Exercice Lab: SVM (Support Vector Machines)

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.

Please notice that the practice labs (except the last week assignment) are optional and are provided for you to practice and understand the topic. Therefore, you do not need to submit those, as they are not graded, and won't be updated as complete. Just run the codes to see the results, and feel free to change it.

Ce cours utilise l'outil d'un tiers, Lab: SVM (Support Vector Machines), pour améliorer votre expérience d'apprentissage. L'outil référence des informations de base comme votre nom, votre adresse e-mail et votre ID Coursera.

SVM (Support Vector Machines)

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

```python
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
```

Code
```
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

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: Sign up now for free

```
[ ]:  #Click here and press Shift+Enter
      !wget -O cell_samples.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cog
```

Code
```
#Click here and press Shift+Enter
!wget -O cell_samples.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/cell_samples.csv
```

Load Data From CSV File

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

[3]:

	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

Code
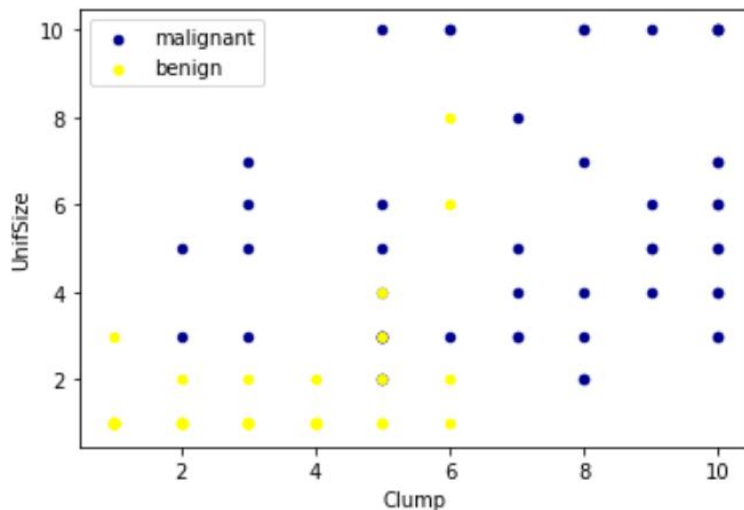cell_df = pd.read_csv("cell_samples.csv")
cell_df.head()

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).

Lets look at the distribution of the classes based on Clump thickness and Uniformity of cell size:

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


Code
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

Lets first look at columns data types:

```
[ ]:  cell_df.dtypes
```

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

Code
cell_df.dtypes

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
```

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

Code
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes

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

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

```
[7]:  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.

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

```
[8]:  array([2, 2, 2, 2, 2])
```

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

Train/Test dataset

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

```
[ ]:  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)
```

Code
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)

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.

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

Code
from sklearn import svm

```python
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

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

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

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

Evaluation

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

Code
```python
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

```python
[ ]:  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')
```

Code

```python
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')
```

```python
# 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
```
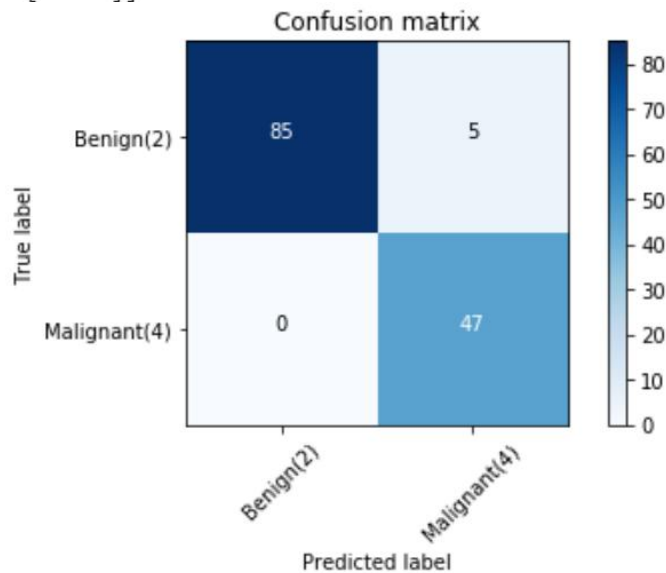
```
                precision    recall  f1-score   support

           2        1.00      0.94      0.97        90
           4        0.90      1.00      0.95        47

   micro avg        0.96      0.96      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

Code
```
# 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')
```

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

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

Code
```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

Lets try jaccard index for accuracy:

```
[ ]:  from sklearn.metrics import jaccard_similarity_score
      jaccard_similarity_score(y_test, yhat)
```

Code
```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

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?

```
[ ]:  # write your code here
```

```
Double-click __here__ for the solution.

<!-- Your answer is below:

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'))
print("Jaccard score: %.4f" % jaccard_similarity_score(y_test, yhat2))

-->
```

Code
```
Double-click __here__ for the solution.

<!-- Your answer is below:

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'))
print("Jaccard score: %.4f" % jaccard_similarity_score(y_test, yhat2))

-->
```

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

Thanks for completing this lesson! ¶

Author: Saeed Aghabozorgi