Computer-Aided Diagnostic System for Liver Lesions

By

Sobia Nawaz
May, 2008.

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER ENGINEERING
in Fulfillment of the Requirements
for the Degree of

MASTERS OF SCIENCE

WITH A MAJOR IN SOFTWARE ENGINEERING

Supervised by:

Dr. Amir Hanif Dar

Thesis committee

Lt. Col. Sajid Nazir
Lt. Col. Dr. Farooque Azam
Dr. Shaleeza Sohail

Department of Computer Engineering,
College of E & ME
NUST

College of Electrical & Mechanical Engineering

National University of Science & Technology

Computer-Aided Diagnostic System for Liver Lesions

By

Sobia Nawaz
Nov, 2007.

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER ENGINEERING
in Fulfillment of the Requirements
for the Degree of

MASTERS OF SCIENCE

WITH A MAJOR IN SOFTWARE ENGINEERING

Supervised by:

Dr. Amir Hanif Dar

Thesis committee

?

Department of Computer Engineering,
College of E & ME
NUST

College of Electrical & Mechanical Engineering

National University of Science & Technology

Abstract

Liver diseases are among the leading causes of death worldwide. The most useful approach for controlling the growth of disease to reach at severe condition is to treat these diseases at the early stages. Early treatment requires early diagnosis, which needs an accurate and reliable diagnostic procedure. The aim of this study is to develop a computer-aided diagnostic (CAD) system to achieve aforementioned objective. Computed tomography (CT) is one of the most common and robust imaging techniques for the detection of liver lesions such as hepatocellular carcinoma. Although in recent years the quality of CT images has been significantly improved, however in some cases image interpretation by human beings is often limited. So we tried to develop an automated system to detect and classify liver anomalies using CT images. Region of interest from CT images was segmented using Active contours (snakes) algorithm [2] and segmented image is used to extract statistical features by co occurrence matrix [4]. To facilitate the classification of hepatic lesions, Support Vector Machine [10] and neural networks are used. The results show that it is possible to automatically identify patients with liver lesions like Hemengioma, Hepatoma or Cirhhosis based on texture features and that the machine performance is marvelous and can assist human experts.

TABLE OF CONTENTS

Computer-Aided Diagnostic System for Liver Lesions

By

Sobia Nawaz
Nov, 2007.

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER ENGINEERING
in Fulfillment of the Requirements
for the Degree of

MASTERS OF SCIENCE

WITH A MAJOR IN SOFTWARE ENGINEERING

Supervised by:

Dr. Amir Hanif Dar

Thesis committee

Brig. Dr Muhammad Younas Javed
Dr. Shaleeza Sohail
Dr. Ghalib Assadullah Shah

Department of Computer Engineering,
College of E & ME
NUST

College of Electrical & Mechanical Engineering

National University of Science & Technology

Abstract

Liver diseases are among the leading causes of death worldwide. The most useful approach for controlling the growth of disease to reach at severe level is to treat these diseases at the early stages. Early treatment requires early diagnosis, which needs an accurate and reliable diagnostic procedure. The aim of this study is to develop a computer-aided diagnostic (CAD) system to achieve aforementioned objective. Computed tomography (CT) is one of the most common and robust imaging techniques for the detection of liver lesions such as hepatocellular carcinoma. Although in recent years the quality of CT images has been significantly improved, however in some cases image interpretation by human beings is often limited. So we tried to develop an automated system to detect and classify liver anomalies using CT images. Region of interest from CT images was segmented using Active contours (snakes) algorithm [2] and segmented image is used to extract statistical features by co occurrence matrix [4]. To facilitate the classification of hepatic lesions, Support Vector Machine [10] and neural networks are used. The results show that it is possible to automatically identify patients with liver lesions like Hemengioma, Hepatoma or Cirhhosis based on texture features and that the machine performance is marvelous and can assist human experts.

Overview of Chapters

Chapter one contains a brief described of the problem statement i.e. CAD System for liver lesions.

Chapter two contains the introduction to the technique used for image acquisition i.e. Computed Tomography (CT).

Chapter three contains the image segmentation techniques, indicated by the range of examples. The method used in CAD system is based on Active contours.

Chapter four contains the description of the objects or regions that have been segmented out of an image. Detail of the Haralick co-occurrence matrices is given as it is used in the feature extraction of CAD system.

Chapter five contains various classification algorithms.

Chapter six contains the detailed introduction of Support Vector Machine, its formalization, nonlinear SVM and Kernel functions.

Chapter seven contains the introduction to Backpropagation, and its derivation.

TABLE OF CONTENTS

Chapter no 1

Introduction

1.1 Motivation

Liver diseases, especially liver cancers, are among the leading causes of death worldwide. Primary liver cancer (cancer that starts in the liver) affects approximately 1,000,000 people each year [1]. The most useful approach for reducing deaths due to liver diseases is to treat these diseases in the early stages. Early treatment requires early diagnosis, which requires an accurate and reliable diagnostic procedure. Medical image processing plays an essential role in early diagnosis. Nowadays, in modern medicine, there is a growing need of medical image analysis systems and software, in particular for computer-aided diagnosis. The overall objective of Computer Aided Diagnostic (CAD) systems incorporating medical imaging systems or subsystems is to enable the early diagnosis, disease monitoring, and better treatment. The advantages of these systems can be summarized as follows:

Standardization Diagnoses obtained from different laboratories using similar criteria can be verified.

Sensitivity Findings on a particular subject may be compared with a database of normal values and/or a decision can be made by a CAD system, that whether or not an abnormality exists.

Specificity Findings may be compared with databases for various diseases and/or a decision can be made by the CAD system with respect to the type of abnormality.

Equivalence Results from a series of examinations of the same patient may be compared to decide whether there is evidence of disease progression or of response to treatment. In addition, the findings of different CAD systems can be compared to determine which are more sensitive and specific.

Efficacy The results of different treatments can be more properly evaluated. Medical imaging provides vital information for CAD systems.

The evolution in medical image processing and artificial intelligence techniques has given researchers the opportunity to investigate the potential of computer-aided diagnostic (CAD) systems for the classification of liver tissues. Development of such systems consists of few major steps like image capturing, preprocessing the image, segmentation of the region of interest, texture analysis and extraction of the desired features from segmented images and finally classification of the image into different

diseases. Several techniques can be used for these steps, the techniques used at each step in this thesis are mentioned in the section 3.

Currently, the confirmed diagnosis used widely for the liver cancer is needle biopsy. The needle biopsy, however, is an invasive technique. It has risks of bleeding and infection and is generally not recommended. Therefore the examination of liver tissue pathology is performed with various medical imaging modalities such as ultrasonography (US), computed tomography (CT), or magnetic resonance imaging (MRI). One of the most common and robust imaging techniques for the detection of liver lesions such as hepatocellular carcinoma is CT. These medical images are interpreted by radiologists. Although in recent years the quality of CT images has been significantly improved, however in some cases image interpretation by human beings is often limited due to the non-systematic search patterns of themselves, the presence of structural noise in the image, and the presentation of complex disease states requiring the integration of vast amount of image data and clinical information.

Recently, computer-aided diagnosis (CAD) systems' output (from a computerized analysis of medical images) is used by the radiologists as a ''second opinion'' in detecting lesions, assessing extent of disease, and making diagnostic decisions, is being used to improve the interpretation components of medical imaging . In addition, computer-aided surgery (CAS) that is the future technology in which surgery is performed on computerized surgical planning and image-guided surgery by analyzing region-of-interest (ROI) in the medical image. Volume measurement is also of major importance in different fields of medical imaging where physicians need some quantitative assessments for surgical decisions. This thesis presents a technique for computer-aided identification of few liver lesions.

1.2 Background

Various approaches for CAD systems have been proposed, most of them using US B-scan, MR and CT images, based on different image characteristics, such as texture features, estimated from first and second-order gray level statistics, fractal dimension estimators and Gabor Texture. The usefulness of texture analysis for medical images dates back to the 1970s, when images were first digitized. Suttan and Hall used texture measures in pulmonary disease identification experiments to discriminate between

normal and abnormal lungs. Chien and Fu used texture measures for the classification of lung diseases and Hall performed pneumoconiosis classification from radiographs of coal workers [2]. These studies were mostly aimed at signifying the usefulness of texture features to detect normal-abnormal class differences, combined with various classifiers. Texture analysis of liver CT images based on the Spatial Gray Level Dependence Matrix (SGLDM), has been applied to a probabilistic Neural Network (NN) for the characterization of hepatic tissue (Hepatoma and Hemangioma) [3]. A CAD to classify liver tissue has been proposed based on different sets of features and using ensembles of neural network classifiers [4]. These techniques helped to reduce the work load of radiologists, In this paper, a CAD system based on texture features of liver CT images, using multiple SVM classification scheme, is presented aiming to discriminating four hepatic tissue types: Normal, Hemangioma, Hepatoma and Cirrhosis.

The Liver

The Liver is thought to be one of the most important organs in the body as it is not only the largest organ in the human body but also plays a role in some vital bodily functions, The liver regulates most chemical levels in the blood and excretes a product called bile, which helps carry away waste products from the liver. All the blood leaving the stomach and intestines passes through the liver. The liver processes this blood and breaks down the nutrients and drugs into forms that are easier to use for the rest of the body. More than 500 vital functions have been identified with the liver [5]. Some of which include fuel management, nitrogen excretion, water balance and detoxification.

Interestingly the liver is the only organ that can regenerate itself when a portion is removed. It is positioned in the upper right of the abdomen under the lower left ribs as shown in the figure 1.1, and due to its softness will tend to be shaped differently in each person. Almost a third of the total blood within the human body travels through the liver each minute so it is essential that a surgeon take extreme care while operating, to avoid the harm to major blood vessels [6].

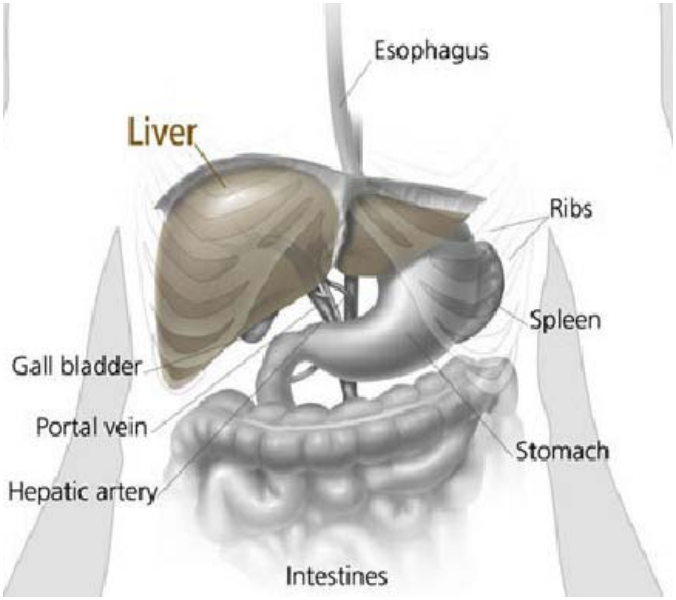


Figure 1.1: Location of liver in the body

Research in CAD for both mammogram and chest radiographs is rapidly growing; however, CAD research for liver lesions is to be insufficient because the liver segmentation that plays an important role for CAD is difficult. CAD systems help to reduce the work load of radiologists, as it is difficult, even for experienced doctors, to make a 100% accurate diagnosis. In order to assist clinicians in diagnosis and to reduce the number of required biopsies Computer Aided Diagnosis (CAD) systems for the characterization of hepatic tissue can be employed [52].

1.3 Methodology

The generic design of a CAD system is presented in Figure.1. Regions of Interest (ROIs) segmented by Active contour (energy minimizing) algorithm on CT images were driven to a feature extraction module, where six different statistical texture feature descriptors were measured on the co-occurrence matrices, which were calculated for five different distances and four different angles. The full feature sets were fed to the Support Vector Machine classifier. SVM classifier is trained and tested in hierarchical order where the first SVM classifier was able to classify between normal and diseased tissues, Image classified as diseased is then fed to the second SVM used to classify between Hemangioma and other disease (Not Hemangioma), Image classified as other disease is then fed to the third SVM which classifies between Hepatoma and Cirrhosis.

Figure 1.2: Generic design off CAD

1.4 Scope of the Project

- To provide an interface to the user to use CAD system.
- The user should be able to select an image file and segment its ROI.
- The user should be able to extract the features from ROI automatically.
- To provide the facility to train the system using support vector machine.
- To provide the facility to test the image for liver lesions.

1.5 Summary

The main purpose of the material presented in this chapter is to provide an introduction of the CAD system, its need in the medical diagnostic systems, and a brief description of the scope of the software, more importantly about past, current and future areas of application of this technology.

Chapter no 2

IMAGE ACQUISITION AND PREPROCESSING

2.1 Introduction

To design the Computer Aided Diagnostic (CAD) system, a database of CT images was acquired from human livers. The images were classified into different classes of liver lesions based on traditional histological analysis. This chapter describes the image acquisition methodology used, and how the database was created.

2.2 Image Acquisition

The study of medical imaging is concerned with the interaction of all forms of radiation with tissue and the development of appropriate technology to extract clinically useful information (usually displayed in an image format) from observation of this technology. The examination of liver tissue pathology is performed with various medical imaging modalities such as ultrasonography (US), computed tomography (CT), or magnetic resonance imaging (MRI). One of the most common and robust imaging techniques for the detection of liver lesions such as hepatocellular carcinoma is CT. Although in recent years the quality of CT images has been significantly improved, however in some cases image interpretation by human beings is often limited that's why even experienced radiologists resort to confirmation of diagnosis by administration of contrast agents or invasive procedures. Computerized decision support systems can be employed to assist clinicians in diagnosis and to reduce the number of required invasive procedures.

Figure 2.1: CT Scanning

2.3 Computed Tomography (CT)

In the mid 1970s, computed axial tomography (CAT) scanners became available, thus revolutionizing medical imaging. Cumbersome, expensive, and time consuming at first, and images are acquired slice-by-slice. An X-ray tube is rotating around patient to acquire a slice then patient is moved to acquire the next slice [7].

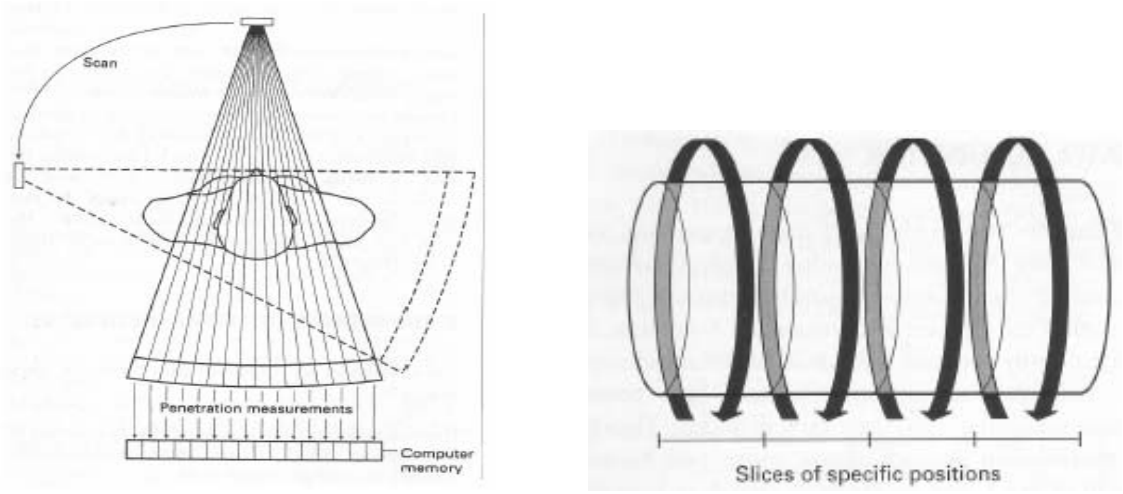


Figure 2.2: Slice-by-slice scanning

Newer generations of CT scanners permit helical (or spiral) scans of complete organ volumes within seconds. The fast scans allow images during breath-holding, thus minimizing respiratory motion artifacts. This technology can also be applied to the chest, abdomen, pelvis, brain and extremities. A patient is moved 10mm/s (24cm / single scan) to get 1mm-1cm thick slices, no shifting of anatomical structures and slice can be reconstructed with an arbitrary orientation with (a single breath) volume. We get better spatial resolution (better reconstruction) by parallel system of detectors which generates a large data of thin slices by capturing slices at a time.

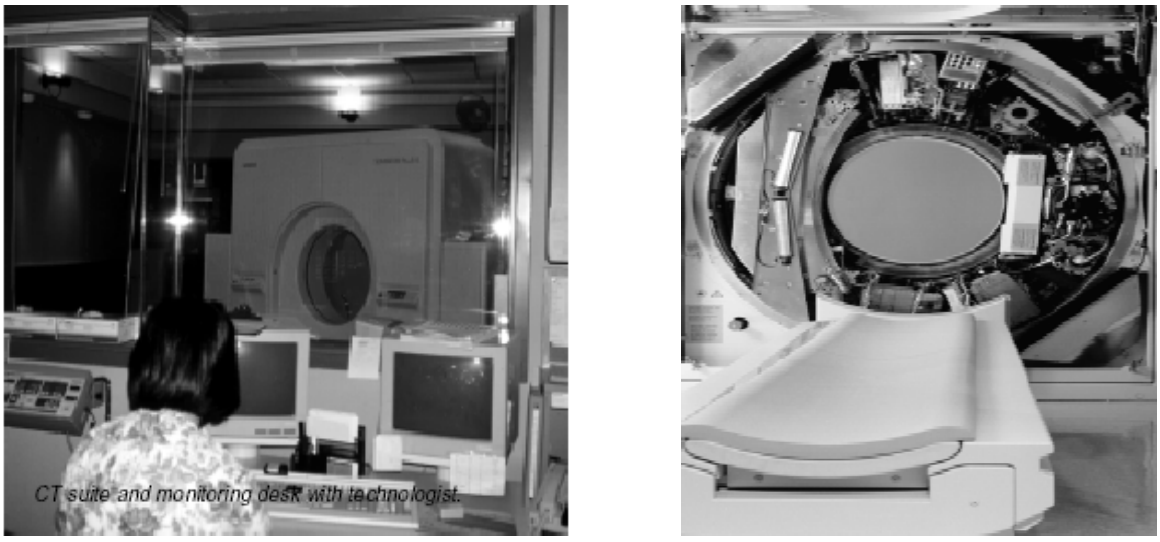


Figure 2.3: CT suite and monitoring desk

Within the gantry, a row of radiation detectors encircles the patient, while a rotating x-ray beam passes through the patient. The multiple transmitted beams are registered and back-projected, so that a transaxial "slice" of high resolution and contrast can be generated. Newer computational techniques have made it possible to create three-dimensional renderings as well as coronal (front to back) and sagittal (side to side) plane slices [7].

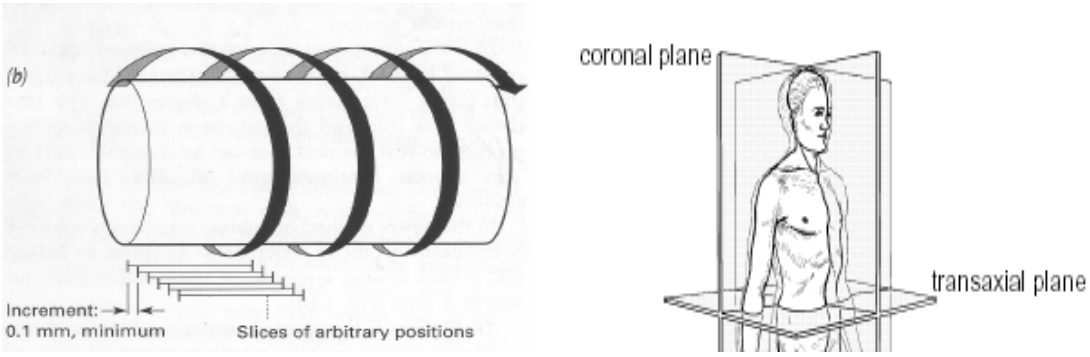


Figure 2.4: Spiral (volume) scanning

CT is an imaging based on a mathematical formalism that states that if an object is viewed from a number of different angles than a cross-sectional image of it can be computed (reconstructed).

2.4 Image Database

CT images of liver used in this project are collected from CT is us [8]. CT is us an organization created and maintained by The Advanced Medical Imaging Laboratory (AMIL). The AMIL is part of the Department of Radiology at the Johns Hopkins Medical Institutions in Baltimore, MD. AMIL is a multidisciplinary team dedicated to research, education, and the advancement of patient care using medical imaging with a focus on spiral CT and 3D imaging.

Each CT images is diagnosed by experience radiologists by traditional histological analysis. Normally medical images are available in DICOM format and their format is changed in order to use them in image processing, but images at CT is us are provided in JPEG format. So these images are used without any changes in their format. A database of 160 gray scale CT images from different patients was developed.

A subset of exemplar images of each class was selected from the original set: 150 images of (histology-verified) diseased tissue i.e. 50 images of Hemengioma, 50 images of Hepatoma and 50 images of Cirhhosis, and 32 images of normal tissue. This dataset was used to study texture classification schemes for liver lesions detection.

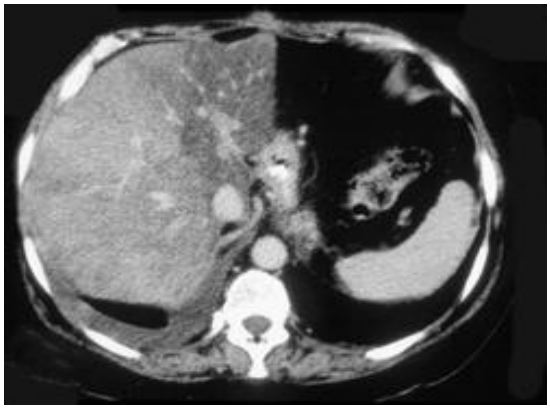


Figure 2.5: CT image containing liver

2.5 Summary

The main purpose of the material presented in this chapter is to provide introduction to the technique used for image acquisition i.e. Computed Tomography (CT). It's a way of capturing the image. After that detail of database used is provided.

Chapter no 3

Image Segmentation

3.1 Introduction

Segmentation subdivides an image into its constituent regions or objects. The level to which the subdivision is carried depends on the problem being solved. That is the segmentation should stop when the objects of interest in an application have been isolated. Segmentation accuracy determines the eventual success or failure of computerized analysis procedures. Image segmentation algorithms generally are based on one of two basic properties of intensity values: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in intensity, such as points, lines and edges in an image. The principal approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria. Thresholding, region growing, region splitting and merging are examples of methods in this category [9]. This thesis uses contour based techniques used for image segmentation, given in detail below.

3.2 Discontinuities based Segmentation

There are several techniques for detecting the three basic types of gray-level discontinuities in a digital image: points, lines and edges. The most common way to look for discontinuities is to run a mask through the image.

The detection of isolated points in an image is straightforward in principle. Using the mask shown in the Figure 3.1, we say that a point has been detected at the location on which the mask is centered.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 3.1: Point detection mask

An isolated point (a point whose gray level is significantly different from its background and which is located in a homogenous or nearly homogenous area) will be quite different

from its surroundings, and thus be easily detectable by this type of mask. Note that the mask coefficients sum to zero, indicating that the mask response will be zero in the area of constant gray level [10].

Consider the masks shown in Figure 3.2, if the first mask was moved around an image, it would respond more strongly to lines (one pixel thick) oriented horizontally. With a constant background, the maximum response would result when the line passed through the middle row of the mask. The second mask responds best to lines oriented at +45°; the third mask to vertical lines; and the fourth mask to lines in the -45° direction. The preferred direction of each mask is weighted with a larger coefficient (i.e., 2) than other possible directions.

-1	-1	-1
2	2	2
-1	-1	-1

-1	-1	2
-1	2	-1
2	-1	-1

-1	2	-1
-1	2	-1
-1	2	-1

2	-1	-1
-1	2	-1
-1	-1	2

Figure 3.2: Line detection masks

Edge detection is the most common approach for detecting meaningful discontinuities in gray level. An edge is a set of connected pixels that lie on the boundary between two regions. First and second order derivatives are used for the detection of edges in an image.

The first derivative is positive at the points of transition into and out of the ramp as we move from left to right along the profile; it is constant for the points in the ramp; and is zero in the areas of constant gray level. The second derivative is positive at the transition associated with the dark side of the edge, negative at the transition associated with the light side of the edge, and zero along the ramp and in areas of constant gray levels.

Figure 3.3: Two regions separated by vertical edge

Magnitude of the first derivative is used to detect the presence of an edge at a point in an image (i.e., to determine if a point is on a ramp). Similarly, the sign of the second derivative can be used to determine whether an edge pixel lies on the dark or light side of an edge.

3.3 Region-Based Segmentation

In this section we discuss segmentation techniques that are based on finding the regions directly. Theses are region growing, region splitting and region merging.

3.3.1 Basic formulation

Let R represent the entire image region. We may view segmentation as a process that partitions R into n subregions, R1,R2,….Rn, such that

(a) $\bigcup_{i=1}^{n} Ri = R.$

(b) R_i is a connected region, i=1,2…., n.

(c) $R_i \cap R_j = \emptyset$ for all i and j, $i \neq j$.

(d) $P(R_i)$= TRUE for i=1,2,3,…,n.

(e) $P(R_i \cup R_j)$=FALSE for $i \neq j$.

Here, $P(R_i)$ is a logical predicate defined over the points in set R_i and $\emptyset$ is the null set.

Condition (a) indicates that the segmentation must be complete; that is, every pixel must be in region. Condition (b) requires that points in a region must be connected in some predefined sense. Condition (c) indicates that the region must be disjoint. Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region, for example $P(R_i)$=TRUE if all pixels in R_i have the same gray level. Finally, Condition (e) indicates that regions R_i and R_j are different in the sense of predicate P.

3.3.2 Region Growing

Region growing is a procedure that groups pixels or sub regions into larger regions based on predefined criteria. The basic approach is to start with a set of "seed" points and form these grow regions by appending to each seed those neighboring pixels that have properties similar to the seed (such as specific range of gray level or color). Selecting a set of one or more starting points often can be based on the nature of the problem. When a priori information is not available, the procedure is to compute at every pixel the same set of properties that ultimately will be used to assign pixels to regions during the growing process. If the result of these computations shows clusters of values, the pixels whose properties place them near the centriod of these clusters can be used as seeds [11].

The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available. For example the analysis of land-use satellite imagery depends heavily on the use of color. This problem would be significantly more difficult, or even impossible, to handle without the inherent information available in color images. When the images are monochrome, region analysis must be carried out with the set of descriptors based on gray levels and spatial properties (such as moments or texture).

Descriptors alone can yield misleading results if connectivity or adjacency information is not used in the region-growing process. For example, visualize a random arrangement of pixels with only three distinct gray-level values. Grouping pixels with the same gray level to form a "region" without paying attention to connectivity would yield a segmentation result that is meaningless in the context of this discussion.

Another problem in region growing is the formation of a stopping rule. Basically, growing a region should stop when no more pixels satisfy the criteria for inclusion in the region. Criteria such as gray level, texture, and color are local in nature and do not take into account the "history" of region growth. Additional criteria that increase the power of region-growing algorithm utilize the concept of size, likeness between candidate pixel and the pixel grown so far(such as comparison of gray level of a candidate and the average gray level of grown region), and the shape of the region being grown. The use of these types of descriptors is based on the assumptions that a modal of expected results is at least partially available.

a	b
c	d

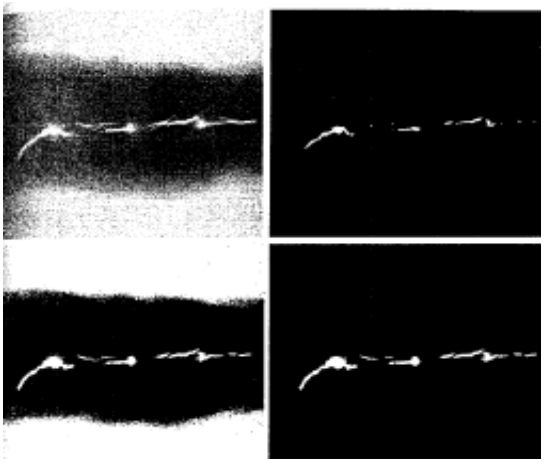


Figure 3.4: (a) Image showing defective welds. (b) Seed points. (c) Result of region growing. (d) Result after all the pixels in (c) were analyzed for 8-connectivity to the seed points

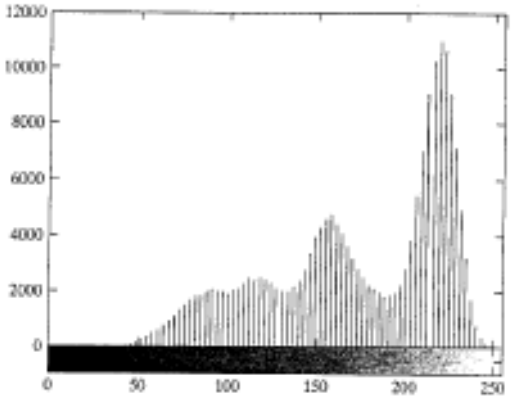


Figure 3.5: Histogram of Fig. 3.4(a)

Figure 3.4(a) shows an X-ray image of a weld (the horizontal dark region) containing several cracks and porosities (the bright, white streaks running horizontally through the middle of the image). Region growing is used to segment the regions of weld failures. These segmented features could be used for inspection, for inclusion in the database of historical studies, for controlling an automated welding system, and for other numerous applications. The first order of business is to determine the initial seed points. In this

application, it is known that the pixels of defective welds tend to have the maximum allowable digit value (255 in this case). Based on this information, we selected as starting points all pixels having values of 255 the points thus extracted from the original image are shown in the Figure 3.4(b). Note that many of the points are clustered into seed regions.

The next step is to choose criteria for region growing. In this particular example we choose two criteria for a pixel to be annexed to a region [12]:

(1) The absolute gray level difference between any pixel and the seed had to be less than 65. This number is based on the histogram shown in the Figure 3.5 and represents the difference between 255 and the location of the first major valley to the left, which is representative of the highest gray level value in the dark region.

(2) To be included in one of the regions, the pixel had to be 8-connected to at least one pixel in that region. If a pixel was found to be connected to more than one region, the region was merged.

Figure 3.4(c) shows the regions that resulted by starting with the seeds in Figure 3.4(b) and utilizing the criteria defined previously. Superimposing the boundaries of these regions on the original image Figure 3.4(d) reveals that the region-growing procedure did indeed segment the defective welds with an acceptable degree of accuracy. It is of interest to note that it was not necessary to specify stopping rule in this case because the criteria for region growing were sufficient to isolate the features of interest.

Problems having multimodal histograms are generally best solved using region-based approaches. The histogram shown in Figure 3.5 is an excellent example of clean multimodal histogram.

3.3.3 Region Splitting and Merging

The procedure just discussed grows region from a set of seed points. Another alternative is to subdivide an image initially into a set of arbitrary, disjointed regions and then merge and/or split the regions in an attempt to satisfy the conditions stated in section 3.2.1. A split and merge algorithm that iteratively works toward satisfying these constraints is developed next.

Let R represent the entire image region and select a predicate P. One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i, $P(R_i)$ = TRUE. We start with the entire region. If P(R) = FALSE, we divide the image into quadrants. If P is FALSE for any quadrant, we subdivide that quadrant into subquadrant, and so on. This particular splitting technique has a convenient representation in the form of a so called quadtree (that is, a tree in which nodes have exactly four descendants), as illustrated in Figure 3.7. Note that the root of the tree corresponds to the entire image and that each node corresponds to a subdivision. In this case, only R_4 was subdivided further.

<table>
<tr><td rowspan="2">R_1</td><td colspan="2" rowspan="2">R_2</td></tr>
<tr></tr>
<tr><td rowspan="2">R_3</td><td>R_{41}</td><td>R_{42}</td></tr>
<tr><td>R_{43}</td><td>R_{44}</td></tr>
</table>

Figure 3.6: Partitioned image

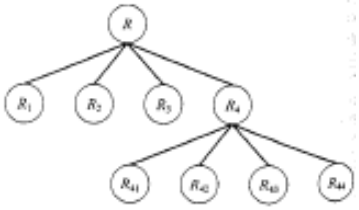


Figure 3.7: Corresponding quadtree

If only splitting were used, the final partition likely would contain adjacent regions with identical properties. This drawback may be remedied by allowing merging, as well as splitting. Satisfying the constraints of section 3.2.1 requires merging only adjacent regions whose combined pixels satisfy the predicate P. That is, two adjacent regions R_j and R_k are merged only if $P(R_j \cup R_k)$ = TRUE. The preceding discussion may be summarized by the following procedure, in which, at any step we

1. Split into four disjoint quadrants any region R_i for which $P(R_i) = FALSE$.

2. Merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k) = TRUE$.

3. Stop when no further merging or splitting is possible.

Several variations of the preceding basic theme are possible. For example one possibility is to split the image initially into a set of blocks. Further splitting is carried out as described previously, but merging is initially limited to groups of four blocks that are descendants in the quadtree representation and that satisfy the predicate P. when no further mergings of this type are possible, the procedure is terminated by one final merging of regions satisfying step 2. At this point, the merged regions may be of different sizes. The principal advantage of this approach is that it uses the same quadtree for splitting and merging, until the final merging step.

3.4 Active contours

Active contours, or snakes, are computer-generated curves that move within images to find object boundaries (note that the 3D version is often known as deformable models or active surfaces in the literature). They are often used in computer vision and image analysis to detect and locate objects, and to describe their shape. For example, a snake might be used to automatically find a manufactured part on an assembly line; one might be used to find the outline of an organ in a medical image; or one might be used to automatically identify characters on a postal letter. The shape of many objects is not easily represented by rigid primitives. For example natural objects, such as bananas, have similar recognizable shapes. But no two bananas are exactly the same. In medical imaging, objects are similar but not exact. And some objects, such as lips, change over time [13].

The segmentation of structure from 2D and 3D images is an important first step in analyzing medical data. For example, it is necessary to segment the brain in an MR image, before it can be rendered in 3D for visualization purposes. Segmentation can also be used to automatically detect the head and abdomen of a fetus from an ultrasound image. The boundaries can then be used to get quantitative estimates of organ sizes and provide aid in any necessary diagnoses. Another important application is registration.

It may be easier or at least less error prone to segment objects in multiple images prior to registration. This is especially true in images from different modalities such as CT and MRI. Image-guided surgery is one other important application of segmentation. Recent advances in technology have made it possible to acquire images of the patient while the surgery is taking place. The goal is then to segment relevant regions of interest and overlay them on an image of the patient to help guide the surgeon in his work. Segmentation is therefore a very important task in medical imaging. However, manual segmentation is not only a tedious and time consuming process, it is also inaccurate. It is therefore desirable to use algorithms that are accurate and require as little user interaction as possible.

3.4.1 Background

Active contours or "snakes" can be used to segment objects automatically. The basic idea is the evolution of a curve, or curves subject to constraints from the input data. The curve should evolve until its boundary segments the object of interest. This framework has been used successfully by Kass to extract boundaries and edges. One potential problem with this approach is that the topology of the region to be segmented must be known in advance. During evolution, curves may change connectivity and split. Although this topological constraint may be reasonable in the segmentation of the liver, it would certainly be undesirable when segmenting blood vessels in an MR image. An algorithm to overcome these difficulties was first introduced by Osher and Sethian. They model the propagating curve as a specific level set of a higher dimensional surface. It is common practice to model this surface as a function of time. So as time progresses, the surface can change to take on the desired shape [14].

3.4.2 Level Sets

Mathematical Formulation

Let Ω be a bounded open subset of R^2, with $\partial\Omega$ as its boundary. Then a two dimensional image u_0 can be defined as u_0: $\Omega \rightarrow$ R. In this case Ω is just a fixed rectangular grid. Now consider the evolving curve C in Ω, as the boundary of an open subset ω of Ω. In other words, $\omega \subseteq \Omega$, and C is the boundary of ω ($C = \partial\omega$) [15].

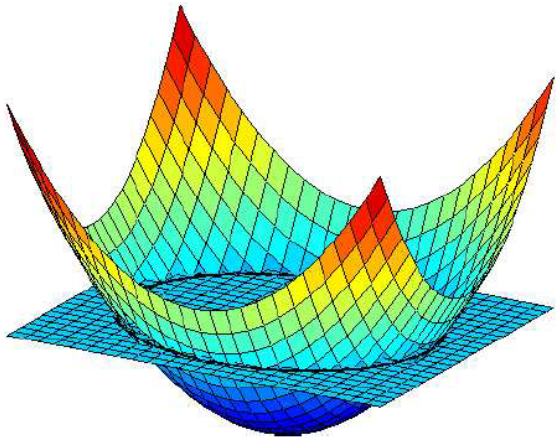


Figure3.8: Intersection of higher dimensional surface with the data set results in a level set

The main idea is to embed this propagating curve as the zero level set of a higher dimensional function Φ. We define the function as follows:

$$\Phi(x, y, t = 0) = \pm d \tag{3.1}$$

Where d is the distance from (x, y) to $\partial\omega$ at t = 0, and the plus (minus) sign is chosen if the point (x, y) is outside (inside) the subset ω. Now, the goal is to produce an equation for the evolution of the curve. Evolving the curve in the direction of its normal amounts to solving the partial differential equation:

$$\frac{\partial \phi}{\partial t} = F\,|\nabla \phi\,|, \phi(x, y, 0) = \phi_0(x, y) \tag{3.2}$$

Where the set {(x, y), Φ_0(x, y) = 0} defines the initial contour, and F is the speed of propagation. For certain forms of the speed function F, this reduces to a standard Hamilton-Jacobi equation. There are several major advantages to this formulation. The

first is that Φ(x, y, t) always remains a function as long as F is smooth. As the surface Φ evolves, the curve C may break, merge, and change topology.

Another advantage is that geometric properties of the curve are easily determined from a particular level set of the surface Φ. For example, the normal vector for any point on the curve C is given by:

$$\vec{n} = \nabla\phi$$

and the curvature K is obtained from the divergence of the gradient of the unit normal vector to the front:

$$k = div\left(\frac{\nabla\phi}{|\nabla\phi|}\right) = \frac{\phi xx\phi^2 y - 2\phi x\phi y\phi xy + \phi yy\phi^2 x}{\left(\phi^2 x + \phi^2 y\right)^{\frac{3}{2}}} \tag{3.3}$$

Finally, another advantage is that we are able to evolve curves in dimensions higher than two. The above formulae can be easily extended to deal with higher dimensions. This is useful in propagating a curve to segment volume data.

There are different active contour models, one based on an edge-stopping function, while other is an energy minimization algorithm. The first method can only detect objects defined by a strong gradient, while the second method does not have this constraint. Both methods can be put into a level-set framework using a Lipschitz function Φ for automatic topology changes. Second model is implemented, so discussed in detail and validated with numerical results.

3.4.3 Active Contours without Edges

As the curve C can be viewed as the boundary of an open subset ω of Ω (i.e. C = $\partial\omega$). Denote the region ω by inside(C) and the region $\Omega \setminus \overline{\omega}$ by outside(C). Now rather than basing the model on an edge-stopping function, we will halt the evolution of the curve with an energy minimization approach [16].

Consider a simple case where the image u_0 is formed by two regions of piecewise constant intensity. Denote the intensity values by u_0^0 and u_0^1. Furthermore, assume that the object to

be detected has a region whose boundary is C_0 and intensity u_0^1. Then inside (C_0), the intensity of u_0 is approximately u_0^1, whereas outside (C_0) the intensity of u_0 is approximately u_0^0. Then consider the fitting term:

$$F_1(C) + F_2(C) = \int_{\text{inside}(C)} |u_0(x,y) - c_1|^2 \, dxdy + \int_{\text{outside}(C)} |u_0(x,y) - c_2|^2 \, dxdy \quad (3.4)$$

Where C is a curve, and the constants c1, c2 are the averages of u0 inside and outside of C respectively. Consider Figure 3.9. If the curve C is outside the object, then F1(C) > 0, F2(C) ≈ 0. If the curve is inside the object, then F1(C) ≈ 0, F2(C) > 0. If the curve is both inside and outside the object, then F1(C) > 0; F2(C) > 0. However, if the curve C is exactly on our object boundary C_0, then F1(C) ≈ 0; F2(C) ≈ 0, and our fitting term is minimized.

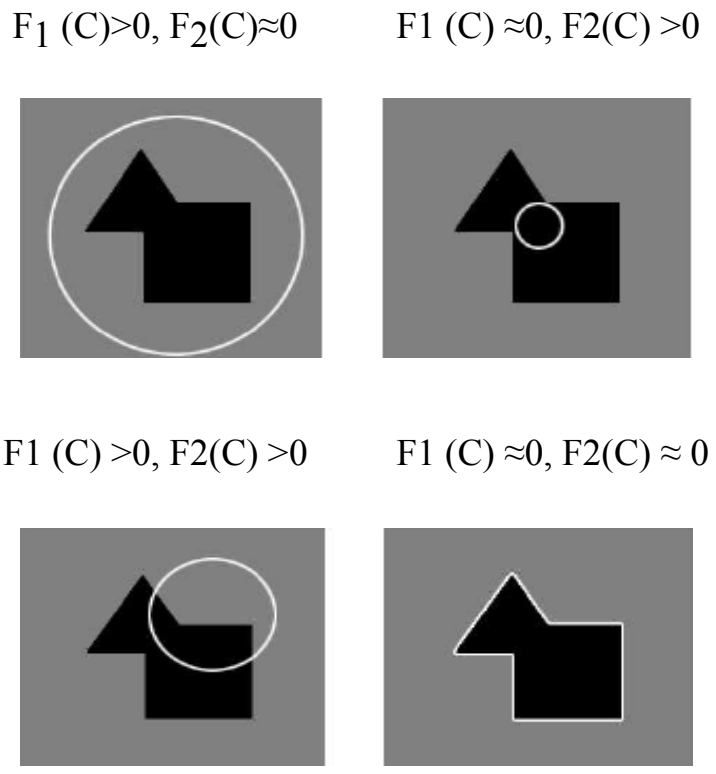


Figure 3.9: All possible cases in position of the curve

We also consider adding some regularization terms. Therefore we will also try to minimize the length of the curve and the area of the region inside the curve. So we introduce the energy function E:

$$E(C,c_1,c_2) = \mu \cdot \text{Length}(C) + \nu \cdot \text{Area}(\text{inside}(C)) + \lambda_1 \int_{\text{inside}(C)} |u_0(x,y) - c_1|^2 \, dxdy + \lambda_2 \int_{\text{outside}(C)} |u_0(x,y) - c_2|^2 \, dxdy$$

Where $\mu \geq 0$, $\nu \geq 0$, $\lambda_1 > 0$, $\lambda_2 > 0$ are fixed parameters. So our goal is to find C; c_1, c_2 such that E(C, c1, c2) is minimized. Mathematically, we want to solve:

$$\inf_{C,c_1,c_2} E(C, c_1, c_2)$$

This problem can be formulated using level sets as follows. The evolving curve C can be represented by the zero level set of the signed distance function ϕ as in (3.1). So we replace the unknown variable C by ϕ. Now consider the Heaviside function H, and the Dirac measure δ:

$$H(z) = \begin{cases} 1, & if \quad z \geq 0 \\ 0, & if \quad z < 0 \end{cases} \qquad , \delta(z) = \frac{d}{dz} h(z) \tag{3.5}$$

We can rewrite the length of Φ = 0 and the area of the region inside (Φ = 0) using these functions. The Heaviside function is positive inside our curve and zero elsewhere, so the area of the region is just the integral of the Heaviside function of Φ. The gradient of the Heaviside function defines our curve, so integrating over this region gives the length of the curve. Mathematically:

$$Area(\phi = 0) = \int_\Omega H(\phi(x,y))dxdy \tag{3.6}$$

$$Length(\phi = 0) = |\nabla \int_\Omega H(\phi(x,y))|dxdy \tag{3.7}$$

$$= \int_\Omega \delta(\phi(x,y)) \, |\nabla \phi(x,y)| \, dxdy \tag{3.8}$$

Similarly, we can rewrite the previous energy equations so that they are defined over the entire domain Ω rather than separated into inside (C) = $\Phi > 0$ and outside (C) = $\Phi < 0$:

$$\int_{\phi>0} |u_0(x,y)-c_1|^2\,dxdy = \int_{\Omega} |u_0(x,y)-c_1|^2 H(\phi(x,y))dxdy \tag{3.9}$$

$$\int_{\phi>0} |u_0(x,y)-c_2|^2\,dxdy = \int_{\Omega} |u_0(x,y)-c_2|^2 (1-H(\phi(x,y)))dxdy \tag{3.10}$$

Therefore our energy function E(C, c1, Φ) can be written as:

$$\begin{aligned} E(C,c_1,c_2) = {} & \mu \int_{\Omega} \delta(\phi(x,y)) \,|\nabla\phi(x,y)dxdy \\ & + \nu \int_{\Omega} H(\phi(x,y))dxdy \\ & + \lambda_1 \int_{\Omega} |u_0(x,y) - c_1|^2 H(\phi(x,y))\,dxdy \\ & + \lambda_2 \int_{\Omega} |u_0(x,y) - c_2|^2 (1-H(\phi(x,y)))\,dxdy \end{aligned} \tag{3.11}$$

The constants c_1, c_2 are the averages of u_0 in $\Phi \geq 0$ and $\Phi < 0$ respectively.
So they are easily computed as:

$$c_{1(\phi)} = \frac{\int_{\Omega} u_0(x,y)H(\phi(x,y))dxdy}{\int_{\Omega} H(\phi(x,y))dxdy} \tag{3.12}$$

and

$$c_{2(\phi)} = \frac{\int_{\Omega} u_0(x,y)(1-H(\phi(x,y)))dxdy}{\int_{\Omega} (1-H(\phi(x,y))dxdy} \tag{3.13}$$

Now we can deduce the Euler-Lagrange partial differential equation from (3.11). We parameterize the descent direction by $t \geq 0$, so the equation $\Phi(x, y, t)$ is:

$$\frac{\partial\phi}{\partial t} = \partial(\phi)[\mu div(\frac{\nabla\phi}{|\nabla\phi|}) - \nu - \lambda_1(u_0-c_1)^2 + \lambda_2(u_0-c_2)^2] = 0$$

(3.14)

In order to solve this partial differential equation, we first need to regularize H(z) and δ(z). Chan and Vese propose:

$$H_\varepsilon(z) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{z}{\varepsilon}\right) \tag{3.15}$$

Implying that δ(z) regularizes to:

$$\delta_\varepsilon(z) = \frac{1}{\pi}\cdot\frac{\varepsilon}{\varepsilon^2 + z^2} \tag{3.16}$$

It is easy to see that as ε→ 0, H ε(z) converges to H(z) and $\delta_\varepsilon(z)$ converges to δ (z). The authors mention that with these regularizations, the algorithm has the tendency to compute a global minimizer. Chan and Vese give the following discretization and linearization of (3.14):

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} = \delta_\varepsilon \phi_{i,j}^{n}\left[\begin{array}{l} \dfrac{\mu}{h^2}\Delta^x - \left(\dfrac{\Delta^x + \phi_{i,j}^{n+1}}{\sqrt{(\Delta^x + \phi_{i,j}^{n})^2/(h^2) + (\phi_{i,j+1}^{n} - \phi_{i,j-1}^{n})^2/(2h)^2}}\right) \\ + \dfrac{\mu}{h^2}\Delta^y - \left(\dfrac{\Delta^y + \phi_{i,j}^{n+1}}{\sqrt{(\phi_{i+1,j}^{n} - \phi_{i-1,j}^{n})^2/(2h^2) + (\Delta^y + \phi_{i,j}^{n})^2/(h)^2}}\right) \\ -\nu - \lambda_1(u_{0,i,j} - c_1(\phi^n))^2 + \lambda_2(u_{0,i,j} - c_2(\phi^n))^2 \end{array}\right] \tag{3.17}$$

This linear system also depends on the forward differences of $\phi_{i,j}^{n+1}$, which is an unknown. However these can be solved using the Jacobi method. In practice, the number of iterations until convergence was found to be small. The segmentation algorithm is [51]:

Algorithm 4: Energy Minimization Algorithm with Jacobi Method

Initialize Φ^0 by $\Phi_{0,\, n=0}$

for fixed number of iterations **do**

Compute $c_1(\Phi^n)$ and $c_2(\Phi^n)$ by (3.12) and (3.13)

Estimate forward differences of Φ^{n+1} using Jacobi method

Compute Φ^{n+1} by (3.17)

end

Another way to discretize and linearize the system is to directly estimate the curvature K from $\phi_{i,j}^n$ using (3.3). This leads to a much simpler system where the Jacobi method is not necessary:

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^n}{\Delta t} = \delta_\varepsilon \phi_{i,j}^n [\mu \mathrm{K} - \nu - \lambda_1 (u_{0,i,j} - c_1(\phi^n))^2 + \lambda_2 (u_{0,i,j} - c_2(\phi^n))^2] \qquad (3.18)$$

So the second simpler algorithm is:

Algorithm 5: Energy Minimization Algorithm

Initialize Φ^0 by $\Phi_{0,\, n=0}$

for fixed number of iterations **do**

Compute $c_1(\Phi^n)$ and $c_2(\Phi^n)$ by (3.12) and (3.13)

Compute curvature term K by (3.3)

Compute Φ^{n+1} by (3.18)

End

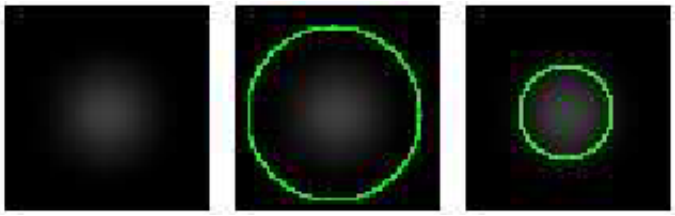


Figure 3.10: Objects with smooth contours can not be segmented by the edge-stopping approach

The curve will eventually collapse in on itself. However, using the energy minimization approach, we achieve the desired segmentation.

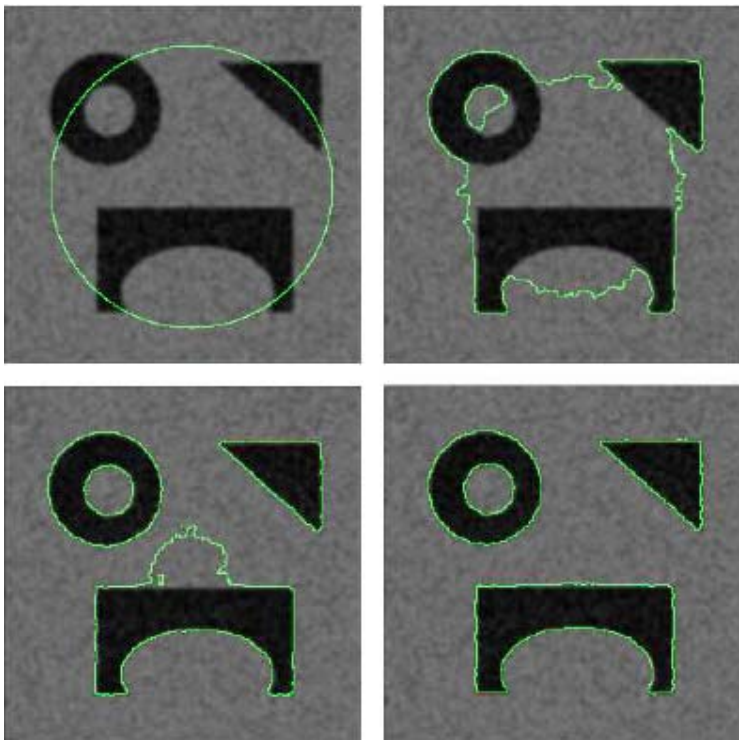


Figure 3.11: Artificial image corrupted with Gaussian noise. Segmentation using energy minimization.

3.4.4 Runtime Analysis

Since we are dealing with sampled discrete data, it is unlikely that the signed distance function (3.1) is ever exactly zero. So we need to form an approximation for the level set $\Phi = 0$. We define a pixel $\Phi_{i,j}$ to be on the zero level set if there is a change of sign within a 4 pixel neighborhood. Mathematically, the pixel is on the front if:

$$\max(\phi_{i,j},\phi_{i+1,j},\phi_{i,j+1},\phi_{i+1,j+1}) > 0 \tag{3.19}$$

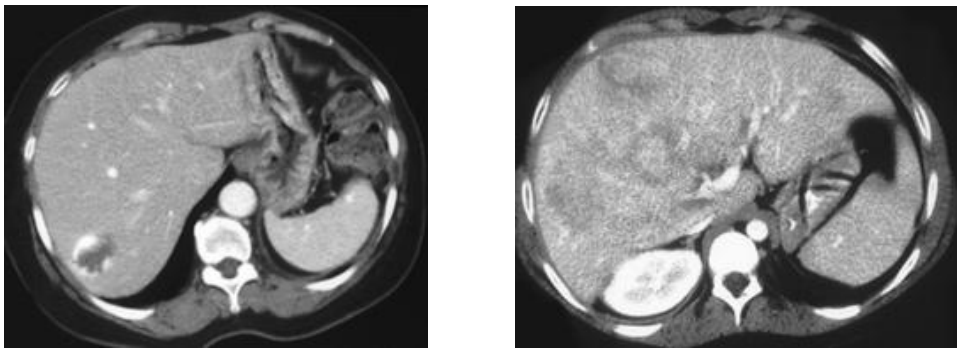
and

$$\max(\phi_{i,j},\phi_{i+1,j},\phi_{i,j+1},\phi_{i+1,j+1}) < 0 \tag{3.20}$$

First, assume that an image to be segmented is n×n. (It does not have to be square, but it makes the analysis slightly easier). Finding the points on our curve will take $O(n^2)$ time. A reasonable assumption is that the total number of pixels in our advancing curve is 0(n). This bound stems from the fact that our curve lies on the border of our object.

In practice, both energy minimization algorithms can be analyzed the same way, as the Jacobi method converges in a small, constant number of time steps. Both algorithms compute c_1 and c_2, which are essentially just the averages of u_0 inside and outside the region defined by C respectively. This can be done in $O(n^2)$ time. Then it is simply a matter of updating each pixel in $\Phi_{i,j}$ for k iterations. This gives a time bound of $O(kn^2)$.

In all of the energy minimization experiments, the parameters were chosen as follows: $\lambda_1 = \lambda_2 = 1$, h = 1; and $\Delta t = 0.1$. Varying the value of v generally had little or no effect on the segmentation results. The ε parameter for the regularized Heaviside and Dirac functions was set to 1 as suggested by Chan and Vese. The length parameter μ was varied between images. For images in which as many objects as possible have to be detected μ should be small (i.e. $\mu \approx 0.000 - 255^2$). For images in which we only wish to segment larger objects, we choose $\mu \approx 0.1 - 255^2$



(a) **(b)**

Figure 3.12: CT images of lower abdomen containing liver
(a) Liver with Hemengioma (b) Liver with Hepatoma

Figure 3.13: Segmentation of the above images (a) and (b) respectively using energy minimization

3.5 Summary

Image segmentation is an essential preliminary step in most automatic pattern recognition and scene analysis problems. As indicated by the range of examples presented in this chapter, the choice of one segmentation technique over another depends on the characteristics of the problem being considered. The method used in CAD system is based on Active contours.

Chapter no 4

Texture Analysis

4.1 Introduction

Visually, texture refers to the variation in image intensities which form certain repeated patterns. Texture is often used by physicians for diagnostic purposes. In medical image processing, texture is especially important, because it is difficult to classify human body organ tissues using shape or gray level information. This is because of the uncertainty introduced by the unlimited variability in organ shape distortion and the potential absolute gray level variability due to the imaging device. While gray levels purely describe point-wise properties of images, texture uses these gray levels to derive some notion of spatial distribution of tonal variations, surface orientation and scenic depth [50]. Furthermore, contrary to the discrimination of morphologic information (shape, size), there is evidence that the human visual system has difficulties in the discrimination of textural information that is related to higher-order statistics or spectral properties in an image. Consequently, texture analysis can potentially augment the visual skills of the radiologist by extracting features that may be relevant to the diagnostic problem but they are not necessarily visually extractable.

4.1.1 Texture

Texture may be defined as the local variation in intensity between pixels in a small region of an image. If pixel intensity were represented as a surface (elevation) then texture would describe the “roughness” of the surface.

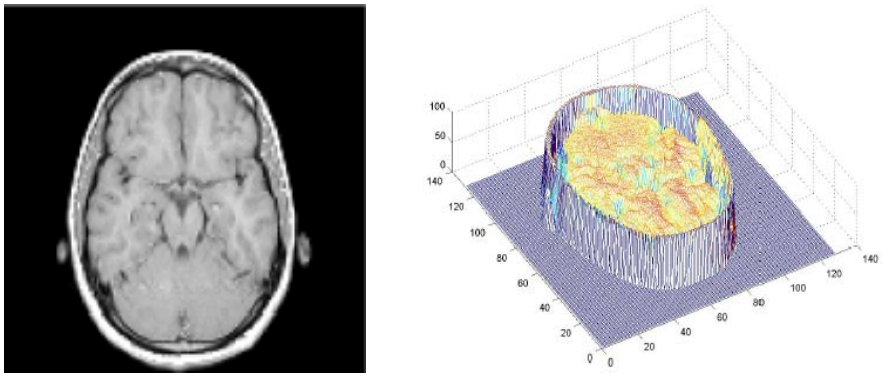


Figure 4.1: Texture Visualization

Although there is no accepted mathematical definition for image texture, it can be thought loosely of as repeated patterns of pixels. The addition of noise to the patterns and their repetition frequencies result in textures that can appear to be random and unstructured. Image texture can be used to:

- Recognize objects
- Determine the shape of objects
- Judge the condition of objects
- Detect diseases

Examples of texture

Following are the examples of 2-D textured images:

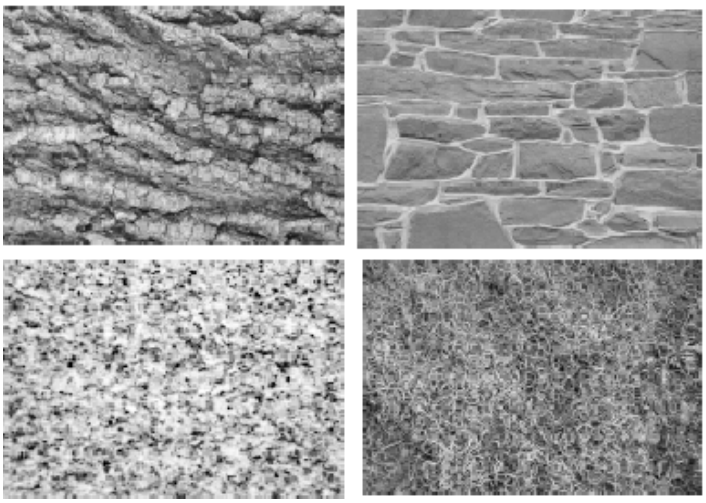
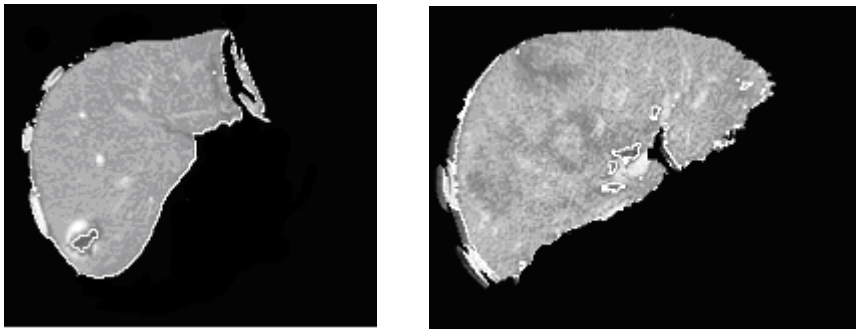


Figure 4.2: 2-D textured images

In digital images, texture refers to the spatial interrelationships and arrangement of image pixel intensities or gray levels. The task of texture discrimination is often easy for humans but is quite difficult to be modeled and performed by modern digital computers. At the core of this problem is the need to mathematically model the cellular-level structure present in these images as texture. Texture analysis is a way to achieve that; it results in a set of feature metrics that describe the characteristics of different tissue patterns.

4.1.2 Background

A large number of schemes have been proposed for texture feature extraction and this remains an active area of research. Some of the most popular methods are statistical methods, Fourier power spectrum-based methods, Laws' texture energy measures, geometrical methods, and random field model-based methods. More recently, texture features based on fractal-based models, Gabor and wavelet decomposition-based methods, run-length encoding, and co-occurrence matrices have received significant attention. While there has not been any conclusive study to prove the superiority of one method over the other methods of capturing texture, These techniques have been successfully applied in many fields including, remote sensing, industrial inspection, medical image analysis, and document image processing. Computer based methods of texture analysis were originally developed for use in satellite applications, geological surveys, remote sensing, and other related applications. For medical applications, texture operations can aid in the detection of small focal lesions against a uniform tissue background.



(a) **(b)**

Figure 4.3 (a) CT image of Hemangioma, (b) CT image of Hepatoma

In this study, texture features of the segmented images shown above, were extracted from Haralick co-occurrence method, which is a well known, established method that has been proven to correlate well with what experts generally look for in texture features. Also, it has been used successfully to produce good results in classification studies of normal tissues in CT images of chest and abdomen.

4.2 Texture Feature Extraction methods

A wide range of techniques, ranging from second order statistics to syntactic partitioning, is in existence. These techniques are broadly categorized into three groups: statistical, structural, and spectral. Statistical approaches yield characterization of textures as smooth, coarse, grainy and so on. Structural techniques deal with the arrangement of image primitives, such as the description of texture based on regularly spaced parallel lines [17]. Spectral techniques are based on properties of the Fourier spectrum and are used primarily to detect global periodicity in an image by identifying high energy, narrow peaks in the spectrum.

4.2.1 Statistical Approaches

One of the simplest approaches for describing texture is to use statistical moments of the gray level histogram of an image or region [17]. Let z be a random variable denoting gray levels and let $P(z_i)$, $i = 0,1,2\ldots,L-1$, be the corresponding histogram

$$P(zi) = \frac{h(z_i)}{N} \tag{4.1}$$

where $h(z_i)$ is the image histogram, and N is the total number of pixels in the image. The normalized image histogram can be thought of as a probability density function for the gray levels of the image. If $p(z_i)$ denotes the probability of each gray-level value z_i, the following features can be calculated:

Mean estimates the value around which central clustering occurs:

$$\mu = \sum_{zi=0}^{L-1} (z_i)p(z_i) \tag{4.2}$$

where $L = 2^B$ is the number of quantized gray levels, where B is the number of bits.

Standard Deviation measures the variability about the mean and is computed as:

$$\sigma = \left[\sum_{zi=0}^{L-1} (z_i - \mu)^2 p(z_i)\right]^{1/2} \tag{4.3}$$

The second moment [the variance $\sigma^2(z) = \mu_2(z)$] is of particular importance in texture description. It is a measure of gray-level contrast that can be used to establish descriptors of relative smoothness. For examples, the measure

$$R = 1 - \frac{1}{1 + \sigma^2(z)} \tag{4.4}$$

is 0 for areas of constant intensity (the variance is zero there) and approaches 1 for large values of $\sigma^2(z)$. Because variance values tend to be large for gray scale images with values, for example, in the range 0 to 255, it is a good idea to normalize the variance to the interval [0, 1] for use in Eq. (4.4).This is done simply by dividing $\sigma^2(z)$ by $(L-1)^2$ in

Eq. (4.4). The standard deviation, $\sigma(z)$, also is used frequently as a measure of texture because values of the standard deviation tend to be more intuitive to many people.

Coefficient of Variation is another measure of the deviation of a variable from its mean. It is calculated as follows:

$$\eta = \frac{\sigma}{\mu} \tag{4.5}$$

Skewness or the third moment characterizes the degree of asymmetry of a distribution around its mean. A positive value of skewness indicates a distribution with an asymmetric tail extending out towards larger z_i values; a negative value indicates a distribution whose tail extends towards smaller z_i values. Skewness is given by:

$$skew = \sigma^{-3} \sum_{i=0}^{L-1} (z_i - \mu)^3 \, p(z_i) \tag{4.6}$$

Kurtosis or the fourth moment measures the peakedness or flatness of a distribution relative to a normal distribution. A positive kurtosis generally indicates a more peaked distribution, termed *leptokurtic*, whereas a negative kurtosis indicates a more flat curve, and is termed *platykurtic*. The conventional definition of kurtosis is

$$kurt = \sigma^{-4} \sum_{i=0}^{L-1} (z_i - \mu)^4 \, p(z_i) - 3 \tag{4.7}$$

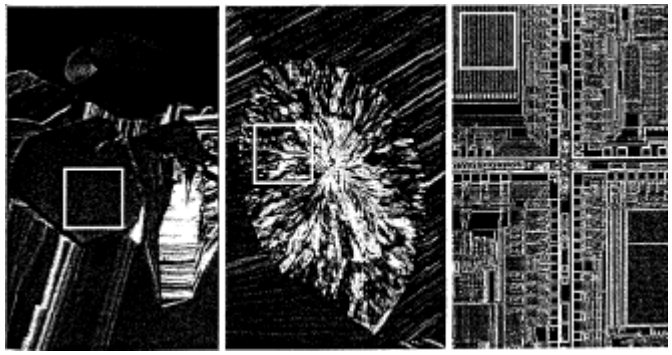
The fifth and higher moments are not so easily related to histogram shape, but they do provide further quantitative discrimination of texture content. Some useful additional texture measures based on histograms include a measure of "uniformity," given by

$$U = \sum_{i=0}^{L-1} p^2(zi) \tag{4.8}$$

and an average entropy measure is defined as

$$e = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(zi) \tag{4.9}$$

Because the p's have values in the range[0,1] and their sum equals 1, measure U is maximum for an image in which all gray levels are equal (maximally uniform), and decreases from there. Entropy is a measure of variability and is 0 for a constant image.



a **b** **c**

Figure 4.4: The white square mark, from left to right, smooth, coarse, and regular textures. These are optical microscope images of a superconductor, human cholesterol, and a microprocessor respectively

Table 4.1 summarizes the values of the preceding measures for the three types of textures (smooth, coarse and regular) highlighted in Figure 4.4. The mean just tells us the average gray level of each region and is useful only as a rough idea of intensity, not really texture. The standard deviation is much more informative; the numbers clearly show that the first texture has significantly less variability in gray level (it is smoother) than the other two textures. The coarse texture shows up clearly in this measure. As expected, the same

comments hold for R, because it measures essentially the same thing as the standard deviation. The third moment generally is useful for determining the degree of symmetry of histograms and whether they are skewed to the left (negative value) or the right (positive value) [17].

Texture	Mean	Standard deviation	R(normalized)	Third moment	Uniformity	Entropy
Smooth	82.64	11.79	0.002	-0.10	0.026	5.434
Coarse	143.56	74.63	0.079	-0.151	0.005	7.738
Regular	99.72	33.73	0.017	0.750	0.013	6.674

Table 4.1: **Texture measures for Smooth, Coarse and Regular images**

This gives a rough idea of whether the gray levels are biased toward the dark or light side of the mean. In terms of texture, the information derived from the third moment is useful only when variations between measurements are large. Looking at the measure of uniformity, we again conclude that the texture of smooth image is more uniform than the rest and that the most random (lowest uniformity) corresponds to the coarse texture. This is not surprising. Finally the entropy values are in the opposite order and thus lead us to the same conclusions as the uniformity measure did. The smooth texture has the lowest variation in gray level and the coarse texture the most. The regular is in between the two extremes with respect to both these measures.

Measures of texture computed using only histograms suffer from the limitation that they carry no information regarding the relative position of the pixels with respect to each other. One way to bring this type of information into the texture analysis process is to use co-occurrence matrix, which considers not only the distribution of intensities, but also the positions of the pixels with equal or nearly equal intensity value.

4.2.1.1 Co-occurrence Matrix

A 2D co-occurrence matrix A, is an n x n matrix, where n is the number of gray levels within an image. For reasons of computational efficiency, the number of gray levels can be reduced if one chooses to bin them, thus reducing the size of the co-occurrence matrix. The matrix acts as an accumulator so that A[*i , j*] counts the number of pixel pairs having the intensities *i* and *j*. Pixel pairs are defined by a distance and direction which can be represented by a displacement vector d = (*dx, dy*), where *dx* represents the number of pixels moved along the x-axis, and *dy* represents the number of pixels moved along the y-axis of the image slices [18].

1	1	2	2	5
3	2	3	1	1
0	1	1	0	1
3	2	4	0	1
2	1	1	2	2

For example for the above given 5×5 digital image matrix, a co-occurrence matrix is developed as follows (E–W direction only). First, the number of different pixel values is determined. Second, these pixel values are ranked, smallest to largest. Third, the digital image is scanned in the direction noted (E–W in this case) to determine the frequency with which one of these pixel values follows another. With respect to the digital image presented earlier, six different pixel values are observed: 0–5. Hence, the co-occurrence matrix is a 6× 6 matrix (note that, in this case, the co-occurrence matrix is *larger* than the input image); let this matrix be called [A]

[A] =

	0	**1**	**2**	**3**	**4**	**5**
0	0	3	0	0	0	0
1	1	4	2	0	0	0
2	0	1	2	1	1	1
3	0	1	2	0	0	0
4	1	0	0	0	0	0
5	0	0	0	0	0	0

Figure 4.5: co-occurrence matrix

The Haralick co-occurrence texture model and its texture descriptors capture the spatial dependence of gray-level values and texture structures within an image. There are many statistics that can be used; however, due to the redundancy and the high correlation in these statistics, only ten statistics are advocated for feature representation in this application. We are using the following ten descriptors given by Equations (4.10) through (4.19), where P is the normalized co-occurrence matrix, (i, j) is the pair of gray level intensities i and j, and M by N is the size of the co-occurrence matrix [19]:

$$Entropy = -\sum_{i}^{M} \sum_{j}^{N} p[i,j] \log p[i,j] \tag{4.10}$$

$$Energy = \sum_{i}^{M} \sum_{j}^{N} p^2[i,j] \tag{4.11}$$

$$Contrast = \sum_{i}^{M} \sum_{j}^{N} (i-j)^2 \, p\ [i,j] \tag{4.12}$$

$$Homogeneity = \sum_{i}^{M} \sum_{j}^{N} \frac{p\ [i,j]}{1+|i-j|} \tag{4.13}$$

$$SumMean = \frac{1}{2}\sum_{i}^{M}\sum_{j}^{N}(i * p\ [i,j] + j * p\ [i,j] \tag{4.14}$$

$$Variance = \frac{1}{2}\sum_{i}^{M}\sum_{j}^{N}((i-\mu)^2\ p\ [i,j] + (j-\mu)^2\ p\ [i,j] \tag{4.15}$$

$$Maximum\_probability = Max_{i,j}^{M,N} P[i,j] \tag{4.16}$$

$$Inverse\_Difference\_Moment = \sum_{i}^{M}\sum_{j}^{N}\frac{p\ [i,j]}{|i-j|^k} \tag{4.17}$$

$$Cluster\_Tendency = \sum_{i}^{M}\sum_{j}^{N}(i+j-2\mu)^k\ p\ [i,j] \tag{4.18}$$

$$Correlation = \sum_{i}^{M}\sum_{j}^{N}\frac{(i-\mu)(j-\mu)p\ [i,j]}{\sigma^2} \tag{4.19}$$

These descriptors can be calculated at both local (pixel) and global (organ) level depending on the tasks to be used for and the fundamental structures present in the images. Pixel level properties are calculated to be able to isolate regional properties within an image, while global-level features summarize the whole image and represent it as one entity [20].

While co-occurrence matrices are normally defined for a fixed distance and direction. To compute features, the normalized co-occurrence matrices are calculated in four directions (0^0, 45^0, 90^0, and 135^0) and with five displacements (d = 1, 2, 4, 6, 8) generating twenty matrices per segmented image. These rotations and displacements are only in-plane since the images being considered are only 2-dimensional axial slices. For each of the twenty matrices the Haralick features are calculated which can be related to specific characteristics in the image. Then these values are averaged and recorded as a *mean-based feature vector*

for the corresponding segmented image. This vector is further used in the next step for classification of liver lesions.

4.2.1.2 Problems with texture parameters in use

Once the neighborhood size is determined, a co-occurrence matrix is calculated for each neighborhood within the corresponding region. While co-occurrence matrices are normally defined for a fixed distance and direction when calculated at the global level, for the pixel-level approach, we do not calculate the co-occurrence along fixed directions and displacements. Instead we consider all pixel pairs within that neighborhood such that there will be enough samples (pairs) for calculating the co-occurrence matrix in order to produce statistically significant results. Thus, implementation produces a single co-occurrence matrix for each pixel rather than for each choice of distance and direction. Then, for each co-occurrence matrix (each pixel), Haralick features are calculated, which can be related to specific characteristics in the image [21]. Figure 4.4 (b-d) illustrates the image representations of different pixel-level texture features for the original CT image from Figure 4.4 (a).

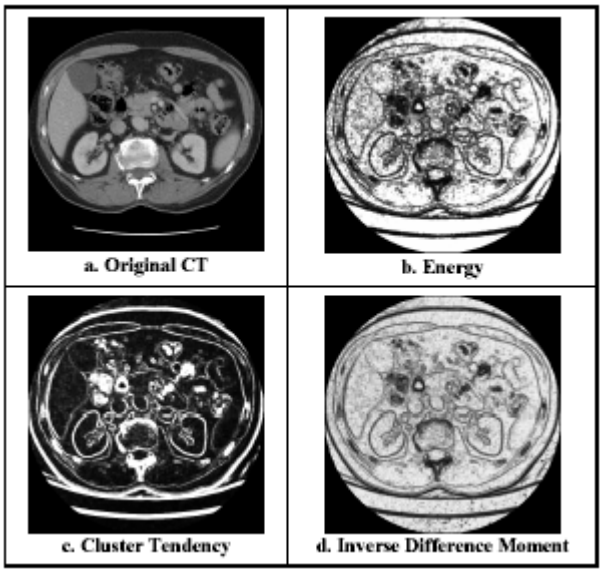


Figure 4.6: Texture features for the original CT image

4.2.2 Structural Approaches

As mentioned at the beginning of this chapter, a second major category of texture description is based on the structural concepts. Suppose that we have a rule of the form S → aS, which indicates that the symbol S may be rewritten as aS (for example, three applications of this rule would yield the string aaaS) [17]. If a represents a circle Figure 4.6(a) and meaning of "circles to the right" is assigned to a string of the form aaa …, the rule S → aS allows generation of the texture pattern shown in Figure 4.6(b).Suppose next that we add some new rules to this scheme: S → bA, A → cA, A → c, A → bS, S → a, where the presence of a b means "circle down" and the presence of a c means "circle to the left." We can now generate a string of the form aaabccbaa that corresponds to a 3 * 3 matrix of circles. Larger texture patterns such as the one shown in figure 4.6(c), can be generated easily in the same way [22].

The basic idea in the forgoing discussion is that a simple "texture primitive" can be used to form more complex texture patterns by means of some rules that limit the number of possible arrangements of the primitive(s).

a

b

c

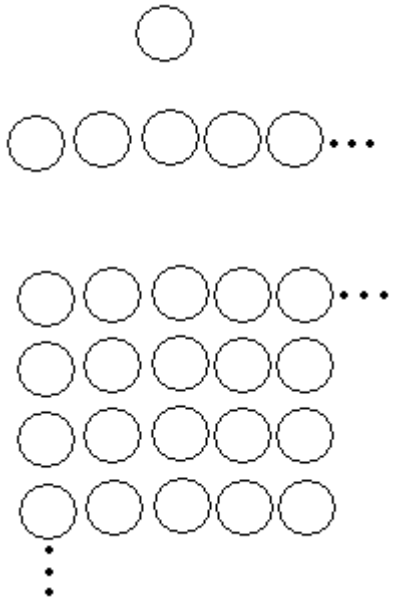


Figure 4.7: (a) Texture primitive. (b) Pattern generated by the rule S→aS.
(c) 2-D texture pattern generated by this and other rules

4.2.3 Spectral Approaches

Fourier spectrum is ideally suited for describing the directionality of periodic or almost periodic 2-D pattern in an image. These global texture patterns, although easily distinguishable as concentrations of high-energy bursts in the spectrum, generally are quite difficult to detect with spatial methods because of the local nature of these techniques. Three features of the Fourier spectrum are useful for texture description [23]:

(1) Prominent peaks in the spectrum give the principle direction of the texture patterns.

(2) The location of the peaks in the frequency plane gives fundamental spatial period of the patterns.

(3) Eliminating any periodic components via filtering leaves nonperiodic image elements, which can then be described by statistical techniques.

As spectrum is symmetric about the origin, so only half of the frequency plane needs to be considered. Thus for the purpose of analysis, every periodic pattern is associated with only one peak in the spectrum, rather than two.

4.3 Summary

The descriptions of the objects or regions that have been segmented out of an image are early steps in the operation of most automated processes. These descriptions for example, constitute the input to the object recognition methods developed in the following chapter. As indicated by the range of the description techniques, the choice of one method is determined by the problem under consideration. Detail of the Haralick co-occurrence matrices is given as it is used in the feature extraction of CAD system.

Chapter No 5

Classification

5.1 Introduction

Artificial intelligence is a part of computer science that tries to make computers more intelligent. One of the basic requirements for any intelligent behavior is learning. Most of the researchers today agree that there is no intelligence without learning. Therefore machine learning is one of the major branches of artificial intelligence. In unsupervised learning one typically tries to uncover hidden regularities (e.g. clusters) or to detect anomalies in the data (for instance some unusual machine function or a network intrusion). In supervised learning, there is a label associated with each example. It is supposed to be the answer to a question about the example. If the label is discrete, then the task is called classification problem otherwise, for real valued labels we speak of a regression problem. Based on these examples (including the labels), one is particularly interested to predict the answer for other cases before they are explicitly observed. Hence, learning is not only a question of remembering but also of generalization to unseen cases.

Classification algorithms have wide range of application in many areas. They are used in medicine for drug trial analysis and MRI data analysis, in finance for share analysis and index prediction, in data communication for signal decoding and error correction, in computer vision for face recognition and pattern recognition applications, in voice recognition, in management for market prediction and uncountable other areas [29].

In order to train classifier data is sometimes available e.g. data about correct diagnoses are often available in the form of medical records in specialized hospitals or their departments. All that has to be done is to input the patient records with known correct diagnoses into a computer program to run a learning algorithm. This is of course an oversimplification, but in principle, the medical diagnostic knowledge can be automatically derived from the description of cases solved in the past. The derived classifier can then be used either to assist the physician when diagnosing new patients order to improve the diagnostic speed, accuracy and/or reliability, or to train students or physicians non-specialists to diagnose patients in a special diagnostic problem [30].

5.2 History

The earliest known system of classification is that of Aristotle, who attempted in the 4th century B.C. to group organisms into two classes i.e. plants and animals. The animal class was further divided into blood and bloodless and was also divided into three sub classes according to their movement i.e. walking, flying and swimming. Carolus Linnaeus, Swedish scientist from 18th century modified the Aristotle system by classifying plants and animals according to similarities in form. He divided living things into two kingdoms i.e. plant kingdom and animal kingdom. Furthermore he divided each of the kingdoms into smaller groups called genera and then divided each general into smaller groups called species [31].

As soon as electronic computers came into use in the 1950s and 1960s, the algorithms were developed that enabled modeling and analyzing large sets of data. From the very beginning, three major branches of machine learning emerged. Classical work in symbolic learning is described by Hunt [30]. Through the years, all three branches developed advanced methods: statistical and pattern recognition methods, such as the k-nearest neighbors, discriminate analysis, and Bayesian classifier, induction learning of symbolic rules, such as top down induction of decision trees, decision rules and induction of logic programs, and artificial neural networks, such as the multilayered feedforward neural network with backpropagation learning, and the Hopfield's associative memory. In the previous chapter, methods for feature calculation have been described. This chapter discusses how the selected features are used for classification. In particular, the theory of machine classifiers applied in this study and the specific modifications made to these classifiers. The machine learning algorithms applied is support vector machine.

5.3 Problem of classification

A classification problem deals with the association of a given input pattern to one of the distinct classes. Patterns are specified by a number of features so it is natural to think of them as d-dimensional vectors, where d is the number of different features. This gives rise to a concept of feature space. Patterns are points in the d-dimensional space and classes are sub-spaces. Classification problem task is to determine which of the regions a given

pattern falls into. If classes do not overlap they are said to be separable and, in principle, one can design a decision rule which will successfully classify any input pattern. A decision rule determines a decision boundary which partitions the feature space into regions associated with each class [32]. It represents our best solution to the classification problem.

Figure 5.1 illustrates a 2-dimensional feature space with three classes occupying regions of the space. The goal is to design a decision rule which is easy to compute and yields the smallest possible probability of misclassification of input patterns from the feature space.

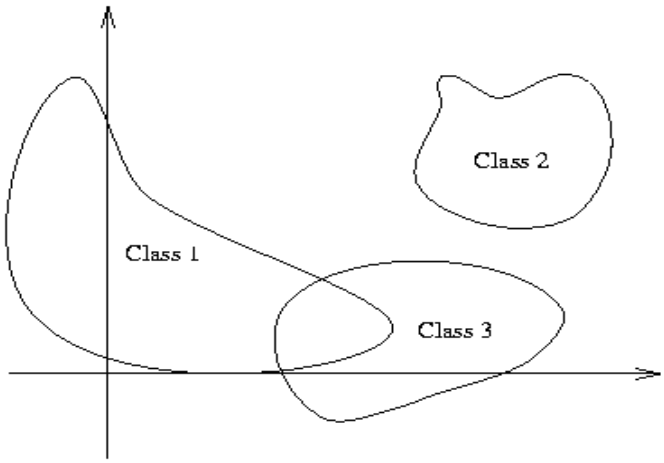


Figure 5.1: Two Dimensional Feature Space with Three Classes

Our information about the classes is usually derived from some finite sample of patterns with known class affiliations. This sample is called a training set. If we make a decision boundary complex enough every pattern in the training set will be properly classified using the underlying decision rule, even if the distributions of patterns overlap. Decision boundary function is a tradeoff between function complexity and error/bias. If we make the decision function complex then we can cover all or most of the samples in the training data, other option is to decrease the complexity and live with errors. Many algorithms have been developed which construct this decision boundary.

Classifiers are designed with a purpose of classifying unknown patterns and it is unlikely that an overly complex decision boundary would provide good generalization as it was

tuned to perform extremely well on the training set. This is known as over-fitting the training set [33]. Figure 5.2 shows a decision boundary over-fitting a training set distributed according to the classes of the Figure 5.1.

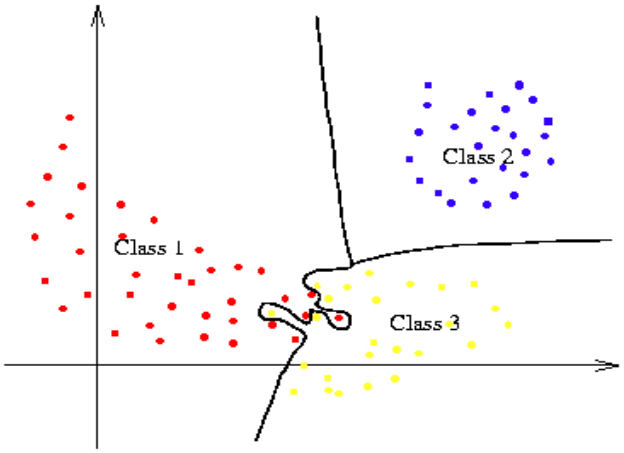


Figure 5.2: Decision boundary between the classes

5.4 Supervised Classification

An important task in Machine Learning is classification, also referred to as pattern recognition, where one attempts to build algorithms capable of automatically constructing methods for distinguishing between different exemplars, based on their differentiating patterns. Watanabe described a pattern as “the opposite of chaos; it is an entity, vaguely defined, that could be given a name.” Examples of patterns are human faces, text documents, handwritten letters or digits, EEG signals, and the DNA sequences that may cause a certain disease. More formally, the goal of a (supervised) classification task is to find a functional mapping between the input data X, describing the input pattern, to a class label Y (e.g. -1 or +1), such that Y = f(X). The construction of the mapping is based on so-called training data supplied to the classification algorithm in the form of $\{(X^1, Y^1), (X^2, Y^2), (X^3, Y^3), \ldots. (X^m, Y^m)\}$. Where X^i is a vector of n values, and Y is the class label. The aim is to accurately predict the correct label on unseen data. A pattern (also: “example”) is described by its features. These are the characteristics of the examples for a given problem.

For instance, in a face recognition task some features could be the color of the eyes or the distance between the eyes. Thus, the input to a pattern recognition task can be viewed as a two dimensional matrix, whose axes are the examples and the features [30].

Pattern classification tasks are often divided into several sub-tasks:

1. Data collection and representation.
2. Feature selection and/or feature reduction.
3. Classification.

Data collection and representation are mostly problem-specific. In broad terms, one should try to find invariant features that describe the differences in classes as best as possible. Feature selection and feature reduction attempt to reduce the dimensionality (i.e. the number of features) for the remaining steps of the task. Finally, the classification phase of the process finds the actual mapping between patterns and labels (or targets). In many applications the second step is not essential or is implicitly performed in the third step [34].

5.5 Classification Algorithms

Although Machine Learning is a relatively young field of research, there exist more learning algorithms. There are six common methods. The first four methods are traditional techniques that have been widely used in the past and work reasonably well when analyzing low dimensional data sets with not too few labeled training examples. Other two methods (Support Vector Machines & Boosting) have received a lot of attention in the machine learning community recently [35]. They are able to solve high-dimensional problems with very few examples (e.g. fifty) quite accurately and also work efficiently when examples are abundant (for instance several hundred thousands of examples).

5.5.1 Traditional Techniques

Nearest Neighbor

Nearest Neighbor classifier is the easiest and one of the effective classifier for any task. The common feature of NN is that it stores all training instances in a memory structure, and uses them directly for classification. The simplest form of memory structure is the multi-dimensional space defined by the attributes in the instance vectors. Each training

instance is represented as a point in that space. The classification procedure includes finding the distances of every point from the testing instance vector. The distances are usually found out using the Euclidean formula. The class to which the majority of nearest neighbors belongs is the predicted class for the testing example. Euclidean distance for a testing vector $\boldsymbol{X = (x_1, x_2, x_3 \ldots\ldots x_N)}$ and training vector $\boldsymbol{X` = (x`_1, x`_2, x`_3 \ldots\ldots x`_N)}$ can be found out as

$$Euclidean_{Distance}(X, X') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + \ldots\ldots + (x_N - x_N')^2}$$

k-Nearest Neighbor Classification

Arguably the simplest method is the k-Nearest Neighbor classifier. Here the k points of the training data closest to the test point are found, and a label is given to the test point by a majority vote between the k points. This method is highly intuitive and attains – given its simplicity – remarkably low classification errors, but it is computationally expensive and requires a large memory to store the training data [36].

Linear Discriminant Analysis

Linear Discriminant Analysis computes a hyperplane in the input space that minimizes the within-class variance and maximizes the between class distance. It can be efficiently computed in the linear case even with large data sets. However, often a linear separation is not sufficient. Nonlinear extensions by using kernels exist however, making it difficult to apply it to problems with large training sets.

Decision Trees

Another intuitive class of classification algorithms is decision trees. These algorithms solve the classification problem by repeatedly partitioning the input space, so as to build a tree whose nodes are as pure as possible (that is, they contain points of a single class). Classification of a new test point is achieved by moving from top to bottom along the branches of the tree, starting from the root node, until a terminal node is reached. Decision trees are simple yet effective classification schemes for small datasets. The computational complexity scales unfavorably with the number of dimensions of the data [37]. Large

datasets tend to result in complicated trees, which in turn require a large memory for storage. The C4.5 implementation by Quinlan is frequently used.

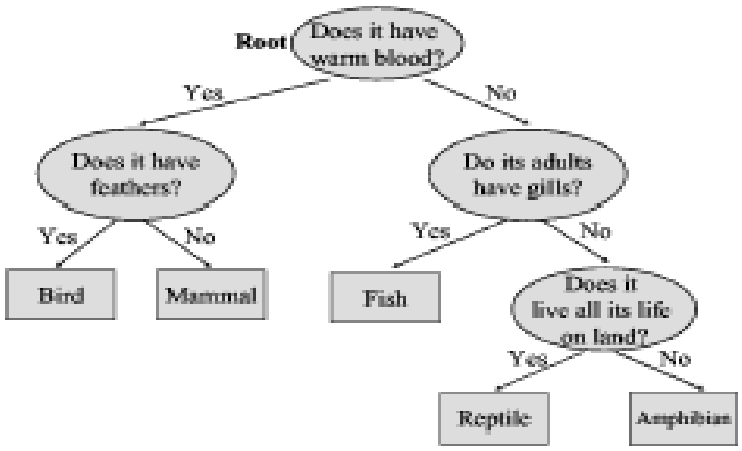


Figure 5.3: An example a decision tree

Neural Networks

Neural Networks are perhaps one of the most commonly used approaches to classification. An artificial neural network is an information processing system that has certain performance characteristic in common with biological neural networks [38]. Artificial neural networks have been developed as generalization of mathematical models of human cognition or neural biology, based on the assumption that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neuron over connection links.
3. Each connection has associated weight, which in a typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A simple scheme for a neural network is shown in Figure 5.4. A neural network is characterized by its pattern of connection between the neurons (called its architecture). Its

method of determining the weights on the connection (called its training, or algorithm, or learning) and its activation function.

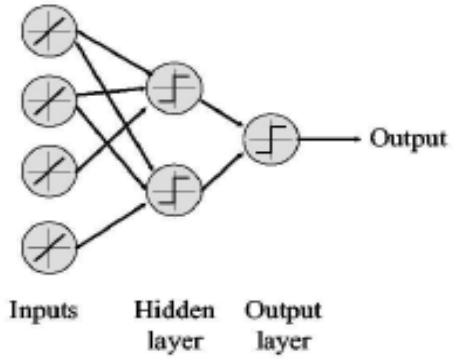


Figure 5.4: A schematic diagram of a neural network. Each circle in the hidden and output layer is a computational element known as a neuron

5.5.2 Large Margin Algorithms

Machine learning rests upon the theoretical foundation of Statistical Learning Theory which provides conditions and guarantees for good generalization of learning algorithms. Within the last decade, large margin classification techniques have emerged as a practical result of the theory of generalization. Roughly speaking, the margin is the distance of the example to the separation boundary and a large margin classifier generates decision boundaries with large margins to almost all training examples. The two most widely studied classes of large margin classifiers are Support Vector Machines (SVMs) and Boosting.

Support Vector Machines

Support Vector Machines work by mapping the training data into a feature space by the aid of a so-called kernel function and then separating the data using a large margin hyperplane. Intuitively, the kernel computes a similarity between two given examples. Most commonly used kernel functions are RBF kernels

$$k(x, x') = \exp\left(\frac{|| x - x' ||^2}{\sigma^2}\right)$$

and polynomial kernels

$$k(x,x') = (x,x')^d$$

The SVM finds a large margin separation between the training examples and previously unseen examples will often be close to the training examples. Hence, the large margin then ensures that these examples are correctly classified as well, i.e., the decision rule generalizes. For so-called positive definite kernels, the optimization problem can be solved efficiently and SVMs have an interpretation as a hyperplane separation in a high dimensional feature space. Support Vector Machines have been used on million dimensional data sets and in other cases with more than a million examples. SVMs are discussed in detail in the next chapter.

Boosting

The basic idea of boosting and ensemble learning algorithms in general is to iteratively combine relatively simple base hypotheses, sometimes called rules of thumb, for the final prediction. One uses a so-called base learner that generates the base hypotheses. In boosting the base hypotheses are linearly combined. In the case of two-class classification, the final prediction is the weighted majority of the votes. The combination of these simple rules can boost the performance drastically. It has been shown that Boosting has strong ties to support vector machines and large margin classification. Boosting techniques have been used on very high dimensional data sets and can quite easily deal with hundred thousands of examples.

5.6 Summary

Material in this chapter is introductory in nature i.e. fundamentals of machine learning classifiers, their types and classification algorithms available under supervised learning like decision trees, neural networks and support vector machine are the main topics.

Chapter no 6

Support vector machine

6.1 Introduction

A classifier is something that takes a feature set as an input and produces a class label. Classifiers are built by taking a set of labeled examples and using them to come up with a rule that assigns a label to any new example. Support vector machines map input vectors to a higher dimensional space where a maximal separating hyperplane is constructed [39]. A **hyperplane** is a concept in geometry. It is a higher-dimensional generalization of the concepts of a line in Euclidean plane geometry and a plane in 3-dimensional Euclidean geometry. The most familiar kinds of hyperplane are linear hyperplanes; less familiar is the projective hyperplane. In a one-dimensional space (a straight line), a hyperplane is a point; it divides a line into two rays. In two-dimensional space (such as the *xy* plane), a hyperplane is a line; it divides the plane into two half-planes. In three-dimensional space, a hyperplane is an ordinary plane; it divides the space into two half-spaces. This concept can also be applied to four-dimensional space and beyond, where the dividing object is simply referred to as a "hyperplane" [40].

Two parallel hyperplanes are constructed on each side of the hyperplane that separates the data. The separating hyperplane is the hyperplane that maximizes the distance between the two parallel hyperplanes. An assumption is made that the larger the margin or distance between these parallel hyperplanes the better the generalisation error of the classifier will be. Many linear classifiers (hyperplanes) separate the data. However, only one achieves maximum separation.

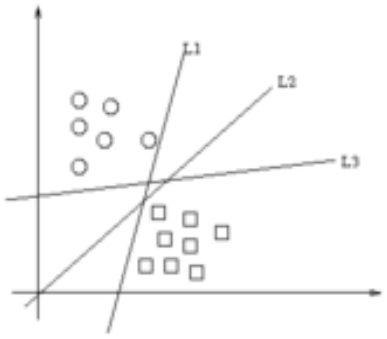


Figure 6.1: Linear hyperplanes

Often we are interested in classifying data as a part of a machine-learning process. In the field of machine learning, the goal of classification is to combine items that have similar feature values, into groups. A **linear classifier** achieves this by making a classification decision based on the value of the linear combination of the features. If the input feature vector to the classifier is a real vector $\vec{x}$, then the output score is

$$y = f(\vec{w}.\vec{x}) = f\left(\sum_j w_j x_j\right) \tag{6.1}$$

Where $\vec{w}$ is a real vector of weights and *f* is a function that converts the dot product of the two vectors into the desired output. The weight vector $\vec{w}$ is learned from a set of labeled training samples. Often *f* is a simple function that maps all values above a certain threshold to the first class and all other values to the second class. A more complex *f* might give the probability that an item belongs to a certain class. For a two-class classification problem, one can visualize the operation of a linear classifier as splitting a high-dimensional input space with a hyperplanes: all points on one side of the hyperplane are classified as "yes", while the others are classified as "no".

A linear classifier is often used in situations where the speed of classification is an issue, since it is often the fastest classifier, especially when $\vec{x}$ is sparse. However, decision trees can be also be faster. Also, linear classifiers often work very well when the number of dimensions in $\vec{x}$ is large, as in document classification, where each element in $\vec{x}$ is typically the number of counts of a word in a document.

In SVM each data point will be represented by a *p*-dimensional vector (a list of *p* numbers). Each of these data points belongs to only one of the two classes. We are interested in whether we can separate them with a "p minus 1" dimensional hyperplanes. This is a typical form of linear classifier. There are many linear classifiers that might satisfy this property. However, we are additionally interested in finding out if we can achieve maximum separation (margin) between the two classes. It means we pick the hyperplane so that the distance from the hyperplane to the nearest data point is maximized.

That is to say that the nearest distance between a point in one *separated* hyperplane and a point in the other *separated* hyperplane is maximized. Now, if such a hyperplane exists, it is clearly of interest and is known as the *maximum-margin hyperplane* and such a linear classifier is known as a **maximum margin** classifier.

6.2 Formalization

A data set that can be successfully split by a linear separator is called linearly separable, for example data points generated by two Gaussian distributions with different means but with same standard deviation.

Let us assume that we are dealing with a two class classification problem where the y is either 1 or −1, (i.e. $\boldsymbol{y} = \{-1, +1\}$) a constant denoting the class to which the point $\mathbf{X_i}$ belongs. Each $\mathbf{X_i}$ is a *p*-dimensional real vector, usually of normalised (normalizing constant) [0,1] or [-1,1] values. The scaling is important to guard against variables (attributes) with larger variance that might otherwise dominate the classification. We can view this as *training data*, which denotes the correct classification which we would like the SVM to eventually distinguish, by means of the dividing (or separating) hyperplane, which takes the form as shown in Fig. 6.2.

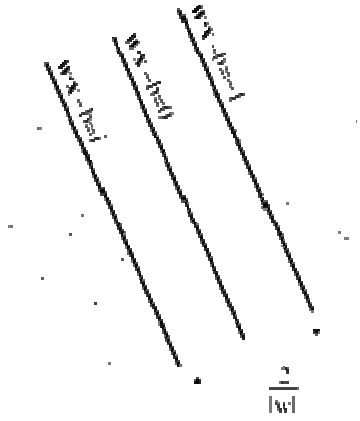


Figure 6.2: Maximum-margin hyperplanes

If the data are separable by ($\boldsymbol{w}$, $\boldsymbol{b}$) then we can multiply both sides of equation by any number without affecting the equality, so they are also separable by any (positive) multiple

of (***w***, ***b***) and hence there exist an infinite number of representations for the same separating hyperplane.

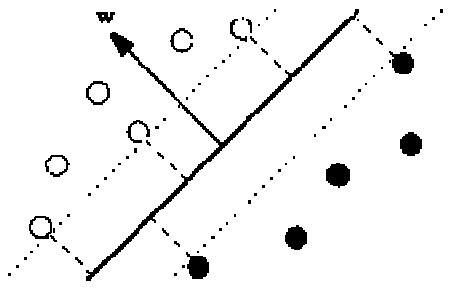


Figure 6.3: **Linear classifier and margins**

A linear classifier is defined by the normal vector ***w*** of a hyperplane and **b** is an offset, and proportional to perpendicular distance from origin to the separator, we consider **b** as a 0^{th} component of the weight vector ***w*** and its data vector **x** will always be equal to 1 i.e. $x_0 = 1$ and $w_0 = b$ i.e. so w.x = 0, the decision boundary is (*solid line*) as shown in figure 6.2

$$\{ \boldsymbol{x} \mid (\boldsymbol{w}^T \boldsymbol{x}) + \mathbf{b} = \mathbf{0} \} \tag{6.2}$$

Each of the two half spaces induced by this hyperplane corresponds to one class, i.e.

$$\boldsymbol{f}(\mathbf{x}) = \mathbf{sgn}((\boldsymbol{w}^T \boldsymbol{x}) + \mathbf{b}) \tag{6.3}$$

The margin of a linear classifier is the minimal distance of any training point to the hyperplane. For the case shown in the Fig. 6.3, it is the distance between the *dotted lines* and the *solid line.* A hyperplane is defined as a function

$$\boldsymbol{f}(\mathbf{x}) = (\boldsymbol{w}^T \boldsymbol{x}) + \mathbf{b} \tag{6.4}$$

Where ***w*** is normalized such that

$$\min_{i=1,\ldots,M} |f(\mathbf{x}_i)| = 1 .$$

Notice that none of the training examples produces an absolute output that is smaller than one and the examples closest the hyperplane have exactly an output of one, i.e.

$$(\boldsymbol{w}^{T}\boldsymbol{x}) + \mathbf{b} = \pm 1$$

In above figures these are the objects (support vectors) which are connected to the decision boundary (dashed lines). Since we assumed the sample to be linearly separable, we can turn any $\boldsymbol{f}$ that separates the data into a hyperplane by suitably normalizing the weight vector $\boldsymbol{w}$ and adjusting the threshold $\boldsymbol{b}$ correspondingly.

The *margin* is defined to be the minimal Euclidean distance between any training example $\boldsymbol{x_i}$ and the separating hyperplane. Intuitively, the margin measures how good the separation between the two classes by a hyperplane is, the margin can be measured by the length of the weight vector $\boldsymbol{w}$. If we subtract the perpendicular distance to the origin we get the distance of x from hyperplane rather than from the origin. The distance measure from hyperplane is signed. It is zero for the points on the hyperplane, positive for points in the side of space towards which the normal vector points and negative for points on the other side as shown in figure 6.2 (we can switch this direction if we take b = -1)

Consider two support vectors $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ from different classes. The margin is given by the projection of the distance between these two points on the direction perpendicular to the hyperplane. We pick the separator which has maximum margin to its closest points on either side. It reduces the variance of hypothesis. Placing the separator very close to positive or negative points is a kind of overfitting and it makes the hypothesis very dependent on input data. So pick margin to the closest positive and negative points be 1. We compute the distance between the hyperplanes by combining $+1(w.x^{+} + b) = 1$ and $+1(w.x^{-} + b) = 1$. Then dividing by length of w gives perpendicular distance i.e. $2/|\boldsymbol{w}|$,

$$\left(\frac{w^{T}}{\|w\|}(x_1 - x_2)\right) = \frac{2}{\|w\|} \qquad (6.5)$$

The smaller the norm of the weight vector $\boldsymbol{w}$, the larger the margin, so we want to minimize $|\boldsymbol{w}|$.

A particularly important insight is that the complexity only indirectly depends on the dimensionality of the data. This is very much in contrast to e.g. density estimation, where the problems become more difficult as the dimensionality of the data increases. For SVM classification, if we can achieve a large margin the problem remains simple.

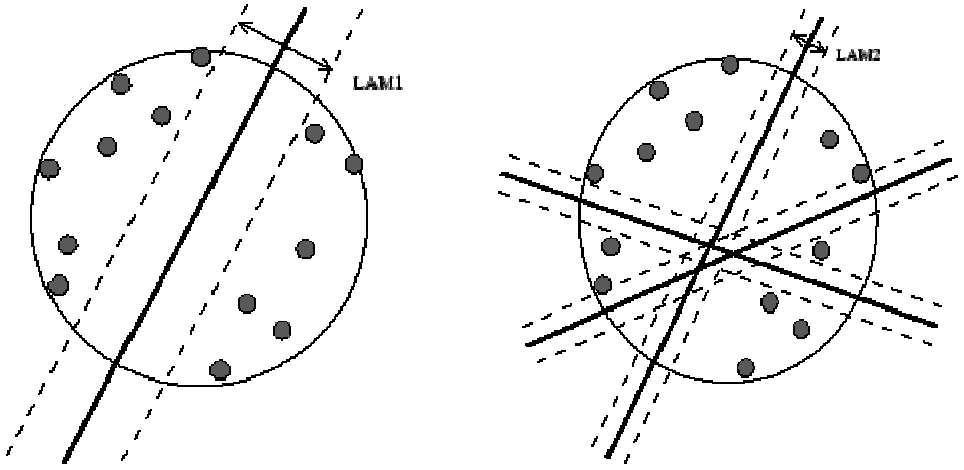


Figure 6.4: Large and small margins

We can see that why a large margin reduces the complexity of a linear hyperplane classifier. If we choose hyperplanes with a large margin, there is only a small number of possibilities to separate the data. On the contrary, if we allow smaller margins there are more separating hyperplanes. Maximum-margin hyperplanes for a SVM trained with samples from two classes. Samples along the hyperplanes are called the support vectors.

$$\boldsymbol{w} \cdot \mathbf{x} - \mathbf{b} = \mathbf{0}$$

The vector $\boldsymbol{w}$ points perpendicular to the separating hyperplane. Adding the offset parameter b allows us to increase the margin. In its absence, the hyperplane is forced to pass through the origin, restricting the solution. As we are interested in the maximum margin, we are interested in the support vectors and the parallel hyperplanes (to the optimal hyperplane) closest to these support vectors in either class.

6.3 Soft Margin

In 1995, Cortes and Vapnik suggested a modified maximum margin idea [39] that allows for mislabeled examples. If there exists no hyperplane that can split the "yes" and "no" examples, the *Soft Margin* method will choose a hyperplane that splits the examples as cleanly as possible, while still maximizing the distance to the nearest cleanly split examples. This work popularized the expression *Support Vector Machine* or *SVM*. The method introduces slack variables, ξ_i, which measure the degree of misclassification of the datum x_i

$$c_i(w.x_i - b) \geq 1 - \xi_i \qquad 1 \leq i \leq n \tag{6.6}$$

The parameters of the maximum-margin hyperplane are derived by solving the optimization. There exist several specialized algorithms for quickly solving the QP problem that arises from SVMs, mostly reliant on heuristics for breaking the problem down into smaller, more-manageable chunks. The objective function is then increased by a function which penalises non-zero ξ_i, and the optimisation becomes a trade off between a large margin, and a small error penalty. If the penalty function is linear, the equation becomes:

$$\min \| w \|^2 + C\sum_i \xi_i \quad \text{such that} \quad c_i(w.x_i - b) \geq 1 - \xi_i \qquad 1 \leq i \leq n \tag{6.7}$$

We are now at the point to merge the ideas into a single algorithm, Support Vector Machines, suitable for a wide range of practical application. The main goal of this algorithm is to find a weight vector $\boldsymbol{w}$ separating the data with the largest possible margin. Assume that the data are separable. Our goal is to find the smallest possible $\boldsymbol{w}$ without committing any error. This can be expressed by the following quadratic optimization problem:

$$\min_{w,b} \frac{1}{2} \| w \|^2$$

$$\text{subject to} \quad y_i\,((\boldsymbol{w}^{\mathbf{T}}\boldsymbol{x_i}) + \mathbf{b} \geq \mathbf{1}, \qquad \forall_i = 1, ..., M \tag{6.8}$$

The constraints in (6.10) assure that ***w*** and ***b*** are chosen such that no example has a distance to the hyperplane smaller than one. The problem can be solved directly by using a quadratic optimizer. Notice that the optimal solution renders a hyperplane. In contrast to many neural networks one can always find the *global* minimum. In fact, all minima of (6.10) are global minima, although they might not be unique as e.g. in the case when M< N, where N is the dimensionality of the data.

In the formulation (6.10), referred to as the primal formulation, we are bound to use the original data **x**. These constraints (in 6.10) along with the objective of minimizing $|\boldsymbol{w}|$ can be solved by incorporating the constrained as an additional term and using Lagrange multipliers [41]. We obtain the following Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{M} \alpha_i (y_i((w^T x_i) + b) - 1) \tag{6.9}$$

The task is to minimize (6.9) with respect to ***w, b*** and to maximize it with respect to α_i. At the optimal point, we consider alphas as constant for now and figure out what values of w and b would optimize L for those fixed alphas, we can solve this by taking partial derivatives of L with respect to w and b and setting them to zero, getting two constraints. We have the following equations:

$$\frac{\partial L}{\partial b} = 0 \qquad \text{and} \qquad \frac{\partial L}{\partial w} = 0$$

We get two more constraints

$$\sum_{i=1}^{M} \alpha_i y_i = 0 \qquad \text{and} \qquad w = \sum_{i=1}^{M} \alpha_i y_i x_i \tag{6.10}$$

Here w is a weighted sum of the input points, by substituting this expression in L, we get L as a function of alphas. Now we have an expression involving only alphas and x's and y,s. This function is known as dual langrangian.

From the right equation of (6.12), we find that $\boldsymbol{w}$ is contained in the subspace spanned by the $\boldsymbol{x_i}$ in the training set. By substituting (6.12) into (6.11), we get the dual quadratic optimization problem:

$$\max_{\alpha} \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \tag{6.11}$$

$$\text{subject to} \quad \alpha_i \geq 0 \,, \; i = 1,\ldots..,\text{M} \tag{6.12}$$

$$\sum_{i=1}^{M} \alpha_i y_i = 0 \tag{6.13}$$

In solution most of the alphas will be zero, corresponding to data points that do not provide binding constraints on the choice of the weights. A few of the data points will have their alphas be nonzero, they will satisfy their constraints with equality (that is, their margin is equal to 1). These are called support vectors and they are the ones used to define the maximum margin separator. We can remove all the other data points and still get the same separator. This learning method is called a support vector machine, or SVM. Thus, by solving the dual optimization problem, one obtains the coefficients $\boldsymbol{\alpha_i}$, $\boldsymbol{i}$ = 1,…., M, which one needs to express the solution $\boldsymbol{w}$. This leads to the decision function:

$$f(\text{x}) = \text{sgn}((w^{\text{T}} x) + \text{b})$$

$$= \text{sgn}\left(\sum_{i=1}^{M} y_i \alpha_i (x_i^T x) + b \right) \tag{6.14}$$

Learning depends only on dot products of sample pair. Recognition depends only on dot products of unknown with samples. It means given an unknown vector u, predict class (1 or -1) as follows:

$$h(u) = \text{sgn}\left(\sum_{i=1}^{M} \alpha_i y_i x^i . u + b \right)$$

The sum is over M support vectors, the classifier depends only on the support vectors, not on all training points. The maximum margin constraint helps reduce the variance of the SVM hypotheses, minimum magnitude weight vector drasitically cuts down on the size of the hypothesis class and helps avoid overfitting. SVM traing process gaurantees a unique global maximum.

When Straight Lines Go Crooked

The simplest way to divide two groups is with a straight line, flat plane or an N-dimensional hyperplane. But if the points are separated by a nonlinear region such as shown below. In this case we need a nonlinear dividing line as shown in the figure below.

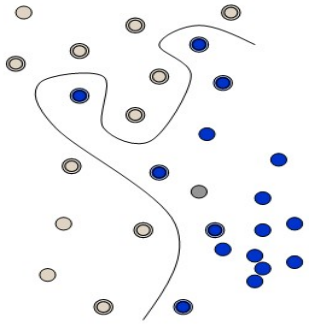


Figure 6.5: Nonlinear dividing line

6.4 Non-linear SVM

The original optimal hyperplane algorithm proposed by Vladimir Vapnik in 1963 was a linear classifier. However, in 1992, Bernhard Boser, Isabelle Guyon and Vapnik suggested a way to create non-linear classifiers by applying the kernel trick (originally proposed by Aizerman) to maximum-margin hyperplanes [42]. Its an important advantage of the SVM that it is not necessary to implement this transformation and to determine the separating hyperplane in the possibly very-high dimensional feature space, instead a kernel representation can be used, where the solution is written as a weighted sum of the values of certain kernel function evaluated at the support vectors. The resulting algorithm is formally

similar, except that every dot product is replaced by a non-linear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in the transformed feature space. The transformation may be non-linear and the transformed space is high dimensional; thus the classifier is a hyperplane in the high-dimensional feature space it may be non-linear in the original input space.

If the kernel used is a Gaussian radial basis function, the corresponding feature space is a Hilbert space of infinite dimension. Maximum margin classifiers are well regularized, so the infinite dimension does not spoil the results.

6.4.1 The Kernel Trick

Rather than fitting nonlinear curves to the data, SVM handles this by using a *kernel function* to map the data into a different space where a hyperplane can be used to do the separation.

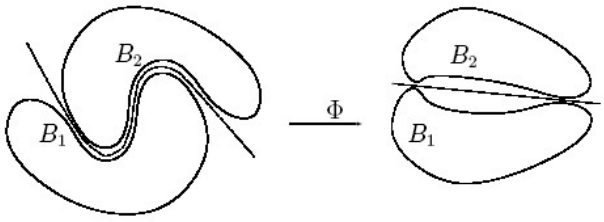


Figure 6.6: Data in different space

The basic idea of the so called *kernel-methods* is to first preprocess the data by some non-linear mapping Φ and then to apply the same linear algorithm as before but in the image space of Φ as shown in Fig. 6.7. More formally we apply the mapping as

$$\Phi : R^N \to \varepsilon \text{ , } \quad x \to \Phi(x)$$

to the data $x_1, x_2, \ldots, x_M \in \mathbf{X}$, and consider our algorithm in ε instead of **X**, i.e. the sample is preprocessed as

$$\{(\Phi(x_1), y_1, \ldots, \Phi(x_M), y_1)\} \subseteq (\varepsilon \times y)^M$$

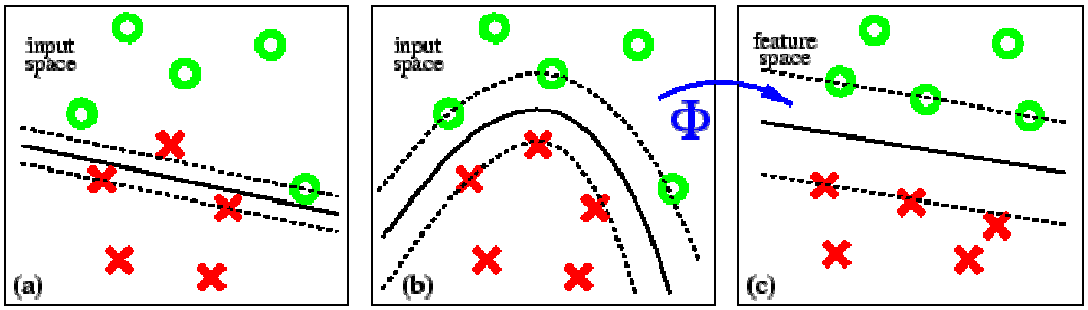


Figure 6.7: Three different views on the same two class separation problem

(a) A linear separation of the input points is not possible without errors. Even allowing misclassification of one data point results in a small margin. (b) A better separation is provided by a non-linear surface in the input space. (c) This non-linear surface corresponds to a linear surface in a feature space. Data points are mapped from input space to feature space by the function Φ induced by the kernel function ***k***. The kernel function may transform the data into a higher dimensional space to make it possible to perform the separation. Separation may be easier in higher dimensions.

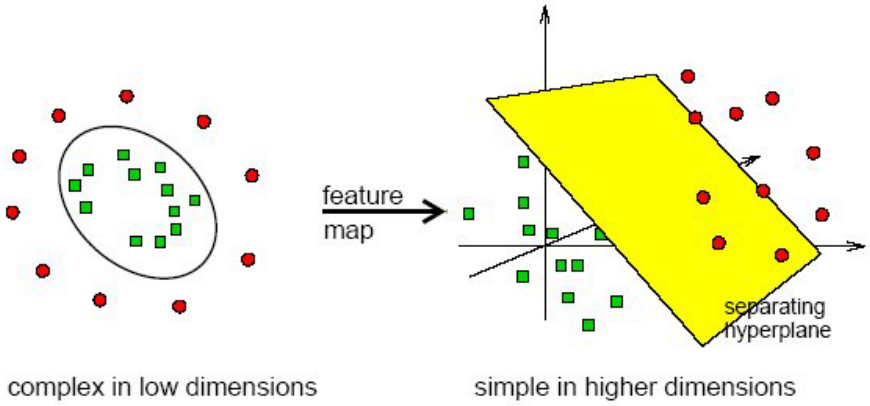


Figure 6.8: **Separation in low and higher domain**

In certain applications we might have sufficient knowledge about our problem such that we can design an appropriate Φ by hand. If this mapping is not too complex to compute and the space ε is not too high-dimensional, we might just explicitly apply this mapping to our data. Something similar is done for (single hidden layer) neural networks, radial basis

networks or Boosting algorithms, where the input data are mapped to some representation given by the hidden layer, the RBF bumps or the hypotheses space, respectively. The difference with kernel-methods is that for a suitably chosen Φ we get an algorithm that has powerful non-linearities but is still very intuitive and retains most of the favorable properties of its linear input space version.

The problem with explicitly using the mapping Φ to construct a feature space is that the resulting space can be extremely high-dimensional. As an example consider the case when the input space *X* consists of images of 16 * 16 pixels, i.e. 256 dimensional vectors, and we choose 5th order monomials as non-linear features. The dimensionality of such space would be

$$\binom{5+256-1}{5} \approx 10^{10}$$

Such a mapping would clearly be intractable to carry out explicitly. We are not only facing the technical problem of storing the data and doing the computations, but we are also introducing problems due to the fact that we are now working in an extremely sparse sampled space. The problems concerning the storage and the manipulation of the high dimensional data can be avoided. It turns out that for a certain class of mappings we are well able to compute scalar products in this new space even if it is extremely high dimensional. Simplifying the above example of computing all 5^{th} order products of 256 pixels to that of computing all 2^{nd} order products of two "pixels", i.e.

$$x = (x_1, x_2) \qquad \text{and} \qquad \phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

the computation of a scalar product between two such feature space vectors can be readily reformulated in terms of a kernel function *k*:

$$(\phi(x)^T \phi(z))$$

$$= ((x_1, x_2)(z_1, z_2)^T)^2$$

$$= (x^T z)^2$$

$$=: k(x, z)$$

This finding generalizes: For $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N$, and $d \in N$ the kernel function, computes a scalar product in the space of all products of d vector entries of x and z.

$$k(x, z) = (x^T z)^d$$

The *kernel trick* is to take the original algorithm and formulate it such, that we only use $\Phi(x)$ in scalar products. Then, if we can efficiently evaluate these scalar products, we do not need to carry out the mapping Φ explicitly and can still solve the problem in the huge feature space ε. Furthermore, we do not need to know the mapping Φ but only the kernel function.

Algorithm: The Support Vector Machine with regularization parameter C.
Given labeled sequences $(x_1, y_1), \ldots, (x_m, y_m)$ ($x \in X$ and $y \in \{-1, +1\}$) and a kernel k, the SVM computes a function

$$f(s) = \sum_{i=1}^{m} \alpha_i k(x_i, x) + b$$

where the coefficients α_i and b are found by solving the optimization problem

minimize $\sum_{i,j=1}^{m} \alpha_i \alpha_j k(s_i, s_j) + C \sum_{i=1}^{m} \varepsilon_i$ subject to $y_i f(x_i) \geq 1 - \varepsilon_i$

6.5 Standard choices for kernels

Now we need to find some mappings from low to high dimensional space, which have convenient kernel function associated with them. The simplest case is one where phi is the identity function and K is just the dot product i.e. linear kernel given in table 6.1. One such other kernel function called polynomial kernel given in table 6.1. It is the dot product raised to a power; the actual power is a parameter of the learning algorithm that determines

the properties of the solution. Another popular kernel function is Radial basis kernel, an exponential of the square of the distance between vectors, divided by sigma squared given in the table below. This is the formula for a Gaussian bump in the feature space, where sigma is the standard deviation of the Gaussian. Sigma is a parameter of the learning that determines the properties of the solution. Table 6.1 lists some of the most widely used basic kernel functions.

Linear kernel	$\Phi(x^i).\Phi(x^k) = K(x^i, x^k) = x^i . x^k$
Polynomial kernel (n^{th} order)	$K(x^i, x^k) = (1+ x^i .x^k)^n$
Radial basis kernel	$k(x^i, x^k) = e^{\frac{-\| x^i, x^k \|^2}{2\sigma^2}}$

Table 6.1: Common kernel functions

6.5.1 Properties of Kernels

Besides being useful tools for the computation of dot products in high or infinite dimensional spaces, kernels possess some additional properties that make them an interesting choice in algorithms. Using a particular positive definite kernel corresponds to an *implicit* choice of a regularization operator. For translation-invariant kernels, the regularization properties can be expressed conveniently in Fourier space in terms of the frequencies. For example, Gaussian kernels correspond to a general smoothness assumption in all k-th order derivatives. Vice versa, using this correspondence, kernels matching a certain prior about the frequency content of the data can be constructed so as to reflect our prior problem knowledge [45].

Another particularly useful feature of kernel functions is that we are not restricted to kernels that operate on vectorial data. In principle, it is also possible to define positive kernels for e.g. strings or graphs, i.e. making it possible to embed discrete objects into a metric space and apply metric-based algorithms [43]. Furthermore, many algorithms can be formulated using so called *conditionally positive definite* kernels, which are a superclass of the positive definite kernels considered so far. They can be interpreted as generalized non-linear dissimilarity measures (opposed to just the scalar product) and are applicable e.g. in SVM and kernel PCA.

6.5.2 Radial Basis Function(RBF) as kernel

Normally a Gaussian is used as the RBF, Figure 6.9 shows a two-dimensional version of such a kernel. From eqn 6.20, the output of the kernel is dependent on the Euclidean distance of x_j from x_i (one of these will be the support vector and the other will be the testing data point). The support vector will be the centre of the RBF and σ will determine the area of influence this support vector has over the data space [44].

$$k(x^i, x^k) = e^{\frac{-\| x^i, x^k \|^2}{2\sigma^2}} \tag{6.20}$$

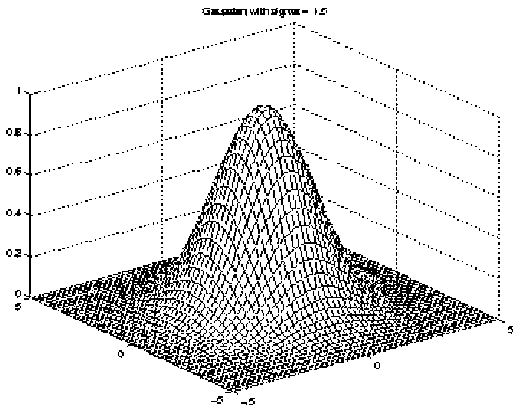


Figure 6.9: The Radius Basis Function kernel

A larger value of σ will give a smoother decision surface and more regular decision boundary. This is because an RBF with large σ will allow a support vector to have a strong influence over a larger area. Figures 6.10 and 6.11 show the decision surface and boundaries for two different σ values. A larger σ value also increases the α value (the Lagrange multiplier) for the classifier. When one support vector influences a larger area, all other support vectors in the area will increase in α-value to counter this influence. Hence all α-values will reach a balance at a larger magnitude. A larger σ-value will also reduce the number of support vectors. Since each support vector can cover a larger space, fewer are needed to define a boundary.

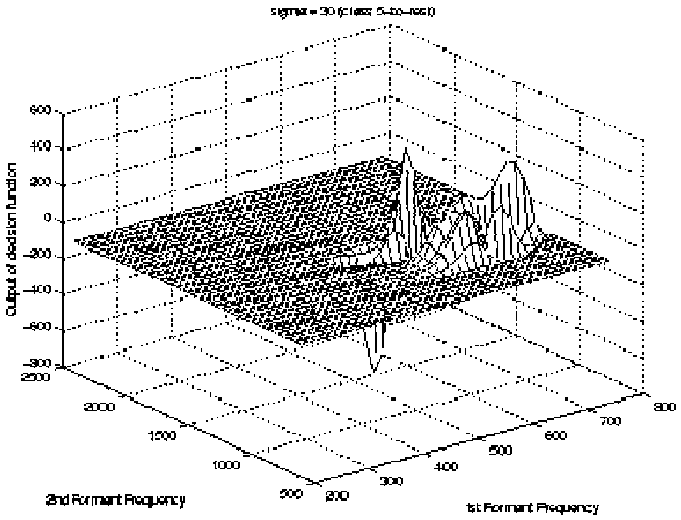


Figure 6.10: Decision surface of small σ

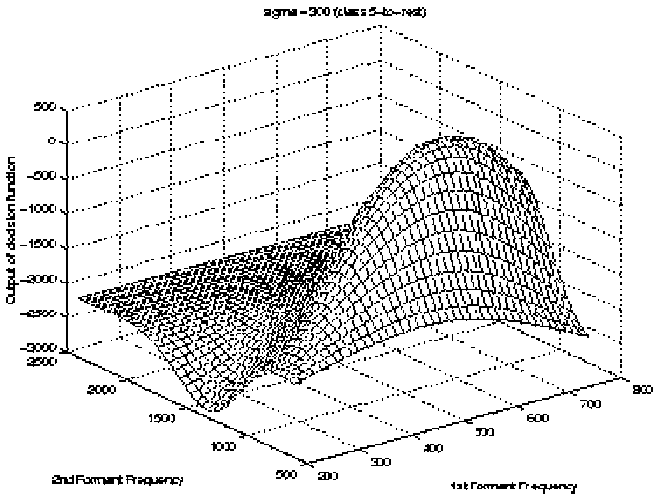


Figure 6.11: Decision surface of large σ

This means that the estimate of ||*w*|| will increase. The estimation of the VC dimension of the SVM depends on the norm of *w* and also on the radius of the sphere that encompasses all the data as σ increases, the value of *R* will decrease. This will balance the increase in ||*w*||. The experiments on the effect of different σ-values show that the expected risk loosely corresponds to the accuracy of the classifier tested on testing data.

6.6 Cross Validation error

We can estimate the error on new data by computing the cross-validation error on the training data. If we look at the linearly separable case, it is easy to see that the expected value of leave-one-out cross-validation error is bounded by the proportion of support vectors. If we take a data point that is not a support vector from the training set, the computation of the separator will not be affected and so it will be classified correctly. If we take a support vector out, then the classifier will in general change and there may be an error. So, the expected generalization error depends on the number of support vectors and not on the dimension.

Note that using a radial basis kernel with very small sigma gives us a high expected number of support vectors and therefore a high expected cross-validation error, as expected. Yet, a radial basis kernel with large sigma, although of similar dimensionality,

has fewer expected support vectors and is likely to generalize better. We shouldn't take this bound too seriously; it is not actually very predictive of generalization performance in practice but it does point out an important property of SVMs, that generalization performance is more related to expected number of support vectors than to dimensionality of the transformed feature space.

6.7 Summary

One key point is that SVMs have a training method that guarantees a unique global optimum. This eliminates many problems in other approaches to machine learning. The other advantage of SVMs is that there are relatively few parameters to be chosen: the constant used to trade off classification error and width of the margin; and the kernel parameter, such as sigma in the radial basis kernel. And, last is the kernel trick. That is, the whole process depends only on the dot products of the feature vectors, which is the key to the generalization to non-linear classifiers.

Chapter No 7

Backpropagation Neural Net

7.1 Introduction

The demonstration of the limitations of neural network was a significant factor in the decline of interest in neural network in 1970's.The discovery and widespread dissemination of an effective general method of training a multi-layer neural network play a major role in the emergence of neural network as a tool for solving a wide variety of problems.

Backpropagation or generalized delta rule is simply a *gradient descent* method to *minimize the total squared error* of the output computed by the net.

The very general nature of the backpropagation training method means that a backpropagation net (multilayer, feedforward net trained by the backpropogation) can be used to solve many problems in many areas.

A back-propagation neural network is only practical in certain situations. Here are some situations where a BP NN might be a good idea:

A large amount of input/output data is available, but you're not sure how to relate it to the output.

The problem appears to have overwhelming complexity, but there is clearly a solution.

It is easy to create a number of examples of the correct behavior.

The solution to the problem may change over time, within the bounds of the given input and output parameters (i.e., today 2+2=4, but in the future we may find that 2+2=3.8).

Outputs can be "fuzzy", or non-numeric.

One of the most common applications of NNs is in image processing. Some examples would be: classifying the diseases in a CAD system, identifying hand-written characters; matching a photograph of a person's face with a different photo in a database; performing data compression on an image with minimal loss of content. Other applications could be: voice recognition; stock market prediction. All of these problems involve large amounts of data, and complex relationships between the different parameters.

It is important to remember that with a NN solution, you do not have to understand the solution at all! This is a major advantage of NN approaches. With more traditional techniques, you must understand the inputs, and the algorithms, and the outputs in great detail, to have any hope of implementing something that works. With a NN, you simply show it: "this is the correct output, given this input". With an adequate amount of training, the network will mimic the function that you are demonstrating. Further, with a NN, it is OK to apply some inputs that turn out to be irrelevant to the solution - during the training process; the network will learn to ignore any inputs that don't contribute to the output. Conversely, if you leave out some critical inputs, then you will find out because the network will fail to converge on a solution.

If your goal is stock market prediction, you don't need to know anything about economics, you only need to acquire the input and output data. There are several ways to do so for neural recognizers.

7.1.1 Architecture

A multilayer network with one layer of hidden units (the Z unit) is shown in figure. The output units (the y units) and the hidden units also may have biases (as shown).The bias on a typical output unit Y_k is denoted by w_{0k}: the bias on a typical hidden unit Z_j is denoted by v_{oj}.

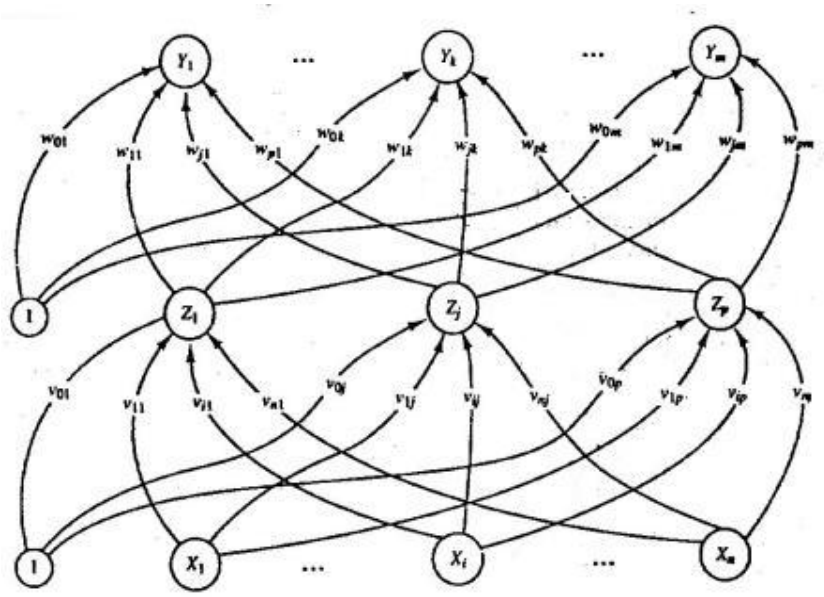


Figure 7.1: Backpropagation neural network with one hidden layer

The bias terms act like weights on connection from units whose output is always 1.Only the direction of information flow for the feed forward phase of operation is shown. During the back propagation phase of learning, signals are sent in the reverse direction.

Following is the algorithm for one hidden layer which is adequate for large number of applications especially for handwritten character recognition.

7.1.2 Algorithm

Training a network by back propagation involves three stages:

- The feedforward of the input training pattern
- The backpropagation of the associated error
- The adjustment of weights.

During feed forward, each input unit (X_i) receives an input signal and broadcasts this signal to each of the hidden units Z_1, Z_2, …… Z_p.

Each hidden unit then computes its activation and sends its signal (z_j) to each output unit. Each output unit (Y_k) computes its activation (y_k) to form the response of the net for the given input unit.

During training, each output unit compares its computed activation y_k with its target value t_k to determine the associate error for that pattern with that unit. Based on this error, the factor δ_k (k=1…….m) is computed, δ_k is used to distribute the error at the output Y_k back to all units in the previous layer (the hidden units that are connected to Y_k).It is also used (later) to update the weights between the output and the hidden layer. In a similar manner the factor δ_j (j=1…….p) is computed for each hidden unit Z_j. It is not necessary to propagate the error back to the input layer, but δ_j is used to update the weights between the hidden layer & the input layer.

After all of the δ factors have been determined, the weights for all layers are adjusted simultaneously. The adjustment to the weight w_{jk} (from hidden unit Z_j to output unit Y_k) is based on the factor δ_k and the activation z_j of the hidden unit Z_j. The adjustment to the weight vij (from input unit X_i to hidden unit Z_j) is based on the factor δ_j and the activation v_{ij} of the input unit.

7.1.3 Nomenclature

The nomenclature, we use in the training algorithms for the backpropogation net is as follows.

x Input training vector

x = (x_1, x_2,……x_i…….x_n)

t Output target vector

t = (t_1, t2,……..t_k…….t_m)

δ_k Portion of the correction weight adjustment for wij that is due to an error that output unit Y_k, also the information about the error at unit Y_k that is propagated back to the hidden unit that feed into unit Y_k.

δ_j Portion of the correction weight adjustment for vij that is due to backpropagation of information from the output layer to the hidden Zj.

α Learning rate

Xi Input unit i

For an input unit, the input signal and output signal are the same, namely xi

$\mathbf{v_{oj}}$ Bias on hidden unit j

$\mathbf{Z_j}$ Hidden unit j

The net output to the Z_j is denoted by z_in_j

$z\_in_j = v_{ok} + \Sigma x_i v_{ij}$

The output signal (activation) of Z_j is denoted z_j

$z_j = f(z\_in_j)$

$\mathbf{w_{ok}}$ Bias on output unit k

$\mathbf{Y_k}$ Output unit k

The net output to the Y_k is denoted by y_in_k

$y\_in_k = w_{ok} + \Sigma z_i w_{ik}$

The output signal (activation) of Y_k is denoted y_k

$y_k = f(y\_in_k)$

7.1.4 Activation Function

The activation function for a backpropagation net should have several important characteristics. It should be continuous, differentiable and monotonically non- decreasing. Furthermore for computational efficiency, it is desirable that its derivative be easy to compute. For the most commonly used activation functions the value of the derivative (at a particular value of the independent variable) can be expressed in terms of the value of the function(at the value of independent variable),the function is expected to saturate i-e approach finite maximum and minimum values asymptotically.

One of the most typical activation functions is the **binary sigmoid** function, which has range **(0, 1)** and is defined as

$f_1(x) = 1/1+\exp(-x)$

with

$f'_1(x) = f_1(x)[1-f'_1(x)]$

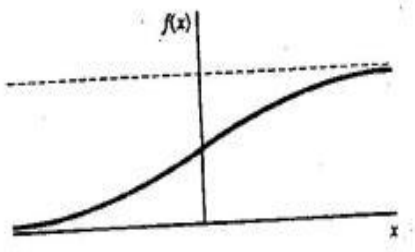


Figure 7.2: Binary sigmoid

Another common activation function is **bipolar sigmoid,** which has range of **(-1, 1**) and is defined as

$f_2(x) = [2/1+\exp(-x)]-1$

with

$f'_2(x) = 1/2[1+f2(x)][1-f2(x)]$

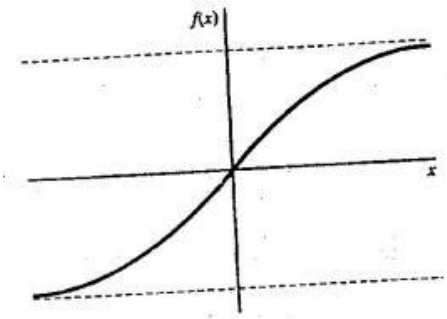


Figure 7.3: Bipolar sigmoid

Note that bipolar sigmoid function is closely related to the function

$\text{Tanh}(x) = e^x - e^{-x}/e^x + e^{-x}$

The sigmoid function explained above can be translated to the right or left by the use of an additive constant on the independent variable. However this is not necessary, since the trainable bias serves the same role.

The *steepness of the logistic sigmoid* can be modified by a slope parameter σ. This more general sigmoid function (with range between (0 and 1) is

$f(x)=1/1+\exp(-\sigma x)$

with

$f'(x)=\sigma f(x)[1-f'(x)]$

7.1.5 Training algorithm

Either of the activation function can be used in the standard back propagation algorithm. The form of data (especially the target values) is an important factor in choosing appropriate function. Note that because of the simple relationship between the value of the function and its derivative, no additional evaluations of the exponential are required to compute the derivatives needed during the backproagation phase of algorithm.

The algorithm is as follows

Step 0 Initialize weights
(Set to small random values)

Step1 While stopping condition is false, do step 2-9

Step2 For each training pairs do Steps 3-8

Feedforward

Step 3 Each input unit (Xi, i=1, 2……n) receives input signal xi, and broadcasts this signal to all units in the layer above the hidden units)

Step4 Each hidden unit (Zj, j=1, 2……..p) sums its weighted input signals

$z\_in_i = voj + \Sigma xivij$

applies its activation function to compute its output signals

$zj = f(z\_in_j)$

and sends the signal to all units in the layer above (output units).

Step5 Each output unit (Y_k, k=1, 2……..p) sums its weighted input signals

$y\_ink = w_{ok} + \Sigma z_i w_{ik}$

and applies its activation function to compute its output signals

$y_k = f(y\_in_k)$

Backpropogation of error:

Step6: Each output unit (Y_K, k=1…..m) receives a target pattern corresponding to the input training pattern, computes its error information term

$\delta_k = (t_k - y_k)\, f'(y\_in_k)$

calculates its weight correction term (used to update wjk later)

$\Delta w_{jk} = \alpha\, \delta_k\, z_j$

Calculates its bias correction term (used to update wok later)

$\Delta w_{ok} = \alpha\delta_k$

And sends δk to units in the layer below.

Step7: Each hidden unit (Z_j, j=1…..p) sums delta inputs (from units in the layer above)

$\delta\_in_k = \Sigma\, \delta_k\, w_{jk}$

multiplies by the derivative of its activation function to calculate its error information term

$\delta_j = \delta\_in_j\, f'(z_j)$

and calculates its weight correction term(used to update voj later)

$\Delta v_{ij} = \alpha\, \delta j\, xi$

And calculates its bias correction term used to update v_{oj}, later

$\Delta v_{oj} = \alpha\delta_j$

Update weights and biases

Step8 Each output unit (Y_k,k=1…….m)updates its bias and weights (j=1……p)

w_{jk} (new)=w_{jk}(old)+ Δw_{jk}

Each hidden unit (Zj:j=1……p) updates its bias and weights (i=0,……n)

v_{ij}(new)=v_{ij}(old)+ Δv_{ij}

Step9 Test stopping condition

In implementing this algorithm, separate arrays will be used for the deltas for the output units(step6,δk) and the deltas for the hidden units(step7,δj)

An epoch is one cycle through the entire set of training vectors. Typically many epochs required for training backpropagation neural net.

The mathematical bias for the backpropagation algorithm is the optimization technique known as gradient descent. The gradient of the function (in this case the function is the error and the variables are the weights of the net) gives the direction in which the function increases more rapidly; the negative gradient gives the direction in which the function decreases more rapidly.

7.2 Random Initialization of weights

The choice of initial weights influence whether the net reaches a global or only local minimum of the error & if so, how quickly it converges. The update of the weight between two units depends on both the derivative of the upper unit's activation function & the activation of lower unit. For this reason it is important to avoid the choices of initial weights that would make it likely that either activations or derivative of activation are zero. The values for the initial weights must not be too large, or the initial input signals to each hidden or output unit will be likely fall in the region where the derivative of sigmoid function has a very small value (the so-called saturation region).On the other hand, if the initial weights are too small, the net input to a hidden or output unit will be close to zero, which also causes extremely slow learning.

A common procedure to initialize the weights (and biases) to random values between -0.5 and 0.5(or between -1 and +1 or some other suitable interval).The values may be positive or negative because the final weights after training may be of either sign also.

Nguyen-Widrow Initialization

This approach is based on a geometrical analysis of the response of the hidden neurons to single input. Weights from the hidden units to the output input are initialized to random values between 0.5 and 0.5(or between -1 and +1) , as commonly the case .

The initialization of the weights from the input unit to the hidden units is designed to improve the ability of the hidden units to learn. This is accomplished by

distributing the initial weights, for each input pattern, it is likely that the net input to one o the hidden units will be in the range in which that hidden neuron will learn them most readily.

n number of input unit

p number of hidden unit

β scale factor

$\beta = 0.7(p)^{1/n}$

Initialize its weight vectors (from the input unit)

Compute $\| v_j(old) \| = (v_{1j}(old)^2 + v_{2j}(old)^2 + ... + v_{nj}(old)^2)^{½}$

Reinitialize weights: $v_{ij} = \beta v_{ij}(old) \ / \ \| v_{ij}(old) \|$

7.3 Training period

Since the usual motivation for applying backpropogation net is to achieve a balance between correct response to new input patterns (i.e. the balance between memorization or generalization), it is not necessarily advantageous to continue training until the total squared error actually reaches the minimum. Hecht-Neilson (1990) suggests using two sets of data during training patterns and a set of training-testing patterns. These two sets are disjoint .Weight adjustments are based on the training pattern; however at intervals during training the error is computed using training-testing patterns. As long as training the error begins to decrease training continues .When the error begins to increase, the net is starting to memorize the training patterns too specifically (and starting to lose its ability to generalize).At this point training is terminated.

7.4 Number of training pairs

A relationship among the number of training patterns available P, the number of weights to be trained W, and the accuracy of classification expected e, is summarized in the following rules of thumb

$$W/P = e$$

$$P = W/e$$

Where

P = No of training patterns

W = Number of weights to be trained

E = Accuracy of classification

7.5 Number of hidden layers

Although a single hidden layer is sufficient to solve any function approximation problem, some problems may be easier to solve using net with two hidden layers. For handwritten character recognition, two hidden layer will be used. You may not need any hidden layers at all. Linear and generalized linear models are useful in a wide variety of applications (McCullagh and Nelder 1989). And even if the function you want to learn is mildly nonlinear, you may get better generalization with a simple linear model than with a complicated nonlinear model if there is too little data or too much noise to estimate the non-linearities accurately. If we have only one input, there seems to be no advantage to using more than one hidden layer. But things get much more complicated when there are two or more inputs.

The best number of hidden units depends in a complex way on:

- the numbers of input and output units
- the number of training cases
- the amount of noise in the targets
- the complexity of the function or classification to be learned
- the architecture
- the type of hidden unit activation function
- the training algorithm
- regularization

In most situations, there is no way to determine the best number of hidden units without training several networks and estimating the generalization error of each. If you have too few hidden units, you will get high training error and high generalization error due to under-fitting and high statistical bias. If you have too many hidden units, you may get low training error but still have high generalization error due to high variance. Many NN programs may actually need closer to 100 hidden units to get zero training error.

7.6 Performance of Net

Several modifications can be made to the backpropagaion algorithm which may improve its performance in some situations. The modification involves changes to the weight update procedure.

7.6.1 Momentum

A simple way to improve the standard back propagation learning algorithm is to smooth the weight changes by over relaxation i.e. by adding the momentum. In backpropagation with momentum , the weight change is in a direction that is a combination of current gradient and previous gradient .This is a modification of gradient descent whose advantage arise chiefly when some training data are very different from the majority of the data (and possibly even incorrect) . It is desirable to use small learning rate to avoid a major disruption of the direction of learning when a very unusual pair of training pattern is presented. However it is also preferable to maintain training at fairly rapid pace as long as the training data are relatively similar

The momentum term may improve the convergence rate and the steady state performance of the algorithm. In order to use momentum, weight (or weight updates) from one or more previous raining pattern must be saved.

7.6.2 Batch Updating

In some cases it is advantageous to accumulate the weight correction term for several patterns (or even an entire epoch if there are not too many parameters) and make a single weight adjustment (equal to the average of the weight correction term) for each weight rather than updating the weight rather than updating the weights after each pattern is presented. This procedure has smoothing effect on the correction term. In some cases this smoothing may increase the chance of convergence to local minimum.

7.6.3 Adaptive Learning Rate

Researches have attempted to improve the speed of training by changing the learning rate during training. The adapted learning attempts to heap learning rate at each iterative steps large as possible while keep the learning process stable.

7.6.4 Delta Bar Delta

The general approach of delta bar delta algorithm is to allow each weight to have its own learning rate to let the learning rates by vary the time as training progresses. If the weight change is in the same direction for several time steps, the learning rate for that weight should be increased. However if the direction if the weights change alternates, the learning rate should be decreased.The delta bar delta consists of a weight upgrade rule and a learning update rule.

7.7 Summary

The topics covered are the fundamental to understanding Backpropagation feedforward Neural Network. Basically its architecture, algorithm, characteristics of activation functions and training methods are discussed in detail.

Chapter 7

Experiments and Results

7.1 Introduction

This chapter contains description of the experiments conducted along with their results. The evaluation measure mainly used was accuracy. Results were compiled for evaluation of normal or diseased liver. Data is trained and tested with the help of cross validation. At the end of this chapter observations based on the results are presented, which are carried out with the help of Operating Characteristics of ROC.

7.2 Cross Validation

In cross validation we randomly split the set of labeled training samples data into two parts: one is used as the traditional training set for adjusting model parameters in the classifier. The other set the validation set is used to estimate the generalization error [46]. Since our ultimate goal is low generalization error, we train the classifier until we reach a minimum of this validation error, as sketched in Fig. 7.1.

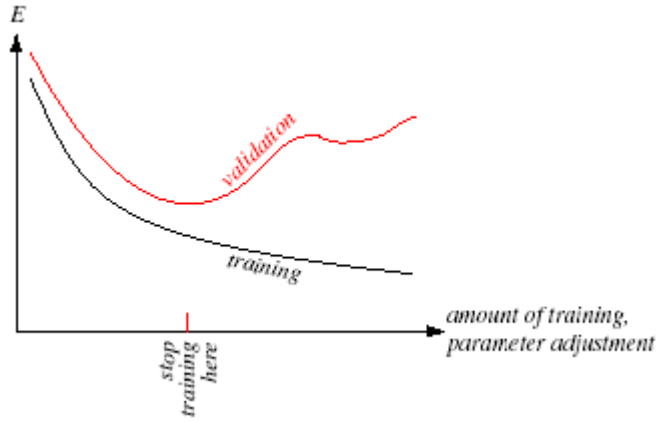


Figure 7.1: Cross Validation

In cross validation, the data set *D* is split into two parts. The first (e.g., 90% of the patterns) is used as a standard training set for setting free parameters in the classifier model; the other (e.g., 10%) is the validation set and is meant to represent the full generalization task. For most problems, the training error decreases monotonically during training, as shown in black in above figure. Typically, the error on the validation set decreases, but then

increases, an indication that the classifier may be overfitting the training data. In cross validation, training or parameter adjustment is stopped at the first minimum of the validation error.

Cross validation can be applied to virtually every classification method, where the specific form of learning or parameter adjustment depends upon the general training method. For example, in neural networks of a fixed topology, the amount of training is the number of epochs or presentations of the training set. Alternatively, the number of hidden units can be set via cross validation. Likewise, an optimal value of k in the k-nearest neighbor classifier can be set by cross validation [47].

Cross validation is heuristic and cannot give improved classifiers in every case. Nevertheless, it is extremely simple and for many real-world problems is found to improve generalization accuracy. There are several heuristics for choosing the portion γ of D to be used as a validation set ($0 < \gamma < 1$). Nearly always, a smaller portion of the data should be used as validation set ($\gamma < 0.5$) because the validation set is used merely to set a single global property of the classifier (i.e., when to stop adjusting parameters) rather than the large number of classifier parameters learned using the training set. If a classifier has a large number of free parameters or degrees of freedom, then a larger portion of D should be used as a training set, i.e., γ should be reduced. A traditional default is to split the data with $\gamma = 0.1$, which has proven effective in many applications. Finally, when the number of degrees of freedom in the classifier is small compared to the number of training points, the predicted generalization error is relatively insensitive to the choice of γ.

A simple generalization of the above method is *m-fold cross validation*. Here the training set is randomly divided into m disjoint sets of equal size n/m, where n is again the total number of patterns in D. The classifier is trained m times, each time with a different set held out as a validation set. The estimated performance is the mean of these m errors. In the limit where $m = n$, the method is in effect the leave-one-out approach.

Cross validation is a heuristic and may be not work on every problem. Indeed, there are problems for which *anti-cross validation* is effective, halting the adjustment of parameters when the validation error is the first local *maximum*. As such, in any particular problem

designers must be prepared to explore different values of γ, and possibly abandon the use of cross validation altogether if performance cannot be improved.

Cross validation is, at base, an empirical approach that tests the classifier experimentally. Once we train a classifier using cross validation, the validation error gives an estimate of the accuracy of the final classifier on the unknown test set. If the true but unknown error rate of the classifier is *p*, and if *k* of the *ń* independent, randomly drawn test samples are misclassified, then *k* has the binomial distribution

$$p(k) = \binom{n'}{k} p^k (1-p)^{n'-k}$$

Thus, the fraction of test samples misclassified is exactly the maximum likelihood estimate for *p.*

$$\hat{p} = \frac{k}{n'}$$

7.3 Experimental Setup

We used m folds cross validation in our experiments, where m is 10. The data set was divided into ten parts and then the experiments were repeated ten times, each time reserving different part for the testing and the remaining nine for the training purposes. All the results were then averaged over the entire set of experiments.

Two sets of experimental setup were used for the liver classification. In the first set of experiments the data was trained and tested with support vector machine. All the classes were trained and tested using the features set of for normal, Hepatoma, Hemangioma and Cirrhosis respectively.

SVM is trained three times in a hierarchical order, first time it is trained to classify between normal and diseased liver, which includes liver infected with Hemengioma, Hepatoma, or cirrhosis. Data for training is provided in such a way that 20 positive examples are provided for normal liver and rest of the 75 is negative (25 for each disease). Disease type cannot be classified at this stage. For this purpose we trained the second

SVM, which is trained to classify between Hemangioma and other disease (not Hemangioma) i.e., any other disease but other disease cannot be identified at this stage. Training set for second SVM includes 25 positive examples for Hemangioma and 50 negative examples for rest of the two diseases. Now this SVM is able to classify between Hemangioma and other disease (not Hemangioma). Third SVM is trained to classify between Hepatoma and Cirhhosis, where 25 positive and 25 negatives examples are used for Hepatoma and Cirrhosis respectively. Data sets used at each training stage are arranged in files and shown in hierarchical order in the Fig. 7.2.

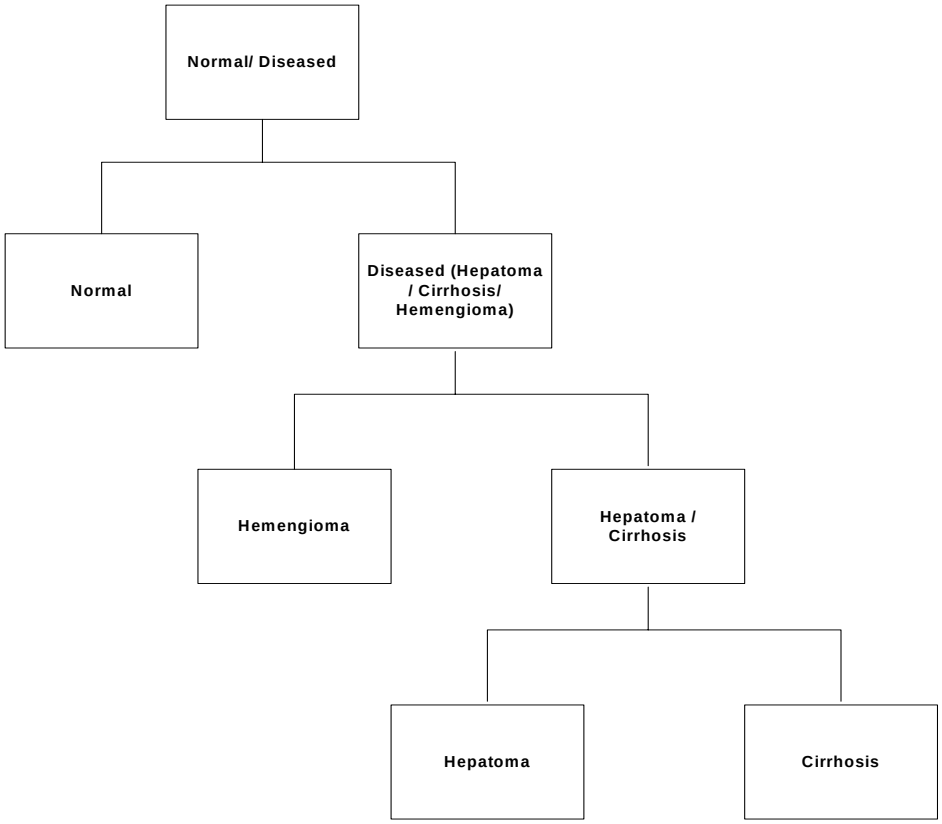


Figure 7.2: Hierarchy of CV training sets

Tables 7.1, 7.2 and 7.3 summarize the results of experimental setup. Table 7.1 shows results of classifying normal or diseased liver. Average cross validation error is about 18.9% and standard deviation is 15.8 for kernel parameters 0.4, 0.44, 0.484 and 0.5324.

Classification accuracy for diseased liver is 98.65% and for that of normal liver is 100%. As we increase the kernel parameter average cross validation error and standard deviation start decreasing, for kernel parameter 0.644204, 0.708624 and 0.779487 average cross validation error is 16.7% and standard deviation is 14.1033 and it gives 100% correct results for diseased liver and 93.8% for normal liver It shows best accuracy as it is more important to get more accurate results for diseased liver than normal, as already mentioned that identifying the disease at early stage is particularly important to cure it, and it's the key objective of CAD system.

Table 7.2 shows results of classifying Hemengioma. Average cross validation error is 30% and standard deviation is about 17 for kernel parameter 0.4, and shows 91.3% results for Hemangioma and about 70% for other diseases (not Hemangioma). As we increase kernel parameters average cross validation error and standard deviation increase. Classification accuracy for Hemengioma decreases to 89.13%, but increases for that of the other diseases (any other than Hemangioma).

Kernel parameter	**Avg CV error**	**Std deviation**	**Avg. Confusion matrix for predicted class**			
			TP	**FN**	**TN**	**FP**
0.4	18.889%	15.7571	98.65%	1.35%	100.00%	0.00%
0.44	18.889%	15.7571	98.65%	1.35%	100.00%	0.00%
0.484	18.889%	15.7571	98.65%	1.35%	100.00%	0.00%
0.5324	18.889%	15.7571	98.65%	1.35%	100.00%	0.00%
0.58564	17.778%	14.9989	98.65%	1.35%	93.75%	6.25%
0.644204	16.667%	14.1033	100.00%	0.00%	93.75%	6.25%
0.708624	16.667%	14.1033	100.00%	0.00%	93.75%	6.25%
0.779487	16.667%	14.1033	100.00%	0.00%	93.75%	6.25%

Table 7.1: Results of Experimental Setup for classifying Normal and diseased liver

Kernel parameter	Avg CV error	Std deviation	Avg. Confusion matrix for predicted class			
			TP	FN	TN	FP
0.4	30.000%	17.1031	91.30%	8.70%	70.83%	29.17%
0.44	31.429%	18.8080	91.30%	8.70%	75.00%	25.00%
0.484	31.429%	18.8080	91.30%	8.70%	75.00%	25.00%
0.5324	32.857%	19.1071	89.13%	10.87%	75.00%	25.00%
0.58564	32.857%	19.1071	89.13%	10.87%	75.00%	25.00%
0.644204	32.857%	19.1071	89.13%	10.87%	75.00%	25.00%
0.708624	32.857%	19.1071	89.13%	10.87%	75.00%	25.00%
0.779487	32.857%	19.1071	89.13%	10.87%	75.00%	25.00%

Table 7.2: Results of Experimental Setup for classifying Hemengioma and other disease (not Hemangioma)

Table 7.3 shows results of classifying Hepatoma and cirrhosis. This time results are not extraordinary. As average cross validation error remains between 58 to 60%, which is very high. Standard deviation varies from 14 to 18. Classification accuracy for Hepatoma and cirrhosis is around 50%, which is not very promising.

Kernel parameter	Avg CV error	Std deviation	Avg. Confusion matrix for predicted class			
			TP	FN	TN	FP
0.4	58.000%	14.7573	40.00%	60.00%	56.00%	44.00%
0.44	58.000%	14.7573	40.00%	60.00%	56.00%	44.00%
0.484	60.000%	16.3299	36.00%	64.00%	56.00%	44.00%
0.5324	58.000%	14.7573	40.00%	60.00%	56.00%	44.00%
0.58564	58.000%	14.7573	40.00%	60.00%	56.00%	44.00%
0.644204	58.000%	14.7573	44.00%	56.00%	60.00%	40.00%
0.708624	60.000%	13.3333	44.00%	56.00%	64.00%	36.00%
0.779487	60.000%	18.8562	44.00%	56.00%	64.00%	36.00%

Table 7.3: Results of Experimental Setup for classifying Hepatoma & Cirrhosis

7.4 Evaluation of Performance

In order to measure the performance of the ensemble of classifiers few operating characteristics based on ROC methodology were calculated, which use the information collected by cross validation. Whole process opted, is described in detail bellow.

7.4.l Receiver Operating Characteristic (ROC) Methodology

A simple way to asses the classification accuracy of a classifier is by measuring the average error or misclassification rate. Traditionally, assessment of classification performance in diagnostic systems is performed using ROC analysis because classifiers will tend to achieve a low misclassification rate by sacrificing the more rare class. Consider, for example, a scenario in which only 5% of samples are diseased, and the remaining is normal. A classifier that always blindly states that the disease is absent will be correct 95% of the time. As this is undesirable in medical applications, because if we neglect the misclassification for a diseased image and classify it as a normal image then disease may grow to the next stage which can be severe and may be hard to treat, so a method to modify the classifier's performance in favor of the disease class is needed. An ROC curve is helpful in achieving this goal.

Specifically, it allows visualization of the tradeoff between true-positive decisions and false-positive decisions for an ensemble of classifier configurations. Table 7.4 shows the list of possible outcomes of diagnostic tests.

Test Result	**Disease Status**	
	Present	Absent
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

Table 7.4: Possible outcomes of diagnostic tests

Consider a classifier that outputs a scalar value for a given pattern vector x showing the likelihood of the pattern belonging to the positive class. This scalar likelihood value, output from the classifier, is denoted here as g(x). Fig. 7.3 shows a typical distribution of

classifier outputs for a two-class dataset. Typically, classifiers assign category labels to test patterns by computing their likelihood score and comparing that score to a pre-determined threshold. For example, in a linear discriminant classifier the output discriminant score is often compared to zero. Depending on the convention used, if the discriminant score is greater than zero then the test pattern is assigned a positive class label, otherwise it is assigned a negative class label. One of the goals of ROC analysis is to provide a threshold (other than zero) that will favor the detection of the positive class.

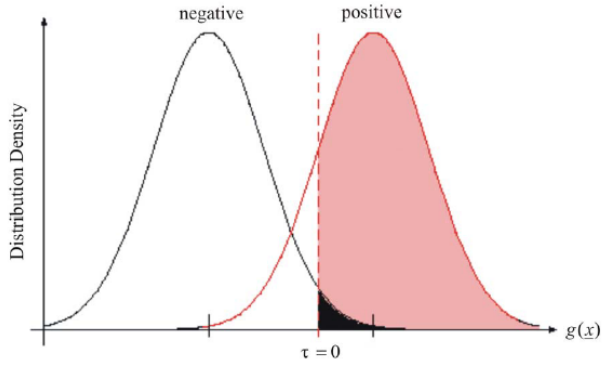


Figure 7.3: Typical distribution of classifier output g(x) on a two-class dataset.

At a given threshold value, there are two quantities that can be defined: sensitivity and specificity. Sensitivity, also known as the true positive fraction (TPF), is the fraction of positively labeled test data classified as being in the positive class. In Fig. 7.3, this is the shaded area under the positive class density curve, to the right of the decision threshold (zero), excluding the darkened region. On the other hand specificity, also known as the true negative fraction (TNF), is the fraction of negatively labeled samples classified as being in the negative class [48].

Along with specificity and sensitivity other characteristics like positive predictivity and negative predictivity of the classifier is also calculated. These commonly used operating characteristics quantities of diagnostic tests are mathematically defined in Table 7.4 bellow.

Characteristic	Formula	Definition
Sensitivity (TPF)	$\frac{TP}{TP+FN}$	Proportion of people with condition who test positive.
Specificity (TNF)	$\frac{TN}{TN+FP}$	Proportion of people without condition who test negative.
Positive Predictivity	$\frac{TP}{TP+FP}$	Proportion of people with positive test who have condition.
Negative Predictivity	$\frac{TN}{TN+FN}$	Proportion of people with negative test who do not have condition.

Table 7.5: Operating Characteristics of diagnostic tests

The receiver operating characteristic curve is a plot of TPF versus FPF (False Negative Fraction) for a range of values. The *receiver* can *operate* at any point on the curve by using an appropriate decision threshold. In general, it is better to operate on the lower left part of the ROC curve to keep the FPF small, even at the expense of a low TPF. Fig. 7.4 shows a few example ROC curves. The diagonal straight line in the figure is what would be obtained by "biased" random guessing [49].

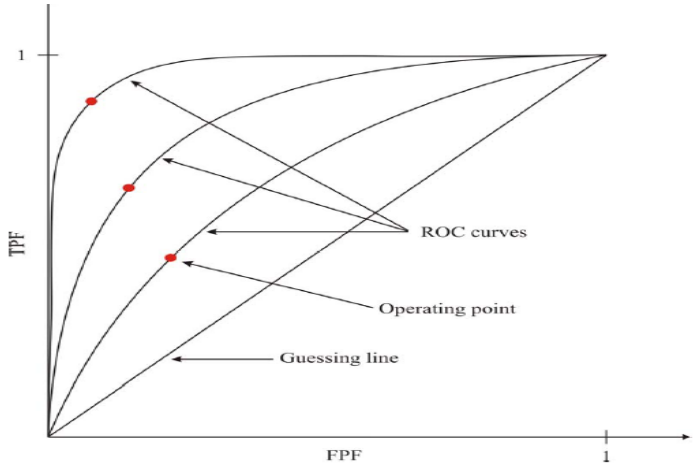


Figure 7.4 Example ROC curves.

In the context of ROC curves, the accuracy of a classifier is estimated by computing the area under the ROC curve. The value summarizes the quality of classification over an ensemble of misclassifications [49].The greater the area, the higher the classification accuracy.

7.4.2 Performance Measure

To determine the efficacy of each SVM classifier, the performance is estimated via ROC analysis. Results obtained from the classifiers are used for calculating the operating characteristics shown in table 7.5, where the total accuracy was computed by calculating the ratio of the number of correct responses to the total number of images. Some of the quantities have been previously described in Table 7.4.

Characteristic	Values for ker.par. 0.4	Values for ker.par. 0.44	Values for ker.par. 0.484	Values for ker.par. 0.5324	Values for ker.par. 0.58564	Values for ker.par. 0.644204	Values for ker.par. 0.708624	Values for ker.par. 0.779487
TP	98.65%	98.65%	98.65%	98.65%	98.65%	100.00%	100.00%	100.00%
FN	1.35%	1.35%	1.35%	1.35%	1.35%	0.00%	0.00%	0.00%
TN	100.00%	100.00%	100.00%	100.00%	93.75%	93.75%	93.75%	93.75%
FP	0.00%	0.00%	0.00%	0.00%	6.25%	6.25%	6.25%	6.25%
Sensitivity (TPF)	98.65%	98.65%	98.65%	98.65%	98.65%	100%	100%	100%
Specificity (TNF)	100%	100%	100%	100%	93.75%	93.75%	93.75%	93.75%
Positive Predictivity	100%	100%	100%	100%	94.11%	94.11%	94.11%	94.11%
Negative Predictivity	98.66%	98.66%	98.66%	98.66%	100%	100%	100%	100%

Table 7.6: Results of SVM classifier trained for normal and diseased liver

To reduce biases in the classifiers due to case selection, training and testing were repeated several times, each with a different training and test dataset. Test values were averaged to provide a measurement of the effectiveness of that classifier.

Characteristic	Values for ker.par. 0.4	Values for ker.par. 0.44	Values for ker.par. 0.484	Values for ker.par. 0.5324	Values for ker.par. 0.58564	Values for ker.par. 0.644204	Values for ker.par. 0.708624	Values for ker.par. 0.779487
TP	91.30%	91.30%	91.30%	89.13%	89.13%	89.13%	89.13%	89.13%
FN	8.70%	8.70%	8.70%	10.87%	10.87%	10.87%	10.87%	10.87%
TN	70.83%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%
FP	29.17%	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%
Sensitivity (TPF)	91.30%	91.30%	91.30%	89.13%	89.13%	89.13%	89.13%	89.13%
Specificity (TNF)	70.83%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%
Positive Predictivity	75.78%	78.50%	78.50%	65.71%	65.71%	65.71%	65.71%	65.71%
Negative Predictivity	89.06%	89.60%	89.60%	87.346%	87.34%	87.34%	87.34%	87.34%

Table 7.7: Results of SVM classifier trained for Hemangioma and other diseased (not Hemangioma) liver

Characteristic	Values for ker.par. 0.4	Values for ker.par. 0.44	Values for ker.par. 0.484	Values for ker.par. 0.5324	Values for ker.par. 0.58564	Values for ker.par. 0.644204	Values for ker.par. 0.708624	Values for ker.par. 0.779487
TP	40.00%	40.00%	36.00%	40.00%	40.00%	44.00%	44.00%	44.00%
FN	60.00%	60.00%	64.00%	60.00%	60.00%	56.00%	56.00%	56.00%
TN	56.00%	56.00%	56.00%	56.00%	56.00%	60.00%	64.00%	64.00%
FP	44.00%	44.00%	44.00%	44.00%	44.00%	40.0%	36.00%	36.00%
Sensitivity (TPF)	40.00%	40.00%	36.00%	40.00%	40.00%	44.00%	44.00%	44.00%
Specificity (TNF)	56.00%	56.00%	56.00%	56.00%	56.00%	60.00%	64.00%	64.00%
Positive Predictivity	47.61%	47.61%	45.00%	47.61%	47.61%	52.38%	40%	40%
Negative Predictivity	48.27%	48.27%	46.66%	48.276%	48.27%	51.72%	53.33%	53.33%

Table 7.8: Results of SVM classifier trained for Hepatoma and Cirrhosis

Results of sensitivity and specificity show that classifiers perform well at lower kernel parameters except few cases where these characteristics measures start improving in performance. As we already observed that the performance of third SVM is not very promising.

7.5 SUMMARY

In this chapter the detailed results of the experiments were presented. The results were compiled using ensemble of classifiers i.e. three sets of SVMs. Cross validation was used to train and test the classifiers and to calculate the errors. ROC characteristics are calculated for different kernel parameters.