Assignment 1

ECE657A

XIYAO LIU

20483947

Load Datasets ¶

In [22]:
```
from sklearn import datasets
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

data = load_iris()
data.keys()
```

Out[22]:
```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [23]:
```
data.data
```

Out[23]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
```

```
       [4.9,  2.4,  3.3,  1. ],
       [6.6,  2.9,  4.6,  1.3],
       [5.2,  2.7,  3.9,  1.4],
       [5. ,  2. ,  3.5,  1. ],
       [5.9,  3. ,  4.2,  1.5],
       [6. ,  2.2,  4. ,  1. ],
       [6.1,  2.9,  4.7,  1.4],
       [5.6,  2.9,  3.6,  1.3],
       [6.7,  3.1,  4.4,  1.4],
       [5.6,  3. ,  4.5,  1.5],
       [5.8,  2.7,  4.1,  1. ],
       [6.2,  2.2,  4.5,  1.5],
       [5.6,  2.5,  3.9,  1.1],
       [5.9,  3.2,  4.8,  1.8],
       [6.1,  2.8,  4. ,  1.3],
       [6.3,  2.5,  4.9,  1.5],
       [6.1,  2.8,  4.7,  1.2],
       [6.4,  2.9,  4.3,  1.3],
       [6.6,  3. ,  4.4,  1.4],
       [6.8,  2.8,  4.8,  1.4],
       [6.7,  3. ,  5. ,  1.7],
       [6. ,  2.9,  4.5,  1.5],
       [5.7,  2.6,  3.5,  1. ],
       [5.5,  2.4,  3.8,  1.1],
       [5.5,  2.4,  3.7,  1. ],
       [5.8,  2.7,  3.9,  1.2],
       [6. ,  2.7,  5.1,  1.6],
       [5.4,  3. ,  4.5,  1.5],
       [6. ,  3.4,  4.5,  1.6],
       [6.7,  3.1,  4.7,  1.5],
       [6.3,  2.3,  4.4,  1.3],
       [5.6,  3. ,  4.1,  1.3],
       [5.5,  2.5,  4. ,  1.3],
       [5.5,  2.6,  4.4,  1.2],
       [6.1,  3. ,  4.6,  1.4],
       [5.8,  2.6,  4. ,  1.2],
       [5. ,  2.3,  3.3,  1. ],
       [5.6,  2.7,  4.2,  1.3],
       [5.7,  3. ,  4.2,  1.2],
       [5.7,  2.9,  4.2,  1.3],
       [6.2,  2.9,  4.3,  1.3],
       [5.1,  2.5,  3. ,  1.1],
       [5.7,  2.8,  4.1,  1.3],
       [6.3,  3.3,  6. ,  2.5],
       [5.8,  2.7,  5.1,  1.9],
       [7.1,  3. ,  5.9,  2.1],
       [6.3,  2.9,  5.6,  1.8],
       [6.5,  3. ,  5.8,  2.2],
       [7.6,  3. ,  6.6,  2.1],
       [4.9,  2.5,  4.5,  1.7],
       [7.3,  2.9,  6.3,  1.8],
       [6.7,  2.5,  5.8,  1.8],
       [7.2,  3.6,  6.1,  2.5],
       [6.5,  3.2,  5.1,  2. ],
       [6.4,  2.7,  5.3,  1.9],
       [6.8,  3. ,  5.5,  2.1],
       [5.7,  2.5,  5. ,  2. ],
```

```
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

In [24]:
```
data.target
```

Out[24]:
```
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, 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])
```

In [25]:
```
data.target_names
```

Out[25]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [26]:
```
data.feature_names
```

Out[26]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

Dataframe

In [27]:
```
def toDataframe():

    columns = np.append(data.feature_names,['target'])
    bodydata = np.append(data.data, data.target[:,None], axis=1)
    index = pd.RangeIndex(start=1, stop=len(data.data)+1, step=1)

    return pd.DataFrame(columns=columns, data=bodydata, index=index)


data_df=toDataframe()
data_df
```

Out[27]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
1	5.1	3.5	1.4	0.2	0.0
2	4.9	3.0	1.4	0.2	0.0
3	4.7	3.2	1.3	0.2	0.0
4	4.6	3.1	1.5	0.2	0.0
5	5.0	3.6	1.4	0.2	0.0
6	5.4	3.9	1.7	0.4	0.0
7	4.6	3.4	1.4	0.3	0.0
8	5.0	3.4	1.5	0.2	0.0
9	4.4	2.9	1.4	0.2	0.0
10	4.9	3.1	1.5	0.1	0.0
11	5.4	3.7	1.5	0.2	0.0
12	4.8	3.4	1.6	0.2	0.0
13	4.8	3.0	1.4	0.1	0.0
14	4.3	3.0	1.1	0.1	0.0
15	5.8	4.0	1.2	0.2	0.0
16	5.7	4.4	1.5	0.4	0.0
17	5.4	3.9	1.3	0.4	0.0
18	5.1	3.5	1.4	0.3	0.0
19	5.7	3.8	1.7	0.3	0.0
20	5.1	3.8	1.5	0.3	0.0
21	5.4	3.4	1.7	0.2	0.0
22	5.1	3.7	1.5	0.4	0.0
23	4.6	3.6	1.0	0.2	0.0
24	5.1	3.3	1.7	0.5	0.0
25	4.8	3.4	1.9	0.2	0.0
26	5.0	3.0	1.6	0.2	0.0
27	5.0	3.4	1.6	0.4	0.0
28	5.2	3.5	1.5	0.2	0.0
29	5.2	3.4	1.4	0.2	0.0
30	4.7	3.2	1.6	0.2	0.0
...	...	...	...	...	...
121	6.9	3.2	5.7	2.3	2.0
122	5.6	2.8	4.9	2.0	2.0
123	7.7	2.8	6.7	2.0	2.0
124	6.3	2.7	4.9	1.8	2.0

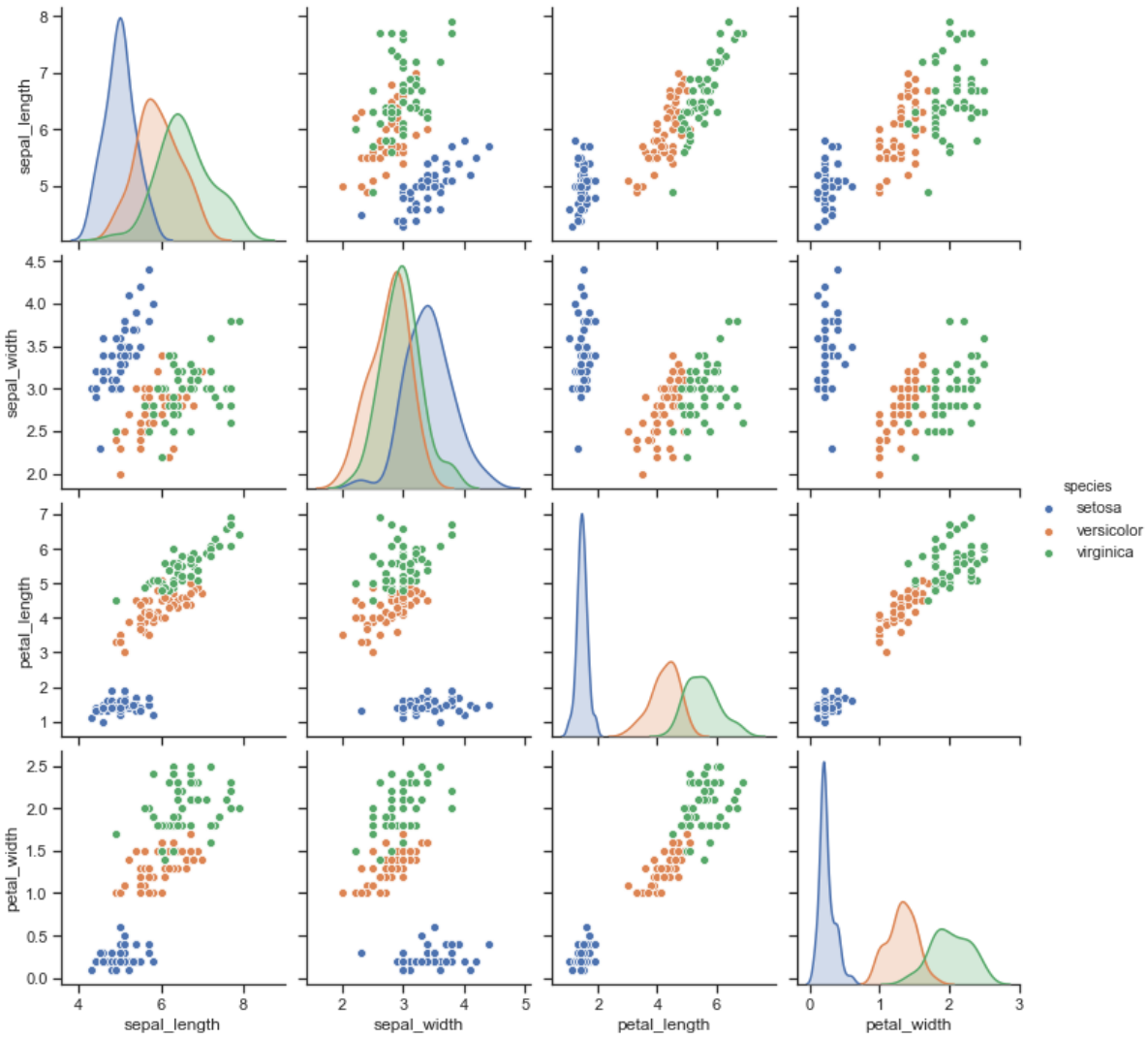
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
125	6.7	3.3	5.7	2.1	2.0
126	7.2	3.2	6.0	1.8	2.0
127	6.2	2.8	4.8	1.8	2.0
128	6.1	3.0	4.9	1.8	2.0
129	6.4	2.8	5.6	2.1	2.0
130	7.2	3.0	5.8	1.6	2.0
131	7.4	2.8	6.1	1.9	2.0
132	7.9	3.8	6.4	2.0	2.0
133	6.4	2.8	5.6	2.2	2.0
134	6.3	2.8	5.1	1.5	2.0
135	6.1	2.6	5.6	1.4	2.0
136	7.7	3.0	6.1	2.3	2.0
137	6.3	3.4	5.6	2.4	2.0
138	6.4	3.1	5.5	1.8	2.0
139	6.0	3.0	4.8	1.8	2.0
140	6.9	3.1	5.4	2.1	2.0
141	6.7	3.1	5.6	2.4	2.0
142	6.9	3.1	5.1	2.3	2.0
143	5.8	2.7	5.1	1.9	2.0
144	6.8	3.2	5.9	2.3	2.0
145	6.7	3.3	5.7	2.5	2.0
146	6.7	3.0	5.2	2.3	2.0
147	6.3	2.5	5.0	1.9	2.0
148	6.5	3.0	5.2	2.0	2.0
149	6.2	3.4	5.4	2.3	2.0
150	5.9	3.0	5.1	1.8	2.0

150 rows × 5 columns

Question1

In [28]:
```
#Question1

sns.set(style="ticks")

df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
```

Out[28]: <seaborn.axisgrid.PairGrid at 0x10ffba70>



Interpretation of the separability of the three classes in terms of different features (dimensions).

The data set contains 150 records in 3 categories, each with 50 pieces of data. Each record has 4 characteristics: sepal length, sepal width, petal length, and petal width. These 4 characteristics can be used to predict the iris flower belongs to which specific type such iris-setosa, iris-versicolor and iris-virginica. If features with objects belonging to the same category such as all setosa have similar values, we can find well-defined classfication in the data visualization. When the sepal length is fixed, the three types of Iris flowers are clearly distinguished in terms of petal width and petal length, although Versicolor and Virginica are slightly mixed. In particular, setosa's petal width and petal length are both significantly smaller than the other two. Secondly, Versicolor's petal width and petal length are smaller than those of Virginica. In the second scatter plot in the leftmost column, it can be seen that Versicolor and Virginiana are mixed with more points, and the distribution of Setosa is clear and obvious. When the sepal width is fixed, the three types of Iris flowers are well classified in terms of petal width and petal length. The distribution slightly worse on sepal length feature. Although Versicolor and Virginica have mixed separations, it is still very easy to distinguish different types of flowers. When the petal length is fixed, The three categories of Iris flowers are clearly distinguished in terms of sepal length, sepal width and petal width. Setosa's sepal width is slightly higher than the other two, and its sepal length is smaller than the other two. When the petal width is fixed, the three types of Iris flowers are clearly classified in three aspects: sepal length, sepal width, and petal length as well. Even Versicolor and Virginica still have few data points mixed together. The overall distribution is still obvious. These graphs show that a classifier trained using these functions may learn to classify various flower types reasonably.

Question 2: KNN

In [29]:
```
#Question 2: KNN

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

#1. First, divide the data into train, validation, and test sets (60%, 20%, 20%)

X_train, X_test, y_train, y_test= train_test_split(data.data,data.target, test_size=0.2
, random_state=42)

X_train, X_val, y_train, y_val= train_test_split(X_train,y_train, test_size=0.25, rando
m_state=42)
```

In [30]:
```
#2. Train the model with each classifier's default parameters. Use the train set and te
st the model on the test set. Store the accuracy of the model.
knn = KNeighborsClassifier(n_neighbors = 5) #5 is the default value

#Fit the model using X_train as training data and y_train as target values
knn.fit(X_train, y_train)
#Accuracy classification score
Accuracy_score_5=knn.score(X_test, y_test)
Accuracy_score_5
```

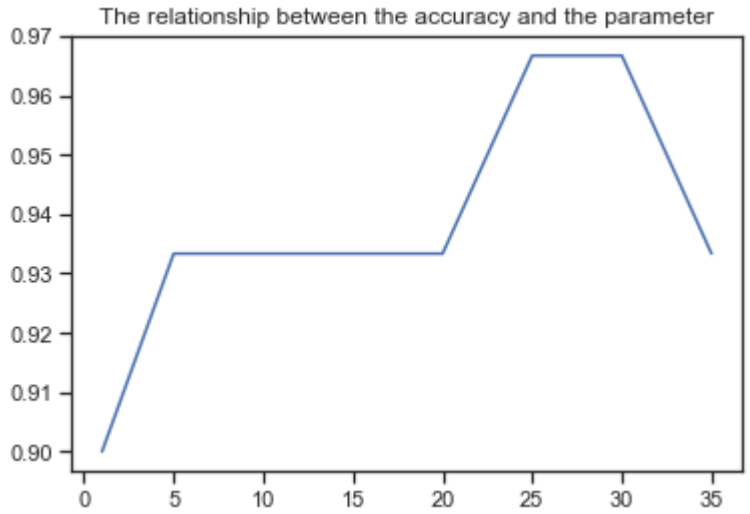
Out[30]: 0.9666666666666667

In [31]:
```
#3. find k for KNN using the cross validation data
k_parameter= [1, 5, 10, 15, 20, 25, 30, 35]
cache=[] #cache to store the result
for par in k_parameter:
    knn=KNeighborsClassifier(n_neighbors = par)
    knn.fit(X_train, y_train)
    cache.append(knn.score(X_val, y_val))

    print(knn.score(X_val, y_val))

cache
```

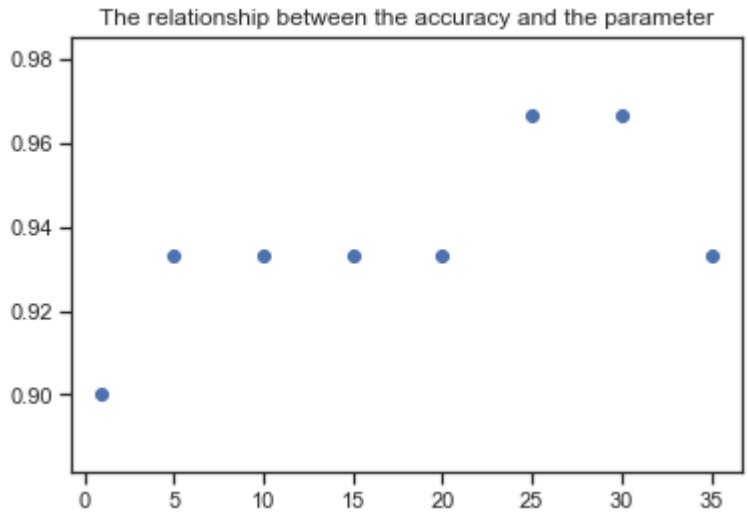
```
0.9
0.9333333333333333
0.9333333333333333
0.9333333333333333
0.9333333333333333
0.9666666666666667
0.9666666666666667
0.9333333333333333
```

Out[31]:
```
[0.9,
 0.9333333333333333,
 0.9333333333333333,
 0.9333333333333333,
 0.9333333333333333,
 0.9666666666666667,
 0.9666666666666667,
 0.9333333333333333]
```

In [32]:
```
#Plot a figure that shows the relationship between the accuracy and the parameter. Repor
t the best k in terms of classification accuracy.

plt.title('The relationship between the accuracy and the parameter')
plt.plot(k_parameter,cache)
plt.show()
```



In [33]:
```
plt.title('The relationship between the accuracy and the parameter')
plt.scatter(k_parameter,cache)
plt.show()
```



According to the two figures, when K is equal to 25 and 30, the accuracy is the highest. But based on speed of convergence and generalisation I pick 30 to be the best parameter. I set up the K to be 30 because of the accuracy. The lower value of k will predict a more locally model, and higher value of k will predict a more globally model. The smaller K 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.

In [34]:
```
#4.Using the best found parameters, train the model using the train set and test the mo
del on the test set. Report the accuracy of the model.
#Using the parameter is 25
knn25 = KNeighborsClassifier(n_neighbors = 25)
knn25.fit(X_train, y_train)
#Accuracy classification score when k=25
Accuracy_score_25=knn25.score(X_test, y_test)
Accuracy_score_25
```

Out[34]: 1.0

In [35]:
```
#Using the parameter 30
knn35 = KNeighborsClassifier(n_neighbors = 30)
knn35.fit(X_train, y_train)
#Accuracy classification score when k=35
Accuracy_score_35=knn35.score(X_test, y_test)
Accuracy_score_35
```

Out[35]: 1.0

According to the above results, when k is equal to 25 and 30 to the KNN classifier, the accuracy is the highest. But based on speed of convergence and generalisation I pick 30 to be the best parameter. I set up the K to be 30 because of the accuracy. The lower value of k will predict a more locally model, and higher value of k will predict a more globally model. The smaller K 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.Its accuracy is 100% on the test set.The accuracy of the model is 100% on the test set when K=30.

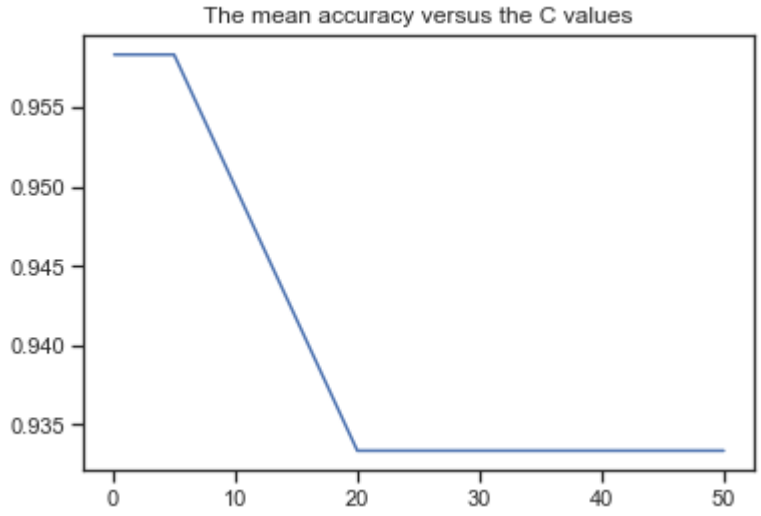
Question 3: SVM

In [36]:
```
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# First, randomly divide data into (80%, 20%) portions of train-validation and test set
s (with random state=42).
X_train1, X_test1, y_train1, y_test1= train_test_split(data.data,data.target, test_size
=0.2, random_state=42)

# apply 10-fold cross validation on the train-validation set. In every fold, 90% of dat
a is used for training and 10% of
#data for validation. For every C value, a mean accuracy of the folds is found.
C_array=[0.1, 0.5, 1, 2, 5, 10, 20, 50]

cache1=[] #cache to store the result
for c in C_array:
    clf_svm = SVC(kernel='linear', C=c)
    scores = cross_val_score(clf_svm, X_train1, y_train1, cv=10)
    cache1.append(np.mean(scores))

    print(scores)

cache1
```

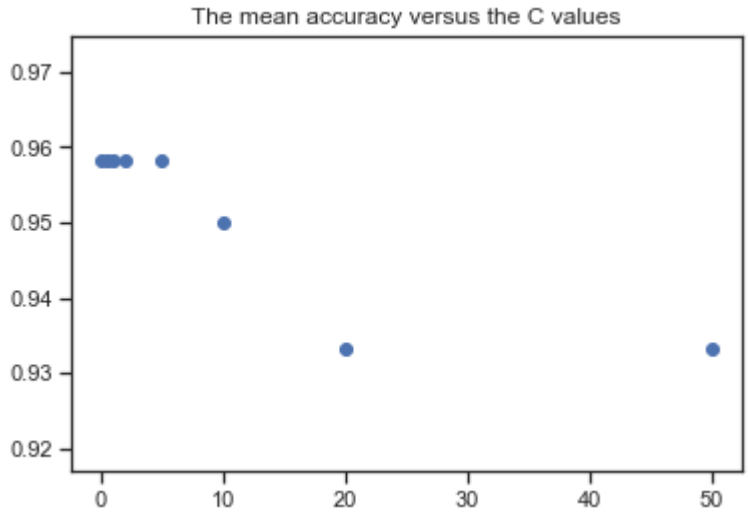
```
[1.         1.         1.         1.         0.83333333 0.83333333
 1.         1.         1.         0.91666667]
[1.         1.         0.91666667 1.         0.91666667 0.83333333
 1.         1.         1.         0.91666667]
[1.         1.         0.91666667 1.         0.91666667 0.83333333
 1.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

Out[36]:
```
[0.9583333333333334,
 0.9583333333333333,
 0.9583333333333333,
 0.9583333333333334,
 0.9583333333333334,
 0.95,
 0.9333333333333332,
 0.9333333333333332]
```

In [37]:
```
#Plot the mean accuracy versus the C values.
plt.title('The mean accuracy versus the C values')
plt.plot(C_array,cache1)
plt.show()
```



In [38]:
```
plt.title('The mean accuracy versus the C values')
plt.scatter(C_array,cache1)
plt.show()
```



According to the two figures, not only one parameter has the highest accuracy. C=5 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. But if C is too small to trend to zero, it may cause underfitting problem.

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.

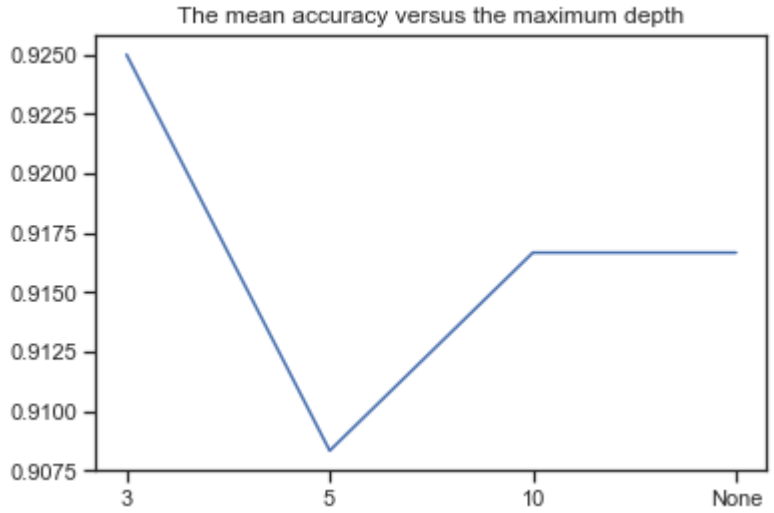
Question 4: Tree-based Classifiers

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
```

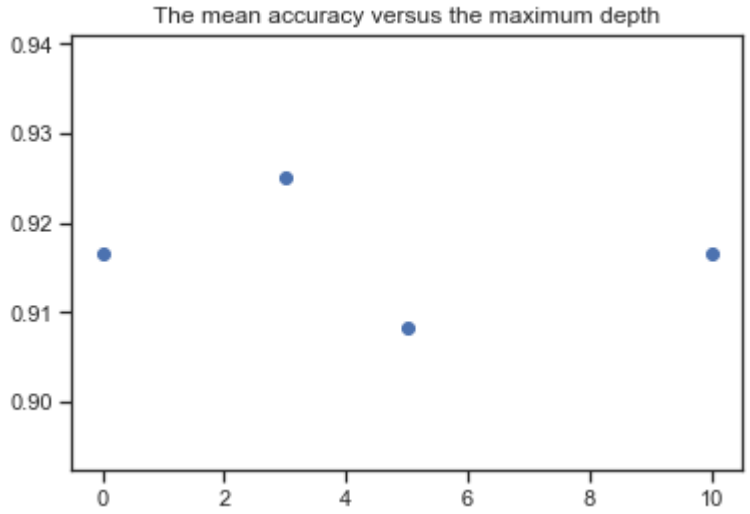
```
[0.91666667 1.         1.         1.         0.75       0.83333333
 1.         1.         0.83333333 0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 0.83333333 0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.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

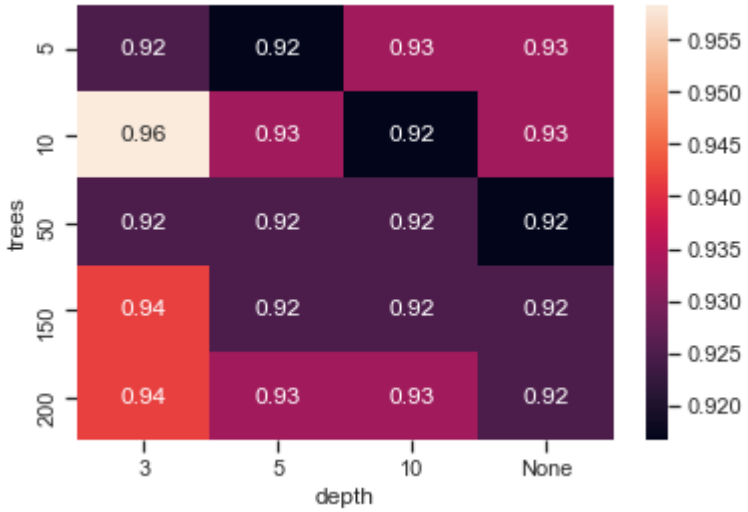
In [46]:
```
#For random forest:
#• number of trees: {5, 10, 50, 150, 200}
#• max depth: {3, 5, 10, None (grow until the end)}

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]:
```
# 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
```

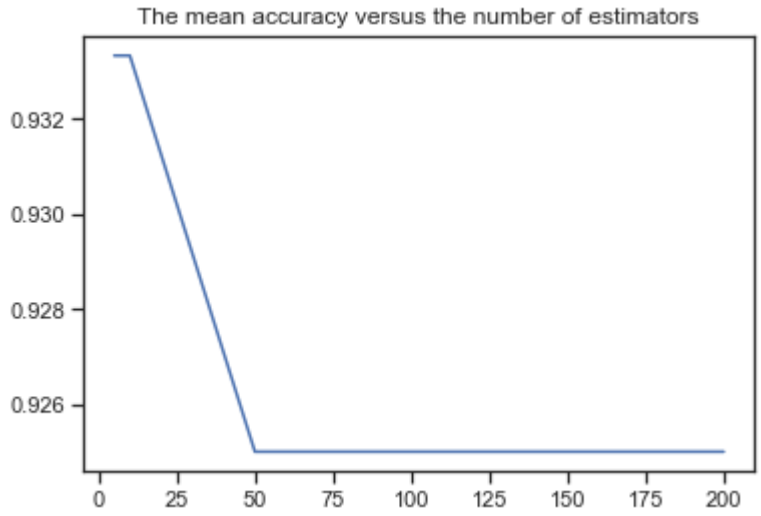
According to the the above result, when the number of tree =5 and maximum depth=none,the number of tree=50 and maximum depth=3, the number of tree=150 and maximum depth=3, the number of tree=200 and maximum depth=3, the accuracy is the highest. I set up the maximum depth to be 3 and the number of tree 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
```

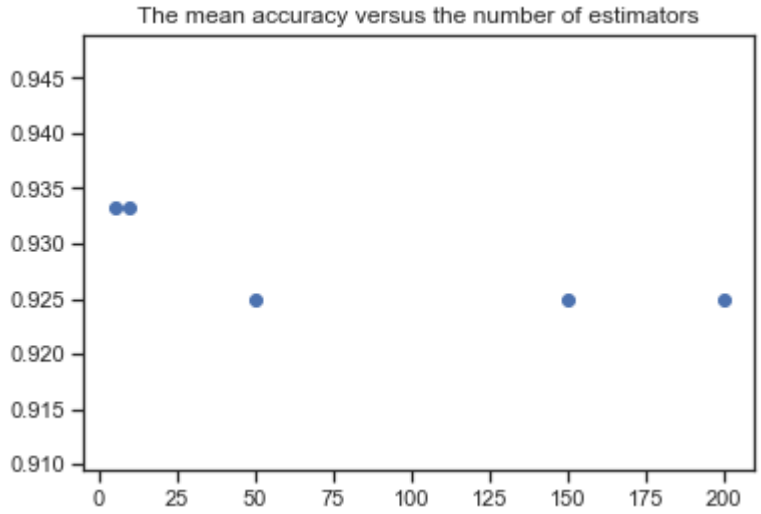
```
[1.         1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
[1.         1.         1.         1.         0.66666667 0.83333333
 1.         0.91666667 1.         0.91666667]
[0.91666667 1.         1.         1.         0.66666667 0.83333333
 1.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

1. What was the effect of changing k for KNN. Was the accuracy always affected the same way with an increase of k? Why do you think this happened? #What was the effect of changing k for KNN

If the value of K is changed to be small, the model has high complexity and is prone to overfitting. The learning estimation error will increase, and the prediction result is very sensitive to the neighboring instance points. A large K value can reduce the estimation error of learning, but the approximate error of learning will increase, and training examples that are far away from the input instance will also affect the prediction, which will cause prediction errors, and the complexity of the model will decrease as the value of k increases. If k is too small, the classification result is easily affected by noise points; if k is too large, the neighbors may contain too many points of other categories. Overal, the changing of K value in KNN is very important for the classification result. The K value is too small and the model is too complicated. The value of K is too large, resulting in fuzzy classification. Some peole use Cross Validation to choose k value, some use Bayes, and others use bootstrap.

Was the accuracy always affected the same way with an increase of k?

No.According to the above results, the testing accuracy didn't always increase the same way with an increases of k. At the beginning, the accuracy was at the peakn and then it fell as k increased.For some datasets, when the k neighboring points is large enough, the accuracy will naturally tends to the expected value of the sample. Ouer case is the same.

Why do you think this happened?

Because if k is small, the model would be more complex with high variance and low bias. It could lead to overfitting situation. But when k is too large, the model would be simple with high bias and low variance, it might cause underfitting problem. When k is increasing to be too large, the neighbors may contain too many points of other categories.So the model would not make sense and the accuracy will naturally tends to a fixed value.

1. What was the relative effect of changing the max depths for decision tree, random forests, and gradient boosting? Explain the reason for this.

The max depth defines the maximum depth that the tree is allowed to grow up.When the max depth is being larger, it will be better to fit data for the model, but it may also cause overfitting. Because the the tree is deeper, there will be more complex the decision rules. The tree works from nodes of if / else problems and get results. The deeper the tree, the more it splits, and the better it can capture information about the data. And The decision tree model is prone to produce an overly complex model when the max depth is being larger, and such a model will have poor generalization performance on data. But if the max depth is too small,it may cause underfit problem to the model and increase teating error.

1. Comment on the effect of the number of estimators for Gradient Tree Boosting and what was the relative effect performance of gradient boosting compared with random forest. Explain the reason for this.

The number of estimators is the number of boosting stages to perform.It means the number of iterations. In general, the number of estimators is too small, and the model will be easy to underfit. If n_estimators is too large, there is easy to overfit. Usually, a moderate value is selected. The default is 100 in sklearn. Because the Gradient Tree Boosting method is not just the simple application of ensemble ideas, but also the learning of residuals.

Random forest is an algorithm based on decision tree, it just uses the idea of integration to improve the classification performance of single decision tree. The main feature is that it is not easy to fall into overfitting due to the random selection of samples and features. The main steps of the random forest algorithm are: use Bootstrap to randomly select n samples from the sample set, and randomly select K attributes from all the attributes, and select the best segmentation attribute as the node to establish a classifier (CART, SVM, etc. Repeat the above m times That is, m classifiers are established, and the voting results are used to determine which type of data. The gradient tree boosting is also a decision tree model based on integrated ideas. The algorithm is to generate a new tree for each iteration(Boosting stage). And then to calculate the loss function at each training sample set, and then generate a new decision tree through the greedy strategy. Calculate the predicted value corresponding to each leaf node, and use the newly generated decision tree added to the model. The most essential difference between Gradient Tree Boosting and random forest is that each tree in Gradient Tree Boosting learns the residuals of the sum of the conclusions of all the trees, and the residual is the true value minus the predicted value.

The detailed reason are: 1: Random forests vote the results of multiple decision trees to get the final result. The training results of different trees have not been further optimized and improved, which is called the bagging algorithm. The boosting algorithm is to build a weak learner at each step of the iteration to make up for the deficiencies of the original model. 2:Gradient Tree Boosting uses the boosting algorithm, and builds a weak learner at each step of the iteration to make up for the lack of the original model. Gradient Tree Boosting is to build a learner along the direction of gradient descent through each iteration. And by setting different loss functions, it can handle various learning tasks (multi-classification, regression, etc.).

1. What does the parameter C define in the SVM classifier? What effect did you observe and why do you think this happened?

C is the regularization parameter.The parameter C defines that how much we want to avoid misclassifying for each training set.It controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights.

I observed when c was small the accuracy was higher than when it was larger in Queation 3. When c was equal to and larger than 20, The accuracy was lower and treading to be consistancy obviously.

I think this is because C is a factor that regulates the interval and accuracy. The larger the value of C, the more unwilling to abandon those outliers; the smaller the value of c, the less attention is paid to those outliers. The larger c indicates that the error cannot be tolerated and the over-fitting is easy. The C is too small, it is to underfit easily,the classifier would look for a larger-margin separating hyperplane,even if that hyperplane misclassifies more points. Whatever C is too large or too small, there will be poor generalization ability. When C approaches infinity, the problem is that samples with classification errors are not allowed, then this is a hard-margin SVM problem. (overfitting) When C approaches 0, we no longer care about whether the classification is correct, but only the larger the interval, the better, then we will not get a meaningful solution and the algorithm will not converge. (Underfitting)