Practical Session 4

Biomedical Signals and Images

ETRO: Deparment of Electronics and Informatics

Vrije Universiteit Brussel

Jakub Ceranka, Hamza Mekhzoum, Evgenia Papavasileiou and Jef Vandemeulebroucke

Students names: Siyan Luo(0594750) & Faiza Tasnia (0563547)
Academic Year : 2022-2023

Purpose

The purpose of this exercise session is to obtain insight into feature extraction techniques applied on medical image processing as well as processing of the extracted features. For more information on these concepts see the related material on the course slides.

The jupyter notebook should be submitted as the report of each practical session by teams of **two** students. In colab you should download the notebook in the format *.ipynb and save it as a pdf version through print->save as pdf. Both the jupyter notebook and the pdf should be uploaded on canvas in a zip file before the deadline.The zip file should be named as Surname1Name1_Surname2Name2.zip. The **deadline** for the report submission is **January 2th 2022, at 23.59**. Any report sent after the deadline will not be graded.

Required libraries

During this practical session, the following libraries will be used:

- Numpy
- Simple ITK
- Matplotlib
- Sklearn
- pandas

To import any external library, you need to import it using the **import** statement followed by the name of the library.

Part 1: Feature Extraction

In the first part of this practical session you are given a CT image of a lung, together with the segmentation mask and you are asked to extract features belonging to two important groups of features; first order statistics and shape features.

Load the image

1. Download the DICOM image.
2. Use SimpleITK functions SimpleITK.ImageSeriesReader() with GetGDCMSeriesFileNames() to read the DICOM series by providing the whole path where the Dicom series is stored. Documentation can be found here: (https://simpleitk.readthedocs.io/en/master/link_DicomSeriesReader_docs.html)
3. Use function SimpleITK.ReadImage() to obtain the itk images by providing the obtained dicom series.
4. Use function SimpleITK.GetArrayFromImage() to convert the itk image of the previous step into a numpy array.

To see the documentation of a function you can type help(the name of a function). For example:
help(SimpleITK.GetImageFromArray) returns
GetImageFromArray(arr, isVector=None)
Get a SimpleITK Image from a numpy array. If isVector is True, then the Image will have a Vector pixel type, and the last dimension of the array will be considered the component index. By default when isVector is None, 4D images are automatically considered 3D vector images.

In [ ]:

```
from google.colab import drive
drive.mount('/content/gdrive')
from google.colab import files
files.upload()
```

In [ ]:

```
!pip install SimpleITK
```

In [21]:

```
import SimpleITK as sitk
# help(SimpleITK.GetImageFromArray)

# Use SimpleITK functions SimpleITK.ImageSeriesReader() with GetGDCMSeriesFileNames()
# to read the DICOM series by providing the whole path where the Dicom series is stored.
reader = sitk.ImageSeriesReader()
dicom_names = reader.GetGDCMSeriesFileNames("/content/LUNG")
# reader.SetFileNames(dicom_names)
# image = reader.Execute()
image = sitk.ReadImage(dicom_names)
# Use function SimpleITK.ReadImage() to obtain the itk images by providing the obtained dicom series.
# Use function SimpleITK.GetArrayFromImage() to convert the itk image of the previous step into a numpy array.
img_arr = sitk.GetArrayFromImage(image)

```

Now load the mask

5. Download the mask.
6. Use SimpleITK function sitk.ReadImage() to read the mask by giving as input the whole path where the mask is stored.
7. Convert the itk mask to a numpy array with function SimpleITK.GetArrayFromImage().

In [ ]:

```
# Use SimpleITK function sitk.ReadImage() to read the mask by giving as input the whole path where the mask is stored.
image_mask = sitk.ReadImage("/content/lung_mask.mhd")
# Convert the itk mask to a numpy array with function SimpleITK.GetArrayFromImage().
mask_arr = sitk.GetArrayFromImage(image_mask)

```

Note: You can find some examples of SimpleITK's functions here. (http://simpleitk.github.io/SimpleITK-Notebooks/01_Image_Basics.html) On that page it is mentioned that: "The order of index and dimensions need careful attention during conversion. ITK's Image class does not have a bracket operator. It has a GetPixel which takes an ITK Index object as an argument, which is an array ordered as (x,y,z). This is the convention that SimpleITK's Image class uses for the GetPixel method as well. While in numpy, an array is indexed in the opposite order (z,y,x)."

To observe this, get the size of the itkimage resulting from function SimpleITK.ReadImage in step 3. To do this use function GetSize()
Now get the size of the numpy image resulting from function SimpleITK.GetArrayFromImage in step 4.

In [22]:

```
# import SimpleITK as sitk
size_img = image.GetSize()
size_arr = len(img_arr)
print(size_img)
print(size_arr)

```

```
(512, 512, 134)
134
```

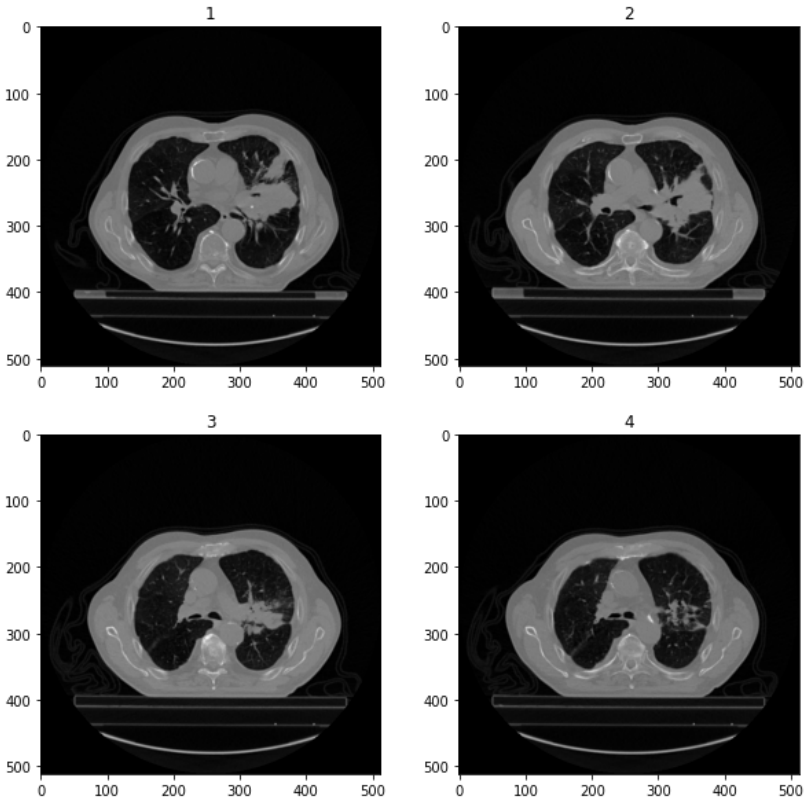
Visualize some slices of the CT image

In a 2X2 matplolib subplot visualize slices **72,77,82,84**
Use function imshow and a gray colormap

Remember python's indexes start from zero

In [ ]:

```
from pylab import show,title,figure,imshow,subplot,subplots_adjust, hist

h,w=2,2    # figure height and width
figure(figsize=(10,10))
subplots_adjust(hspace=.2)
subplot(h,w,1)
imshow(img_arr[72] , cmap='gray')
title('1')

subplot(h,w,2)
imshow(img_arr[77], cmap='gray')
title('2')

subplot(h,w,3)
imshow(img_arr[82], cmap='gray')
title('3')

subplot(h,w,4)
imshow(img_arr[84], cmap='gray')
title('4')
show()
```

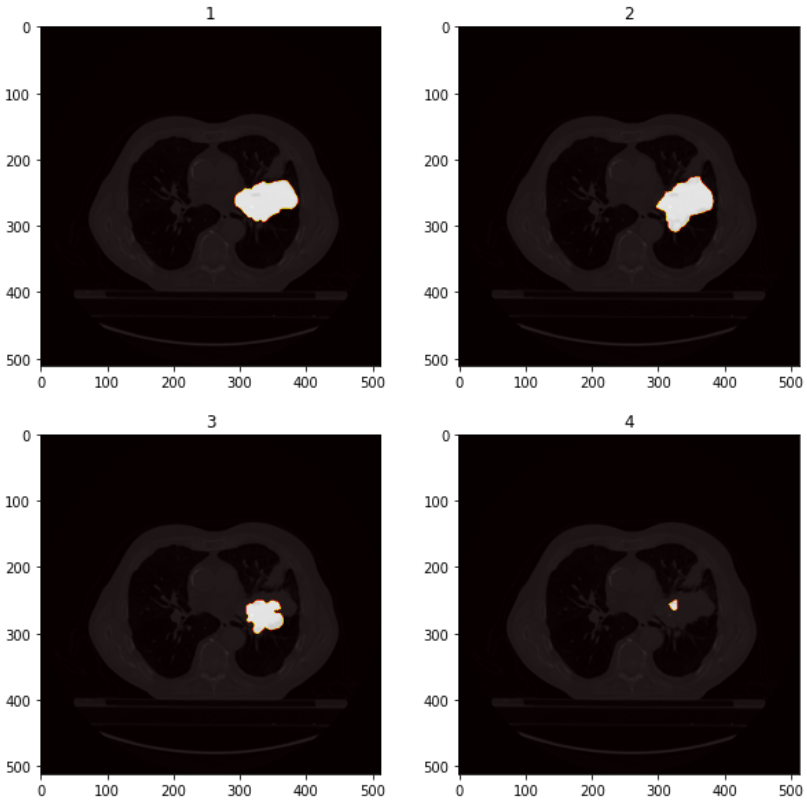


Overlay with the masks

Overlay the above slices with the masks. When plotting the masks, define a value for the parameter 'alpha' of function imshow() and use a different colormap.

In [ ]:

```
h,w=2,2    # figure height and width
figure(figsize=(10,10))
subplots_adjust(hspace=.2)
subplot(h,w,1)
imshow(img_arr[72] , cmap='gray')
imshow(mask_arr[72] , alpha=0.8, cmap='hot')
title('1')

subplot(h,w,2)
imshow(img_arr[72] , cmap='gray')
imshow(mask_arr[77], alpha=0.8, cmap='hot')
title('2')

subplot(h,w,3)
imshow(img_arr[72] , cmap='gray')
imshow(mask_arr[82], alpha=0.8, cmap='hot')
title('3')

subplot(h,w,4)
imshow(img_arr[72] , cmap='gray')
imshow(mask_arr[84], alpha=0.8, cmap='hot')
title('4')
show()
```



Feature Extraction: First Order Statistics

To begin with you are going to extract first order statistics, i.e. features that describe the distribution of voxel intensities within the image region of interest (ROI).
To obtain the ROI, take only the part of the image *where the mask has true labels, i.e. when mask==1*. Use function numpy.where() to get the coordinates and apply them to the numpy image.
Convert the obtained image to type 'float' (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.astype.html). This is necessary for the calculations that follow.

In [ ]:

```
import numpy as np

ROI_mask_index = np.where(mask_arr == 1)

ROI = img_arr[ROI_mask_index]
ROI = ROI.astype(float)

```

Let X be the set of N_p voxels included in the ROI. Given the definitions of the features, calculate the values of the features in the ROI you obtained in the previous step.
Note: you can use existing numpy functions to calculate some features. For example, numpy.mean to calculate feature 'average gray level intesity'.

1. Minimum

The minimum gray level intensity within the ROI.

In [ ]:

```
import numpy as np
X = ROI
# roi_index=np.where(mask_array==1)
# Roi=itk_array[roi_index]
minimum=np.min(X)
print("Minimum gray level intensity: ",minimum)
```

```
Minimum gray level intensity:  -1021.0
```

2. Maximum

The maximum gray level intesity within the ROI.

In [ ]:

```
maximum=np.max(X)
print("Maximum gray level intensity: ",maximum)
```

```
Maximum gray level intensity:  1040.0
```

3. Range

The range of gray values in the ROI.
$Range = max(X(i)) - min(X(i))$

In [ ]:

```
Range=maximum-minimum
print("Range :", Range)
```

```
Range : 2061.0
```

4. Average

The average gray level intesity within the ROI.
$Mean = \frac{1}{N_p} \sum_{i=1}^{N_p} X(i)$

In [ ]:

```
Mean=np.mean(X)
print("Mean: ",Mean)
```

```
Mean:  -62.91303788866646
```

5. Energy

Energy is a measure of the magnitude of voxel values in an image.

$Energy = \sum_{i=1}^{N} (X(i) + c)^2$

c is a parameter which is used to shift the intensities to prevent negative values in X.

When using CT images a good practice is to use the value 1024. Can you explain why?
Your answer here: Because the standard scale of CT images is [-1024,3071] HU. By adding 1024 to X(i), we can guarantee the result is a positive value

In [ ]:

```
c=1024
Energy=np.sum((X+c)**2)
print("Energy: ", Energy)
```

```
Energy:  52509710748.0
```

6. Root Mean Square

The squared root of the mean of the squared intensities in the ROI.
$RMS = \sqrt{\frac{1}{N_p} \sum_{i=1}^{N_p} (X(i) + c)^2}$

In [ ]:

```
RMS=np.sqrt(Energy/len(X))
print("Root Mean Square: ", RMS)
```

```
Root Mean Square:  982.0302345956065
```

7. Standard Deviation

In [ ]:

```
STD=np.std(X)
print("Standard Deviation: ", STD)
```

```
Standard Deviation:  201.73059490198818
```

Feature Extraction: Shape Features

In this exercise you are going to extract some shape features which are descriptors of the three-dimensional size and shape of the ROI and they are independent from the gray level intensity distribution in the ROI. Calculate the following features

1. Volume

$Volume \approx N_p * V_i$

You can approximate volume by multiplying the total number of pixels with the volume of each pixel. To do this you need to get the size of a pixel by using the function GetSpacing(). The spacing is the geometric distance between image samples along each dimension.

In [ ]:

```
volume_pixel1=image.GetSpacing()[0]
volume_pixel2=image.GetSpacing()[1]
volume_pixel3=image.GetSpacing()[2]

Volume=np.size(X)*volume_pixel1*volume_pixel2*volume_pixel3

print("Volume: ",Volume)
```

```
Volume:  155779.83856201172
```

3. Surface Area to Volume ratio

Given that the surface area is 23501.6761259, calculate the surface area to volume ratio.
This feature gives an insight of the shape of the ROI. A lower value indicates a more compact (sphere-like) shape.

In [ ]:

```
surface_area=23501.6761259
S_t_V=surface_area/Volume
print("Surface area to volume ratio", S_t_V)
```

```
Surface area to volume ratio 0.15086468404924314
```

4. Compactness

a measure of how compact the shape of the tumor is relative to a sphere (most compact). It is therefore correlated to Sphericity and redundant. It is provided here for completeness. The value range is [0,1/(6pi)], where a value of 16π indicates a perfect sphere.

$Compactness = \frac{Volume}{\sqrt{\pi \cdot Surface^3}}$

In [ ]:

```
Compactness=Volume/np.sqrt(np.pi * surface_area**3)
print("Compactness: ",Compactness)
```

```
Compactness:  0.024394282987816377
```

5. Shperical Disproportion

Spherical Disproportion is the ratio of the surface area of the tumor region to the surface area of a sphere with the same volume as the tumor region, and by definition, the inverse of Sphericity. Therefore, the value range is greater than 1 with a value of 1 indicating a perfect sphere.

$SphericalDisporportion = \frac{Surface}{4 \cdot \pi \cdot R^2}$ where R the radius of a sphere with volume equal to the volume of the tumor, i.e. $R = \sqrt[3]{\frac{3 \cdot Volume}{4 \cdot \pi}}$

In [ ]:

```
R=np.cbrt((3*Volume)/(4*np.pi))
SphericalDisporportion= surface_area/(4*np.pi*R**2)
print("SphericalDisporportion: ",SphericalDisporportion)
```

```
SphericalDisporportion:  1.6785742100053986
```

Part 2: Feature Visualization

In this exercise you are given the values of the above features extracted from the CT images of 122 patients. Download the file features.csv that contains all the features extracted from these patients.

- Load the features in pandas dataframe using function pd.read_csv() (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html). Define ';' as the delimeter.
- Print the first 10 rows of the dataframe using function head() (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.head.html)
- Drop the last column of the frame using function drop (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html) The last column of this dataframe contains the labels assigned to each lesion found in the CT scan, i.e. benign (no cancer) or malignant.

In [ ]:

```
import pandas as pd
df = pd.read_csv('/content/features.csv',delimiter=';')
```

- Print the first 10 rows of the dataframe using function head() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.head.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.head.html)

In [ ]:

```
df.head(10)
```

Out[26]:

	min	max	range	average	energy	rms	std	volume	surfacde	surfaceovervolume	sphercal disporportion	compactness
0	-1024	615	1639	-49.124253	7183072911	239.522345	234.430718	358532.54670	35689.207210	0.099542	1.462291	0.030002
1	-961	230	1191	-100.581455	868119204	267.731628	248.120124	34680.90216	9287.689178	0.267804	1.805912	0.021860
2	-994	607	1601	-47.559032	863753559	171.308006	164.573908	84208.48846	19128.126360	0.227152	2.058817	0.017959
3	-1024	2621	3645	2.184743	1016158941	186.989867	186.977103	83221.56539	19715.454000	0.236903	2.138777	0.016961
4	-966	488	1454	-66.045163	829535347	241.757718	232.561455	40642.89029	8104.716252	0.199413	1.417742	0.031427
5	-1024	250	1274	-0.368995	977138398	168.774045	168.773641	98232.48845	22623.867650	0.230309	2.197418	0.016287
6	-1022	278	1300	-202.641291	3124963798	335.461261	267.340167	79518.94740	15885.806820	0.199774	1.776416	0.022407
7	-1024	348	1372	-196.001238	4705189658	391.527095	338.935069	87894.93938	14212.063330	0.161694	1.486610	0.029269
8	-999	1062	2061	-54.506685	7088061300	306.172019	301.281142	216524.40380	26208.432130	0.121041	1.502977	0.028792
9	-1024	601	1625	-184.827145	5508300763	378.702329	330.536504	109984.64950	17154.506130	0.155972	1.545276	0.027618

Drop the last column of the frame using function drop https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html) The last column of this dataframe contains the labels assigned to each lesion found in the CT scan, i.e. benign (no cancer) or malignant.

In [ ]:

```
dff=df.drop(['label'],axis=1)
label=df['label']
dff.head()
```

Out[27]:

	min	max	range	average	energy	rms	std	volume	surfacde	surfaceovervolume	sphercal disporportion	compactness
0	-1024	615	1639	-49.124253	7183072911	239.522345	234.430718	358532.54670	35689.207210	0.099542	1.462291	0.030002
1	-961	230	1191	-100.581455	868119204	267.731628	248.120124	34680.90216	9287.689178	0.267804	1.805912	0.021860
2	-994	607	1601	-47.559032	863753559	171.308006	164.573908	84208.48846	19128.126360	0.227152	2.058817	0.017959
3	-1024	2621	3645	2.184743	1016158941	186.989867	186.977103	83221.56539	19715.454000	0.236903	2.138777	0.016961
4	-966	488	1454	-66.045163	829535347	241.757718	232.561455	40642.89029	8104.716252	0.199413	1.417742	0.031427

Normalize your features in the range $[0, 1]$. A normalized feature is obtained like this

$$Normalized\ Feature = \frac{original\ Feature - min(Feature)}{max(Feature) - min(Feature))}$$

However, using scikit-learn you can do that with this function (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html)

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(dff)
normalized_dff=scaler.transform(dff)
print(normalized_dff)
```

```
[[0.         0.18242344 0.24868217 ... 0.04323827 0.13567252 0.68976912]
 [0.15       0.05426099 0.10976744 ... 0.31976044 0.33977194 0.3996299 ]
 [0.07142857 0.17976032 0.23689922 ... 0.25295249 0.48998872 0.26059142]
 ...
 [0.51666667 0.00765646 0.01860465 ... 0.93878644 0.10328467 0.75248568]
 [0.47142857 0.01098535 0.0275969  ... 0.63183732 0.06729892 0.82969199]
 [0.18333333 0.01131824 0.06542636 ... 0.47942076 0.1716458  0.62662645]]
```

In [ ]:

```
import numpy as np
be_index=np.where(df['label']==0)
ma_index=np.where(df['label']==1)

column=dff.columns
```

Get two sets of datasets; one where the label is benign (benignData) and one where the label is malignant (MalignantData)

In [ ]:

```
be_data=normalized_dff[be_index]
ma_data=normalized_dff[ma_index]

benignData=pd.DataFrame(data=be_data,columns=column) #creating dataframe for benign
MalignantData=pd.DataFrame(data=ma_data,columns=column) #creating dataframe for maligant
```

In [ ]:

```
#benign data
benignData.head()
```

Out[31]:

	min	max	range	average	energy	rms	std	volume	surfacde	surfaceovervolume	sphercal disporportion	compactness
0	0.000000	0.182423	0.248682	0.807346	1.000000	0.302702	0.620971	0.542413	0.733007	0.043238	0.135673	0.689769
1	0.150000	0.054261	0.109767	0.732371	0.119012	0.349388	0.666168	0.051161	0.180696	0.319760	0.339772	0.399630
2	0.071429	0.179760	0.236899	0.809627	0.118403	0.189809	0.390332	0.126290	0.386554	0.252952	0.489989	0.260591
3	0.000000	0.850200	0.870698	0.882106	0.139664	0.215762	0.464299	0.124793	0.398841	0.268978	0.537482	0.225040
4	0.138095	0.140146	0.191318	0.782692	0.113629	0.306402	0.614799	0.060205	0.155949	0.207366	0.109212	0.740556

In [ ]:

```
#maligant data
MalignantData.head()
```

Out[32]:

	min	max	range	average	energy	rms	std	volume	surfacde	surfaceovervolume	sphercal disporportion	compactness
0	0.369048	0.064581	0.090853	0.729380	0.079071	0.359726	0.685673	0.032222	0.093439	0.258494	0.062822	0.839911
1	0.033333	0.349867	0.400310	0.837791	0.449567	0.397962	0.823381	0.157897	0.383226	0.176416	0.313620	0.429013
2	0.726190	0.185752	0.157209	0.880921	0.010209	0.090968	0.215356	0.029329	0.148356	0.506748	0.545380	0.219447
3	0.135714	0.051265	0.108837	0.699806	0.167530	0.343373	0.618695	0.074279	0.266093	0.319788	0.478321	0.269861
4	0.100000	0.342210	0.384496	0.936112	0.159018	0.328908	0.680041	0.075417	0.250831	0.289606	0.400891	0.337494

Print the mean and standard deviation of all the features of the benign and malignant datasets separately

In [ ]:

```
stats1=benignData.describe()
stats1.head()
```

Out[33]:

	min	max	range	average	energy	rms	std	volume	surfacde	surfaceovervolume	sphercal disporportion	compac
count	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.0
mean	0.182403	0.192067	0.233910	0.779848	0.193957	0.264923	0.470261	0.171963	0.351346	0.240227	0.267617	0.5
std	0.215762	0.192145	0.183944	0.173397	0.219193	0.180587	0.222559	0.188817	0.264855	0.146503	0.188470	0.2
min	0.000000	0.002330	0.000000	0.000000	0.000000	0.000000	0.000000	0.002568	0.011501	0.000000	0.023472	0.0
25%	0.006548	0.055093	0.112093	0.716035	0.047830	0.142442	0.313014	0.056142	0.159099	0.137894	0.134011	0.3

In [ ]:

```
mean_b=stats1.iloc[1] #mean
std_b=stats1.iloc[2] #std
print("Mean and Standard deviation of all features of the benign data \n")
print("Mean \n", mean_b)
print()
print("Standard deviation \n", std_b)
```

```
Mean and Standard deviation of all features of the benign data

Mean
 min                      0.182403
max                      0.192067
range                    0.233910
average                  0.779848
energy                   0.193957
rms                      0.264923
std                      0.470261
volume                   0.171963
surfacde                 0.351346
surfaceovervolume        0.240227
sphercal disporportion   0.267617
compactness              0.533461
Name: mean, dtype: float64

Standard deviation
 min                      0.215762
max                      0.192145
range                    0.183944
average                  0.173397
energy                   0.219193
rms                      0.180587
std                      0.222559
volume                   0.188817
surfacde                 0.264855
surfaceovervolume        0.146503
sphercal disporportion   0.188470
compactness              0.216510
Name: std, dtype: float64
```

In [ ]:

```
stats2=MalignantData.describe()
stats2.head()
mean_m=stats2.iloc[1] #mean
std_m=stats2.iloc[2] #std
print("Mean and Standard deviation of all features of the maligant data \n")
print("Mean \n", mean_m)
print()
print("Standard deviation \n", std_m)
```

```
Mean and Standard deviation of all features of the maligant data

Mean
 min                      0.230172
max                      0.085225
range                    0.128169
average                  0.682161
energy                   0.069107
rms                      0.332215
std                      0.526026
volume                   0.061274
surfacde                 0.162306
surfaceovervolume        0.427097
sphercal disporportion   0.220892
compactness              0.601598
Name: mean, dtype: float64

Standard deviation
 min                      0.185202
max                      0.101578
range                    0.103652
average                  0.178778
energy                   0.085923
rms                      0.193349
std                      0.218694
volume                   0.084512
surfacde                 0.182559
surfaceovervolume        0.232530
sphercal disporportion   0.178147
compactness              0.232613
Name: std, dtype: float64
```

Use matplotlib's violin plot (https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.violinplot.html) to plot the distribution of the first feature in the benign and the malignant dataset. You can find an example here (https://www.tutorialspoint.com/matplotlib/matplotlib_violin_plot.htm)

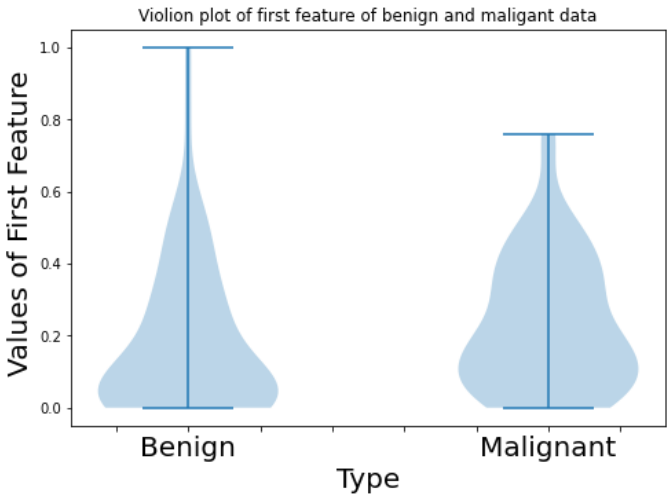
In [ ]:

```
#first feature  is min
ff_b=benignData['min']
ff_m = MalignantData['min']
```

In [ ]:

```
import matplotlib.pyplot as plt

## combine these different collections into a list
data_to_plot = [ff_b,ff_m]

# Create a figure instance
fig = plt.figure()

# Create an axes instance
ax = fig.add_axes([0,0,1,1])

# Create the boxplot
plt.violinplot(data_to_plot)
ax.set_xticklabels(['','','Benign', '','', '','','Malignant'], fontsize = 20)
plt.title("Violion plot of first feature of benign and maligant data")
plt.xlabel('Type', fontsize = 20)
plt.ylabel(' Values of First Feature', fontsize = 20)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarra
y from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shape
s) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
/usr/local/lib/python3.8/dist-packages/numpy/core/fromnumeric.py:1970: VisibleDeprecationWarning: Creating an ndarray f
rom ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes)
is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  result = asarray(a).shape
```



Plot the boxplots of the benign ad malignant datasets in two separate graphs. A boxplot shows the distribution of a feature using its quartiles. To do this you can use function pyplot.boxplot().

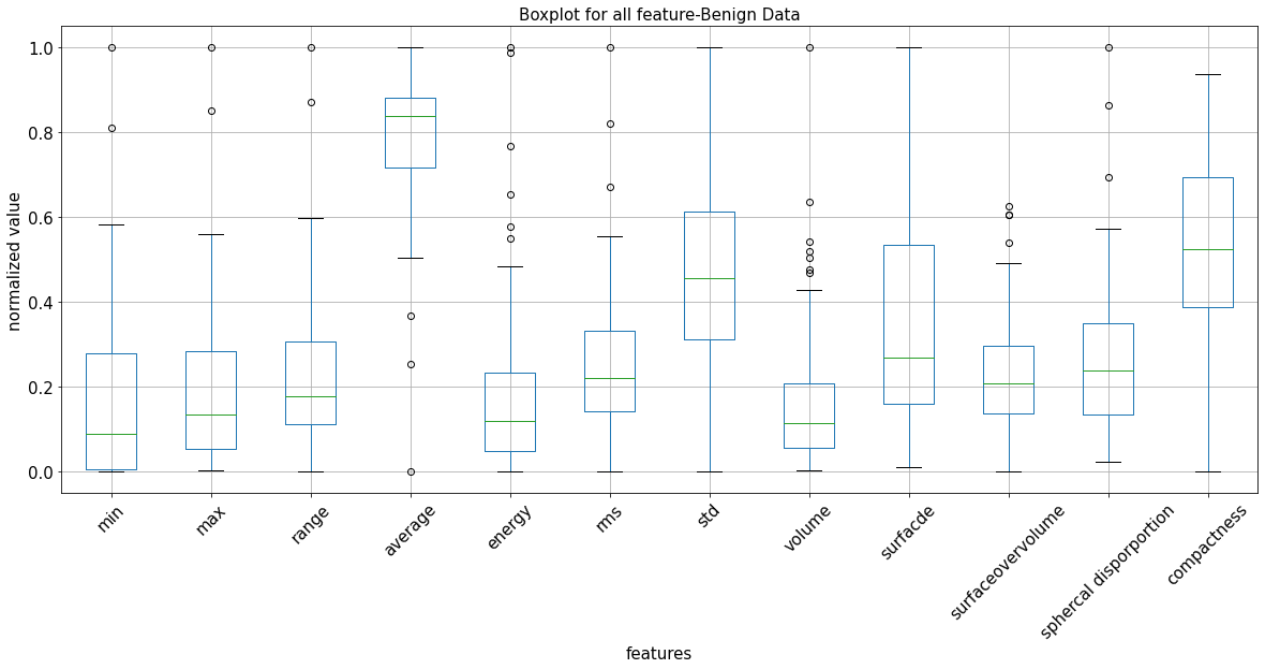
In [ ]:

```
print(column.values)
```

```
['min' 'max' 'range' 'average' 'energy' 'rms' 'std' 'volume' 'surfacde'
 'surfaceovervolume' 'sphercal disporportion' 'compactness']
```

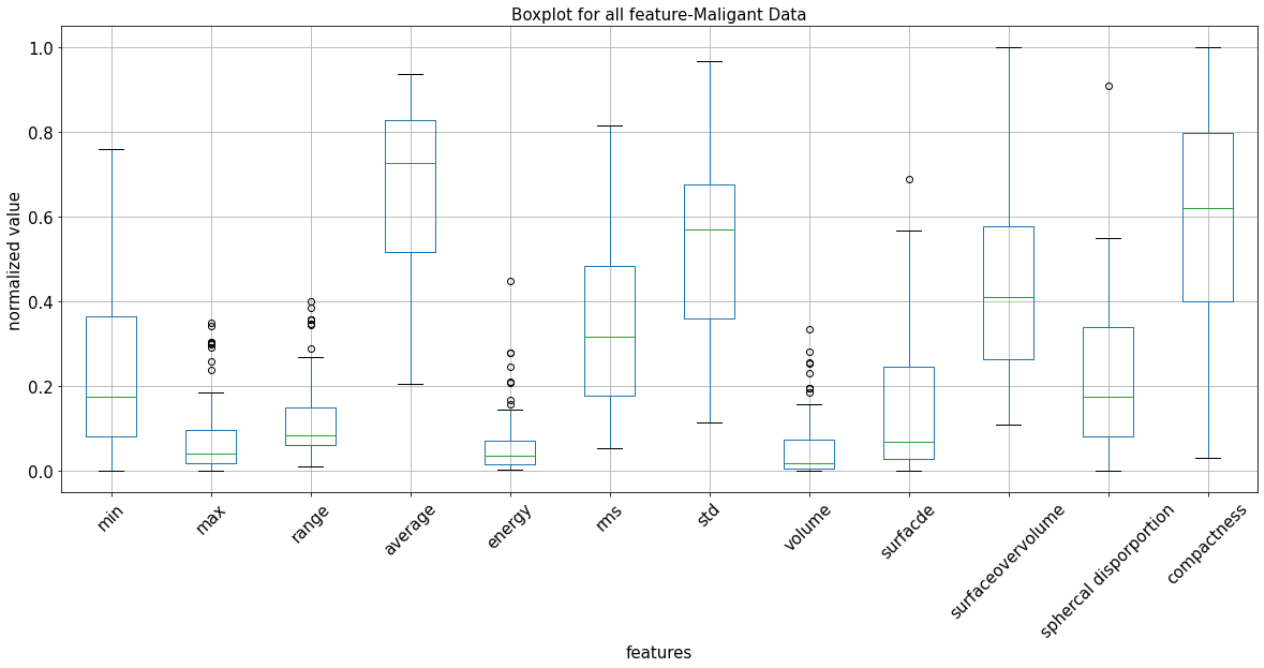
In [ ]:

```
fig = plt.figure(figsize=(20,8))
plt.title("Boxplot for all feature-Benign Data", fontsize=15)
plt.xlabel("features", fontsize=15)
plt.ylabel("normalized value", fontsize=15)
boxplot = benignData.boxplot(column=['min','max','range', 'average','energy', 'rms','std','volume', 'surfacde', 'surfaceovervolum
                                     'compactness'], rot=45, fontsize=15)
```



In [ ]:

```
fig = plt.figure(figsize=(20,8))
plt.title("Boxplot for all feature-Maligant Data", fontsize=15)
plt.xlabel("features", fontsize=15)
plt.ylabel("normalized value", fontsize=15)
boxplot = MalignantData.boxplot(column=['min','max','range', 'average','energy', 'rms','std','volume', 'surfacde', 'surfaceovervo
                                     'compactness'], rot=45, fontsize=15)
```



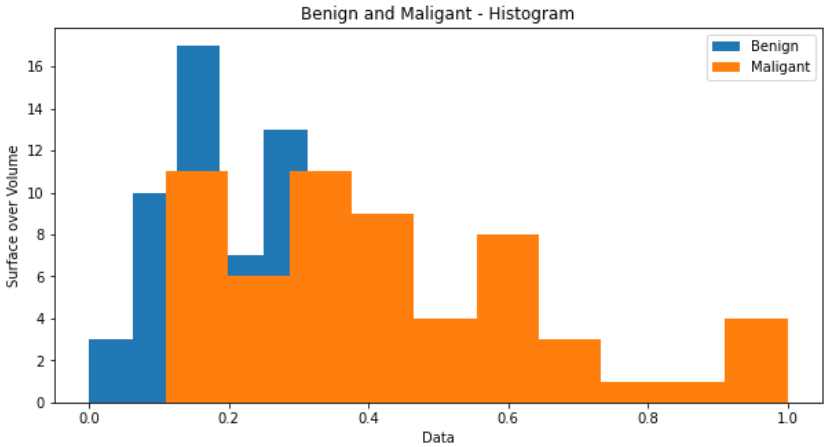
Plot the histogram of the feature surface over volume of the benign and malignant datasets in one graph. The histogram shows the distribution of a single feature. To do this you can use function pyplot.hist.

In [ ]:

```
surface_volume_b = benignData['surfaceovervolume']
surface_volume_m = MalignantData['surfaceovervolume']
```

In [ ]:

```
fig = plt.figure(figsize=(10,5))
plt.hist(surface_volume_b, label='Benign')
plt.hist(surface_volume_m, label='Maligant')
plt.ylabel('Surface over Volume')
plt.xlabel('Data')
plt.title('Benign and Maligant - Histogram')
plt.legend()
plt.show()

```



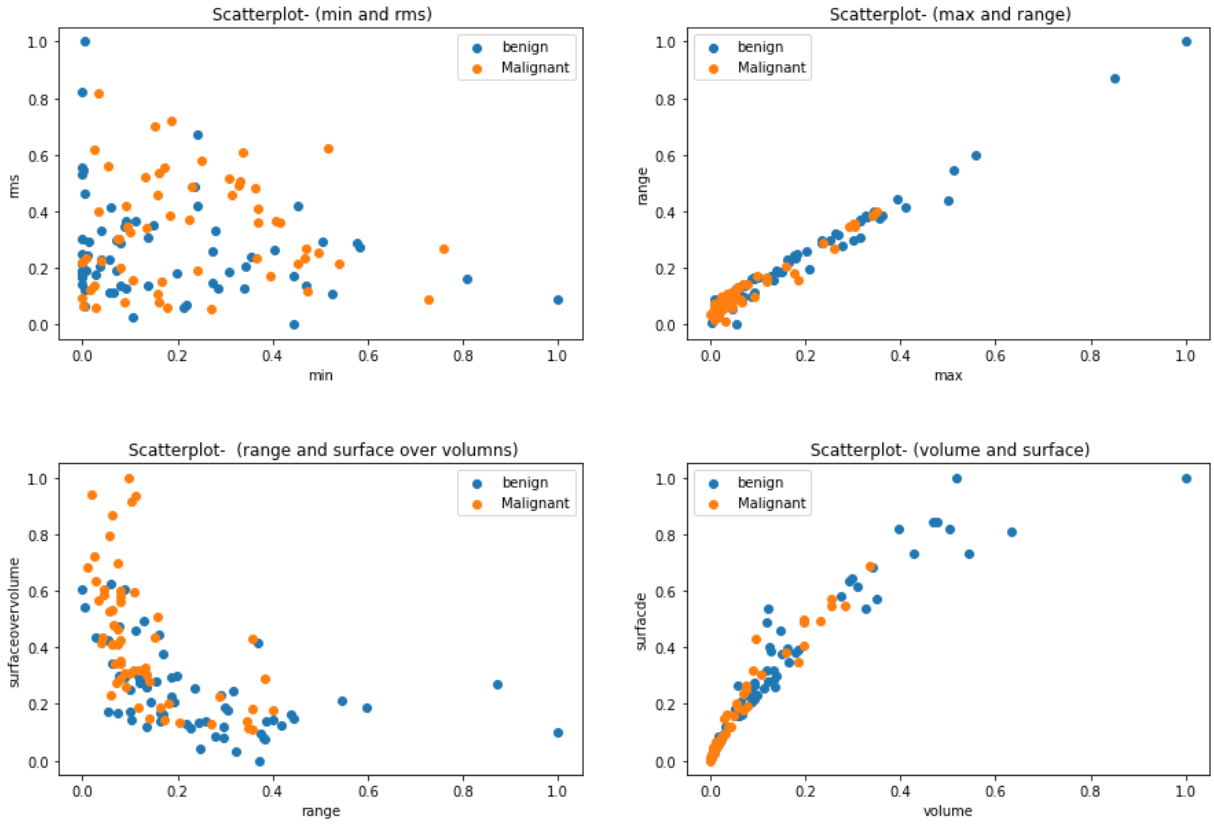
Finally, plot in a 2X2 figure the scatter plots of the following combinations of features:

- min and rms
- max and range
- range and surface over volume
- volume and surface

A scatterplot shows the relation between two features.

In [ ]:

```
h,w=2,2    # figure height and width
figure(figsize=(15,10))
subplots_adjust(hspace=.4)

subplot(h,w,1)
plt.scatter(benignData['min'],benignData['rms'],label='benign')
plt.scatter(MalignantData['min'],MalignantData['rms'],label='Malignant')
plt.xlabel('min')
plt.ylabel('rms')
plt.legend()
title(' Scatterplot- (min and rms)')

subplot(h,w,2)
plt.scatter(benignData['max'],benignData['range'],label='benign')
plt.scatter(MalignantData['max'],MalignantData['range'],label='Malignant')
plt.legend()
plt.xlabel('max')
plt.ylabel('range')
title(' Scatterplot- (max and range)')

subplot(h,w,3)
plt.scatter(benignData['range'],benignData['surfaceovervolume'],label='benign')
plt.scatter(MalignantData['range'],MalignantData['surfaceovervolume'],label='Malignant')
plt.legend()
plt.xlabel('range')
plt.ylabel('surfaceovervolume')
title(' Scatterplot-  (range and surface over volumns)')


subplot(h,w,4)
plt.scatter(benignData['volume'],benignData['surfacde'],label='benign')
plt.scatter(MalignantData['volume'],MalignantData['surfacde'],label='Malignant')
plt.legend()
plt.xlabel('volume')
plt.ylabel('surfacde')
title(' Scatterplot- (volume and surface)')
show()
```



Describe the relationships you see between the features

Your answer here: In the scatter plots(0,1), we can observe that both for benign and malignant data, the relationship bwtween max and range is linear. In the scatter plots(0,1), the relationship between volume and surface is also linear, but less fitted than the prevoius one. In the other two plots, the relationships are non-linear.

Part 3: A simple classifier

Take the features resulting from dropping the last column of the dataframe in the beginning of part 2. This is the set of features describing the ROIs of the 122 patients. The dropped column is the label assigned to the ROI.
In this exercise you will train a classifier with a portion of the features' set and use it to predict the label of an unseen set of features

1. Divide your dataset in training and testing parts so that 70% of the dataset is a training set and 30% a test set. (use random state=40). https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
2. Normalize the training set and apply the normalization on the test set. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler)
3. Use the normalized training set to train an SVM classifier and predict on the unseen testing set. Use the default values of the classifier found here (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)
4. Calculate the score of your classifier using this function (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.score)

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC

label = df["label"]
# print(label)
dataset = dff
X_train, X_test, y_train, y_test = train_test_split(dataset, label, train_size=0.3, random_state=40)
# print(y_train)

scaler = MinMaxScaler()
scaler.fit(X_train)
normalized_training=scaler.transform(X_train)
scaler.fit(X_test)
normalized_test=scaler.transform(X_test)

clf = SVC()
clf.fit(normalized_training, y_train)
score = clf.score(normalized_test,y_test)
print(score)
```

```
0.47674418604651164
```