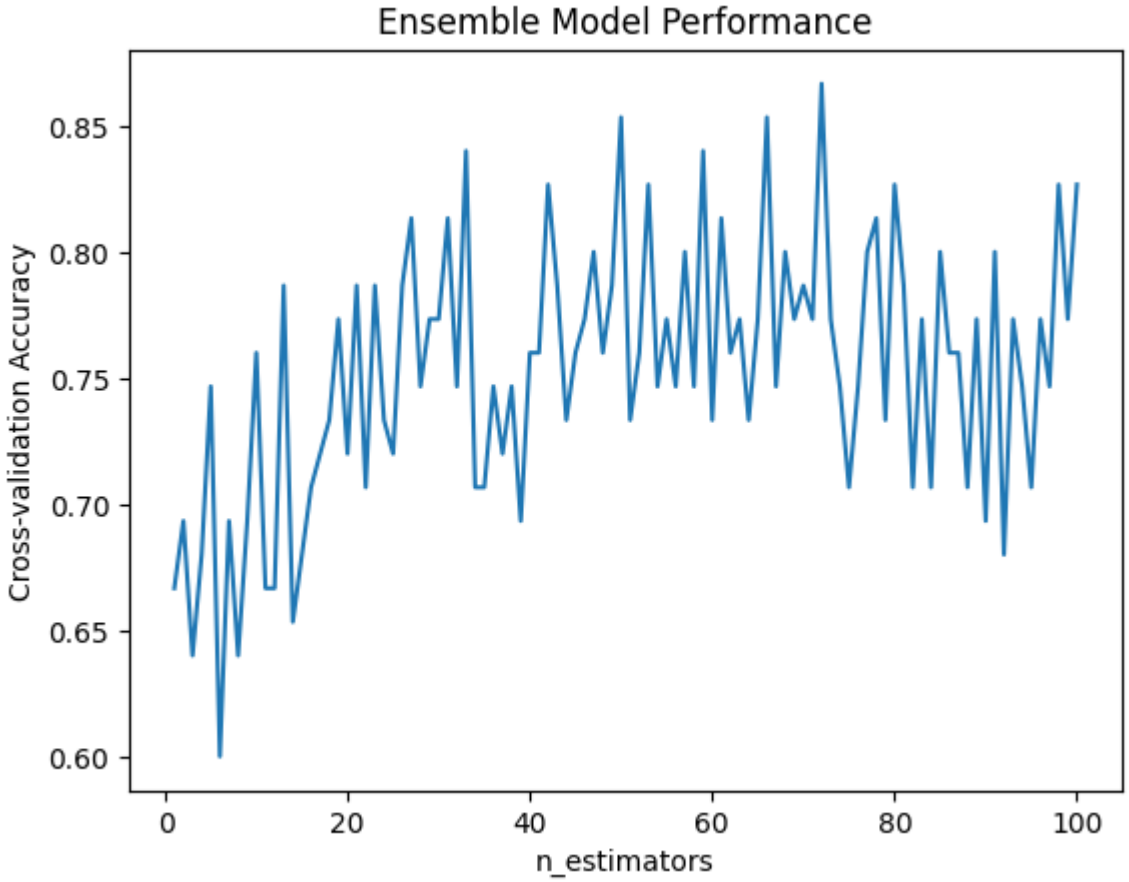
1.2.2.1 Scikit-learn's Gradient Tree Boosting

1.2.2.1.1 Tree max_depth = 1

In [ ]:
```python
model=ensemble.GradientBoostingClassifier(max_depth=1)
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
```

```
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()
# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Accuracy: 0.7529333333333333
Standard Deviation of Accuracy: 0.05106701479428771
Max Accuracy: 0.8666666666666667
Min Accuracy: 0.6
```

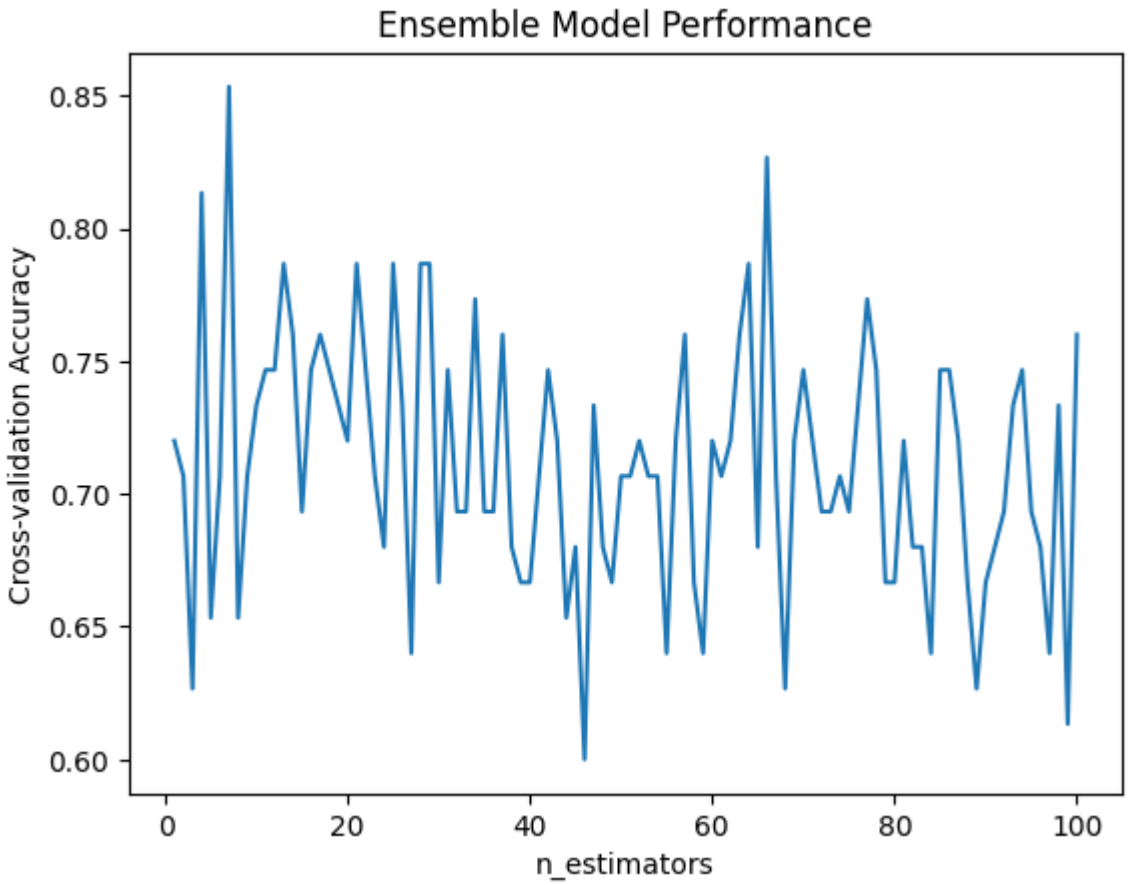
1.2.2.1.2 Tree max_depth = 3

In [ ]:

```
model=ensemble.GradientBoostingClassifier(max_depth=3)
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
```

```
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Accuracy: 0.7112
Standard Deviation of Accuracy: 0.04778544641299156
Max Accuracy: 0.8533333333333334
Min Accuracy: 0.6
```

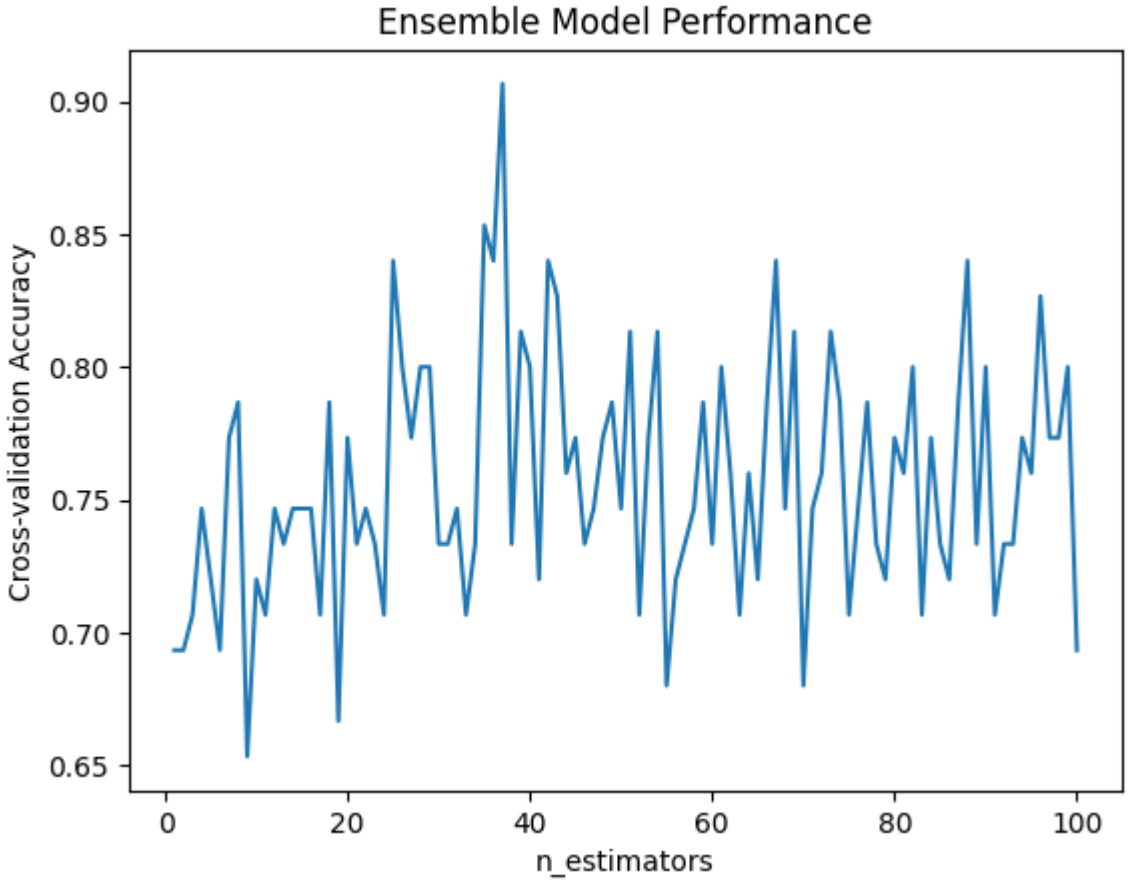
1.2.2.2 XGBoost (eXtreme Gradient Boosting)

1.2.2.2.1 Tree max_depth = 1

In [ ]:
```
model=xgboost.XGBClassifier(max_depth=1)
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
```

```
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()

# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Accuracy: 0.7570666666666664
Standard Deviation of Accuracy: 0.045629620009619
Max Accuracy: 0.9066666666666666
Min Accuracy: 0.6533333333333333
```

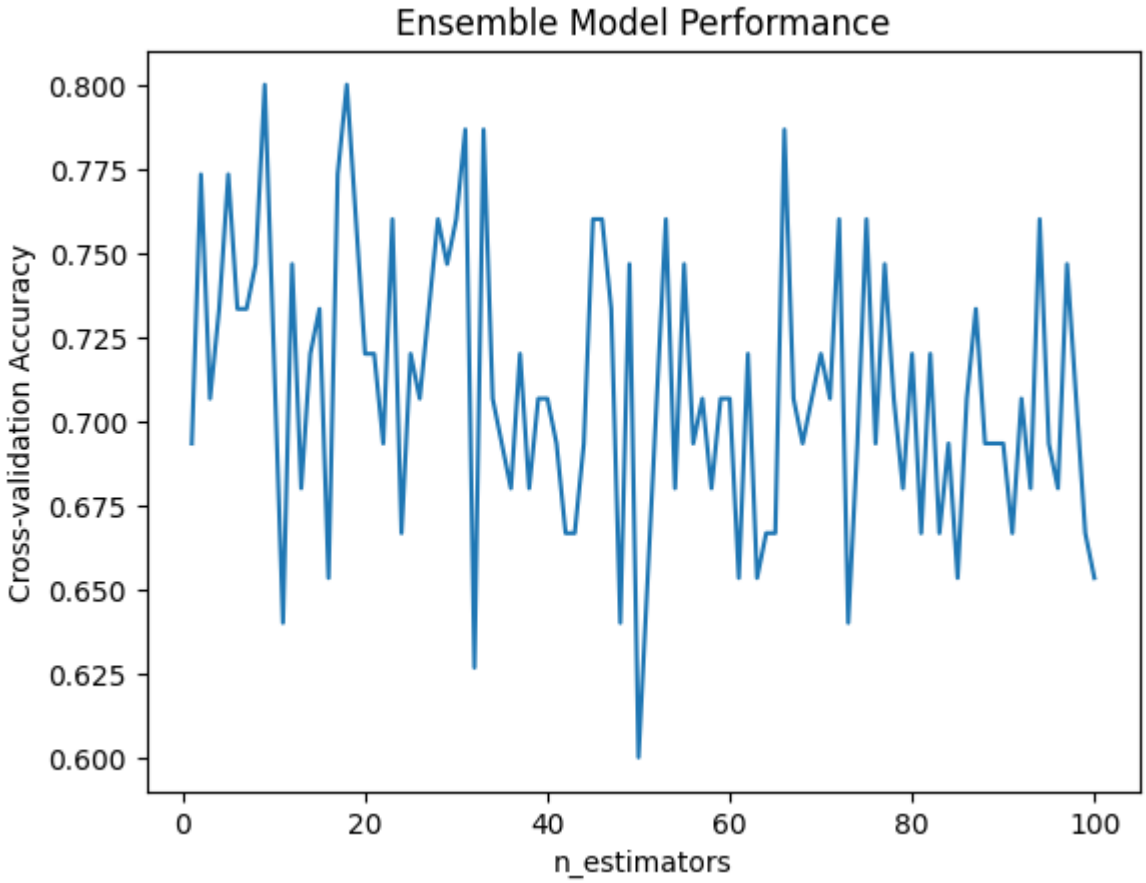
1.2.2.2.2 Tree max_depth = 3

In [ ]:

```
model=xgboost.XGBClassifier(max_depth=3)
# Assume EnsembleModels returns accs list
accs = EnsembleModels(model, 100)

# 計算統計描述指標
mean_accuracy = np.mean(accs)
std_accuracy = np.std(accs)
max_accuracy = np.max(accs)
min_accuracy = np.min(accs)

# 繪製準確度隨 n_estimators 變化的圖表
plt.plot(range(1, 101), accs)
plt.xlabel('n_estimators')
plt.ylabel('Cross-validation Accuracy')
plt.title('Ensemble Model Performance')
plt.show()
```

```
# 統計描述
print(f'Mean Accuracy: {mean_accuracy}')
print(f'Standard Deviation of Accuracy: {std_accuracy}')
print(f'Max Accuracy: {max_accuracy}')
print(f'Min Accuracy: {min_accuracy}')
```

```
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 3
2 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 8
9 90 91 92 93 94 95 96 97 98 99 100
```




```
Mean Accuracy: 0.7096000000000001
Standard Deviation of Accuracy: 0.0407738478275802
Max Accuracy: 0.8
Min Accuracy: 0.6
```

2 根據以上的觀察回答以下的問題 (6 分)

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)

- 從平均值、最大值、最小值上來看，**複雜模型的正確率是的確比簡單模型正確率還來得高的**。（而在 Precision 的統計描述上我們也能看到的確複雜模型的 precision是高於簡單模型的，所以在此，並不是由於我們選錯了統計標的。）
- 在此也利用了 **hyperopt** ，測試了從一到十的 max_depth，哪個模型正確率越高，結果發現 max_depth為三時，模型正確率是最高的，再更複雜也不會變好。

- 要解釋複雜模型（1.1.2）的正確率為何比簡單模型（1.1.1）的正確率還要高，是因為 Bagging 這個 ensemble model，對於每個 Bootstrap 子集，會使用相同的學習算法進行模型的訓練。最後，再將所有訓練好的模型進行集成，平均或投票方式，形成最終的模型。

- 在這個過程中， Bagging 主要通過平均多個模型的預測結果，來降低模型的變異數，提高整體模型的 robustness。此處，**每棵樹都是獨立建立的，所以，即使樹太深而產生了錯誤，也會在最後平均的時後消除這個效果**，因此比較不容易 overfitting。在測試中，一直到 max_depth 超過4之後，loss 才會緩慢的上升。

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)

原本猜想會不會是與我們使用 StratifiedShuffle 有關，因為我們的鳶尾花模型在三個類別上數量相同，好像沒必要使用分層的隨機分配。但在測試後（改使用簡單隨機分配），發現結果依舊。

In [ ]:
```
# @title
# sss=model_selection.ShuffleSplit(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_estimat
#     for n in range(1,Max_n_estimators+1):
#         print(n,end=' ') # showing progress
#         acc=[] # cross-validation accuracies of the ensemble model w/ n_estimators=
#         for train_index, test_index in sss.split(X, Y): # 5-fold cross-validation o
#             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-training
#             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 accuracies ac
#     return(accs)
```

- **在 Adaptive Boosting 模型上，複雜模型（1.2.1.2）的正確率比簡單模型（1.2.1.1）還高。而在 Gradient Boosting 的模型上，包含 Scikit-learn's Gradient Tree Boosting 以及 eXtreme Gradient Boosting，則都是簡單模型的正確率比複雜模型高。**

- Boosting 和 Bagging 比較大的不同，是他主要專注在錯誤預測的樣本，透過不斷的迭代增強下一個子模型，像是 Gradiant Boosting ，他會在每一步優化前一個弱學習模型的 residual，**模型的訓練過程是依賴前一個子模型的結果的**。這樣的方法，在當每個決策樹的深度變深後會出現問題，因為樹的深度越深，每個弱模型變得不再簡單，容易出現 overfitting，而這個效應並不會像 Bagging 一樣被平均的方式給消除，**他會不斷影響下一個子模型的預測**，最後導致簡單模型（深度為一）的正確率比複雜模型（深度為三）高。

- 但 Adaptive Boosting 的情況就不太一樣，他不是調整 residual ，而是調整樣本權重。在每一輪迭代中，將先前子模型錯誤預測的樣本賦予更高的權重，讓新的子模型更關注這些難以預測的樣本。AdaBoost 的核心思想是通過不斷關注先前模型錯誤分類的實例來提高模型性能。**由於Adaptive Boosting 在處理每個子模型時，並不像 Gradient Boosting 一樣精細**，比起 Gradient Boosting ，受 overfitting 的影響比較小，也因此在深度到3之後的正確率還比1來得高。在後來的測試中也發現，正確率是在深度超過三之後才緩慢上升的。

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)

- Boosting模型是透過不斷的迭代前個弱學習器而得到的強模型器，**模型的訓練過程是依賴前一個子模型的結果的**，因此照理來說，隨著estimator越來越後面後，模型的正確率就應該上升，也就是為什麼簡單模型（深度為一）正確率大致上會隨著n_estimators數目變多而增加。
- 但正如上一題所陳述的，當樹的深度越深，每個弱模型變得不再簡單，容易出現 overfitting，此時當estimators越後面時，受到的偏差影響便約多，導致複雜的Boosting模型（深度為三）的正確率大致上不會隨著n_estimators數目變多而增加。

- 而在 Bagging 模型中，**每個樹是彼此獨立的**，他是透過平均每個樹（弱模型），而得來的強模型。因此正確率不應隨著n_estimators數目變多而有上升或下降的趨勢，不論深度為何。