```
In [39]: #Train the model using the train-validation set.
         clf_svm.fit(X_train1, y_train1)

Out[39]: SVC(C=50, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

```
In [40]: # Finally, report the test accuracy.
         clf_test= SVC(kernel='linear', C=5)
         clf_test.fit(X_train1, y_train1)
         scores_test=clf_test.score(X_test1, y_test1)
         print(scores_test)

         0.9666666666666667
```

So the test accuracy is 0.9666666666666667 on the test set when C is 5.It is the best C parameter for the SVM classifier. I set up the maximum depth to be 5 based on speed of convergence and generalisation The greater of C, the less tolerance for error.I also tried C=2,but it may also have overfitting problem duo to the test score is 100%. The model's accuracy is 0.9666666666666667 on the test set when C=5.

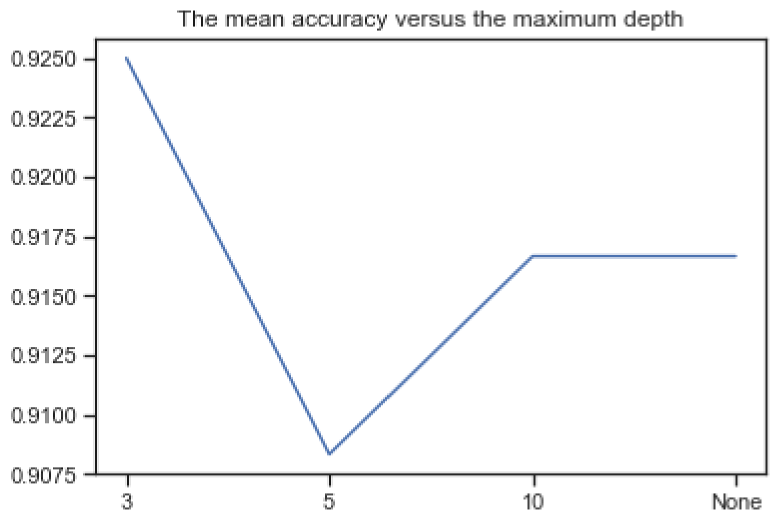
Question 4: Tree-based Classifiers

In [41]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(data.data, data.target, test_si
ze=0.2, random_state = 42)


#For decision tree: max depth: {3, 5, 10, None (grow until the end)}
from sklearn.tree import DecisionTreeClassifier

Depth=[3, 5, 10, None]

cache2=[] #cache to store the result
for depth in Depth:
    clf_dtree = DecisionTreeClassifier(max_depth=depth)
    scores_dtree = cross_val_score(clf_dtree, X_train2, y_train2, cv=10)
    cache2.append(np.mean(scores_dtree))

    print(scores_dtree)

cache2
```

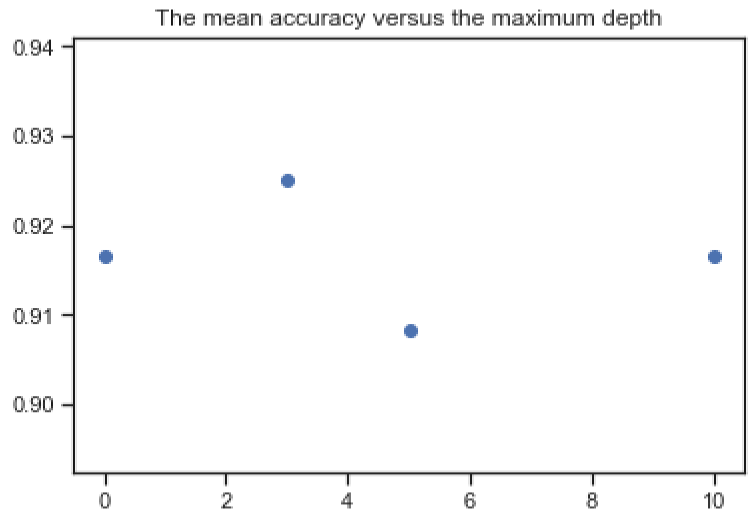
```
[0.91666667 1.         1.         1.         0.75       0.83333333
 1.         1.         0.83333333 0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 0.83333333 0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 0.91666667 0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 0.91666667 0.91666667]
```

Out[41]:
```
[0.925, 0.9083333333333332, 0.9166666666666666, 0.9166666666666666]
```

In [42]:
```
# Plot the mean accuracy versus the maximum depth.

plt.title('The mean accuracy versus the maximum depth')
temp=[3, 5, 10, "None"] #Adjust the Depth array to print "None" on the plot graph
plt.plot(temp,cache2)
plt.show()
```



In [43]:
```
temp1=[3, 5, 10, 0]#Adjust the Depth array to print "None" on scatter plot graph
plt.title('The mean accuracy versus the maximum depth')
plt.scatter(temp1,cache2)
plt.show()
```



In [44]:
```
#Train the model using the train-validation set.
clf_dtree_test= DecisionTreeClassifier(max_depth=3)
clf_dtree_test.fit(X_train2, y_train2)
```

Out[44]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [45]:
```
# Finally, report the test accuracy.
scores_test_dtree=clf_dtree_test.score(X_test2, y_test2)
print(scores_test_dtree)
```

```
1.0
```

According to the the above result, when the maximum depth is equal to 3, the accuracy is the highest and it is 1.0 on the test set. So the best maximum depth in term of decision tree classifier accuracy is 10.

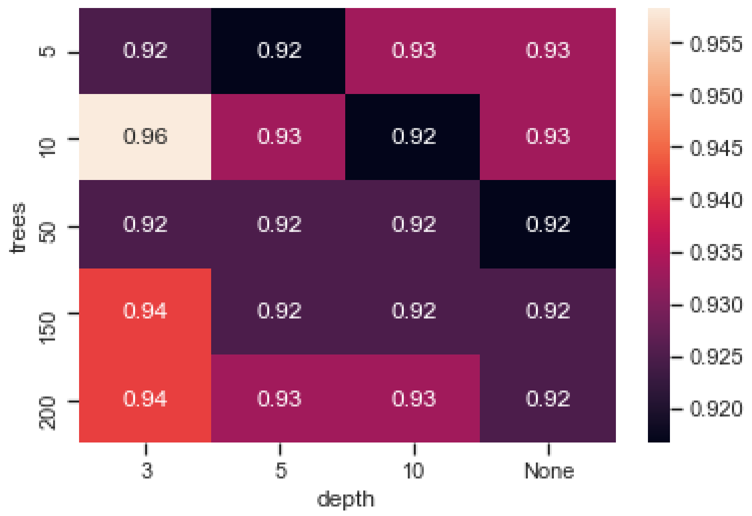
In [46]:
```
#For random forest:
#• number of trees: {5, 10, 50, 150, 200}
#• max depth: {3, 5, 10, None (grow until the end)}

from sklearn.ensemble import RandomForestClassifier

numberoftree=[5, 10, 50, 150, 200]

cache3=[] #cache to store the result
totalcache=[]
for number in numberoftree:
    for i in range(len(Depth)):
        clf_rforest= RandomForestClassifier(max_depth=Depth[i], n_estimators=number)
        scores_rforest = cross_val_score(clf_rforest, X_train2, y_train2, cv=10)
        mean_scores_rforest=np.mean(scores_rforest)
        cache3.append(mean_scores_rforest)
        print(scores_rforest)
        print(number,Depth[i],mean_scores_rforest)
    totalcache.append(cache3)
    cache3=[]

totalcache
```

```
[1.         1.         0.91666667 1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
5 3 0.9249999999999998
[1.         1.         0.91666667 1.         0.66666667 0.83333333
 1.         0.91666667 0.91666667 0.91666667]
5 5 0.9166666666666666
[1.         1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
5 10 0.9333333333333332
[0.91666667 1.         0.91666667 1.         0.66666667 1.
 1.         0.91666667 1.         0.91666667]
5 None 0.9333333333333332
[1.         1.         1.         1.         0.75       0.91666667
 1.         1.         1.         0.91666667]
10 3 0.9583333333333333
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
10 5 0.9333333333333332
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
10 10 0.9166666666666666
[0.91666667 1.         1.         1.         0.66666667 0.91666667
 1.         0.91666667 1.         0.91666667]
10 None 0.9333333333333332
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
50 3 0.9249999999999998
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
50 5 0.9249999999999998
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
50 10 0.9249999999999998
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
50 None 0.9166666666666666
[0.91666667 1.         1.         1.         0.66666667 0.91666667
 1.         1.         1.         0.91666667]
150 3 0.9416666666666667
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
150 5 0.9249999999999998
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
150 10 0.9249999999999998
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
150 None 0.9249999999999998
[0.91666667 1.         1.         1.         0.66666667 0.91666667
 1.         1.         1.         0.91666667]
200 3 0.9416666666666667
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
200 5 0.9333333333333332
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
200 10 0.9333333333333332
```

```
[0.91666667 1.         0.91666667 1.         0.66666667 0.83333333
 1.         1.         1.         0.91666667]
200 None 0.9249999999999998
```

Out[46]:
```
[[0.9249999999999998,
  0.9166666666666666,
  0.9333333333333332,
  0.9333333333333332],
 [0.9583333333333333,
  0.9333333333333332,
  0.9166666666666666,
  0.9333333333333332],
 [0.9249999999999998,
  0.9249999999999998,
  0.9249999999999998,
  0.9166666666666666],
 [0.9416666666666667,
  0.9249999999999998,
  0.9249999999999998,
  0.9249999999999998],
 [0.9416666666666667,
  0.9333333333333332,
  0.9333333333333332,
  0.9249999999999998]]
```

In [47]:
```python
# Plot a heat plot.
temp2=[3, 5, 10,'None'] #Adjust the Depth array to print "None" on the heatmap
ax = sns.heatmap(totalcache,annot=True,xticklabels=temp2, yticklabels=numberoftree)
plt.xlabel('depth')
plt.ylabel('trees');
```



I set up the maximum depth to be 3 and the number of tree to be 150 based on speed of convergence and generalisation because they are intermediate values comparing other parameter combinations which have the highest accuracy. The more number of the tree and deeper tree depth can make the model more complex and consume too much to the computer, it is not good to the complexity(big O) and speed of convergence. It may also have overfitting problem. If I choose a too small maximum depth and the number of tree, the model may not have good robustness and generalisation when a large amount data input to the model.

In [48]:
```
#Train the model using the train-validation set.
clf_rforest_test= RandomForestClassifier(max_depth=3, n_estimators=150)
clf_rforest_test.fit(X_train2, y_train2)
```

Out[48]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=3, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=150,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [49]:
```
# Finally, report the test accuracy.
scores_test_rforest= clf_rforest_test.score(X_test2, y_test2)

print(scores_test_rforest)
```

```
1.0
```

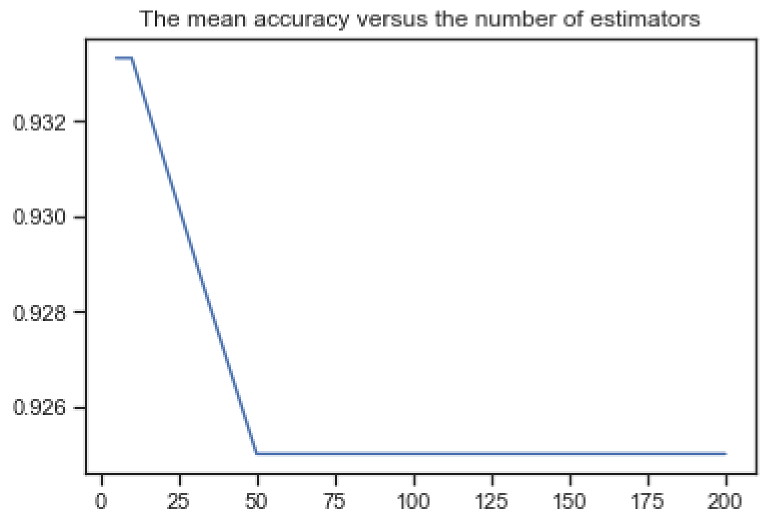
According to the the above result, when the number of tree =5 and maximum depth=none,the number of tree=50 and maximum depth=3, the number of tree=150 and maximum depth=3, the number of tree=200 and maximum depth=3, the accuracy is the highest. I set up the maximum depth to be 3 and the number of tree to be 150 based on speed of convergence and generalisation because they are intermediate values comparing other parameter combinations which have the highest accuracy. The more number of the tree and deeper tree depth can make the model more complex and consume too much to the computer, it is not good to the complexity(big O) and speed of convergence. It may also have overfitting problem. If I choose a too small maximum depth and the number of tree, the model may not have good robustness and generalisation when a large amount data input to the model. Its accuracy is 100% on the test set.

In [50]:
```
#For Gradient Tree Boosting
#• number of estimators: {5, 10, 50, 150, 200}
from sklearn.ensemble import GradientBoostingClassifier

estimators=[5, 10, 50, 150, 200]

cache4=[] #cache to store the result
for estimator in estimators:
    clf_estimators= GradientBoostingClassifier(n_estimators=estimator)
    scores_estimators = cross_val_score(clf_estimators, X_train2, y_train2, cv=10)
    cache4.append(np.mean(scores_estimators))

    print(scores_estimators)

cache4
```

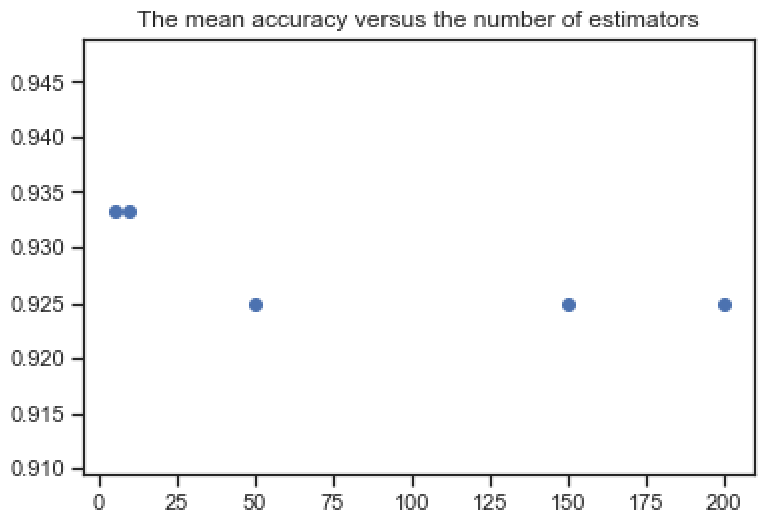
```
[1.         1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
[1.         1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
```

Out[50]:
```
[0.9333333333333332,
 0.9333333333333332,
 0.9249999999999998,
 0.9249999999999998,
 0.9249999999999998]
```

In [51]:
```
# Plot the mean accuracy versus the number of estimators.

plt.title('The mean accuracy versus the number of estimators')
plt.plot(estimators,cache4)
plt.show()
```



In [52]:
```
plt.title('The mean accuracy versus the number of estimators')
plt.scatter(estimators,cache4)
plt.show()
```



I picked 10 based on speed of convergence and generalisation. 10 is an intermediate value comparing the other one which has the highest accuracy too. The more the number of boosting stages can make the model more complex and consume too much to the computer, it is not good to the complexity(big O) and speed of convergence. The model may also have overfitting problem.If I choose a too small maximum depth and the number of tree, the model may not have good robustness and generalisation when a large amount data input to the model.

In [53]:
```
#Train the model using the train-validation set.
clf_estimators_test= GradientBoostingClassifier(n_estimators=10)
clf_estimators_test.fit(X_train2, y_train2)
```

Out[53]:
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=10,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [54]:
```
# Finally, report the test accuracy.
scores_test_estimators= clf_estimators_test.score(X_test2, y_test2)

print(scores_test_estimators)
```

```
1.0
```

According to the the above result, when the number of estimators is equal to 5 and 10, the accuracy is the highest. I set up the n_estimators to be 10 and its accuracy is 100% on the test set. I picked 10 based on speed of convergence and generalisation. 10 is an intermediate value comparing the other one which has the highest accuracy too. The more the number of boosting stages can make the model more complex and consume too much to the computer, it is not good to the complexity(big O) and speed of convergence. The model may also have overfitting problem.If I choose a too small maximum depth and the number of tree, the model may not have good robustness and generalisation when a large amount data input to the model.

Question 5: Analysis

1. Explain why you had to split the dataset into train and test sets?

We train machine learning models to use the existing data to predict unknown data. Generally, we call the model's ability to predict unknown data known as the generalization ability. In order to evaluate the generalization ability of a model, we usually divide the data into a training set and a test set. The training set is used to train the model, and the test set is used to evaluate the model's generalization ability. We can use the data from the training set to train the model, and then use the error on the test set as the generalized error of the final model in dealing with real-world scenarios. With the test set, we want to verify the final effect of the model. Just calculate the error on the test set for the trained model, and we can consider this error to be an approximation of the generalized error. We only need to let our trained model The minimum error on the test set is sufficient. Thus, we had to split the dataset into train and test sets

1. Explain why when finding the best parameters for KNN you didn’t evaluate directly on the test set and had to use a validation test.

In machine learning, when developing a model, it is always necessary to adjust the parameters of the model, such as changing parameters for KNN to get better accuracy. This adjustment process needs to provide a feedback signal on the trained model by validation test data. Modify the model and parameters is the role of the validation set, which will also cause information leakage of the validation set. The more feedback, the more information is leaked, that is, the model understands the validation set more clearly, which will eventually cause the model to overfit on the validation set. At this time, you need a dataset that is completely new to the model,so the test set is used to measure the quality of the model.