SVM (Support Vector Machines)

Objectives

After completing this lab you will be able to:

- Use scikit-learn to Support Vector Machine to classify

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Load the Cancer data

The example is based on a dataset that is publicly available from the UCI Machine Learning Repository (Asuncion and Newman, 2007)[http://mlearn.ics.uci.edu/MLRepository.html]. The dataset consists of several hundred human cell sample records, each of which contains the values of a set of cell characteristics. The fields in each record are:

Field name	Description
ID	Clump thickness
Clump	Clump thickness
UnifSize	Uniformity of cell size
UnifShape	Uniformity of cell shape
MargAdh	Marginal adhesion
SingEpiSize	Single epithelial cell size
BareNuc	Bare nuclei
BlandChrom	Bland chromatin
NormNucl	Normal nucleoli
Mit	Mitoses
Class	Benign or malignant

For the purposes of this example, we're using a dataset that has a relatively small number of predictors in each record. To download the data, we will use `!wget` to download it from IBM Object Storage.

Load Data From CSV File

In [2]:
```
cell_df = pd.read_csv("cell_samples.csv")
cell_df
```

Out[2]:

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNu
0	1000025	5	1	1	1	2	1	3	
1	1002945	5	4	4	5	7	10	3	
2	1015425	3	1	1	1	2	2	3	
3	1016277	6	8	8	1	3	4	3	
4	1017023	4	1	1	3	2	1	3	
...	...	...	...	...	...	...	...	...	
694	776715	3	1	1	1	3	2	1	
695	841769	2	1	1	1	2	1	1	
696	888820	5	10	10	3	7	3	8	1
697	897471	4	8	6	4	3	4	10	
698	897471	4	8	8	5	4	5	10	

699 rows × 11 columns

In [3]:
```
cell_df.head()
```

Out[3]:

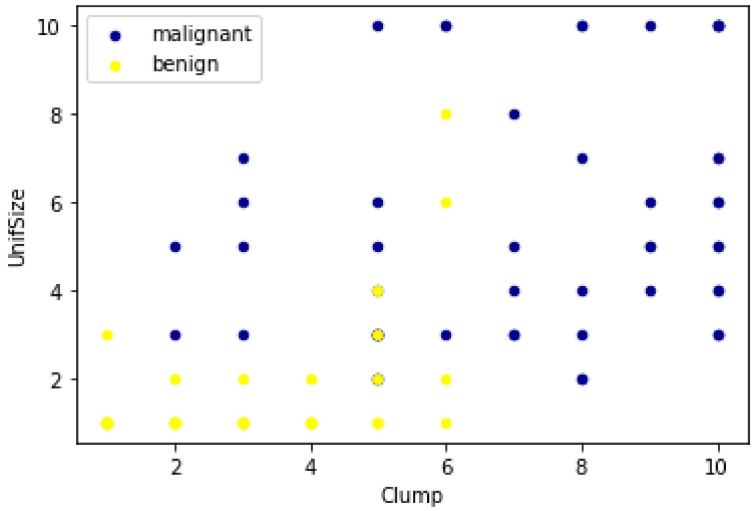
	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl
0	1000025	5	1	1	1	2	1	3	1
1	1002945	5	4	4	5	7	10	3	2
2	1015425	3	1	1	1	2	2	3	1
3	1016277	6	8	8	1	3	4	3	7
4	1017023	4	1	1	3	2	1	3	1

In [4]:
```
cell_df.tail()
```

Out[4]:

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNuc
694	776715	3	1	1	1	3	2	1	1
695	841769	2	1	1	1	2	1	1	1
696	888820	5	10	10	3	7	3	8	10
697	897471	4	8	6	4	3	4	10	6
698	897471	4	8	8	5	4	5	10	4

In [5]:
```
ax = cell_df[cell_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump', y='Unif
cell_df[cell_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize'
plt.show()
```



Data pre-processing and selection

Lets first look at columns data types:

In [6]:
```
cell_df.dtypes
```

Out[6]:
```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc        object
BlandChrom      int64
NormNucl        int64
Mit             int64
Class           int64
dtype: object
```

It looks like the **BareNuc** column includes some values that are not numerical. We can drop those rows:

In [7]:
```
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes
```

```
<ipython-input-7-df8bc795062c>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
```

Out[7]:
```
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 [8]:
```
feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize',
X = np.asarray(feature_df)
X[0:5]
```

Out[8]:
```
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 [9]:
```
cell_df['Class'] = cell_df['Class'].astype('int')
y = np.asarray(cell_df['Class'])
y [0:5]
```

```
<ipython-input-9-4c19bccd33b6>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  cell_df['Class'] = cell_df['Class'].astype('int')
```

Out[9]:
```
array([2, 2, 2, 2, 2])
```

Train/Test dataset

Okay, we split our dataset into train and test set:

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_
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)

In [11]:
```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

Out[11]:
```
SVC()
```

After being fitted, the model can then be used to predict new values:

In [12]:
```
yhat = clf.predict(X_test)
yhat [0:5]
```

Out[12]:
```
array([2, 4, 2, 4, 2])
```

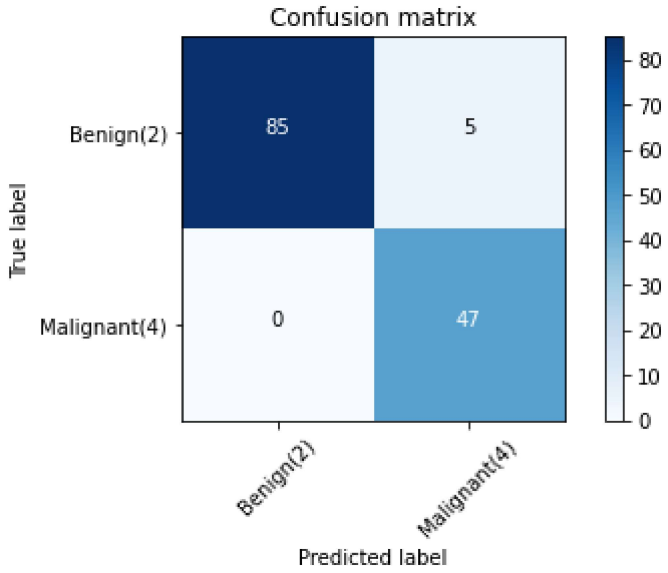
Evaluation

In [13]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

In [14]:
```
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 [15]:
```
# 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=
```

```
              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]]
```



In [16]:
```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

Out[16]: 0.9639038982104676

THE END