Support Vector Machine (SVM)

is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems.

The objective of the support vector machine algorithm

is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

About the Data Set - Cell Samples

Dataset is publicly available from the UCI Machine Learning Repository (Asuncion and Newman, 2007). The dataset consists of several hundred human cell sample records, each of which contains the values of a set of cell characteristics.

In [1]:
```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:
```
import os
os.chdir(r'C:\Users\HP\Downloads\cell_samples Data')
cell_df = pd.read_csv("cell_samples.csv")
cell_df.head()
```

Out[2]:

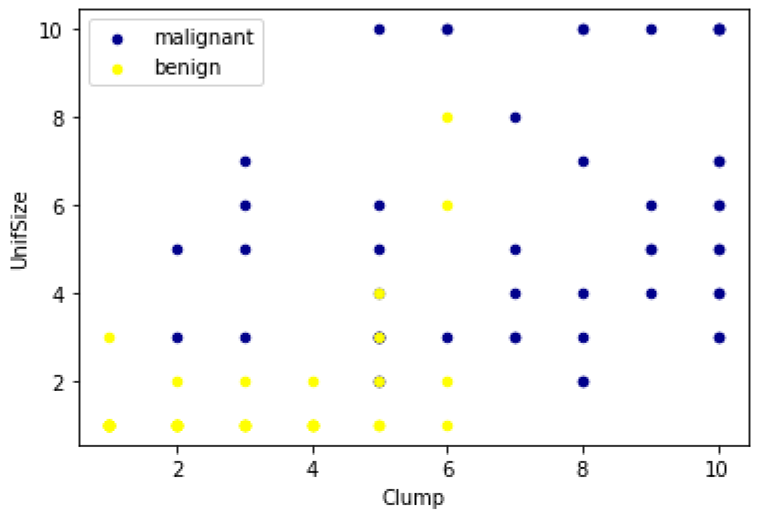
	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

The ID field contains the patient identifiers. The characteristics of the cell samples from each patient are contained in fields Clump to Mit. The values are graded from 1 to 10, with 1 being the closest to benign.

The Class field contains the diagnosis, as confirmed by separate medical procedures, as to whether the samples are benign (value = 2) or malignant (value = 4).

Let's look at the distribution of the classes based on Clump thickness and Uniformity of cell size:

```
In [3]: ax = cell_df[cell_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump',
                                                   y='UnifSize', color='DarkBlue',
                                                   label='malignant');
        cell_df[cell_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize',
                                                color='Yellow', label='benign', ax=ax);
        plt.show()
```



```
In [4]: cell_df.dtypes

Out[4]: ID                int64
        Clump             int64
        UnifSize          int64
        UnifShape         int64
        MargAdh           int64
        SingEpiSize       int64
        BareNuc          object
        BlandChrom        int64
        NormNucl          int64
        Mit               int64
        Class             int64
        dtype: object
```

```
In [5]: # It looks like the BareNuc column includes some values that are not numerical.
        cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
        cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
        cell_df.dtypes

Out[5]: ID               int64
        Clump            int64
        UnifSize         int64
        UnifShape        int64
        MargAdh          int64
        SingEpiSize      int64
        BareNuc          int32
        BlandChrom       int64
        NormNucl         int64
        Mit              int64
        Class            int64
        dtype: object
```

```
In [6]: feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh',
                             'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
        X = np.asarray(feature_df)
        X[0:5]

Out[6]: array([[ 5,  1,  1,  1,  2,  1,  3,  1,  1],
       [ 5,  4,  4,  5,  7, 10,  3,  2,  1],
       [ 3,  1,  1,  1,  2,  2,  3,  1,  1],
       [ 6,  8,  8,  1,  3,  4,  3,  7,  1],
       [ 4,  1,  1,  3,  2,  1,  3,  1,  1]], dtype=int64)
```

We want the model to predict the value of Class (that is, benign (=2) or malignant (=4)). As this field can have one of only two possible values, we need to change its measurement level to reflect this.

In [7]:
```
cell_df['Class'] = cell_df['Class'].astype('int')
y = np.asarray(cell_df['Class'])
y [0:5]
```

Out[7]:
```
array([2, 2, 2, 2, 2])
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2,
                                                     random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (546, 9) (546,)
Test set: (137, 9) (137,)
```

Modeling (SVM with Scikit-learn)

The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data into a higher dimensional space is called kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as:

1.Linear 2.Polynomial 3.Radial basis function (RBF) 4.Sigmoid

In [9]:
```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

Out[9]:
```
SVC()
```

In [10]:
```
# predict new values:
yhat = clf.predict(X_test)
yhat [0:5]
```

Out[10]:
```
array([2, 4, 2, 4, 2])
```

Evaluation

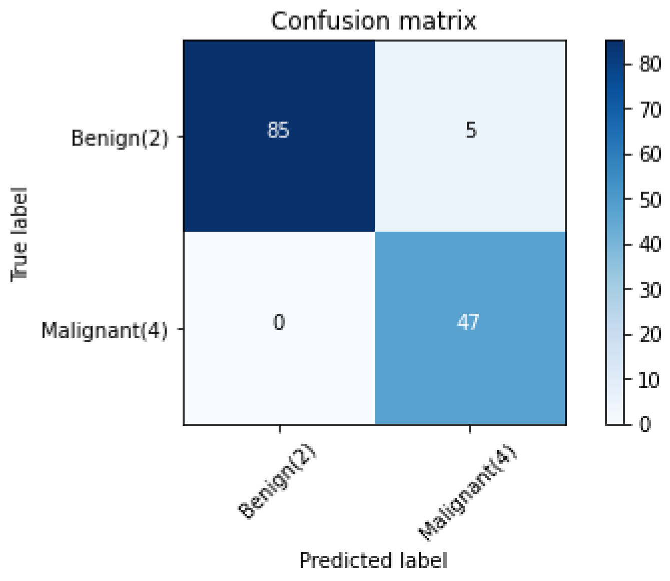
In [11]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

In [12]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [13]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[2,4])
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Benign(2)','Malignant(4)'],
                      normalize= False,  title='Confusion matrix')
```

```
              precision    recall  f1-score   support

           2       1.00      0.94      0.97        90
           4       0.90      1.00      0.95        47

    accuracy                           0.96       137
   macro avg       0.95      0.97      0.96       137
weighted avg       0.97      0.96      0.96       137

Confusion matrix, without normalization
[[85  5]
 [ 0 47]]
```



f1_score

In [14]:
```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

Out[14]:
```
0.9639038982104676
```

Rebuilding model with linear kernel

In [15]:
```
clf2 = svm.SVC(kernel='linear')
clf2.fit(X_train, y_train)
yhat2 = clf2.predict(X_test)
print("Avg F1-score: %.4f" % f1_score(y_test, yhat2, average='weighted'))
```

```
Avg F1-score: 0.9639
```

In [ ]:
```

```