Minia University
Faculty of Engineering
Computer&System Dep.

Study of Detection Brain Tumor with machine learning

By

Mustafa Hamdi
Muhammad Fathi
Muhammad Kamal
Samar Muhammad
Shoruk Ahmed
AL.Shaimaa Taha
Hager Atef

Supervised by
Prof. Dr. Reda R.Gharieb

July 2017

Acknowledgment

First of all, we Thank ALLAH.

The project team members would like to express their thanks to everybody at the Computers and Systems department at Minia University for their creative suggestions and precious help with any problems that arose during our work.

We would like to specially thank ***Prof. Dr. Reda R.Gharieb and Prof. Dr. M.Moness*** *for their priceless support and help during the whole project course. And also we thank the teaching assistances in the department for their great efforts and for giving us the advices that have helped us to achieve our project's goals.*

Finally, we thank all the department technicians and administration staffs who helped us during our work times. Great thanks to all of our colleagues who helped us.

Preface

Tumor is an uncontrolled growth of tissues in any part of the body. Tumors are of different types and hence they have different treatments. Detection of tumor in the earlier stages makes the treatment possible.

Here we review different segmentation methods associated with feature extraction from Magnetic Resonance Imaging (MRI) of brain. We also discuss different machine learning and classification algorithms that use to classify normal and cancerous tissues.

Finally, we propose an automatic tumor detection system.

List of Figures

Figure 1.1 Detection stages 7
Figure 1.2 Project stages 9
Figure 3.1 Features steps 22
Figure 3.3 Symmetric matrix 25
Figure 3.4 Co-occurenence matrix 25
Figure 3.5 Different histogram for different contrast 27
Figure 3.61 Rondom forests 36
Figure 3.17 Brain tumor 37
Figure 3.18 Before and after normalization 39
Figure 4.1 SVM desciption 43
Figure 4.2 Scenario a 43
Figure 4.3 Margin 44
Figure 4.8 Radia kernel 46

Table of Contents

Acknowledgment II
Preface III
Table of Contents V
Chapter 1 Introduction 7
1.1. Preprocessing 8
1.2. Segmentation 8
1.3. Feature reduction 8
1.4. Feature classification 8
Chapter 2 Preprocessing 10
2.1. Introduction: 10
2.2. Magnetic resonance imaging (MRI) 10
2.2.1. Types of MRI: 10
2.3. First: Convert to gray style: 10
2.4. Types of image noise: 11
2.5. Second: Filters: 12
2.5.1. linear filter: 12
2.5.2. Nonlinear filter: 12
2.5.2.1. Types of nonlinear filter ..... **Error! Bookmark not defined.**
2.6. Third: Binarization using threshold 14
Types of thresholds 14
2.6.1.3. THRESH_TRUNC 14
2.6.1.4. THRESH_TOZERO 15
2.6.1.5. THRESH_TOZERO_INV 15
2.7. Fourth: Morphological transformations 16
2.7.1. Morphological operators 16
2.8. Fifth: Removing edges 18
2.9. Sixth: Threshold again 18
2.10. Seventh: Edge Detector 19
Chapter 3 Feature 22
3.1. Introduction 21
3.2. Feature selection 22
3.3. Feature extraction 33
●Feature Reduction in this project: 38

Chapter 4 Machine Learning 43

4.1. Why classification and classification problem 41
4.2. Classification algorithms used in this project 41
4.3. evaluation classifier 47

References 59

Appendix (A) Abbreviations 61

Appendix (B) Developed Codes 62

Members Rule 86

Chapter 1 Introduction

A brain tumor is any mass that results from abnormal and uncontrolled cells growing in the brain.
Brain tumors are basically categorized on the basis of origin, location, area of the tumor and biological characteristics of the tissue.

Various type of brain tumors are:
1- GLIOMAS: Glioma develops from Glial cells which are supporting cells in the brain.
2- METASTASIS: Is the second type of tumors. They spread to another part through the brain tumor blood Stream.
3- ASTROCYTOMA: it is slowly grows, rarely spreads to other parts of the central nervous system (CNS), it is borders are not well defined. At any stage of age, a cystic formation may occur.

Tumor detection consists of several stages. These stages can be schematically shown in the following figure.

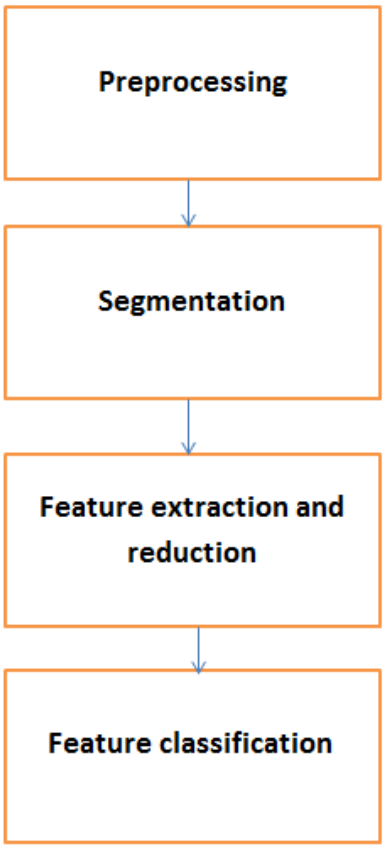


Figure 1.1: Detection stages

Tumor detection stages start with a preprocessing method, then a segmentation method, feature extraction and reduction and finally at the end machine learning algorithm for classification.

1.1. Preprocessing

Pre-processing of image before segmentation is critical for accurate detection of tumor. In this stage, we perform noise and artifacts reduction and sharpening of edges. There are little chances of noise being present in the modern MRI images. Therefore the main task of the preprocessing is to sharpen edges in the image.

1.2. Segmentation

Segmentation is the process of dividing the image into smaller objects. So that objects in the same cluster as similar as possible and objects in different clusters are as dissimilar as possible. Performing segmentation means carrying out the following steps.

The Segmentation steps:

1- Binarization using a thresholding technique.
2- Enhancing the image using morphological transformation.
3- Segmentation.
4- Finally, edge detection.

1.3. Feature extraction

There are two steps should take Feature selection and feature extraction.
It is useful to reduce processing time and complexity, helpful in identify brain tumor where exactly located and prediction next stages.
Feature selection has three types can be selected: texture feature, shape feature, intensity feature.

We use some texture and shape features such as homogeneity, energy, dissimilarity, contrast, correlation, mean, skewness and kurtosis all of these are texture. Shape such as area- Centroid - Compactness.

Feature extraction transforms the data in the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, as in principal component analysis (PCA).

1.4. Feature classification

This classifier is a part of machine learning that gives computers the ability to learn. It is a set of learning methods that analyze data pattern which is used for classification.

We compare several classifiers:

1- Support vector machine.
2- K-nearest neighbors.
3- K-Means.
4- Linear regression. (Linear classifier).

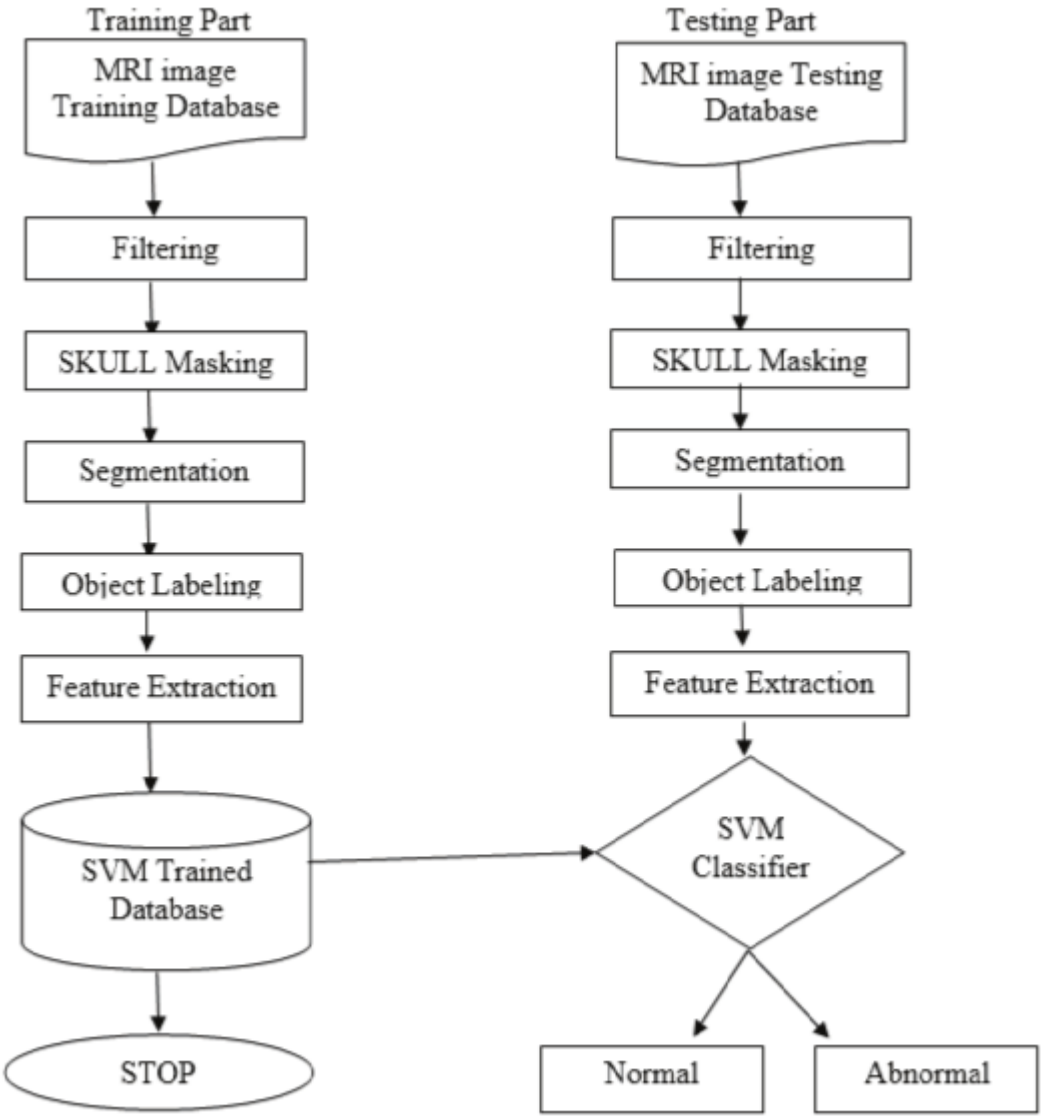


Figure 1.2.: Project stages

Chapter 2 Preprocessing

2.1. Introduction:

There are many imaging modalities like: x-ray, ultrasonography, single-photon emission computed tomography (SPECT) and MRI. The safety one to a brain is MRI so it used a lot and we used it in our project.

2.2. Magnetic resonance imaging (MRI)

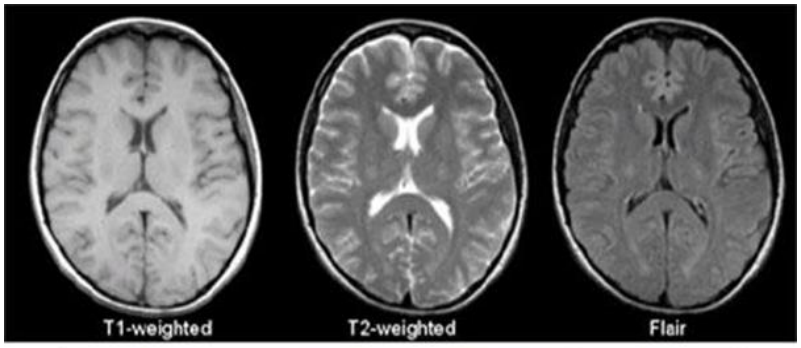
It is a medical imaging technique used in radiology to form pictures of the anatomy and the physiological processes of the body in both health and disease. An MRI image has information on it like name of the patient, doctor, time and date…etc. This image is defined as a three-dimensional function, f(x, y, z), where x, y, and z are spatial coordinates, and the amplitude at any point (x, y, z) is called the intensity level of the image at that point.

2.2.1. Types of MRI:

2.2.1.1. T1W (T1-Weignted): allow an easy annotation of the healthy. Make the brain tumor border become bright.

2.2.1.2. T2W(T2-Weignted): edema region can appear brighter than other

2.2.1.3. FLAIR: this type we used because it separates the edema region from the CSF.



Tissue	T1-Weighted	T2-Weighted	Flair
CSF	Dark	Bright	Dark
White Matter	Light	Dark Gray	Dark Gray
Cortex	Gray	Light Gray	Light Gray
Fat (within bone marrow)	Bright	Light	Light
Inflammation (infection, demyelination)	Dark	Bright	Bright

2.3. First: Convert to gray style:

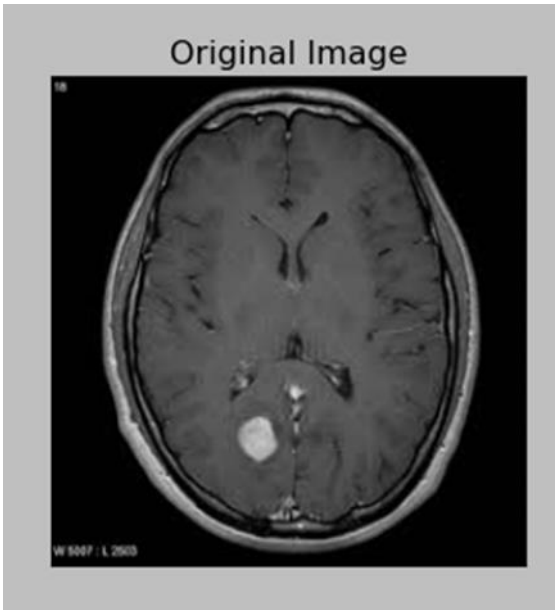
At first, we should remove any colors in the image to convert it to the gray scale by CvtColor function. Converting an image from 3d to 2d makes the processing faster.

Python: cv.CvtColor (**Image_Input**, image_output, code)

And it's written in our code like:

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

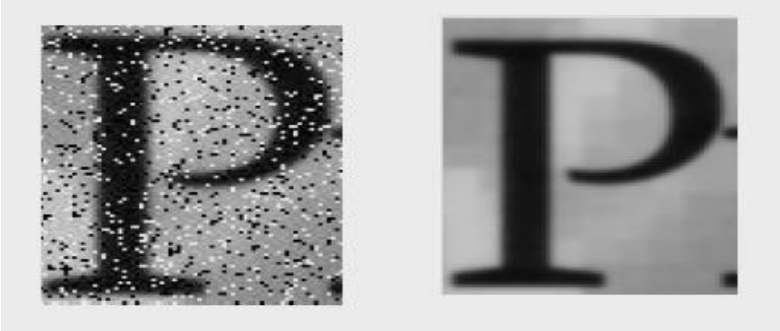
The result like:



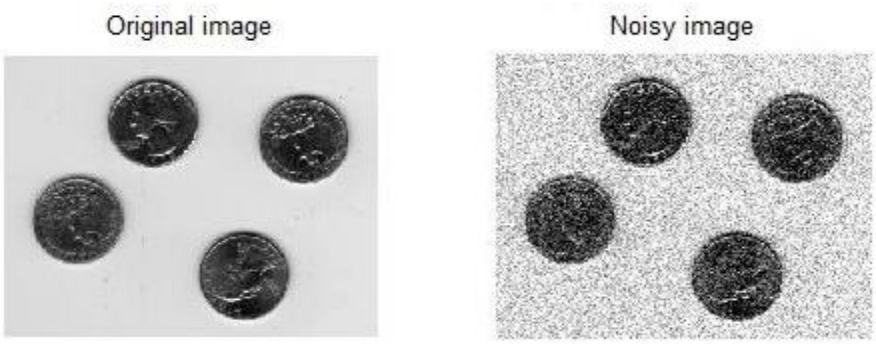
2.4. Types of image noise:

There are two types of noise we encountered in our MRI IMAGE:

1. **Salt and Pepper noise**: it appears like randomly sparse white, black or both pixels over the image.



2. **Gaussian noise:** caused by the random fluctuations in the signal it's modeled by random values add to an image, has a probability density function (pdf) of the normal distribution.

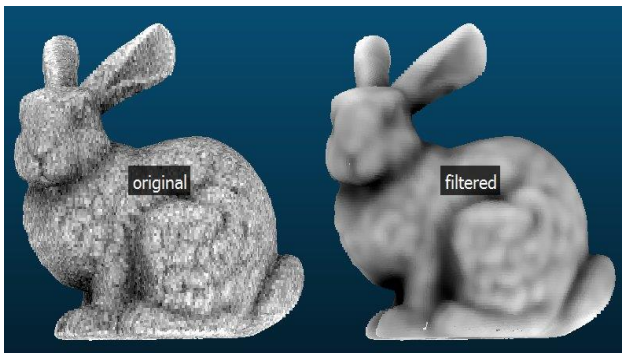


2.5. Second: Filters:

Filters used as a prepossessing to remove the noise from the image
Filters are a heart of an image enhancement **there are two types of filters:**

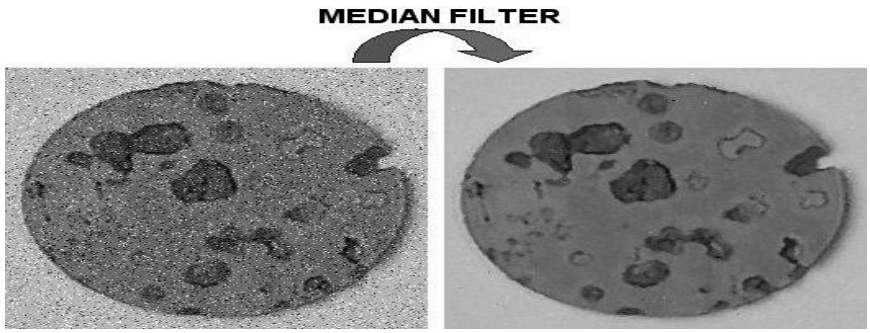
2.5.1. linear filter:

1) The design of this filter is Simple and it tends to blur the sharp edges. Such as:
Gaussian filter: used for smoothing by convolution operation to remove noise and blur. We do this process on all pixels of an image enhancement so it means spatial domain.
The kernel of the Gaussian filter must be odd and positive.



2.5.2. Nonlinear filter:

The design of this filter is difficult but very effective at removing noise. It's more powerful than linear filters because it is able to reduce noise levels without blurring edges. Such as:
Median filter: it used for reducing the intensity variation between one pixel and neighbors pixels. It spatial domain.



Median filter operation:

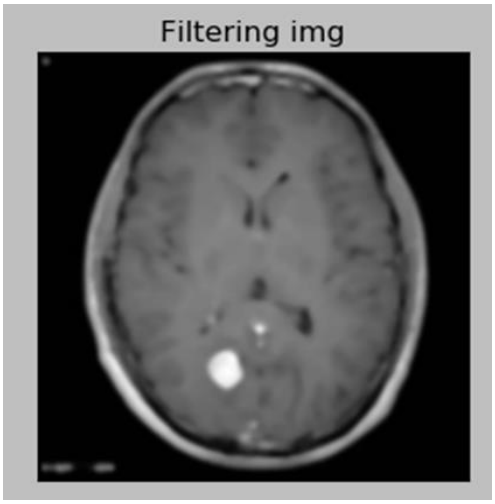
1- Sorting the pixel value of the mask given.
2- Determine the median value of the pixel.
3- Replaces pixel value with the median value of the mask.

It's powerful because it removing salt and pepper noise and impulse noise but it removing image details.

It is in our code like:

```
#Filtering
median = cv2.medianBlur(img,5)
Gaussian = cv2.GaussianBlur(median,(5,5),0)
```

The result like:



2.6. Third: Binarization using thresholding techniques

Threshold has three types, the general type in our code is:

Simple Thresholding: It's the global method, using this function: **cv2.threshold**

- **Python:**cv.**Threshold**(Image_input,image_output, threshold, maxValue, thresholdTye)

2.6.1. Types of thresholds

1) **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

2) **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

3) **THRESH_TRUNC**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

4) **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

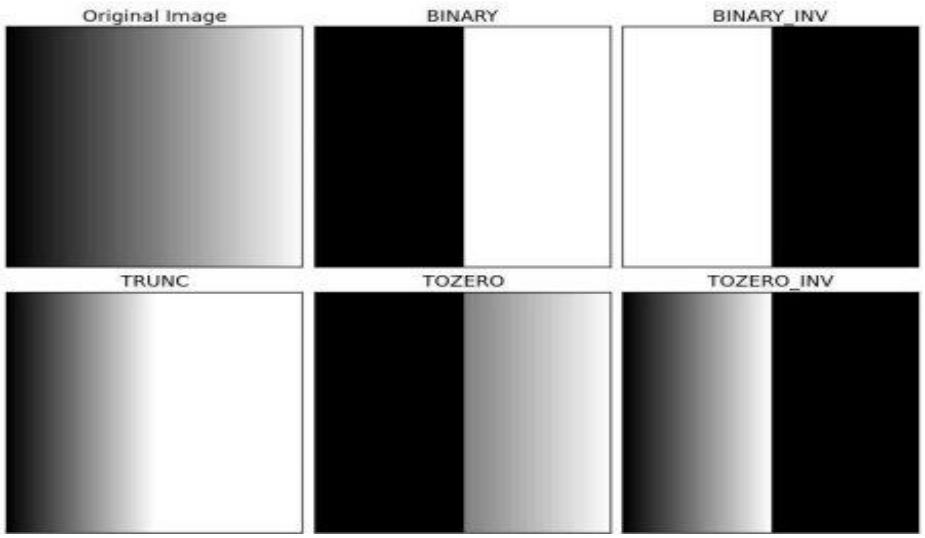
5) **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

These types in the code will be like:

```
img = cv2.imread('gradient.png',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
```

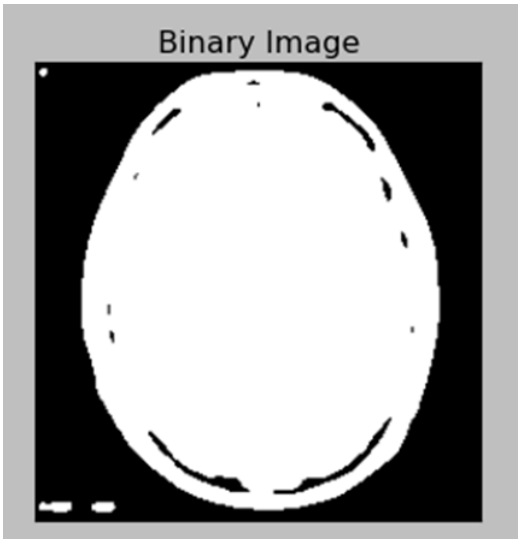
And the given result will appear like:



We used THRESH_BINARY type in our code like:

```
#binarization
ret, thresh1 = cv2.threshold(img,25,255,cv2.THRESH_BINARY)
```

Result like:



2.7. Fourth: Morphological transformations

It removes noise from a brain to make sure that all brain is converted to white, to manage from removing boundary (skull) without effect on other parts of a brain. That is done by morphology transformation. Morphological transformations are some simple operations. It is on binary images. It needs two inputs, one is our original image, and the second one is called structuring element or kernel.

- **Morphological operators**

1) Erosion:

It erodes away the boundaries of foreground object by making a kernel slides through the image (as in 2D convolution). If only all the pixels under the kernel is 1, a pixel in the original image (either 1 or 0) will be considered 1, otherwise, it made equal to zero. It is useful for removing small white cause the simply white region decreases in an image

In example code:

```
img = cv2.imread('j.png',0)
Kernel = np.ones((5,5),np.uint8)
Erosion = cv2.erode(img, Kernel, iterations = 1)
```

Where,

- (5, 5) – Is a size of the kernel
- Iterations – It chooses by testing values to obtain the best thing.

2) Dilation:

It is just the opposite of erosion. So it increases the white region in the image.

In code:

```
Dilation = cv2.dilate(Img, kernel, iterations = 1)
```

3) Opening:

It's **erosion followed by dilation**. Here we use the function: cv2.morphologyEx().

In code:

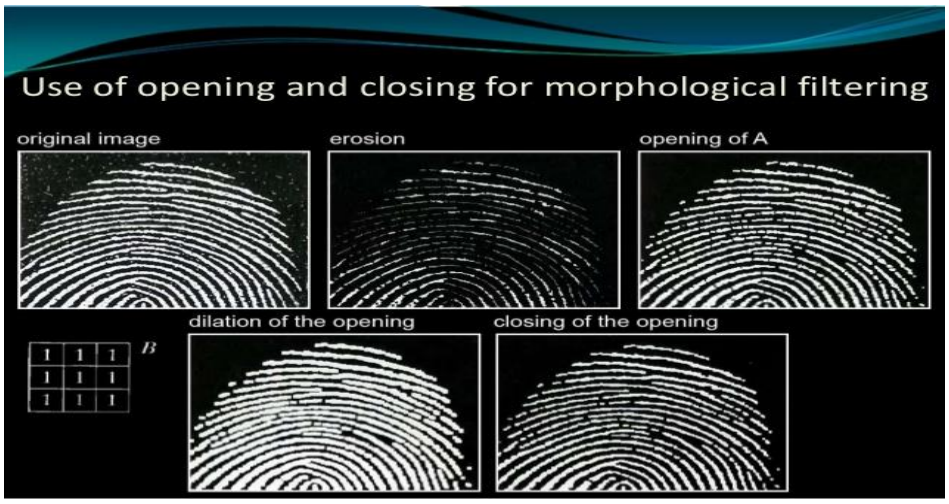
```
Opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

4) Closing:

It is a reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes or small black points on the image.

In code:

```
Closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

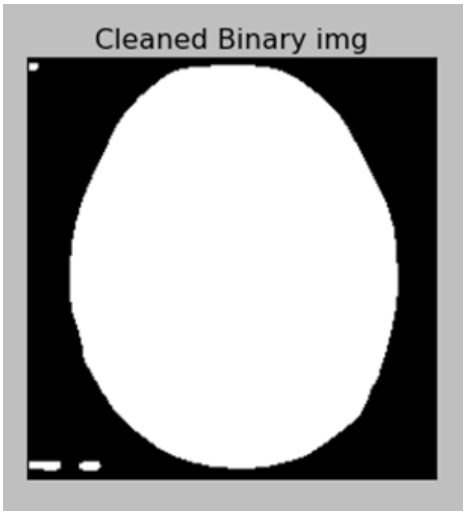


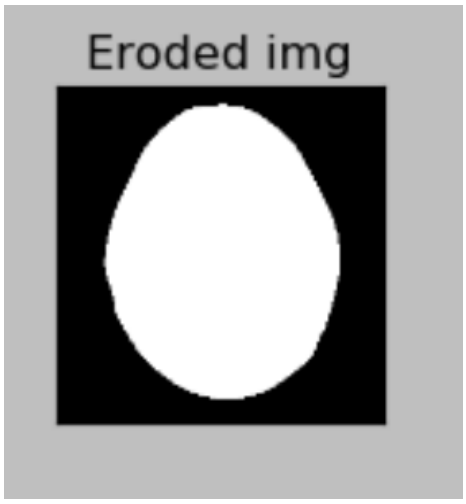
In our code closing transformation used first to remove noise then we use Erosion to remove skull like:

```
#removing skull
erosion = cv2.erode(closing1,kernel,iterations = 6)
NO_skull = img * (-erosion);
```

```
#convert to white
kernel = np.ones((7,7),np.uint8)
closing1 = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)
```

Result like:

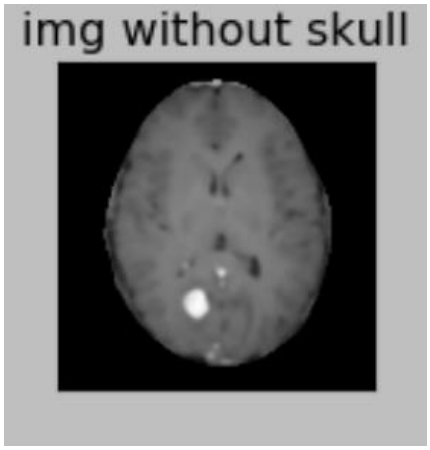




2.8. Fifth: Removing edges

Convert obtained image to the original one by multiply what we obtain with the original after filtering.

Result like:



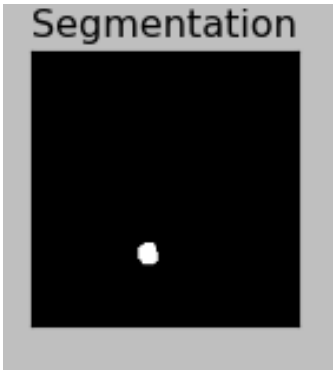
2.9. Sixth: Threshold again

Use threshold again to detecting tumor but change the value of threshold

We make it like:

```
#Segmentation
ret, thresh2 = cv2.threshold(img,120,255,cv2.THRESH_BINARY)
```

Result like:



2.10. Seventh: Edge Detector

We detecting tumor by using Canny Edge Detector which is developed by John F in 1986. Canny Edge detector aims to satisfy three main criteria low error rate, good localization, and minimal response.

1.10.1 steps of Canny Edge Detector:

1.10.1.1 Noise Reduction: First step is to remove the noise in the image with a 5x5 Gaussian filter.

1.10.1.2 Finding Intensity Gradient of the image:

A) Filtering with Sobel kernel in both horizontal (Gx) and vertical direction (Gy):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

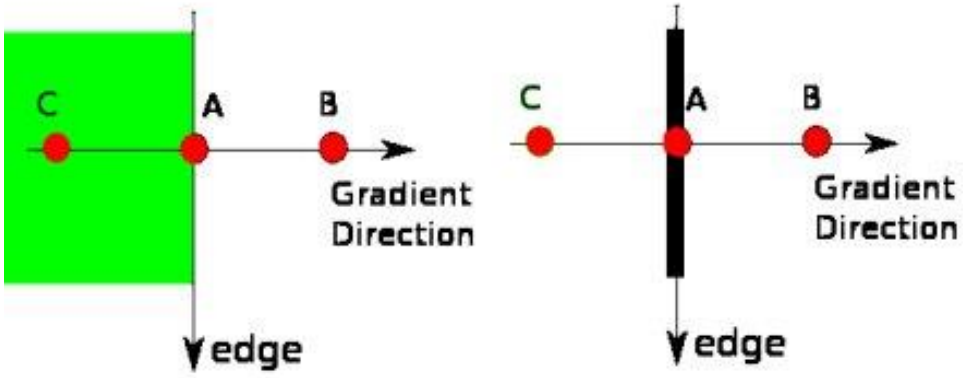
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

B) Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

1.10.1.3 Non-maximum Suppression: a full scan of image is done to remove any unwanted pixels After getting gradient magnitude and direction, every pixel is checked if it is a local maximum in its neighborhood in the direction of gradient

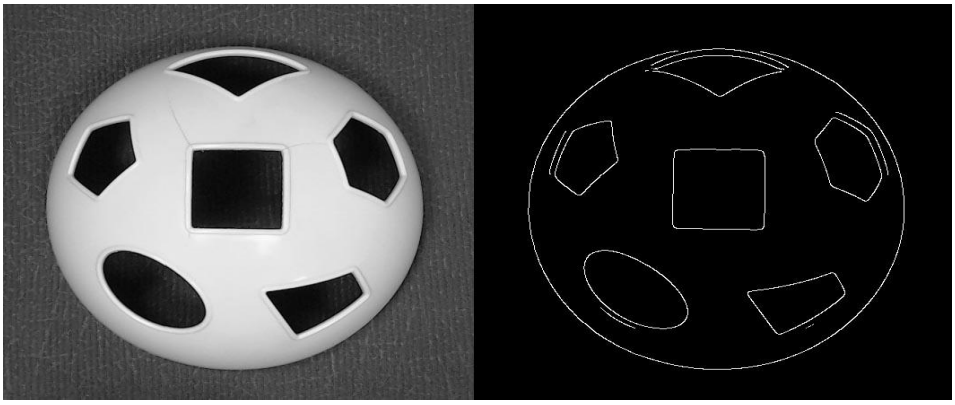


Point A is on the edge (in the vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So to see if it forms a local maximum, point A is checked with point B and C. if so, it is considered for next stage, otherwise, it is suppressed (put to zero).

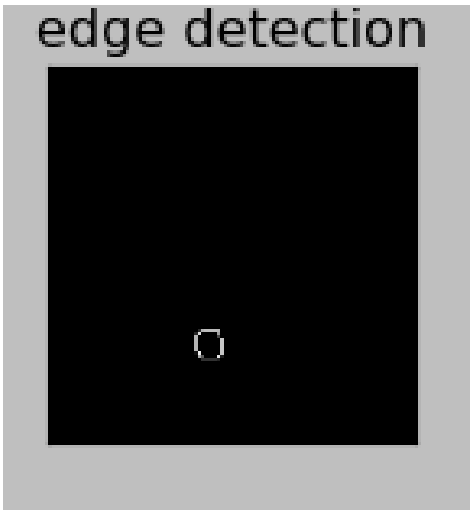
1.10.1.4 Hysteresis Thresholding: Canny does use two thresholds (upper and lower): if a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge. If a pixel gradient value is below the *lower* threshold, then it is rejected. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

In code:

```
#edge detection
edges = cv2.Canny(img,100,200)
```



Result like:



Chapter 3 Feature extraction and reduction

3.1. Introduction

Features are properties that quantify some significant characteristics of the object. After getting an image when ending segmentation, start features steps on an original image. Applying features selection then feature reduction using PCA [1] finally features classification.

Feature extraction the best subset of the existing features contains the least number of dimensions that contribute high accuracy. Feature extraction (reduction) is transforming the input data into the set of feature. Features are given as input to classifier.

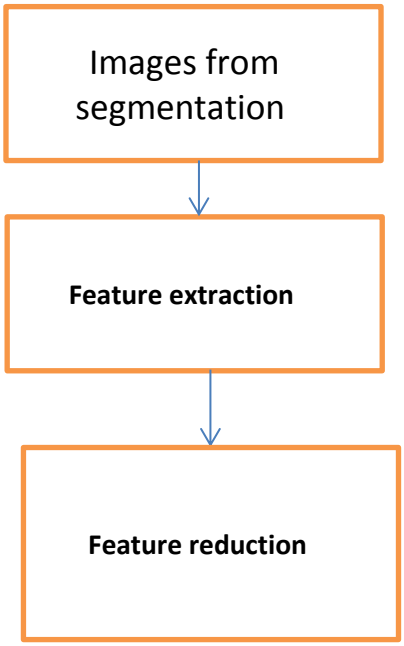
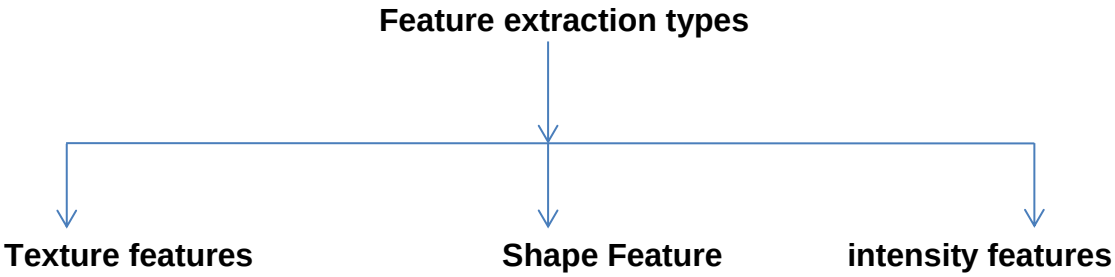


Figure3.1: schematic diagram of features extraction process

The features which extracted should carry enough information about image and should be easy to compute in order to approach the feasibility of a large image collection and rapid retrieval.

3.2. Feature extraction

This stage is important to identifying brain tumor where is exactly located and helps in predicting next stage. There is three kinds of features can be selected and extracted.



- **Texture features:** based on surface properties. Such as homogeneity- energy- means …etc.

- **Shape features**: based on the boundary of an image. Such as: area- Centroid - Compactness …etc.

- **Intensity:** based on region properties. Such as: median- intensity …etc.

Our book shows only features we use in the project.

3.2.1. Texture features

It is characterized by the spatial distribution of gray levels in a neighborhood. Since texture shows its characteristics both by pixel co-ordinates and pixel values. Image texture depends on the scale or resolution at which it's displayed.

It is divided to two types:

Texture based on gray level co-occurrence matrix GLCM

Concerned with properties of two pixels at specific relative positions, the matrix describe how many times a pair of pixels appears in an image, the first pixel is referenced the other is a neighbor, the goal is to assign an unknown sample image to one of a set of known texture classes.

How to calculate GLCM matrix

1. Create framework matrix, if the original image has G Gray levels, then the dimension of the co-occurrence matrix is GxG.
2. Describe spatial relation between reference and neighbor pixels using two parameters (d, Θ), which d is the distance between the pair of pixels and Θ is the angle between them.

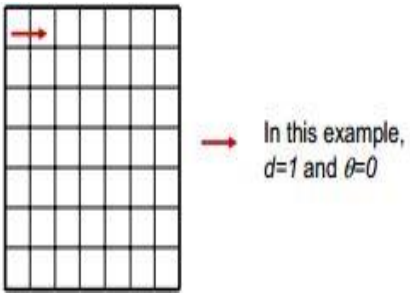


Figure3.2: Spatial relation (d, Θ)

3. Count the occurrence and fill in framework matrix by checking all pixel pairs with distance d and direction Θ assign value i to pixel 1 and value j to pixel 2.

0	1	2	1	3
0	0	2	2	3
0	1	2	2	3
2	2	3	3	2
1	2	3	2	2

First element in matrix = c (i, j) = c (0, 0)

0	1	2	1	3
0	0	2	2	3
0	1	2	2	3
2	2	3	3	2
1	2	3	2	2

Second element in matrix = c (i, j) = c (0, 1)

0 →	1	2	1	3
0	0	2	2	3
0 →	1	2	2	3
2	2	3	3	2
1	2	3	2	2

Continue to calculate we get our matrix

1	2	1	0
0	0	3	1
0	1	4	4
0	0	2	1

GLCM matrix should be symmetric, so we add the matrix to its transpose

2	2	1	0
2	0	4	1
1	4	8	6
0	1	6	2

Figure3.3: Symmetric matrix

Turn matrix to probabilities by normalization the Last row in original Image has no neighbors to the right, so the number of pixel pairs that we can compute is:

$$N_Y \times (N_x - 1) = 5x\ (5-1) = 20$$

Where

N_Y number of columns, N_x number of rows

To get our co-occurrence matrix, divide every pixel by number of pixel pairs

(I,j)	0	1	2	3
0	2/20	2/20	1/20	0
1	2/20	0	4/20	1/20
2	1/20	4/20	8/20	6/20
3	0	1/20	6/20	2/20

Figure3.4: Co-occurrence matrix

Coded like this:

```
# GLCM Texture Features
glcm = greycomatrix(im, [5], [0], symmetric=True, normed=True)
```

Use function greycomatrix from skimage.feature library.

Parameters:

Distance d = [5] , angle Θ = [0]

The matrix should be symmetric and normed like discussed before so assigned true to parameters (symmetric, normed).

Result:

Co-occurrence matrix saved in GLCM variable.

Contrast

Measure the quantity of local changes in an image.

It returns the measure of intensity contrast between the pixel and its neighbor.

Rang = [0 (size (GLCM, 1)-1) ^2], If it is small, texture becomes strong.

Correlation

How correlated pixel is to its neighbor.it is the measure of gray tone linear changes in the image.

Rang = [-1 1].

-1 indicating prefect negative correlation.

1 indicating prefect positive correlation .

Homogeneity (inverse different moment)

It scaled the local changes of image texture.it returns value that measures the closeness of the distribution of elements in the GLCM.

Rang = [0 1].

It became large if local texture only has minimal changes.

Energy

It measures the degree of pixel pair repetitions. Only similarity gray level pixel is present.

Rang = [0 1].

Dissimilarity

Measure of it allows comparison between segmentations created by different algorithms.

Texture based on histogram

Concerned with properties of one pixel.it is important to calculate statistical value such as: mean, skew ….Etc.

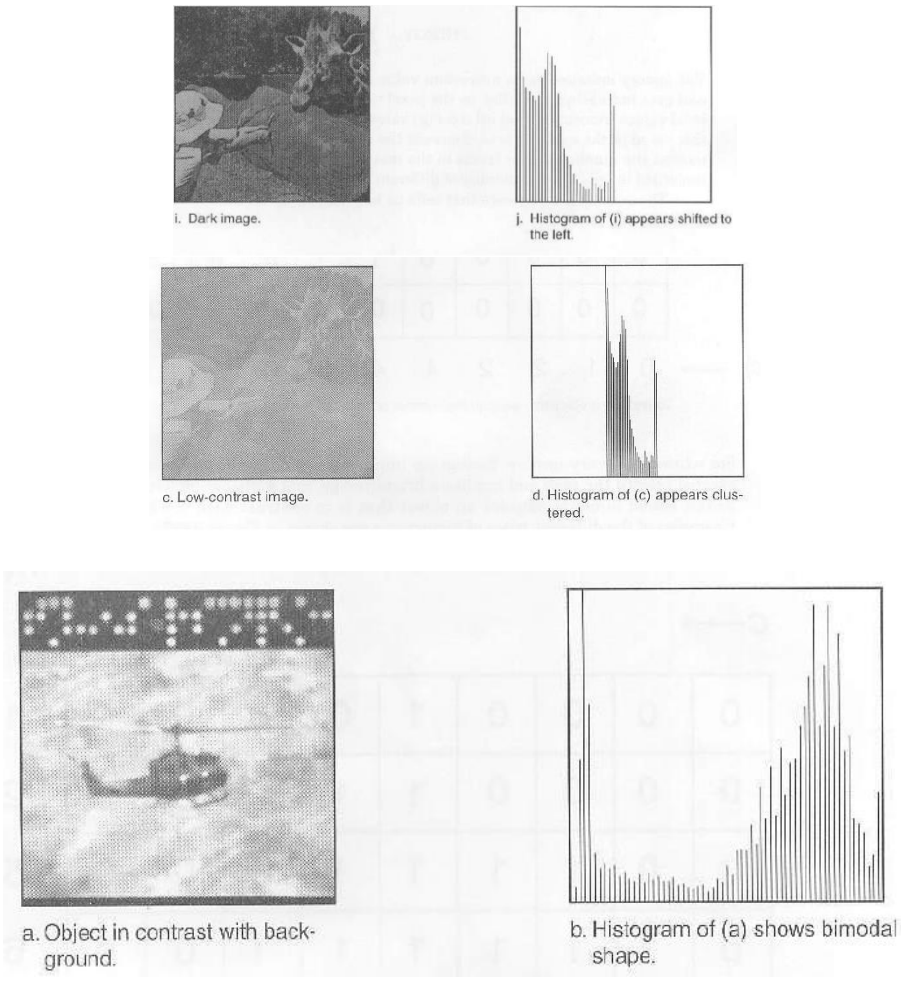


Figure3.5: different histogram for different contrast image

Mean

It is the average value, so it tells us something about the general brightness.

-a bright image has a high mean.

-a dark image has a low mean.

Skew

Measure the asymmetry. The image should have low skew (one peak at each side of the mean).

Kurtosis

How uniform is the gray level distribution

Standard deviation (SD)

Known as the square root of variance describes contrast. High contrast should have high standard deviation

3.2.2. Shape features

Features describe the boundary of the image like area and centroid, help to extract specific information about the shape of the region.

To calculate shape features we should know image moments, An Image moment is a number calculated using a certain formula, asset of moments computed from the digital image represented characteristics of image shape and give information about the geometric feature of the image.

In two-dimensional intensity distribution, the geometric function of pixels values with respect to spatial locations in the image can extract shape information such as image area and coordinates centroid.

The first order moment will give the center of mass, where the mass of a pixel is its intensity; second order moment will give how this mass varies around the center of mass

Area

Calculate a number of pixels in the region. For Calculating area, we need to calculate its zeroes moment

In a binary image, there are two possibilities 0 or 1, so in white pixel add 1 to moment to get all area.

.

Centroid

The coordinates for the region center of mass.

$$centroid = \left(\frac{\mu_{1,0}}{\mu_{0,0}}, \frac{\mu_{0,1}}{\mu_{0,0}}\right)$$

Compactness

Calculated from dividing the area by perimeter square, where perimeter is the length of the boundary of the object, when region more likely to be a circle it goes to maximum compactness

Eccentricity

The ratio between the major axis and the minor axis in the region detected, when the ratio is 1, the region becomes a circle.

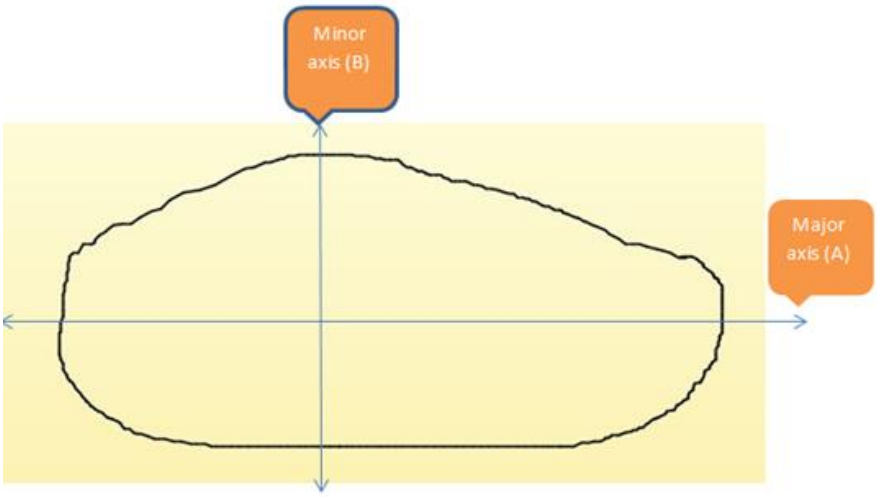


Figure3.6

Major axis (A): longest horizontal axis in region.
Minor axis (B): longest vertical axis in region.

Eccentricity = $\frac{\text{length of A}}{\text{length of B}}$

Euler number

Describe relation between holes and connected region in our region.
Let's say our image is all black color represented by variable S and have some holes represented by variable N.
So our Euler number become

$$U = S - N$$

This feature called as a logical invariant descriptor because its value doesn't change if image zoomed or oriented but also shouldn't increase the degree of zoom till holes disappear or become unrecognizing.

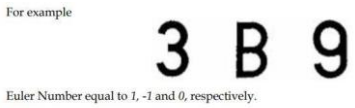


Figure3.7

Convex area

To know convex hull we should first define a polygon.

Polygon is a plane figure with at least three straight sides and angles.

Convex polygon is defined as a polygon with all its interior angels less than 180.This means that all the vertices of the polygon will point outwards, away from the interior of the shape.

Convex hull of set of point Q is the smallest convex polygon P for which each point in set Q is ether on the boundary of P or in its interior.
Let's see for example two regions S1, S2

Figure3.8

In case S1 region is convex.

In case s2 region in not convex.

Figure3.9 Convex hell

So **Convex area** Defined as convex hull area used to calculate solidity feature of image

Solidity

Defined as the ratio of area over convex area

Orientation

The angle between the X-axis and the major axis of the ellipse that has the same second-moments as the region. Ranging from -pi/2 to pi/2 in counter-clockwise direction.

Extent

The proportion of the pixels in the bounding box that are also in the region. Computed as the area divided by the area of bounding box, where bounding box is the area of smallest rectangular that fit around region.

Code:

Import libraries and classes used

```
from preprocessing import Preprocessing
# from segmentaion import Segmentation
from matplotlib import pyplot as plt
from skimage import io
%matplotlib inline
import numpy as np
import pandas as pd
from skimage.feature import greycomatrix, greycoprops
from skimage import data
from skimage.color import rgb2gray
from scipy.stats import kurtosis ,skew
import skimage
from skimage import measure
from skimage.measure import label
from skimage.measure import regionprops
from skimage import io
from skimage.io import imread
```

Figure3.10

Define arrays to every feature to save results in

```
#texture array
ds = []
cr = []
cn = []
am = []
en = []
ho = []
Class = []
av_kur1 = []
av_sk1 = []
std = []
Mean = []
# shape array
centroid =[]
area =[]
extent = []
orientation = []
convex_area = []
solidity = []
eccentricity =  []
euler_number = []
```

Figure3.11

Calculate some variables need to extract shape features

1. Columns center and rows center
2. Moment central
3. Label image
4. Properties (props)

Label image

Determine connected region in detected region and assigned label number to it, the first region take label number 0, and second region take label number 1, and so on, and all connected regions are assigned the same integer value.

Class label used to determine that region, its parameter

- Our segmented image
- Connectivity
  A maximum number of hops to consider a pixel as a neighbor, it accepted value from 1 to im.nidm.

Properties

Our feature define as properties in class regionprops, it contain every shape feature, just call it like this regionprops.feature_needed, its parameter label image which calculate in the previous step .

```
# Shape feature
m = skimage.measure.moments(im)
cr = m[0, 1] / m[0, 0]
cc = m[1, 0] / m[0, 0]
measure.moments_central(im, cr, cc)
label_img = label(im, connectivity=im.ndim)
props = regionprops(label_img)
```

Figure3.12

Get feature values and append it to their arrays, and append class value if normal 0 and if abnormal 1

```
area.append(props[0].area)
extent.append(props[0].extent)
orientation.append(props[0].orientation)
convex_area.append( props[0].convex_area)
solidity.append(props[0].solidity)
eccentricity.append(props[0].eccentricity)
euler_number.append(props[0].euler_number)
Class.append(0)
```

Figure3.13

Appends feature into the array which out from calculating the statistics specified in properties from the gray-level co-occurrence matrix GLCM.

```
# # GLCM Texture Features
glcm = greycomatrix(im, [5], [0], symmetric=True, normed=True)
 ds.append(greycoprops(glcm, 'dissimilarity')[0,0])
 crr.append(greycoprops(glcm, 'correlation')[0,0])
 cn.append(greycoprops(glcm, 'contrast')[0,0])
 am.append(greycoprops(glcm, 'ASM')[0,0])
 en.append(greycoprops(glcm, 'energy')[0,0])
 ho.append(greycoprops(glcm, 'homogeneity')[0,0])
```

Figure3.14

Create frame and assign values of shape and texture feature to it, and save results to file named feature_all.csv.

```
dfr = pd.DataFrame({'area' :area,
                                    'extent' : extent,
                                    'orientation' :orientation ,
                                    'convex_area':convex_area,
                                    'solidity':solidity,
                                    'eccentricity':eccentricity,
                                    'euler_number':euler_number,
                                    'dissimilarity' :ds,
                                    'correlation' : crr,
                                    'contrast' :cn ,
                                    'ASM':am,
                                    'energy':en,
                                    'homogeneity':ho,
                                    'kurtosis':av_kur1,
                                    "skew":av_sk1,
                                    "Standard deviation":std,
                                    "Mean":Mean,
                                     'class':Class})
dfr.to_csv('feature_all.csv')
```

Figure3.15

Result of our shape and texture feature code

	area	class	convex_area	eccentricity	euler_number	extent	orientation	solidity	ASM	Mean	Standard deviation	class.1	contrast	correlation
0	639	1	1134	0.999982	1	0.394444	0.000368	0.563492	-1.968855	1.973530	0.848617	1.0	3.942310	-2.001362
1	601	1	1857	0.999112	1	0.226621	-0.028372	0.323640	1.087424	-1.069509	-1.204707	1.0	-0.266962	-0.808870
2	40006	1	63361	0.831824	1	0.517596	-1.435621	0.631398	0.292740	-0.491327	-0.108214	1.0	-0.460868	0.501245
3	3146	1	23052	0.913541	1	0.071810	-0.335735	0.136474	-0.725334	0.034367	0.490886	1.0	0.809064	-0.181822
4	211	1	375	0.999788	1	0.394393	0.003083	0.562667	1.105803	-1.079292	-1.229847	1.0	-0.124837	-1.227367
5	563	1	991	0.999977	1	0.397880	0.000436	0.568113	-0.558792	0.248113	0.664908	1.0	0.046791	0.484695
6	220	1	400	0.999810	1	0.382609	0.002933	0.550000	-0.328013	0.391019	0.761596	1.0	-0.688322	1.028482
7	1	1	1	0.000000	1	1.000000	0.785398	1.000000	-0.495760	0.322026	0.716347	1.0	-0.072288	0.583063
8	1523	1	14957	0.986649	1	0.056850	0.019457	0.101825	-0.840771	0.944285	0.984818	1.0	0.106637	0.589299
9	20600	1	35529	0.885557	1	0.369467	-1.365782	0.579808	0.861069	-0.888501	-0.793070	1.0	-0.482114	0.028953
10	1	1	1	0.000000	-19	1.000000	0.785398	1.000000	-1.401444	1.761947	0.937667	1.0	1.640881	-0.413793
11	1	1	1	0.000000	-19	1.000000	0.785398	1.000000	0.891853	-1.025682	-1.099513	1.0	-0.153366	-0.827737
12	651	1	3286	0.998600	1	0.123694	0.001357	0.198113	0.700916	-0.984022	-1.003329	1.0	0.118194	-1.101837
13	227	1	404	0.999821	1	0.391379	0.002895	0.561881	1.794935	-1.326463	-1.963049	1.0	-0.446367	-3.322867
14	1123	1	3958	0.957626	1	0.197225	-0.121029	0.283729	1.091690	-1.016645	-1.076069	1.0	-0.630068	-0.046289
15	16	1	32	0.933488	1	0.355556	0.294694	0.500000	0.107993	-0.538622	-0.179239	1.0	0.140699	-0.149824
16	180	1	326	0.999695	1	0.387097	0.004373	0.552147	-0.567754	0.887011	0.972809	1.0	-0.684258	1.082837
17	323	1	727	0.999914	1	0.398765	0.000870	0.444292	0.962756	-0.824398	-0.662934	1.0	-1.002608	0.814808
31	350	0	1292	0.998659	1	0.189394	-0.024836	0.270898	0.566234	-0.638193	-0.485026	0.0	-0.275159	-0.125596
32	49284	0	76806	0.697463	1	0.447434	-0.676745	0.641669	0.778545	-0.805292	-0.798597	0.0	-0.003674	-1.212161
33	419	0	3771	0.991560	1	0.062743	-0.065399	0.111111	-0.313949	-0.147348	0.198588	0.0	1.313655	-0.347720
34	2447	0	8088	0.991300	1	0.188231	-0.081710	0.302547	-0.930934	0.475942	0.756970	0.0	1.389692	0.241287
35	333	0	968	0.999320	1	0.213462	0.017713	0.344008	-1.033036	1.274983	1.145691	0.0	-0.078056	1.130631
36	4098	0	120326	0.884608	1	0.022718	-0.878436	0.034057	1.000000	1.000000	1.000000	1.0	1.000000	1.000000
37	3315	0	9405	0.774829	1	0.225648	0.172261	0.352472	1.000000	1.000000	1.000000	1.0	1.000000	1.000000
38	1047	0	8326	0.995388	1	0.071278	-0.006280	0.125751	1.000000	1.000000	1.000000	1.0	1.000000	1.000000

3.3. Feature Reduction

In machine learning and statistics, **dimensionality reduction** or **dimension reduction** is the process of reducing the number of random variables under consideration, via obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

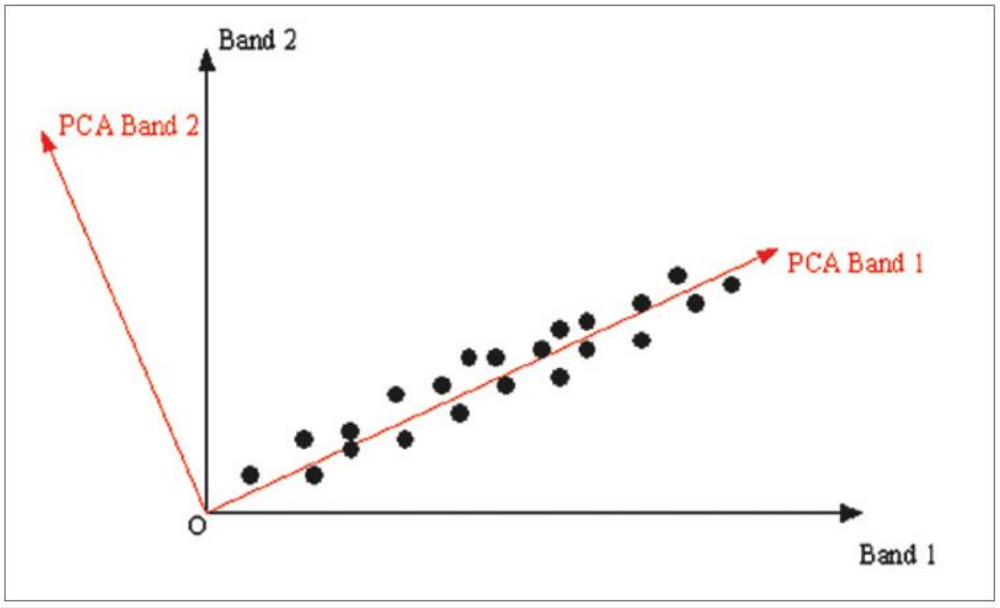
We talk about feature extraction in previous chapter, now we will explain feature extraction o reduction.

Feature extraction transforms the data in the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, as in principal component analysis (PCA), but many nonlinear dimensionality reduction techniques also exist. For multidimensional data, tensor representation can be used in dimensionality reduction through multilinker subspace learning.

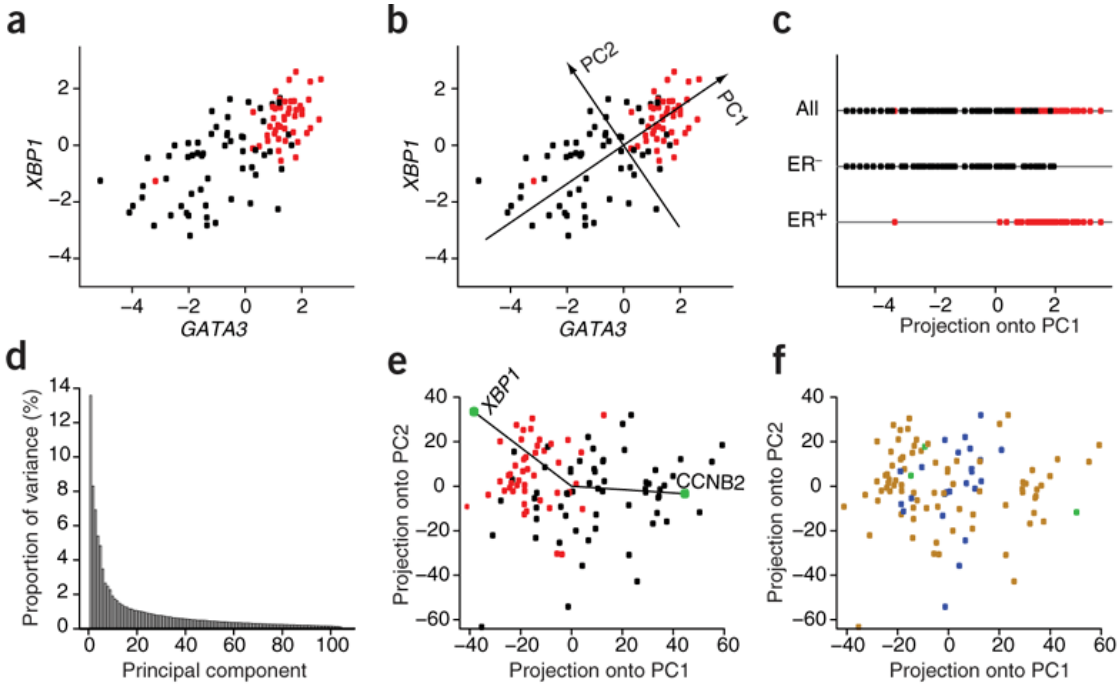
Methods of feature reduction

- **Principal Component Analysis (PCA):**

  Principal Component Analysis (PCA) is a statistical procedure that orthogonally transforms the original *n* coordinates of a data set into a new set of *n* coordinates called principal components. As a result of the transformation, the first principal component has the largest possible variance; each succeeding component has the highest possible variance under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. Keeping only the first $m < n$ components reduces the data dimensionality while retaining most of the data information, i.e. the variation in the data. Notice that the PCA transformation is sensitive to the relative scaling of the original variables. Data column ranges need to be normalized before applying PCA. Also notice that the new coordinates (PCs) are not real system-produced variables anymore. Applying PCA to your data set loses its interpretability. If interpretability of the results is important for your analysis, PCA is not the transformation for our project.



(a)



(b)

Figure3.16 image a and b are Examples of PCA on the right is the original data set, and on the left is the Transformed data after applying PCA.

- **Missing Values Ratio.**

  Data columns with too many missing values are unlikely to carry much useful information. Thus data columns with number of missing values greater than a given threshold can be removed. The higher the threshold, the more aggressive the reduction.

- **Low Variance Filter:**

  Similarly to the previous technique, data columns with little changes in the data carry little information. Thus all data columns with variance lower than a given threshold are removed. A word of caution: variance is range dependent; therefore normalization is required before applying this technique.

- **High Correlation Filter:**

  Data columns with very similar trends are also likely to carry very similar information. In this case, only one of them will suffice to feed the machine learning model. Here we calculate the correlation coefficient between numerical columns and between nominal columns as the Pearson's Product Moment Coefficient and the Pearson's chi square value respectively. Pairs of columns with correlation coefficient higher than a threshold are reduced to only

one. A word of caution: correlation is scale sensitive; therefore column normalization is required for a meaningful correlation comparison.

- **Random Forests / Ensemble Trees:**

Decision Tree Ensembles, also referred to as random forests, are useful for feature selection in addition to being effective classifiers. One approach to dimensionality reduction is to generate a large and carefully constructed set of trees against a target attribute and then use each attribute's usage statistics to find the most informative subset of features. Specifically, we can generate a large set (2000) of very shallow trees (2 levels), with each tree being trained on a small fraction (3) of the total number of attributes. If an attribute is often selected as best split, it is most likely an informative feature to retain. A score calculated on the attribute usage statistics in the random forest tells us – relative to the other attributes – which are the most predictive attributes. Ex im 1.2

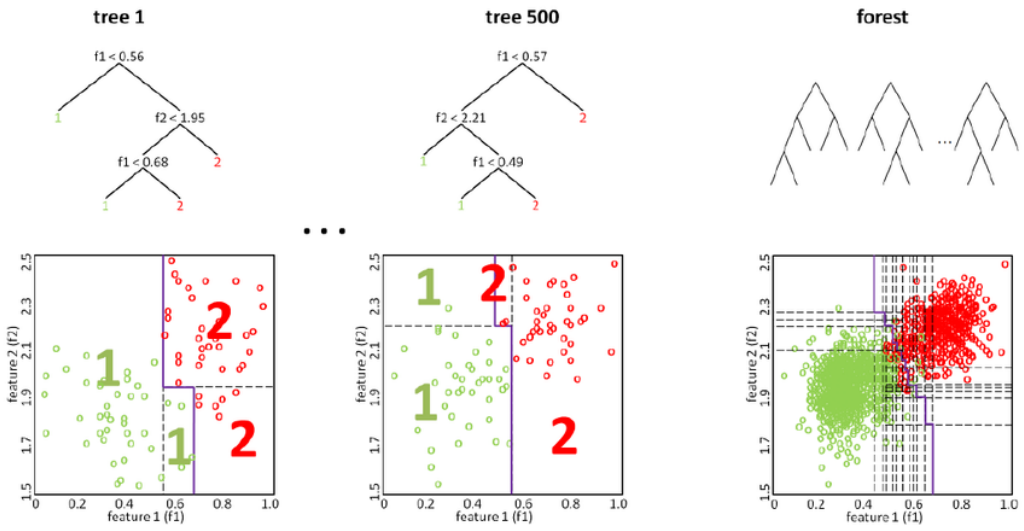


Figure3.16 Random Forests

- **Backward Feature Elimination:**

In this technique, at a given iteration, the selected classification algorithm is trained on *n* input features. Then we remove one input feature at a time and train the same model on *n-1* input features *n* times. The input feature whose removal has produced the smallest increase in the error rate is removed, leaving us with *n-1* input features. The classification is then repeated using *n-2* features, and so on. Each iteration *k* produces a model trained on *n-k* features

and an error rate *e (k).* Selecting the maximum tolerable error rate, we define the smallest number of features necessary to reach that classification performance with the selected machine learning algorithm.

- **Forward Feature Construction:**

This is the inverse process to the Backward Feature Elimination. We start with 1 feature only, progressively adding 1 feature at a time, i.e. the feature that produces the highest increase in performance. Both algorithms, Backward Feature Elimination and Forward Feature Construction, are quite time and computationally expensive. They are practically only applicable to a data set with an already relatively low number of input columns.

- **Feature Reduction in this project:**

We use Feature selection and extract 17 Feature from our Tumor image array

Figure3.17 Brain Tumor

These Features are:

1 -Texture Feature:

ASM - Mean - Standard deviation- contrast -correlation - dissimilarity - energy- homogeneity - kurtosis - skew

2 -Shape Feature:

Area - convex area - Eccentricity - Solidity - Euler number - Extent - Orientation

A - First we use Low Variance Filter to remove features that have low variance data samples

These Features are:

1 - Solidity

2 - Euler number

3- Extent

This is output of these features:

	solidity	euler_number	extent
0	1.000000	1	1.000000
1	1.000000	1	1.000000
2	1.000000	1	1.000000
3	1.000000	1	1.000000
4	1.000000	1	1.000000
5	1.000000	1	1.000000
6	1.000000	1	1.000000
7	0.833333	1	0.555556
8	1.000000	1	1.000000
9	1.000000	1	1.000000
10	1.000000	1	1.000000
11	1.000000	1	1.000000
12	1.000000	1	1.000000
13	1.000000	1	1.000000
14	1.000000	1	1.000000
15	1.000000	1	1.000000

B - Then we normalize the rest of features to make feature values Convergent as possible using Standard deviation and Mean

And this done by compute mean of a group of one feature and subtract it from each sample in this feature group:

Feaure_group = Feaure_group / std(feature_group)

```
ds_class_1.loc[0:,'Mean'] = (ds_class_1['Mean']) / ds_class_1['Mean'].std()
ds_class_1.loc[0:,'ASM'] = (ds_class_1['ASM']  / ds_class_1['ASM'].std())
ds_class_1.loc[0:,'contrast'] = (ds_class_1['contrast']) / ds_class_1['contrast'].std()
```

This is an ex. In our code doing normalization

This our data before and after we done normalization:

ASM	Mean	Standard deviation	area	class	contrast	convex_area	correlation	dissimilarity	eccentricity	energy	euler_number	extent	homogeneity	k
0.285761	68.329483	80.143673	1	1	649.131029	1	0.949574	6.389174	0.000000	0.534566	1	1.0	0.644000	-1.
0.705255	20.817018	53.099576	4	1	513.964730	1	0.909389	4.205555	0.960659	0.839795	1	1.0	0.868637	18.
0.199130	65.610310	69.660937	2	1	1148.323650	1	0.881897	10.931032	1.000000	0.446240	1	1.0	0.557320	1.
0.485445	32.539409	59.810339	1	1	1172.741442	1	0.836718	10.137509	0.000000	0.696739	1	1.0	0.742990	6.
0.156314	71.760435	68.979068	1	1	1211.991816	1	0.872515	11.240619	0.000000	0.395366	1	1.0	0.528619	7.

Figure3.18, before Normalization

ASM	Mean	Standard deviation	area	class	contrast	convex_area	correlation	dissimilarity	eccentricity	energy	homogeneity	kurtosis	orientation	
1.092141	2.624224	3.840221	0.970061	1	0.939040	0.970061	9.055433	1.083513	0.000000	2.669151	3.858496	-0.124202	1.157596	0.
2.695391	0.799487	2.544357	3.880245	1	0.743507	0.970061	8.672212	0.713202	1.994929	4.193190	5.204400	2.290448	-1.021004	2.
0.761049	2.519793	3.337922	1.940122	1	1.661177	0.970061	8.410037	1.853747	2.076624	2.228128	3.339160	0.177210	1.157596	0.
1.855307	1.249690	2.865914	0.970061	1	1.696500	0.970061	7.979204	1.719177	0.000000	3.478897	4.451593	0.800922	1.157596	1.
0.597412	2.755991	3.305250	0.970061	1	1.753280	0.970061	8.320575	1.906249	0.000000	1.974107	3.167200	0.862657	1.157596	-0.

Figure3.18, after Normalization

C - Finally we use PCA and linear_model(Lasso) to do Linear dimensionality reduction :

1- Lasso Model

First we divide data into two categories:

- Feature (feature that we need to reduce it)
- Labels (is the class of each set of features)

In this code, Lasso algorithm fit feature and labels

Then return zero if feature has no impact in training and number if it has impact.

```
df = pd.read_csv("Feature_all_optomized.csv", index_col=0)
feature = df.drop(['class'],1)
labels =df['class']
regression = Lasso(alpha=0.1)
regression.fit(feature, labels)
regression.predict([[0.62169290804739175,0.92176444484776776, 1]])
print(regression.coef_)
print(regression.intercept_)
```

And this is the output:

```
[-0.       -0.        0.       -0.       -0.       -0.06734644
 -0.04956929 -0.        0.       -0.       -0.09523231  0.        0.
  0.       ]
```

Lasso Algorithm select only 3 Features, homogeneity, correlation, convex_area

2- PCA

PCA take one important parameter that is the core of this algorithm, it is number of component we need this algorithm to return or in other meaning number of dimensions we need it to reduce about.

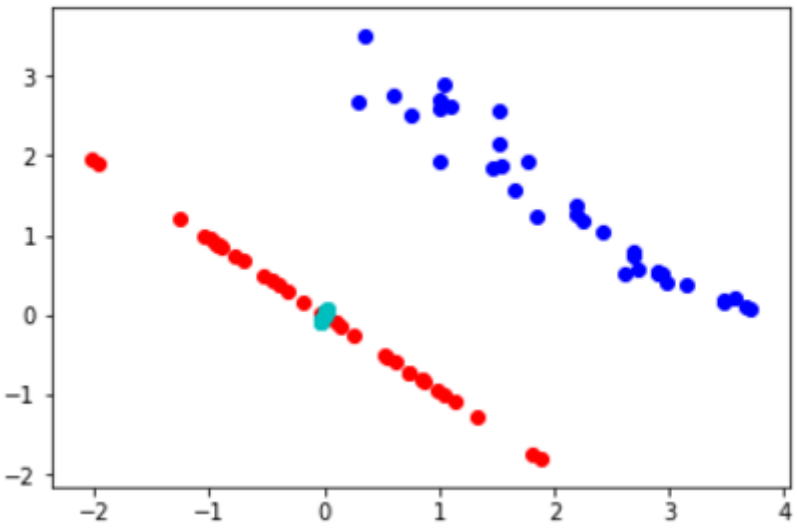
It takes all features and reduces them linearly in a specific number of component (dimensions).

Example from our code:

```
# PCA
from sklearn.decomposition import PCA , IncrementalPCA

def do_pca(data):
    pca = PCA(n_components=2)
    pca.fit(data)
    return pca

data = np.array(feature)
pca = do_pca(data)
print(pca.explained_variance_ratio_)
first_pc = pca.components_[0]
second_pc = pca.components_[1]
```

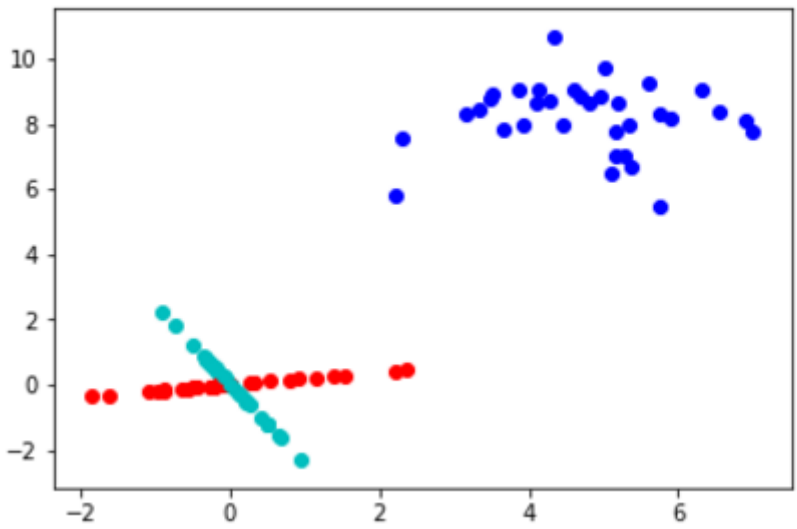
And this is the output for 14 input features:

[0.49518394 0.17667961]



This is the output from only 3 features the selected by Lasso algorithm:

[0.45313957 0.30782284]



The output from these two algorithms proves that only three features will affect or have the best impact in our classification problem.

Chapter 4 Machine Learning

4.1. Why Classification and Classification Problem?

In machine learning and statistics, **classification** is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

An algorithm that implements classification, especially in a concrete implementation, is known as a **classifier**. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm that maps input data to a category.

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as *instances*, the explanatory variables are termed *features* (grouped into a feature vector), and the possible categories to be predicted are *classes*. Other fields may use different terminology: e.g. in community ecology, the term "classification" normally refers to cluster analysis, i.e. a type of unsupervised learning, rather than the supervised learning described in this article.

4.2. Classification algorithms used in this project:

4.2.1. Supervised learning:

The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.

4.2.1.1. *Support vector machine (SVM):* is an algorithm of supervised machine, is used for both classification and regression.

- **Description of algorithm**

It distributes data items as points in n dimensional space where n is number of features, the data points is called the support vectors .Then the hyper plane is detected to separate points into two classes to made classification very well as fig (4-1).
The hyper plane is line used for splitting input data distribute in n dimensional space.

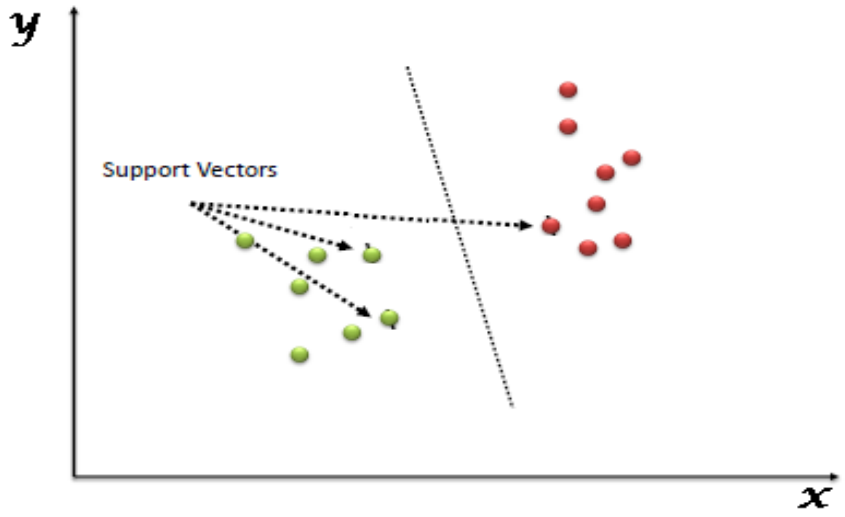


Fig4-1(SVM description)

- **How to select the right hyper planes?**

There are many scenarios to select the best hyper plane.

Scenario (a): assume there are two classes (star and circle)
The input data are distributed as two classes and there is three hyper planes detected as a figure (4-2). The right hyper plane is the line (C) because the hyper plane (c) is high margin compare to a hyper plane (a) and hyper plane (b).

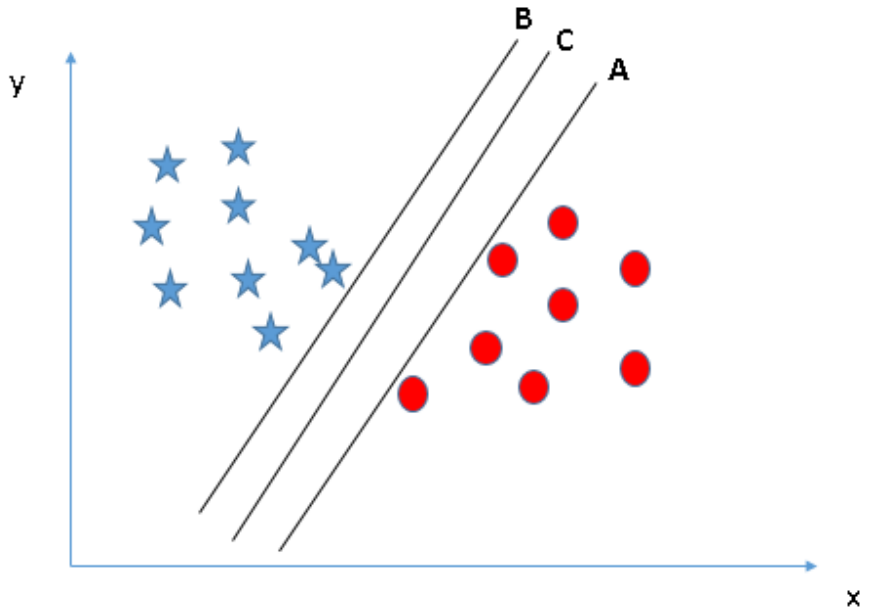


Fig (4-2) scenario (a)

Margin is maximum distance between the hyper plane and nearest data point

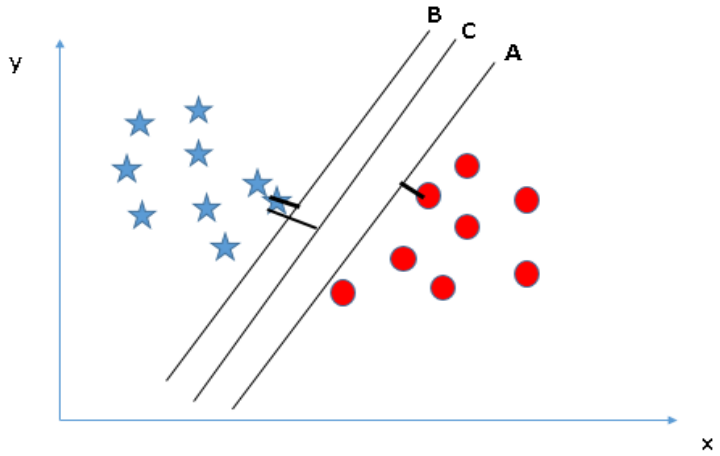


Fig (4-3) margin

Scenario (b):

Data of one class distribute with the other class as fig (4-4).

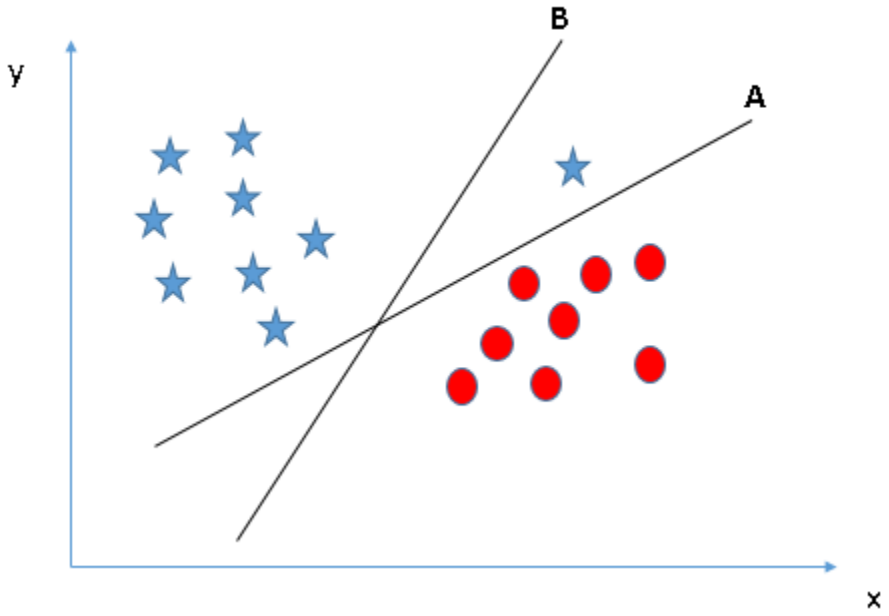


Fig (4-4)

The hyper plane (a) is the correct because of it classified all class correctly.

Scenario (c):

Data of one class distribute with the other class and one of star class lies on outlier as fig (4-5).

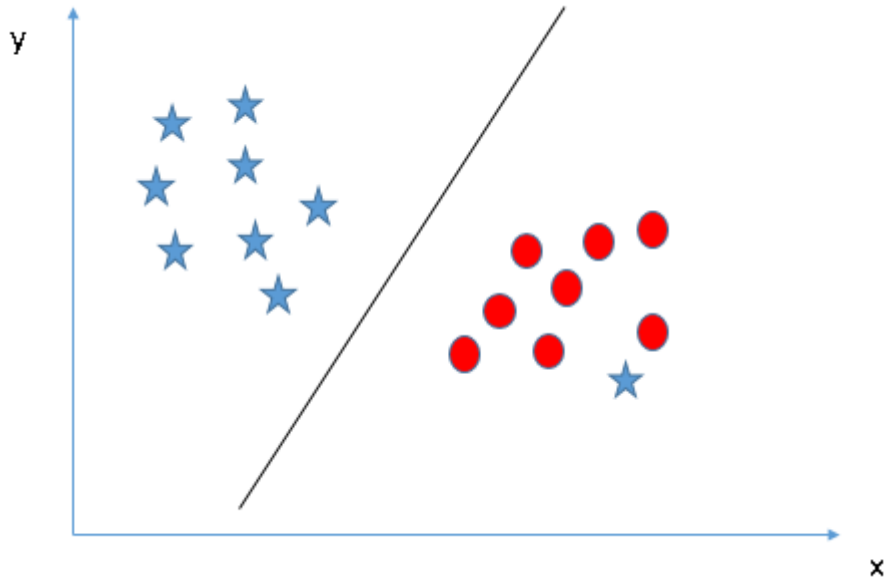


Fig (4-5)

The algorithm ignores the outlier star class that lies in circle class and made hyper plane as fig (4-5).

- **Hard margin classifier:** the data is completely separated by using linear kernel.
- **Soft margin classifier:** is an extended of hard margin classifier, real data are not separated perfectly by hyper plane into two classes by using linear (default) kernel.

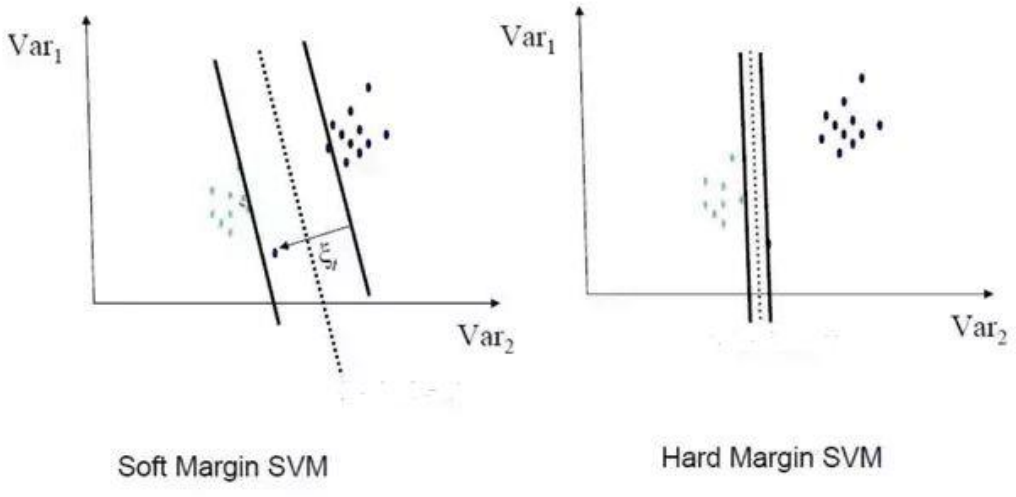


Fig (4-6)

4.2.1.2. Support Vector Machines (Kernels Tricks)

The kernel is the mathematical way to make optimization or transformation input space into high dimensions. The SVM is built in practice by using a kernel.

In python we use the library called **scikit-*learn***

1- Linear kernel SVM:

The kernel is calculating as the distance between the support vector and new input.

The kernel can calculate from that ***equation***:

$$K(x, xi) = sum(x * xi)$$

2- Polynomial Kernel SVM:

That allows curved lines in in input data. it must have specified the degree of polynomial by hande.it calculate by that equation:

$$K(x,xi) = 1 + sum(x * xi)^d$$

If d = 1, the performance of kernel equal to **linear kernel**.

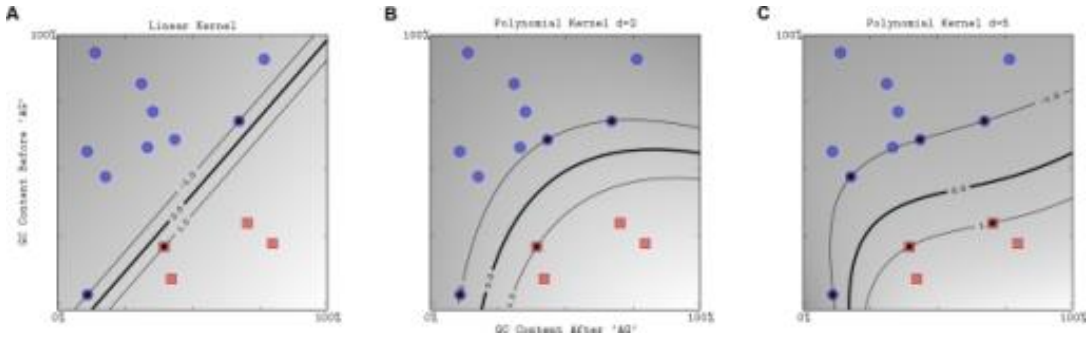


Fig (4-7)

3-Radial kernel SVM:

The kernel creates complex regions with features space. It is complex kernel, and it calculate from that equation

$$K(x,xi) = exp(-gamma * sum((x - xi^2))$$

Where the gamma have default value .1 and it is often between 0 and 1.

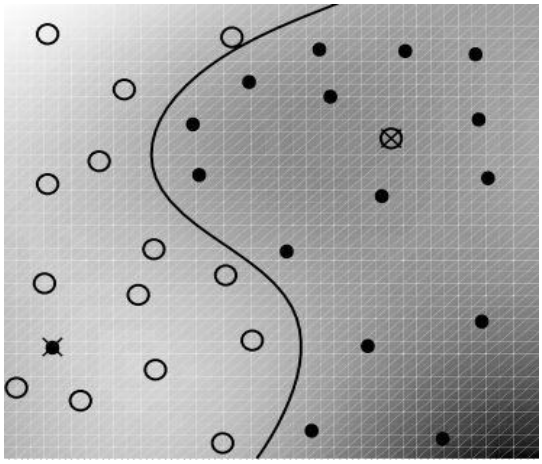


Fig (4-8) radial kernel

4.2.2. K nearest-neighbors algorithm

Is a supervised algorithm, it is calculating as the minimum distance between the unknown instance and the training data samples. The k is number of neighbors and must be odd and positive. It stores dataset so it no learning required.

The equation is

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + (x_n - x'_n)^2}$$

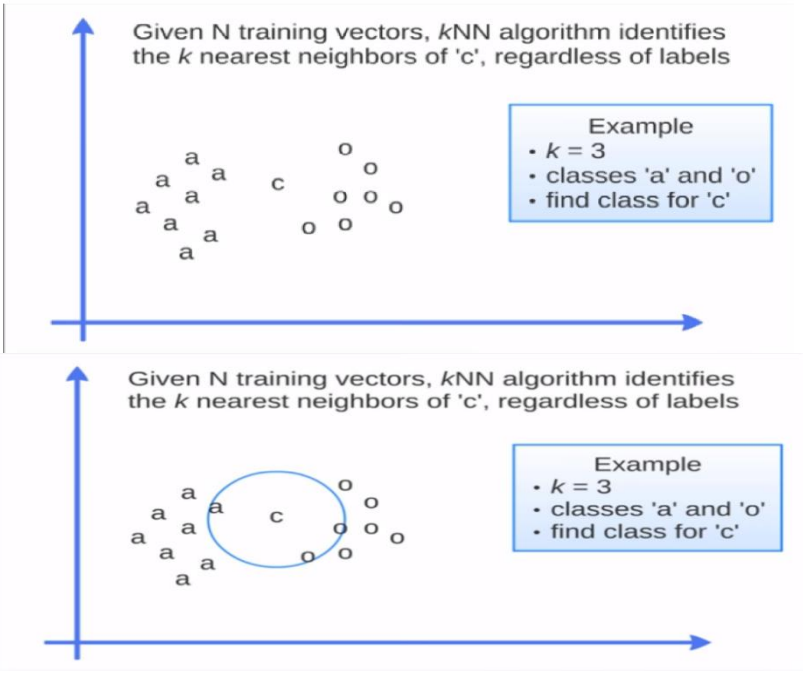
4.2.2.1. The knn classifier: is a non-parametric, instance based and lazy learning.

1) Instanced based: is the training instance to use in predication.

2) Lazy learning: not required learning.

3) Non parametric: not has assumption about of functional form of problem solved.

Example:



C is belonging to the (o) class because it has two neighbors of (o) class and one neighbors of (a) class.

4.2.3. Linear regression

Linear regression is very simple in representation. The representation is a linear equation between set of input values (x) and the predicted output for that set (y). As such, both input values (x) and output values (y) are numeric.

This linear equation assigns one scale factor to each input value, called a coefficient and represented by the capital Greek letter Beta (B). One additional

coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model is:

$$y = B0 + B1*x$$

In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients (e.g. B0 and B1 in the above example).

The complexity of a regression model like linear regression refers to the number of coefficients used in the model.

When a coefficient becomes zero, the prediction made from the model (0 * x = 0). This becomes relevant if you look at regularization methods that change the learning algorithm to reduce the complexity of regression models by putting pressure on the absolute size of the coefficients, driving some to zero.

Learning a linear regression model means estimating the values of the coefficients used in the representation with the available data.

Four techniques to prepare a linear regression model: These methods are effective to use when there is collinearity in input values and ordinary least squares would over fit the training data.

1. Simple Linear Regression

When we have a single input, we can use statistics to estimate the coefficients. This requires that you calculate statistical properties from the data such as means, standard deviations, correlations and covariance. All of the data must be available to traverse and calculate statistics.

2. Ordinary Least Squares

When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients.

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all of the squared errors together. This is the quantity that ordinary least squares seek to minimize.

This approach treats the data as a matrix and uses linear algebra operations to estimate the optimal values for the coefficients.

3. Gradient Descent

When there are one or more inputs you can use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data.

This operation is called Gradient Descent and works by starting with random values for each coefficient. The sum of the squared errors is calculated for each pair of input and output values. A learning rate is used as a scale factor and the coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved or no further improvement is possible.

It is useful when you have a very large dataset either in the number of rows or the number of columns that may not fit into memory.

4. Regularization

These seek to both minimize the sum of the squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of them.

Making Predictions with Linear Regression

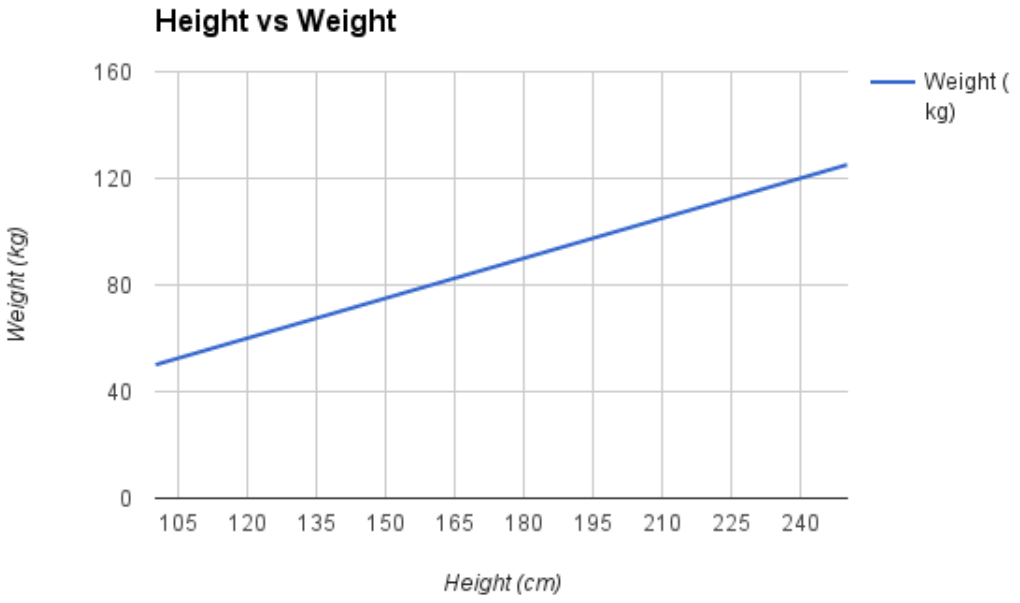
We can see that the above equation could be plotted height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

For example, let's use B0 = 0.1 and B1 = 0.5. Let's plug them in and calculate the weight (in kilograms) for a person with the height of 182 centimeters.

Weight = 0.1 + 0.05 * 182

Weight = 91.1

As a line in two-dimensions. The B0 is our starting point regardless of what height we have. We can run through a bunch of heights from 100 to 250 centimeters and plug them to the equation and get weight values, creating our line.



Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let's make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

$$y = B0 + B1 * x1$$

Or

$$Weight = B0 + B1 * height$$

Where B0 is the bias coefficient and B1 is the coefficient for the Sample Height vs Weight Linear Regression.

4.2.4. K-Means:

***K*-means clustering** is an unsupervised learning, which is used when you have unlabeled data (data without defined categories or groups). The goal of this algorithm is to find groups in this data, with the number of groups represented by (*K)*.

The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the *K*-means clustering algorithm are:

1. The centroids of the *K* clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

Business Uses

The *K*-means clustering algorithm is used to find groups which have not been explicitly labeled in the data. This can be used to confirm business assumptions about what types of groups exist or to identify unknown groups in complex data sets. Once the algorithm has been run and the groups are defined, any new data can be easily assigned to the correct group.

This is a versatile algorithm that can be used for any type of grouping. Some examples of use cases are:

- Behavioral segmentation:
  - Segment by purchase history
  - Segment by activities on application, website, or platform
  - Define personas based on interests
  - Create profiles based on activity monitoring
- Inventory categorization:
  - Group inventory by sales activity
  - Group inventory by manufacturing metrics
- Sorting sensor measurements:
  - Detect activity types in motion sensors
  - Group images
  - Separate audio
  - Identify groups in health monitoring
- Detecting bots or anomalies:
  - Separate valid activity groups from bots
  - Group valid activity to clean up outlier detection

In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.

Algorithm

The *K*-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters *K* and the data set. The data set is a collection of features for each data point. The algorithms start with initial estimates for the *K* centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. Data assignment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if *ci* is the collection of centroids in set *C*, then each data point *x* is assigned to a cluster based on

$$\underset{c_i \in C}{\operatorname{argmin}}\ dist(c_i,\ x)^2$$

Where *dist* (·) is the standard (*L*2) Euclidean distance. Let the set of data point assignments for each *ith* cluster centroid are *Si*.

2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|}\sum_{x_i \in S_i} x_i$$

The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

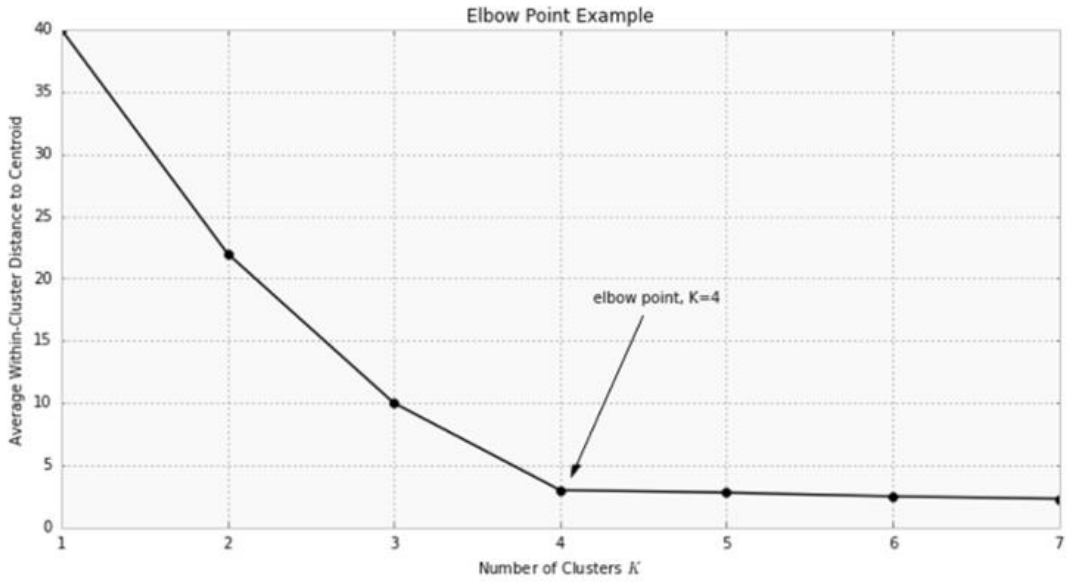
Choosing *K*

The algorithm described above finds the clusters and data set labels for a particular pre-chosen *K*. To find the number of clusters in the data, the user needs to run the *K*-means clustering algorithm for a range of *K* values and compare the results. In general, there is no method for determining the exact value of *K*, but an accurate estimate can be obtained using the following techniques.

One of the metrics that are commonly used to compare results across different values of *K* is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of *K* is plotted and the

"elbow point," where the rate of decrease sharply shifts, can be used to roughly determine *K*.

A number of other techniques exist for validating *K*, including cross-validation, information criteria, the information theoretic jump method, the silhouette method, and the G-means algorithm. In addition, monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each *K*.



4.3. Evaluating a Classifier:

The **evaluation of binary classifiers** compares two methods of assigning a binary attribute, one of which is usually a standard method and the other is being investigated. There are many metrics that can be used to measure the performance of a classifier or predictor; different fields have different preferences for specific metrics due to different goals. For example, in medicine sensitivity and specificity are often used, while in computer science precision and recall are preferred. An important distinction is between metrics that are independent on the prevalence (how often each category occurs in the population), and metrics that depend on the prevalence – both types are useful, but they have very different properties.

4.3.1. Methods of Evaluating a Classifier used in this project :

1 - Classification accuracy:

is the percentage of Correct Predictions , it computed using the true test class labels and predicted class labels , ex in image 3.1

```
('True: ', array([1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1]))
('pred: ', array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]))
```

image3.1

This how to calculate accuracy in our project:

```
# Calculate accuracy
from sklearn import metrics
print(metrics.accuracy_score(y_test , y_pred_class))
```

0.764705882353

2- Null Accuracy:

Accuracy that could achieved by always predicting the most frequency class

```
# calculating null accuracy(for binary only)
max(y_test.mean(), 1 - y_train.mean())
```

0.8235294117647058

Conclusion:

- Classification accuracy is the **easiest classification metric to understand**
- But, it does not tell you the **underlying distribution** of response values
- And, it does not tell you what **"types" of errors** your classifier is making

3 - Confusion Matrix:

Table that describes the performance of a classification model.

n=192	**Predicted: 0**	**Predicted: 1**
Actual: 0	118	12
Actual: 1	47	15

- Every observation in the testing set is represented in **exactly one box**
- It's a 2x2 matrix because there are **2 response classes**
- The format shown here is **not** universal

Basic terminology

- **True Positives (TP):** we *correctly* predicted that they *do* have diabetes

- **True Negatives (TN):** we *correctly* predicted that they *don't* have diabetes
- **False Positives (FP):** we *incorrectly* predicted that they *do* have diabetes (a "Type I error")
- **False Negatives (FN):** we *incorrectly* predicted that they *don't* have diabetes (a "Type II error")

Applying Confusion Matrix in this project , this is the output in image shown :

```
]: # important : true values first then predicted valuse
confusion = metrics.confusion_matrix(y_test, y_pred_class)
print(confusion)
```

```
[[ 0  3]
 [ 1 13]]
```

And this in more details:

```
('TP :', 13)
('TN', 0)
('FP', 3)
('FN', 1)
```

a- Metrics Computed from a confusion matrix:

1 - Classifier Accuracy: Overall, how often is the classifier correct:

```
print((TP + TN) / float(TP + TN + FP + FN))
print( clf.score(x_test, y_test))
```

```
0.764705882353
```

2- Classification Error (Misclassification rate): overall, how often is the classifier incorrect:

```
print(( FP + FN) / float(TP + TN + FP + FN))
print( 1 - clf.score(x_test, y_test) )
```

```
0.235294117647
0.235294117647
```

2- Sensitivity: when the actual values are positive, how often the prediction is correct **or** how 'sensitive' is the classifier to detecting positive instances.
Also known as 'true positive rate ' or recall:

```
print( TP / float(TP + FN))
print( metrics.recall_score(y_test, y_pred_class) )
```

```
0.928571428571
0.928571428571
```

4- Specificity: when the actual value is negative, how often is the prediction correct **or** how 'specific' or 'selective' is the classifier in prediction positive instances:

```
print( TN / float(TN + FP ))
```

5- Precision: when a positive value is predicted, how often the prediction is correct or how "precise" is the classifier when predicting positive instances:

```
print( TP / float(TP + FP))
print( metrics.precision_score(y_test, y_pred_class))
```

```
0.8125
0.8125
```

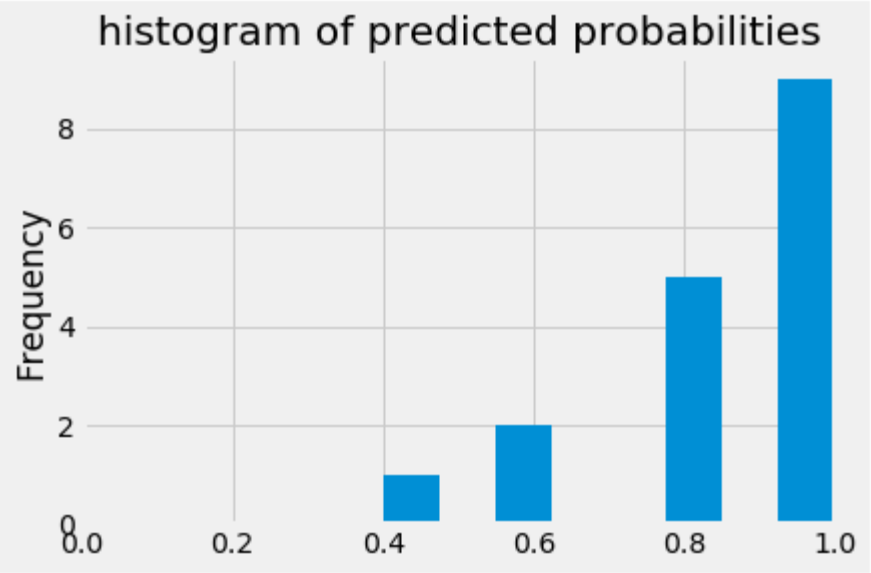
6 - Adjusting the Classifier Threshold:

- **Threshold of 0.5** is used by default (for binary problems) to convert predicted probabilities into class predictions
- Threshold can be **adjusted** to increase sensitivity or specificity
- Sensitivity and specificity have an **inverse relationship**

This is some output of class zero and one sample predictions:

```
clf.predict_proba(x_test)[0:10, :]
```

```
array([[ 0. ,  1. ],
       [ 0.2,  0.8],
       [ 0.4,  0.6],
       [ 0. ,  1. ],
       [ 0.2,  0.8],
       [ 0. ,  1. ],
       [ 0. ,  1. ],
       [ 0.2,  0.8],
       [ 0. ,  1. ],
       [ 0. ,  1. ]])
```

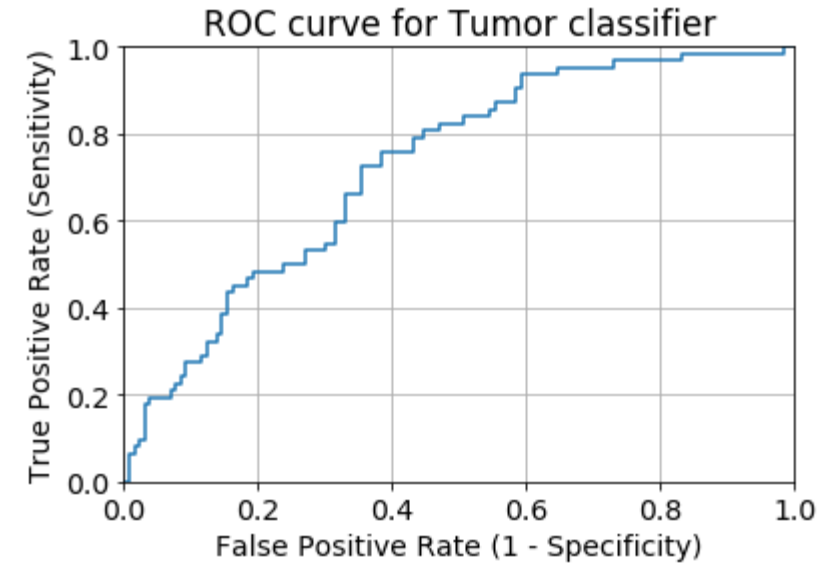


7- ROC Curves and Area under the Curve:

Different values of threshold produce different sensitivity and specificity.
Question: Wouldn't it be nice if we could see how sensitivity and specificity are affected by various thresholds, without actually changing the threshold?

Answer: Plot the ROC curve!

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for Tumor classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



Classification Performance Of The Classifiers

Classifier	Results					
	Accuracy	Misclassification Rate	Sensitivity Or Recall rate	Specificity	Precision	AUC
KNN	93.9%	6%	86.1%	97.8%	95.3%	96.9%
K means	94.2%	5.7%	90.7%	96.17%	92.6%	_
Linear regression	94.2%	5.7%	90.7%	96.1	92.6%	_
SVM	94.6%	5.3%	1%	91.8%	86.6%	98.03%

References

Websites:

- http://lecturer.ukdw.ac.id/~mahas/dossier/comvis_05.pdf
- http://scikit-image.org/docs/dev/auto_examples/
- http://scikitimage.org/docs/dev/auto_examples/features_detection/plot_glcm.html#sphx-glr-auto-examples-features-detection-plot-glcm-py
- https://www.youtube.com/watch?v=eNBZI-qYhpg
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4153941/
- https://www.mathworks.com/help/images/create-a-gray-level-co-occurrence-matrix.html
- https://en.wikipedia.org/wiki/Magnetic_resonance_imaging
- http://opencv.org/
- https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm\
- https://books.google.com.eg/books?id=f9-YCgAAQBAJ&pg=PA142&lpg=PA142&dq=filter+Weighted+Median+code+python&source=bl&ots=iWUAKGQh_g&sig=hAsauYiCEiUYCiQtBF-Pbdz67JM&hl=ar&sa=X&ved=0ahUKEwiD9cmF6rrSAhWMAsAKHeLtBYIQ6AEIKjAC#v=onepage&q=filter%20Weighted%20Median%20code%20python&f=false
- https://pythonprogramming.net/thresholding-image-analysis-python-opencv-tutorial/
- http://www.learnopencv.com/opencv-threshold-python-cpp
- http://www.indjst.org/index.php/indjst/article/viewFile/102154/73634
- https://en.wikipedia.org/wiki/Convex_hull
- https://octave.sourceforge.io/image/function/regionprops.html
- https://books.google.com.eg/books?id=SEAOgA-oCIkC&pg=PA1&lpg=PA1&dq=image+analysis+moment+definition&source=bl&ots=nhdXHdFpE1&sig=SrqIKkmKczIPvSQqMqKHz2rt1dY&hl=en&sa=X&sqi=2&ved=0ahUKEwjX3er6hoHVAhWBXRQKHdGxDKoQ6AEIPzAE#v=onepage&q=image%20analysis%20moment%20definition&f=false
- http://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node36.html
- http://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.moments
- http://scikit-image.org/docs/dev/api/skimage.measure.html?highlight=regionprops#skimage.measure.regionprops

- http://www.cs.mun.ca/~donald/bsc/node11.html
- https://en.wikipedia.org/wiki/Statistical_classification
- https://www.knime.org/blog/seven-techniques-for-data-dimensionality-reduction
- http://scikit-learn.org/stable/modules/decomposition.html#decompositions
- https://en.wikipedia.org/wiki/Dimensionality_reduction
- https://www.researchgate.net/profile/Erika_Amstalden_van_Hove/publication/228540194/figure/fig1/AS:301839783350274@1448975620988/Figure-1-The-Random-Forest-classifier-is-an-ensemble-of-decision-trees-where-the-single.png

Books:

- Chityala, Ravishankar; Pudipeddi, Sridevi Image Processing and Acquisition using Python

Articles:

- http://shodhganga.inflibnet.ac.in/bitstream/10603/13306/11/11_chapter%206.pdf
- https://www.researchgate.net/publication/49596787_A_Hybrid_Approach_for_Automatic_Classification_of_Brain_MRI_Using_Genetic_Algorithm_and_Support_Vector_Machine
- http://shodhganga.inflibnet.ac.in/bitstream/10603/20682/14/14_chapter%205.pdf
- Segmentation, Feature Extraction and Classification of Astrocytoma in MR Images (Indian Journal of Science and Technology,)
- BRAIN TUMOR MRI IMAGE CLASSIFICATION WITH FEATURE SELECTION AND EXTRACTION USING LINEAR DISCRIMINANT ANALYSIS (V.P.Gladis Pushpa Rathi1 and Dr.S.Palani)
- Brain Tumor Segmentation by FCM and Enhancement by ANN, using GLCM Based Feature Extraction (**Salabha Varghese)**

Appendix (A) Abbreviations

- **MRI……..Magnetic resonance imaging.**
- **CNS……..…Central nervous system.**
- **PCA……..……principal component analysis.**
- **T1W…….. (T1-Weighted).**
- **T2W…….. (T2-Weighted).**
- **CSF……Cerebrospinal Fluid**
- **src …… Source.**
- **pdf……..probability density function.**
- **2D ……two dimensional.**
- **GLCM……gray level co-occurrence matrix.**
- **SD……..Standard deviation.**
- **SVM……..Support vector machine.**
- **FN……..False Negatives.**
- **FP ……..False Positives.**
- **TN……True Negatives.**
- **TP……True Positives.**
- **PCA……..Principal component analysis.**

Appendix (B) Developed Codes

jupyter preprocessing.py ✔ Last Tuesday at 3:15 AM

File Edit View Language

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 22 15:42:10 2017

@author: mu7ammad
"""

import cv2
import numpy as np

from segmentaion import Segmentation

class Preprocessing(object):
    grayImage   = ''
    image       = ''
    closing1    = ''
    thresh1     = ''
    NO_skull    = ''
    erosion     = ''
    blur        = ''
    no_noise    = ''
    edge        = ''
    tumourImage = ''

    def preproces(self, originalImageUrl):
        image = cv2.imread(str(originalImageUrl));
        self.grayImage = cv2.cvtColor(image, cv2.COL(

        self.image = Segmentation(self.grayImage)

```

```
    # binarization
    def binarization(self):
        self.closing1, self.thresh1 = self.image.binarization()
        #plt.imshow(closing1,'gray')

    # removing_skul
    def removingSkul(self):
        self.NO_skull, self.erosion = self.image.removing_skul(self.closing1)
        #plt.imshow(NO_skull,'gray')

    # enhance image to segmentation
    def enhanceImage(self):
        self.blur = self.image.enhance_image_t_seg(self.NO_skull)
        #plt.imshow(blur,'gray')

    # Segmentation
    def segmentation(self):
        self.no_noise, self.thresh2 = self.image.to_Segmentation(self.blur)
        #plt.imshow(no_noise,'gray')

        self.edge = self.image.Edge_Detection(self.thresh2)

        # Plot
        #segm.Show_plots(img,thresh1,closing1,erosion,blur,NO_skull,no_noise)

    def getInfectedRegion(self):
        col1 = 0
        col2 = 0
        row1 = 0
        row2 = 0

        for i in xrange(self.no_noise.shape[1]):
```

```
        for i in xrange(self.no_noise.shape[1]):
            for j in xrange(self.no_noise.shape[0]):
                if self.no_noise.item(j, i) > 0:
                    if col1 == 0 & col2 == 0:
                        col1 = j
                        row1 = i
                    else:
                        if col1 > j:
                            col1 = j
                        else:
                            if col2 < j:
                                col2 = j

                        if row1 > i:
                            row1 = i
                        else:
                            if row2 < i:
                                row2 = i

        # draw rectangle to select tumour region
        #cv2.rectangle(self.no_noise, (row1, col1), (row2, col2), (255,0,0), 2)

        # replace tumour region by original pixles values
        self.no_noise[col1:col2, row1:row2] = self.grayImage[col1:col2, row1:row2]  * (- self.no_noise[col1:col2, row1:row2]    )

        # save again for good preview
        cv2.imwrite("./tmp/no_noise.jpg", self.no_noise)

        # tumourImage
        self.tumourImage = self.no_noise[col1:col2, row1:row2]
```

```
        # tumourImage
        self.tumourImage = self.no_noise[col1:col2, row1:row2]
        cv2.imwrite("./tmp/tumourImage.jpg", self.tumourImage)
        return col1, col2, row1, row2


preprocessing = Preprocessing()

preprocessing.preproces('/home/mu7ammad/workspace/Pythonwork/segmentation/images/FLAIR/img10.jpg')
preprocessing.binarization()
preprocessing.removingSkul()
preprocessing.enhanceImage()
preprocessing.segmentation()
preprocessing.getInfectedRegion()
```

jupyter segmentaion.py ✔ 04/22/2017

File Edit View Language

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 22 14:51:51 2017

@author: mu7ammad
"""

import cv2
import numpy as np
from matplotlib import pyplot as plt

kernel = np.ones((7,7),np.uint8)

class Segmentation(object):
    def __init__(self,Image):
        self.Image = Image
    def binarization(self):
        ret, thresh1 = cv2.threshold(self.Image,30,255,cv2.THRESH_BINARY)
        #convert to white
        closing1 = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)
        return closing1, thresh1

    #removing skul
    def removing_skul(self, closing1):
        erosion = cv2.erode(closing1,kernel,iterations = 6)
        NO_skull = self.Image * (-erosion)
        return NO skull ,erosion
```

```
    #enhance image to segmentation
    def enhance_image_t_seg(self, NO_skull):
        median = cv2.medianBlur(NO_skull,5)
        blur = cv2.GaussianBlur(median,(5,5),0)
        return blur

    def to_Segmentation(self, blur):
        ret, thresh2 = cv2.threshold(blur,120,255,cv2.THRESH_BINARY)
        img = thresh2
        #Remove noise
        no_noise = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
        return no_noise,thresh2

    #edge detection
    def Edge_Detection(self,thresh2):
        edges = cv2.Canny(thresh2,100,200)
        return edges

    def Show_plots(image,thresh1,closing1,erosion,blur,NO_skull,thresh2,opening):
        #show images
        titles = ['Original Image','Binary Image','Cleaned Binary img', 'Eroded img'
          ,'img without skull','Filtering img','Segmentation','final image']
        images = [image, thresh1, closing1, erosion, blur, NO_skull, thresh2, opening]
        for i in xrange(8):
            plt.subplot(4,4,i+1),plt.imshow(images[i],'gray')
            plt.title(titles[i])
            plt.xticks([]),plt.yticks([])
        plt.show()
```

jupyter **Preprocessing & Feature** Last Checkpoint: Yesterday at 3:37 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

In [26]:

```
from preprocessing import Preprocessing
from segmentaion import Segmentation
from matplotlib import pyplot as plt
from skimage import io
%matplotlib inline
import numpy as np
import pandas as pd
from skimage.feature import greycomatrix, greycoprops
from skimage import data
from skimage.color import rgb2gray
from scipy.stats import kurtosis ,skew
import skimage
from skimage import measure
from skimage.measure import label
from skimage.measure import regionprops
from skimage import io
from skimage.io import imread
import cv2

ds = []
crr = []
cn = []
am = []
en = []
```

```
en = []
ho = []
Class = []
av_kur1 = []
av_sk1 = []
std = []
Mean = []
# ---------------------------------------
centroid =[]
area =[]
extent = []
orientation = []
convex_area = []
solidity = []
eccentricity =  []
euler_number = []
```

In [2]:
```
# preprocessing and Segmentation
preprocessing = Preprocessing()
preprocessing.preproces('/home/mu7ammad/workspace/Pythonwork/nmpy/GP/im_Pr/SG/G_P/D2/no8.jpg')
preprocessing.binarization()
preprocessing.removingSkul()
preprocessing.enhanceImage()
preprocessing.segmentation()
image = preprocessing.getInfectedRegion()
```

In [3]:
```
# read image
# print(image)
im = io.imread('tmp/tumourImage.jpg')
plt.imshow(im, 'gray')
```

In [32]:
```
# Shape feature

# convert image to binart
thresh_img = cv2.threshold(im,127,255,cv2.THRESH_BINARY)[1]
image = thresh_img
m = skimage.measure.moments(image)
cr = m[0, 1] / m[0, 0]
cc = m[1, 0] / m[0, 0]

measure.moments_central(image, cr, cc)
#img = util.img_as_ubyte(data.coins()) > 110
label_img = label(image)
props = regionprops(label_img)
area.append(props[0]['area'])
extent.append(props[1].extent)
orientation.append(props[1].orientation)
convex_area.append( props[1].convex_area)
solidity.append(props[1].solidity)
eccentricity.append(props[1].eccentricity)
euler_number.append(props[2].euler_number)
Class.append(0)
print ( area)
print( extent)
print(orientation)
print(convex_area)
print(solidity)
print(eccentricity)
print(euler_number)

# # GLCM Texture Features
glcm = greycomatrix(im, [5], [0], symmetric=True, normed=True)
 ds.append(greycoprops(glcm, 'dissimilarity')[0,0])
```

```
# # GLCM Texture Features
glcm = greycomatrix(im, [5], [0], symmetric=True, normed=True)
 ds.append(greycoprops(glcm, 'dissimilarity')[0,0])
 crr.append(greycoprops(glcm, 'correlation')[0,0])
 cn.append(greycoprops(glcm, 'contrast')[0,0])
 am.append(greycoprops(glcm, 'ASM')[0,0])
 en.append(greycoprops(glcm, 'energy')[0,0])
 ho.append(greycoprops(glcm, 'homogeneity')[0,0])

# print('dissimilarity',ds)
# print('correlation',crr)
# print('contrast',cn)
# print('ASM',am)
# print('energy',en)
# print('homogeneity',ho)
# # kurtosis ,skew
# kur = kurtosis(im)
# sk1 = skew(im)
# av_kur1.append(np.average(kur))
# print(av_kur1)
# av_sk1.append(np.average(sk1))
# print(av_sk1)
# # Mean
# Mean.append(np.mean(im))
# print(Mean)
# # Standard deviation
# std.append(np.std(im))
```

In [122]:

```
dfr = pd.DataFrame({'area' :area,
                                    'extent' : extent,
                                    'orientation' :orientation ,
                                    'convex_area':convex_area,
                                    'solidity':solidity,
                                    'eccentricity':eccentricity,
                                    'euler_number':euler_number,
                                    'dissimilarity' :ds,
                                    'correlation' : crr,
                                    'contrast' :cn ,
                                    'ASM':am,
                                    'energy':en,
                                    'homogeneity':ho,
                                    'kurtosis':av_kur1,
                                    "skew":av_sk1,
                                    "Standard deviation":std,
                                    "Mean":Mean,
                                     'class':Class})
dfr.to_csv('feature_all.csv')
```

end of feature extraction

In [156]:
```
# df1 = pd.read_csv('Feature.csv', index_col=0)

# frames = [df,df1]
# result = pd.concat(frames, ignore_index=True)
df.to_csv("Feature1.csv")
```

Type *Markdown* and LaTeX: α^2

In [15]:
```
df = pd.read_csv("Feature1.csv", index_col=0)
print(df.head())
```

```
        ASM        Mean  Standard deviation  class      contrast  correlation  \
0  0.185655  157.536530          123.564873      1  13242.350111     0.566160
1  0.719861   26.550056           77.775920      1   3450.499768     0.703877
2  0.580958   51.437662          102.227630      1   2999.424930     0.855177
3  0.403010   74.065998          115.587507      1   5953.613842     0.776292
4  0.723073   26.128951           77.215294      1   3781.120309     0.655546

   dissimilarity    energy  homogeneity   kurtosis      skew
0      52.502389  0.430877     0.653675   1.391882 -0.841090
1      13.639474  0.848446     0.921099  42.618590  5.375214
2      11.895388  0.762206     0.919983   6.490145  2.314223
3      23.620536  0.634830     0.839195   0.690082  1.132864
4      14.946301  0.850337     0.912255  31.131325  4.634517
```

In [258]:
```
df_n = pd.read_csv('Feature_all.csv' )
df_n.drop('Unnamed: 0', 1, inplace=True)
df_n.replace('NaN', 1, inplace=True)
df_n
```

```
# feature selection with regression for texture feature only
from sklearn.linear_model import Lasso

df = pd.read_csv('brain-cancer.data')
# replace (?) with -99999 in data set
df.replace('?', -99999, inplace=True )
# # remove  id colunm cause it has no effect in learning
df.drop(['id'],1, inplace=True)
feature = df.drop(['class'],1)
labels =df['class']
classifier_f = open('n_y.pickle','wb')
pickle.dump(labels, classifier_f)
classifier_f.close()
regression = Lasso(alpha=0.1)
regression.fit(feature, labels)
# regression.predict([[0.62169290804739175,0.92176444484776776, 1]])
print(regression.coef_)
print(regression.intercept_)
```

```
[  7.80227669e-02   4.93247634e-02   6.73377622e-02   3.87812490e-02
   3.85950344e-03   2.42998184e-06   5.94999809e-02   3.44621575e-02
   0.00000000e+00]
1.55522222219
```

```
# feature = df[['homogeneity', 'correlation','convex_area']]
# labels =df['class']
# # print(labels)
# print(feature)
regression = Lasso(alpha=0.1)
regression.fit(feature, labels)
regression.predict([[0.62169290804739175,0.92176444484776776, 1]])
print(regression.coef_)
```

jupyter Feature Reduction (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 2

Code

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
```

In [3]:
```
ds = pd.read_csv('feature_all.csv', index_col=0)
# drop feature that will not have impact
# ds.drop(['solidity','euler_number' , 'extent'], 1 , inplace=True)
ds
```

Out[3]:

	ASM	Mean	Standard deviation	area	class	contrast	convex_area	correlation	dissimilarity	eccentricity	energy	euler_number	extent	homoge
0	0.285761	68.329483	80.143673	1	1	649.131029	1	0.949574	6.389174	0.000000	0.534566	1	1.000000	0.64
1	0.705255	20.817018	53.099576	4	1	513.964730	1	0.909389	4.205555	0.960659	0.839795	1	1.000000	0.86
2	0.199130	65.610310	69.660937	2	1	1148.323650	1	0.881897	10.931032	1.000000	0.446240	1	1.000000	0.55
3	0.485445	32.539409	59.810339	1	1	1172.741442	1	0.836718	10.137509	0.000000	0.696739	1	1.000000	0.74
4	0.156314	71.760435	68.979068	1	1	1211.991816	1	0.872515	11.240619	0.000000	0.395366	1	1.000000	0.52
5	0.463674	50.102691	84.501250	4	1	1238.504372	1	0.913783	11.061152	0.790569	0.680936	1	1.000000	0.71
6	0.907863	4.988141	25.988603	1	1	172.607495	1	0.872368	1.554613	0.000000	0.952818	1	1.000000	0.95
7	0.260564	70.583559	81.069658	1	1	842.722556	6	0.936146	8.949130	0.000000	0.510454	1	0.555556	0.58
8	0.573069	36.023047	73.131522	1	1	805.433210	1	0.925122	7.788582	0.000000	0.757013	1	1.000000	0.78
9	0.633504	27.142666	61.111553	1	1	533.784332	1	0.929096	5.000503	0.000000	0.795929	1	1.000000	0.82
28	0.261620	68.575738	80.258912	1	0	1070.372701	1	0.917187	9.589195	0.000000	0.511488	1	1.000000	0.60
29	0.902652	4.931866	26.000587	1	0	339.060142	2	0.698788	2.587353	0.000000	0.950080	1	1.000000	0.95
30	0.734374	13.271575	41.045604	1	0	764.440458	1	0.774556	6.075485	0.000000	0.856956	1	1.000000	0.87
31	0.796651	8.850966	36.167901	1	0	744.449611	2	0.719489	5.844501	0.000000	0.892553	1	0.500000	0.90
32	0.556797	29.835254	67.609871	3	0	1915.346237	1	0.792155	14.092974	0.816497	0.746188	1	1.000000	0.77
33	0.417277	37.402680	62.858996	1	0	1324.236072	1	0.833564	11.814829	0.000000	0.645970	1	1.000000	0.69

```
y = ds['class']
classifier_f = open('Y_labels.pickle','wb')
pickle.dump(y, classifier_f)
classifier_f.close()

y
# df1 = ds[['solidity','euler_number' , 'extent']]
# df1
```

```
# compute mean and std for all class 1 samples
# sample - (all feature samples std)

ds_class_1 = ds[(ds['class' ] > 0)]

ds_class_1.loc[0:,'Mean'] = (ds_class_1['Mean']) / ds_class_1['Mean'].std()
ds_class_1.loc[0:,'ASM'] = (ds_class_1['ASM']  / ds_class_1['ASM'].std())
ds_class_1.loc[0:,'contrast'] = (ds_class_1['contrast']) / ds_class_1['contrast'].std()
ds_class_1.loc[0:,'correlation'] = (ds_class_1['correlation']  /  ds_class_1['correlation'].std())
ds_class_1.loc[0:,'dissimilarity'] = (ds_class_1['dissimilarity'] / ds_class_1['dissimilarity'].std())
ds_class_1.loc[0:,'energy'] = (ds_class_1['energy']  /  ds_class_1['energy'].std())
ds_class_1.loc[0:,'kurtosis'] = (ds_class_1['kurtosis'] /  ds_class_1['kurtosis'].std())
ds_class_1.loc[0:,'skew'] = (ds_class_1['skew']  / ds_class_1['skew'].std())
ds_class_1.loc[0:,'Standard deviation'] = (ds_class_1['Standard deviation']  / ds_class_1['Standard deviation'].std())
ds_class_1.loc[0:,'area'] = (ds_class_1['area']) / ds_class_1['area'].std()
ds_class_1.loc[0:,'homogeneity'] = (ds_class_1['homogeneity']  / ds_class_1['homogeneity'].std())
ds_class_1.loc[0:,'orientation'] = (ds_class_1['orientation']  /  ds_class_1['orientation'].std())
ds_class_1.loc[0:,'convex_area'] = (ds_class_1['convex_area'] / ds_class_1['convex_area'].std())
ds_class_1.loc[0:,'eccentricity'] = (ds_class_1['eccentricity'] /  ds_class_1['eccentricity'].std())

ds_class_1.head()

# old Version
# ds_class_1['Mean'] = (ds_class_1['Mean'] - ds_class_1['Mean'].mean()) / ds_class_1['Mean'].std()
```

```
# compute mean and std for all class 1 samples
# sample - (all feature samples std)

ds_class_1 = ds[(ds['class' ] > 0)]

ds_class_1.loc[0:,'Mean'] = (ds_class_1['Mean']) / ds_class_1['Mean'].std()
ds_class_1.loc[0:,'ASM'] = (ds_class_1['ASM']  / ds_class_1['ASM'].std())
ds_class_1.loc[0:,'contrast'] = (ds_class_1['contrast']) / ds_class_1['contrast'].std()
ds_class_1.loc[0:,'correlation'] = (ds_class_1['correlation']  /  ds_class_1['correlation'].std())
ds_class_1.loc[0:,'dissimilarity'] = (ds_class_1['dissimilarity'] / ds_class_1['dissimilarity'].std())
ds_class_1.loc[0:,'energy'] = (ds_class_1['energy']  /  ds_class_1['energy'].std())
ds_class_1.loc[0:,'kurtosis'] = (ds_class_1['kurtosis'] /  ds_class_1['kurtosis'].std())
ds_class_1.loc[0:,'skew'] = (ds_class_1['skew']  / ds_class_1['skew'].std())
ds_class_1.loc[0:,'Standard deviation'] = (ds_class_1['Standard deviation']  / ds_class_1['Standard deviation'].std())
ds_class_1.loc[0:,'area'] = (ds_class_1['area']) / ds_class_1['area'].std()
ds_class_1.loc[0:,'homogeneity'] = (ds_class_1['homogeneity']  / ds_class_1['homogeneity'].std())
ds_class_1.loc[0:,'orientation'] = (ds_class_1['orientation']  /  ds_class_1['orientation'].std())
ds_class_1.loc[0:,'convex_area'] = (ds_class_1['convex_area'] / ds_class_1['convex_area'].std())
ds_class_1.loc[0:,'eccentricity'] = (ds_class_1['eccentricity'] /  ds_class_1['eccentricity'].std())

ds_class_1.head()

# old Version
# ds_class_1['Mean'] = (ds_class_1['Mean'] - ds_class_1['Mean'].mean()) / ds_class_1['Mean'].std()
```

	ASM	Mean	Standard deviation	area	class	contrast	convex_area	correlation	dissimilarity	eccentricity	energy	homogeneity	kurtosis	orientation
0	1.092141	2.624224	3.840221	0.970061	1	0.939040	0.970061	9.055433	1.083513	0.000000	2.669151	3.858496	-0.124202	1.157596
1	2.695391	0.799487	2.544357	3.880245	1	0.743507	0.970061	8.672212	0.713202	1.994929	4.193190	5.204400	2.290448	-1.021004
2	0.761049	2.519793	3.337922	1.940122	1	1.661177	0.970061	8.410037	1.853747	2.076624	2.228128	3.339160	0.177210	1.157596
3	1.855307	1.249690	2.865914	0.970061	1	1.696500	0.970061	7.979204	1.719177	0.000000	3.478897	4.451593	0.800922	1.157596
4	0.597412	2.755991	3.305250	0.970061	1	1.753280	0.970061	8.320575	1.906249	0.000000	1.974107	3.167200	0.862657	1.157596

```
# compute mean and std for all class( 0 )amples
# sample - (all feature samples std

ds_class_0= ds[(ds['class' ] < 1 )]

ds_class_0.loc[0:,'Mean'] = (ds_class_0['Mean']) / ds_class_0['Mean'].std()
ds_class_0.loc[0:,'ASM'] = (ds_class_0['ASM']  / ds_class_0['ASM'].std())
ds_class_0.loc[0:,'contrast'] = (ds_class_0['contrast']) / ds_class_0['contrast'].std()
ds_class_0.loc[0:,'correlation'] = (ds_class_0['correlation']  /  ds_class_0['correlation'].std())
ds_class_0.loc[0:,'dissimilarity'] = (ds_class_0['dissimilarity'] / ds_class_0['dissimilarity'].std())
ds_class_0.loc[0:,'energy'] = (ds_class_0['energy']  /  ds_class_0['energy'].std())
ds_class_0.loc[0:,'kurtosis'] = (ds_class_0['kurtosis'] /  ds_class_0['kurtosis'].std())
ds_class_0.loc[0:,'skew'] = (ds_class_0['skew']  / ds_class_0['skew'].std())
ds_class_0.loc[0:,'Standard deviation'] = (ds_class_0['Standard deviation']  / ds_class_0['Standard deviation'].std())
ds_class_0.loc[0:,'area'] = (ds_class_0['area']) / ds_class_0['area'].std()
ds_class_0.loc[0:,'homogeneity'] = (ds_class_0['homogeneity']  / ds_class_0['homogeneity'].std())
ds_class_0.loc[0:,'orientation'] = (ds_class_0['orientation']  /  ds_class_0['orientation'].std())
ds_class_0.loc[0:,'convex_area'] = (ds_class_0['convex_area'] / ds_class_0['convex_area'].std())
ds_class_0.loc[0:,'eccentricity'] = (ds_class_0['eccentricity'] /  ds_class_0['eccentricity'].std())
ds_class_0
```

	ASM	Mean	Standard deviation	area	class	contrast	convex_area	correlation	dissimilarity	eccentricity	energy	homogeneity	kurtosis	orientation
27	3.675442	0.103896	0.759560	1.322876	0	0.351665	2.04939	7.777457	0.367719	0.000000	5.711422	6.987396	2.766757	0.881917
28	1.033749	2.892825	3.455344	1.322876	0	1.819276	2.04939	10.680891	2.069208	0.000000	3.028985	4.348078	-0.124227	0.881917
29	3.566690	0.208048	1.119389	1.322876	0	0.576289	4.09878	8.137577	0.558313	0.000000	5.626291	6.887265	-0.204947	0.881917
30	2.901766	0.559853	1.767115	1.322876	0	1.299293	2.04939	9.019912	1.311000	0.000000	5.074820	6.324473	1.030161	0.881917
31	3.147842	0.373372	1.557117	1.322876	0	1.265316	4.09878	8.378647	1.261158	0.000000	5.285620	6.533516	0.941616	0.881917
32	2.200098	1.258582	2.910772	3.968627	0	3.255448	2.04939	9.224852	3.041057	2.645751	4.418862	5.606383	0.688473	-1.763834
33	1.648805	1.577809	2.706235	1.322876	0	2.250759	2.04939	9.707077	2.549467	0.000000	3.825375	5.011526	0.370145	0.881917

```
# concating all  features
frames = [ds_class_1 , ds_class_0]
ds_new = pd.concat(frames, ignore_index=True)
ds_new.to_csv('Feature_all_optomized.csv')
```

```
# feature selection with regression for texture feature only
from sklearn.linear_model import Lasso

df = pd.read_csv('brain-cancer.data')
# replace (?) with -99999 in data set
df.replace('?', -99999, inplace=True )
# # remove  id colunm cause it has no effect in learning
df.drop(['id'],1, inplace=True)
feature = df.drop(['class'],1)
labels =df['class']
classifier_f = open('n_y.pickle','wb')
pickle.dump(labels, classifier_f)
classifier_f.close()
regression = Lasso(alpha=0.1)
regression.fit(feature, labels)
# regression.predict([[0.62169290804739175,0.92176444484776776, 1]])
print(regression.coef_)
print(regression.intercept_)
```

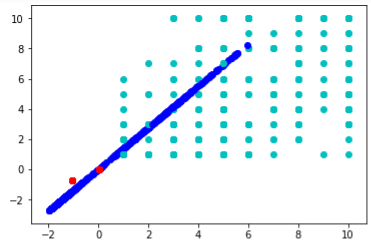
```
[  7.80227669e-02   4.93247634e-02   6.73377622e-02   3.87812490e-02
   3.85950344e-03   2.42998184e-06   5.94999809e-02   3.44621575e-02
   0.00000000e+00]
1.55522222219
```

```
# feature = df[['homogeneity', 'correlation','convex_area']]
# labels =df['class']
# # print(labels)
# print(feature)
regression = Lasso(alpha=0.1)
regression.fit(feature, labels)
regression.predict([[0.62169290804739175,0.92176444484776776, 1]])
print(regression.coef_)
```

```
print(regression.intercept_)
```

```
[-0.09523236 -0.04956931 -0.06734667]
1.75573420055
```

```
# PCA
from sklearn.decomposition import PCA , IncrementalPCA

def do_pca(data):
    pca = PCA(n_components=2)
    pca.fit(data)
    return pca

data = np.array(feature)
pca = do_pca(data)
print(pca.explained_variance_ratio_)
first_pc = pca.components_[0]
second_pc = pca.components_[1]

transformed_data = pca.transform(data)
# fil = open("pca_trans_data.pickle", 'wb')
# pickle.dump( transformed_data, fil)
# fil.close()

for ii , jj in zip(transformed_data, data):
    plt.scatter( first_pc[0] * ii[0], first_pc[1]*ii[0], color='r' )
    plt.scatter( second_pc[0] * ii[1], second_pc[1]*ii[1], color='b' )
    plt.scatter(jj[0], jj[1], color='c')


plt.show()
```



In [ ]:

In [16]:
```
# IncrementalPCA
ipca =  IncrementalPCA(n_components=3)
print(ipca)
ipca.fit(feature)
x =ipca.transform(feature)
print(ipca.explained_variance_ratio_)
```

```
IncrementalPCA(batch_size=None, copy=True, n_components=3, whiten=False)
[ 0.43748164  0.30472306  0.23822362]
```

In [17]:
```
# ipca.get_covariance()
ipca.get_precision()
ipca.get_params(deep=True)
```

Gaussian random projection

In [27]:
```
from sklearn import random_projection

transform  = random_projection.GaussianRandomProjection(n_components=5)
feature_new = transform.fit_transform(feature)
classifier_f = open('n_GaussianRandomProjection.pickle','wb')
pickle.dump(feature_new, classifier_f)
classifier_f.close()
print(feature_new)
```

```
[[  1.41077893  -1.2972483   -1.07751863   0.77263765   0.84400981]
 [ -4.90712009  -3.1844581   -2.72473208   7.15423346  -3.60527062]
 [  0.43485394  -0.43889526  -0.82541805   1.19678383   0.79633641]
 ...,
 [ 11.30531959  -2.3013658    0.53556658   7.10337376  -6.1247155 ]
 [  6.97983086  -0.78569674  -3.0884124    7.94605818  -4.78154031]
 [  4.16263553  -1.55291019  -2.75221864   8.36268833  -8.16222221]]
```

In [52]:

Out[52]: (34, 2)

In [64]:
```
# FeatureAgglomeration

from sklearn.cluster import FeatureAgglomeration

agglo = FeatureAgglomeration( n_clusters=2)

agglo.fit(feature)
X_reduced = agglo.transform(feature)
# X_reduced
```

jupyter Final_Test_evaluate (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

In [1]:
```
# import Moduals and lib.
import Evaluation
import numpy as np
from sklearn import preprocessing, neighbors, svm
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
import warnings
from matplotlib import pyplot as plt
from matplotlib import style
from collections import Counter
import pickle
import random
from sklearn import metrics
style.use('fivethirtyeight')
```

Preparing Data

In [2]:
```
#  Load Feaure data
fil = open('n_GaussianRandomProjection.pickle', 'rb')
X = pickle.load(fil)
fil.close()

# Load  Labels of this Feature
fil = open('n_y.pickle', 'rb')
Y = pickle.load(fil)
fil.close()
for i in range(len(Y)):
    if Y[i]  == 2:
        Y[i] = 0
    elif Y[i] == 4:
        Y[i] = 1

# split data into Train and Teast

x_train , x_test, y_train , y_test = train_test_split(X, Y,  test_size=0.4)
```

In [ ]:

Train and Test with SVM , KNN, Linear Regerassion,Kmeans

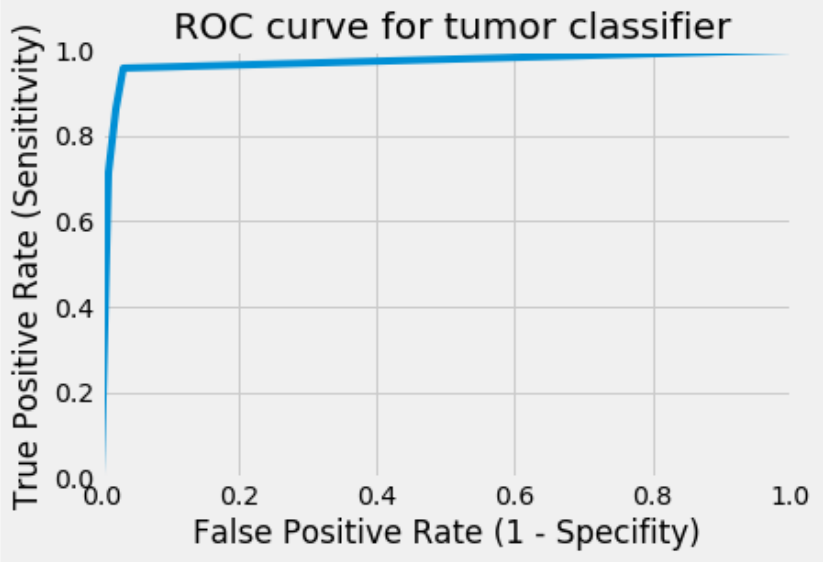
In [3]:
```
# implementing the k-nearest neighbors classifier
Knn_Cl = neighbors.KNeighborsClassifier(n_neighbors=3)
Knn_Cl.fit(x_train, y_train)

# prediction
Knn_y_pred_class = Knn_Cl.predict(x_test)
```

Evaluate KNN

In [4]:
```
evaluation  = Evaluation.evaluation_fun (Knn_Cl, y_test, Knn_y_pred_class,x_test)
```

```
evaluation.Accuracy()
evaluation.Confusion_Matrix()
evaluation.ROC_Curves()
```

```
('Accuracy: ', 0.93928571428571428)
('Null Accuracy: ', 0.6642857142857144)
('TP :', 81)
('TN', 182)
('FP', 4)
('FN', 13)
('Misclassifier Rate :', 0.060714285714285714)
(' Sensitivity or recall rate :', 0.86170212765957444)
('Specificity: ', 0.978494623655914)
('FALSE Positive Rate: ', 0.021505376344086023)
('Precision: ', 0.95294117647058818)
0.96922900938
```

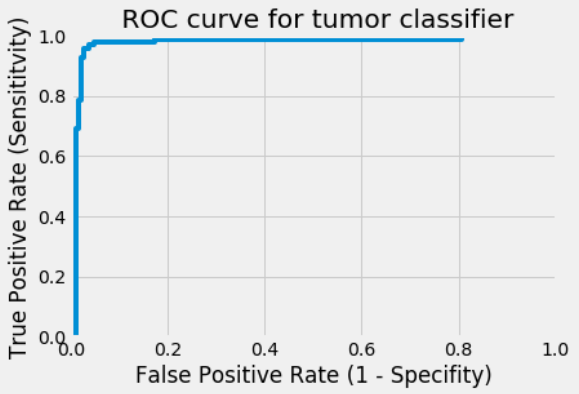


SVM

```
]: # implementing the svm classifier
SVM_CL= svm.SVC(probability=True)
SVM_CL.fit(x_train,y_train)

# prediction
SVM_y_pred_class = SVM_CL.predict(x_test)
y_pred_prob = SVM_CL.predict_proba(x_test)[:, 1]
```

```
evaluation  = Evaluation.evaluation_fun (SVM_CL, y_test, SVM_y_pred_class, x_test)
evaluation.Accuracy()
evaluation.Confusion_Matrix()
evaluation.ROC_Curves()
```

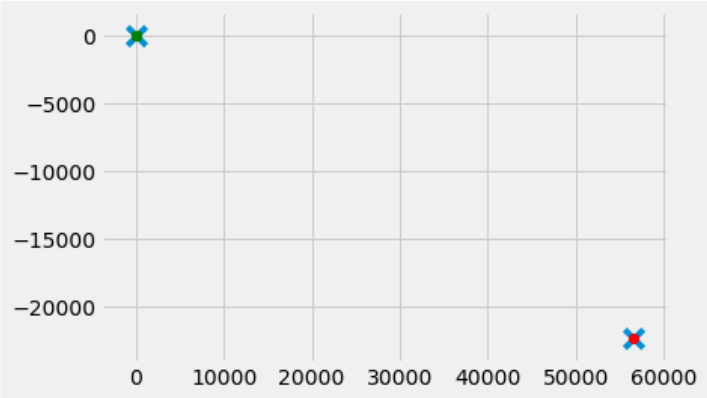
```
('Accuracy: ', 0.96071428571428574)
('Null Accuracy: ', 0.6642857142857144)
('TP :', 90)
('TN', 179)
('FP', 7)
('FN', 4)
('Misclassifier Rate :', 0.039285714285714285)
(' Sensitivity or recall rate :', 0.95744680851063835)
('Specificity: ', 0.9623655913978495)
('FALSE Positive Rate: ', 0.037634408602150539)
('Precision: ', 0.92783505154639179)
0.975863646763
```



K-Means

```
KM_CL =  KMeans(n_clusters =2)
KM_CL.fit(X)
centroids = KM_CL.cluster_centers_
labels = KM_CL.labels_

colors =10* ["g.", "r.","c.","b.","k."]

for i in range(len(X)):
    plt.plot(X[i][0] , X[i][1], colors[labels[i]], markersize=10)
```

```
    plt.plot(X[i][0] , X[i][1], colors[labels[i]], markersize=10)
plt.scatter(centroids[:,0],centroids[:,1], marker='x', s=150, linewidths=5)
plt.show()
```




```
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1 , len(predict_me))
    #     print(predict_me)
    prediction = KM_CL.predict(predict_me)
    if prediction[0] == Y[i]:
        correct +=1

print(correct / float(len(X)))
```

```
from statistics import mode
import nltk
class VoteClassifier(ClassifierI):
    def __init__ (self, *classifiers):
        self._classifiers = classifiers

    def classify(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)
        return mode(votes)

    def confidence(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)

        choice_votes = votes.count(mode(votes))
        print(choice_votes)
        conf = float(choice_votes) / len(votes)
        return conf

voted_classifier =  VoteClassifier(Knn_Cl, SVM_CL, KM_CL)
print(" Voted_classifier  accuracy percent", (nltk.classify.accuracy(voted_classifier, x_test)) *100)
print("Classification :", voted_classifier.classify(x_test[0][0]), "confidence %:", float(voted_classifier.confidence(x_test[0][0
print("Classification :", voted_classifier.classify(x_test[1][0]), "confidence %:", float(voted_classifier.confidence(x_test[1][0
print("Classification :", voted_classifier.classify(x_test[2][0]), "confidence %:", float(voted_classifier.confidence(x_test[2][0
print("Classification :", voted_classifier.classify(x_test[3][0]), "confidence %:",float( voted_classifier.confidence(x_test[3][0
print("Classification :", voted_classifier.classify(x_test[4][0]), "confidence %:", float(voted_classifier.confidence(x_test[4][0
print("Classification :", voted_classifier.classify(x_test[5][0]), "confidence %:", float(voted_classifier.confidence(x_test[5][0
```

jupyter Evaluation.py ✔ an hour ago

File Edit View Language

```python
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import style
from sklearn import metrics
style.use('fivethirtyeight')



class evaluation_fun():
    def __init__ (self,  classifier, y_test, y_pred_class,x_test):
        self.classifier = classifier
        self.y_test = y_test
        self.x_test = x_test
        self.y_pred_class = y_pred_class


    def Accuracy(self):
        print('Accuracy: ', metrics.accuracy_score(self.y_test , self.y_pred_class))
        print('Null Accuracy: ', max(self.y_test.mean(),  1 - self.y_test.mean()))

    def Confusion_Matrix(self):
        confusion = metrics.confusion_matrix(self.y_test, self.y_pred_class)
        TP = confusion[1, 1]
        TN = confusion[0, 0]
        FP = confusion[0, 1]
        FN = confusion[1, 0]
        print('TP :' , TP)
        print('TN', TN)
```

```python
    def Confusion_Matrix(self):
        confusion = metrics.confusion_matrix(self.y_test, self.y_pred_class)
        TP = confusion[1, 1]
        TN = confusion[0, 0]
        FP = confusion[0, 1]
        FN = confusion[1, 0]
        print('TP :' , TP)
        print('TN', TN)
        print('FP', FP)
        print('FN', FN)
        print('Misclassifier Rate :',(FP + FN) / float(TP + TN + FP + FN))
        print( " Sensitivity or recall rate :", TP / float(TP + FN))
        print( 'Specificity: ' ,TN / float(TN + FP ))
        print('FALSE Positive Rate: ',  FP / float( TN + FP ))
        print("Precision: ",  TP / float(TP + FP))



    def ROC_Curves(self):
        y_pred_prob = self.classifier.predict_proba(self.x_test)[:, 1]
        fpr , tpr, thresholds = metrics.roc_curve(self.y_test ,y_pred_prob)
        print(metrics.roc_auc_score(self.y_test, y_pred_prob))
        plt.plot(fpr, tpr)
        plt.xlim([0.0 , 1.0])
        plt.ylim([0.0 , 1.0])
        plt.title(' ROC curve for tumor classifier')
        plt.xlabel('False Positive Rate (1 - Specifity)')
        plt.ylabel('True Positive Rate (Sensititvity)' )
        plt.grid(True)
        plt.show()

```

jupyter Training_testing Last Checkpoint: Last Tuesday at 4:43 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

In [1]:
```
import numpy as np
from sklearn import preprocessing, neighbors, svm
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
import pandas as pd
import warnings
from matplotlib import pyplot as plt
from matplotlib import style
from collections import Counter
import pickle
import random
style.use('fivethirtyeight')
```

In [12]:
```
# Preparing Data
df = pd.read_csv('Feature_all_optomized.csv')

# define train and test data
# X = np.array(df.drop(['class'], 1)) # put every thing in array except 'class' label
Y = np.array(df['class'])  # array have 'class' only
fil = open('GaussianRandomProjection.pickle', 'rb')
X = pickle.load(fil)
fil.close()
# split data in train and test groups
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.4)
```

```
# implementing the k-nearest neighbors classifier
clf = neighbors.KNeighborsClassifier()
clf.fit(x_train, y_train)
# classifier_f = open('KN_algo.pickle','wb')
# pickle.dump(clf, classifier_f)
# classifier_f.close()
# test the accuracy of algorithm|
accuracy = clf.score(x_test, y_test)
print(accuracy)
```

```
0.857142857143
```

In [5]:
```
# save classifier
classifier_f = open('KN_algo.pickle','wb')
pickle.dump(clf, classifier_f)
classifier_f.close()
```

In [ ]:
```
# make a prediction

ex_measure = np.array([0.185655,157.536530,123.564873,13242.350111,0.566160,52.502389,0.430877, 0.653675,1.391882,5.375214])
# print(ex_measure)

ex_measure = ex_measure.reshape(1, -1)
# print(ex_measure)
prediction = clf.predict(ex_measure)

print(prediction)
```

SVM Classifier

In [14]:
```
# implementing the svm classifier
clf1 = svm.SVC()
clf1.fit(x_train,y_train)

# test the accuracy of algorithm
accuracy = clf1.score(x_test, y_test)
print(accuracy)
```

```
0.857142857143
```

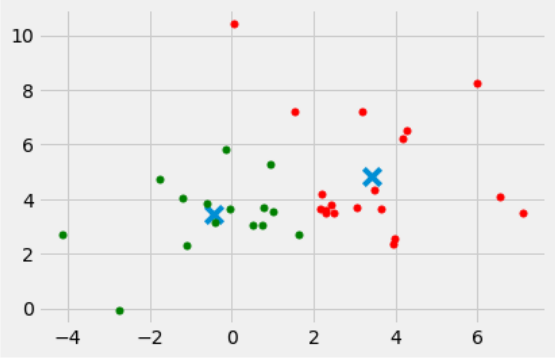
In [56]:
```
# save classifier
classifier_f = open('SVM.pickle','wb')
pickle.dump(clf1, classifier_f)
classifier_f.close()
```

In [36]:
```
ex_measure = np.array([0.185655,157.536530,123.564873,13242.350111,0.566160,52.502389,0.430877, 0.653675,1.391882,5.375214])
# print(ex_measure)

ex_measure = ex_measure.reshape(1, -1)
# print(ex_measure)
prediction = clf1.predict(ex_measure)

print(prediction)
```

```
[1]
```

KMeans

In [15]:
```
clf =  KMeans(n_clusters =2)
clf.fit(X)
centroids = clf.cluster_centers_
labels = clf.labels_

colors =10* ["g.", "r.","c.","b.","k."]

for i in range(len(X)):
    plt.plot(X[i][0] , X[i][1], colors[labels[i]], markersize=10)
plt.scatter(centroids[:,0],centroids[:,1], marker='x', s=150, linewidths=5)
plt.show()
```



In [20]:
```
# data = preprocessing.scale(X)
data = X
clf = KMeans(n_clusters=2)
clf.fit(data)
# print(clf)

correct = 0
for i in range(len(data)):
    predict_me = np.array(data[i].astype(float))
    predict_me = predict_me.reshape(-1 , len(predict_me))
    #     print(predict_me)
    prediction = clf.predict(predict_me)
    if prediction[0] == Y[i]:
        correct +=1

print(correct / float(len(X)))
```

```
0.529411764706
```

In [57]:
```
# save classifier
classifier_f = open('Kmean.pickle','wb')
pickle.dump(clf, classifier_f)
classifier_f.close()
```

In [5]:
```
# new data with Feature Reduction

fil = open('pca_trans_data.pickle', 'rb')
data = pickle.load(fil)
fil.close()
```

In [21]:
```
clf = KMeans(n_clusters=2)
clf.fit(data)
# print(clf)

correct = 0
for i in range(len(data)):
    predict_me = np.array(data[i].astype(float))
    predict_me = predict_me.reshape(-1 , len(predict_me))
    #     print(predict_me)
    prediction = clf.predict(predict_me)
    if prediction[0] == Y[i]:
        correct +=1

print(correct / float(len(X)))
```

```
0.470588235294
```

In [15]:
```
fil = open('y.pickle', 'rb')
Y = pickle.load(fil)
fil.close()
x_train, x_test, y_train, y_test = train_test_split(data, Y, test_size = 0.3)
# implementing the k-nearest neighbors classifier
clf = neighbors.KNeighborsClassifier()
clf.fit(x_train, y_train)
classifier_f = open('KN_algo.pickle','wb')
pickle.dump(clf, classifier_f)
classifier_f.close()
# test the accuracy of algorithm|
accuracy = clf.score(x_test, y_test)
print(accuracy)
```

In [22]:
```
# implementing the svm classifier
clf1 = svm.SVC()
clf1.fit(x_train,y_train)

# test the accuracy of algorithm
accuracy = clf1.score(x_test, y_test)
print(accuracy)
```

```
0.857142857143
```

Null Accuracy : Accuracy that could achieved by always predicting the most frequency class

In [49]:
```
# Examine the class Distribution of the testing set
y_test.value_counts()
```

Out[49]:
```
1    12
0     2
Name: class, dtype: int64
```

In [50]:
```
# calculate the precentage of ones
y_test.mean()

# calculate the precenage of zeros
1 - y_train.mean()

# calculating null accuracy(for binary only)
max(y_test.mean(),  1 - y_train.mean())
```

Out[50]:
```
0.8571428571428571
```

In [51]:
```
# calculating null accuracy(for multi-class )
print(y_test.value_counts().head(1) / len(y_test))
```

```
1    0.857143
Name: class, dtype: float64
```

In [52]:
```
# comparing the true and predicted responses

print( 'True: ' , y_test.values[0:])
print( 'pred: ' , y_pred_class)
```

Confusion Matrix

Table that Describes the preformance of a classification Model

In [53]:
```
# important : true values first then predicted valuse
confusion = metrics.confusion_matrix(y_test, y_pred_class)
TP = confusion[1,1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1 , 0]
```

In [54]:
```
print(TP ,TN,FP,FN)
```

```
(11, 0, 2, 1)
```

Metrics Computed from a confusion matric

classifier Accuracy : Overall, how often is the classifier correct ?

In [58]:
```

print((TP + TN) / float(TP + TN + FP + FN))
print( clf.score(x_test, y_test))
```

```
0.785714285714
0.785714285714
```

Classification Error : overall, how often is the classifier incorrect ?

Misclassifier Rate

In [59]:
```
print(( FP + FN) / float(TP + TN + FP + FN))
print( 1 - clf.score(x_test, y_test) )
```

```
0.214285714286
0.214285714286
```

Sensitivity : when the actual is positive, how often is the prediction correct ?

*** How 'sensitive' is the classifier to detecting positive instances ?**

*** also known as 'true positive rate ' or recall**

In [61]:
```
print( TP / float(TP + FN))
print( metrics.recall_score(y_test, y_pred_class) )
```

```
0.916666666667
0.916666666667
```

Specifity : when the actual value is negative, how often is the prediction correct ?

*** How 'specific' or 'selective' is the classifier in predictin positive instances ?**

Specifity : when the actual value is negative, how often is the prediction correct ?

*** How 'specific' or 'selective' is the classifier in predictin positive instances ?**

In [62]:
```
print( TN / float(TN + FP ))
```

```
0.0
```

FALSE Positive rate : when the actual value in negative, how often is the prediction incorrect ?

In [64]:
```
print( FP / float( TN + FP ))
```

```
1.0
```

In [ ]:
```
FALSE Positive rate : when the actual value in negative, how often is the prediction incorrect ?
```

jupyter model_evaluation (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Markdown

Comparing machine learning models in scikit-learn

From the video series: Introduction to machine learning with scikit-learn

Agenda

- How do I choose **which model to use** for my supervised learning task?
- How do I choose the **best tuning parameters** for that model?
- How do I estimate the **likely performance of my model** on out-of-sample data?

Review

- Classification task: Predicting the species of an unknown iris
- Used three classification models: KNN (K=1), KNN (K=5), logistic regression
- Need a way to choose between the models

Solution: Model evaluation procedures

Evaluation procedure #1: Train and test on the entire dataset

1. Train the model on the **entire dataset**.
2. Test the model on the **same dataset**, and evaluate how well we did by comparing the **predicted** response values with the **true** response values.

In [2]:
```
# read in the iris data
from sklearn.datasets import load_iris
iris = load_iris()

# create X (features) and y (response)
X = iris.data
y = iris.target
```

Logistic regression

In [3]:
```
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X, y)

# predict the response values for the observations in X
logreg.predict(X)
```

In [4]:
```
# store the predicted response values
y_pred = logreg.predict(X)

# check how many predictions were generated
len(y_pred)
```

Out[4]: 150

Classification accuracy:

- **Proportion** of correct predictions
- Common **evaluation metric** for classification problems

In [5]:
```
# compute classification accuracy for the logistic regression model
from sklearn import metrics
print(metrics.accuracy_score(y, y_pred))
```

```
0.96
```

- Known as **training accuracy** when you train and test the model on the same data

KNN (K=5)

In [6]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)
y_pred = knn.predict(X)
print(metrics.accuracy_score(y, y_pred))
```

KNN (K=1)

In [7]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
y_pred = knn.predict(X)
print(metrics.accuracy_score(y, y_pred))
```

```
1.0
```

Problems with training and testing on the same data

- Goal is to estimate likely performance of a model on **out-of-sample data**
- But, maximizing training accuracy rewards **overly complex models** that won't necessarily generalize
- Unnecessarily complex models **overfit** the training data

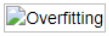


Image Credit: Overfitting by Chabacano. Licensed under GFDL via Wikimedia Commons.

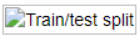
Evaluation procedure #2: Train/test split

1. Split the dataset into two pieces: a **training set** and a **testing set**.
2. Train the model on the **training set**.
3. Test the model on the **testing set**, and evaluate how well we did.

In [8]:
```
# print the shapes of X and y
print(X.shape)
print(y.shape)
```

```
(150L, 4L)
(150L,)
```

In [9]:
```
# STEP 1: split X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=4)
```



What did this accomplish?

- Model can be trained and tested on **different data**
- Response values are known for the testing set, and thus **predictions can be evaluated**
- **Testing accuracy** is a better estimate than training accuracy of out-of-sample performance

In [10]:
```
# print the shapes of the new X objects
print(X_train.shape)
print(X_test.shape)
```

```
(90L, 4L)
(60L, 4L)
```

In [11]:
```
# print the shapes of the new y objects
print(y_train.shape)
print(y_test.shape)
```

```
(90L,)
(60L,)
```

In [12]:
```
# STEP 2: train the model on the training set
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Out[12]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [13]:
```
# STEP 3: make predictions on the testing set
y_pred = logreg.predict(X_test)

# compare actual response values (y_test) with predicted response values (y_pred)
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.95
```

Repeat for KNN with K=5:

In [14]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

In [15]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

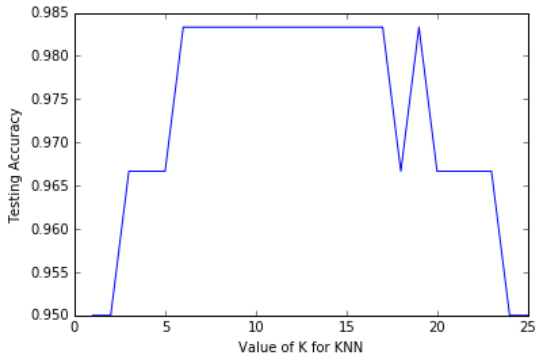
```
0.95
```

Can we locate an even better value for K?

In [16]:
```
# try K=1 through K=25 and record testing accuracy
k_range = list(range(1, 26))
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))
```

In [17]:
```
# import Matplotlib (scientific plotting library)
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Out[17]: `<matplotlib.text.Text at 0x8ffad68>`



- **Training accuracy** rises as model complexity increases
- **Testing accuracy** penalizes models that are too complex or not complex enough
- For KNN models, complexity is determined by the **value of K** (lower value = more complex)

Making predictions on out-of-sample data

In [18]:
```
# instantiate the model with the best known parameters
knn = KNeighborsClassifier(n_neighbors=11)

# train the model with X and y (not X_train and y_train)
knn.fit(X, y)

# make a prediction for an out-of-sample observation
knn.predict([[3, 5, 4, 2]])
```

Out[18]: `array([1])`

Downsides of train/test split?

- Provides a **high-variance estimate** of out-of-sample accuracy
- **K-fold cross-validation** overcomes this limitation
- But, train/test split is still useful because of its **flexibility and speed**

S

Members Rule

Member name	Rule
1.Samar Muhammed	• Segmentation
2.Shrouk Ahmed	• Preprocessing • Book formating
3.Hagar Atef	• Searching about feature • Select shape figure
4.Alshimaa Taha	• Searching about feature • Select texture figure
5.Muhammed Fathi	• Feature extraction • Machine learning • **Team leader**
6.Muhammed Kamal	• Machine learning
7.Mustafa Hamdy	• Machine learning