M.Tech. Dissertation Series

Classification of Normal and Fatty Liver Ultrasound Images

A dissertation submitted in partial fulfillment of the requirements for the M.Tech.(Computer Science) degree of the Indian Statistical Institute

By

Manoj Kumar Nanda

With the guidance of

Dr. Pradipta Maji

Assistant Professor
Machine Intelligence Unit
Indian Statistical Institute, Kolkata

and

Dr. Kuntal Ghosh

Lecturer
Machine Intelligence Unit
Indian Statistical Institute, Kolkata

INDIAN STATISTICAL INSTITUTE

203,Barrackpore Trunk Road

Kolkata 700108

CERTIFICATE

This is to certify that the thesis entitled **"Classification of Normal and Fatty Liver ultrasound Images"** is submitted in the partial fulfillment of the degree of M. Tech. in Computer Science at Indian Statistical Institute, Kolkata. It is fully adequate in scope and quality as a dissertation for the required degree.

This thesis is a faithful record of research work carried out by Manoj Kumar Nanda under my supervision and guidance. It is further certified that no part of this thesis has been submitted to any other university or institute for the award of any degree or diploma.

Dr. Kuntal Ghosh
(Supervisor)
MIU, ISI – Kolklata

Dr. Pradipta Maji
(Supervisor)
MIU, ISI – Kolklata

Countersigned
(External Examiner)
Date:

Acknowledgments

I take my pleasure in thanking my Guides **Dr. Pradipta Maji** (Asst. Professor, ISI-Kolkata) and **Dr. Kuntal Ghosh** (Lecturer, ISI-Kolkata) for their valuable guidence and readiness for anytime-help throughout the thesis work. Their pleasent and encouraging words have always kept my spirits up.

Using this opportunity, I want to thank Shashank singh (M.Tech.(CS), ISI Kolkata) for his invaluable help and encouragement, due to which only, it could be possible to complete this work in time. I also want to thank Kalikinkar Mandal(M.Tech.(CS), ISI Kolkata) and my other Friends for their help in this work.

Finally, I want to thank My Family for their unlimited patience and support, and kind blessings.

Manoj Kumar Nanda
July, 2009
Homepage : http://sites.google.com/site/manojkrnanda

Contents

Chapter 1

Introduction

Liver diseases are taken seriously because of the liver's vital importance to the life of the patient. Fatty liver is a common liver disease caused by accumulation of fat in liver cells via the process of steatosis. It occurs when the fat content of the hepatocytes increases. It is the initial and most common histologic response to excessive alcohol ingestion. At present the global incidence rate of fatty liver is still increasing due to growth of obesity, alcoholism and diabetes, affecting estimated 10-24% of worlds population. Fatty liver could be reversible in its early stage, But if left untreated it may lead to inflammation of liver and patient's liver may get permanantly damaged. So the early detection and treatment is crucial for control of the disease.

1.1 Imaging Modalities for fatty liver Diagnosis

The main diagnostic methods for fatty liver are Biopsy, B-mode ultrasound, CT and MRI. Liver biopsy, the diagnostic GOLD standard (as doctors say) is not well accepted by patients because it is invasive and it also has disadvantage that it poses a risk of cancer spreading if it cuts through a localized cancer area. MRI and CT-scan, on the other hand are quite expensive. Since B-scan ultrasound is non-invasive, inexpensive and easy to operate, it is the most commonly used modality to diagnose fatty liver. Doctors often make a presumptive diagnosis based on the B-mode ultrasound and then patient is diagnosed with a biopsy if case is found suspicious.

1.2 Objective

The purpose of this work was to design and implement Automatic Classifier(s), which, as accurately as possible, could classify fatty and normal livers using ultrasound images of the liver parenchyma, based on the traditional criteria used by the physicians in the diagnostic process by visual inspection of the ultrasound images.

1.3 Importance of the work

Diagnostic accuracy of fatty livers using only visual interpretation is currently estimated to be around 70-75%. Clinical diagnosis of B-mode ultrasound images of fatty liver, to a large extent, relies on the image quality and experience of technicians and doctors. Using the subjective judgment and non-quantitative description, doctors determine the incidence and the severity of fatty liver. However, the poor image quality, speckle noise, and use of different types of ultrasound imagers, and various physical conditions of patients obstruct a unified diagnostic standard. Therefore, a computer-aided liver ultrasound image quantitative analysis is necessary and will contribute to establishing a clinical objective fatty liver diagnosis method and standard, and improve clinical diagnostic accuracy, repeatability, and efficiency.

1.4 Outline of the work

In this work, after preprocessing of B-mode ultrasound images of normal and fatty liver, ROIs of 4 different sizes 16×16, 32×32, 48×48 and 64×64 were selected as per doctors' suggestions. 14 differnet features in total were extracted from each type of ROIs to distinguish between the two categories. Features were extracted according to the different characteristics of fatty liver and healthy liver. 12 Features were texture features and 2 other features were non-texture feature (i.e. independent of neighbourhood pixels). Some of the features were contrast, angular second moment, GLCM correlation etc.

After extraction, all the features were normalized to make the dataset balanced. Four different classifiers viz. Naive Classifier, K-NN Classifier, Multilayer Perceptron and Support vector Machine(SVM) were applied independently to classify the images into normal or fatty class. Classifiers were trained using the clinical ultrasound images of both fatty liver and normal liver. All classifiers were then tested with the available dataset using leave-one-out cross-validation method to classify normal and fatty livers.

All the classifiers achieved a high recognition rate. The diagnostic results are satisfactorily consistent with those made by doctors. This Methodology could be used for computer-aided diagnosis of fatty liver, and help doctors identify the fatty liver ultrasonic images rapidly, objectively and accurately.

1.5 Contemporary work

Many studies have been investigated on computer-aided diagnosis of liver diseases. Generally the diagnostic problem was transformed into pattern recognition of liver ultrasound image characteristics. Grayscale histogram, grayscale co-occurrence matrix, texture features and the neighboring point features were used to identify liver diseases.

Liu et al. computed the fractal parameters from ultrasound image textures and applied the second-order BP neural network to classify normal liver and fatty liver ultrasonic images. Wang et al. analyzed ultrasonic image texture features of normal liver and fatty liver, composed the optimal eigenvectors, and used the self-organizing feature mapping artificial neural network as the classification algorithm.

Zhu et al. analyzed ultrasound image texture of fatty liver and normal liver by a statistical run-length method and extracted feature parameters to provide a quantitative analysis for the fatty liver diagnosis.

Huang et al. utilized the wavelet transform for multi-resolution analysis of normal liver and fatty liver ultrasonic images, and used probabilistic neural network on the extracted feature parameters for fatty liver diagnosis.

Lupoor et al. selected the second-order statistical characteristics of grayscale co-occurrence matrix, average grayscale and grayscale change rate from liver ultrasound images, and then made use of K-means classifier to classify the normal liver, fatty liver, hepatitis and cirrhosis.

The above methods commonly applied the statistical analysis to the ultrasound image features on one selected region of interest (ROI) from liver ultrasound images, and took advantages of a certain pattern classifier for the diagnosis. These methods could obtain good diagnosis results. Garra et al. compared the performance of computer-based tissue characterization systems to that of human observer on several types of diffuse liver disease. The results suggested that the use of quantitative tissue characterization could significantly increase the usefulness of ultrasound for the evaluation of diffuse liver disease.

1.6 Organization of Remaining Text

In Chapter-2, the basics of ultrasound have been discussed. The main features of this chapters include The methodology of how ultrasound imaging works, different modes of ultrasound and Advantages and diadvantages of ultrasound.

In Chapter-3, the process of Image collection and format conversion has been discussed. The details of diffenent ROI selection can also be found here.

Chapter-4 contains the details about the features used in this work. Process of Feature extraction, feature normalization and application of various classifiers is also included in this chapter. However, Theory of each classifier is included in the appendix-B.

Details of Experiments and Results of the work can be found in Chapter-5. How the fatty liver can be differentiated with that of normal, is shown with the help of some scatter-diagrams using different features values as axis parameters.

Chapter-6 contains a brief discussion about the scope of future work in the area of normal and fatty liver classification.

Complete Source code of implementation of the whole process which was developed by the author himself can be found in this chapter. In his/her own interest, reader is encouraged to go through the code so that, how the whole process worked, could be understood clearly.

finally, A list of references and some web-url's who helped in this work is given in last chapter for interested readers.

Chapter 2

Basics of Ultrasonography

Medical imaging is one of the powerful tool for gaining insight into the normal and pathological processes that affect health. Now a days the various imaging modalities, such as microscopy,computer tomography, ultrasound ,medical resonance imaging (MRI) and PET etc, are used in medical decision making processes and in surgical actions. The use of ultrasonography as an imaging modality has become widely spread because of its ability to visualize main organs with no deleterious effects. Besides taking still images, real time moving image can also be obtained to guide biopsy procedures.

The basic idea of ultrasonic imaging is to send a fine beam of ultrasonic waves (which are infact high frequency broadband sound waves in the megahertz range) through the human tissues and then receive the characteristic echo reflections from the intemal body structures to form the ultrasound image. The different gray levels of this image represent the acoustic properties of the human tissues such as attenuation of acoustic waves, speed of sound, and acoustic impedance of the different body structures. All these factors contribute to the shape and intensity of the returned waves according to the underlying tissue properties and hence, this fact is the basis for the use of ultrasonography as an imaging technique. The main limitation for ultrasound is its inherent inability to visualize air-containing or bony structures. This limitation does not apply for most of the abdominal body structures as they are composed of soft tissues and blood vessels [l].

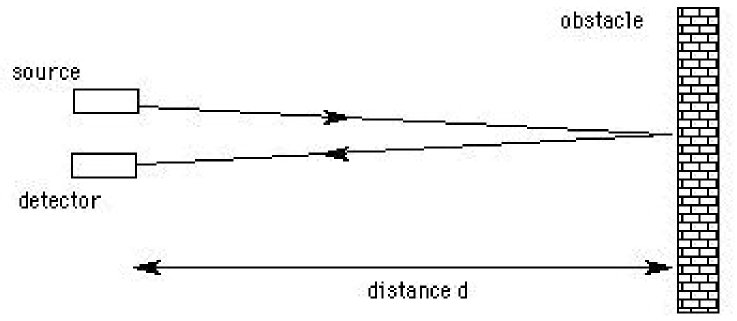
Ultrasonography (sonography) uses a probe containing one or more acoustic transducers to send pulses of sound into a material. Whenever a sound wave encounters a material with a different density (acoustical impedance), part of the sound wave is reflected back to the probe and is detected as an echo. The time it takes for the echo to travel back to the probe is measured and used to calculate the depth of the tissue interface causing the echo. The greater the difference between acoustic impedances, the larger the echo is. If the pulse hits gases or solids, the density difference is so great that most of the acoustic energy is reflected and it becomes impossible to see deeper. The frequencies used for medical imaging are generally in the range of 1 to 18 MHz. Higher frequencies have a correspondingly smaller wavelength, and can be used to make sonograms with smaller details. However, the attenuation of the sound wave is increased at higher frequencies, so in order to have better penetration of deeper tissues, a lower frequency (3-5 MHz) is used.

Ultrasonography is mainly done in following 3-steps :

1. Producing the sound waves :
   A sound wave is typically produced by a piezoelectric transducer encased in a probe. Strong, short electrical pulses from the ultrasound machine make the transducer ring at the desired frequency. The frequencies can be any-

where between 2 and 18 MHz. The sound is focused either by the shape of the transducer, a lens in front of the transducer, or a complex set of control pulses from the ultrasound scanner machine. This focusing produces an arc-shaped sound wave from the face of the transducer. The wave travels into the body and comes into focus at a desired depth.
Older technology transducers focus their beam with physical lenses. Newer technology transducers use phased array techniques to enable the sonographic machine to change the direction and depth of focus. Almost all piezoelectric transducers are made of ceramic.
Materials on the face of the transducer enable the sound to be transmitted efficiently into the body (usually seeming to be a rubbery coating, a form of impedance matching). In addition, a water-based gel is placed between the patient's skin and the probe.
The sound wave is partially reflected from the layers between different tissues. Specifically, sound is reflected anywhere there are density changes in the body: e.g. blood cells in blood plasma, small structures in organs, etc. Some of the reflections return to the transducer.

2. Receiving the echoes :
   The return of the sound wave to the transducer results in the same process that it took to send the sound wave, except in reverse. The return sound wave vibrates the transducer, the transducer turns the vibrations into electrical pulses that travel to the ultrasonic scanner where they are processed and transformed into a digital image.



3. Forming the image Using : $distance = speed \times time$

The sonographic scanner must determine three things from each received echo:

- How long it took the echo to be received from when the sound was transmitted.
- From this the focal length for the phased array is deduced, enabling a sharp image of that echo at that depth (this is not possible while producing a sound wave).
- How strong the echo was. It could be noted that sound wave is not a click, but a pulse with a specific carrier frequency. Moving objects change this frequency on reflection, so that it is only a matter of electronics to have simultaneous Doppler sonography.

Once the ultrasonic scanner determines these three things, it can locate which pixel in the image to light up and to what intensity and at what hue if frequency is processed (see redshift for a natural mapping to hue). Transforming the received signal into a digital image may be explained by using a blank spreadsheet as an analogy. We imagine our transducer is a long, flat transducer at the top of the sheet. We will send pulses down the 'columns' of our spreadsheet (A, B, C, etc.). We listen at each column for any return echoes. When we hear an echo, we note how long it took for the echo to return. The longer the wait, the deeper the row (1,2,3, etc.). The strength of the echo determines the brightness setting for that cell (white for a strong echo, black for a weak echo, and varying shades of grey for everything in between.) When all the echoes are recorded on the sheet, we have a greyscale image.

To generate a 2D-image, the ultrasonic beam is swept. A transducer may be swept mechanically by rotating or swinging. Or a 1D phased array transducer may be use to sweep the beam electronically. The received data is processed and used to construct the image. The image is then a 2D representation of the slice into the body.

3D images can be generated by acquiring a series of adjacent 2D images. Commonly a specialised probe that mechanically scans a conventional 2D-image transducer is used. However, since the mechanical scanning is slow, it is difficult to make 3D images of moving tissues. Recently, 2D phased array transducers that can sweep the beam in 3D have been developed. These can image faster and can even be used to make live 3D images of a beating heart.

Doppler ultrasonography is used to study blood flow and muscle motion. The different detected speeds are represented in color for ease of interpretation, for example leaky heart valves: the leak shows up as a flash of unique color. Colors may alternatively be used to represent the amplitudes of the received echoes.

Modes of UltraSonography Four different modes of ultrasound are used in medical imaging. These are:

- A-mode:
  A-mode is the simplest type of ultrasound. A single transducer scans a line through the body with the echoes plotted on screen as a function of depth. Therapeutic ultrasound aimed at a specific tumor or calculus is also A-mode, to allow for pinpoint accurate focus of the destructive wave energy.

- B-mode:
  In B-mode ultrasound, a linear array of transducers simultaneously scans a plane through the body that can be viewed as a two-dimensional image on screen. In this method the amplitude of each returning signal is not simply displayed on a graph or CRO screen. Instead the amplitude controls the brightness of the spot which represents this reflection, the b for brightness giving rise to the name B-scan. So a single pulse of ultrasound passing into a series of tissues will give rise to a series of spots, with the brightness of the spots corresponding to the amplitude of the reflection from different layers. The largest amplitude gives rise to a spot with the greatest brightness(strong white) . The smallest amplitude gives rise to a spot which is almost black. Intermediate amplitudes give various shades of gray.

- M-mode:
  M stands for motion. In m-mode a rapid sequence of B-mode scans whose images follow each other in sequence on screen enables doctors to see and measure range of motion, as the organ boundaries that produce reflections move relative to the probe.
- Doppler mode:
  This mode makes use of the Doppler effect in measuring and visualizing blood flow

2.1 Strengths and Weaknesses of Ultrasonography

Strengths

- It images muscle, soft tissue, and bone surfaces very well and is particularly useful for delineating the interfaces between solid and fluid-filled spaces.
- It renders ”live” images, where the operator can dynamically select the most useful section for diagnosing and documenting changes, often enabling rapid diagnoses. Live images also allow for ultrasound-guided biopsies or injections, which can be cumbersome with other imaging modalities.
- It shows the structure of organs
- It has no known long-term side effects and rarely causes any discomfort to the patient

- Equipment is widely available and comparatively flexible
- Small, easily carried scanners are available; examinations can be performed at the bedside
- Relatively inexpensive compared to other modalities, such as computed X-ray tomography, DEXA or MRI.
- Spatial resolution is better in high frequency ultrasound transducers than it is in most other imaging modalities.
- It is Non-invasive

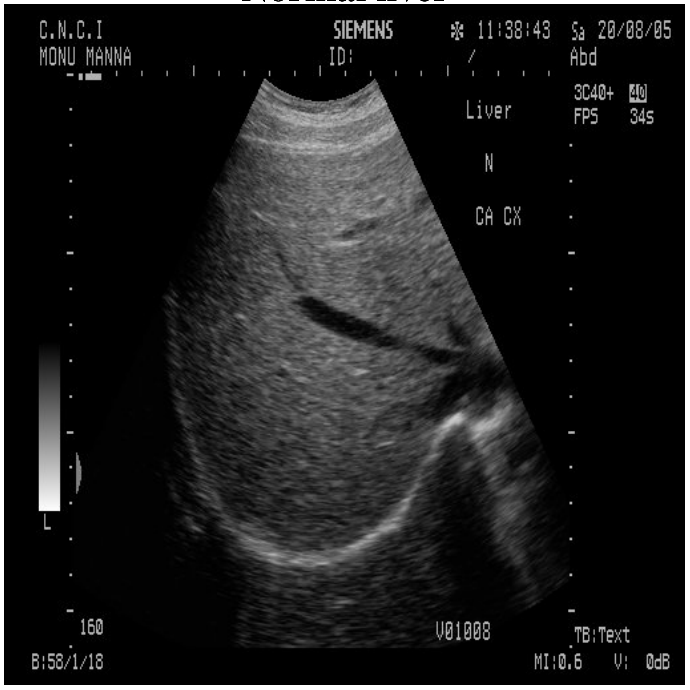
Weaknesses

- Sonographic devices have trouble penetrating bone. For example, sonography of the adult brain is very limited though improvements are being made in transcranial ultrasonography.
- Sonography performs very poorly when there is a gas between the transducer and the organ of interest, due to the extreme differences in acoustic impedance. For example, overlying gas in the gastrointestinal tract often makes ultrasound scanning of the pancreas difficult, and lung imaging is not possible (apart from demarcating pleural effusions).
- Even in the absence of bone or air, the depth penetration of ultrasound may be limited depending on the frequency of imaging. Consequently, there might be difficulties imaging structures deep in the body, especially in obese patients.
- The method is operator-dependent. A high level of skill and experience is needed to acquire good-quality images and make accurate diagnoses.
- There is no scout image as there is with CT and MR. Once an image has been acquired there is no exact way to tell which part of the body was imaged.

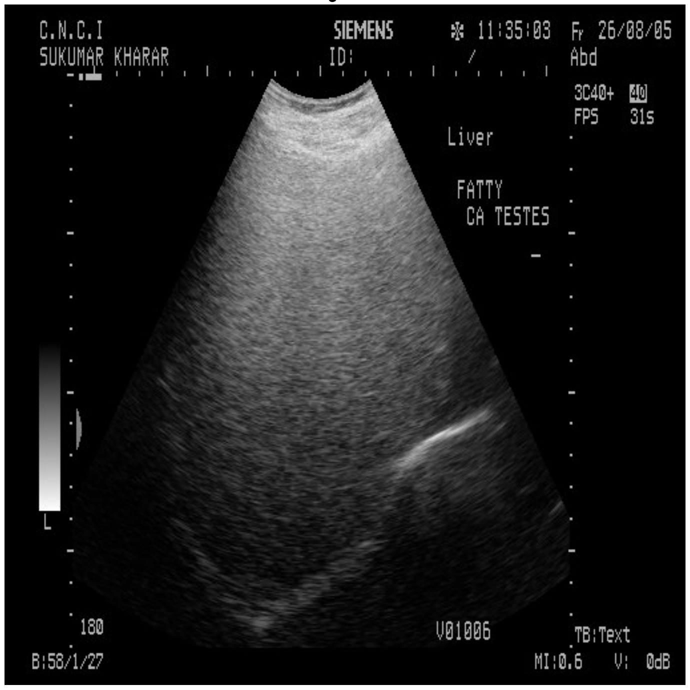
Risks and side-effects Ultrasonography is generally considered a ”safe” imaging modality. However slight detrimental effects have been occasionally observed. Diagnostic ultrasound studies of the fetus are generally considered to be safe during pregnancy. This diagnostic procedure should be performed only when there is a valid medical indication, and the lowest possible ultrasonic exposure setting should be used to gain the necessary diagnostic information under the as low as reasonably achievable or ALARA principle.

How the US-images of liver look like ?

—-Normal liver—-



—-Fatty liver—-



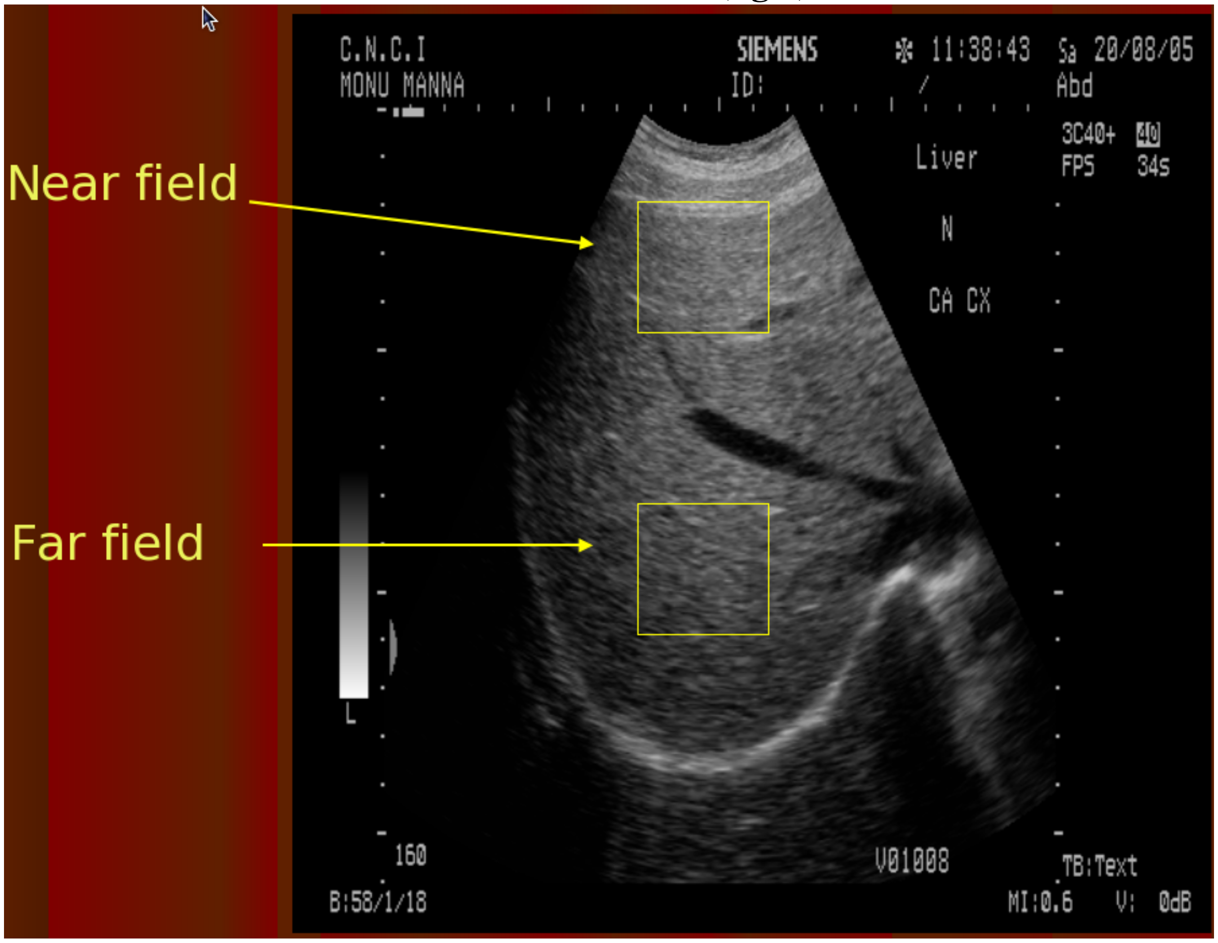
Chapter 3

Image Collection and ROI Selection

Outline

1. Image acquisition :
    - 71 Normal Liver Images
    - 29 Fatty Liver Images
2. Image format conversion :
    - DICOM to PGM Conversion
3. ROI selection (64x64)
    - One in near-field (fig.1)
    - Other in far-field (fig.1)
4. Sub-ROI selection :
    - 48×48 Sub-ROI
    - 32×32 Sub-ROI
    - 16×16 Sub-ROI

—-Near and Far field (fig.1)—-



1. Image acquision : A set of 100 images in total was acquired for this study from a local hospital. 71 cases of normal liver and 29 cases of fatty liver of varying degrees were confirmed by experienced ultrasound practitioners. The model of ultrasound device was SIEMENS, Bitmap (bmp) image format of 256-level gray-scale (8 bits grayscale image depth) was adopted and all images are of a size of 512×512. During image acquisition, the whole liver area was examined from different orientations and a single frame was saved which the operator thought would be most suitable for the fatty liver diagnosis. Except slight changes on the overall gain, the other parameters of the imaging device remained unchanged in the whole process.

2. Format conversion : All acquired image were initially in DICOM format. By a specially written computer program all images were converted to PGM format to ease the implementation process. Size was kept same as original

image i.e. 512x512 and gray scale range was 0-255.

3. ROI Selection (64×64) : To analyze ultrasound image characteristics quantitatively, two ROIs were selected as suggested by the doctors.

   - First ROI was selected in the near-field(at the bottom of subcutaneous fat - fig.1)
   - Other in the far-field (located at the top of the liver capsule - fig.1)

   ROI selection must avoid blood vessels, hepatic duct and other such structures. 64×64 was the maximum possible ROI size which satisfied these condition for all images.

4. Sub-ROI selection : Two ROIs (near-field and far-field) of size 64x64 were cropped manually using GNU-Image-Manipulation-Program from each of the 100 images. Then by a specially written computer program in C three sub-ROIs of sizes 16x16, 32x32 and 48x48 were cropped from center of each 64x64 ROI image. This was done to determine the relation between accuracy of the classifier and ROI size ? If smaller ROI would give same accuray as a larger one than the former should be preferred for applications since it would give rise to faster calculation.

Chapter 4

Feature Extraction, Normalization and Applying Classifiers

CO-OCCURENCE Matrix (aka GLCM) A co-occurrence matrix, also referred to as a co-occurrence distribution or GLCM, is defined over an image to be the distribution of co-occurring values at a given offset. Informally, GLCM is a tabulation of **How often** different combinations of gray levels occur in an image. Mathematically, a co-occurrence matrix C is defined over an nxm image I, parameterized by an offset x, y, as :

$$C_{(d,\theta)}(i,j) = \sum_{n}\sum_{n} \begin{cases} 1 & if\ I(m,n) = i\ and\ I(m+d, n+\theta) = j \\ 0 & otherwise \end{cases} \tag{4.1}$$

where d=distance, θ=direction, n=size of image, I is the image

GLCM texture considers the relation between two pixels at a time called the reference pixel and neighbourhood pixels.

Usual Rotations = 0, 45, 90, 135

Symmetric GLCM : Each pixel pair is counted twice, Once forward and once backward. for example, the pixel pair (m,n) will contribute to both of the (m,n)th and (n,m)th cell of GLCM.
If G is non-symmetric GLCM, Then one may calculate G_{sym} as follows :
$G_{sym} = G + G^T$

GLCM Normalization : This is the last step before texture features can be extracted. In this step each entry of GLCM contains a probability instead of count : $P_{ij} = \frac{G_{ij}}{\sum\limits_{i,j=0}^{N-1} G_{ij}}$ where N is the no. of gray levels

Some Properties of GLCM

1. GLCM is always a square matrix
2. Rows/Columns of GLCM represents quantization level of image
3. Diagonal elements represents pixel pairs with no grayscale difference $\rightarrow$High probabilities at diagonal $\Rightarrow$ **Low contrast** (most pixels are identical to neighbours) and vice verca
4. Subdiagonals elements represents no. of pixels with grayscale difference of 1.

5. The farther away from the diagonal, the greater the difference between pixel grayscales which implies **More Contrast**

4.1 Features

4.1.1 GLCM based features

Most of the features from this class are **Weighted Averages** of the normalized GLCM Probabilities. Following GLCM based features were extracted as input to the classifiers :

Contrast Group

Contrast(CON), Dissimilarity(DIS) and Homogeneity(HOM) belong to this group. Features in this groups use weights based on the distances from the GLCM diagonal.

1. Contrast (CON) : It is basically the sum of square variance, and is defined as follows :

$$CON = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left((i-j)^2 * P_{ij} \right) \tag{4.2}$$

   It is easy to interpret that as we move farther from the diagonal of GLCM, the difference of i and j gets larger, and the value of CON will be more as discussed earlier.

2. Dissimilarity (DIS) : dissimilarity is defined as follows :

$$DIS = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(|i-j| * P_{ij} \right)) \tag{4.3}$$

3. Homogeneity (HOM) : It is also known as **Inverse Difference Moment**. It is defined as follows :

$$HOM = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(\frac{P_{ij}}{1+(i-j)^2} \right) \tag{4.4}$$

Measures related to orderliness

: Features in this groups are Angular Second Moment(ASM), Energy, aximum Probability, Entropy(ENT) and Sum Entropy(SUM_ENT)

1. Angular Second Moment(ASM) : It is calculated as follows :

$$ASM = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (P_{ij} * P_{ij}) \tag{4.5}$$

2. Energy : This definition is taken from the classical mechanics :

$$Energy = sqrtASM \tag{4.6}$$

3. Maximum Probability (MAX) : MAX = Largest probability found in GLCM It is easy to see that a High value of MAX occurs if one pair of pixel-values

dominate all others

4. Entropy (ENT) : Entropy is defined as follows :

$$ENT = -\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\{P_{ij} * (log\ P_{ij})\} \tag{4.7}$$

Physically, Entropy is the amount of energy which is lost during a reaction. It cannot be used to do useful work.
Energy in this context id the opposite of Entropy which can be used to do useful work.

5. Sum Entropy (SUM_ENT) : Sum Entropy is calculated as follows :

$$P_{sum}(K) = \sum_i\sum_j P_{ij} for i+j = K and K = 2, 3, ..., 2N_G where N_G is the total no. of gra \tag{4.8}$$

Now the Sum Entropy SUM_ENT(d,θ) is calculated as follows :

$$SUM\_ENT(d,\theta) = \sum_{K=2}^{2N_G}\{P_{sum}(K) * log(P_{sum}(K))\} \tag{4.9}$$

Descriptive statistical measures of GLCM

GLCM Mean, GLCM Variance, GLCM Standard deviation and GLCM-Correlation falls under this category. Definition of all these is described as follows :

1. GLCM Mean :

$$\mu_{ref} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(i * P_{ij}) \tag{4.10}$$

$$\mu_{nbd} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(j * P_{ij}) \tag{4.11}$$

Clearly, for the Symmetric GLCM both values are identical

2. GLCM Variance :

$$\sigma_{ref}^2 = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}((i-\mu_{ref})^2 * P_{ij}) \tag{4.12}$$

$$Similarly, \quad \sigma_{nbd}^2 = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}((j-\mu_{ref})^2 * P_{ij}) \tag{4.13}$$

3. GLCM Standard deviation :

$$\sigma_{ref} = \sqrt{\sigma_{ref}^2} \tag{4.14}$$

$$Similarly, \quad \sigma_{nbd} = \sqrt{\sigma_{nbd}^2} \tag{4.15}$$

4. GLCM Correlation :

$$GLCMCORR = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left\{P_{ij} * \left(\frac{(i-\mu_{ref})(j-\mu_{nbd})}{\sigma_{ref} * \sigma_{nbd}}\right)\right\} \tag{4.16}$$

   It is a measure of gray-tone linear dependencies in the image.

Other Features

Two other features viz. Near Field Light Spot Density(NFLSD) and Near-Far Field Grayscale Ratio(NFFGR) were also extracted. These are basically non-texture features which don't use the relationship between the neighbouring pixels. These are defined as follows :

1. Near-field Light-Spot Density (NFLSD) : Ultrasonic images contain a large number of light-spots, which represent high-frequency signal within the images. A high-pass filter was applied to the original ROI image in the near-field, denoted by $I_1(x, y)$, where (x, y) are pixel coordinates and the density of light-spots in the area was measured in the following steps:

   (a) A Laplace of Gaussian LoG operator with the following mask was applied to retain the high frequency component denoted by $f(x, y)$ from

$I_1(x, y)$

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

(b) Thresholding : Thresholding was applied with a threshold $T(= 200)$ to the filtered image $f(x, y)$ and the resultant image $g(x, y)$ was calculated as follows :

$$g(x,y) = \begin{cases} 255 & f(x,y) \geq T \\ 0 & f(x,y) < 0 \end{cases}$$

(c) The connected areas were identified with $8 - nbd$ connectivity in the binary image $g(x, y)$ and the number of the connected areas was extracted as a measure of light-spot density within the ROI.

2. Near-Far field grayscale ratio : NFFGR was calculated by ratio of summed grayscale in the near-field ROI to far-field ROI, shown as follows:

$$NFFGR = \frac{\sum\sum I_{nf}(x,y)}{\sum\sum I_{ff}(x,y)} \tag{4.17}$$

where the summations are over all pixel of Near-field and Far-field ROI images, respectively.

Feature Extraction Each of the above 14 features were extracted from all 100 US-images using a specially written computer program in C. This was done for all 4 ROIs of size 16, 32, 48 and 64.
All these features were stored in 4 text files each corresponding to one ROI-Size. Each file consisted of 100x14 feature matrix plus one addition column to store the class information of corresponding feature vector. Values of one feature with another varied greatly So it was necessary to balace the data so that no feature gets advantage over another when classification algorithm would be applied on them.

4.2 Feature Normalization

Feature normalizaion was done as follows :

$$y_i^j = \frac{x_i^j - min^j}{max^j - min^j} \tag{4.18}$$

where i is the index of current feature vector, j is current feature, y_i^j is the new value of point x_i^j and min^j and max^j are minimum and maximum values of that feature.
Clearly after this normalization all feature value of each point lie between the range $[0, 1]$.
Again all feauture vectors were stored in 4 different text files (for each ROI-Size). Each file consits of a matrix of size 100x15, 14 columns were feature values and one column represented class label.

4.3 Generation of all Possible Feature Subsets

In case of Naive Bayes Classifier and SVM it seemed possible to apply the classifiers on all possible subsets of set of 14-feature-Set due to less complex nature of classifier. By using a computer program written in Perl, all possible subsets $2^{14} - 1 = 16383$ were generated with each subset having class label as first column. This was done for all 4 different ROI-sizes.

Note :
In case of MLP and K-NN, however, due to computational complexity it was infeasible to apply the classifiers on all subsets for each ROI-Size. Hence based on all results obtained ROI-Size 64x64 was selected on which these classifiers would be applied.

Source code for all the classifier except SVM were developed in C Language by the author himself. However, for SVM Classifier the libsvm-2.89 was obtained from “http://www.csie.ntu.edu.tw/ cjlin/libsvm/” and used as a black box.

4.4 Significance of the extracted features

- Texture features : Texture feature used here were proposed by Harlick et al in 1979 and till now these are widely used in many image processing application.

- NFLSD : When the fat content in the liver increases, more of the ultrasonic pulse is absorbed and scattered. So near-field light-spots are brighter and finer for the liver parenchyma and grayscale in the far-field attenuates. Near-field light-spot density reflects this kind of grayscale change for near-filed and far-field.

- NFFGR : Fat in the liver absorbs and scatters more energy of ultrasound waves, thereby attenuating the ultrasound echo signal. In ultrasound images of fatty liver, grayscale in the far-field is lower than that in the near-field. The higher the liver fat content is, the greater attenuation. Because ultrasound devices are always pre-adjusted on the basis of healthy livers before examination, grayscale in the near-field and far-field is generally uniform for healthy livers.

For the theory of the classifiers please refer to the appendix.

Chapter 5

Experiments and Results

5.1 Outline

After selecting the features from the images for various ROIs, all the features were normalized and datasets were generated as was discussed in previous chapter.

Due to the computational complexity, At first the Input Domain of classifiers was reduced to a single ROI-Size (64×64).

Classifiers were applied to as many subset sizes as possible. Naive-Bayes and SVM classifier were applied to all possible subsets. However, due to computational complexity, MLP and K-NN classifiers were applied only to all possible subsets of size 1, 2, 3, 12, 13 and 14. Additionally, K-NN was also applied to all subsets of size 11.

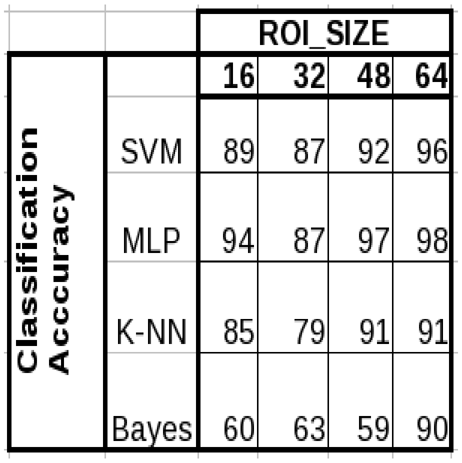
All the classifiers used **Leave-One-Out Cross-Validation** . In which the following methodology was used : Let us suppose that whole dataset S contains N points (feature vectors). In this method the whole dataset S was partitioned into to sets A containing $N - 1$ points and B containing 1 points. Now, the classifiers were applied with A as training set and B as test set. This process was repeated N times, with each time a single different point in B. The average accuracy of all N times was taken as the measure of accuracy of each classifier.

For K-NN classifier the results were collected for K=1,3,5,7 and 9.
In case of MLP the no. of hidden layer were taken 2, no. of neurons in each hidden

layer were taken 10 and the learning rate of 0.25 was taken after many experiments with different values.

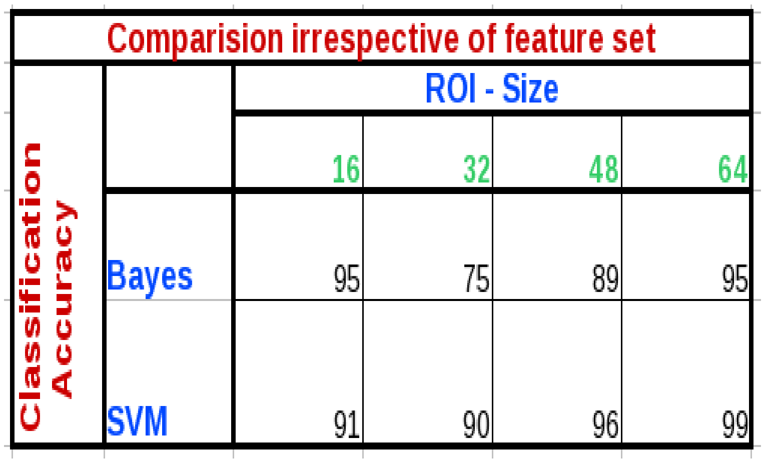
5.2 Optimal ROI-Size Selection

All 14 features were extracted from each of the 4 ROIs (viz. 16×16, 32×32, 48×48 and 64×64) and classifiers were applied using a single set containing all features. Out of all 4 ROIs 64×64 was found best for which each classifier showed best accuracy. Accuracy of all classifiers w.r.t. ROI-Size is presented in fig-1.
Some other results were also taken, and based on all results ROI-Size 64×64 was found best. Further analysis was only done with this ROI-size only.



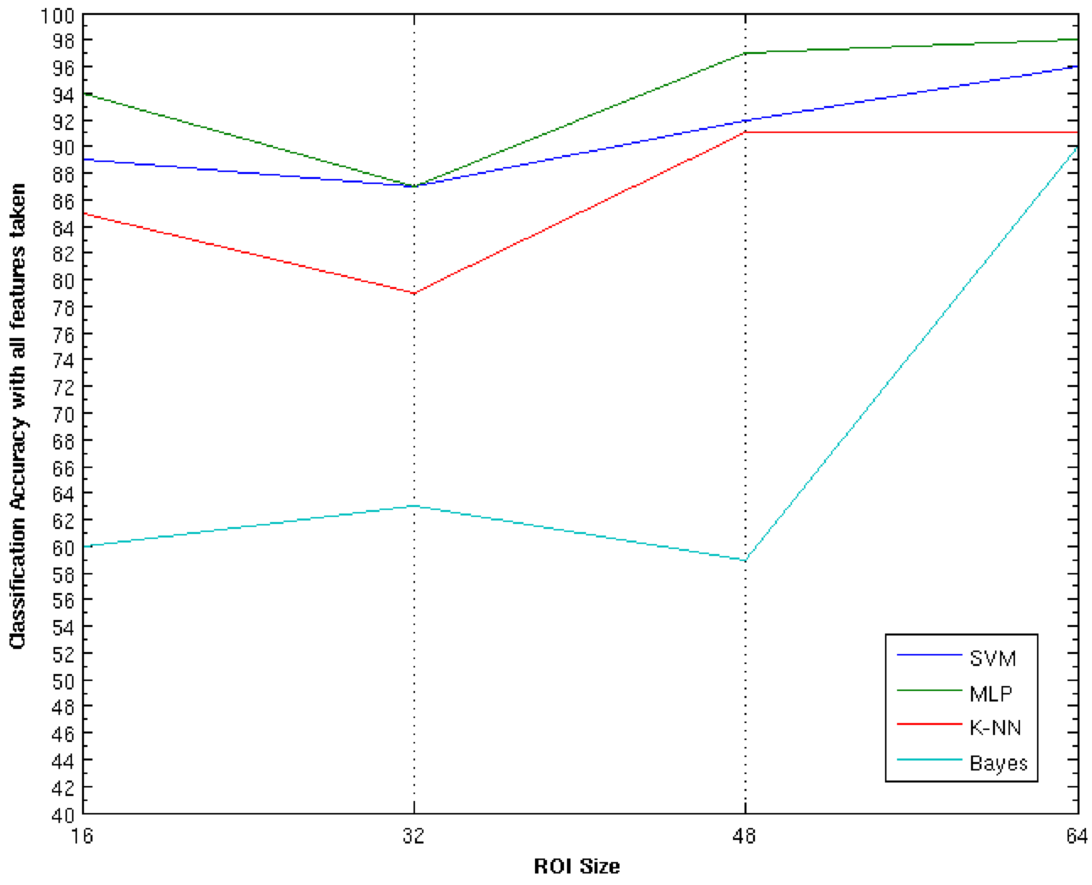
		ROI_SIZE			
		16	32	48	64
Classification Acccuracy	SVM	89	87	92	96
	MLP	94	87	97	98
	K-NN	85	79	91	91
	Bayes	60	63	59	90

(a) Feature set= $\{F_1, F_2, ..., F_{14}\}$



Comparision irrespective of feature set					
		ROI - Size			
		16	32	48	64
Classification Accuracy	Bayes	95	75	89	95
	SVM	91	90	96	99

(b) Feature set= $\{all\ possible\ 2^{14}\ sets\}$ (fig-2)



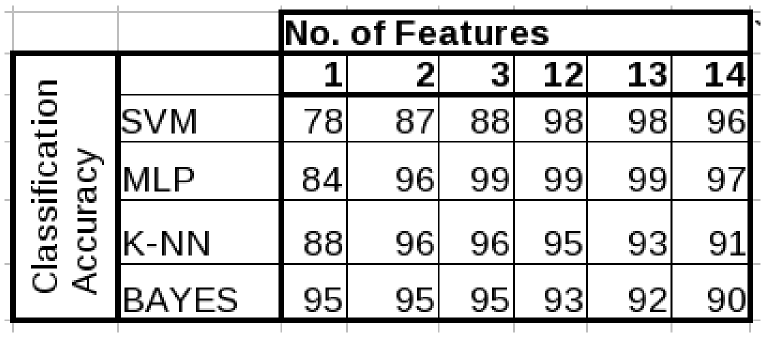
(c) Plot for table 3.1(a) (fig-3)

Figure 5.1: ROI-Size Vs Accuracy

5.3 Accuracy w.r.t. Number of features

In this part of the work, classification Accuracy was calculated using all possible subsets of few fixed sizes. Due to computational complexity (of K-NN and MLP), it was decided to consider all subsets of size 1, 2, 3, 12, 13 and 14. Total subset hence derived were ${}^{14}C_1 +^{14} C_2 +^{14} C_3 +^{14} C_{12} +^{14} C_{13} +^{14} C_{14} = 575$. Classification Accuracy for all 4 classifiers was calculated for each of these 575 sets. Best Results of each classifiers are presented in the fig 3.2 (table and plot) :

It can be seen from figures that subsets of size-13 gives best accuracy for each classifier. In all cases except Naive Bayes Classifier, Accuracy increases with number of features. Mlp obtains 99% accuracy with with feature subset size 3, 12 and 13. Since subsets of size 13 are in total 91, whereas that of size-14 is only one, accuracy for that single set is outperformed by some subset of size 13.



		No. of Features					
Classification Accuracy		1	2	3	12	13	14
	SVM	78	87	88	98	98	96
	MLP	84	96	99	99	99	97
	K-NN	88	96	96	95	93	91
	BAYES	95	95	95	93	92	90

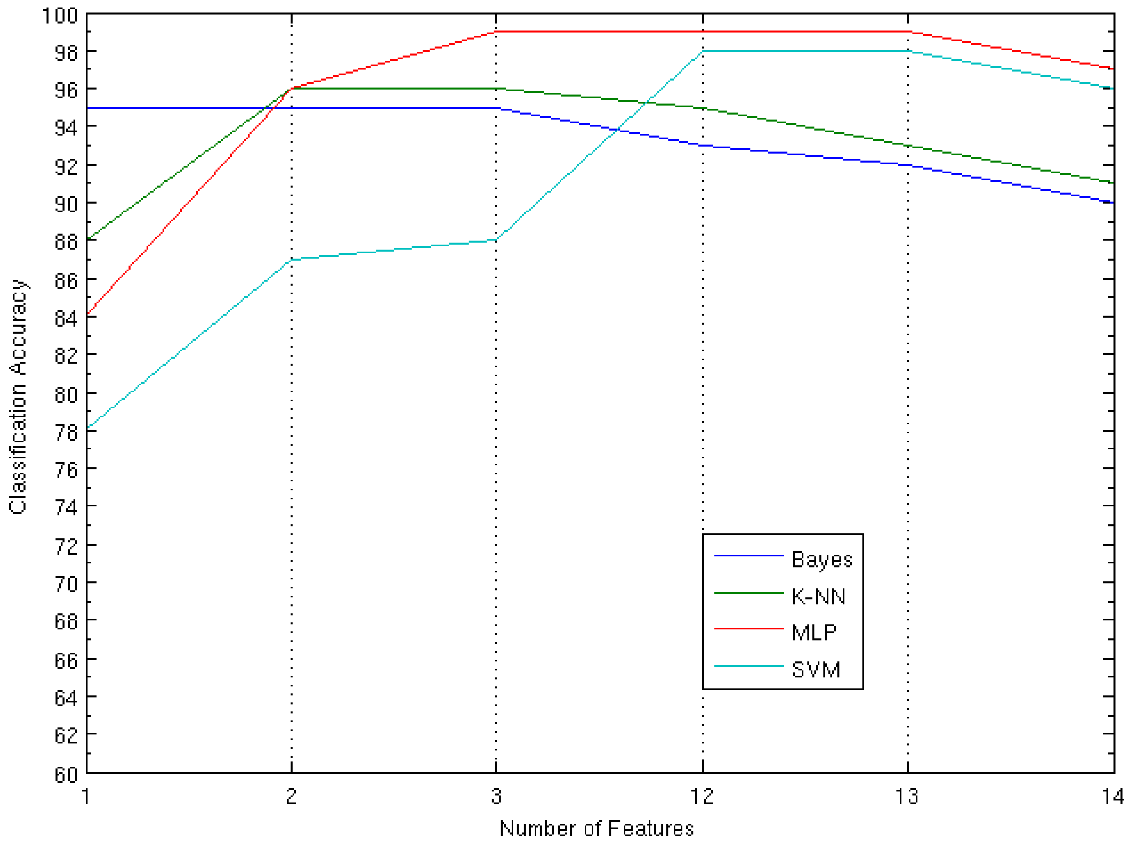


Figure 5.2: Accuracy Vs No. of Features

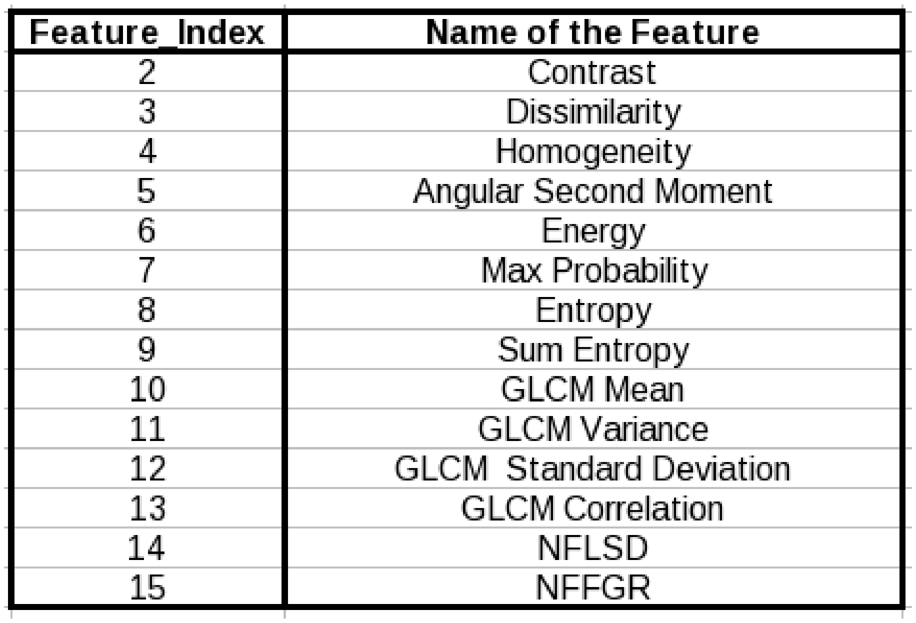
5.3.1 Result for Feature set = All possible 2-feature subsets$({}^{14}C_2)$

Finally, we want to plot the features of both types of livers and want to visualize that how both classes differs w.r.t. feature value. Visualization becomes difficult if more than 2 features are considered.

2 Best features were seleted for each classifier, using the Best accuracy of each classifier over all ${}^{14}C_2 = 91$ subsets of size 2. Then using the feature values for both the classes(viz. normal and fatty), Scatter diagram were created for each Classifier. Best Accuracy for each classifier was as follows :

- Naive-Bayes : features = {GLCM Variance, GLCM Correlation}, Accuracy=95%
- K-NN : features = { Contrast, Homogeneity}, Accuracy=96%
- MLP : features = { Contrast, Dissimilarity}, Accuracy=96%
- SVM : features = { Contrast, Homogeneity}, Accuracy=87%

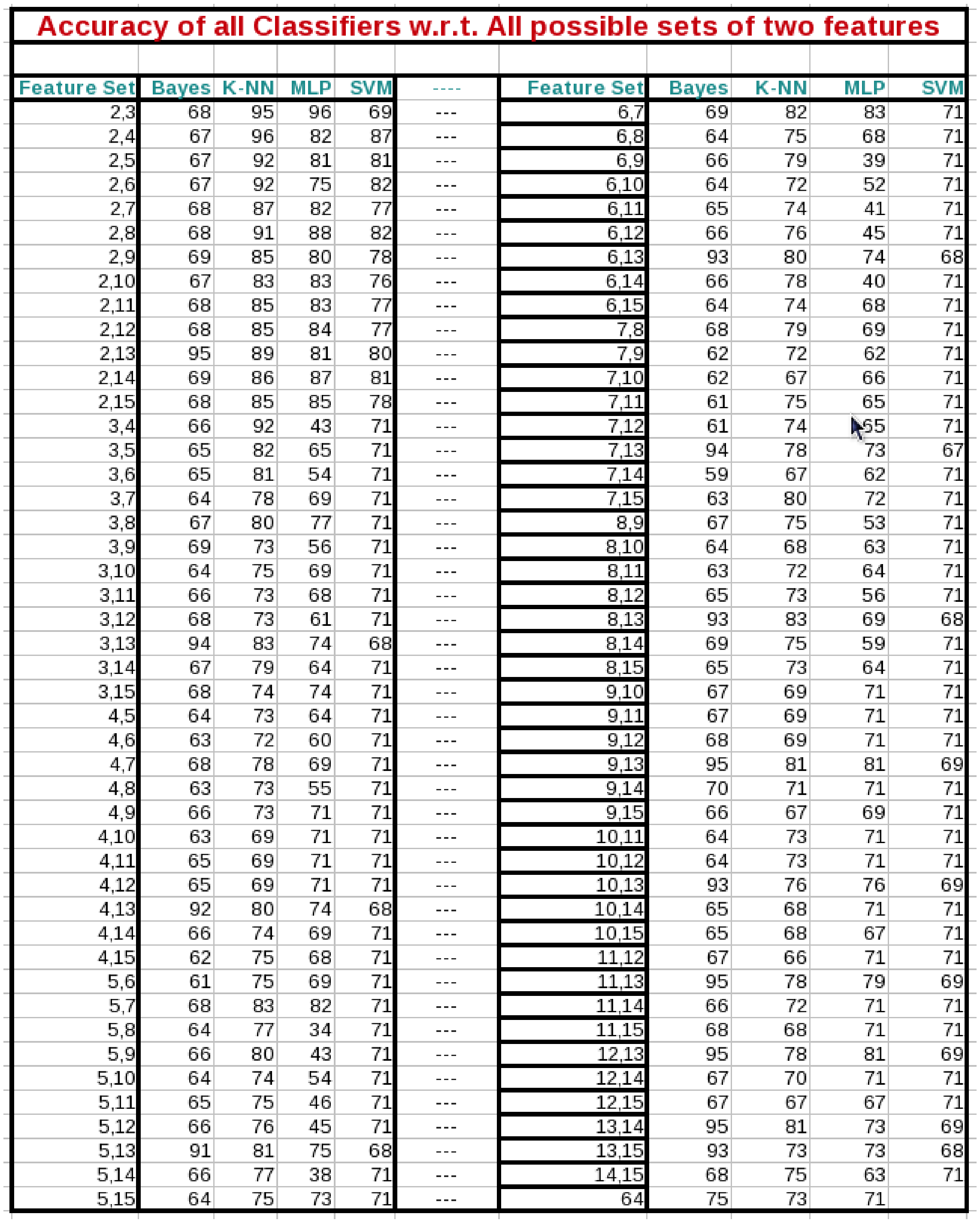
Best two features for K-NN and SVM were same.
Following Scatter diagrams for each classifiers are consturcted using corresponding best two feature values from each point(i.e. feature vector) of all 100 points, which was our original dataset.
It can be easily visulized using these scatter diagrams that how feature values of normal and fatty liver differ with each other, and how the classifiers achieved a high recognition rate.



Feature_Index	Name of the Feature
2	Contrast
3	Dissimilarity
4	Homogeneity
5	Angular Second Moment
6	Energy
7	Max Probability
8	Entropy
9	Sum Entropy
10	GLCM Mean
11	GLCM Variance
12	GLCM Standard Deviation
13	GLCM Correlation
14	NFLSD
15	NFFGR

Figure 5.3: Feature Indices and Names

Notation : Indices instead of feature names, as shown in table 3.3, are used to denote corresponding features in table 3.4.



Accuracy of all Classifiers w.r.t. All possible sets of two features										
Feature Set	Bayes	K-NN	MLP	SVM	----	Feature Set	Bayes	K-NN	MLP	SVM
2,3	68	95	96	69	---	6,7	69	82	83	71
2,4	67	96	82	87	---	6,8	64	75	68	71
2,5	67	92	81	81	---	6,9	66	79	39	71
2,6	67	92	75	82	---	6,10	64	72	52	71
2,7	68	87	82	77	---	6,11	65	74	41	71
2,8	68	91	88	82	---	6,12	66	76	45	71
2,9	69	85	80	78	---	6,13	93	80	74	68
2,10	67	83	83	76	---	6,14	66	78	40	71
2,11	68	85	83	77	---	6,15	64	74	68	71
2,12	68	85	84	77	---	7,8	68	79	69	71
2,13	95	89	81	80	---	7,9	62	72	62	71
2,14	69	86	87	81	---	7,10	62	67	66	71
2,15	68	85	85	78	---	7,11	61	75	65	71
3,4	66	92	43	71	---	7,12	61	74	65	71
3,5	65	82	65	71	---	7,13	94	78	73	67
3,6	65	81	54	71	---	7,14	59	67	62	71
3,7	64	78	69	71	---	7,15	63	80	72	71
3,8	67	80	77	71	---	8,9	67	75	53	71
3,9	69	73	56	71	---	8,10	64	68	63	71
3,10	64	75	69	71	---	8,11	63	72	64	71
3,11	66	73	68	71	---	8,12	65	73	56	71
3,12	68	73	61	71	---	8,13	93	83	69	68
3,13	94	83	74	68	---	8,14	69	75	59	71
3,14	67	79	64	71	---	8,15	65	73	64	71
3,15	68	74	74	71	---	9,10	67	69	71	71
4,5	64	73	64	71	---	9,11	67	69	71	71
4,6	63	72	60	71	---	9,12	68	69	71	71
4,7	68	78	69	71	---	9,13	95	81	81	69
4,8	63	73	55	71	---	9,14	70	71	71	71
4,9	66	73	71	71	---	9,15	66	67	69	71
4,10	63	69	71	71	---	10,11	64	73	71	71
4,11	65	69	71	71	---	10,12	64	73	71	71
4,12	65	69	71	71	---	10,13	93	76	76	69
4,13	92	80	74	68	---	10,14	65	68	71	71
4,14	66	74	69	71	---	10,15	65	68	67	71
4,15	62	75	68	71	---	11,12	67	66	71	71
5,6	61	75	69	71	---	11,13	95	78	79	69
5,7	68	83	82	71	---	11,14	66	72	71	71
5,8	64	77	34	71	---	11,15	68	68	71	71
5,9	66	80	43	71	---	12,13	95	78	81	69
5,10	64	74	54	71	---	12,14	67	70	71	71
5,11	65	75	46	71	---	12,15	67	67	67	71
5,12	66	76	45	71	---	13,14	95	81	73	69
5,13	91	81	75	68	---	13,15	93	73	73	68
5,14	66	77	38	71	---	14,15	68	75	63	71
5,15	64	75	73	71	---	64	75	73	71	

Figure 5.4: Accuracy-Table for all 2-subsets

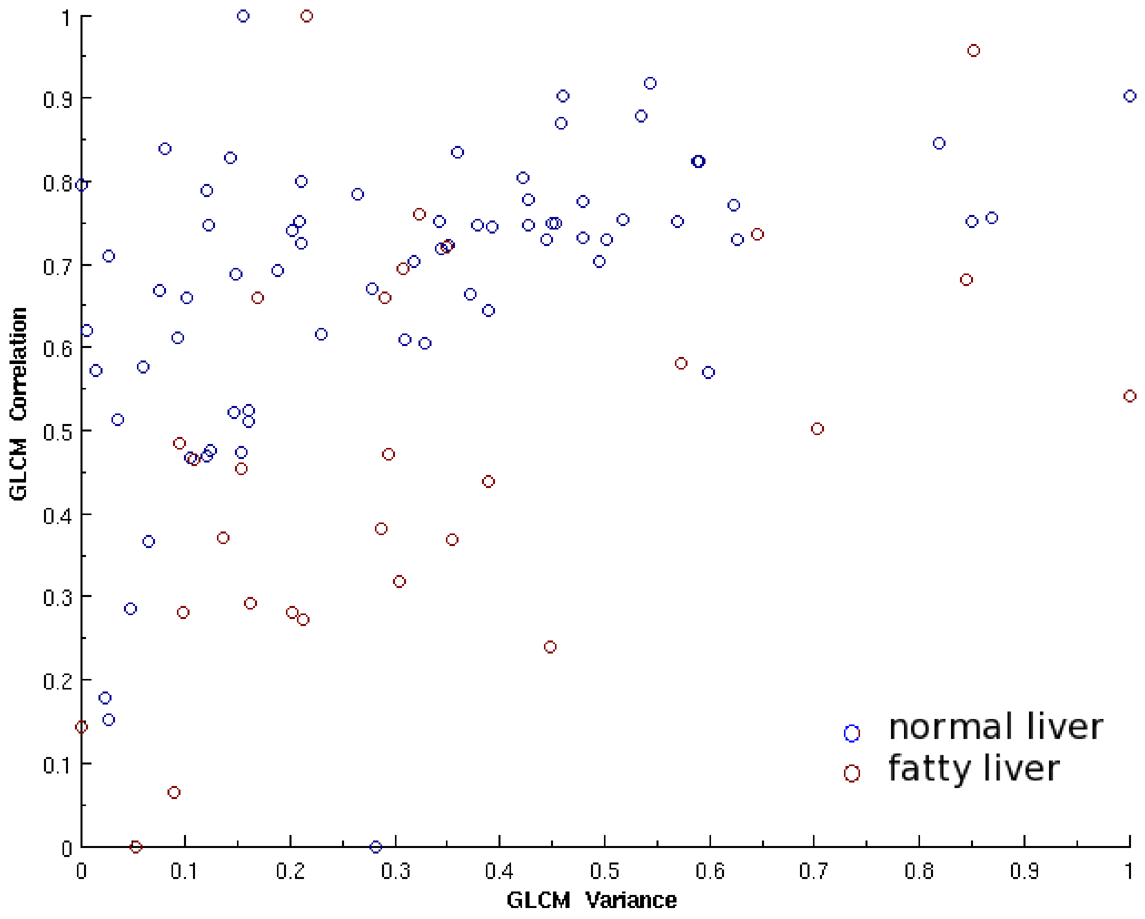


Figure 5.5: Scatter Diagrams for 2-Best Features [GLCM Variance and GLCM Correlation] for Naive-Bayes Classifier

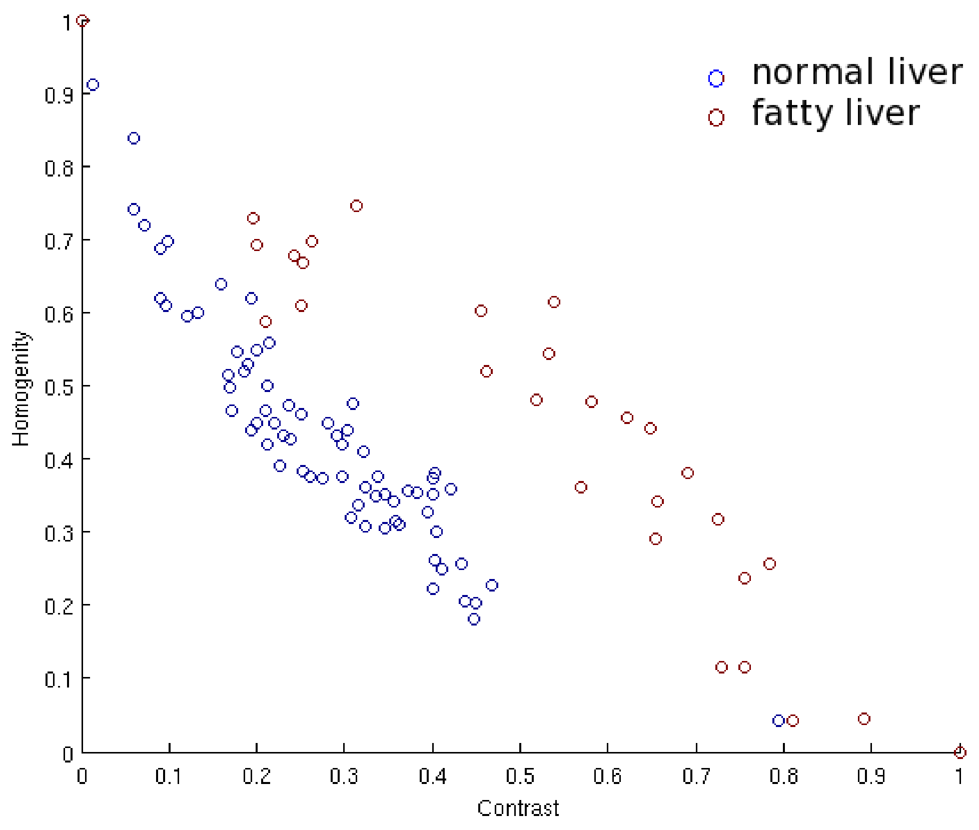


Figure 5.6: Scatter Diagrams for 2-Best Features [Contrast and Homogeneity] for K-NN and SVM

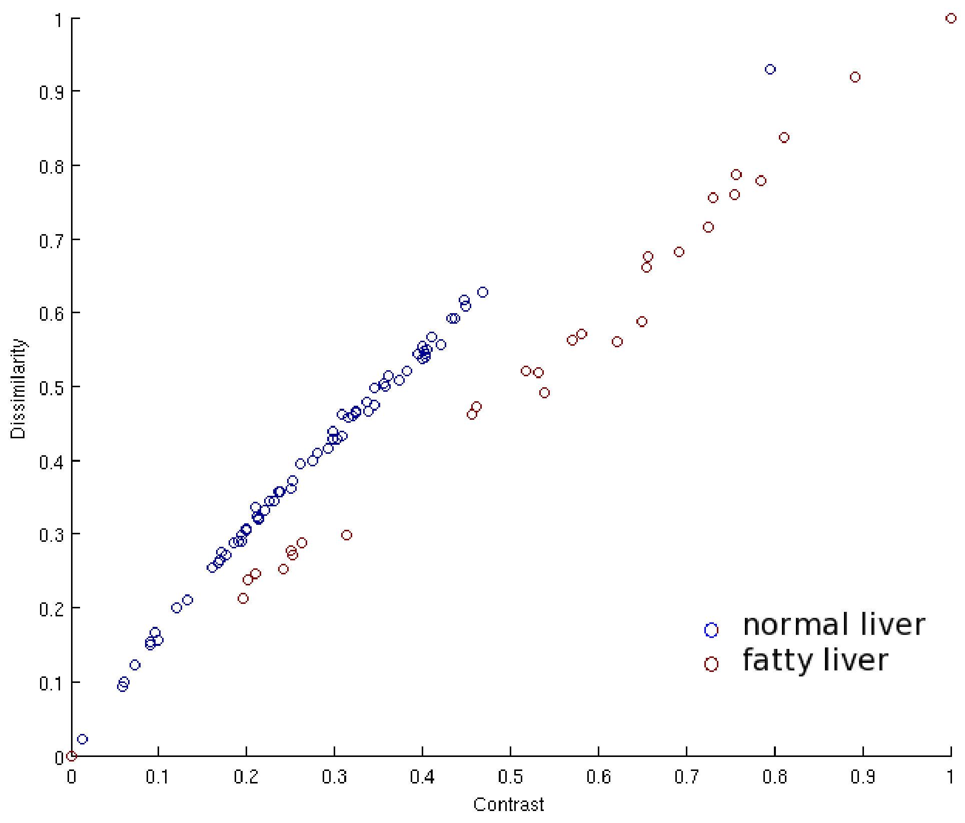


Figure 5.7: Scatter Diagrams for 2-Best Features [Contrast and Dissimilarity] for MLP

One may ask, that If According to 3 classifiers(viz. K-NN, MLP and SVM), **Contrast** is one of the best two features for classification, then why the performance of Contrast is not good as a single feature?

However, It is apprent from the Scatter Diagrams that if we take projection of all points on x-axis, (which is equivalent to using only Contrast for classification), the overlapping between points of two classes is quite more than the case when two features are used, and the seperating boundary is more difficult to find.

Chapter 6

Discussion and Scope of Future Work

Although, All the classifiers achieved a high recognition rate, yet there is much still to be explored in this field. Next step could be the quantification of the severity of disease, so that besides getting an idea of only the class of the image, doctors could also know that if the liver is fatty, what is the level of fat content ?

Obtained Results are highly dependent on the selection of ROI area. With different ROIs completely different results are obtained. Future work may be done towards automating the process of ROI-Selection.

In this work only LEAVE-ONE-OUT Cross-Validation was used to test the accuracy of the classifier, future work may go beyond that by using LEAVE-K-OUT-OF-N technique.

Appendix A

Source Code [Written in C, Perl and Linux Shell Scripting]

A.1 header.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<limits.h>
#include<unistd.h>




#ifndef NFLSD_THRESHOLD
#define NFLSD_THRESHOLD 200
#endif

#ifndef MAX_FILENAME_LENGTH
#define MAX_FILENAME_LENGTH 255
#endif

/*-------------Gaussian filter Parameters-------------*/
#ifndef FILTERING
#define FILTERING  0   //Gaussian filtering (1 for yes)
#endif

#ifndef FILTER_WITH_PADDING
#define FILTER_WITH_PADDING 0 // 1=Yes, 0=No(simple filter)
#endif

#ifndef ZERO_PADDING
#define ZERO_PADDING     0     // 1=Yes, 0=No(boundary element copied)
#endif

#ifndef FILTER_SIZE
#define FILTER_SIZE 9 //Gaussian filter size
#endif

#ifndef SIGMA
#define SIGMA 1.0       //Used in finding value of gaussian filter
#endif
/*----------------------------------------------------*/
```

```
#ifndef EXP
#define EXP 2.7182818
#endif

#ifndef PI
#define PI  3.1415926
#endif

#define W_SPACE(x) (x==' '||x=='\n'||x=='\t'||x=='\r'||x=='\b')


extern struct image {
        char type[10];
        char *inStr; // %d for P2, and %c for P5
        int width;
        int height;
        int pmaxval;  // max_value by header, usually=255

        int pmin;      // to optimize GLCM-memory requirement
        int pmax;      // min & max pixel values present in image

        unsigned char **pixel; //2-D array to store pixels of image
}img;

extern struct features {

        //Class_label //1

        // Contrast group
        float con; // 2
        float dis; // 3
        float hom; // 4

        // Orderliness or Similarity group
        float ang_sm; // 5
        float enrg;    // 6
        float max_prob;// 7
        float entr;    // 8
        float sum_entr;// 9

        // Descriptive Statistics based features
        float gLCM_mean;// 10
        float gLCM_var;// 11
        float gLCM_std_dev;// 12
        float gLCM_corr;// 13

        // Other features
        int nFLSD;    // 14 //     Near field light spot density
        float nFFGR; // 15 // Near-Far field gray scale ratio

}cur_feat;

extern struct gLCM {
        float **co_ocr;
        int start_idx;  // starting index = pmin
        int end_idx;     // ending index = pmax
        int size;

}cur_coc;

extern int high_pass_filter[5][5];


// Prototype of functions used
// checks validity and reads input data into memory
void isValidPGM(FILE *, struct image *, char *);
int *readPGM(FILE *, struct image *);
FILE* removeComment(FILE *);

// Preprocessing
void  gaussian_filtering(struct image * );
void create_filter(float ***);
void filter_without_padding(float **, struct image *);
void filter_with_padding(unsigned char ***, float **, struct image *);
  void create_padded_matrix(unsigned char ***, struct image *);
  void convolution(unsigned char **, float **, struct image *);

// GLCM : determine size, calculate , and normalization of GLCM-matrix
void calc_pmin_pmax(struct image *);
void calculateGLCM (struct gLCM *, struct image *, int, int, int);
void normalizeGLCM (struct gLCM *);

// Feature extraction
void calc_GLCM_Features(struct gLCM *, struct features *);
void calc_Enrg_MaxProb_GLCMMean(struct gLCM *, struct features *);
void calc_Stddev_GLCMCorr(struct gLCM *, struct features *);
void store_features(FILE *, struct features *, int);

void calc_NFLSD(struct image *, struct features *);
int check_nbrs(int **, int*, int*, int*, int*, int*, int*, int*, int*, int, int, int, int);
int func (int **, int , int , int , int , int , int , int , int , int , int , int , int , int *);
void assign_nbrs( int **, int , int , int , int , int , int , int , int , int , int , int );

void calc_NFFGR(struct image *, struct image *, struct features *);
//

// memory and file
void check_memory(void *, char *);
void check_file(FILE *, char *);
```

A.2 makefile

```
#################################################################

               # ———> Makefile <———#

#################################################################

OBJECTS = mainreadextract.c chkasgnNbrs.c processpgm.c glcmfeatures.c GLCMcalcandnormalize.c
nffgrnflsd.c gaussianfiltering.c misc.c

A :
        make ExtFeat Normmaxmin > /dev/null
B :
        make A Bayes >/dev/null
K :
        make A Knn      >/dev/null
M :
        make A Mlp  >/dev/null
S :
        make A          >/dev/null

# Feature Extraction #
ExtFeat : $(OBJECTS) header.h
        cc $(OBJECTS) -o ExtFeat -lm -g

# Feature normalization #
Normmaxmin : normalizefeaturesmaxmin.c header.h misc.c
        cc normalizefeaturesmaxmin.c misc.c -o Normmaxmin -lm -g
Normmeanstd : normalizefeaturesmeanstd.c header.h misc.c
        cc normalizefeaturesmeanstd.c misc.c -o Normmeanstd -lm -g

# Bayes classfier #
Bayes :  bayes.c header.h misc.c
        cc bayes.c misc.c -o Bayes -lm -g

# Knn classfier #
Knn : knn.c header.h misc.c
        cc knn.c misc.c -o Knn -lm -g

# MLP classfier #
Mlp : mlp.c header.h misc.c
        cc mlp.c misc.c -o Mlp -lm -g

# Image crop
Crop : crop.c header.h misc.c
        cc crop.c processpgm.c misc.c -o Crop -g

## Delete Executables #
cleanx :
        rm -f ExtFeat
        rm -f Normmaxmin Normmeanstd
        rm -f Bayes
        rm -f Knn
        rm -f Mlp
#################################################################
```

A.3 Bayes.sh

```
#!/bin/bash
                # Does everything for Bayes classifier #


echo ; echo "******************************   WARNING   *******************************"; echo
echo -e "Header file Must be modified accordingly before using this Script\n if the dataset is to be generated again !!!!!!"; e
echo "*************************************************************************"; echo


nor_feat_file="/tmp/$$_Nor_featr.txt"
fty_feat_file="/tmp/$$_Fty_featr.txt"
echo -n "Enter output directory to store all datasets: "
read odir
echo -n "Enter result directory for Bayes-Classifier : "
read resdir

echo
echo "Generate all dataset again : y|n"
read ques
if [ $ques = "y" -o $ques = "Y" ]
then
                echo "Creating required executables.."
        make A >& /dev/null
                echo "Success"; echo

                echo "Extracting Features.."
        ./nor.sh ${nor_feat_file}
        ./fty.sh ${fty_feat_file}
                echo "Success"; echo

                echo "Feature Normalization (using max-min)"
        ./Norm_max_min ${nor_feat_file} 71 14 /tmp/$$_t_nor
        ./Norm_max_min ${fty_feat_file} 29 14 /tmp/$$_t_fty
                echo "Success"; echo

                echo "Generating all datasets : please wait.."
        cat /tmp/$$_t_nor /tmp/$$_t_fty >| /tmp/$$_Dataset
        ./gen_all_datasets.sh ${odir} /tmp/$$_Dataset
                echo "Success"; echo
fi

        echo "Applying Classifer : please wait.."; echo
make Bayes >& /dev/null
mplayer /usr/share/sounds/pop.wav >& /dev/null

./loocv_bayes.sh ${odir} ${resdir}

        echo "Removing files.."
rm -f i/tmp/$$_t_nor /tmp/$$_t_fty ${nor_feat_file} ${fty_feat_file} /tmp/$$_Dataset
#make cleanx
        echo "Success"; echo
echo
echo "Done--Script finished !!"
echo

mplayer /usr/share/sounds/purple/alert.wav >& /dev/null
```

A.4 loocvbayes.sh

```
#!/bin/sh
##############################################################################
# Execute Bayes-Classifier for all files in $ddir directory

        # LOOCV Model is inbuilt in "bayes executable" so this script doesn't handle that
        # It just executes classifier on all files in $ddir

# Results are stored in $resdir/$rtag_[1-14]
##############################################################################

        #hard coded in for loop below to convert filename
if [ $# -eq 2 ]
then
        ddir=$1
        resdir=$2
else
        echo "Enter dataset directory : "
        read ddir  # dataset directory
        echo "Enter result directory : "
        read resdir # results directory
fi
rtag="NumFeat__"
nor_size=71; fty_size=29;

# deleting old files if any
echo; echo "Delete all files in \"$resdir\" , (y|n) : ";
#read ques
ques="y"
if [ $ques = "y" -o $ques = "Y" ]
then
        rm -f $resdir/*;
        echo "files deleted.."
else
        echo; echo "Exiting..."; echo;
        exit 1;
fi

# Executing classifier
echo "Please wait.."
for cdir in ${ddir}/*
do
        for file in ${cdir}/*
        do
                num_feat=`basename ${cdir}`
                outfile=${resdir}/${rtag}${num_feat};
                ./Bayes ${file} ${nor_size} ${fty_size} $num_feat >> ${outfile}
        done
done

# converting full pathname to basename
typeset -x ddir
for file in ${resdir}/${rtag}*
do
        perl -ne 's/$ENV{"ddir"}\/.*\///g; print "$_";' -i $file
done



echo; echo "Done, Results are stored in \"$resdir\"."; echo


##############################################################################
```

A.5 loocvknn.sh

```
#!/bin/sh
##########################################################################

#       Executes Knn-Classifier for each file in each subdir of $ddir

#       for each file :
#               # for each K value in range(K) :
#                       # Create all possible trng-tst sets for 'LOOCV' model
#                       # Execute Knn classifier for each trng-tst set pair

#       Results are redirected to "$resdir/$num_feat/$K"
#   Many instances can work parallely

##########################################################################

# --> check ?? --> dfile="dataset.txt"; trng_f="trng.txt"; tst_f="tst.txt"; num_feat=14

echo "Command_line Usage :  ---$0---dataset_dir---result_dir---"
if [ $# -eq 2 ]
then
        ddir="$1"
        resdir="$2"
else
        echo -n "Enter dataset directory :  "
        read ddir  # dataset directory
        echo -n "Enter result directory :  "
        read resdir # results directory
fi
# trng_f = assigned in loop below
tst_f="/tmp/$$_tst.txt"
K_range="1 3 5 7 9"
tot_size=100;
trng_size=99;
tst_size=1;
rdirtag="NumFeat__"
Ktag="K__"
typeset -x pid=$$
numfeat_list="1 2 3 4 5 6 7 8 9 10 11 12 13 14"

tot_files=`ls ${ddir}/*/*|wc -l`
prg_count=0;

# deleting old files if any
echo; echo "Delete all files in \"$resdir\" , (y|n) : ";
#read ques
ques="y"
if [ $ques = "y" -o $ques = "Y" ]
then
        rm -rf $resdir/*;
        #rm -f $resdir/${rdirtag}[0-9]*/${Ktag}[0-9]*;
        echo "files deleted.."; echo
else
        echo; echo "Exiting..."; echo;
        exit 1;
fi

echo "Creating executables : "
make K > /dev/null
echo "Success "; echo

echo "Applying Knn-Classifier to all files in \"$ddir\" "
echo "Please wait... "
```

```
# Classification start, for each datafile in $ddir/[1-14], for each K in K_range
for cdir in ${ddir}/*
do
        for file in ${cdir}/*
        do
                num_feat=`basename ${cdir}`
                #tot_size=`wc -l < ${file}`
                #trng_size=`expr $tot_size - 1`
                #tst_size=1
                outdir="${resdir}/${rdirtag}${num_feat}";
                if [ ! -d $outdir ]
                then
                        mkdir -p $outdir
                        if [ $? -ne 0 ]
                        then
                                exit 1
                        fi
                fi

                # We assign same name to trng_f, so that classifier "Knn"..
                #    can recognize the name of the datafile it is operating on..
                #               and that name along with other results
                trng_f="/tmp/$$_`basename $file`"
                for K in `echo ${K_range}`
                do
                        i=1;
                        while [ $i -le $tot_size ]  # only 'ith' point is tst_dataset
                        do
                                # constructing training dataset (in file $trng_f)
                                  head -n  `expr $i - 1` ${file}  >| ${trng_f}
                                  tail -n +`expr $i + 1` ${file}  >> ${trng_f}

                                # constructing test dataset (in file $tst_f)
                                  tail -n +$i ${file} | head -n 1 >| ${tst_f}

                                # Applying Knn classifier to current sets
                                ./Knn $trng_f $trng_size $tst_f $tst_size $K $num_feat >> "${outdir}/${Ktag}${K}"

                        i=`expr $i + 1`;
                        done # while
                done # for(K)
                prg_count=`expr $prg_count + 1`
                echo "Progress : $prg_count (file `basename $file`) out of $tot_files complete.."
                rm ${trng_f} ${tst_f}
        done # for(file)
done # for(cdir)

# replacing "/tmp/" with null in all result files
find ${resdir}/ -type f -print0 | xargs -0 perl -ne 's/\/tmp\/$ENV{"pid"}_//g;print "$_";' -i
echo
echo "Done - Results are stored in ${resdir}".; echo
##################################################################################
mplayer /usr/share/sounds/purple/alert.wav >& /dev/null
```

A.6 loocvmlp.sh

```
#!/bin/sh
###########################################################################

#       Executes Mlp-Classifier for each file in each subdir of $ddir

#       for each file :
                        # Create all possible trng-tst sets for 'LOOCV' model
                        # Execute Mlp classifier for each trng-tst set pair

#       Results are stored in "$resdir/"

###########################################################################

echo "Command_line Usage :  ---$0---dataset_dir---result_dir---"; echo
if [ $# -eq 2 ]
then
        ddir="$1"
        resdir="$2"
else
        echo -n "Enter dataset directory :  "
        read ddir  # dataset directory
        echo -n "Enter result directory :  "
        read resdir # results directory
fi

# trng_f = assigned in loop below, since datafile name is need to be passed result file
tst_f="/tmp/$$_tst.txt"
tot_size=100;
trng_size=99;
tst_size=1;
rtag="NumFeat__"
typeset -x pid=$$

tot_files=`ls ${ddir}/*/*|wc -l`
prg_count=0;  # progress

# deleting old files if any
echo; echo "Delete all files in \"$resdir\" , (y|n) : ";
#read ques
ques="y"
if [ $ques = "y" -o $ques = "Y" ]
then
        rm -f $resdir/*;
        echo "files deleted.."; echo
else
        echo; echo "Exiting..."; echo;
        exit 1;
fi

echo "Creating executables : "
make M > /dev/null
echo "Success "; echo

echo "Applying Mlp-Classifier to all files in \"$ddir\" "
echo "Please wait... "
# Classification start, for each datafile in $ddir/[1-14]
for cdir in ${ddir}/*
do
        # for one value of "num_feat", avg will be written to single file ${rtag}_{num_feat}
        avg="";
        num_feat=`basename ${cdir}`
        outfile=${resdir}/${rtag}_${num_feat};
        for file in ${cdir}/* # current data file
        do
                #tot_size=`wc -l < ${file}`
                #trng_size=`expr $tot_size - 1`
                #tst_size=1

                # We assign same name to trng_f, so that classifier "Knn"..
                        #    can recognize the name of the datafile it is operating on..
                        #                and that name along with other results

                i=1;
                trng_f="/tmp/$$_`basename $file`"
                misclass_rate=0;
                while [ $i -le $tot_size ]  # only 'ith' point is tst_dataset
                do
                        # constructing training dataset (in file $trng_f)
                          head -n  `expr $i - 1` ${file}  >| ${trng_f}
                          tail -n +`expr $i + 1` ${file}  >> ${trng_f}

                        # constructing test dataset (in file $tst_f)
                          tail -n +$i ${file} | head -n 1 >| ${tst_f}

                        # Applying Mlp-Classifier to current sets
                        ./Mlp $trng_f  $trng_size  $tst_f  $tst_size  $num_feat >> "${outfile}"
                          cur_misclass=$?  #misclassification for current set

                        misclass_rate=`expr ${misclass_rate} + ${cur_misclass}`

                i=`expr $i + 1`;
                done # while

                avg="${avg}`basename $file`--AVG_misclass_rate = ${misclass_rate}%\n"

                prg_count=`expr $prg_count + 1`
                echo "Progress : $prg_count (file `basename $file`) out of $tot_files complete.."; echo
                rm ${trng_f} ${tst_f}

        done # for(file: cdir)
        echo -e "$avg" >> "${outfile}" # '-e' so that \n is not taken literally

done # for(cdir)

# replacing "/tmp/" with null in all result files
find ${resdir}/ -type f -print0 | xargs -0 perl -ne 's/\/tmp\/$ENV{"pid"}_//g;print "$_";' -i
echo
echo "Done - Results are stored in ${resdir}".; echo
##################################################################################
mplayer /usr/share/sounds/purple/alert.wav >& /dev/null
```

A.7 Svm.sh

```
#!/bin/bash

##########################################################################

        # Executes SVM-Classifier in $ddir for all files, using loocv model

        # Store Results in a single file "$resfile"

        # sorts the results according to accuracy
##########################################################################

echo "Command_line Usage :  ---$0---dataset_dir---result_file---> "; echo
if [ $# -eq 2 ]
then
        ddir="$1"
        res_file="$2"
else
        echo -n "Enter dataset directory :  "
        read ddir
        echo -n "Enter result file :  "
        read res_file
fi

tot_files=`ls $ddir/*/d*|wc -l`
prg_count=0;

rm -f ${res_file};

for file in ${ddir}/*/*
do
        ./svm-train -t 0 -v 100 -h 0 $file | \
                                                    grep Accu| \
                        perl -ne '
                                s/Cross Validation Accuracy = /Cross_Vald_Accuracy--- /g;
                                chomp($_);
                                print "$_"

                                ' >> ${res_file};
        echo " ----$file" >> ${res_file};

        prg_count=`expr $prg_count + 1`
        echo "$prg_count out of $tot_files complete.... ";
done

sort -nr -t " " -k 2 ${res_file} -o /tmp/$$_`basename ${res_file}`
mv -f /tmp/$$_`basename ${res_file}` ${res_file}
echo; echo "Done !!"; echo
```

A.8 bayes.c

```
//##########################################################

//------>        Bayes Classifier [uses Loocv model]     <--------

//##########################################################

#include "header.h"

/// functions prototypes
void read_features (FILE *, float **, int *, int, int, char[]);
void calcMean(float **, int , float* , float* , int );
void sigma_diag(float **, int, float *, float *, int, float *, float *);
float covariance( float **, float *, int, int, int, int);
double calcProbDist(float *, float *, float *, char []);
double determinant_diag(float *, int );

float determinant( float **, int );
float cofactor(float **, int i, int j, int );
void matrix_inverse(float **, float **, int , float );

// Global variable declaration
int num_feat;

int main(int argc, char **argv)
{

        int i, j, k, ii;

        FILE *infp =NULL;
        char infile[MAX_FILENAME_LENGTH];

        float **set;
        int *class_real, tst_class_est;
        float **trng_set, *tst_set ; //trng_set 2-dim, tst-set 1-dim, since "LOOCV"
        int nor_size, fty_size, tot_size;

        float *nor_mean, *fty_mean ;
        float *nor_sigma_diag, *fty_sigma_diag;  // diagonal entries of dispersion matrix
        float nor_pr_prob[2], fty_pr_prob[2];
        double nor_prob_dist, fty_prob_dist;

        int mis_count=0;
        float mis_class_rate=0;

        if (argc != 5)
        {
           dprintf(STDERR_FILENO,"Error : \n");
           dprintf(STDERR_FILENO,"Usage:--%s--dataset_file --nor_size --fty_size --No_of_feat\n",argv[0]);
           exit(1);
        }

        strcpy (infile, argv[1]);
        nor_size = atoi(argv[2]);
        fty_size = atoi(argv[3]);
        num_feat = atoi(argv[4]);
        tot_size = nor_size + fty_size;

        // PRIOR_PROBABILITIES : [0]s when trng set contains, point from normal class
        nor_pr_prob[0] =0.7;
        fty_pr_prob[0] =0.3;
        nor_pr_prob[1] =0.7;
        fty_pr_prob[1] =0.3;
        /* Estimated prior probabilities
        nor_pr_prob[0] = (nor_size -1) / (tot_size -1);
        fty_pr_prob[0] =     (fty_size) / (tot_size -1);
        nor_pr_prob[1] =     (nor_size) / (tot_size -1);
        fty_pr_prob[1] = (fty_size -1) / (tot_size -1);
        */

        // input file
        check_file( (infp = fopen(infile, "r") ), infile);

        //allococating memory
        check_memory ( (set = (float **)calloc(tot_size, sizeof(float *)) ), "set");
        for(i=0 ; i< tot_size; i++)
                check_memory( (set[i] = (float *)calloc(num_feat, sizeof(float)) ), "set[i]");

        check_memory( (class_real = (int *)calloc(tot_size, sizeof(int)) ), "class_real");

        check_memory( (trng_set = (float **)calloc(tot_size -1, sizeof(float *)) ) , "trng_set");
        for(i=0 ; i< tot_size -1; i++)
                check_memory( (trng_set[i]=(float *)calloc(num_feat, sizeof(float)) ), "trng_set[i]");

        // for loocv it is a single point
        check_memory ( (tst_set = (float *)calloc(num_feat, sizeof(float)) ) , "tst_set");

        check_memory( (nor_mean = (float *)malloc( nor_size * sizeof( float ) ) ), "nor_mean");
        check_memory( (fty_mean = (float *)malloc( fty_size * sizeof( float ) ) ), "fty_mean");
        check_memory( (nor_sigma_diag = (float *)calloc( num_feat, sizeof( float )) ), "nor_sigma_diag");
        check_memory( (fty_sigma_diag = (float *)calloc( num_feat, sizeof( float )) ) , "fty_sigma_diag");
        // Memory Allocation ends

        // reading data of "Normal set"
                read_features(infp, set, class_real, nor_size, 0, infile);
        // reading data of "Fatty set"
                read_features(infp, set, class_real, fty_size, nor_size, infile);

// loop for "Loocv model" starts here
        for(ii=0; ii< tot_size; ii++)
        {
                if(ii< nor_size)
                // Construct tst and trng sets
                   for(j=0; j< num_feat; j++)
                                tst_set[j] = set[ii][j];
                k=0;
                for(i=0; i< tot_size; i++) {
                        if (i != ii) {
                                for(j=0; j< num_feat; j++) {
                                        trng_set[k][j] = set[i][j];
                                }
                                k++;
                        }
                }

                if(ii < nor_size )
                        calcMean(trng_set, tot_size -1, nor_mean, fty_mean, nor_size -1);
                else
                        calcMean(trng_set, tot_size -1, nor_mean, fty_mean, nor_size);

                // Dispersion Matrices
                if(ii< nor_size)
                  sigma_diag(trng_set,tot_size -1,nor_mean,fty_mean,nor_size -1,nor_sigma_diag,fty_sigma_diag);
                else
                  sigma_diag(trng_set,tot_size -1,nor_mean,fty_mean,nor_size,nor_sigma_diag,fty_sigma_diag);

                // classification of point(single) in tst set
```

```
            nor_prob_dist = calcProbDist(tst_set, nor_mean, nor_sigma_diag, infile);
            fty_prob_dist = calcProbDist(tst_set, fty_mean, fty_sigma_diag, infile);
                            // printf("nor_prob_dist = %f\n", nor_prob_dist);
            if(ii< nor_size)
            {
                    if( (nor_pr_prob[0] * nor_prob_dist) < (fty_pr_prob[0] * fty_prob_dist) )
                            tst_class_est = 1; // 1 for fatty
                    else
                            tst_class_est = 0; // 0 for normal
            }
            else
            {
                    if( (nor_pr_prob[1] * nor_prob_dist) < (fty_pr_prob[1] * fty_prob_dist) )
                            tst_class_est = 1; // 1 for fatty
                    else
                            tst_class_est = 0; // 0 for normal
            }
            // classification ends

            // calculating misclassification
            if( tst_class_est != class_real[ii])
                            mis_count++;
    } //for(ii)

    mis_class_rate = ( ((float) mis_count) / (tot_size) ) * 100;
    //printf("\nmis_count = %d\n", mis_count);
    printf("%s_", infile);
    printf("---mis_class_rate = %f\n", mis_class_rate);

// Printing to Check

    /* for(i=0; i<fty_size; i++)   {
            for(j=0; j<num_feat; j++)
                    printf("%f  ", fty_set[i][j]);
            printf("\nclass[%d] = %d\n\n", i, fty_class_real[i]);
       } */

// Printing ends


    //freeing memory
    for(i=0 ; i< tot_size; i++)
            free( set[i] );
    free(set);
    free (class_real);

    for(i=0 ; i< tot_size -1; i++)
            free(trng_set[i]);
    free(trng_set);
    free(tst_set);

    free( nor_mean );
    free( fty_mean );
    free( nor_sigma_diag );
    free( fty_sigma_diag );

    fclose(infp);
    return 0;
}

double determinant_diag(float *array_diag, int size)
{
    int i;
    double product=1;
```

```
        for(i=0; i< size; i++)
                product *= array_diag[i];

        return product;
}

double calcProbDist(float *x, float *mean, float *sigma_diag, char infile[])
{
        double det, numer, denom;
        int i, j, k;
        float t1, t2;
        double sum = 0;

        // for naive-bayes classifier
                det = determinant_diag(sigma_diag, num_feat) ;

        // for bayes classifier
        // Warning : Value of Sigma is not available upto here
        //determinant( float **sigma, int size)

        t1 = sqrt ( 2* PI);
        t2 = pow  ( t1, num_feat );
        denom = t2 * sqrt(det);  // denominator

        for(i=0; i< num_feat; i++)
                sum += ( ( (x[i]-mean[i]) *(x[i]-mean[i]) ) / sigma_diag[i]  );
        numer = exp( (-0.5) * sum );

                        //printf("\nDeterminant = %f", det);
                        //printf("\nsum = %f\n", sum);
                        //printf("numer = %f\n", numer);
                        //printf("denom= %f\n", denom);
                        //printf("prob_dist = %f\n", numer/denom);
        return (numer / denom) ;
}

void sigma_diag(float **set,int size,float *mean1,float *mean2,int size1,float *sigma1,float *sigma2)
{
        int i, j, k;

        for(i=0; i< num_feat; i++)
        {
                sigma1[i] = covariance(set, mean1, 0, size1-1, i, i);
                sigma2[i] = covariance(set, mean2, size1, size-1, i, i);

                // if above values are zero we add small epsilon to these
                // otherwise it will lead to zero probability for that feature
                 if(sigma1[i]==0)
                        sigma1[i]=0.4; //based on other values
                 if(sigma2[i]==0)
                        sigma2[i]=0.4; //based on other values
        }

        return;
}

float covariance( float **set, float *mean, int str_idx, int end_idx, int X, int Y)
{
        int i, j, k;
        float temp;
        float cov = 0;

        for(i=str_idx; i<= end_idx; i++)
                cov += ( set[i][X] - mean[X] ) * ( set[i][Y] - mean[Y] ) ;
        cov /= (end_idx -str_idx +1);

        return cov;
}


void calcMean(float **set, int size, float* mean1, float* mean2, int size1)
{
        int i, j;

        for(i=0; i< num_feat; i++)  // initailization
                mean1[i] = mean2[i] = 0;

        // class-1
        for(i=0; i< size1; i++) // loop for data set
                for(j=0; j< num_feat; j++)
                        mean1[j] += set[i][j];

        // class-2
        for(i=size1; i< size; i++) // loop for data set
                for(j=0; j< num_feat; j++)
                        mean2[j] += set[i][j];

        for(j=0; j< num_feat; j++)
        {
                mean1[j] = mean1[j] / size1;
                mean2[j] = mean2[j] / (size - size1);
        }

        return;
}

void read_features (FILE *fp, float **set, int *class, int size, int w_idx, char filename[])
{
        int i, j, k;
        char tmp;

        i = w_idx;
        while ( i< (w_idx +size) )
        {
           fscanf(fp, "%d", &class[i]);
           for (j=0; j< num_feat; j++)
                   fscanf(fp, "%f", &set[i][j] ) ;

           tmp = fgetc(fp); //space after float
           tmp = fgetc(fp);
           if( tmp != '\n' ) // check that newline was/has encountered or not
           {
              dprintf(STDERR_FILENO,"Error: In file \"%s\" :\n", filename);
              dprintf(STDERR_FILENO,"\"Num_of_feature=(%d)\" is not matching datafile!\n", num_feat);
              exit(1);
           }
           i++;
        }

        return ;
}




void matrix_inverse(float **mtrx, float **inv, int size, float det)
{
        float **adj;
        int i, j;
        if(det==0)
        {
                dprintf(STDERR_FILENO,"Error : \n");
                dprintf(STDERR_FILENO,"Inverse of the matrix sigma does not exist.\n");
                exit(1);
        }
/*
size = 2;
for(i=0; i<size; i++)
        for(j=0; j<size; j++)
                scanf("%f", &mtrx[i][j]);
det = determinant(mtrx, size);*/

        // Calculating Inverse
        for(i=0;i< size ; ++i)
                for(j=0; j< size; j++)
                        inv[i][j] = ( cofactor(mtrx, j, i, size) ) / det;
/*
for(i=0; i<size; i++) {
        for(j=0; j<size; j++)   {
                printf("%f\t", inv[i][j]);
        }
        printf("\n");
*/
        return;

}

float cofactor(float **mtrx, int i, int j, int size)
{
        int k,l;
        float **cofct;
        float result;

        // Memory allocation for cofactor
        cofct = (float **)calloc( size-1, sizeof(float *));
        for(k=0; k< size-1; k++)
                cofct[k] = (float *)calloc( size-1, sizeof(float));


        for(k=0; k< size-1; ++k)
        {
                for(l=0; l< size-1; l++)
                {
                        if ( (k< i) && (l< j) )
                                cofct[k][l] = mtrx[k][l];

                        if( (k< i) && (l>= j) )
                                cofct[k][l] = mtrx[k][l+1];

                        if( (k>= i) && (l< j) )
                                cofct[k][l] = mtrx[k+1][l];

                        if( (k>= i) && (l>= j) )
                                cofct[k][l] = mtrx[k+1][l+1];
                }
        }

        result =  pow(-1, i+j) * determinant( cofct, size-1 );

        // Freeing memory for cofactor
        for(k=0; k< size-1; k++)
                free(cofct[k]);
        free( cofct );

        return result;
}

float determinant( float **mtrx, int size)
{
        int i,j,k;
        float det = 0;
        float **minor;


        if(size==1)
                return mtrx[0][0];
        else
        {
                // memory allocation for minor
                minor = (float **)calloc(size-1, sizeof(float *));
                for(i=0; i< size-1; i++)
                        minor[i] = (float *)calloc(size-1, sizeof(float));

                for(k=0; k< size; k++)
                {
                        for(i=0; i< size-1; i++)  // constructing minor
                        {
                                for(j=0; j< size-1; j++)  // constructing minor
                                {
                                        if (j<k)
                                                minor[i][j] = mtrx[i+1][j];
                                        else
                                                minor[i][j] = mtrx[i+1][j+1];
                                }
                        }
                        // Call recursively on 'Minor'
                        det += pow(-1,k)* mtrx[0][k] * determinant(minor, size-1);
                }
        }

        for(i=0; i< size-1; i++)
                free(minor[i]);
        free(minor);
        if (det==0)
        {
                dprintf(STDERR_FILENO, "Error : (In function \"determinant\")\n determinant of dispersion matrix=0\n Stop !");
                exit(2);
        }
        return det;

}
```

A.9 knn.c

```
// K-NN classifier

#include "header.h"

struct neighbour {
        float dist;
        int idx;
        // "idx" is required since "class" array is not sorted alongwith nbrs
};

// function prototypes
void read_features (FILE *, float **, int *, int );
void distance(struct neighbour *, int , float *, float *);
void sort(struct neighbour *, int );

// global variable declaration
int num_feat;
   // Value of num_feat doesn't change in one run
   // It will be accessed by all functions

int main(int argc, char **argv)
{
        struct neighbour *nbrs =NULL;

        int i, j, k, ii;

        FILE *trng_fp =NULL, *tst_fp =NULL;
        char trng_file[MAX_FILENAME_LENGTH], tst_file[MAX_FILENAME_LENGTH];

        float **trng_set, **tst_set ;
        int *trng_class, *tst_class_real, *tst_class_est;
        int trng_size, tst_size;

        int K; // K-NN value
        int count0, count1, tmp, mis_count=0;
        float mis_class_rate=0;

        if (argc != 7)
        {
                dprintf(STDERR_FILENO,"\nError :\n");
                dprintf(STDERR_FILENO,"Usage: %s trng_file trng_size tst_file tst_size K_val Num_Features\n",argv[0]);
                exit(1);
        }

        strcpy(trng_file, argv[1]);
        trng_size = atoi(argv[2]);
        strcpy(tst_file, argv[3]);
        tst_size = atoi(argv[4]);

        K = atoi(argv[5]);           // K nearest nbrs
        num_feat = atoi(argv[6]);    // number of features

        check_file( (trng_fp =fopen(trng_file, "r")), trng_file);
        check_file( (tst_fp =fopen(tst_file, "r")), tst_file);

        // value of 'K'can't be greater than total training data
        if(K > trng_size)
        {
                dprintf(STDERR_FILENO,"Error :\n");
                dprintf(STDERR_FILENO,"value of K(=%d) must not be more than training_data_size(=%d) !\n", K, trng_size);
                exit(1);
        }
```

```
//allocating memory
check_memory( (nbrs = (struct neighbour *)malloc(trng_size* sizeof(struct neighbour)) ), "nbrs");

check_memory( (trng_set = (float **)calloc(trng_size, sizeof(float *))), "trng_set");
for(i=0 ; i< trng_size; i++)
        check_memory( (trng_set[i]=(float *)calloc(num_feat, sizeof(float))), "trng_set[i]");

check_memory( (trng_class = (int *)calloc(trng_size, sizeof(int)) ), "trng_class");

check_memory( (tst_set = (float **)calloc(tst_size, sizeof(float *)) ), "tst_set");
for(i=0 ; i< tst_size; i++)
        check_memory( (tst_set[i]=(float *)calloc(num_feat, sizeof(float)) ), "tst_set[i]");

check_memory( (tst_class_real = (int *)calloc(tst_size, sizeof(int)) ), "tst_class_real");
check_memory( (tst_class_est = (int *)calloc(tst_size, sizeof(int))), "tst_class_real");

// reading trainging data from file into "trng_set"
read_features(trng_fp, trng_set, trng_class, trng_size);

// reading test data from file into "tst_set"
read_features(tst_fp, tst_set, tst_class_real, tst_size);

// Main K-NN loop for test data
count0 = 0; count1 = 0;
for(i=0; i< tst_size; i++)
{
        // tst_set[i] = current data point array
        for( j=0; j< trng_size; j++)  //jth nbr
                distance(nbrs, j, tst_set[i], trng_set[j] );
        // only single point is classified at a time, hence "nbrs" is a one dimensional array
        // no_of_entries in "nbrs" = no_of_nbrs = trng_size
        // each entry stores "distance" + "idx"
        // "idx" initially is ascending order but will change after sorting
        // "idx" is required since "class" array is not sorted alongwith nbrs

        sort(nbrs, trng_size);   // sort according to disrtance

        for(k=0; k< K; k++)
        {
                tmp = nbrs[k].idx;
                if( trng_class[tmp] == 0.0 )
                        count0++;
                else if( trng_class[tmp] == 1.0)
                        count1++;
                else
                {
                        dprintf(STDERR_FILENO,"Error : \nClass = %f\n", trng_class[tmp]);
                        exit(1);
                }
        }
        if( count0 > count1 )
                tst_class_est[i] = 0;
        else
                tst_class_est[i] = 1;
        //printf("%d\t%d\n", tst_class_real[i], tst_class_est[i]);
} //for(i)

mis_count = 0;
for(i=0; i< tst_size; i++)
{
        if( tst_class_est[i] != tst_class_real[i])
                mis_count++;
}
mis_class_rate = ( ((float) mis_count) / tst_size ) * (100);
```

```
        //printf("\nmis_count = %d\n", mis_count);


        printf("%s", trng_file);
        printf("---mis_class_rate = %f\n", mis_class_rate);

        /*      for(i=0; i<tst_size; i++) {
                for(j=0; j<num_feat; j++) {
                printf("%f  ", tst_set[i][j]);
                }
                printf("\nclass[%d] = %d\n\n", i, tst_class_real[i]);
                }
                */

        // freeing up memory
        free(nbrs);

        free(trng_class);
        for(i=0 ; i< trng_size; i++)
                free(trng_set[i]);
        free(trng_set);

        free(tst_class_real);
        free(tst_class_est);
        for(i=0 ; i< tst_size; i++)
                free(tst_set[i]);
        free(tst_set);

        fclose(trng_fp);
        fclose(tst_fp);

        return 0;
}

void sort(struct neighbour *nbrs, int trng_size)
{
        int i, j, k;
        struct neighbour *ntmp;

        check_memory( (ntmp = (struct neighbour *)malloc(trng_size* sizeof(struct neighbour)) ), "ntmp");
        for(i=0; i<trng_size; i++)
        {
                ntmp[i].dist = nbrs[i].dist;
                ntmp[i].idx  = nbrs[i].idx;
        }


        for(i =0; i< trng_size; i++)
        {
                for(j =0; j <i; j++)
                {
                        if( ntmp[i].dist < nbrs[j].dist )
                        {
                                for( k =i; k >j; k--)
                                {
                                        nbrs[k].dist = nbrs[k-1].dist;
                                        nbrs[k].idx = nbrs[k-1].idx;

                                }
                                nbrs[j].dist = ntmp[i].dist;
                                nbrs[j].idx = ntmp[i].idx;
                                break;
                        }
                }
                if(j==i)
                {
                        nbrs[i].dist = ntmp[i].dist;
                        nbrs[i].idx = ntmp[i].idx;
                }
        }
        return;
}

void distance(struct neighbour *nbrs, int idx, float *point1, float *point2)
{
        int i, j, k;
        float distance;
        float t1, t2, t3;

        distance = 0;
        for(i=0; i< num_feat; i++)
        {
                t1 = ( point1[i] - point2[i] ) * ( point1[i] - point2[i] ) ;
                distance += t1;
        }
        distance = sqrt(distance);

        nbrs[idx].dist  =  distance ;
        nbrs[idx].idx   =  idx;

        return;
}


void read_features (FILE *fp, float **set, int *class, int size)
{
        int i, j, k;
        char tmp;

        i=0;
        while ( i< size )  // 'i' represents line number of data file
        {
                fscanf(fp, "%d", &class[i]);

                for (j=0; j< num_feat; j++)  // num_feat is global variable
                        fscanf(fp, "%f", &set[i][j] ) ;

                tmp = fgetc(fp); //space after float
                tmp = fgetc(fp);
                if( tmp != '\n' ) // check that newline was/has encountered or not
                {
                        dprintf(STDERR_FILENO, "Error :\n");
                        dprintf(STDERR_FILENO,"Input Argument \"Number_of_feature(=%d)\"  is not matching data_file !\n",num_f
                        exit(1);
                }
                i++;
        }

        /*      i=0;
                while ( i< size )
                {
                fscanf(fp, "%d", &class[i]);

                j=0;
                fscanf(fp, "%f", &set[i][j++] ) ;
                fscanf(fp, "%f", &set[i][j++] ) ;
                fscanf(fp, "%f", &set[i][j++] ) ;
                fscanf(fp, "%f", &set[i][j++] ) ;
                fscanf(fp, "%f", &set[i][j++] ) ;
```

```
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;
            fscanf(fp, "%f", &set[i][j++] ) ;

            i++;

            }  */

}
```

A.10 mlp.c

```
//###########################################################################

//        ——————>                  MLP    Classifier                   <————————

//###########################################################################

#include "header.h"


#define MAX_NO_POINTS 110
#define MAX_NO_FEATURES 17

#define NO_OF_LAYERS 5
#define NO_OF_NEURONS 10 // in each layer except input and output layer

#ifndef LEARNING_RATE
#define LEARNING_RATE 0.25
#endif

#define ITERATION 500

typedef struct
{
        float y;       // Output of this node
        float actv_fn;// Activation function
        float delta;  // delta-local gradient
        float v;       // Local induced field
        int type;      // 0 if input node, 1 if hidden node, 2 if output node
        float *weight;// Array of weights going out from this node
} NEURON;

void read_data(FILE *, float [][MAX_NO_FEATURES], int , int );
void update_weight_for_data(NEURON ***, int, int, float[], int *);
float test_network(NEURON ***, float[][MAX_NO_FEATURES], int , int, int, int *);
void print_output(NEURON **, int);
void print_net(NEURON **,int *);
void generate_data(int[][MAX_NO_FEATURES] ,int );
void create_neural_network(NEURON ***, int, int, int *);

int main(int argc, char **argv)
{
        FILE *trng_fp =NULL, *tst_fp =NULL;
        float trng_data[MAX_NO_POINTS][MAX_NO_FEATURES], tst_data[MAX_NO_POINTS][MAX_NO_FEATURES];
        int trng_size, tst_size;
        int no_of_features, no_of_classes;
        NEURON **node_structure;
        int i,j;
        char trng_file[MAX_FILENAME_LENGTH], tst_file[MAX_FILENAME_LENGTH];
        int *num_of_neuron =NULL; // this array stores num_of_neurons in each layer
        float mis_class_rate;

        if (argc != 6)
        {
                dprintf(STDERR_FILENO, "Error :\n");
                dprintf(STDERR_FILENO, "Usage : %s  trng_file  trng_size  tst_file  tst_size  no_of_features\n", argv[0]);
                exit(1);
        }
        strcpy(trng_file, argv[1]);
        trng_size = atoi(argv[2]);
        strcpy(tst_file, argv[3]);
        tst_size = atoi(argv[4]);
        no_of_features = atoi(argv[5]) ;
        no_of_classes=2;

        check_file( (trng_fp = fopen(trng_file, "r")), trng_file);
        check_file( (tst_fp = fopen(tst_file, "r")), tst_file);

        check_memory( (num_of_neuron = (int *)calloc(NO_OF_LAYERS, sizeof(int)) ), "num_of_neuron");

        // reading data into memory
        read_data(trng_fp, trng_data, no_of_features, trng_size);
        fclose(trng_fp);
        read_data(tst_fp, tst_data, no_of_features, tst_size);
        fclose(tst_fp);

        // Creation of Neural Network, Initialization of weights
        create_neural_network(&node_structure, no_of_features, no_of_classes, num_of_neuron);

        //print_net(node_structure, num_of_neuron);

        // Training -- Weight updation
        for(i=0; i< ITERATION; i++)
        {
                for(j=0; j< trng_size; j++)
                {
                        // trng_data[j] is current trng_data point
                        update_weight_for_data( &node_structure, no_of_features, no_of_classes, trng_data[j], num_of_neuron);
                }
        }

        //printf("file : %s\n", trng_file);
        // Testing of network
        printf("%s_", trng_file);
        mis_class_rate =test_network(&node_structure, tst_data, no_of_features, no_of_classes, tst_size, num_of_neuron);


        //printf("\n");
        //      printf("\nLearning rate  = %f\n", LEARNING_RATE);

        // Printing to check
        /*
                for(i=0; i< trng_size; i++)     {
                for (j=0; j< no_of_features; j++) {
                printf("%f ", trng_data[i][j]);
                }
                printf("%d \n", (int)trng_data[i][j]);
                }
                exit(2);
                */
        // freeing memory
        for(i=0; i< NO_OF_LAYERS; i++) // for each layer
                for(j=0; j< num_of_neuron[i]; j++) // for each neuron in current layer
                        free(node_structure[i][j].weight);

        for(i=0; i< NO_OF_LAYERS; i++)
                free (node_structure[i]);
        free(node_structure);

        free(num_of_neuron);

        return ((int) mis_class_rate);

}

void read_data(FILE *fp, float data[][MAX_NO_FEATURES], int num_feat, int size)
{
        int i, j, k;
        char tmp;
        int class[MAX_NO_POINTS];

        i=0;
        while ( i< size )  // 'i' is line_no. of data file
        {
                for (j=0; j< num_feat; j++)
                        fscanf(fp, "%f", &data[i][j] ) ;
                fscanf(fp, "%d", &class[i]);

                tmp = fgetc(fp);
                tmp = fgetc(fp);
                if( tmp != '\n' ) // check that newline was/has encountered or not
                {
                        dprintf(STDERR_FILENO,"Error :\n");
                        dprintf(STDERR_FILENO,"\"Number_of_feature(=%d)\" is not matching data_file !\n",num_feat);
                        exit(1);
                }

                i++;
        } // while(i)

        for(i=0; i< size; i++)
                data[i][j] = class[i];

        return;
}


void print_net(NEURON **nn, int *num_of_neuron)
{
        int i,j,k;

        printf("From\t\tTo\n");
        printf("Layer—node\tLayer—node\t Weight\n");
        printf("############################################");

        for(i=0; i< NO_OF_LAYERS-1; i++)
        {
                printf("\n#%d",i+1);
                for(j=0; j< num_of_neuron[i]; j++)
                {
                        for(k=0; k< num_of_neuron[i+1]; k++)
                        {
                                printf("\n%d———%d\t\t%d———%d\t\t%f", i+1, j+1, i+2, k+1, nn[i][j].weight[k]);
                        }
                }
        }
        printf("\n#####################################################");
        printf("\n\n");

        return;

}

void create_neural_network(NEURON ***node1, int no_of_features, int no_of_classes, int *num_of_neuron)
{
        int i,j,k;
        int stime;
        long ltime;

        ltime = time(NULL);
        stime = (unsigned) ltime/2;
        srand(stime*2);
        // Allocating memory
        // "no_of_features" neurons in each layer
        check_memory( (*node1=(NEURON **)malloc(NO_OF_LAYERS *sizeof(NEURON *))), "*node1");
        for(i=0; i< NO_OF_LAYERS; i++)
        {
                if(i ==NO_OF_LAYERS-1) //for output layer
                {
                        num_of_neuron[i] = no_of_classes;
                        check_memory( ( (*node1)[i] =(NEURON *)malloc(no_of_classes *sizeof(NEURON))), "(*node1)[i])");
                }
                else if(i ==0) // for input layer
                {
                        num_of_neuron[i] = no_of_features;
                        check_memory( ( (*node1)[i] =(NEURON *)malloc(no_of_features *sizeof(NEURON)) ), "(*node1)[i]");
                }
                else // hidden layer
                {
                        num_of_neuron[i] = NO_OF_NEURONS;
                        check_memory( ((*node1)[i] =(NEURON *)malloc(NO_OF_NEURONS *sizeof(NEURON)) ), "(*node1)[i]");
                }
        }

        // Initialization of weights
        for(i=0; i< NO_OF_LAYERS; i++) // for each layer
        {
                for(j=0; j< num_of_neuron[i]; j++) // for each neuron in current layer
                {

                        if(i != NO_OF_LAYERS-1) // if not last layer
                        {
                                // allocation memory
                                check_memory( ((*node1)[i][j].weight = (float *)malloc(num_of_neuron[i+1]\
                                                                *sizeof(float) )), "(*node1)[i][j].weight");
                                for(k=0; k< num_of_neuron[i+1]; k++)
                                {
                                        (*node1)[i][j].weight[k] = rand()%10;
                                        (*node1)[i][j].weight[k] /= 10;
                                        (*node1)[i][j].weight[k] -= 0.5;

                                }
                                if(i==0)
                                        (*node1)[i][j].type=0; // input neuron
                                else
                                        (*node1)[i][j].type=1; // hidden neuron
                        }

                        else  // if last layer
                        {
                                (*node1)[i][j].weight = NULL;
                                (*node1)[i][j].type = 2;
                        }
                }
        } // for(i)

        return;
}

void update_weight_for_data(NEURON ***node, int no_of_features, int no_of_classes, float data[], int *num_of_neuron )
{
        int i, j, k;
        float err, sum;


        // take input at 0th layer
        for(j=0; j< num_of_neuron[0]; j++)
```

```
                (*node)[0][j].y = data[j];

        // calculate Vj and Yj
        for(i=1; i< NO_OF_LAYERS; i++)
        {
                for(j=0; j< num_of_neuron[i]; j++)
                {
                        (*node)[i][j].v = 0;
                        for(k=0; k< num_of_neuron[i-1]; k++)
                                (*node)[i][j].v += (*node)[i-1][k].weight[j] * (*node)[i-1][k].y;
                        (*node)[i][j].y = 1/ (1+exp( (-1) *((*node)[i][j].v)) );

                        //    Sigmoid: f(x)= 1 /(1 + exp(-ax) ) ---- take a=1
                }
        }

        /* BACKPROPAGATION STARTS   */
        // error for output layer
        //
        // desired response vector for the current data point is :
        //      err[0]=0, err[1]=0, err[2]=0......"err[data[no_of_feature]]=1"... rest_all_zero...
        //                                                                              =class_of_data
        for(j=0; j< num_of_neuron[NO_OF_LAYERS-1]; j++)
        {
                if(j == data[no_of_features] )
                        err = 1 - (*node)[NO_OF_LAYERS-1][j].y;
                else
                        err = 0 - (*node)[NO_OF_LAYERS-1][j].y;
                // eq. 4.46 : for output layer ---- Sj = err(j) * Yj(1-Yj)
                (*node)[NO_OF_LAYERS-1][j].delta = err * ((*node)[NO_OF_LAYERS-1][j].y) * (1- (*node)[NO_OF_LAYERS-1][j].y);
        }
        // for other layers
        for(i=NO_OF_LAYERS-2; i >=0; i--)
        {
                for(j=0; j< num_of_neuron[i]; j++)
                {
                        (*node)[i][j].delta = 0;

                        for(k=0; k< num_of_neuron[i+1]; k++)
                        {
                                ((*node)[i][j].delta) += ((*node)[i][j].weight[k]) * ((*node)[i+1][k].delta);

                                ((*node)[i][j].weight[k]) += LEARNING_RATE * ((*node)[i+1][k].delta) * ((*node)[i][j].y);
                        }
                        ((*node)[i][j].delta) = ((*node)[i][j].delta) * ((*node)[i][j].y) * (1-(*node)[i][j].y);
                }
        }
        /* BACKPROPAGATION ENDS */

        return;
}


float test_network(NEURON ***node, float data[][MAX_NO_FEATURES], int no_of_features, int no_of_classes, int no_of_pts, int *nu
{
        int i, j, k, dp, max;
        float err, sum;
        int misclass=0;
        float mis_class_rate;


        for(dp=0; dp< no_of_pts; dp++)
        {
                for(j=0; j< num_of_neuron[0]; j++)
                        (*node)[0][j].y = data[dp][j];
                for(i=1; i< NO_OF_LAYERS; i++)
                {
                        for(j=0; j< num_of_neuron[i]; j++)
                        {
                                (*node)[i][j].v = 0;
                                for(k=0; k< num_of_neuron[i-1]; k++)
                                        (*node)[i][j].v += (*node)[i-1][k].weight[j] * (*node)[i-1][k].y;
                                (*node)[i][j].y = 1 /( 1 + exp( (-1)* (*node)[i][j].v) );
                        }
                }
                //print_output(*node, num_of_neuron[NO_OF_LAYERS-1]);

                max = 0;
                for(j=1; j< num_of_neuron[NO_OF_LAYERS-1]; j++)
                        if( (*node)[NO_OF_LAYERS-1][j].y > (*node)[NO_OF_LAYERS-1][max].y )
                                max = j;

                //printf("max=%d\tclass=%f\n\n", max, data[dp][no_of_features]);
                if( data[dp][no_of_features] != (float)max )
                        misclass++;
        }
        mis_class_rate =((float)misclass)/no_of_pts;
        printf("---mis_class_rate = %f\n", mis_class_rate );

        return mis_class_rate;
}

void print_output(NEURON **node_structure, int neu_in_last)
{
        int j;
        printf("\n");
        for(j=0; j< neu_in_last; j++)
                printf("%d--%f  ",j, node_structure[NO_OF_LAYERS-1][j].y );
        return;
}

void generate_data(int data[][MAX_NO_FEATURES], int n)
{
        int i;
        int stime;
        long ltime;

        ltime = time(NULL);
        stime = (unsigned) ltime/2;
        srand(stime);

        for(i=0;i<n;i++)
        {
                data[i][0]=rand()%300+150;
                data[i][1]=rand()%300+50;
                if((((long)data[i][0]-300)*((long)data[i][0]-300)+((long)data[i][1]-200)*((long)data[i][1]-200))<100*100)
                        data[i][2]=0;
                else
                        data[i][2]=1;
        }
}

int create_data(int data[][MAX_NO_FEATURES],int no_of_features,int no_of_classes,int no_of_pts)
{
        int i,j,rnd;
        int stime;
        long ltime;

        ltime = time(NULL);
        stime = (unsigned) ltime/2;
        srand(stime);
        for(i=0;i<no_of_pts;i++)
        {
                for(j=0;j<no_of_features;j++)
                {
                        data[i][j]=rand()%10;
                        data[i][j]-=5;
                        printf("%d\t",data[i][j]);
                }
                rnd=rand()% no_of_classes;
                data[i][j]=rnd;
                printf("%d\n",data[i][j]);
        }
}
```

A.11 mainreadextract.c

```
// Reading ROI-images and extracting features
// Gaussian Filtering is done before extraction

#include "header.h"

// initialization of high pass filter
int high_pass_filter[5][5] = {
      0, 0, -1, 0, 0,
      0, -1, -2, -1, 0,
     -1, -2, 16, -2, -1,
      0, -1, -2, -1, 0,
      0, 0, -1, 0, 0
};


int main(int argc,char* argv[])
{
        FILE *fp_nf =NULL, *fp_ff =NULL, *fp_feat =NULL ;
        char fname_nf[MAX_FILENAME_LENGTH], fname_ff[MAX_FILENAME_LENGTH];
        int valid=0;
        int i, j;
        char fname_feat[MAX_FILENAME_LENGTH], mode[2];// mode= write or append
        int class;

        // current structures
        struct image img_nf;
        struct image img_ff;
        struct gLCM coc, coc1, coc2, coc3, coc4;
        struct features feat;

        if(argc!=6)
        {
                dprintf(STDERR_FILENO, "Error : \n");
                dprintf(STDERR_FILENO, "USAGE: %s  nf_File  ff_File  feat_ouput_file  mode  Class\n",argv[0]);
                exit(1);
        }

        strcpy(fname_nf, argv[1]);
        strcpy(fname_ff, argv[2]);
        strcpy(fname_feat, argv[3]);
        strcpy(mode, argv[4]);
        class = atoi( argv[5] );

        // Reading near-field file
        fp_nf = fopen(fname_nf,"r");
        check_file(fp_nf, fname_nf);
        isValidPGM(fp_nf, &img_nf, fname_nf);
        readPGM(fp_nf, &img_nf);  //finally reading call
        fclose(fp_nf);

        // Reading far-field file
        fp_ff = fopen(fname_ff,"r");
        check_file(fp_ff, fname_ff);
        isValidPGM(fp_ff, &img_ff, fname_ff);
        readPGM(fp_ff, &img_ff);  //finally reading call
        fclose(fp_ff);

        // validity of output file
        if( strcmp(mode, "w") && strcmp(mode, "a") )
        {
                dprintf(STDERR_FILENO, "Error : \n");
                dprintf(STDERR_FILENO, "Valid modes are \'w\'(write) and \'a\'(append)  !!\n");
                exit(1);
        }

        fp_feat = fopen(fname_feat, mode);
        check_file(fp_feat, fname_feat);

        //filterning
        if(FILTERING == 1)
        {
                gaussian_filtering( &img_nf);
                gaussian_filtering( &img_ff);
        }

        // In file misc.c
          calc_pmin_pmax(&img_nf); //GLCM is only constructed for near-field

        //calculate GLCMs
          calculateGLCM( &coc, &img_nf, 0, 1, 1);
/*
        calculateGLCM( &coc1, &img_nf, 0, 1, 1); // angle=0, dist=1,sym
        calculateGLCM( &coc2, &img_nf, 7, 1, 1); // angle=+45
        calculateGLCM( &coc3, &img_nf, 6, 1, 1); // angle=+90
        calculateGLCM( &coc4, &img_nf, 5, 1, 1); // angle=+135

        //calculate average GLCM
        calculateAvgGLCM(&coc, &coc1, &coc2, &coc3, &coc4); // avg of all above
*/
        normalizeGLCM (&coc);

        calc_GLCM_Features(&coc, &feat);

        calc_NFLSD(&img_nf, &feat);

        calc_NFFGR(&img_nf, &img_ff, &feat);

    //writing features to file
        store_features(fp_feat, &feat, class);
    fclose(fp_feat);


// printing to check :

        /*
        //image
        for (i=0; i < img_nf.height; i++) {
                for (j=0; j < img_nf.width; j++) {
                        printf(”%u\n”,(unsigned char)img_nf.pixel[i][j]);
                }
        }
        */


        /*
        // co-occurence
        printf(”min=%d\nmax=%d, start_idx=%d, end_idx=%d, size=%d\n”, img_nf.pmin, img_nf.pmax,coc.start_idx, coc.end_idx, coc.
        for(i=0; i< coc.size; i++) {
                for(j=0; j< coc.size; j++)
                {
                        //if(coc.co_ocr[i][j] >1 )
                                //printf(”%d\t”,coc.co_ocr[i][j]);
                                if(coc.co_ocr[i][j] != coc.co_ocr[j][i] )
                                {
                                        //printf(”false i=%d, j=%d\n”, i, j);
                                }
                                //printf(”%f\n”,coc.co_ocr[i][j]);
                }
                //printf("\n");
        }
        */



        /*
        //contrast
        printf("\ncon = %f\ndis = %f\nhom = %f\n\nasm = %f\nenrg=%f\nmax_prob=%f\nentr = %f\n\ngl_mean = %f\ngl_var=%f\ngl_std_
        printf("\nnFLSD = %d\nnFFGR = %f \n\n", feat.nFLSD, feat.nFFGR);
        printf("\nsum_entropy = %f\n\n", feat.sum_entr);
        */


        /*        freeing memory              */
        // image memory
        for(i=0; i< img_nf.height; i++)
                free( img_nf.pixel[i] );
        free( img_nf.pixel );

        for(i=0; i< img_ff.height; i++)
                free( img_ff.pixel[i] );
        free( img_ff.pixel );

        // co-occurence matrix memory
        for(i=0; i< coc.size; i++)
                free( coc.co_ocr[i] );
        free( coc.co_ocr );

        return 0;
}
```

A.12 processpgm.c

```
#include "header.h"

// returns 1/2 for valid, 0 for invalid pgm file
void isValidPGM(FILE *fp, struct image *img, char *fname)
{
        char magicNumber[20];

        fscanf(fp,"%s",magicNumber);
        magicNumber[19]='\0';
        if( !strcmp("P2",magicNumber) )
        {
                strcpy(img->type, "P2");
                img->inStr="%d";
                return;
        }
        else if( !strcmp("P5",magicNumber) )
        {
                strcpy(img->type, "P5");
                img->inStr="%c";
                return;
        }
        else
        {
                dprintf(STDERR_FILENO," Error : \n");
                dprintf(STDERR_FILENO," file %s is not a valid pgm image!\n",fname);
                exit(2);
        }
}

int *readPGM(FILE *fp, struct image *img)
{
        int i,j;
        int temp=0;

        fp = removeComment(fp);
        fscanf(fp,"%d",&img->width);

        fp = removeComment(fp);
        fscanf(fp,"%d",&img->height);

        fp = removeComment(fp);
        fscanf(fp,"%d",&img->pmaxval);

        if( !strcmp(img->type, "P5")  )
               temp = fgetc(fp);  // to eat trailing newline char of maxval line
                       // no need for "P2" since data will be read using '%d' not by '%c'

#ifdef DEBUG
  printf("\n**********************************************************************\n");
  printf("width:%d\theight:%d\tmaxval:%d",img->width, img->height, img->pmaxval);
  printf("\n**********************************************************************\n");
#endif
        // allocating memory for image pixels
        check_memory( (img->pixel = (unsigned char **)malloc( img->height * sizeof( unsigned char *) )), "img->pixel");
        for (i=0; i < img->height; i++)
               check_memory( (img->pixel[i]=(unsigned char *)malloc( img->width*sizeof(unsigned char))), "img->pixel[i]");

        // reading pixel data now
        for (i=0; i < img->height; i++)
               for (j=0; j < img->width; j++)
                       fscanf(fp, img->inStr, &img->pixel[i][j]);
        return 0;
}


FILE* removeComment(FILE *fp)
{
        char ch, buff[1000];

        while(1)
        {
                ch=fgetc(fp);
                while(W_SPACE(ch))
                        ch=fgetc(fp);

                if(ch=='#')
                        fgets(buff,1000,fp);
                else
                        break;
        }
        ungetc(ch,fp);  //put current char back to file

        return fp;
}
```

A.13 gaussianfiltering.c

```
// Apply gaussian filter to input img (struct img)
// FILTER_SIZE must be defined in header file
// SIGMA also must be defined in header file


#include "header.h"

void gaussian_filtering(struct image *img)
{
        int i, j;
        float **filter=NULL;
        int nrows, ncols, trows, tcols, center;
        unsigned char **tmp_pad;

        nrows = img->height;      ncols = img->width;
        center= (FILTER_SIZE -1) /2;

        // create Gaussian filter of size="FILTER_SIZE"
           create_filter( &filter);

        // calling appropriate filtering routine
                if(FILTER_WITH_PADDING ==1)
                        filter_with_padding(&tmp_pad, filter, img);
                else if(FILTER_WITH_PADDING ==0)
                        filter_without_padding(filter, img);
                else
                {
                        dprintf(STDERR_FILENO, "Error : Invalid value for FILTER_WITH_PADDING in header file !!\n");
                        exit(5);
                }


        /*for(i=0; i< FILTER_SIZE; i++)
         {
                for(j=0; j< FILTER_SIZE; j++)  {
                        printf("%.2f ", filter[i][j]);
                }
                printf("\n");
         }*/

        // freeing memory
        for(i=0; i< FILTER_SIZE; i++)
                free( filter[i]  );
        free(filter);
        return;
}  // end of function


void filter_with_padding(unsigned char ***tmp_pad, float **filter, struct image *img)
{
        // first create a big matrix by padding original image at boundaries
           create_padded_matrix(tmp_pad, img);

        // now take conv(big_matrix, filter) and store in 'img'
           convolution(*tmp_pad, filter, img);

        return;
}

void create_padded_matrix(unsigned char ***tmp_pad, struct image *img)
{
        int C, D, E, trows, tcols;
```

```
    int i, j;
    int center;
    int nrows, ncols;

    nrows = img->height;
    ncols = img->width;
    center= (FILTER_SIZE -1) /2;
    // create tmp_pad image padded(at boundaries) with boundary elements of original image in approriate amount
    // tmp_pad image contains (FILTER_SIZE-1)/2 (==center) MORE rows and cols in each direction THAN original image
    //  so size of tmp image = (nrows + FILTER_SIZE -1, ncols + FILTER_SIZE -1)

    trows = nrows + FILTER_SIZE -1;
    tcols = ncols + FILTER_SIZE -1;
    // allocating memory
    check_memory( (*tmp_pad = (unsigned char **)malloc(trows *sizeof(unsigned char *)) ), "*tmp_pad");
    for(i=0; i <trows; i++)
            check_memory( ( (*tmp_pad)[i] = (unsigned char *)malloc(tcols *sizeof(unsigned char)) ), "tmp_pad[i]");

    // copying into tmp_pad
    C = center;
    D = ((trows-1)-center);
    E = ((tcols-1)-center);
    for(i=0; i< trows; i++)
    {
            for(j=0; j <tcols; j++)
            {
                    if( i < C || i > D || j < C || j > E )  // for these elements padding is necessary
                    {
                            if(ZERO_PADDING ==1)
                                    (*tmp_pad)[i][j] = 0;

                            // tmp_pad has 9 parts (directions) : NW, N, NE ;   W,E ; SW, S, SE
and Middle(copy of original)
                            // We use C, D, E to control
                            else if( !ZERO_PADDING )
                            {
                                    if(i<C && j<C)   // NW
                                            (*tmp_pad)[i][j] = img->pixel[0][0];
                                    else if(i<C && C<=j && j<=E )    // N
                                            (*tmp_pad)[i][j] = img->pixel[0][j-center];
                                    else if(i<C && j>E)      // NE
                                            (*tmp_pad)[i][j] = img->pixel[0][ncols-1];
                                    else if(C<=i && i<=D && j<C)     // W
                                            (*tmp_pad)[i][j] = img->pixel[i-center][0];
                                    else if(C<=i && i<=D && j>E)     // E
                                            (*tmp_pad)[i][j] = img->pixel[i-center][ncols-1];
                                    else if(i>D && j<C)      // SW
                                            (*tmp_pad)[i][j] = img->pixel[nrows-1][0];
                                    else if(i>D && C<=j && j<=E )    // S
                                            (*tmp_pad)[i][j] = img->pixel[nrows-1][j-center];
                                    else if(i>D && j>E )     // SE
                                            (*tmp_pad)[i][j] = img->pixel[nrows-1][ncols-1];
                                    else
                                    {
                                            dprintf(STDERR_FILENO, "Some error in applying padding for Gaussian filter !\n
                                            exit(5);
                                    }
                            }
                            else
                            {
                                    dprintf(STDERR_FILENO, "Error : Invalid value of \"ZERO_PADDING\" in header file !\n");
                                    exit(5);
                            }
                    }
```

```
                        else  // These elements are copied from original image
                                // Middle
                                // for tmp_pad(i,j) "center"rows and cols are counted more than img(i,j) on both top and left
                                (*tmp_pad)[i][j] = img->pixel [i-center][j-center];
                }
        }   // end for(i)
        return;
}



void convolution(unsigned char **tmp_pad, float **filter, struct image *img)
{
        int i, j, k, ii, jj, kk, ti, tj;
        int nrows, ncols, trows, tcols;
        int center;
        float avg;

        nrows = img->height;
        ncols = img->width;
        trows = nrows + FILTER_SIZE -1;  tcols = ncols + FILTER_SIZE -1;
        center= (FILTER_SIZE -1) /2;

        //filtering the image
        for(i=0; i< nrows; i++)
        {
                for(j=0; j< ncols; j++)
                {
                        ti = i+center; //corresponding indices of tmp_pad
                        tj = j+center;

                        avg = 0;
                        for(ii = ti-center, k=0; ii<= ti+center; ii++, k++)
                                for(jj = tj-center, kk=0; jj<= tj+center; jj++, kk++)
                                        avg = avg + tmp_pad[ii][jj] * filter[k][kk];

                        img->pixel[i][j] = (unsigned char)avg;

                }

        }  // for(i) ends

        // freeing tmp_pad's memory
        for(i=0; i <trows; i++)
                free(tmp_pad[i]);
        free(tmp_pad);
        return;
}


void filter_without_padding(float **filter, struct image *img)
{
        int i, j, k, ii, jj, kk;
        int nrows, ncols, center;
        float avg;
        unsigned char **tmp;  // to store a copy of img->pixel[][]

        nrows = img->height;
        ncols = img->width;
        center= (FILTER_SIZE -1) /2;

        // allocating memory
        check_memory( (tmp = (unsigned char **)malloc(nrows *sizeof(unsigned char *)) ), "tmp");
        for(i=0; i <nrows; i++)
                check_memory( (tmp[i] = (unsigned char *)malloc(ncols *sizeof(unsigned char))), "tmp[i]");
        // copying original pixels into tmp[][]
        for(i=0; i< nrows; i++)
                for(j=0; j< ncols; j++)
                        tmp[i][j] = img->pixel[i][j];

        // filtering
        for(i=0; i< nrows; i++)
        {
                for(j=0; j< ncols; j++)
                {
                        avg = 0;
                        //boundary elements as it is
                        if( i < center || i > ((nrows-1)-center) || j < center || j > ((ncols-1)-center) )
                        {
                                        ;        // do nothing
                        }

                        // convolution of appropriate element
                        else
                        {
                                for(ii = i-center, k=0; ii<= i+center; ii++, k++)
                                        for(jj = j-center, kk=0; jj<= j+center; jj++, kk++)
                                                avg = avg + tmp[ii][jj] * filter[k][kk];
                                img->pixel[i][j] = (unsigned char)avg;
                        }

                } // for(j) ends

        }  // for(i) ends
        return;
}

void create_filter(float ***filter_ptr)
{
        int i, j, center;
        float temp, t1, t2, t3;

        center= (FILTER_SIZE -1) /2;

        // allocating memory for filter
        check_memory( (*filter_ptr = (float **)calloc(FILTER_SIZE, sizeof(float *))), "*filter_ptr");
        for(i=0; i< FILTER_SIZE; i++)
                check_memory( ((*filter_ptr)[i] = (float *)calloc(FILTER_SIZE, sizeof(float ))), "*filter_ptr[i]");

        // calculating filter
        for(i=0; i< FILTER_SIZE; i++)
        {
                for(j=0; j< FILTER_SIZE; j++)
                {
                        t1 =  pow(i-center, 2) ;
                        t2 =  pow(j-center, 2) ;
                                        //printf("t1=%f\nt2=%f\n", t1, t2);
                        t3 = (-1/2.0)*  ( 1.0/ ( (float)SIGMA*(float)SIGMA ) )  * ( t1 + t2 );
                                        //printf("t3=%f\n", t3);
                        temp = exp(t3)  ;
                                        //printf("temp=%f\n",temp);
                        (*filter_ptr)[i][j] =  temp / ( 2* PI* SIGMA* (float)SIGMA  )     ;
                }
        }
        /*for(i=0; i< FILTER_SIZE; i++)
        {
                for(j=0; j< FILTER_SIZE; j++)
                {
                        printf("%f    ", (*filter_ptr)[i][j]);
                }
                printf("\n");
        }*/

        return;
}
```

A.14 GLCMcalcandnormalize.c

```
#include "header.h"


// start of function 'calculateGLCM'
/*
   "angle" value                    Angle
   0                                 0
   1                                -45
   2                                -90
   3                                -135
   4                                -180
   5                                +135
   6                                +90
   7                                +45
   */

/* symmetric: 1 for symmetric version  0 for non-symmetric version */
void calculateGLCM(struct gLCM *cptr, struct image *img, int angle, int dist, int sym)
{
        int i,j,k;
        int dir_x, dir_y;
        int pref, pnbd;

        if(angle <0 || angle >7)
        {
                dprintf(STDERR_FILENO, "Error : \n");
                dprintf(STDERR_FILENO, "Incorrect angle(=%d) to calculate GLCM ! \n", angle);
                exit(1);
        }
        cptr->start_idx = img->pmin;
        cptr->end_idx = img->pmax;
        cptr->size = cptr->end_idx - cptr->start_idx + 1 ;

        // memory allocation for co-occurence matrix
        check_memory( (cptr->co_ocr = (float **)calloc(cptr->size, sizeof(float *)) ), "cptr->co_ocr");
        for(i=0; i< cptr->size; i++)
                check_memory( (cptr->co_ocr[i] = (float *)calloc(cptr->size, sizeof(float))), "cptr->co_ocr[i]");

        switch(angle)
        {
                case 0:
                        dir_x = 0;
                        dir_y = 1;
                case 1:
                        dir_x = 1;
                        dir_y = 1;
                case 2:
                        dir_x = 1;
                        dir_y = 0;
                case 3:
                        dir_x = 1;
                        dir_y = -1;
                case 4:
                        dir_x = 0;
                        dir_y = -1;
                case 5:
                        dir_x = -1;
                        dir_y = -1;
                case 6:
                        dir_x = -1;
                        dir_y = 0;
                case 7:
                        dir_x = -1;
                        dir_y = 1;
        }

        dir_x = dir_x*dist;
        dir_y = dir_y*dist;

        if(sym == 0)  // i.e. non-symmetric
        {
                for(i=0; i< img->height; i++)
                {
                        for(j=0; j< img->width; j++)
                        {
                                if((i+dir_x >=0) && (i+dir_x < img->height) && (j+dir_y >= 0) && (j+dir_y < img->width))
                                {
                                        pref = img->pixel[i][j];
                                        pnbd = img->pixel[i+dir_x][j+dir_y];
                                        (cptr->co_ocr[ pref- (cptr->start_idx)][ pnbd - (cptr->start_idx)])++;
                                        // index starts from 0, and our origin is shiftd to (pmin, pmin)
                                }
                        }
                }
        }

        else if(sym == 1)
        {
                for(i=0; i< img->height; i++)
                {
                        for(j=0; j< img->width; j++)
                        {
                                if((i+dir_x >=0)&& (i+dir_x < img->height)&& (j+dir_y >= 0)&& (j+dir_y < img->width))
                                {
                                        pref = img->pixel[i][j];
                                        pnbd = img->pixel[i+dir_x][j+dir_y];
                                        (cptr->co_ocr [pref- (cptr->start_idx)] [pnbd- (cptr->start_idx)] )++;
                                        //index starts from 0,origin is shiftd to(min,min)


                                        (cptr->co_ocr [pnbd- (cptr->start_idx)] [pref- (cptr->start_idx)] )++;
                                        //contributes to both (i,j) and (j,i)
                                }
                        }
                }
        }

        else
        {
                dprintf(STDERR_FILENO, "Error : \n");
                dprintf(STDERR_FILENO, "Wrong input %d for co-occurence  matrix !\n", sym);
                dprintf(STDERR_FILENO, "Valid inputs are 0 (for non-sym) and 1 (for sym).\n");
                exit(1);
        }

} // end calculateGLCM


void normalizeGLCM (struct gLCM *cptr)
{
        int i,j;
        float nsum = 0;

        for( i=0; i< cptr->size; i++)
                for(j=0; j< cptr->size; j++)
                        nsum = nsum + cptr->co_ocr[i][j] ;
        //printf("\nNormalizarion Sum = %f\n", nsum);
        for( i=0; i< cptr->size; i++)
                for(j=0; j< cptr->size; j++)
                        cptr->co_ocr[i][j] =  ( cptr->co_ocr[i][j] / nsum ) ;   //normalization

        return ;

}
// end normalizeGLCM

// calculates avg GLCM of 4 GLCMs
void  calculateAvgGLCM(struct gLCM *cptr, struct gLCM *cptr1,struct gLCM *cptr2,struct gLCM *cptr3,struct gLCM *cptr4)
{
        int i, j;
        float sum;

        if( (cptr1->size != cptr2->size) || (cptr1->size != cptr3->size)|| (cptr1->size != cptr4->size)|| (cptr2->size != cptr3
        {
                dprintf(STDERR_FILENO, "Error:\n");
                dprintf(STDERR_FILENO, "Sizes of 4 co-occurence matrices is not same !\n");
                exit(4);
        }

        cptr->start_idx = cptr1->start_idx;
          cptr->end_idx = cptr1->end_idx;
        cptr->size = cptr->end_idx - cptr->start_idx + 1 ;

        // memory allocation for co-occurence matrix
        check_memory( (cptr->co_ocr = (float **)calloc(cptr->size, sizeof(float *)) ), "cptr->co_ocr");
        for(i=0; i< cptr->size; i++)
                check_memory( (cptr->co_ocr[i] = (float *)calloc(cptr->size, sizeof(float)) ), "cptr->co_ocr[i]");

        for(i=0; i< cptr1->size; i++)
        {
                for(j=0; j< cptr1->size; j++)
                {
                        sum = (cptr1->co_ocr[i][j] + cptr2->co_ocr[i][j] + cptr3->co_ocr[i][j] + cptr4->co_ocr[i][j]);
                        cptr->co_ocr[i][j] = sum/4;
                }
        }


        // printing to check
        /*
                for(i=0; i< cptr1->size; i++)
                {
                        printf("GLCM-1:\n");
                        for(j=0; j< cptr1->size; j++)
                                printf("%.2f  ", cptr1->co_ocr[i][j]);
                        printf("\n");

                        printf("GLCM-2:\n");
                        for(j=0; j< cptr2->size; j++)
                                printf("%.2f  ", cptr2->co_ocr[i][j]);
                        printf("\n");

                        printf("GLCM-3:\n");
                        for(j=0; j< cptr3->size; j++)
                                printf("%.2f  ", cptr3->co_ocr[i][j]);
                        printf("\n");

                        printf("GLCM-4:\n");
                        for(j=0; j< cptr4->size; j++)
                                printf("%.2f  ", cptr4->co_ocr[i][j]);
                        printf("\n");
```

```
                    printf("GLCM-avg:\n");
                    for(j=0; j< cptr->size; j++)
                            printf("%.2f  ", cptr->co_ocr[i][j]);

                    printf("\n------------------------------------------------------------------------------------------------
            }
    */


    // freeing the memory of all four GLCMs
    for(i=0; i< cptr1->size; i++)
    {
            free(cptr1->co_ocr[i]);
            free(cptr2->co_ocr[i]);
            free(cptr3->co_ocr[i]);
            free(cptr4->co_ocr[i]);
    }
    free(cptr1->co_ocr);
    free(cptr2->co_ocr);
    free(cptr3->co_ocr);
    free(cptr4->co_ocr);

    return;
}
```

A.15 glcmfeatures.c

```
// Calculation of GLCM based features

#include "header.h"


void calc_GLCM_Features(struct gLCM *cptr, struct features *fptr)
{
        int i,j;
        float *sum=NULL;
        int tot=0;

        //initialization;
        fptr->con=0;
        fptr->dis=0;
        fptr->hom=0;
        fptr->ang_sm=0;
        fptr->entr=0;
        fptr->gLCM_mean=0;
        fptr->sum_entr = 0;

        // for 'sum_entr'
          tot = (2 * (cptr->size) ) -2;
          check_memory( (sum = (float *)calloc( tot+1, sizeof(float)) ), "sum");

        for(i=0; i< cptr->size; i++)
        {
                for(j=0; j< cptr->size; j++)
                {
                        fptr->con = fptr->con + ( (i-j) * (i-j) * cptr->co_ocr[i][j] ) ;
                        fptr->dis = fptr->dis + ( abs(i-j) * cptr->co_ocr[i][j] ) ;
                        fptr->hom = fptr->hom + ( cptr->co_ocr[i][j] / ( 1+ ( (i-j)*(i-j) ) )  );

                        fptr->ang_sm = fptr->ang_sm + ( cptr->co_ocr[i][j] * cptr->co_ocr[i][j] )
;

                        if( cptr->co_ocr[i][j] )  //if non-zero
                                fptr->entr = fptr->entr + (cptr->co_ocr[i][j] * (-1) * log(cptr->co_ocr[i][j]));

                        fptr->gLCM_mean = fptr->gLCM_mean + ( i * cptr->co_ocr[i][j] ); //ref_mean
                        // fptr->gLCM_mean = fptr->gLCM_mean + ( j * cptr->co_ocr[i][j] );  // nbd_mean

                        sum[i+j] += cptr->co_ocr[i][j];
                }
        }

        // calculating sum entropy
        for(i=0; i<= tot; i++)
        {
                if( sum[i] )   // assuming log(0) = 0
                        fptr->sum_entr += ( sum[i] * (-1) * log (sum[i])   ) ;
        }
#ifdef DEBUG
        for(i=0; i<= tot; i++)
                printf("sum[%d] = %f\n", i, sum[i]);
#endif

        // freeing memory
        free(sum);

        // calculate remaining features
        calc_Enrg_MaxProb_GLCMMean(cptr, fptr);
        calc_Stddev_GLCMCorr(cptr, fptr);
        return;
}

void calc_Enrg_MaxProb_GLCMMean(struct gLCM *cptr, struct features *fptr)
{
        int i, j;

        //initialization
        fptr->enrg=0;
        fptr->max_prob=cptr->co_ocr[0][0];
        fptr->gLCM_var=0;

        fptr->enrg = sqrt (fptr->ang_sm) ;
        for(i=0; i< cptr->size; i++)
        {
                for(j=0; j< cptr->size; j++)
                {

                        fptr->max_prob= (cptr->co_ocr[i][j] > fptr->max_prob) ? cptr->co_ocr[i][j] : fptr->max_prob;
                        fptr->gLCM_var += (cptr->co_ocr[i][j]*(i-fptr->gLCM_mean)*(i-fptr->gLCM_mean) );

                }
        }

        return;
}

void calc_Stddev_GLCMCorr(struct gLCM *cptr, struct features *fptr)
{
        int i, j;
        int numer, denom;

        //initialization
        fptr->gLCM_std_dev=0;
        fptr->gLCM_corr=0;

        fptr->gLCM_std_dev = sqrt( fptr->gLCM_var );
        for(i=0; i< cptr->size; i++)
        {
                for(j=0; j< cptr->size; j++)
                {
                        numer = (i- fptr->gLCM_mean) *  ( j- fptr->gLCM_mean );
                        denom =  fptr->gLCM_var ;

                        if(denom)  //for var=0, denom may vanish; (e.g. for fully identical data)
                                fptr->gLCM_corr = fptr->gLCM_corr + ( cptr->co_ocr[i][j] * (numer/denom) ) ;
                        else
                        {
                                dprintf(STDERR_FILENO,"Error : (Handled as follows)\n");
                                dprintf(STDERR_FILENO,"Variance =0;\nCo-variance =Infinity ,Set to a large number \'%d\' !\n",IN
                                fptr->gLCM_corr = (float)INT_MAX;
                        }
                }
        }

        return;
}
```

A.16 nffgrnflsd.c

```
#include "header.h"

void calc_NFFGR(struct image *nf, struct image *ff, struct features *fptr)
{
        int i, j, k;
        int nf_sum=0, ff_sum=0 ;

        for(i=0; i< nf->height; i++)
                for(j=0; j< nf->width; j++)
                        nf_sum += nf->pixel[i][j];

        for(i=0; i< ff->height; i++)
                for(j=0; j< ff->width; j++)
                        ff_sum += ff->pixel[i][j];

        if( ff_sum ) // ff_sum is denominator
                fptr->nFFGR = ( (float) nf_sum / ff_sum );

        else // can't divide by zero
        {
                dprintf(STDERR_FILENO, "Error : (Handled as follows)\n");
                dprintf(STDERR_FILENO, "FULLY BLACK (ZERO) FAR-FIELD !\nAdding 1 to  calculate \'nFFGR\' \n");

                ff_sum += 1;
                fptr->nFFGR = ( (float) nf_sum / ff_sum );
        }

        return ;
}


void calc_NFLSD(struct image* img, struct features *fptr)
{
        int i, j, k, ii, jj, kk ;
        int **mdf_img;
        int avg=0;
        int label=0;
        int **label_matrx;
        int nw, nor, ne, wst, est, sw, sou, se; //  value is -1, 0, 1 for -NA-, unmatched and matched
        int no_match_nbrs=0;
        int nrows = img->height, ncols = img->width;
        int flag_nbrs=0; // all nbrs have been processed or not


        check_memory( (mdf_img = (int **) malloc ( nrows *sizeof(int *) ) ), "mdf_img");
        for (i=0; i< nrows ; i++)
                check_memory( (mdf_img[i] = (int *) calloc ( ncols, sizeof(int) )), "mdf_img[i]");

        check_memory( (label_matrx = (int **) malloc ( nrows *sizeof(int *) )), "label_matrx");
        for (i=0; i< nrows ; i++)
                check_memory( (label_matrx[i] = (int *) calloc ( ncols, sizeof(int) )), "label_matrx[i]");

        // applying high-pass filter
        for(i=0; i< nrows; i++)
        {
                for(j=0; j< ncols; j++)
                {
                        avg = 0;
                        if(i==0 || i==1 || i==nrows-1 || i==nrows-2 || j==0 || j==1 || j==ncols-1 || j==ncols-2)
                        {
                                mdf_img[i][j] = img->pixel[i][j];   //boundary elements as it is
                                //thresholding
                                if(mdf_img[i][j] >= NFLSD_THRESHOLD)
                                        mdf_img[i][j] = 255;
                                else
                                        mdf_img[i][j] = 0;
                        }

                        // convolution of appropriate element
                        else
                        {
                                for(ii = i-2, k=0; ii<= i+2; ii++, k++)
                                        for(jj = j-2, kk=0; jj<= j+2; jj++, kk++)
                                                avg = avg + img->pixel[ii][jj] * high_pass_filter[k][kk];

                                mdf_img[i][j] = avg;

                                //thresholding
                                if(mdf_img[i][j] >= NFLSD_THRESHOLD)
                                        mdf_img[i][j] = 255;
                                else
                                        mdf_img[i][j] = 0;
                        }
                }
        } // end for(i)


        // Initializing label matrix
        for(i=0; i< nrows; i++)
                for(j=0; j< ncols; j++)
                        label_matrx[i][j] = 0;


        //check_nbrs(mdf_img, &nw, &nor, &ne, &wst, &est, &sw, &sou, &se, i, j, nrows, ncols);
        for(i=0; i< nrows; i++)
        {
                for(j=0; j< ncols; j++)
                {
                        flag_nbrs = 0;
                        if( label_matrx[i][j] )
                        {
                                check_nbrs(mdf_img, &nw, &nor, &ne, &wst, &est, &sw, &sou, &se, i, j, nrows, ncols);
                                assign_nbrs(label_matrx, i, j, nw, nor, ne, wst, est, sw, sou, se, label);
                        }
                        else
                        {
                                check_nbrs(mdf_img, &nw, &nor, &ne, &wst, &est, &sw, &sou, &se, i, j, nrows, ncols);

                                if(  (nw ==1)  && (flag_nbrs == 0) )
                                        flag_nbrs = func( label_matrx, i, j, i-1, j-1, nw, nor, ne, wst, est, sw, sou, se, &lab

                                if(  (nor ==1) && (flag_nbrs ==0)  )
                                        flag_nbrs = func( label_matrx, i, j, i-1, j, nw, nor, ne, wst, est, sw, sou, se, &label

                                if(  (ne ==1) && (flag_nbrs ==0)  )
                                        flag_nbrs = func( label_matrx, i, j, i-1, j+1, nw, nor, ne, wst, est, sw, sou, se, &lab

                                if(  (wst ==1) && (flag_nbrs ==0)  )
                                        flag_nbrs = func( label_matrx, i, j, i, j-1, nw, nor, ne, wst, est, sw, sou, se, &label

                                if(  (est ==1) && (flag_nbrs ==0) )
                                        flag_nbrs = func( label_matrx, i, j, i, j+1, nw, nor, ne, wst, est, sw, sou, se, &label

                                if(  (sw ==1) && (flag_nbrs ==0) )
                                        flag_nbrs = func( label_matrx, i, j, i+1, j-1, nw, nor, ne, wst, est, sw, sou, se, &lab
                                if(  (sou ==1) && (flag_nbrs ==0) )
                                        flag_nbrs = func( label_matrx , i, j, i+1, j, nw, nor, ne, wst, est, sw, sou, se, &label

                                if(  (se ==1) && (flag_nbrs ==0) )
                                        flag_nbrs = func( label_matrx , i, j, i+1, j+1, nw, nor, ne, wst, est, sw, sou, se, &lab

                                if( !flag_nbrs )   //no matched nbr is assigned any label, so create new label
                                {
                                        label++;
                                        label_matrx[i][j] = label;
                                        assign_nbrs(label_matrx, i, j, nw, nor, ne, wst, est, sw, sou, se, label);
                                }

                        } //end else
                }
        }  // end for(i)

        fptr->nFLSD = label;

        // freeing memory
        for (i=0; i< nrows ; i++)
                free( mdf_img[i] );
        free(mdf_img);

        for (i=0; i< nrows ; i++)
                free (label_matrx[i] );
        free(label_matrx);

        return;
}
```

A.17 chkasgnNbrs.c

```
#include "header.h"

void assign_nbrs( int **label_matrx, int i, int j,\
                                int nw, int nor, int ne, int wst, int est,\
                                int sw, int sou, int se, int label)
{
        if(nw ==1)
                label_matrx[i-1][j-1] = label;
        if(nor ==1)
                label_matrx[i-1][j] = label;
        if(ne ==1)
                label_matrx[i-1][j+1] = label;
        if(wst ==1)
                label_matrx[i][j-1] = label;
        if(est ==1)
                label_matrx[i][j+1] = label;
        if(sw ==1)
                label_matrx[i+1][j-1] = label;
        if(sou ==1)
                label_matrx[i+1][j] = label;
        if(se ==1)
                label_matrx[i+1][j+1] = label;
        return;
}

// stores in corresponding nbr, the values of..
// 0 for unmatched, 1 for matched and -1 for not-applicable
int check_nbrs(int **img, int* nw, int* nor, int* ne,\
                                int* wst, int* est,\
                                int* sw, int* sou, int* se,\
                                int i, int j, int nrows, int ncols)
{
        int no_match_nbrs =0;

        if(i==0)        //first row : nw, nor, ne are -na-
        {
                *nw  = -1;
                *nor = -1;
                *ne  = -1;
                if(j==0)        //[0][0]th element : wst & sw -na-
                {
                        *wst = -1;
                        *sw  = -1;
                        *est = (img[i][j] == img[i][j+1]) ? 1 : 0 ;
                        *sou = (img[i][j] == img[i+1][j]) ? 1 : 0 ;
                        *se  = (img[i][j] == img[i+1][j+1]) ? 1 : 0 ;
                }
                else if(j == ncols-1) //  est, se are -na-
                {
                        *est = -1;
                        *se  = -1;
                        *wst = (img[i][j] == img[i][j-1]) ? 1 : 0 ;
                        *sw = (img[i][j] == img[i+1][j-1]) ? 1 : 0 ;
                        *sou  = (img[i][j] == img[i+1][j]) ? 1 : 0 ;
                }
                else
                {
                        *wst = (img[i][j] == img[i][j-1]) ? 1 : 0 ;
                        *est = (img[i][j] == img[i][j+1]) ? 1 : 0 ;
                        *sw = (img[i][j] == img[i+1][j-1]) ? 1 : 0 ;
                        *sou = (img[i][j] == img[i+1][j]) ? 1 : 0 ;
                        *se  = (img[i][j] == img[i+1][j+1]) ? 1 : 0 ;
        }
}
else if(i == nrows-1)  // last row : sw, sou, se are -na-
{
        *sw  = -1;
        *sou = -1;
        *se  = -1;
        if(j==0)          //wst & nw -na-
        {
                *wst = -1;
                *nw  = -1;
                *nor = (img[i][j] == img[i-1][j]) ? 1 : 0 ;
                *ne  = (img[i][j] == img[i-1][j+1]) ? 1 : 0 ;
                *est = (img[i][j] == img[i][j+1]) ? 1 : 0 ;
        }
        else if(j == ncols-1) //  ne, est are -na-
        {
                *ne  = -1;
                *est = -1;
                *nw =  (img[i][j] == img[i-1][j-1]) ? 1 : 0 ;
                *nor = (img[i][j] == img[i-1][j]) ? 1 : 0 ;
                *wst = (img[i][j] == img[i][j-1]) ? 1 : 0 ;
        }
        else
        {
                *nw =  (img[i][j] == img[i-1][j-1]) ? 1 : 0 ;
                *nor = (img[i][j] == img[i-1][j]) ? 1 : 0 ;
                *ne  = (img[i][j] == img[i-1][j+1]) ? 1 : 0 ;
                *wst = (img[i][j] == img[i][j-1]) ? 1 : 0 ;
                *est = (img[i][j] == img[i][j+1]) ? 1 : 0 ;
        }
}
else  // rest all rows
{
        if(j==0)          // nw, wst and sw are -na-
        {
                *nw  = -1;
                *wst = -1;
                *sw  = -1;
                *nor = (img[i][j] == img[i-1][j]) ? 1 : 0 ;
                *ne  = (img[i][j] == img[i-1][j+1]) ? 1 : 0 ;
                *est = (img[i][j] == img[i][j+1]) ? 1 : 0 ;
                *sou = (img[i][j] == img[i+1][j]) ? 1 : 0 ;
                *se  = (img[i][j] == img[i+1][j+1]) ? 1 : 0 ;
        }
        else if(j == ncols-1) // ne, est and se are -na-
        {
                *ne  = -1;
                *est = -1;
                *se  = -1;
                *nw =  (img[i][j] == img[i-1][j-1]) ? 1 : 0 ;
                *nor = (img[i][j] == img[i-1][j]) ? 1 : 0 ;
                *wst = (img[i][j] == img[i][j-1]) ? 1 : 0 ;
                *sw = (img[i][j] == img[i+1][j-1]) ? 1 : 0 ;
                *sou = (img[i][j] == img[i+1][j]) ? 1 : 0 ;
        }
        else // all are applicable
        {
                *nw =  (img[i][j] == img[i-1][j-1]) ? 1 : 0 ;
                *nor = (img[i][j] == img[i-1][j]) ? 1 : 0 ;
                *ne  = (img[i][j] == img[i-1][j+1]) ? 1 : 0 ;
                *wst = (img[i][j] == img[i][j-1]) ? 1 : 0 ;
                *est = (img[i][j] == img[i][j+1]) ? 1 : 0 ;
                *sw = (img[i][j] == img[i+1][j-1]) ? 1 : 0 ;
                *sou = (img[i][j] == img[i+1][j]) ? 1 : 0 ;
```

```
                        *se  = (img[i][j] == img[i+1][j+1]) ? 1 : 0 ;
                }
        }

        return 0;
}

int func (int **label_matrx, int i, int j, int ineb, int jneb,\
                        int nw, int nor, int ne, int wst, int est,\
                        int sw, int sou, int se, int *label)
{
        if (label_matrx[ineb][jneb])
        {
                label_matrx[i][j] = label_matrx[ineb][jneb];
                assign_nbrs(label_matrx, i, j, \
                                        nw, nor, ne, wst, est, \
                                        sw, sou, se, label_matrx[ineb][jneb]);
                return 1;
        }
        return 0;
}
```

A.18 misc.c

```
#include "header.h"

void calc_pmin_pmax(struct image *img)
{
        int i, j, k;

        img->pmin=255;
        img->pmax=0;
        for (i=0; i < img->height; i++)
        {
                for (j=0; j < img->width; j++)
                {
                        if( (unsigned int)img->pixel[i][j] >= img->pmax )
                                img->pmax = (unsigned int)img->pixel[i][j] ;
                        if( (unsigned int)img->pixel[i][j] <= img->pmin )
                                img->pmin = (unsigned int)img->pixel[i][j] ;
                }
        }
        return;

}

void check_memory(void *mem, char *var_name)
{
        if (mem==NULL)
        {
                perror(var_name);
                exit(EXIT_FAILURE);
        }
        return;
}


void check_file(FILE *fp, char *filename)
{
        if(fp==NULL)
        {
                perror(filename);
                exit(EXIT_FAILURE);
        }
        return;
}

 void store_features(FILE *fp, struct features *feat, int class)
{
        // first entry is class label
         fprintf( fp,"%d ", class);

         fprintf( fp,"%f ", feat->con);
         fprintf( fp,"%f ", feat->dis);
         fprintf( fp,"%f ", feat->hom);

         fprintf( fp,"%f ",feat->ang_sm);
         fprintf( fp,"%f ",feat->enrg);
         fprintf( fp,"%f ",feat->max_prob);
         fprintf( fp,"%f ",feat->entr);
         fprintf( fp,"%f ",feat->sum_entr);

         fprintf( fp,"%f ", feat->gLCM_mean);
         fprintf( fp,"%f ", feat->gLCM_var);
         fprintf( fp,"%f ", feat->gLCM_std_dev);
         fprintf( fp,"%f ", feat->gLCM_corr);
        fprintf( fp,"%d ", feat->nFLSD);
        fprintf( fp,"%f ", feat->nFFGR);

        fprintf( fp,"\n");

        return ;
}
```

A.19 crop.c

```
// Given an input pgm image and size, output file....
//              will be a "size x size" pgm image which is input image cropped at center
//

#include "header.h"

FILE *write_cropped_image(struct image *, FILE *, int );

int main(int argc, char **argv)
{
        int i, j, valid;
        int crop_size;
        struct image img;
        char infile[MAX_FILENAME_LENGTH], outfile[MAX_FILENAME_LENGTH];
        FILE *infp, *outfp;

        if(argc != 4)
        {
                dprintf(STDERR_FILENO,"Usage : —%s—input_file —out_file —crop_size\n", argv[0]);
                exit(1);
        }
        strcpy(infile, argv[1]);
        strcpy(outfile, argv[2]);
        crop_size=atoi(argv[3]);

        check_file ( (infp=fopen(infile, "r") ), infile);
        isValidPGM(infp, &img, infile);
        readPGM(infp, &img);

        check_file( (outfp=fopen(outfile, "w") ), outfile);

        // image cropping function
        outfp = write_cropped_image(&img, outfp, crop_size);

        fclose(infp);
        fclose(outfp);

        return 0;
}

FILE *write_cropped_image(struct image *img, FILE *outfp, int crop_size)
{
        int i, j, k;
        int margin = (img->width - crop_size)/2;

        // writing header
        fprintf(outfp, "%s\n%d %d\n%d\n", img->type, crop_size, crop_size,  img->pmaxval);

        // writing pixels
        for(i=margin; i< (img->width -margin); i++)
        {
                for(j=margin; j< (img->height -margin); j++)
                {
                        if( !( strcmp (img->type, "P5") ))  // if raw-pgm
                                fprintf(outfp, img->inStr, img->pixel[i][j]);
                        else  // if ascii-pgm (an additional newline)
                        {
                                fprintf(outfp, img->inStr, img->pixel[i][j]);
                                fprintf(outfp,"\n");
                        }
                }
        }
        return outfp;
}
```

A.20 normalizefeaturesmaxmin.c

```
#include "header.h"

// Data normalizer
// first column is treated as class label
// Data must be space-delimited
// Newline character must occur at end of line
//
// Formula used :   x = (x-min) / (max-min) ;

float findMax(float **, int , int );
float findMin(float **, int , int );

int main(int argc, char **argv)
{

        FILE *infp =NULL, *outfp =NULL;
        int i, j, k;
        float *max_arr,*min_arr;
        float tmp;
        float **set;
        int size, no_of_feat;
        int *class;

        if(argc != 5)
        {
                dprintf(STDERR_FILENO, "Error:\n");
                dprintf(STDERR_FILENO,"Usage: %s  Input_file  Size  No_of_features  Output_File\n", argv[0]);
                exit(1);
        }

        size = atoi(argv[2]);
        no_of_feat = atoi(argv[3]);

        // mem alloc
        check_memory( (class = (int *)calloc(size, sizeof(int)) ), "class");
        check_memory( (set=(float **)calloc(size, sizeof(float*)) ), "set");
        for(i=0; i< size; i++)
                check_memory( (set[i] = (float *)calloc(no_of_feat, sizeof(float)) ), "set[i]");
        check_memory( (max_arr = (float *)calloc(no_of_feat, sizeof(float)) ), "max_arr");
        check_memory( (min_arr = (float *)calloc(no_of_feat, sizeof(float)) ), "min_arr");


        check_file( (infp = fopen(argv[1], "r")) , argv[1]);
        check_file( (outfp = fopen(argv[4], "w") ), argv[4]);

        // reading features
        i=0;
        while ( i< size )
        {
                fscanf(infp, "%d", &class[i]); //first column is treated as class label
                for (j=0; j< no_of_feat; j++)
                        fscanf(infp, "%f", &set[i][j] ) ;
                i++;
        }
        fclose(infp);

        // find max and min for each feature
        for(j=0; j< no_of_feat; j++)
        {
                max_arr[j] = findMax(set, j, size); // max for jth feature
                min_arr[j] = findMin(set, j, size); // min for jth feature
                        //printf("max[%d]  =  %f\n", j, max_arr[j]);
                //printf("min[%d]  =  %f\n", j, min_arr[j]);
        }

        // writing normalized feature values to output file
        i=0;
        while ( i< size )
        {
                fprintf(outfp, "%d ", class[i]);
                for (j=0; j< no_of_feat; j++) // writing each normalized data point
                {
                   if( max_arr[j] - min_arr[j] )
                   {
                      tmp = ( set[i][j] - min_arr[j] ) / ( max_arr[j] - min_arr[j] );
                      fprintf(outfp, "%f ", tmp ) ;
                   }
                   else  // all values are same -- ( max == min == all_values )
                   {
                      tmp = 0;
                      fprintf(outfp, "%f ", tmp ) ;
                   }

                } //end for(j)
                fprintf(outfp,"\n"); //new-line at the end of each data point
                i++;
        }
        fclose(outfp);

        // freeing up memory
        free(max_arr);
        free(min_arr);
        free(class);
        for(i=0; i< size; i++)
                free(set[i]);
        free(set);

}

float findMax(float **set, int j, int size)
{
        float max;
        int i;

        max = set[0][j];
        for(i=1; i< size; i++)
        {
                if( set[i][j] > max)
                        max = set[i][j];
        }

        return max;
}


float findMin(float **set, int j, int size)
{
        float min;
        int i;

        min = set[0][j];
        for(i=1; i< size; i++)
        {
                if( set[i][j] < min)
                        min = set[i][j];
        }
        return min;
}
```

A.21 normalizefeaturesmeanstd.c

```
#include "header.h"

// Data normalizer
// first column is treated as class label
// Data must be space-delimited
// Newline character must occur at end of line
//
// Formula used :   x = (x-mean) / standard_dev ;
//

float calcMean(float **, int , int );
float calcStdDev(float **, int , int, float );

int main(int argc, char **argv)
{

        FILE *infp =NULL, *outfp =NULL;
        int i, j, k;
        float *mean_arr,*stdDev_arr;
        float tmp;
        float **set;
        int size, no_of_feat;
        int *class;

        if(argc != 5)
        {
                dprintf(STDERR_FILENO, "Error:\n");
                dprintf(STDERR_FILENO,"Usage: %s  Input_file  Size  No_of_features  Output_File\n", argv[0]);
                exit(1);
        }

        size = atoi(argv[2]);
        no_of_feat = atoi(argv[3]);

        // mem alloc
        check_memory( (set=(float **)calloc(size, sizeof(float*))), "set");
        for(i=0; i< size; i++)
                check_memory( (set[i] = (float *)calloc(no_of_feat, sizeof(float)) ), "set[i]");
        check_memory( (class = (int *)calloc(size, sizeof(int)) ), "class");
        check_memory( (mean_arr = (float *)calloc(no_of_feat, sizeof(float)) ), "mean_arr") ;
        check_memory( (stdDev_arr = (float *)calloc(no_of_feat, sizeof(float))), "stdDev_arr");

        check_file( (infp = fopen(argv[1], "r")) , argv[1]);
        check_file( (outfp = fopen(argv[4], "w")), argv[4]);

        // Reading input file into "set"
        i=0;
        while ( i< size )
        {
                fscanf(infp, "%d", &class[i]); //first column is treated as class label
                for (j=0; j< no_of_feat; j++)
                        fscanf(infp, "%f", &set[i][j] ) ;
                i++;
        }
        fclose(infp);

        // calc mean and std_dev
        for(j=0; j< no_of_feat; j++)
        {
                mean_arr[j] = calcMean(set, j, size); // max for jth feature
                stdDev_arr[j] = calcStdDev(set, j, size, mean_arr[j]); // min for jth feature
                //printf("mean[%d]  =  %f\t\t", j, mean_arr[j]);
                //printf(" stdDev[%d]  =  %f\n", j, stdDev_arr[j]);
        }

        // writing normalized feature values to file
        i=0;
        while ( i< size )
        {
                fprintf(outfp, "%d ", class[i]);
                for (j=0; j< no_of_feat; j++) // writing each normalized data point
                {
                        if( stdDev_arr[j] )
                        {
                                tmp = ( set[i][j] - mean_arr[j] ) / stdDev_arr[j] ;
                                fprintf(outfp, "%f ", tmp ) ;
                        }
                        else
                        {
                                dprintf(STDERR_FILENO,"Error : (Handled as follows)\n");
                                dprintf(STDERR_FILENO,"StdDev==0; Cant divide by 0,Adding 0.1 in denom to divide.\n");
                                tmp = ( set[i][j] - mean_arr[j] ) / (stdDev_arr[j] + 0.1) ;
                                fprintf(outfp, "%f ", tmp ) ;
                        }

                } //end for(j)
                fprintf(outfp,"\n"); //new-line at the end of each data point
                i++;
        }
        fclose(outfp);

        // freeing up memory
        free(mean_arr);
        free(stdDev_arr);
        free(class);
        for(i=0; i< size; i++)
                free(set[i]);
        free(set);

        return 0;
}

float calcMean(float **set, int j, int size)
{
        int i;
        float sum=0;

        for(i=0; i< size; i++) // loop for data set
                sum += set[i][j];

        return (sum / size);
}

float calcStdDev(float **set, int j, int size, float mean)
{
        int i;
        float tmp=0, std_dev=0;

        for(i=0; i< size; i++)
                tmp += ( (set[i][j] - mean) * (set[i][j] - mean) );

        std_dev = sqrt( tmp/size );

        return std_dev;
}
```

A.22 genalldatasets.sh

```
#!/bin/sh

###########################################################################

# Input  : A dataset file with 'N' features[on command line or by user input]

#              ——> After using "cut" a single space is added before each newline char

# Outfile_names : Files are named as "${ofile}_underscore_seperated_feature_list"

# Output : 2^N files, where each file contains one subset feature data

###########################################################################

odir=""
otag="d"
limit=15;
ofile="${otag}"
feat_list="1 2 3 4 5 6 7 8 9 10 11 12 13 14"

# usage : ---./gen_all_datasets.sh——output_dir——base_dataset——
##----output directory and Input datafile
if [ $# -eq 2 ]
then
        odir=$1
        dfile=$2  # if entered at command line
else
        echo -n "Enter output directory : "
        read odir
        echo -n "Enter dataset file : "
        read dfile
fi
##--
if [ ! -e $dfile ]
then
        echo; echo "Error : Cant open file \"$dfile\" .";
                  echo "Perhaps it doesn't exist."; echo
        exit 1;
fi
##--
if [ -d ${odir}/1 ]
then
        echo "Delete all files in \"$odir/[1-14]\" ?  (y|n)"
        #read ques
        ques="y"
        if [ $ques = "y" -o $ques = "Y" ]
        then
                rm -f ${odir}/*/${ofile}*
        else
                echo "Exiting !!"; echo
                exit 2;
        fi
else
        rm -rf ${odir}/*
        cd ${odir}
        mkdir `echo ${feat_list}`
        cd ..
fi
#######################################################################
## Nested loop starting :

        for ((x1=2; x1<= limit; x1++))
        do
                        depth=1;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}"

                        cut -d " " -f 1,$x1 $dfile >| $outfile
                        # To append trailing space before newline char
                          perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"

        for((x2=x1+1; x2<= limit; x2++))
        do
                        depth=2;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}_${x2}"

                        cut -d " " -f 1,$x1,$x2 $dfile >| $outfile
                        # To append trailing space before newline char
                          perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"

        for((x3=x2+1; x3<= limit; x3++))
        do
                        depth=3;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}_${x2}_${x3}"

                        cut -d " " -f 1,$x1,$x2,$x3 $dfile >| $outfile
                        # To append trailing space before newline char
                          perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"
# to diplay progress
#ls ${odir}/* |wc -l

        for((x4=x3+1; x4<= limit; x4++))
        do
                        depth=4;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}_${x2}_${x3}_${x4}"

                        cut -d " " -f 1,$x1,$x2,$x3,$x4 $dfile >| $outfile
                        # To append trailing space before newline char
                          perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"

        for((x5=x4+1; x5<= limit; x5++))
        do
                        depth=5;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}"

                        cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5 $dfile >| $outfile
                        # To append trailing space before newline char
                          perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"

        for((x6=x5+1; x6<= limit; x6++))
        do
                        depth=6;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}"

                        cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6 $dfile >| $outfile
                        # To append trailing space before newline char
                          perl -ne 's/\n/ \n/; print "$_";' -i $outfile
```

```
                ofile="${otag}"

for((x7=x6+1; x7<= limit; x7++))
do
                depth=7;
                ofile="${odir}/${depth}/${ofile}"
                outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}"

                cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7 $dfile >| $outfile
                # To append trailing space before newline char
                  perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                ofile="${otag}"

for((x8=x7+1; x8<= limit; x8++))
do
                depth=8;
                ofile="${odir}/${depth}/${ofile}"
                outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}"

                cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8 $dfile >| $outfile
                # To append trailing space before newline char
                  perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                ofile="${otag}"

for((x9=x8+1; x9<= limit; x9++))
do
                depth=9;
                ofile="${odir}/${depth}/${ofile}"
                outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}_${x9}"

                cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9 $dfile >| $outfile
                # To append trailing space before newline char
                  perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                ofile="${otag}"

for((x10=x9+1; x10<= limit; x10++))
do
                depth=10;
                ofile="${odir}/${depth}/${ofile}"
                outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}_${x9}_${x10}"

                cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9,$x10 $dfile >| $outfile
                # To append trailing space before newline char
                 perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                ofile="${otag}"

for((x11=x10+1; x11<= limit; x11++))
do
                depth=11;
                ofile="${odir}/${depth}/${ofile}"
                outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}_${x9}_${x10}_${x11}"

                cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9,$x10,$x11 $dfile >| $outfile
                # To append trailing space before newline char
                 perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                ofile="${otag}"

for((x12=x11+1; x12<= limit; x12++))
do
                depth=12;
                ofile="${odir}/${depth}/${ofile}"
                outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}_${x9}_${x10}_${x11}_${x12}"

                cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9,$x10,$x11,$x12 $dfile >| $outfile
                # To append trailing space before newline char
```

```
                         perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"

        for((x13=x12+1; x13<= limit; x13++))
        do
                        depth=13;
                        ofile="${odir}/${depth}/${ofile}"
                        outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}_${x9}_${x10}_${x11}_${x12}_${x13}"

                        cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9,$x10,$x11,$x12,$x13 $dfile >| $outfile
                        # To append trailing space before newline char
                         perl -ne 's/\n/ \n/; print "$_";' -i $outfile
                        ofile="${otag}"

        done # x13
        done
        done
        done # x10
        done
        done
        done # x7
        done
        done
        done # x4
        done
        done
        done # x1

######################################################################
ofile="${otag}"

# Special case where all features are taken

x1=2; x2=3; x3=4; x4=5; x5=6; x6=7; x7=8; x8=9;
x9=10; x10=11; x11=12; x12=13; x13=14; x14=15;

depth=14;
ofile="${odir}/${depth}/${ofile}"
outfile="${ofile}_${x1}_${x2}_${x3}_${x4}_${x5}_${x6}_${x7}_${x8}_${x9}_${x10}_${x11}_${x12}_${x13}_${x14}"

cut -d " " -f 1,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9,$x10,$x11,$x12,$x13,$x14 $dfile >| $outfile
# To append trailing space before newline char
 perl -ne 's/\n/ \n/; print "$_";' -i $outfile

##########################################################################
echo
echo "Done !!"
echo "Files successfully created in \"$odir\" directory ."
echo
```

A.23 Miscellaneous Tools

A.23.1 avgknn.sh

```
#!/bin/sh

##########################################################################################

# Finds avg misclassification for each file (each file with different K value)

##########################################################################################
echo; echo ”Command line Usage :  ——$0——result_dir −−−”; echo

rdirtag=”NumFeat__”
Ktag=”K__”
Klist=”13579”

if [ $# −eq 1 ]
then
        resdir=$1
else
        echo; echo −n ”Enter results directory in which Knn−results files are stored : ”
        read resdir
fi

for file in ${resdir}/${rdirtag}∗/${Ktag}[$Klist]
do

        lines=‘wc −l < ${file}‘
        start_idx=1;
        avg_file=”${file}.avg”

        cat /dev/null >| ”$avg_file”
        while [ $start_idx −le $lines ]
        do
                avg=‘tail −n +${start_idx} $file |head −n 100| egrep ”100\.000” |wc −l‘

                echo −ne ”\t$avg% —— ” >> ${avg_file}
                tail −n +${start_idx} $file|head −n 1|cut −d ” ” −f 1 | \
                                                perl −ne ’s/−−−mis_class_rate//g; chomp($_); print ” $_%\n”;’ >> ${avg_file}

                start_idx=‘expr ${start_idx} + 100‘
        done
        echo; echo −e ”file = $file \navg_file = ${avg_file}\n”

done # end for


if false
then
#####################################          ALL avg Part          ########################################
echo
echo −n ”Enter results directory in which knn−result files are stored : ”
read results_dir;

avg_file=”all_avg.txt”;
rdirtag=”NumFeat__”
Ktag=”K__”

rm −f ${results_dir}/${avg_file};

for i in ${results_dir}/${rdirtag}[0−9]∗/∗
do
   perl −e ’print”$ARGV[0] : ”;$x=0; while(<>) {split; $x=$x+$_[2] } $x=$x/$.; print”$x\n”’     $i >> ${results_dir}/${avg_file
   # remember that 3rd field contains misclassification rates
done

# finally calculate average across different files(with different K values)
perl -e 'print"$ARGV[0] : "; $x=0; while(<>) {split; $x=$x+$_[2]} $x=$x/$.; print"$x\n"' \
                           ${results_dir}/${avg_file} >> ${results_dir}/${avg_file} ;

cat ${results_dir}/${avg_file}
##########################################################################################################
fi
```

A.23.2 avgknnsingle.sh

```
#!/bin/bash

####################################################################

#  greps "Avg" tag (at tail of) input .res file and append to $out_file

#  Mainly meant for files created by loocv_knn_single.sh

####################################################################

echo "Usage :  ---$0---\"result_file\"---avg_file----"
if [ $# -eq 2 ]
then
        res_file=$1
        avg_file=$2
else
        echo -n "Enter result_file : "
        read res_file
        echo -n "Enter output avg_file : "
        read avg_file
fi
echo "Warning !! All content will be APPENDED to $avg_file !! "
echo "$avg_file may contain old data !!"


grep -H "Avg_" $res_file >> $avg_file    # -H to print filename
```

A.23.3 avgsvm.sh

```
#!/bin/bash

#######################################################################

# Warning : Only intended for output of libsvm

# Calculates ”Avg misclas_rate” for each feature directory
  # stores result in file ”avg_rate.txt” in each feature's dir

# Calculates GLOBAL ”Avg misclass_rate” for all features
  # stores result in file ”global_avg_rate.txt” above all feature's dirs

#######################################################################

res_dir=”results_svm”
glb_avg_f=”avg_of_avg.txt”
feat_dir=”fnum_”
avg_f=”avg_rate.txt”

for cur_fdir in ${res_dir}/${feat_dir}*  # loop for current feature-directory
do
   # undefine vars
     total=;  correct=;  avg=;  numf=;

   # check if there are any result files in cur-feat-dir
     flag=‘ls ${cur_fdir}/result_* 2>/dev/null | wc -l‘
     if [ $flag -ne 0 ]
     then
               # grepping the values from all files of current feat-dir e.g. fnum_14
                 correct=‘egrep ’1/1’ ${cur_fdir}/result_* | wc -l‘
                 total=‘ls ${cur_fdir}/result_*|       wc -l‘
     fi

   # using perl to calculate avg, since ‘expr‘ cant handle floats
     # execute only if $total is neither null nor zero
     if [ -n ”$total” ]
     then
        if [ $total -ne 0 ]
               then
            avg=‘perl -e ’$x=($ARGV[0] *100) / $ARGV[1]; print $x’ $correct $total | cat‘
               fi
     fi

   # sending the outputs to file ”$avg_f” in each feat-dir
        if [ -n ”$avg” ]
        then
               echo ”Avg_Accuracy = $avg %” >| ${cur_fdir}/${avg_f}
               numf=‘perl -e ’$xx=substr $ARGV[0],index($ARGV[0],$ARGV[1])+length $ARGV[1];print ”$xx”’ $cur_fdir $feat_dir|ca
               echo ”No_of_Features=$numf” >> ${cur_fdir}/${avg_f}
               echo  >> ${cur_fdir}/${avg_f}
        fi

done

###########################################################################################

# Calcualting Avg of Avgs


egrep Avg_Accuracy $res_dir/$feat_dir*/$avg_f | cut -d ” ” -f 3 >| tmp_avg.txt

avg_of_avg=‘perl -e ’$x=0; while(<>) { $x+=$_}; $y=$x/14, print ”$y”’ tmp_avg.txt | cat‘
```

```
echo "Warning : Global Sum is divided by 14" >&2

echo "Avg_of_Avg = $avg_of_avg" >| $res_dir/$glb_avg_f
```

A.23.4 command1bayesNonAvgGLCM.sh

```
#!/bin/bash
###########################################################################

        # To compare performances of...
                # various values of "FILTER_SIZE"(gaussian)- corresponding to one ROI_size

        # It will create num(filter_sizes)*features number..
                # of files in G_cmp_${pad}_${roi_size} dir

        # Run it (cardinality -->) "|{pad}| * |{roi_size}|" no. of times
#########################################################################

#pad="NP"

resdir_tag="res_"
echo
echo -n "for which Padding  (ZP | BP | NP) :  "
read pad
echo -n "for which ROI_size (16 | 32 | 48 | 64) :  "
read roi_size

rm -f G_cmp_${pad}_${roi_size}/*
for file in res_G3_${pad}_${roi_size}/NumFeat__*
        # doesn't matter for G3/G5/G7
        # loop is for each number of feature
        # e.g first for 1, then for 2 .. etc
do
        bsname=`basename $file`;

        # in no-filtering case padding is not applicable
         cat ${resdir_tag}G0_${roi_size}/$bsname | cut -d " " -f 1,3 | \
                        perl -ne 's/---mis_class_rate//; s/ /   /; print "$_";' >| G_cmp_${pad}_${roi_size}/G0_$bsname;

         cat ${resdir_tag}G3_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G3_$bsname;
         cat ${resdir_tag}G5_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G5_$bsname;
         cat ${resdir_tag}G7_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G7_$bsname;
         cat ${resdir_tag}G9_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G9_$bsname;
        cat ${resdir_tag}G11_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G11_$bsname;
        cat ${resdir_tag}G13_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G13_$bsname;
        cat ${resdir_tag}G15_${pad}_${roi_size}/$bsname | cut -d " " -f 3 >| G_cmp_${pad}_${roi_size}/G15_$bsname;
done
```

A.23.5 command2bayesNonAvgGLCM.sh

```
#!/bin/bash

#####################################################

        #       uses files created by "command1_bayes.sh"

        #       calcultes avg for all values of 'FILTER_SIZE'
        #       Also prints original values

#####################################################

pad="NP"
#echo "Using misclassification threshold < 10%"; echo
#threshold_mis=10
echo -n "Enter threshold value of misclassification : "
read threshold_mis
#echo -n "for which padding [ZP | BP | NP ] :  "
#read pad
echo -n "for which ROI_size [16 | 32 | 48 | 64 ] :  "
read roi_size

typeset -x threshold_mis

if [ $PWD != "G_cmp_${pad}_${roi_size}" ]
then
        cd G_cmp_${pad}_${roi_size};
fi
for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14
do
         echo -n "Show for Num_of_Features=$i "; read tmp
        paste G*_NumFeat__${i}  |\
                perl -e '
                        $x0=0; $x1=0; $x2=0; $x3=0; $x4=0; $x5=0; $x6=0; $x7=0; $x8=0;
                        $c0=0; $c1=0; $c2=0; $c3=0; $c4=0; $c5=0; $c6=0; $c7=0; $c8=0;
                        $lastline=0;
                                #print "0x0\t11x11\t13x13\t15x15\t3x3\t5x5\t7x7\t9x9\tFeature_list\n";
                                #print "0x0\t3x3\t5x5\t7x7\t9x9\t11x11\t13x13\t15x15\tFeature_list\n";
                        print "\n\n\n------------------------------ Misclassification rates ---------------------------\n";
                        while (<>)
                        {
                                s/.000000\b//g;
                                @x=split("\t",$_);
                                $x1+=$x[1]; $x2+=$x[2]; $x3+=$x[3]; $x4+=$x[4];
                                $x5+=$x[5]; $x6+=$x[6]; $x7+=$x[7]; $x8+=$x[8];
                                if($x[1] < $ENV{"threshold_mis"}) {
                                        $c1++; }
                                if($x[2] < $ENV{"threshold_mis"}) {
                                        $c2++; }
                                if($x[3] < $ENV{"threshold_mis"}) {
                                        $c3++; }
                                if($x[4] < $ENV{"threshold_mis"}) {
                                        $c4++;}
                                if($x[5] < $ENV{"threshold_mis"}) {
                                        $c5++;}
                                if($x[6] < $ENV{"threshold_mis"}) {
                                        $c6++;}
                                if($x[7] < $ENV{"threshold_mis"}) {
                                        $c7++;}
                                if($x[8] < $ENV{"threshold_mis"}) {
                                        $c8++;}

                                if ($x[1] < $ENV{"threshold_mis"} && $x[2] < $ENV{"threshold_mis"} && $x[3] < $ENV{"threshold_m
                                {
                                        #$_=join("\t", @x);
                                        #print "$_";
                                        chomp($x[8]);
                                        print "$x[1]\t$x[5]\t$x[6]\t$x[7]\t$x[8]\t$x[2]\t$x[3]\t$x[4]\t$x[0]\n";
                                        $lastline++;
                                }

                        }
                        print "------------------------------------------------------------------------------------------------------\n";
                        print "\ntotal (<= $ENV{\"threshold_mis\"} ):";
                        print "\n$c1\t$c5\t$c6\t$c7\t$c8\t$c2\t$c3\t$c4";
                        print "\n--------------------------------------------------------------------------------------------------------";
                        print "\n0x0\t3x3\t5x5\t7x7\t9x9\t11x11\t13x13\t15x15\tFeature_list\n";
                        print "\n\ntotal lines = $lastline" ;
                        #print "\n\nSum:\n$x1\t$x2\t$x3\t$x4\t$x5\t$x6\t$x7\t$x2";
                        #$x1/=$.; $x2/=$.; $x3/=$.; $x4/=$.; $x5/=$.; $x6/=$.; $x7/=$.; $x8/=$.;
                        #printf("\n\nAvg:\n%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $x1, $x2, $x3, $x4, $x5, $x6, $x7
                        ' |less


done

cd ..
```

A.23.6 command3bayes.sh

```
#!/bin/sh

##########################################################################################################

#       Take two features as input and compares performances
#       this script should be used after using...
        #       ..."command1_bayes_(Avg|NonAvg).sh" and "command2_bayes_(Avg|NonAvg).sh"
        #               ...for the Best value of G:(filter_size)

##########################################################################################################

# take input features
echo
echo "Comapare performance of two features :"
echo
echo -n "Warning : valid input is (2-15), and after 10 both values must be consecutive !!"
echo "Enter 1st feature : "
read feat1
echo
echo -n "Enter 2nd feature : "
read feat2
echo
echo "Enter \"res_G?_pad?_ROIsize\" directory in which files for comparison are kept:"
echo "Enter full name:"
read resdir

if [ $PWD != "$resdir" ]
then
        cd "$resdir"
fi
count=$feat1;
for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14
do
        if [ $feat2 -ge 10 ]  #features are consecutive
        then
                cat NumFeat__$i |
                        egrep "d_($feat1|$feat2)_" |
                        egrep -v "d_${feat1}_${feat2}_*" >| tmp;
        else
                cat NumFeat__$i | egrep "d_($feat1|$feat2)_[^${feat1}-${feat2}]" >| tmp;
        fi

        line=`wc -l <tmp`;
        line=`expr $line / 2`;
        echo $line;
        split tmp --lines=$line;
        paste xaa xab |
                perl -ne 's/mis_class_rate//g; print "$_"' |
                perl -e '
                        $x1=0; $x2=0;
                        while(<>) {
                                @num=split;
                                $x1+=$num[2];
                                $x2+=$num[5];
                                print "$_";
                        }

                        print "\ntotal lines = $.";
                        printf("\nSum :\n%.2f\t%.2f", $x1, $x2);
                        $x1 /= $.;
                        $x2 /= $.;
                        printf("\nAvg :\n%.2f\t%.2f\n", $x1, $x2);' | less
        rm -f xaa xab tmp;
        count=`expr $count + 1`;
        if [ $count -eq 15 ]
        then
                exit 0;
        fi
        read temp;
done

echo
echo "Comparision done between following features : $feat1, $feat2"
echo
cd ..
```

A.23.7 loocvknnsingle.sh

```
#!/bin/sh

##########################################################################################################

        #       for a single input datafile, this script constructs all possible trng and tst set based on LOOCV model..
                        # ... and executes Knn classifier for all these set pairs one-by-one
        #       Results are stored in $datafile.res
##########################################################################################################

K_range="1 3 5 7 9"
tot_size=100;
trng_size=99;
tst_size=1;

if [ $# -ne 2 ]
then
        echo "Usage : ---./Knn---datafile---no_of_feat---"
        exit 1
fi

file=$1
num_feat=$2

        #echo "Enter datafile : "
        #read file
        #echo "Enter no.of features :"
        #read num_feat

resfile="`basename ${file}`.res"
typeset -x pid=$$
trng_f="/tmp/$$_`basename ${file}`"
tst_f="/tmp/$$_tst.txt"

cat /dev/null >| "${resfile}"

        for K in `echo ${K_range}`
        do
                echo -e "K_Value------------------------$K" >> "${resfile}"
                i=1;
                while [ $i -le $tot_size ]  # only 'ith' point is tst_dataset
                do
                        # constructing training dataset (in file $trng_f)
                          head -n  `expr $i - 1` ${file}  >| ${trng_f}
                          tail -n +`expr $i + 1` ${file}  >> ${trng_f}

                        # constructing test dataset (in file $tst_f)
                          tail -n +$i ${file} | head -n 1 >| ${tst_f}

                        # Applying Knn classifier to current sets
                        ./Knn $trng_f $trng_size $tst_f $tst_size $K $num_feat >> "${resfile}"

                i=`expr $i + 1`;
                done # while
        done # for(K)

rm $trng_f $tst_f
perl -ne 's/\/tmp\/$ENV{"pid"}_//g; print "$_";' -i "${resfile}"
# counting avg for each K value
line=1;
for i in `echo ${K_range}`
do
        line=`expr $line + 1`;
        echo -n "Avg_for_K=$i----" >> "${resfile}"
        tail -n +$line "${resfile}" |head -100|egrep 100|wc -l >> "${resfile}"
        line=`expr $line + 100`
done
echo
echo "Done !"
echo "Result file = ${resfile} !"
echo
```

A.23.8 loocvmlpsingle.sh

```
#!/bin/sh

######################################################################

                # Executes Mlp classfier(Loocv Model) for a single file

######################################################################
echo; echo "Command line Usage :  ---$0---datafile---no_of_features---"; echo


if [ $# -eq 2 ]
then
        file=$1
        num_feat=$2
else
        echo "Enter datafile : "
        read file
        echo "Enter no.of features :"
        read num_feat
fi

tot_size=100;
trng_size=99;
tst_size=1;
trng_f="/tmp/$$_`basename $file`"
tst_f="/tmp/$$_tst.txt"
typeset -x pid=$$
cat /dev/null >| "${file}.res"
make Mlp > /dev/null

        i=1;
        while [ $i -le $tot_size ]  # only 'ith' point is tst_dataset
        do
                # constructing training dataset (in file $trng_f)
                  head -n  `expr $i - 1` ${file}  >| ${trng_f}
                  tail -n +`expr $i + 1` ${file}  >> ${trng_f}

                # constructing test dataset (in file $tst_f)
                  tail -n +$i ${file} | head -n 1 >| ${tst_f}

                # Applying Knn classifier to current sets
                ./Mlp $trng_f $trng_size $tst_f $tst_size $num_feat >> "${file}.res"

        i=`expr $i + 1`;
        done # while

rm $trng_f $tst_f
perl -ne 's/\/tmp\/$ENV{"pid"}_//g; print "$_";' -i "${file}.res"
echo -n "avg_misclassification(%)=" >> "${file}.res"
egrep "1\.000" "${file}.res" | wc -l >> "${file}.res"
#echo -n "avg_Accuracy(%)="  >> "${file}.result"
#egrep "0\.000" "${file}.result" | wc -l >> "${file}.result"
echo "Done !"
echo "Result file = ${file}.res !"
echo
```

A.23.9 man1.sh

```
#!/bin/bash

## for ROI size 16

roisize=48
for pad in NP
do
        for gsize in 3 5 7 9 11 13 15
        do
                cp -r ../bayes/dset_G${gsize}_${pad}_${roisize} .
                ./convert_format_svm.sh dset_G${gsize}_${pad}_${roisize}
                ./Svm.sh  dset_G${gsize}_${pad}_${roisize} res_G${gsize}_${pad}_${roisize}
                rm -rf dset_G${gsize}_${pad}_${roisize}
        done
done
```

A.23.10 man2.sh

```
#!/bin/bash

## for ROI size 16

roisize=64
for pad in NP
do
        for gsize in 3 5 7 9 11 13 15
        do
                cp -r ../bayes/dset_G${gsize}_${pad}_${roisize} .
                ./convert_format_svm.sh dset_G${gsize}_${pad}_${roisize}
                ./Svm.sh  dset_G${gsize}_${pad}_${roisize} res_G${gsize}_${pad}_${roisize}
                rm -rf dset_G${gsize}_${pad}_${roisize}
        done
done
```

A.23.11 man3.sh

```
#!/bin/bash

## for ROI size 16

roisize=16


for pad in ZP
do
        for gsize in 15
        do
                cp -r ../bayes/dset_G${gsize}_${pad}_${roisize} .
                ./convert_format_svm.sh dset_G${gsize}_${pad}_${roisize}
                ./Svm.sh  dset_G${gsize}_${pad}_${roisize} res_G${gsize}_${pad}_${roisize}
                rm -rf dset_G${gsize}_${pad}_${roisize}
        done
done
```

A.23.12 man4.sh

```
#!/bin/bash

## for ROI size 16

roisize=16


for pad in ZP
do
        for gsize in 5 7
        do
                cp -r ../bayes/dset_G${gsize}_${pad}_${roisize} .
                ./convert_format_svm.sh dset_G${gsize}_${pad}_${roisize}
                ./Svm.sh  dset_G${gsize}_${pad}_${roisize} res_G${gsize}_${pad}_${roisize}
                rm -rf dset_G${gsize}_${pad}_${roisize}
        done
done
```

A.24 nor.sh

```
#!/bin/sh

############################################################
        # output file = $feat_file

############################################################


execu="Ext_Feat"
if [ $# -eq 1 ]
then
        feat_file=$1
else
        feat_file="Nor_featr.txt"
fi

echo -n "Enter ROI_pics directory name : "
read pics_dir

if [ -e "$feat_file" ]
then
        rm -f "$feat_file" ;
fi

                # To process files from '00' to '09' #
for x in 0 1 2 3 4 5 6 7 8 9
do
        ./"${execu}" ${pics_dir}/nor_nf_0${x}.pgm ${pics_dir}/nor_ff_0${x}.pgm "$feat_file"
a 0
done

                # To process files from '10' to '70' #
x=10;
while [ $x -lt 71 ]
do
        ./"${execu}" ${pics_dir}/nor_nf_${x}.pgm ${pics_dir}/nor_ff_${x}.pgm "$feat_file" a
0
        x=`expr $x + 1`;
done



############################################################
```

A.25 fty.sh

```
#!/bin/sh

############################################################

execu="Ext_Feat";

if [ $# -eq 1 ]
then
        feat_file=$1
else
        feat_file="Fty_featr.txt";
fi

echo -n "Enter ROI_pics dir name : "
read pics_dir

if [ -e "$feat_file" ]
then
        rm -f "$feat_file" ;
fi

                # To process files from '00' to '09' #
for x in 0 1 2 3 4 5 6 7 8 9
do
        ./"$execu" ${pics_dir}/fty_nf_0${x}.pgm ${pics_dir}/fty_ff_0${x}.pgm $feat_file a 1
done

                # To process files from '10' to '28' #
x=10;
while [ $x -lt 29 ]
do
        ./"$execu" ${pics_dir}/fty_nf_${x}.pgm ${pics_dir}/fty_ff_${x}.pgm $feat_file a 1;
        x=`expr $x + 1`;
done


############################################################
```

A.26 convertformatmlp.sh

```
#!/bin/bash

#############################################################################################

# Executes "convert_format_mlp.pl" for all files in $ddir (i.e. dataset_dir) recursively

#############################################################################################


        echo
        echo -n "Enter the name of dataset directory :  "
        read ddir;

        echo "convert all files in $ddir (using \"convert_format_mlp.pl\") :  (y|n)"

        read ques
        if [ $ques = "y" -o $ques = "Y" ]
        then
                for file in ${ddir}/*/d_*
                do
                        ./convert_format_mlp.pl     ${file}     tmp
                        # ./convert_format_mlp.pl   inputfile   outfile
                        mv -f tmp $file
                done
        fi
```

A.27 convertformatmlp.pl

```
#!/usr/bin/perl

####################################################################

# Input : dataset file [outputs of "gen_all_datasets.sh" ]

##  Converts data into the form compatible for MLP classifier

#  Transfers  1st column of input_file at last position
#             and stores results in output_file
#
        # Appends a space between last column and trailing newline
####################################################################


## check command line arguments
$argcnt = $#ARGV + 1;
if ($argcnt != 2 ) {
        print "usage: perl_script  input_file  output_file\n";
        exit(1);
}
else {
       $infile = $ARGV[0];
       $outfile = $ARGV[1];
}

open(INFILE, $infile) || die "\nUnable to open input file \"$infile\" : $!";
open(OUTFILE, ">$outfile") || die "\nUnable to open output file \"$outfile\" : $! \n";

# Transfer 1st column at last position
while(<INFILE>)
{
        split;
        $tmp=$_[0];
        for($i=0; $i< $#_; $i++)  ### Here  '$#_' represents no_of_features in current line
        {
                $_[$i] = "$_[$i+1]";
        }
        $_[$i]=$tmp;

        $_ = join(" ", @_);
        print OUTFILE "$_ \n"; # appending one space before newline
}

close(INFILE);
close(OUTFILE);
```

A.28 convertformatsvm.sh

```
#!/bin/bash

###########################################################################

# Executes "convert_format_svm.pl" for all files in $ddir (i.e. dataset_dir) recursively

########################################################################


echo "Command_line Usage : -----$0-----dataset_dir---"; echo
if [ $# -eq 1 ]
then
        ddir="$1"
else
        echo -n "Enter the name of dataset directory :  "
        read ddir;
fi

echo "convert all files in $ddir (using \"convert_format_svm.pl\") :  (y|n)"
#read ques
ques="y"
if [ $ques = "y" -o $ques = "Y" ]
then
        for file in ${ddir}/*/d_*
        do
                ./convert_format_svm.pl     ${file}     /tmp/$$_convert_svm
                # ./convert_format_svm.pl  inputfile  outfile
                mv -f /tmp/${$}_convert_svm $file
        done
fi
```

A.29 convertformatsvm.pl

```
#!/usr/bin/perl

##  Converts data compatible for libsvm


## check command line arguments
$argcnt = $#ARGV + 1;
if ($argcnt != 2 ) {
        print "usage: $0  input_file  output_file\n";
        exit(1);
}
else {
       $infile = $ARGV[0];
       $outfile = $ARGV[1];
}



open(INFILE, $infile) || die "\nUnable to open input file \"$infile\" : $!";
open(OUTFILE, ">$outfile") || die "\nUnable to open output file \"$outfile\" : $! \n";

while(<INFILE>)
{
        split;
        for($i=1; $i<= $#_; $i++)  ### Here  '$#_' represents no_of_features in current line
        {
                $_[$i] = "$i".":"."$_[$i]";
        }
        $_ = join(" ", @_);
        print OUTFILE "$_\n";
}

close(INFILE);
close(OUTFILE);
```

Appendix B

Basic theory of Classifiers

1. Naive Bayes classifier
2. K-NN classfier
3. Multilayer perceptron
4. Support Vector Machine

B.1 Naive Bayes classifier

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes theorem with strong (naive) independence assumptions, or more specifically, independent feature model. Naive Bayes Classifier is a simple but effective Bayesian classifier for vector data (i.e. data with several attributes) that assumes that attributes are independent given the class.
In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature.
The naive Bayes or idiot Bayes assumption is that all the features are conditionally independent given the class label:
Even though this is usually false (since features are usually dependent), the resulting model is easy to fit and works surprisingly well.
Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix

If a given class and feature value never occur together in the training set then the frequency-based probability estimate will be zero. This is problematic since it will wipe out all information in the other probabilities when they are multiplied. It is therefore often desirable to incorporate a small-sample correction in all probability estimates such that no probability is ever set to be exactly zero.

In spite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers often work much better in many complex real-world situations than one might expect.
The goal of bayesian classification is to compute the the probability of C_i given point X_j . Then among all classes, we choose the one that has the largest probability, and classify X_j as being of that class. That is, we predict the class of X_j as arg $max_i\{P(C_i|X_j)\}$.
To compute $P(C_i|X_j)$ simply invert the probability using the Bayes theorem:

$$P(C_i|X_j) = \frac{P(X_j|C_i) * P(C_i)}{P(Xj)} \tag{B.1}$$

Keep in mind that $X_j = (X_J^1, X_j^2, ..., X_j^d)$ is a point in d-dimensions. Thus to compute the conditional probability $P(X_j|C_i)$ requires that we estimate the joint probability distribution (in all d-dimensions) for the point for each of the classes. One way to obtain the joint distribution is to assume that X_j follows a d-dimensional multivariate normal distribution. With this assumption, the problem becomes very hard since there are many parameters to estimate, including Σ, d x d matrix of covariances, and the mean $\mu = (\mu_1, \mu_2, ..., \mu_d)$. So we end up having a total of $d * (d1)2 + d$ parameters to estimate. The main assumption in Naive Bayes is that attributes are all independent. With this assumption, the covariance matrix has a simpler form:

$$\Sigma = I_d = diag(\sigma_1^2, \sigma_2^2, ..., \sigma_d^2) \tag{B.2}$$

and the mean remains $\mu = (\mu_1, \mu_2, ..., \mu_d)$. Now we only have 2 x d parameters to estimate in place of $O(d^2)$ parameters. Thus under the independence assumption, $P(X_j|C_i)$ can be reduced to the equation below:

$$P(X_j|C_i) = \prod_{a=1}^{d} P(X_j^a|C_i) \tag{B.3}$$

where X_j^a is the value of X_j in the a^th dimension

For numeric data we assume that each dimension is normally distributed, and we thus have to estimate σ, μ for each class C_i separately, directly from data. Once the mean and variance per class C_j for each dimension a (viz. σ_i^a, μ_i^a are known, we compute

$$P(X_j^a|C_i) = N(X_j^a|\sigma_i^a, \mu_i^a) = \frac{\exp - \left\{ \frac{(X_j^a - \mu_i^a)^2}{2(\sigma_i^a)^2} \right\}}{\sqrt{2\pi}\sigma_i^a} \tag{B.4}$$

for ease of implementation probability distribution f of the feature vector $X_{n\times1}$ can be calculated as follows :

$$f(X_{n\times1}) = \frac{1}{(\sqrt{2\pi})^n|\Sigma|^{\frac{1}{2}}} \exp - \left\{ \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) \right\} \tag{B.5}$$

Other than this the standard Bayes classifier applies here also.

B.2 K-NN classifier

The k-nearest neighbors classifier (k-NN) is a method for classifying objects based on closest training examples in the feature space. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors. k is a positive integer, typically small. If $k = 1$, then the object is simply assigned to the class of its nearest neighbor.

The neighbors are taken from a set of objects for which the correct classification is known. This can be thought of as the training set for the algorithm. In order to identify neighbors, the objects are represented by position vectors in a multidimensional feature space. It is usual to use the Euclidean distance, though other distance measures, such as the Manhattan distance could in principle be used instead.

The training examples are vectors in a multidimensional feature space. A point in the space is assigned to the class C if it is the most frequent class label among the k-nearest training samples.

The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the actual classification phase, the test sample (whose class is kept unknown to the classifier) is represented as a vector in the feature space. Distances from the new vector to all stored vectors are computed and k closest samples are selected. The test sample is then assigned that class label which is most common among actual class label of these k samples.

The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques, for example, cross-validation.

The accuracy of the k-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling[citation needed]. Another popular approach is to scale features by the mutual information of the training data with the training classes[citation needed].

Strengths:

- Simple to implement and use
- Comprehensible easy to explain prediction
- Robust to noisy data by averaging k-nearest neighbors.

Weaknesses:

- Need a lot of space to store all examples
- Takes more time to classify a new example than with a model (need to calculate and compare distance from new example to all other examples)

B.3 Multilayer Perceptron

A multilayer perceptron is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. It is a modification of the standard single layer perceptron in that it uses three or more layers of neurons (nodes) with nonlinear activation functions, and is more powerful than the single layer perceptron in that it can distinguish data that is not linearly separable, or separable by a hyperplane.

- Activation Function : What makes a multilayer perceptron a good classifier is that each neuron uses a nonlinear activation function which was developed to model the frequency of action potentials, or firing, of biological neurons in the brain. This function is modeled in several ways, but must always be normalizable and differentiable. The two main activation functions used in current applications are both sigmoids, and are as follows

  $$\phi(v_i) = \tanh(v_i) \text{ and } \phi(v_i) = (1 + e^{-v_i})^{-1},$$

  in which the former function is a hyperbolic tangent which ranges from -1 to 1, and the latter is equivalent in shape but ranges from 0 to 1. Here y_i is the output of the i^{th} node (neuron) and v_i is the weighted sum of the input synapses. More specialized activation functions include radial basis functions which are used in another class of supervised neural network models.

- Layers : The multilayer perceptron consists of an input and an output layer with one or more hidden layers of nonlinearly-activating nodes. Each node in one layer connects with a certain weight w_{ij} to every other node in the next layer.

- Learning through backpropagation : Learning occurs in the perceptron by changing connection weights (or synaptic weights) after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron.

The error in output node j in the n^{th} data point is represented by $e_j(n) = d_j(n)y_j(n)$, where d is the target value and y is the value produced by the perceptron. Weight correction (learning) of the nodes is done based on those corrections which minimize the energy of error in the entire output. The error is given by :

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n).$$

By the theory of differentials, we find our change in each weight to be

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

where y_i is the output of the previous neuron and η is the learning rate, which is carefully selected to ensure that the weights converge to a response that is neither too specific nor too general. In programming applications, typically ranges from 0.2 to 0.8.

The derivative to be calculated depends on the input synapse sum v_j, which itself varies. It is easy to prove that for an output node this derivative can be simplified to

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

where ϕ' is the derivative of the activation function described above, which itself does not vary. The relevant derivative for hidden layer can be calculted and following expression can be found

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n)) \sum_k \frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n).$$

Note that this depends on the change in weights of the k^{th} nodes, which represent the output layer. So to change the hidden layer weights, we must first change the output layer weights according to the derivative of the activation function, and so this algorithm represents a backpropagation of the activation function.

B.4 Support Vector Maching (SVM)

SVM is a useful pattern recognition method emerging in the mid-1990s. It is a useful technique for data classification. It transforms a pattern recognition problem into a quadratic programming optimization, in theory, ensuring the overall optimal solution and avoiding the local convergence. SVM deals with the small sample set and high-dimensional nonlinear pattern recognition problems with unique advantages, and thus can be effectively applied to small sample estimation and prediction problems. In this work along with other classifiers , SVM was also used for classification of fatty liver and normal liver, and achieved high recognition rate.

The basic principle of SVM is to find an optimal plane, which can not only aim to separate two sample sets correctly, but also achieve the largest classification distance. When problems can not be divided linearly, it can be resolved through nonlinear transformation into a linear classification problem. Nonlinear transformation is achieved through the definition of the appropriate kernel functions, commonly nuclear polynomial functions such as RBF kernel. Viewing input data as two sets of vectors in an n-dimensional space, an SVM will construct a separating hyperplane in that space, one which maximizes the margin between the two data sets. To calculate the margin, two parallel hyperplanes are constructed, one on each side of the separating hyperplane, which are ”pushed up against” the two data sets. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the neighboring datapoints of both classes, since in general the larger the margin the lower the generalization error of the classifier. SVMs use two key concepts to solve this problem: large-margin separation and kernel functions. The idea of large margin separation can be motivated by classification of points in two dimensions . for example in $2 - d$ to classify the points is to draw a straight line as the separation boundary. One would intuitively draw the line such that it is as far as possible away from the points in both sets, hence keeping the margin as much as possible. Similar way is followed by SVM also and it gives as output the best hyperplane as it can do.

References

1. Textural Features for Image Classification, 1973, ROBERT M. HARALICK, K. SHANMUGAM, ITS'HAK DINSTEIN
2. B-scan Images Analyzed By CNN And Cooccerrence Matrix, Guodong Li1, Huiming Song1, Jianghe Wang2 ,Huiwen Hong2 ,Yanling Liu2
3. FUZZY Neural Network-Based Texure Analysis of Ultrasonic Images, S. Pavlopoulos, E. Kyriatou, D. Koutsouris, K. Blekasl, Ai Stafylopotis, P. Zoumpoulis
4. Fatty Liver Automatic Diagnosis from Ultrasound Images, Ricardo Ribeiro, Jos Seabra, Joo Sanches
5. Classification Algorithms for Quantitative Tissue Characterization of Diffuse Liver Disease from Ultrasound Images, Yasser M. Kadah, Aly A. Farag, Jacek M. Zurada, Ahmed M. Badawi and Abou-Bakr M. Youssef
6. Computer Aided Diagnosis of Fatty Liver Ultrasonic Images Based on Support Vector Machine, Guokuan Li, Yu Luo, Wei Deng, Xiangyang Xu, Aihua Liu and Enmin Song
7. A New Method for Attenuation Coefficient Measurement in the Liver, Yasutomo Fujii et. al.
8. Modeling Log-Compressed Ultrasound Images for Radio Frequency Signal Recovery , Jos Seabra and Joao Sanches
9. Recognition of Fatty Liver Using Hybrid Neural Network, Jiangli Lin, Xian-Hua Shen, Tianfu Wang, Deyu Li, Yan Luo, and Ling Wang
10. Experimental Comparison of Feature Subset Selection Methods, Chulmin Yun and Jihoon Yang

11. Lecture 22: Texture, Bryan S. Morse, Brigham Young University

12. Convex Ultrasound Image Reconstruction with Log-Euclidean Priors, Jose Seabra, Joao Xavier and Joao Sanches

13. Simultaneous Speckle Reduction and Contrast Enhancement for Ultrasound Images: Wavelet Versus Laplacian Pyramid, Ali S. Saad

14. http://www.fp.ucalgary.ca/mhallbey/

15. http://www.mathworks.com/matlabcentral

16. http://www.fatty-liver.com/fatty-liver-ultrasound-pictures/

17. Haykin, Simon (1998). Neural Networks: A Comprehensive Foundation (2 ed.). Prentice Hall. ISBN 0132733501.

18. Belur V. Dasarathy, editor (1991) Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, ISBN 0-8186-8930-7

19. Nearest-Neighbor Methods in Learning and Vision, edited by Shakhnarovish, Darrell, and Indyk, The MIT Press, 2005, ISBN 0-262-19547-X

20. Donald I, MacVicar J, Brown TG. Investigation of abdominal masses by pulsed ultrasound. Lancet 1958;1(7032):1188-95. PMID 13550965

21. Edler I, Hertz CH. The use of ultrasonic reflectoscope for the continuous recording of movements of heart walls. Kungl Fzsiogr Sallsk i Lund Forhandl. 1954;24:5. Reproduced in Clin Physiol Funct Imaging 2004;24:118-36. PMID 15165281.

22. S. A. Kana (2003). Introduction to physics in modern medicine. Tsylor and Francis. ISBN 0-415-30171-8.

23. C. Kasai et al. Real-time two-dimensional blood flow imaging using an autocorrelation technique. IEEE Transactions on Sonics and Ultrasonics 1985:458-464.

24. Ohanyido FO,. Basic Sonology for Doctors in Low Income Settings. Healthquest 2005;3:23.

25. Bushberg JT (2002). The essential physics of medical imaging. Lippincott Williams and Wilkins. ISBN 0-683-30118-7.

26. ””’Fetal Biometry: Vertical Calvarial Diameter and Calvarial Volume ”. July 2000”. http://jdm.sagepub.com/cgi/content/abstract/1/5/205. Retrieved on 2008-09-27.

27. ””’3D BPD Correction”. July 2000”. http://www.obgyn.net/us/cotm/0007/3d-bpd-correction.htm. Retrieved on 2008-09-27.

28. Sonography of the female pelvic floor Clinical indications and techniques

29. A Review of Therapeutic Ultrasound: Effectiveness Studies, Valma J Robertson, Kerry G Baker, Physical Therapy . Volume 81 . Number 7 . July 2001

30. A Review of Therapeutic Ultrasound: Biophysical Effects, , Kerry G Baker, et al., Physical Therapy . Volume 81 . Number 7 . July 2001

31. The Gale Encyclopedia of Medicine, 2nd Edition Volume 1 A-B. Page no.4

32. ”Ellis, George FR, Williams, Ruth M.; Flat and Curved Space-Times 2nd Edition; Oxford University Press, 2000 ”. http://www.amazon.com/gp/reader/019850656
link. Retrieved on January 25 2008.

33. ”Doppler Ultrasound History”. http://www.obgyn.net/ultrasound/ultrasound.asp?page=fe
Retrieved on January 25 2008.

34. Merritt, CR (01 November 1989). ”Ultrasound safety: what are the issues?”. Radiology 173 (2): 304306. PMID 2678243. http://radiology.rsnajnls.org/cgi/reprint/173/
Retrieved on 2008-01-22.

35. Ang Jr., ES; Gluncic V, Duque A et al. (2006). ”Prenatal exposure to ultrasound waves impacts neuronal migration in mice”. Proc Natl Acad Sci USA 103 (34): 1290310. doi:10.1073/pnas.0605294103. PMID 16901978. http://www.pnas.org/cgi/content/full/103/34/12903. Retrieved on 2008-01-22.

36. Kieler H, Cnattingius S, Haglund B, Palmgren J, Axelsson O (2001). ”Sinistrality–a side-effect of prenatal sonography: a comparative study of young men”. Epidemiology (Cambridge, Mass.) 12 (6): 61823. PMID 11679787.

37. Bricker L, Garcia J, Henderson J, et al. (2000). ”Ultrasound screening n pregnancy: a systematic review of the clinical effectiveness, cost-effectiveness and women’s views”. Health technology assessment (Winchester, England) 4 (16): i-vi, 1193. PMID 11070816. http://www.hta.ac.uk/execsumm/summ416.htm.

38. ”History of the AIUM”. http://www.aium.org/aboutAIUM/timeline/1950.asp. Retrieved on November 15 2005.

39. ”The History of Ultrasound: A collection of recollections, articles, interviews and images”. www.obgyn.net. http://www.obgyn.net/us/us.asp?page=/us/news_articles/ul
history-toc. Retrieved on 2006-05-11.

40. Woo, Joseph (2002). ”A short History of the development of Ultrasound in Obstetrics and Gynecology”. ob-ultrasound.net. http://www.ob-ultrasound.net/history1.ht
Retrieved on 2007-08-26.

41. ”Doppler Ultrasound History”. www.obgyn.net. http://www.obgyn.net/displayarticle.asp?
Retrieved on 2006-05-11.

42. Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function Mathematics of Control, Signals, and Systems (MCSS), 2(4), 303314.

43. Medina J, Fernndez-Salazar LI, Garca-Buey L, Moreno-Otero R (2004). ”Approach to the pathogenesis and treatment of nonalcoholic steatohepatitis”. Diabetes Care 27 (8): 205766. doi:10.2337/diacare.27.8.2057.

44. Angulo P (2002). ”Nonalcoholic fatty liver disease”. N. Engl. J. Med. 346 (16): 122131. doi:10.1056/NEJMra011775. PMID 11961152.