IRIS DATA SET

From the depository:

Data Set Information:

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Predicted attribute: class of iris plant.

This is an exceedingly simple domain.

This data differs from the data presented in Fishers article (identified by Steve Chadwick, spchadwick '@' espeedaz.net). The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the error is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" where the errors are in the second and third features.

Attribute Information:

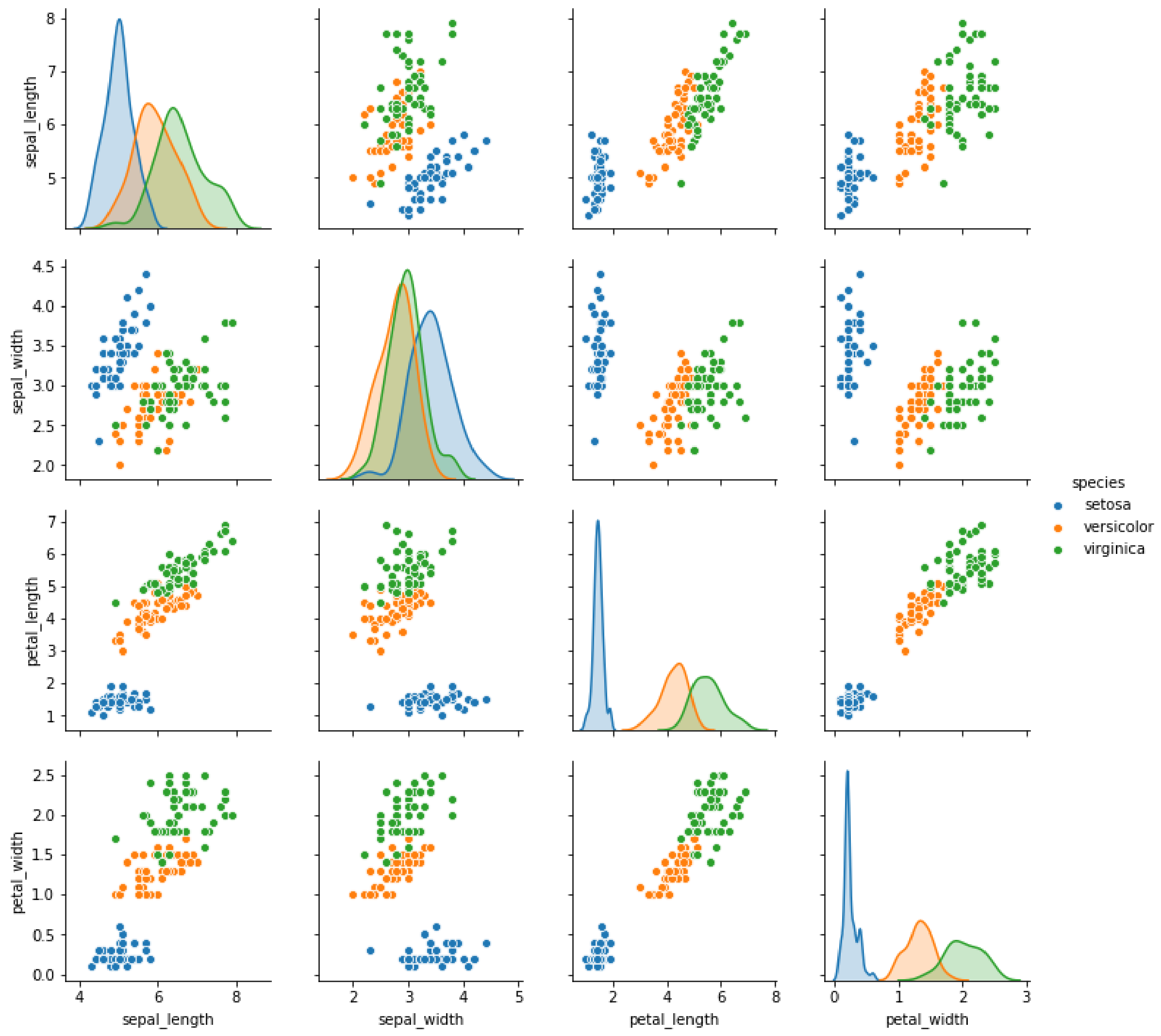
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class: -- Iris Setosa -- Iris Versicolour -- Iris Virginica

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import itertools
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
```

In [5]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [2]:
```
# load dataset
iris = sns.load_dataset("iris")
X = iris.drop(columns=['species'],axis=1)
y = iris['species']
```

```
In [6]: # Visualize the relation among features
        g = sns.pairplot(iris, hue="species")
```



It appears that Setosa is linearly separable whereas the other two aren't

```
In [84]: # Split the data to test and train data set
         from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3);
```

In [85]:
```python
# The following function plots the confusion matrix.

def plot_confusion_matrix(y_true, y_pred, classes,
                          title=None,cmap=plt.cm.Blues):

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45,
             ha="right",rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

LOGISTIC REGRESSION

In [86]:
```python
from sklearn.linear_model import LogisticRegression
model_log = LogisticRegression(solver='lbfgs', multi_class='multinomial'
)
model_log.fit(X_train,y_train)
model_log_pred = model_log.predict(X_test)
```

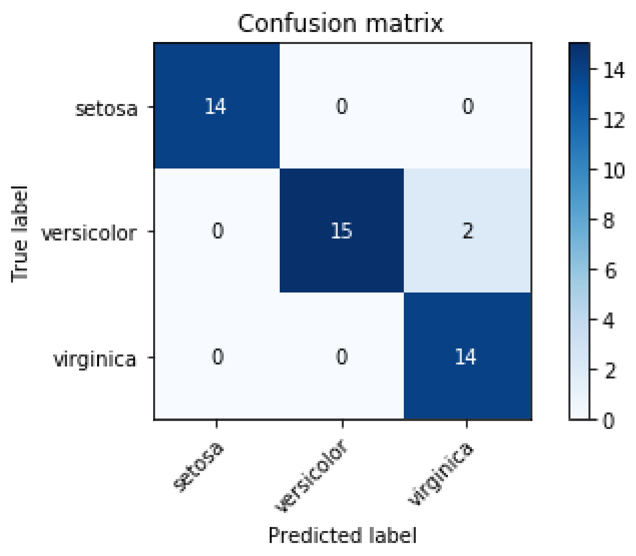
In [87]:
```
print(classification_report(y_test, model_log_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       1.00      0.88      0.94        17
   virginica       0.88      1.00      0.93        14

   micro avg       0.96      0.96      0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

So what do these numbers mean??

- The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
- The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
- The f1 score can be interpreted as a weighted harmonic mean of the precision and recall, where an f1 score reaches its best value at 1 and worst score at 0.
- The support is the number of occurrences of each class in y_true.

In [88]:
```
# Plot confusion matrix
plot_confusion_matrix(y_test, model_log_pred, classes=y.unique(),
                      title='Confusion matrix');
```



As we can see in the confusion matrix above, setosa has been perfectly classified using Logistic Regression, where as there were some errors in classifying versicolor and virginica.

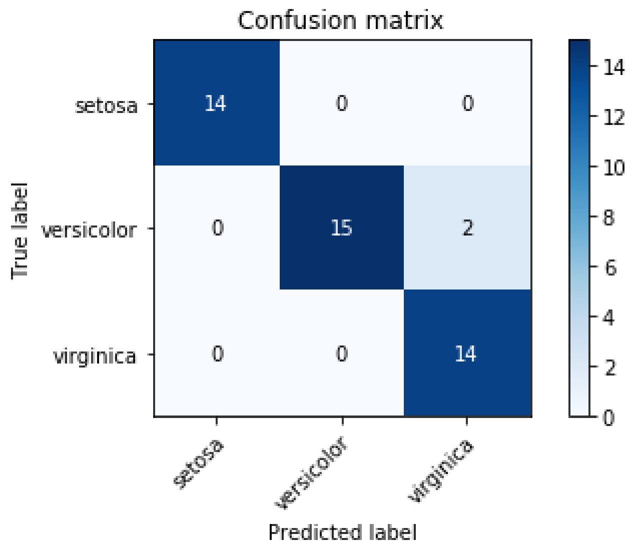
Let's see if we can improve using Decision Tree and Soft-Margin SVM
(We can also test later if Decision Tree is overfitting and see if Random Forest does better)

In [89]:
```
from sklearn import tree
model_dtc = tree.DecisionTreeClassifier()
model_dtc.fit(X_train, y_train)
model_dtc_pred = model_log.predict(X_test)
```

In [90]:
```
print(classification_report(y_test, model_dtc_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       1.00      0.88      0.94        17
   virginica       0.88      1.00      0.93        14

   micro avg       0.96      0.96      0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

In [91]:
```
plot_confusion_matrix(y_test, model_dtc_pred, classes=y.unique(),
                      title='Confusion matrix');
```



Appears to do the same as Logistic Regression

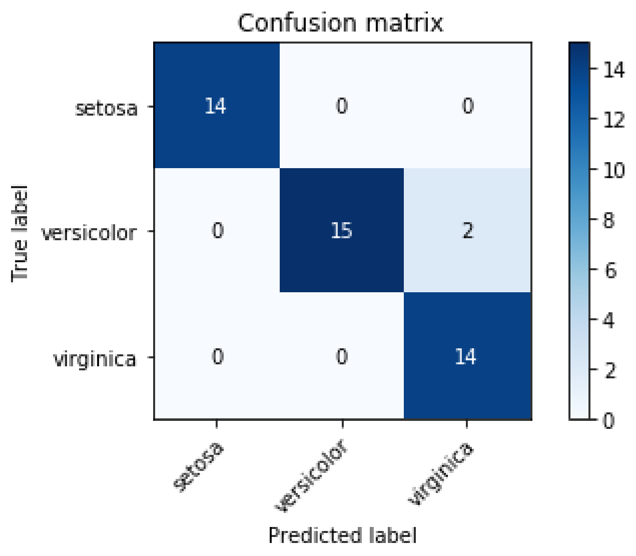
Soft-Margin SVM

In [92]:
```
from sklearn import svm
model_svm = svm.SVC(gamma='scale', decision_function_shape='ovo')
model_svm.fit(X_train, y_train)
model_svm_pred = model_log.predict(X_test)
```

In [93]:
```
print(classification_report(y_test, model_svm_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       1.00      0.88      0.94        17
   virginica       0.88      1.00      0.93        14

   micro avg       0.96      0.96      0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

In [94]:
```
plot_confusion_matrix(y_test, model_svm_pred, classes=y.unique(),
                      title='Confusion matrix');
```



In [97]:
```
from sklearn.metrics import accuracy_score
print('Accuracy scores using')
print('Logistic Regression:',accuracy_score(y_test,model_log_pred))
print('Decision Tree Classifier:',accuracy_score(y_test,model_dtc_pred))
print('Support Vector Machine:',accuracy_score(y_test,model_svm_pred))
```

```
Accuracy scores using
Logistic Regression: 0.9555555555555556
Decision Tree Classifier: 0.9555555555555556
Support Vector Machine: 0.9555555555555556
```

Conclusion

It appears that all the classifiers we tested out give back the same classification rules. This could be because of the limited amount of data set we have. Remember we only have 150 data points, of which only 30% (i.e., 45 points are used for testing purposes). Even with the limited amount of data we have, we were able to achieve an accuracy greater than 95%, and we were able to classify Iris Setosa perfectly. That's pretty good in my opinion.