TP 3

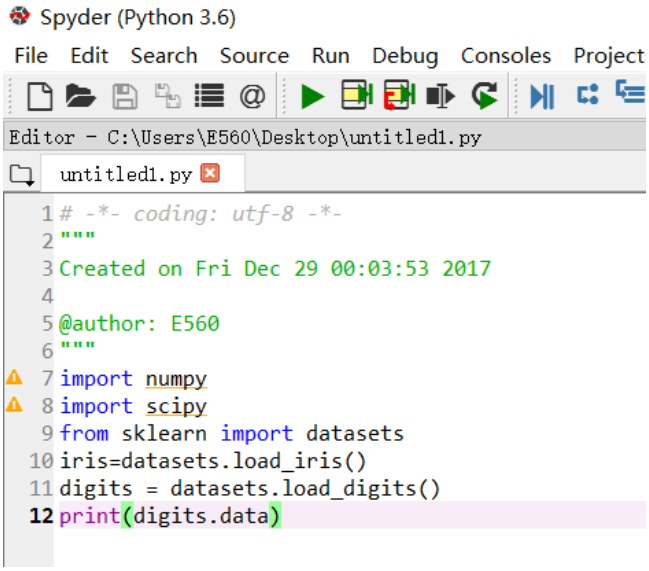
Data analysis and visualization with machine learning

(BI 1) YAO Zeliang
(BI 2) ZHANG Meng

A. An introduction to machine learning with scikit-learn

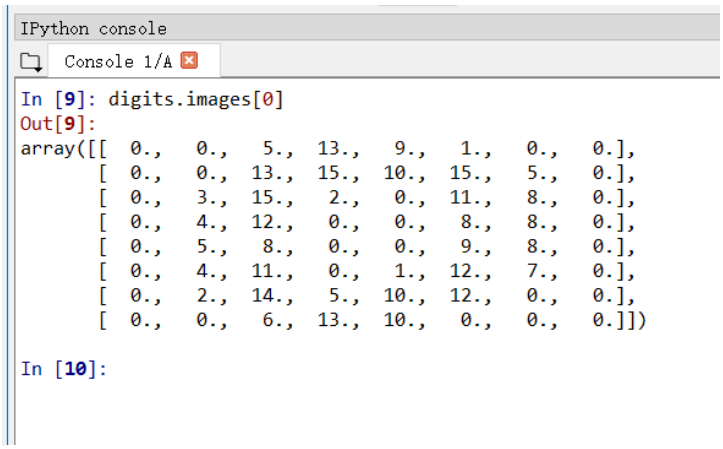
We use Spyder as our tool, and downloaded all the needed packages (Numpy, Scipy, Sklearn…)

Loading an example dataset

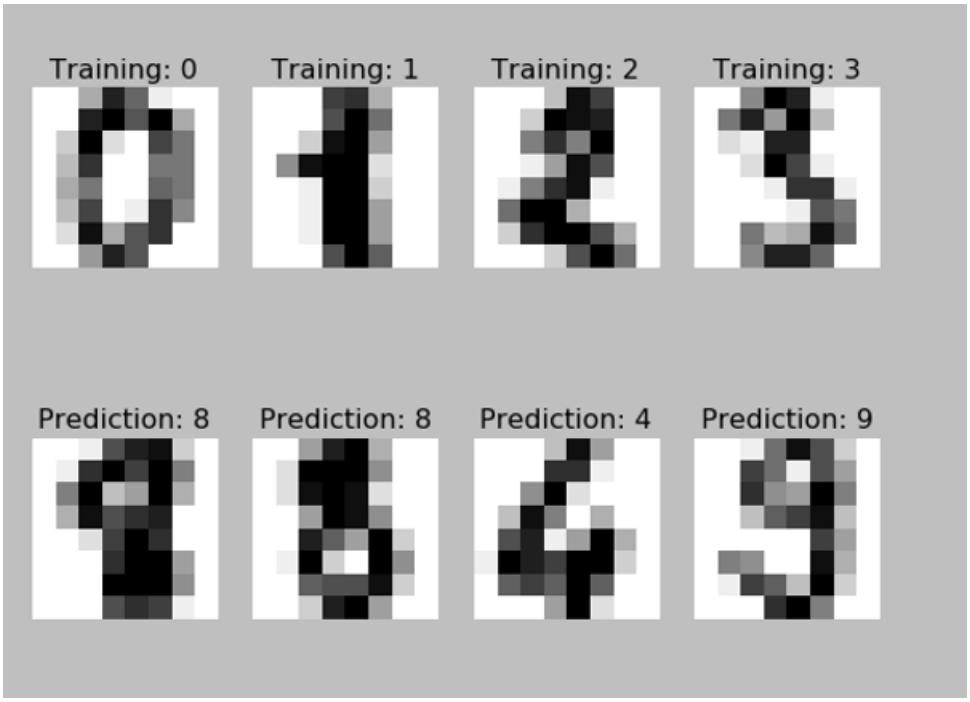



```
In [6]: digits.target
Out[6]: array([0, 1, 2, ..., 8, 9, 8])
```

```
13 digits.images[0]
14
```



The simple example on this dataset showing how the scikit-learn can be used to recognize images of hand-written digits. the result is displayed as following:



Learning and predicting

Editor - C:\Users\E560\Desktop\ui.py

ui.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Dec 29 12:12:23 2017
4
5 @author: E560
6 """
7
8 from sklearn import svm
9 clf = svm.SVC(gamma=0.001, C=100.)
10 print(clf)
11
12
```

iris.data[:3]) stands for iris.data[0], iris.data[1], iris.data[2]. iris.data[-1] stands for the last row. iris.data[-2:-1] stands for the last second row.

We select this training set with the [:-1] Python syntax, which produces a new array that contains all but the last entry of digits.data:

```
clf.fit(digits.data[:-1], digits.target[:-1]) # learn from the estimator
```

```
Out[6]:
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [7]:
```

IPython console History log

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 18 Co

```
clf.predict(digits.data[-1:])
```

```
In [8]: clf.predict(digits.data[-1:])
Out[8]: array([8])

In [9]:
```

IPython console History log

Model persistence

SVC: The clustering algorithm which provides an improvement to the support vector machines (Unsupervised)

```
 7
 8 from sklearn import datasets,svm
 9
10 iris = datasets.load_iris()
11 digits = datasets.load_digits()
12 clf = svm.SVC()
13 X,y=iris.data,iris.target
14 clf.fit(X, y)
15
```

```
In [2]: clf.fit(X, y)
Out[2]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [3]:
```

```
16
17 import pickle
18 s = pickle.dumps(clf)
19 clf2 = pickle.loads(s)
20 clf2.predict(X[0:1])
21 y[0]
22
```

```
In [4]: clf2.predict(X[0:1])
Out[4]: array([0])

In [5]: y[0]
Out[5]: 0
```

In the specific case of the scikit, it may be more interesting to use joblib's replacement of pickle (joblib.dump & joblib.load), which is more efficient on big data, but can only pickle to the disk and not to a string:

```
from sklearn.externals import joblib
joblib.dump(clf,'filename.pkl')
clf=joblib.load('filename.pkl')
```

```
['filename.pkl', 'filename.pkl_01.npy', 'filename.pkl_02.npy', 'filename.pkl_03.npy',
'filename.pkl_04.npy', 'filename.pkl_05.npy', 'filename.pkl_06.npy', 'filename.pkl_07.
npy', 'filename.pkl_08.npy', 'filename.pkl_09.npy', 'filename.pkl_10.npy', 'filename.p
kl_11.npy']
```

Conventions

Type casting

```
28 import numpy as np
29 from sklearn import random_projection
30 rng=np.random.RandomState(0)
31 X=rng.rand(10,2000)
32 X=np.array(X,dtype='float32')
33 X.dtype
34 transformer=random_projection.GaussianRandomProjection()
35 X_new=transformer.fit_transform(X)
36 X_new.dtype
37 print(X_new)
38
```

```
In [17]: X.dtype
    ...:
Out[17]: dtype('float32')

In [18]: X_new.dtype
    ...:
Out[18]: dtype('float64')

In [19]: print(X_new)
    ...:
[[-0.27654272 -0.07215213 -0.4643862  ..., -0.36098978  0.01210819
   0.23948385]
 [-0.18659335 -0.29175477 -0.03042684 ..., -0.33997058  0.17002352
  -0.50133747]
 [-0.6885818  -0.47260746 -0.27856501 ..., -0.03626912  0.10749924
  -0.12012295]
 ...,
 [-0.35957877  0.088365   -0.63337149 ..., -0.5781291   0.36724424
  -0.6619851 ]
 [-0.26259775 -0.51183755 -0.19632529 ...,  0.05222503  0.75795098
  -0.68296535]
 [-0.38909915 -0.22038241 -0.07030496 ...,  0.18810938  0.14059033
  -0.67131159]]
```

X is float32, which is cast to float64 by fit transform(X).

```
38 clf.fit(iris.data,iris.target)
39 list(clf.predict(iris.data[:3]))
40 clf.fit(iris.data,iris.target_names[iris.target])
41 list(clf.predict(iris.data[:3]))
42
```

```
In [25]: clf.fit(iris.data,iris.target)
    ...:
Out[25]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [26]: list(clf.predict(iris.data[:3]))
    ...:
Out[26]: [0, 0, 0]
In [27]: clf.fit(iris.data,iris.target_names[iris.target])
    ...:
Out[27]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [28]: list(clf.predict(iris.data[:3]))
    ...:
Out[28]: ['setosa', 'setosa', 'setosa']
```

The first predict() returns an integer array, since iris.target (an integer array) was used in fit. The second predict returns a string array, since iris.target_names was for fitting.

Refitting and updating parameters

RBF: Radial basic function

http://scikit-learn.org/stable/modules/svm.html

```
42
43
44 from sklearn.svm import SVC
45 import numpy as np
46 rng=np.random.RandomState(0)
47 X=rng.rand(100,10)
48 y=rng.binomial(1,0.5,100)
49 X_test=rng.rand(5,10)
50 clf=SVC()
51 clf.set_params(kernel='linear').fit(X,y)
52 clf.predict(X_test)
53 clf.set_params(kernel='rbf').fit(X,y)
54 clf.predict(X_test)
55
56
```

```
In [32]: clf.set_params(kernel='linear').fit(X,y)
    ...:
Out[32]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [33]: clf.predict(X_test)
    ...:
Out[33]: array([1, 0, 1, 1, 0])
In [34]: clf.set_params(kernel='rbf').fit(X,y)
    ...:
Out[34]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [35]: clf.predict(X_test)
    ...:
Out[35]: array([0, 0, 0, 1, 0])
```

Multiclass vs. multilabel fitting

```
45
46 from sklearn.svm import SVC
47 from sklearn.multiclass import OneVsRestClassifier
48 from sklearn.preprocessing import LabelBinarizer
49 X=[[1,2],[2,4],[4,5],[3,2],[3,1]]
50 y=[0,0,1,1,2]
51 classif=OneVsRestClassifier(estimator=SVC(random_state=0))
52 classif.fit(X,y).predict(X)
53
```

```
In [38]: runfile('C:/Users/E560/Desktop/
ui.py', wdir='C:/Users/E560/Desktop')
[0 0 1 1 2]
```

In the above case, the classifier is fit on a 1d array of multiclass labels and the predict() method therefore provides corresponding multiclass predictions. It is also possible to fit upon a 2d array of binary label indicators:

```
52 y=LabelBinarizer().fit_transform(y)
53 print(classif.fit(X,y).predict(X))
54
```

```
In [47]: runfile('C:/Users/E560/Desktop/ui.py', wdir='C:/
Users/E560/Desktop')
[[1 0 0]
 [1 0 0]
 [0 1 0]
 [0 0 0]
 [0 0 0]]
```

Here, the classifier is fit() on a 2d binary label representation of y, using the LabelBinarizer. In this case predict () returns a 2d array representing the corresponding multilabel predictions.

```
58
59 from sklearn.preprocessing import MultiLabelBinarizer
60 y=[[0,1],[0,2],[1,3],[0,2,3],[2,4]]
61 y=MultiLabelBinarizer().fit_transform(y)
62 print(classif.fit(X,y).predict(X))
63
```

```
In [49]: print(classif.fit(X,y).predict(X))
    ...:
[[1 1 0 0 0]
 [1 0 1 0 0]
 [0 1 0 1 0]
 [1 0 1 0 0]
 [1 0 1 0 0]]
```

In this case, the classifier is fit upon instances each assigned multiple labels. The MultiLabelBinarizer is used to binaries the 2d array of multilabels to fit upon. As a result, predict () returns a 2d array with multiple predicted labels for each instance.

B. Data normalization

1- Crate the following matrice X :

ui.py*

```
1
2
3 from sklearn import preprocessing
4 import numpy as np
5 X=np.array([[1.,-1.,2.],
6            [2., 0., 0.],
7            [0.,1.,-1.]])
8
```

2- Print the matrix and compute the mean of the variables.

```
9 print(X)
10 np.mean(X)
11 np.var(X)
12
```

```
In [3]: runfile('C:/Users/E560/Desktop/ui.py',
wdir='C:/Users/E560/Desktop')
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]

In [4]: np.mean(X)
   ...:
Out[4]: 0.4444444444444442

In [5]: np.var(X)
   ...:
Out[5]: 1.1358024691358024
```

3- Use the scale function to normalize X. Analyze the result.

```
11 X_Normalize=preprocessing.scale(X)
12 print(X_Normalize)
```

```
In [9]: runfile('C:/Users/E560/Desktop/ui.py',
wdir='C:/Users/E560/Desktop')
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
```

For analyzing if the result was normalized, we have to check if mean is close to 0, and

the variance equal to 1.

4- Compute the mean and the variance of the scaled X. What can you conclude?

```
print(np.mean(X_Normalize))
print(np.var(X_Normalize))
```

```
In [13]: print(np.mean(X_Normalize))
4.93432455389e-17

In [14]: print(np.var(X_Normalize))
1.0
```

While mean is indeed close to 0, and the variance is 1, the matrix X is exactly normalized.

C. MinMax Normalization

1- Create the following matrix X2:

```
from sklearn import preprocessing
import numpy as np
X2=np.array([[1.,-1.,2.],
             [2., 0., 0.],
             [0.,1.,-1.]])
```

2- Print the matrix and compute the mean of the variables.

```
 9 print(X2)
10 print(np.mean(X2))
11
```

```
In [16]: print(X2)
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]

In [17]: print(np.mean(X2))
0.444444444444
```

3- Normalize the data using MinMaxScaler. Print the scaled matrix and compute the mean and the variance. What can you conclude?

```
 8
 9
10 min_max_scaler=preprocessing.MinMaxScaler()
11 X2_Normalize=min_max_scaler.fit_transform(X2)
12 print(X2_Normalize)
13 print(np.mean(X2_Normalize))
14 print(np.var(X2_Normalize))
15
```

```
In [20]: print(X2_Normalize)
[[ 0.5         0.          1.        ]
 [ 1.          0.5         0.33333333]
 [ 0.          1.          0.        ]]

In [21]: print(np.mean(X2_Normalize))
0.481481481481

In [22]: print(np.var(X2_Normalize))
0.169410150892
```

MinMaxScaler transforms features by scaling each feature to a given range. By default (with no feature range), the transformation scales each feature individually to lie between 0 and 1. While scaling doesn't meet the normalization, we can't conclude from mean and variance.

D. Data visualization

1- Import the Iris dataset using : iris = datasets.load_iris()

```
1 from sklearn import datasets
2 iris = datasets.load_iris()
3 print(iris.data)
4 print(iris.target)
5 print(iris.target_names)
6 print(iris.feature_names)
```

```
In [3]: runfile('C:/Users/E560/Desktop/ui.py', wdir='C:/Users/E560/Desktop')
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]
 [ 5.4  3.7  1.5  0.2]
 [ 4.8  3.4  1.6  0.2]
 [ 4.8  3.   1.4  0.1]
 [ 4.3  3.   1.1  0.1]
 [ 5.8  4.   1.2  0.2]
 [ 5.7  4.4  1.5  0.4]
 [ 5.4  3.9  1.3  0.4]
 [ 5.1  3.5  1.4  0.3]
 [ 5.7  3.8  1.7  0.3]
 [ 5.1  3.8  1.5  0.3]
 [ 5.4  3.4  1.7  0.2]
 [ 5.1  3.7  1.5  0.4]
```

…

…

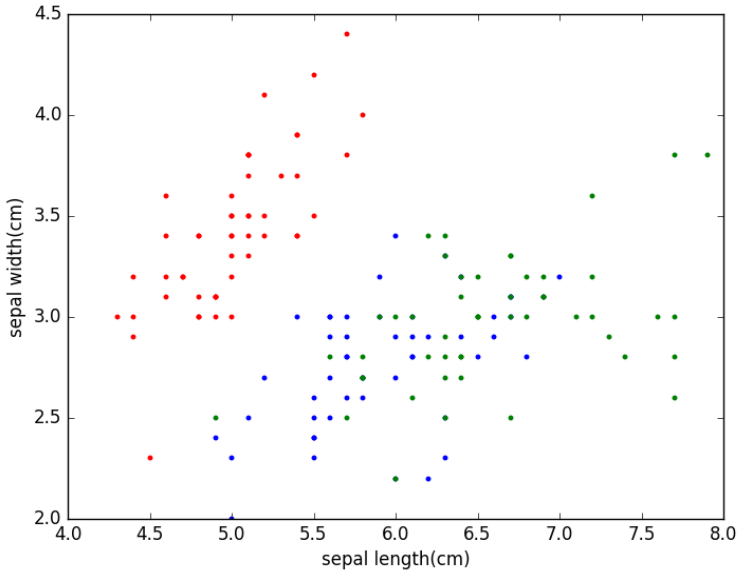
```
 [ 6.5  3.   5.2  2. ]
 [ 6.2  3.4  5.4  2.3]
 [ 5.9  3.   5.1  1.8]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

2- Plot the data points into 2D dimension with all the possible combination between variables and use the label for the color points. Visually, which is the better combination of variables? Justify the answer.

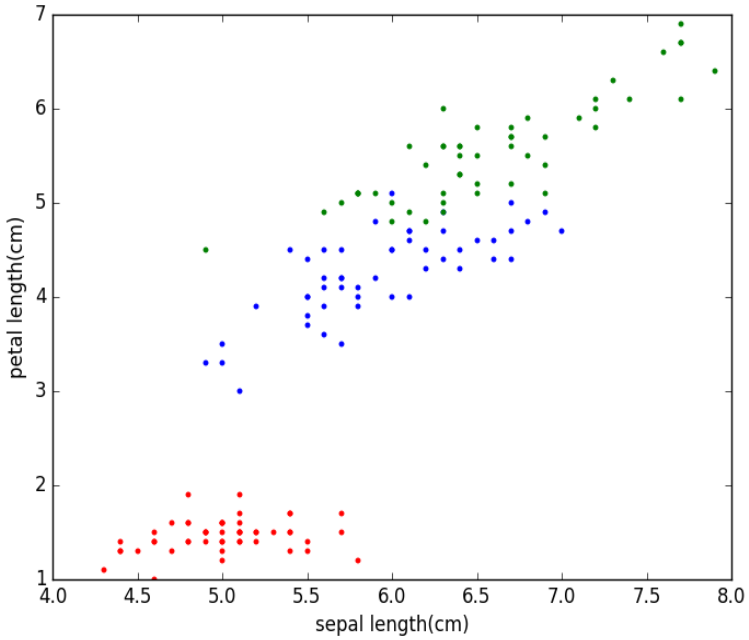
v1 ~v4 refers to iris.feature_name: sepal length ,sepal width, petal length, petal width

```
import numpy as np
import matplotlib.pyplot as plt
```

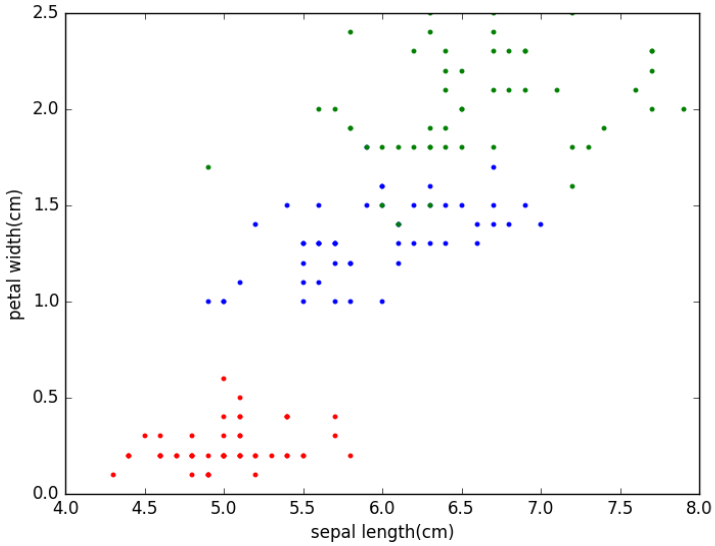
```
plt.plot(v1[:50],v2[:50],'r.',v1[50:100],v2[50:100],'b.',v1[100:],v2[100:],'g.')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.show()
```



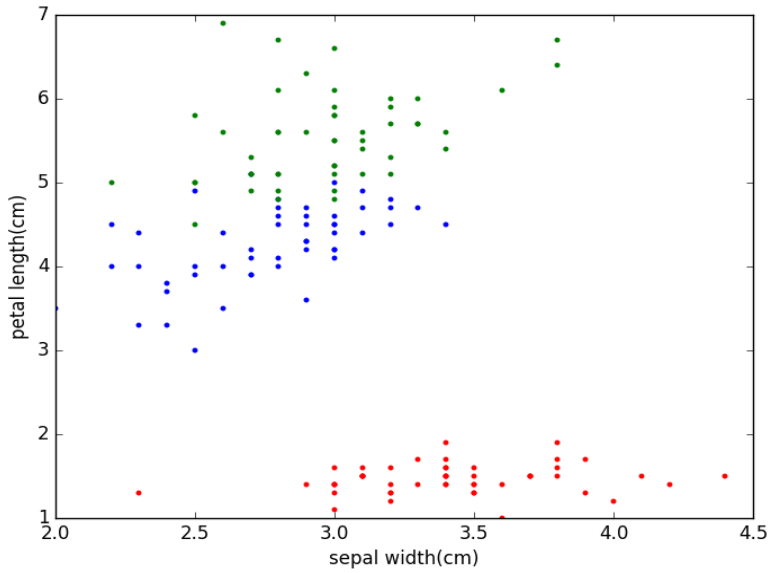

```
plt.plot(v1[:50],v3[:50],'r.',v1[50:100],v3[50:100],'b.',v1[100:],v3[100:],'g.')
plt.xlabel('sepal length(cm)')
plt.ylabel('petal width(cm)')
plt.show()
```



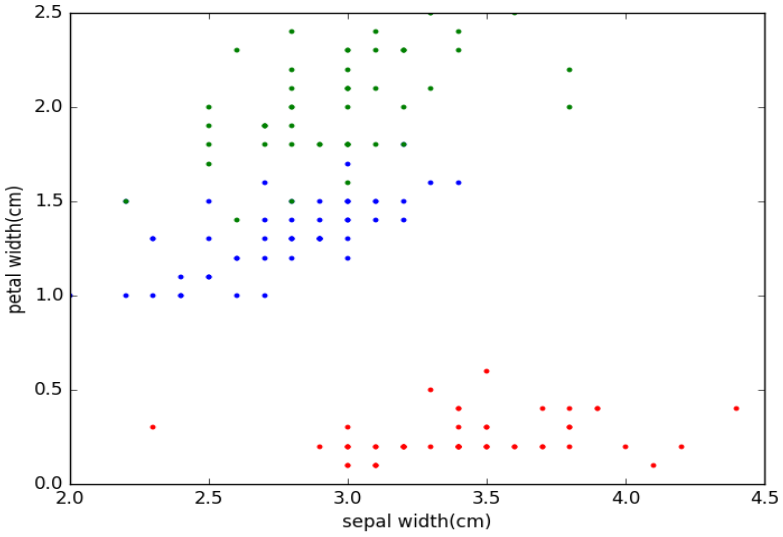

```
30
31 plt.plot(v1[:50],v4[:50],'r.',v1[50:100],v4[50:100],'b.',v1[100:],v4[100:],'g.')
32 plt.xlabel('sepal length(cm)')
33 plt.ylabel('petal width (cm)')
34 plt.show()
35
```



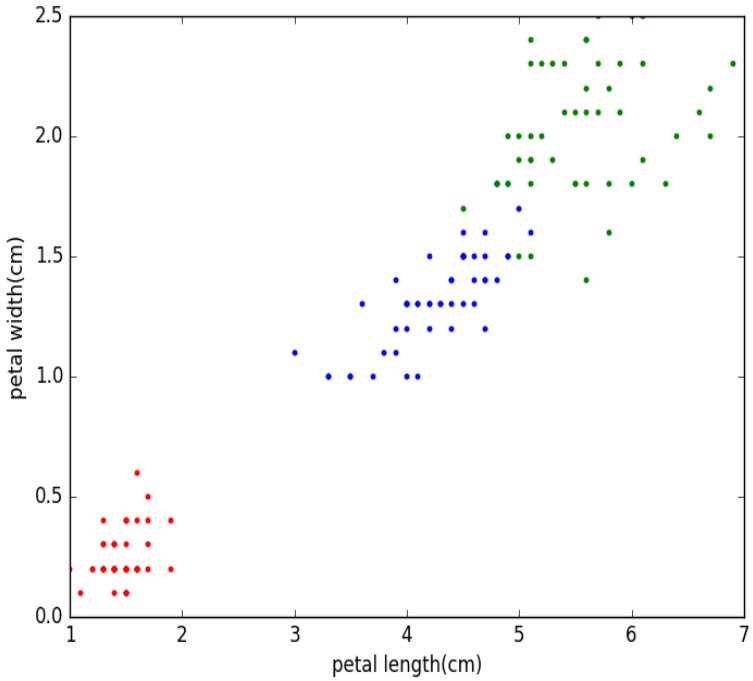

```
21 plt.plot(v2[:50],v3[:50],'r.',v2[50:100],v3[50:100],'b.',v2[100:],v3[100:],'g.')
22 plt.xlabel('sepal width(cm)')
23 plt.ylabel('petal length(cm)')
24 plt.show()
25
```




```
37 plt.plot(v2[:50],v4[:50],'r.',v2[50:100],v4[50:100],'b.',v2[100:],v4[100:],'g.')
38 plt.xlabel('sepal width (cm)')
39 plt.ylabel('petal width (cm)')
40 plt.show()
41
```




```
51
52 plt.plot(v3[:50],v4[:50],'r.',v3[50:100],v4[50:100],'b.',v3[100:],v4[100:],'g.')
53 plt.xlabel('petal length (cm)')
54 plt.ylabel('petal width (cm)')
55 plt.show()
56
```



The last combination of the variables are the best. The points are classified into different colors more apparently (and formed well in its classification).

3- Compute the correlations between each pair of variables by using the corrcoef function of numpy package. Can you validate the answer of the question C.2 by using the obtained correlations? Justify your response.

```
1
2 from sklearn import datasets
3 import numpy as np
4 iris = datasets.load_iris()
5 print(np.corrcoef(iris.data.T))
6
7
```

```
In [37]: runfile('C:/Users/E560/Desktop/ui.py', wdir='C:/Users/E560/
Desktop')
[[ 1.         -0.10936925  0.87175416  0.81795363]
 [-0.10936925  1.         -0.4205161  -0.35654409]
 [ 0.87175416 -0.4205161   1.          0.9627571 ]
 [ 0.81795363 -0.35654409  0.9627571   1.        ]]

In [38]:
```

The correlation coefficient of the last combination is the largest, which validate the answer of C.2.

E. Data reduction and visualization

1- The PCA and LDA methods can be imported from the following packages :

```
>>> from sklearn.decomposition import PCA
>>> from sklearn.lda import LDA
```

```
from sklearn.decomposition import PCA
from sklearn.lda import LDA
```

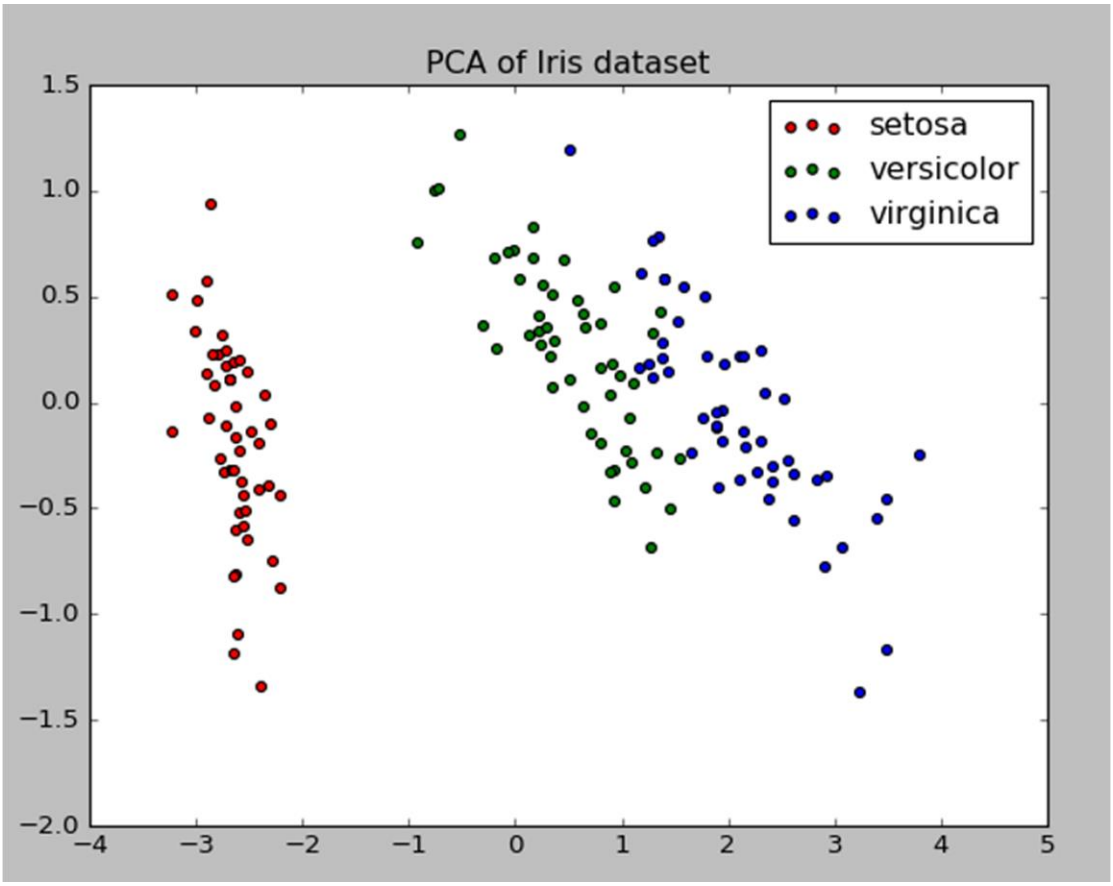
2- Analyze the help of these functions (pca and lda) and apply them on the Iris dataset. You have to use here pca.fit(Iris).transform(Iris) and save the results in IrisPCA for the PCA and IrisLDA for the LDA.

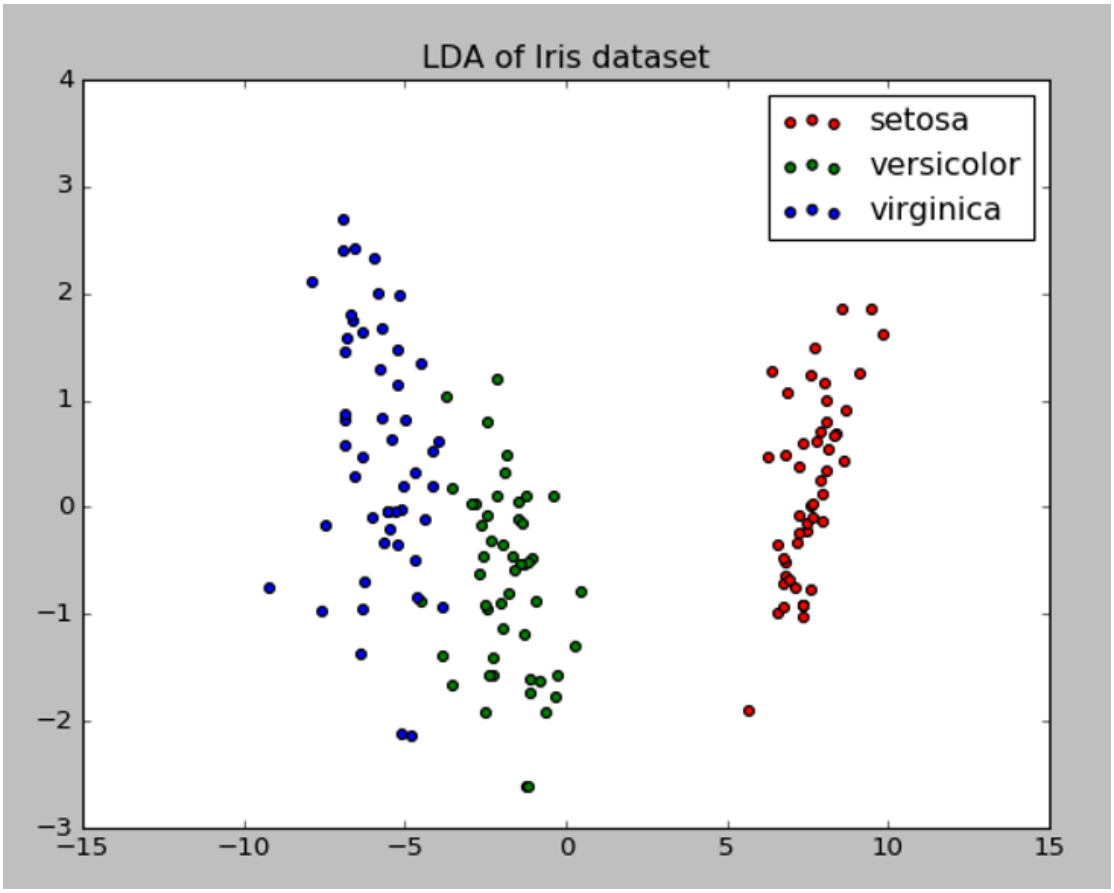
```
from sklearn.decomposition import PCA
from sklearn.lda import LDA
from sklearn import datasets
import matplotlib.pyplot as plt
iris=datasets.load_iris()
X=iris.data
y=iris.target

target_names=iris.target_names
pca=PCA(n_components=2)
a1=pca.fit(X,y).transform(X)
lda=LDA(n_components=2)
a2=lda.fit(X,y).transform(X)
```

pca function implement Principal Component Analysis to the data, and lda function implement Linear Discriminant Analysis to data. Both function can be used for looking for linear combinations of variables which best explain the data.

3- Plot the data points on the new obtained projections: one image for the PCA and another for the LDA and use the label as color for the points. You can use the following function from Phython: figure, scatter, title, xlim, ylim, xlabel, ylabel et show. Which difference you can see between the both results? Explain?

```
plt.figure()
colors=['blue','red','green']
lw=2
for color,i,target_name in zip(colors,[0,1,2],target_names):
    plt.scatter(a1[y==i,0],a1[y==i,1],color=color,label=target_name)
plt.legend()
plt.title('PCA of IRIS dataset')
plt.figure()

for color,i,target_name in zip(colors,[0,1,2],target_names):
    plt.scatter(a2[y==i,0],a2[y==i,1],color=color,label=target_name)
plt.legend()
plt.title('LDA OF IRIS dataset')
plt.show()
```





The two projections have different directions. PCA may throw away all the discriminant information, while LDA uses the labels to reduce the dimensionality. PCA is typically a good method to find a suitable subset of variables, but LDA is a better method in finding discriminant directions.