Naïve Bayes Algorithm

In [1]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
```

In [2]:
```
def accuracy (y_true , y_pred):
    accuracy = np.sum(y_true == y_pred)/ len(y_true)
    return accuracy
```

In [3]:
```
data = datasets.load_breast_cancer()
x , y = data.data , data.target
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3 )
```

In [4]:
```
clf = GaussianNB()
clf.fit(x_train , y_train)
```

Out[4]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [5]:
```
y_pred = clf.predict(x_test)
```

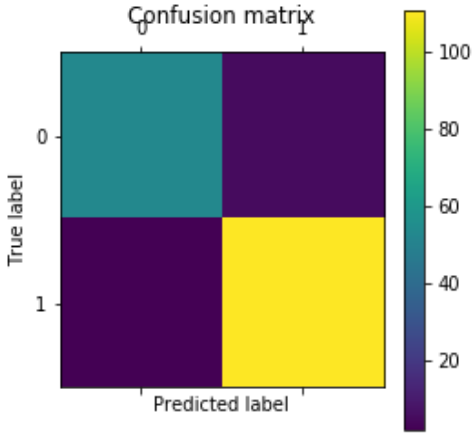
In [6]:
```
acc = accuracy(y_test,y_pred)
```

In [7]:
```
print("Accuracy:",acc)
```

```
Accuracy: 0.9590643274853801
```

In [8]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
plt.matshow(confusion_matrix)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
[[ 53   5]
 [  2 111]]
```



Decision Tree Algorithm

In [1]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
```

In [2]:
```
def accuracy (y_true , y_pred):
    accuracy = np.sum(y_true == y_pred)/ len(y_true)
    return accuracy
```

In [3]:
```
data = datasets.load_breast_cancer()
x , y = data.data , data.target
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3 )
```

In [4]:
```
clf = DecisionTreeClassifier()
clf.fit(x_train , y_train)
```

Out[4]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, 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 [5]:
```
y_pred = clf.predict(x_test)
```

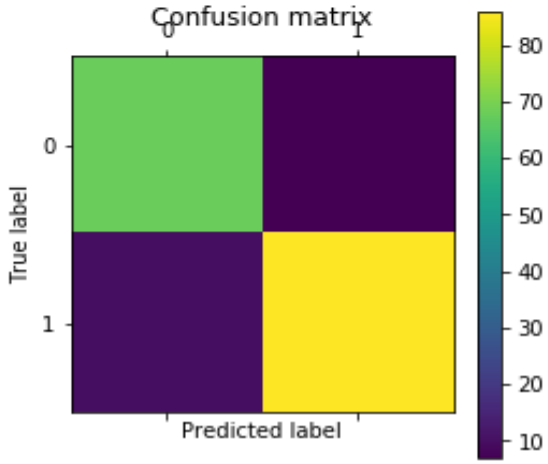
In [6]:
```
acc = accuracy(y_test,y_pred)
```

In [7]:
```
print("Accuracy:",acc)
```

```
Accuracy: 0.9005847953216374
```

In [8]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
plt.matshow(confusion_matrix)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
[[68  7]
 [10 86]]
```



Random Forest Algorithm

In [1]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
```

In [2]:
```
def accuracy (y_true , y_pred):
    accuracy = np.sum(y_true == y_pred)/ len(y_true)
    return accuracy
```

In [3]:
```
data = datasets.load_breast_cancer()
x , y = data.data , data.target
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3 )
```

In [4]:
```
clf = RandomForestClassifier()
clf.fit(x_train , y_train)
```

Out[4]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, 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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [5]:
```
y_pred = clf.predict(x_test)
```

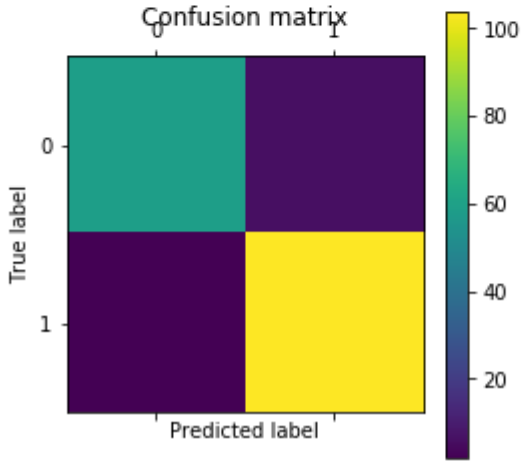
In [6]:
```
acc = accuracy(y_test,y_pred)
```

In [7]:
```
print("Accuracy:",acc)
```

```
Accuracy: 0.9532163742690059
```

In [8]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
plt.matshow(confusion_matrix)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
[[ 59   6]
 [  2 104]]
```



K Neighbors Algorithm

In [1]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
```

In [2]:
```
data = datasets.load_breast_cancer()
x , y = data.data , data.target
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3 )
```

In [3]:
```
clf= KNeighborsClassifier(n_neighbors=5)
clf.fit(x_train,y_train)
```

Out[3]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [4]:
```
clf.classes_
```

Out[4]:
```
array([0, 1])
```

In [5]:
```
y_pred = clf.predict(x_test)
print('The score of the model using KNN in training:',clf.score(x_train,y_train))
print('The score of the model using KNN in testing: ',clf.score(x_test,y_test))
```

```
The score of the model using KNN in training: 0.9447236180904522
The score of the model using KNN in testing:  0.9473684210526315
```

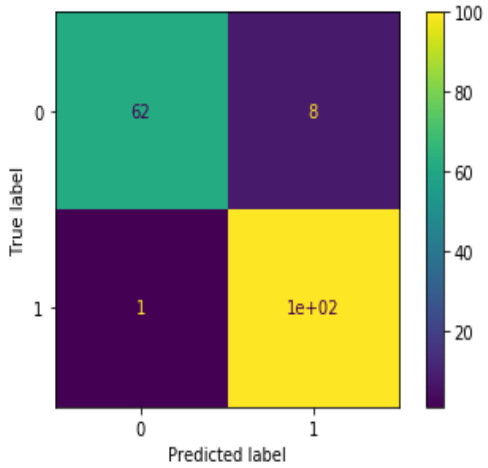
In [6]:
```
print('the first 15 predicted class : ',y_pred[:15])
print('the first 15 actual class :',y_test[:15])
```

```
the first 15 predicted class :  [0 0 1 0 1 0 1 1 1 1 1 1 0 1 0]
the actual first 15 first class : [0 0 1 0 1 0 1 1 1 1 1 1 0 1 0]
```

In [7]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
print('the confusion matrix for the model using KNN \n',confusion_matrix(y_pred,y_test))
plot_confusion_matrix(clf,x_test,y_test)
```

```
the confusion matrix for the model using KNN
 [[ 62   1]
 [  8 100]]
```

Out[7]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23698016f08>



Support Vector Machine (SVC)

In [1]:
```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
import numpy as np
```

In [2]:
```
iris = datasets.load_iris()
iris.feature_names
```

Out[2]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [3]:
```
iris.target_names
```

Out[3]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [4]:
```
x = iris["data"][: , (2,3)]
y = (iris["target"] == 2)astype(np.float64)
```

In [5]:
```
svm_clf = Pipeline([
    ( "scaler" , StandardScaler()),
    ( "linear_svc" , LinearSVC(C=1 , loss="hinge" , random_state =42)), ])
```

In [6]:
```
svm_clf.fit(x,y)
```

Out[6]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('linear_svc',
                 LinearSVC(C=1, class_weight=None, dual=True,
                           fit_intercept=True, intercept_scaling=1,
                           loss='hinge', max_iter=1000, multi_class='ovr',
                           penalty='l2', random_state=42, tol=0.0001,
                           verbose=0))],
         verbose=False)
```

In [7]:
```
svm_clf.predict([[5.5, 1.7]])
```

Out[7]:
```
array([ True])
```