Samarth Chetan
MS in Data Analytics Engineering,
Northeastern University

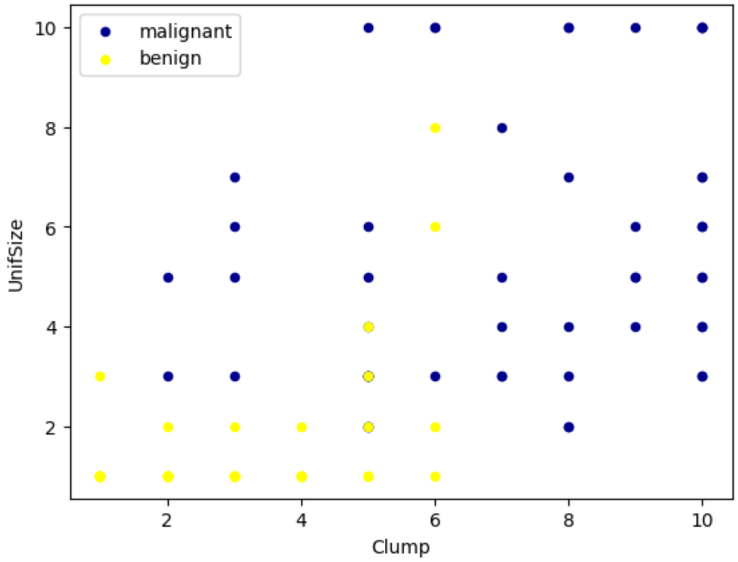
```
# importing necessary libraries to carry out analysis
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import itertools
from sklearn.metrics import f1_score
from sklearn import svm

%matplotlib inline
```

Dataset obtained from UCI.
Link: [http://mlearn.ics.uci.edu/MLRepository.html]

```
# reading the dataset into a dataframe called 'df'
df = pd.read_csv("cell_samples.csv")
df.head()
```

	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

```
# setting tje axes, type, nature of the plot and displaying it
ax = df[df['Class'] == 4][0:50].plot(kind = 'scatter', x = 'Clump', y = 'UnifSize', color = 'DarkBlue', label = 'malignant');
df[df['Class'] == 2][0:50].plot(kind = 'scatter', x = 'Clump', y = 'UnifSize', color = 'Yellow', label = 'benign', ax = ax);
plt.show()
```



```
# looking at different types of data
df.dtypes
```

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

```
# making sure that all the entries are in the suitable format for fitting our model
df = df[pd.to_numeric(df['BareNuc'], errors = 'coerce').notnull()]
df['BareNuc'] = df['BareNuc'].astype('int')
df.dtypes
```

```
    <ipython-input-43-da79b0b62ec6>:3: 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-versu
      df['BareNuc'] = df['BareNuc'].astype('int')
    ID             int64
    Clump          int64
    UnifSize       int64
    UnifShape      int64
    MargAdh        int64
    SingEpiSize    int64
    BareNuc        int64
    BlandChrom     int64
    NormNucl       int64
    Mit            int64
    Class          int64
    dtype: object
```

```
# creating a feature array and storing it under 'feature_df'
feature_df = df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
X = np.asarray(feature_df)
X[0:5]
```

```
    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]])
```

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

```
    <ipython-input-45-648ddf36b7ce>: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-versu
      df['Class'] = df['Class'].astype('int')
    array([2, 2, 2, 2, 2])
```

```
# splitting our dataset into testing and training sets
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,)
```

```
# making use of a SVM classifier for this task
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

```
# using the predictor to make predictions
yhat = clf.predict(X_test)
yhat [0:5]
```

```
array([2, 4, 2, 4, 2])
```

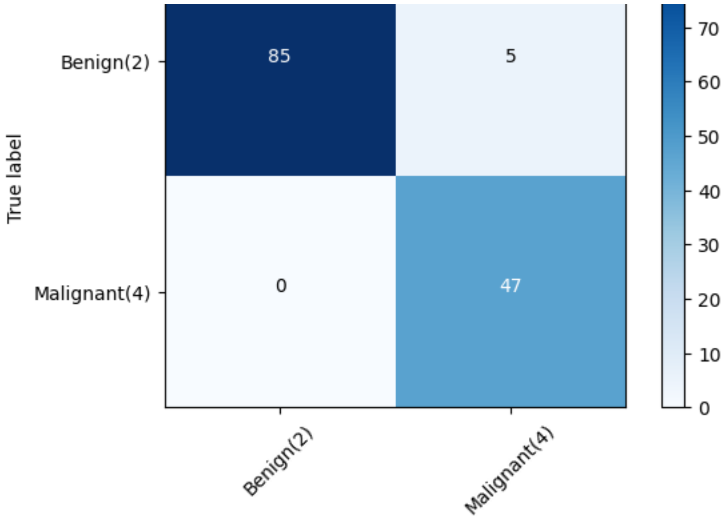
```
# creating a confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize = False,
                          title = 'Confusion matrix',
                          cmap = plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
        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')
```

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

print (classification_report(y_test, yhat))

# plotting
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]
```

```
# getting the F-1 score
f1_score(y_test, yhat, average = 'weighted')
```

```
    0.9639038982104676
```


Benign(2)
Malignant(4)
True label
Benign(2)
Malignant(4)