Les machines à vecteurs de support (SVM : Support Vector Machines) sont une classe des méthodes d'apprentissage statistique basées sur le principe de la maximisation de la marge (séparation des classes). Il existe plusieurs formulations (linéaires, versions à noyaux) qui peuvent s'appliquer sur des données séparables (linéairement) mais aussi sur des données non séparables. L'avantage des SVM est qu'ils n'utilisent pas tous les échantillons d'apprentissage, mais seulement une partie (les vecteurs de support). En conséquence, ces algorithmes demandent moins de mémoire. Dans Scikit-learn, les SVM sont implémentées dans le module sklearn.svm. Nous allons utiliser le jeu de données Iris

In [16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split

# Chargement des données
iris = datasets.load_iris()
iris.feature_names
```

Out[16]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [13]:
```
iris.target_names
```

Out[13]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [18]:
```
df=pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
```

Out[18]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Nous ne conservons que les deux premiers attributs du jeu de données (longueur et largeur des sépales) :

In [23]:
```
X, y = iris.data[:, :2], iris.target
# On conserve 50% du jeu de données pour l'évaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

Entrainer le SVM

In [4]:
```
C = 0.9 # paramètre de régularisation
lin_svc = svm.LinearSVC(C=C)
lin_svc.fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[4]:
```
LinearSVC(C=0.9, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Calculez le score d'échantillons bien classifiés sur le jeu de données de test

In [6]:
```
lin_svc.score(X_test, y_test)
```

Out[6]:
```
0.6933333333333334
```

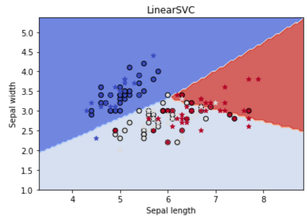
la surface de décision apprise par le modèle :

In [8]:
```
# 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
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)
# 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("LinearSVC")
```

Out[8]:
```
Text(0.5, 1.0, 'LinearSVC')
```



Le classifieur produit des frontières de décision linéaire. Cela suffit à séparer une des trois classes des deux autres, toutefois en ne considérant que les deux premiers attributs, les deux autres classes ne semblent pas linéairement séparables. Il faudrait soit utiliser un modèle non linéaire, soit ajouter des attributs supplémentaires en espérant qu'ils permettront de séparer linéairement les deux classes restantes. Les modèles linéaires LinearSVC() et SVC(kernel='linear'), produisent des résultats légèrement différents à cause du fait qu'ils optimisent des fonctions de coût différentes mais aussi à cause du fait qu'ils gèrent les problèmes multi-classe de manière différente (linearSVC utilise One-vs-All et SVC utilise One-vs-One).

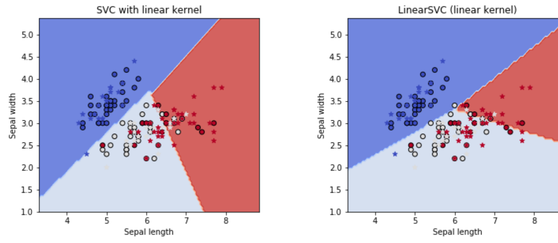
In [9]:
```
lin_svc = svm.LinearSVC(C=C).fit(X_train, y_train)
svc = svm.SVC(kernel='linear', C=C).fit(X_train, y_train)

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()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```



Pour l'instant, nous n'avons exploité que deux variables explicatives. Néanmoins, l'intérêt des machines à vecteur de support linéaires est qu'il est souvent plus facile de trouver des hyperplans séparateurs dans des espaces de grande dimension. Utiliser quatre attributs du jeu de données Iris.

In [10]:
```
X, y = iris.data, iris.target
# On conserve 50% du jeu de données pour l'évaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
lin_svc = svm.LinearSVC(C=C)
lin_svc.fit(X_train, y_train)
lin_svc.score(X_test, y_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[10]:
```
0.9866666666666667
```

Le score augmente (de 0,75 à 0,9 en général) car les deux attributs que nous avons ajouté permettent de mieux séparer les trois classes.

In [ ]:

In [ ]: