```python
In [1]:
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Exercice I

1- Lecture des données

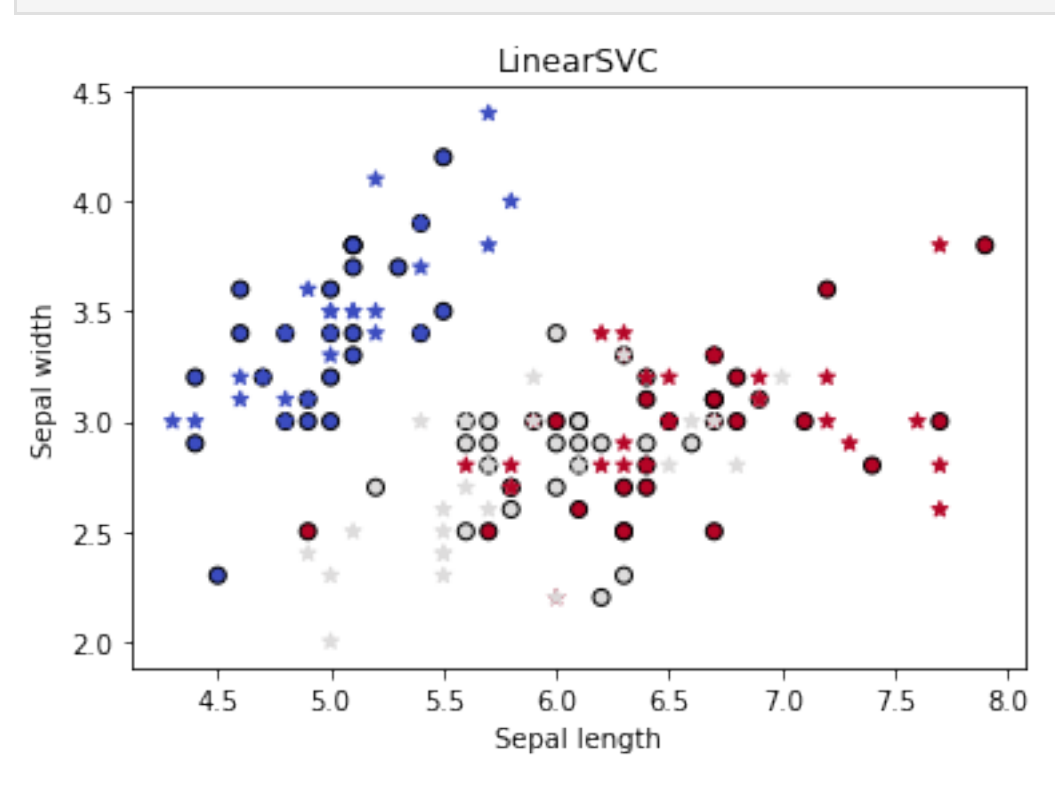
```python
In [2]:
# Chargement des données
iris = datasets.load_iris()
```

2- Partition des données

```python
In [3]:
# On considère 50% pour l'entrainement et 50%=0.5 pour le test
X, y = iris.data[:, :2], iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
class_names = iris.target_names
```

3- Affichage de la dispersion des données

```python
In [4]:
# Sur l'axe des x on aura Sepal length et sur l'axe de y Sepal width
# On a trois couleurs pour les trois espèces
# et deux marker pour entrainement et test
plt.scatter(X_train[:, 0], X_train[:, 1], label="train", edgecolors='k', c=y_train, cmap=plt.cm.coolwarm)
plt.scatter(X_test[:, 0], X_test[:, 1], label="test", marker='*', c=y_test, cmap=plt.cm.coolwarm)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title("LinearSVC")
plt.show()
```



4- Conception du classificateur

```python
In [5]:
# On fixe le paramètre de régularisation C=1.0
# On fixe random state à 10
C = 1.0
lin_svc = svm.LinearSVC(C=C, random_state =10)
lin_svc.fit(X_train, y_train)
```

```
/Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increa
se the number of iterations.
  warnings.warn(
Out[5]: LinearSVC(random_state=10)
```

5- Taux de classification

```python
In [6]:
# On teste sur l'ensemble des données de test
lin_svc.score(X_test, y_test)
```

```
Out[6]: 0.8
```

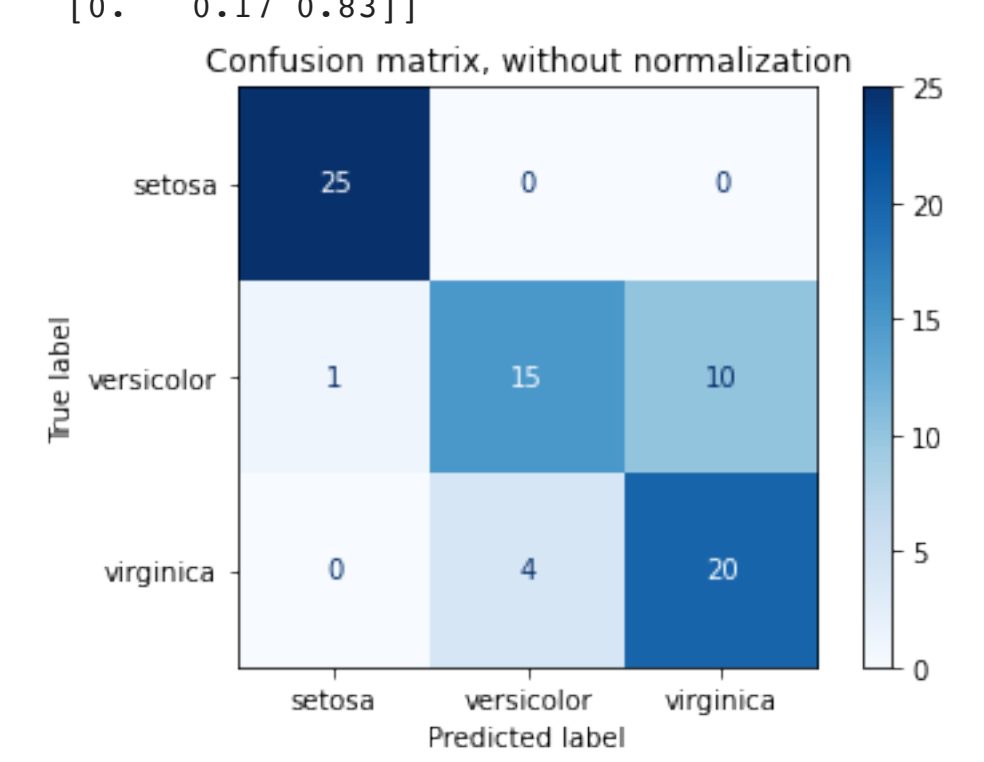
6- Matrice de confusion

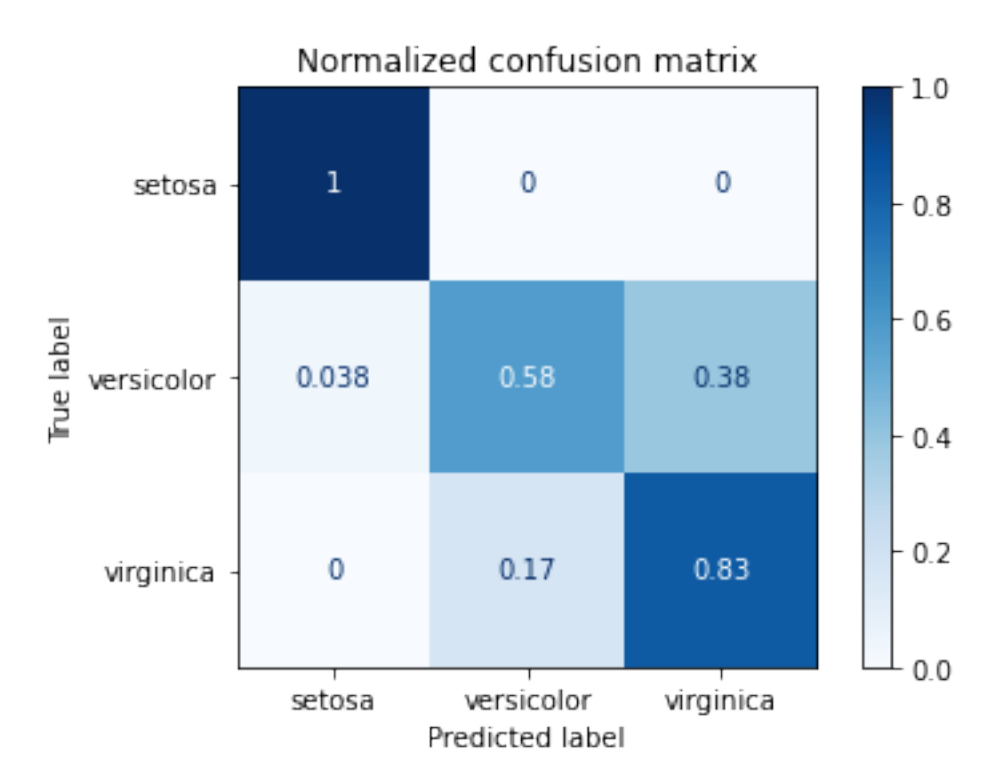
```python
In [7]:
!pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in /Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in /Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages (from scikit-learn) (1.20.3)
Requirement already satisfied: scipy>=1.1.0 in /Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages (from scikit-learn) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: joblib>=0.11 in /Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages (from scikit-learn) (1.1.0)
```

```python
In [8]:
# La matrice de confusion peut être non-normalisée
# Dans ce cas, on affiche le nombre d'observation
# ou normalisée par rapport au nombre d'observation totale
# par classe
# le paramètre precision permet de fixer le nombre de
# chiffre après la virgule

np.set_printoptions(precision=2)

titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    disp = ConfusionMatrixDisplay.from_estimator(
        lin_svc,
        X_test,
        y_test,
        display_labels=class_names,
        cmap=plt.cm.Blues,
        normalize=normalize,
    )
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

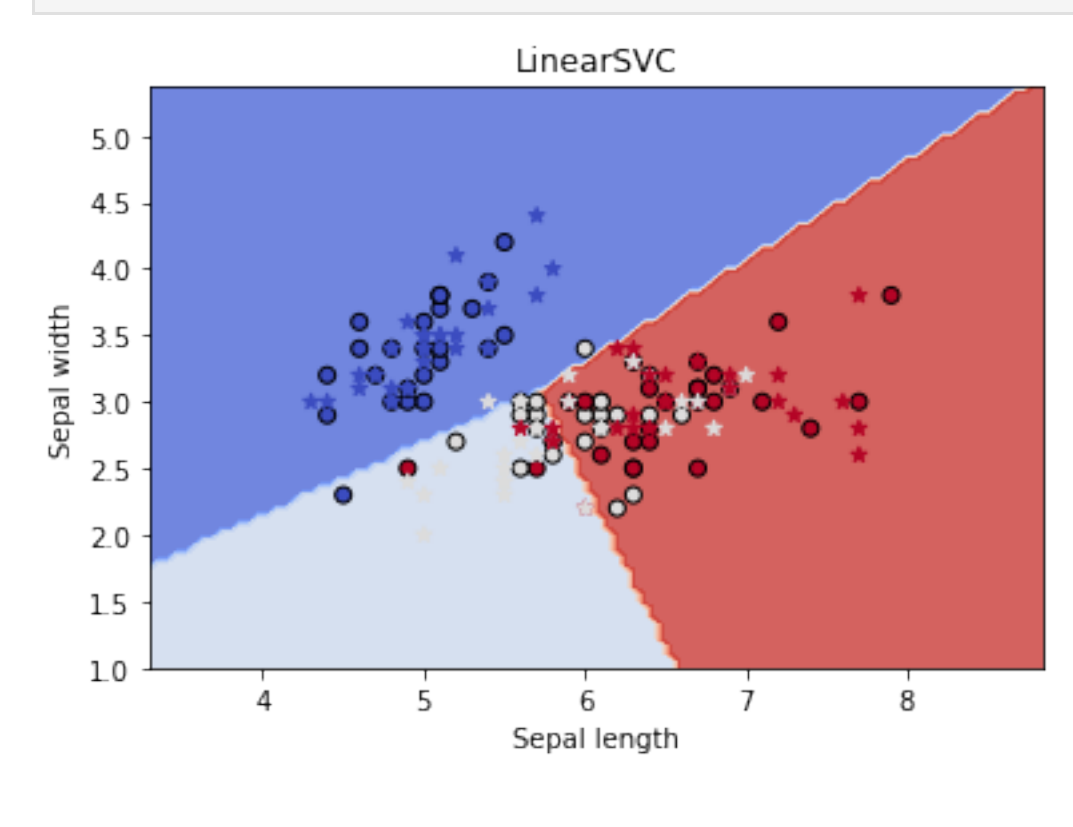
```
Confusion matrix, without normalization
[[25  0  0]
 [ 1 15 10]
 [ 0  4 20]]
Normalized confusion matrix
[[1.   0.   0.  ]
 [0.04 0.58 0.38]
 [0.   0.17 0.83]]
```





7- Visualisation des frontières de décision

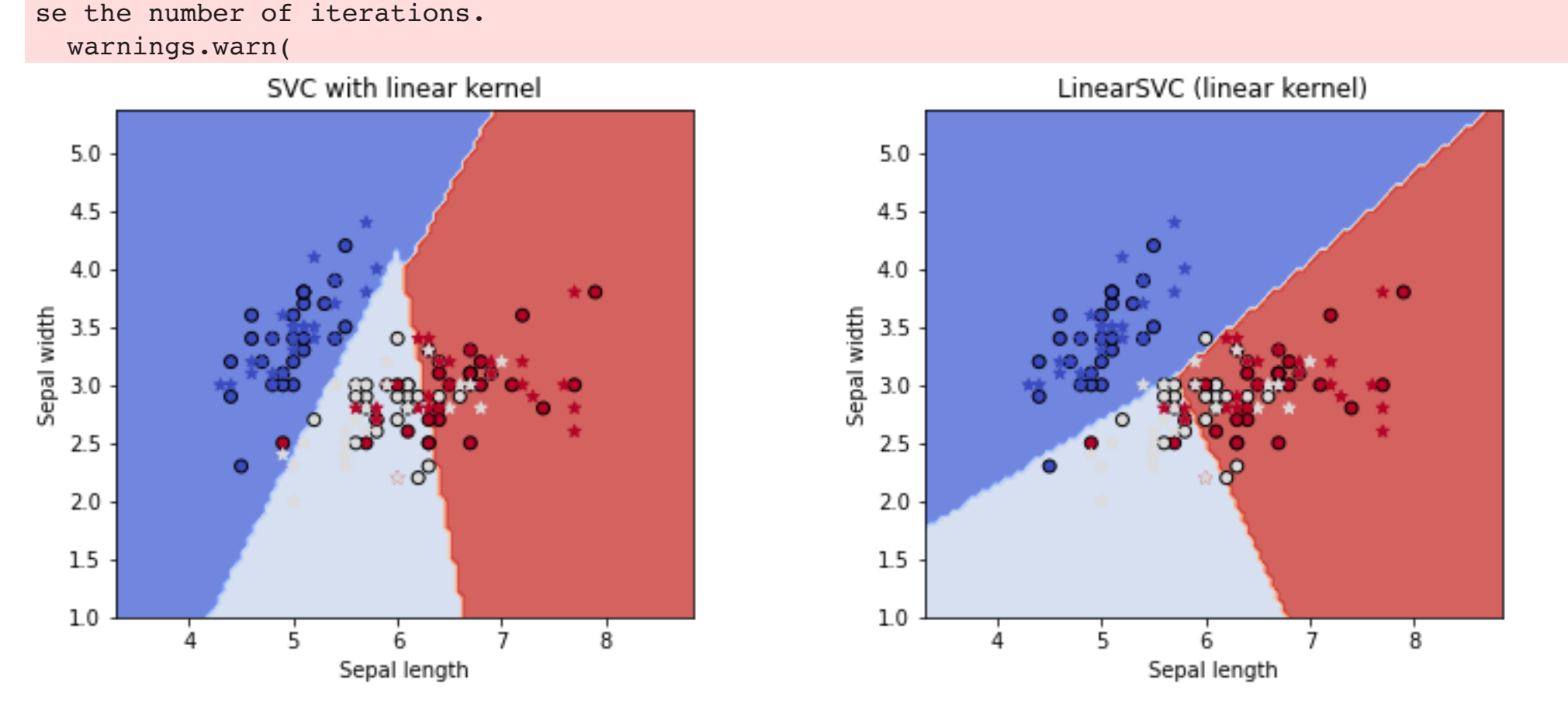
```python
In [9]:
# Créer la surface de décision discretisée
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

# Pour afficher la surface de décision on va discrétiser l'espace avec un pas h
h = max((x_max - x_min) / 100, (y_max - y_min) / 100)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Surface de décision linéaire
Z = lin_svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# On ajoute les observations utilisées pour d'apprentissage et pour le test également

plt.scatter(X_train[:, 0], X_train[:, 1], label="train", edgecolors='k', c=y_train, cmap=plt.cm.coolwarm)
plt.scatter(X_test[:, 0], X_test[:, 1], label="test", marker='*', c=y_test, cmap=plt.cm.coolwarm)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title("LinearSVC")
plt.show()
```



8- Utilisation des modèles lin_svc et svc

```python
In [10]:
# Les modèles linéaires LinearSVC() et SVC(kernel='linear')
# traitent les problèmes multi-classes de manière différente
# linearSVC utilise One-vs-All alors que SVC utilise One-vs-One

lin_svc = svm.LinearSVC(C=C).fit(X_train, y_train)
svc = svm.SVC(kernel='linear', C=C).fit(X_train, y_train)

titles = ['SVC with linear kernel', 'LinearSVC (linear kernel)']

fig = plt.figure(figsize=(12, 4.5))

for i, clf in enumerate((svc, lin_svc)):
    plt.subplot(1, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    # Utiliser une palette de couleurs
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    # Afficher aussi les points d'apprentissage
    plt.scatter(X_train[:, 0], X_train[:, 1], label="train", edgecolors='k', c=y_train, cmap=plt.cm.coolwarm)
    plt.scatter(X_test[:, 0], X_test[:, 1], label="test", marker='*', c=y_test, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.title(titles[i])
plt.show()
```

```
/Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increa
se the number of iterations.
  warnings.warn(
```



9- Comparaison des résultats

Dans les deux cas, on obtient des classifieurs linéaires qui produisent des frontières de décision linéaires. Ces frontières sont légèrement différentes.

10- Utilisation des 4 features

```python
In [11]:
# On utilise les quatres caractéristiques pour l'entrainement du modèle
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
lin_svc = svm.LinearSVC(C=C, random_state =10 )
lin_svc.fit(X_train, y_train)
lin_svc.score(X_test, y_test)
```

```
/Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increa
se the number of iterations.
  warnings.warn(
Out[11]: 0.92
```

L'utilisation des quatre caractéristiques permet d'améliorer les taux de classifications (80% par rapport à 92%)

Excercice II

```python
In [12]:
from sklearn import tree
```

1- Partition des données

```python
In [13]:
# On considère 70% pour l'entrainement et 30%=0.3 pour le test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=0)
class_names = iris.target_names
```

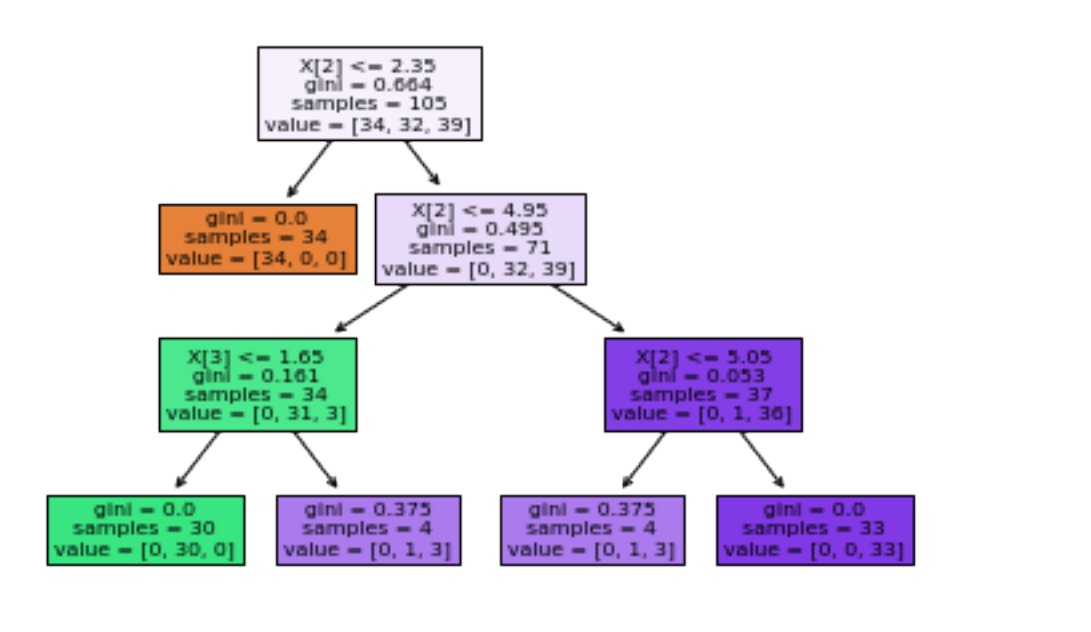
2- Conception d'un arbre de décision de profondeur max de 3

```python
In [14]:
clf = tree.DecisionTreeClassifier(max_depth = 3)
clf.fit(X_train, y_train)
```

```
Out[14]: DecisionTreeClassifier(max_depth=3)
```

```python
In [15]:
tree.plot_tree(clf, filled=True)
```

```
Out[15]: [Text(0.375, 0.875, 'X[2] <= 2.35\ngini = 0.664\nsamples = 105\nvalue = [34, 32, 39]'),
 Text(0.25, 0.625, 'gini = 0.0\nsamples = 34\nvalue = [34, 0, 0]'),
 Text(0.5, 0.625, 'X[2] <= 4.95\ngini = 0.495\nsamples = 71\nvalue = [0, 32, 39]'),
 Text(0.25, 0.375, 'X[3] <= 1.65\ngini = 0.161\nsamples = 34\nvalue = [0, 31, 3]'),
 Text(0.125, 0.125, 'gini = 0.0\nsamples = 30\nvalue = [0, 30, 0]'),
 Text(0.375, 0.125, 'gini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.75, 0.375, 'X[2] <= 5.05\ngini = 0.053\nsamples = 37\nvalue = [0, 1, 36]'),
 Text(0.625, 0.125, 'gini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.875, 0.125, 'gini = 0.0\nsamples = 33\nvalue = [0, 0, 33]')]
```



3- Taux de classification

```python
In [16]:
clf.predict(X_test)
```

```
Out[16]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2, 1, 1, 2, 0, 2, 0,
       0])
```

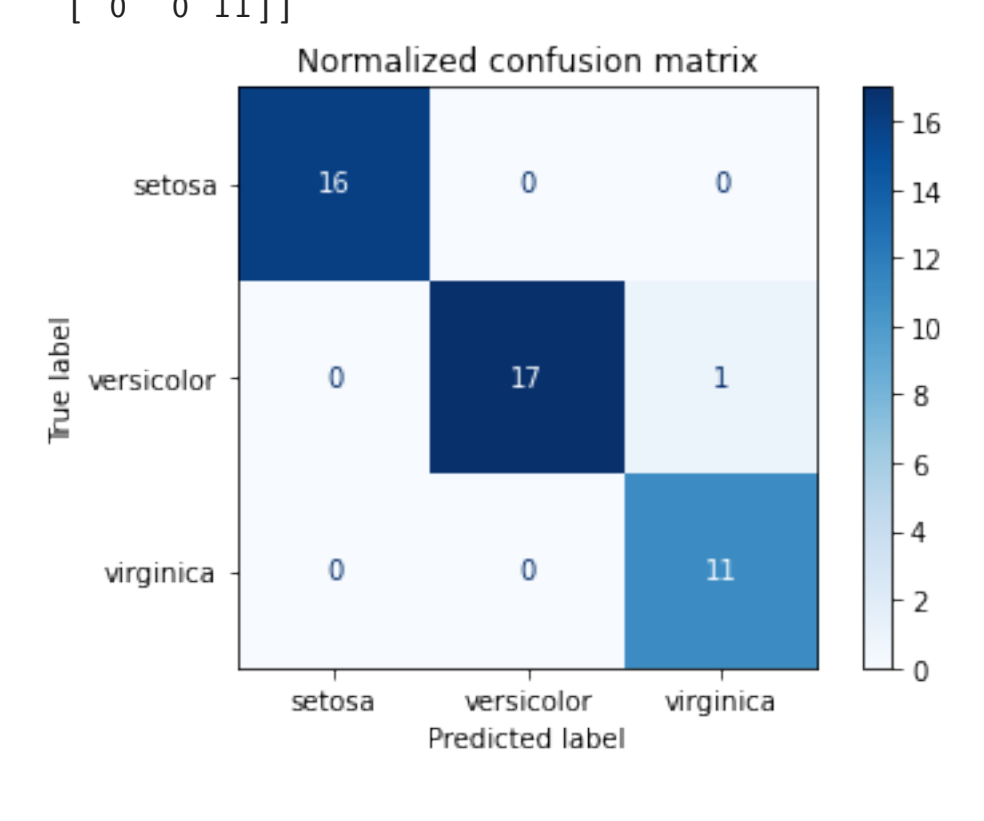
```python
In [17]:
# Taux de classification
clf.score(X_test, y_test)
```

```
Out[17]: 0.9777777777777777
```

Matrice de confusion

```python
In [18]:
disp = ConfusionMatrixDisplay.from_estimator(
        clf,
        X_test,
        y_test,
        display_labels=class_names,
        cmap=plt.cm.Blues,
    )
disp.ax_.set_title(title)

print(title)
print(disp.confusion_matrix)

plt.show()
```

```
Normalized confusion matrix
[[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
```



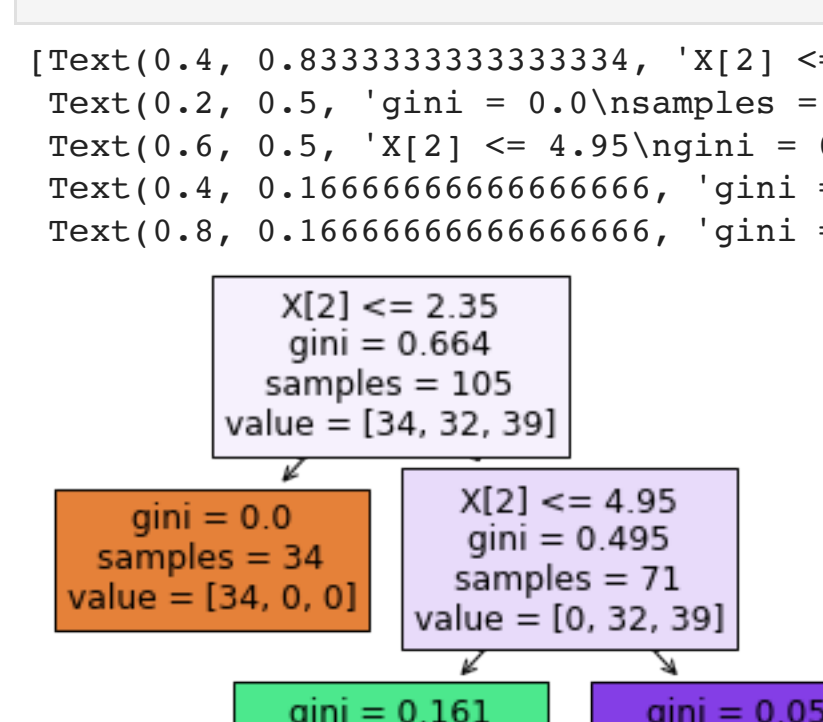
4- Conception d'un arbre de décision avec un nombre minimum d'échantillons requis par feuille égal à 20.

```python
In [19]:
clf = tree.DecisionTreeClassifier(min_samples_leaf = 20)
clf.fit(X_train, y_train)
```

```
Out[19]: DecisionTreeClassifier(min_samples_leaf=20)
```

```python
In [20]:
tree.plot_tree(clf, filled=True)
```

```
Out[20]: [Text(0.4, 0.8333333333333334, 'X[2] <= 2.35\ngini = 0.664\nsamples = 105\nvalue = [34, 32, 39]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 34\nvalue = [34, 0, 0]'),
 Text(0.6, 0.5, 'X[2] <= 4.95\ngini = 0.495\nsamples = 71\nvalue = [0, 32, 39]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.161\nsamples = 34\nvalue = [0, 31, 3]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.053\nsamples = 37\nvalue = [0, 1, 36]')]
```



5- Taux de classification

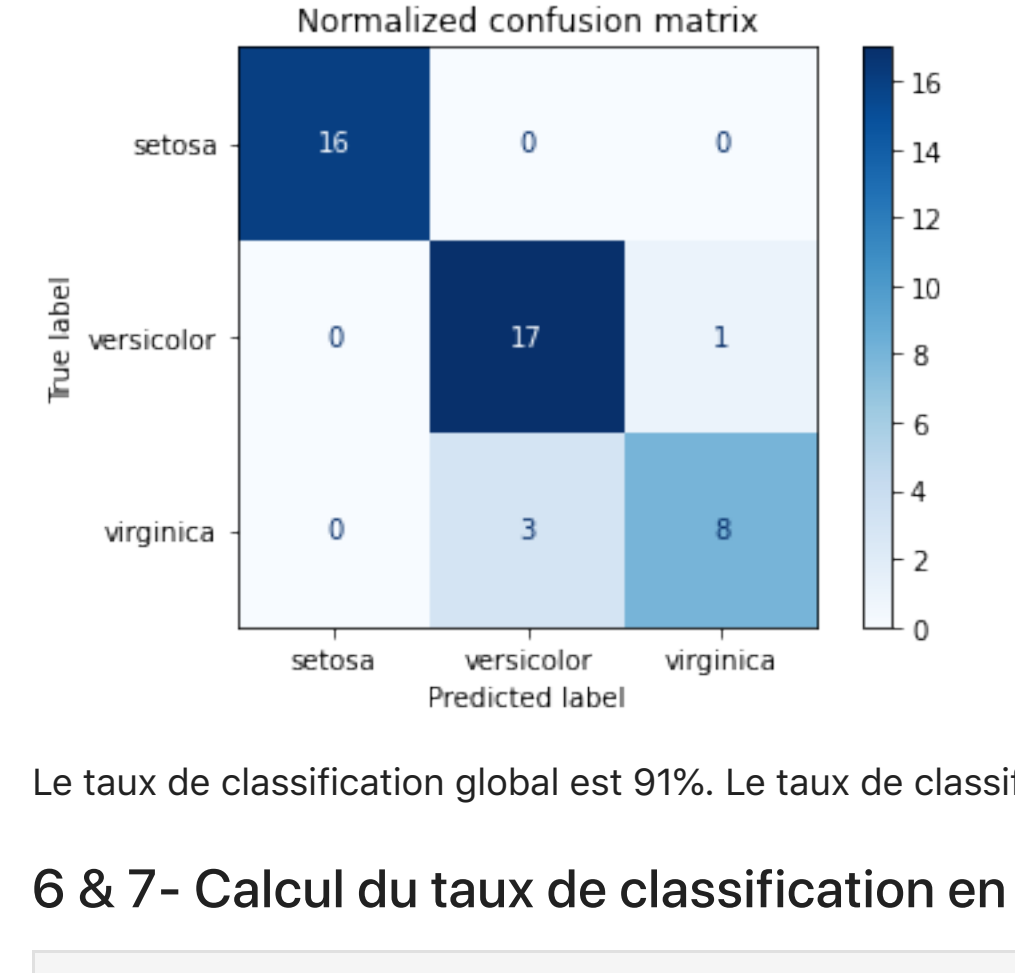
```python
In [21]:
clf.score(X_test, y_test)
```

```
Out[21]: 0.9111111111111111
```

Matrice de confusion

```python
In [22]:
disp = ConfusionMatrixDisplay.from_estimator(
        clf,
        X_test,
        y_test,
        display_labels=class_names,
        cmap=plt.cm.Blues,
    )
disp.ax_.set_title(title)

print(title)
print(disp.confusion_matrix)

plt.show()
```

```
Normalized confusion matrix
[[16  0  0]
 [ 0 17  1]
 [ 0  3  8]]
```



Le taux de classification global est 91%. Le taux de classification de la classe Setosa est de 16/16=100%, celui de la classe virginica est de 8/11=72%

6 & 7- Calcul du taux de classification en fonction des paramètres mdepth et msplit

```python
In [23]:
from sklearn import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(iris.data, iris.target,
            test_size=0.95, random_state=0)

for mdepth in [1, 2, 3, 4, 5, 6, 7]:
    clf = tree.DecisionTreeClassifier(max_depth=mdepth)
    clf = clf.fit(X_train, y_train)
    print(clf.score(X_test, y_test))

for msplit in [2, 3, 5, 10, 15, 20]:
    clf = tree.DecisionTreeClassifier(min_samples_split=msplit)
    clf = clf.fit(X_train, y_train)
    print(clf.score(X_test, y_test))
```

```
0.6573426573426573
0.7972027972027972
0.8881118881118881
0.8881118881118881
0.7062937062937062
0.7972027972027972
0.7972027972027972
0.7062937062937062
0.8881118881118881
0.6573426573426573
0.32167832167832167
0.32167832167832167
0.32167832167832167
```

8- Optimisation des paramètres mdepth et msplit

```python
In [24]:
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

```python
In [25]:
X_train, X_test, y_train, y_test = model_selection.train_test_split(iris.data, iris.target,
    test_size=0.30, random_state=0)

pgrid = {"max_depth": [1, 2, 3, 4, 5, 6, 7],
      "min_samples_split": [2, 3, 5, 10, 15, 20]}
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid=pgrid, cv=10)
grid_search.fit(X_train, y_train)
grid_search.best_estimator_.score(X_test, y_test)
print("Best parameters: ", grid_search.best_params_)
```

```
Best parameters:  {'max_depth': 3, 'min_samples_split': 2}
```

```python
In [ ]:
```

```python
In [ ]:
```