SVM (Support Vector Machines)

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Use scikit-learn to Support Vector Machine to classify

In this notebook, you will use SVM (Support Vector Machines) to build and train a model using human cell records, and classify cells to whether the samples are benign or malignant.

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data is transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

Table of contents

```
In [1]:  !pip install scikit-learn==0.23.1

         Collecting scikit-learn==0.23.1
           Downloading scikit_learn-0.23.1-cp37-cp37m-manylinux1_x86_64.whl (6.8 MB)
              |████████████████████████████████| 6.8 MB 33.5 MB/s
         Requirement already satisfied: scipy>=0.19.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from scikit-learn==0.23.1) (1.7.3)
         Requirement already satisfied: numpy>=1.13.3 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from scikit-learn==0.23.1) (1.21.4)
         Collecting joblib>=0.11
           Downloading joblib-1.1.0-py2.py3-none-any.whl (306 kB)
              |████████████████████████████████| 306 kB 78.8 MB/s
         Collecting threadpoolctl>=2.0.0
           Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
         Installing collected packages: threadpoolctl, joblib, scikit-learn
           Attempting uninstall: scikit-learn
             Found existing installation: scikit-learn 0.20.1
             Uninstalling scikit-learn-0.20.1:
               Successfully uninstalled scikit-learn-0.20.1
         Successfully installed joblib-1.1.0 scikit-learn-0.23.1 threadpoolctl-3.1.0
```

```python
In [2]:  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
```

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

2

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.

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: <u>Sign up now for free</u>

```
In [3]:   #Click here and press Shift+Enter
          !wget -O cell_samples.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Mod

          --2022-03-10 04:56:00--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Modu
          le%203/data/cell_samples.csv
          Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
          Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:
          443... connected.
          HTTP request sent, awaiting response... 200 OK
          Length: 19975 (20K) [text/csv]
          Saving to: 'cell_samples.csv'

          cell_samples.csv    100%[===================>]  19.51K  --.-KB/s    in 0s

          2022-03-10 04:56:00 (273 MB/s) - 'cell_samples.csv' saved [19975/19975]
```

Load Data From CSV File

```
In [4]:   cell_df = pd.read_csv("cell_samples.csv")
          cell_df.head()
```

Out[4]:

	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:

3

In [5]:

```
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()
```
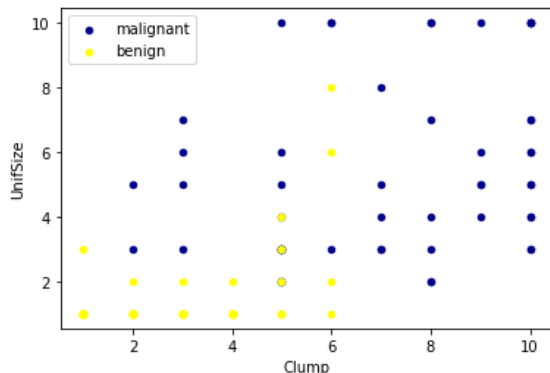

Data pre-processing and selection

Let's 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:

```
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[7]:
```
ID             int64
Clump          int64
UnifSize       int64
UnifShape      int64
MargAdh        int64
SingEpiSize    int64
BareNuc        int64
BlandChrom     int64
NormNucl       int64
Mit            int64
Class          int64
dtype: object
```

In [8]:
```
feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
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]])
```

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]
```

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

Train/Test dataset

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_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

Each of these functions has its characteristics, its pros and cons, and its equation, but as there's no easy way of knowing which function performs best with any given dataset. We usually choose different functions in turn and compare the results. Let's just use the default, RBF (Radial Basis Function) for this lab.

In [11]:
```python
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]:
```python
yhat = clf.predict(X_test)
yhat [0:5]
```

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

Evaluation

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

```
In [16]:   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 [17]:   # 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')
```
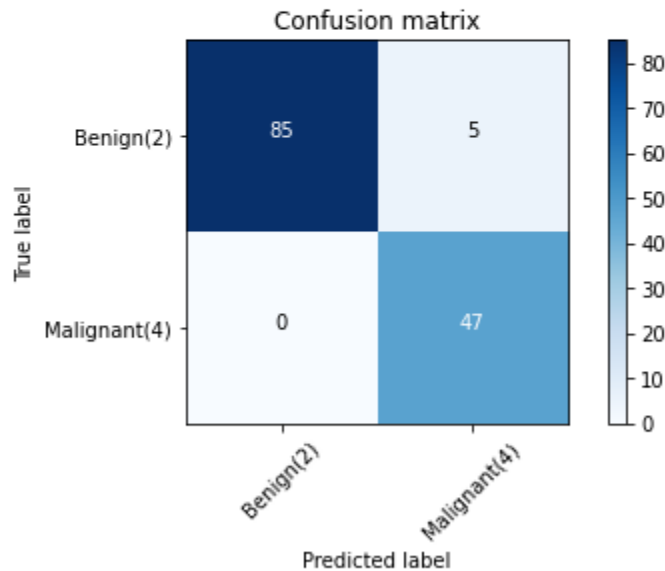
	precision	recall	f1-score	support
2	1.00	0.94	0.97	90
4	0.90	1.00	0.95	47
accuracy			0.96	137
macro avg	0.95	0.97	0.96	137
weighted avg	0.97	0.96	0.96	137

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


Confusion matrix

You can also easily use the **f1_score** from sklearn library:

Practice

Can you rebuild the model, but this time with a **linear** kernel? You can use **kernel='linear'** option, when you define the svm. How the accuracy changes with the new kernel function?

In [18]:
```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

Out[18]: 0.9639038982104676

Let's try the jaccard index for accuracy:

In [19]:
```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat,pos_label=2)
```

Out[19]: 0.9444444444444444