Feature Selection and Tumor Identification in Brain Magnetic Resonance Imaging using Hybrid Swarm Intelligence and Machine Learning

By

ATIQ UR REHMAN

Computer Engineering (MS-69)

Department Of Computer Engineering EME College NUST Rawalpindi

Session (2011-2013)

A thesis submitted to the faculty of Computer Engineering Department, Electrical and Mechanical Engineering College, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

Approved By

Supervisor : Dr. Aasia Khanum

GC Members

1. Dr. Arslan Shokat
2. Dr. Saad Rehman
3. Dr. Farhan Mehmood

Date:

COPY RIGHTS

ACKNOWLEDGEMENT

First I would like to thank Almighty Allah to give me the opportunity and strength to complete my thesis successfully. I thank my parents for their inspiration and encouragement through the progress of this work.

I also would like to thank my supervisor, Dr. Aasia Khanum for her many suggestions and constant support during this work. I am very grateful for those research opportunities she provided me and I believe this research experience will greatly benefit my career in the future. I learned from Dr. Aasia Khanum, that persistent effort and faith are undoubtedly the most important assets.

All of my GC Members, I wish to express my heartiest gratitude to Dr. Arslan Shokat, Dr. Farhan and Dr. Saad Rehman on their invaluable advice and encouraging support. They not only supervised me to materialize the work but also showed me the way to think in different dimensions and work steadily from the very beginning of a research. Their continuous support helped me to feel my confinement with the topic all through my research.

Appreciation is extended to all people who answered my questions and provided me with valuable information and support in my research.

ABSTRACT

Automatic classification of Brain MR Images is one of hotspots in the field of Medical Imaging. Feature Selection of Brain MRI is crucial and it affects the classification results; however selecting optimal Brain MRI features is difficult. Particle Swarm Optimization (PSO) is an evolutionary computing technique which was developed observing the social behavior of bird flocks. PSO is an intelligent technique containing properties like adaptation and self-organizing, these intelligent properties make PSO one of the best techniques for searching the optimal solution for optimization problems. The difficulty mentioned above is solved using the Discrete Binary PSO technique which proved it self quite successful. Population of Particles is generated and the search strategy is applied so that an efficient feature selection process for Brain MRI is proposed. Classification of normal and tumor Brain MRI is carried out using PSO and experimental results show that it obviously reduces the number of features and at the same time it achieves high accuracy level. Three different classifiers, Support Vector Machine (SVM), Naïve Bayes Classifier (NB) and K-Nearest Neighbor Classifier (KNN) are utilized to serve as the 'fitness function' for PSO. A comparison of achieved accuracy with minimum number of features for different classifiers serving PSO is also provided. PSO-SVM and PSO-KNN are observed to achieve better accuracy level using minimum number of selected features than PSO-NB.

Table Of Contents

List Of Figures:

List Of Tables:

Introduction

Brain MRI has proved to be a successful tool in collecting in-depth detail of brain tissues. Now a days medical science highly relies on MRI, and automation of this tool has to be more and more efficient. To make the classification problem of Brain MRI easy and efficient efforts are made and a lot of contributions in this stream are already provided by different scientists. Brain MRI is highly sensitive issue and to classify the normal brain images from abnormal ones is quiet difficult. MRI contains a lot of textural information, so using textural features for classification is beneficial. Using a number of features for classification is also not desirable, so the evolutionary computation technique PSO is utilized for feature selection. A subset of features provided by PSO is passed to different classifiers for classification and the achieved accuracy with minimum number of features is discussed. Three different classifiers are used, namely Support Vector Machine (SVM), Naïve Bayes Classifier (NB) and K-Nearest Neighbor Classifier (KNN).

As stated before, an efficient feature selection process is always required to speed up processing and predictive accuracy for a classifier. We used Discrete Binary Particle Swarm Optimization (PSO) technique for feature selection of Magnetic Resonance (MR) Brain Images and classification is carried out on tumor and normal Images. Here PSO is used for feature selection and Support Vector Machine (SVM), Naïve Bayes Classifier (NB) and K-Nearest Neighbor Classifier (KNN) serves as fitness functions for PSO. The proposed method is applied to a dataset of twenty T1 weighted brain MR images (ten normal images and ten tumor images) and desired classification accuracy is achieved. A comparison of achieved accuracy results for unseen data (testing performance) is also provided and PSO-SVM is observed to achieve comparatively better accuracy level then PSO-NB and PSO-KNN.

Feature Selection has always been a challenging task used in machine learning for classification problem. Feature Selection is a global optimization process in machine learning which reduces the number of features selected by removing irrelevant, redundant and noisy features. By omitting these irrelevant features we are left with more efficient features that are helpful in classification problem. Our main focus is to utilize the textural features provided by Gray Level Cooccurance Matrix (GLCM). A total of twenty three features are evaluated, among these twenty two were GLCM based features and one extra wavelet feature is evaluated. All these features are discussed in detail latter.

CHAPTER 1

Magnetic Resonance Imaging and Brain Tumors

This chapter briefly describes the basics of Magnetic Resonance Imaging (MRI), the basic principle of working of MRI, image acquisition and its use in extracting the in-depth knowledge of soft tissues, brain in particular. Brain tumors and their types are also discussed shortly.

1.1 Brief History Of MRI

In 1950s phenomenon of Nuclear Magnetic Resonance (NMR) was discovered in bulk matter and for years it remained applicable only in the field of spectroscopy. In 1970s the concept of magnetic field gradients was introduced by Lauterbur, he stated that an image based on magnetic resonance could be produced. By the 1980s whole body magnets were being produced in England permitting the first in vivo images of human anatomy. Today the technique, known as MR imaging, is widespread and an estimated 20 million scans are performed worldwide each year. It provides images with excellent soft-tissue contrast which can be acquired in any imaging plane, and unlike CT it does not involve the use of ionizing radiation. It is the imaging modality of choice in brain and spinal cord and is routinely used in many other clinical settings.

Magnetic Resonance Imaging (MRI) provides in-depth detail about the anatomy of soft tissue organs, so it is helpful in collecting details about brain tumor anatomy and cellular structure using MRI. This makes it an important tool to help us diagnose the level and kind of disease. We can easily treat and monitor the disease using this tool. There are certain kinds of brain tumor which are listed below.

1.2 Brain Tumors

Brain tumor is a kind of abnormal and uncontrollable growth of cells, brain tumors are termed as primary and secondary depending upon origin of growth of tumor. If the origin of tumor growth is brain itself then it is termed primary tumor. If the tumor grew from some other part and reached the brain then it is termed secondary. Primary brain tumors do not grow to the other parts of body and these can be either malignant or benign, whereas secondary tumors are always malignant. Both of these types are equally life threatening.

Children and young adults are more likely to be effected by the brain tumors and these are unfortunately second leading cause of cancer-related deaths .Central Brain Tumor Registry of the United States (CBTRUS) cites that there were approximately 64,530 cases of primary brain and central nervous system tumors diagnosed in the year 2011. Overall, exceeding a number of 600,000 people were already living with the disease. [1]

It has always been a challenging task to diagnose the cause of brain tumor and the cause of occurrence is still unknown. There are a few risk factors that are involved which are head

injuries, hereditary syndromes, immunosuppression, prolonged exposure to ionizing radiation, electromagnetic fields, cell phones, or chemicals like formaldehyde and vinyl chloride. None of these, however, is proven to actually cause the disease. [2]
Symptoms of brain tumors include persistent headache, nausea and vomiting, eyesight, hearing and/or speech problems, walking and/or balance difficulties, personality changes, memory lapses, problems with cognition and concentration, and seizures. [2]

1.3 MRI Scan of Brain Tumor

MRI is a useful tool for tumor detection that provides us detailed information about different aspects of tumor such as its type, its position and size. For this same reason, MRI is the most widely used imaging study of choice for the diagnosis procedure, surgery workout and monitoring of treatment outcomes.

1.4 MRI Signal Generation and Image Acquisition

Traditional MRI approach uses three different physical properties of tissue protons (which are listed in Table 1.1) to produce signals that are used to acquire images of different contrast areas, which tells us the anatomy and physiology of the organ under investigation (Brain in our case). [3]

Proton density	Number of hydrogen protons per unit volume of tissue
T1 relaxation time	Time needed to recover 63% of the longitudinal magnetization
T1 relaxation time	Time needed for 63% of the traverse magnetization to be lost

Table 1.1 Protons' Properties Contributing to MRI Signal Generation

Positively-charged particles called Protons are present inside the nucleus of an atom. Human body is mainly made up of water, therefore hydrogen element is present in abundance in human body, as each atom of hydrogen has one proton. To understand the process of formation of MRI signal, we can think of this proton as a very small magnet bar that is moving like a spinning top. This tiny magnet bar can be described as a two-dimensional (2D) vector–also called a *spin vector*–with respect to a Cartesian coordinate system. [4]

In the absence of a magnetic field, the orientation of the spin vector is random. MRI scanners produce a strong static magnetic field (B0), if the spin vector is placed in such strong field B0 then it tends to align itself parallel with B0. As a result of this, its component along the y-axis (*longitudinal magnetization*) has some value which is not zero but the component along the x-axis (*transverse magnetization*) is zero, as a result of this no signal can be produced. [4], [5] When a pulse of radio frequency (RF) is applied, energy is absorbed by the spin vector and it causes the spin vector to move away from the alignment that was formed by spin vector with B0. If the oscillation frequency of the proton matches with the frequency of applied RF pulse then the phenomenon known as resonance occurs. By doing this both the longitudinal and transverse magnetizations have some values other than zero and a signal can be generated (Figure 1.1). [5]

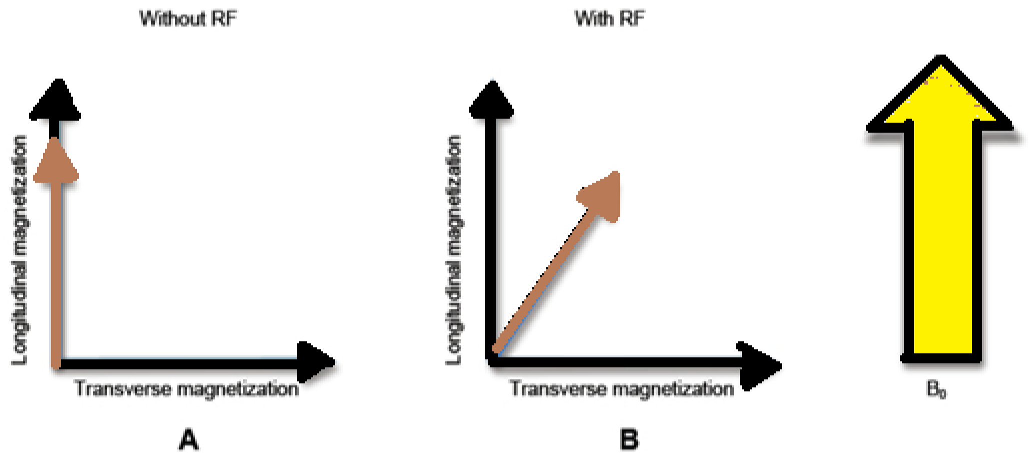


FIGURE 1.1 MRI Signal Generation *(A) Spin vector (brown arrow) aligned parallel with the main magnetic field (B_0). Transverse magnetization is zero, and no signal is generated. (B) Spin vector out of alignment, following the application of an RF pulse. Transverse magnetization is different from zero, and a signal can be generated.*

A process called *relaxation* occurs when RF pulse is removed from B0 because by doing so the energy absorbed by the spin vector is released. On releasing this energy the spin vector again aligns itself with the main magnetic field (B0). When it happens, the transverse magnetization decreases, and the longitudinal magnetization recovers. *T1-relaxation time* is 63% of the time required to recover the longitudinal magnetization. *T2-relaxation time* is 63% of the time in which transverse magnetization is lost. Both T1- and T2-relaxation times are specific to different types of tissue, and are directly proportional to the magnet's field strength. [5]

T1 and T2 are the two main components of the MRI signal. Proton density is also a third main component it is the total number of hydrogen protons per unit volume of the tissue being scanned. Contrast is generated using these three components. This is all done simultaneously, so the image formed is confusing and difficult to read. To handle this problem the contrast is weighted towards one of the three signal components. As a result, the images that can be obtained are: T1-weighted, T2-weighted, and proton density-weighted. [3], [5]

Contrast is obtained in different ways:

- ✓ Creating differences in the T1 *recovery* times of tissues (T1-weighted images).[6]
- ✓ Creating differences in the T2 *decay* times of tissues (T2-weighted images).[6]

- ✓ From differences in the proton density of tissues (proton-weighted images).[6]

While reading an MRI scan, the bright areas are the high-fat-content tissues in T1-weighted images, whereas dark areas are the high-water-content tissues. The opposite is the case for T2-weighted images. Particularly T2-weighted images are useful for pathological investigation studies. Whereas, T1-weighted images are good for anatomical investigations, although they can be used for pathology, if combined with contrast enhancement. Gadolinium (Gd) is currently the standard contrast agent due to its ability to cross the blood-brain barrier (BBB). [6]

Bright areas are the high-proton-density tissues in proton-weighted images, this can be the cerebrospinal fluid, and dark areas are the low-proton-density tissues, such as cortical bone. Proton-density images are used to view anatomy and some pathology. [6]

(Table 1.2, provides an overview of the key features of T1, T2, and proton-weighted images.)

Image Type	**Contrast**	**Tissue Appearance**	**Best For**
T1-weighted	Mainly due to differences in T1 recovery times	High-**fat**-content tissues appear as **bright** areas of high signal intensity (hyperintense) High-**water**-content tissues appear as **dark** areas of low signal intensity (hypointense)	Anatomy and, if used with contrast enhancement, also pathology
T2-weighted	Mainly due to differences in T2 decay times	High-**fat**-content tissues appear as dark areas of low signal intensity (hypointense) High-**water**-content tissues appear as bright areas of high signal intensity (hyper-intense)	pathology
Proton-density weighted	Mainly due to differences in proton density	Low-proton-density tissues appear as dark areas of low signal intensity (hypointense) High-proton-density tissues appear as bright areas of high signal intensity (hyper-intense)	Anatomy, and some pathology

*Table 1.2 Types, Characteristics and Use of Weighted Images****

*Data from Westbrook C. *MRI at a Glance.* Wiley-Blackwell; 2010.

Examples of T1- and T2-weighted MR Brain Images are shown in Figure-1.2 and Figure-1.3

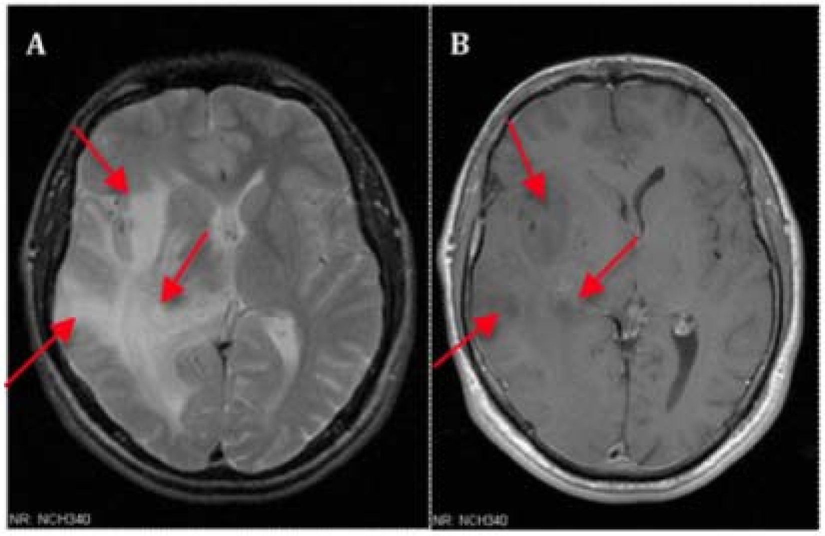


FIGURE 1.2. **(A) Axial T2-weighted image of low-grade glioma** *in the right hemisphere, showing the lesion as a bright area of high-signal intensity.* **(B)** *The same glioma on a* **T1-weighted image** *with contrast enhancement, showing the lesion as a dark area of low-signal intensity. (Photo courtesy of Professor Wolfgang Wick and Professor Martin Bendszus, of the Neurooncology Department of Heidelberg University, Germany.)*

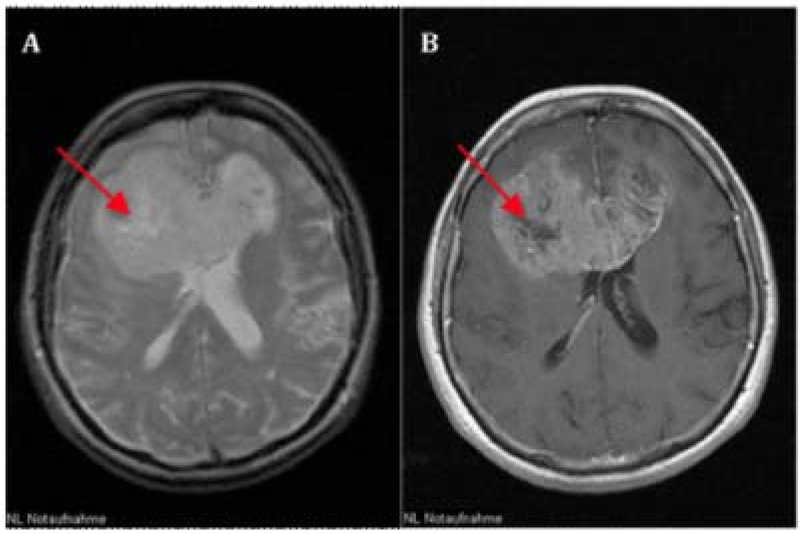


FIGURE 1.3 (A) Axial T2-weighted image of IV-grade glioma *(glioblastoma) in the right hemisphere, showing the lesion as a bright area of high-signal intensity.* **(B)** *The same glioblastoma on a* **T1-weighted image with contrast enhancement**, *showing the lesion as a dark area of low-signal intensity. (Photo courtesy of Professor Wolfgang Wick and Professor Martin Bendszus, of the Neurooncology Department of Heidelberg University, Germany.)*

CHAPTER 2

Particle Swarm Optimization

PSO is a population based global optimization procedure that deals with problems in which a best solution is represented as a point or surface in an m-dimensional space, developed by Kennedy and Eberhart in 1995 [7]. Basically PSO was initiated using two different ideas, first is the idea of swarm intelligence based on experiencing the significant details of collective habits by certain kind of animals, birds and fish, and other idea is the field of evolutionary computation.

This chapter explains the PSO algorithm and its use in feature selection in detail. As PSO is an optimization technique so let's start with optimization.

2.1 Optimization

Optimization is a method in which maximum or minimum value of a function or a process is obtained. Different fields of science like mathematics, economics, and particularly engineering utilize this efficient technique for maximizing performance efficiency. Both minimization and maximization of a function f are referred as optimization [9].

Mathematically, we can define a minimization process as:

$$Given\ f: \mathbb{R}^n \to \mathbb{R}$$

$$Find\ \hat{y} \in \mathbb{R}^n such\ that\ f(\hat{y}) \leq f(y), \forall y \in \mathbb{R}^n$$

Similarly, maximization process of a function f is defined as:

$$Given\ f: \mathbb{R}^n \to \mathbb{R}$$

$$Find\ \hat{y} \in \mathbb{R}^n such\ that\ f(\hat{y}) \geq f(y), \forall y \in \mathbb{R}^n$$

The domain ($\mathbb{R}^n$) of f defined above is actually the search space[9]. Each element (of this domain) is called a 'candidate solution' in the search space, with ŷ being the optimal solution. The number of dimensions of the search space is denoted by the value *n*, and thus the number of parameters involved in the optimization problem. The objective function is actually the function f itself, which maps the search space to the function space. Since a function has only one output, this function space is usually one-dimensional. The function space is then mapped to the one-dimensional fitness space, providing a single fitness value for each set of parameters. This single fitness value determines the optimality of the set of parameters for the desired task. In most cases, the function space can be directly mapped to the fitness space. However, the distinction between function space and fitness space is important in cases such as multi-objective optimization tasks, which include several objective functions drawing input from the same parameter space[11,9, 10].

There are a lot of calculus techniques by which one can easily find the minima and maxima of a known differentiable function f . In real-life optimization tasks, finding an objective function to be optimized is a difficult task and in most of the cases it is unknown. Instead, the parameters (candidate solution) are applied to an objective function which is a kind of "black box" and receive an output value. Solution's fitness is basically the result of this evaluation of a candidate solution. The final task of an optimization process is to find the parameters in the search space that can minimize or maximize this fitness. [11,8]

Figure 2.1 shown below is a simple example of function optimization. This figure shows a selected region the function f, demonstrated as the curve seen in the diagram. This function maps from a one-dimensional parameter space—the set of real numbers R on the horizontal x-axis—to a one-dimensional function space— the set of real numbers R on the vertical y-axis. The x-axis represents the candidate solutions, and the y-axis represents the results of the objective function when applied to these candidate solutions. This type of diagram demonstrates what is called the fitness landscape of an optimization problem [11,9]. The fitness landscape plots the n-dimensional parameter space against the one-dimensional fitness for each of these parameters.[11]

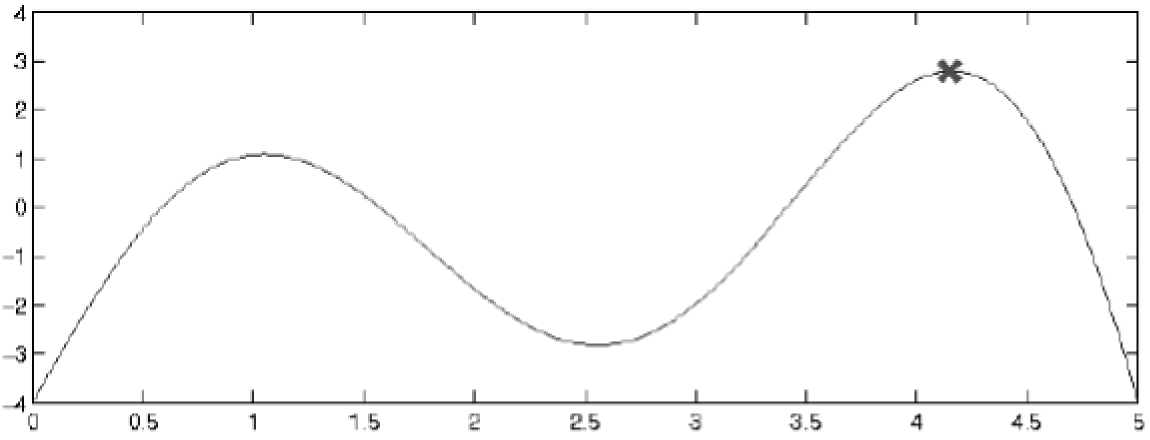


Figure 2.1: Function Maximum

Figure 1 also shows the presence of a local maximum in addition to the marked global maximum. A local maximum is a candidate solution that has a higher value from the objective function than any candidate solution in a particular region of the search space. For example, if we choose the interval [0,2.5] in Figure 1, the objective function has a local maximum located at the approximate value x = 1.05. Many optimization algorithms are only designed to find the local maximum, ignoring other local maxima and the global maximum.[11]

2.2 PSO Algorithm

The particle swarm optimization (PSO) algorithm is an optimization algorithm that works by maintaining different candidate solutions (particles) in search space at the same time, that is it keeps several candidate solutions in the search space simultaneously. It uses a number of particles that constitute a swarm which is trying to find out the best solution while moving in a search space. There are a number of iterations that are performed on this search space of candidate solution depending upon the achieved or desired optimum solution. Each candidate solution is evaluated against the desired objective function during each iteration (this determines the fitness of the candidate solution). As PSO was originated by observing the social behavior of different species like fish and birds, so each candidate solution is considered as a ‘bird’ (particle) ‘flying’ (searching) in a ‘landscape’ (search space) finding the best solution that fulfils an objective function (e-g finding food in case of birds and finding maximum or minimum value in case of optimization).

Initially the candidate solution is chosen at random among the entire search space.
Five-particle PSO searching the global minimum in a one-dimensional search space is shown in Figure 2.2.

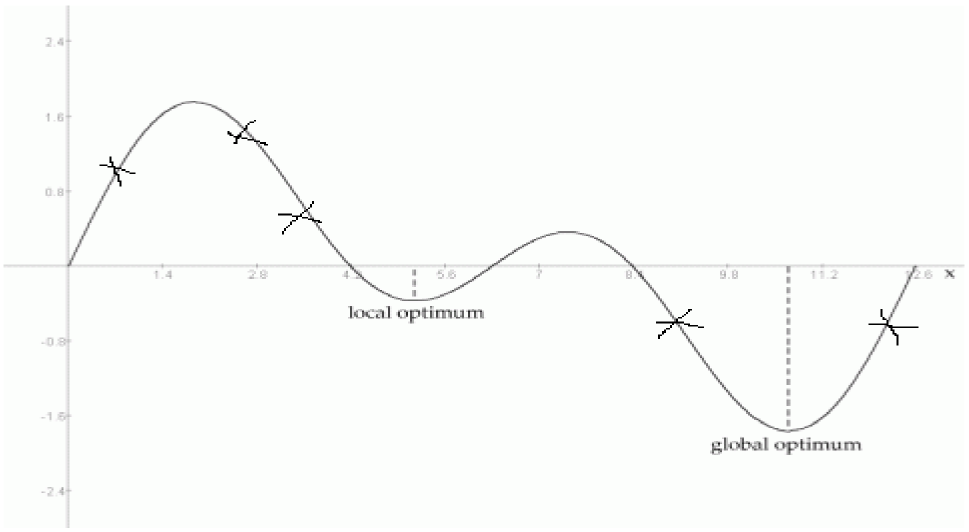


Figure 2.2 PSO Initial State

The five particles chosen at random initially are marked ‘x’ on the curve and the curve actually is the objective function, and search space is all the possible solutions along the x-axis. The objective function (the curve) is actually unseen by PSO, so the algorithm has no knowledge whether a particle is close-to or far away from the optimum solution. PSO will evaluate all the five particles against the objective function, and will make decisions on the resultant fitness values.

There are certain parameters which are tracked by PSO, these include the position and velocity of a particle. PSO keeps track of the achieved fitness value by every candidate solution, the best solution or the fitness value achieved by a particle is known as the ‘personal best’ value. The position of candidate solution which achieved the personal best is memorized. Another important parameter that is to be kept in record is the ‘global best’ value, it is the best value achieved so far by any of the particle among all the particles in the swarm. The record of global best value achieved and position of the particle which achieved this global best value are maintained.

Both the global best and personal best value are updated during every iteration of the algorithm if applicable. That is if during the next iteration the new values of personal best and global best are better than the older ones, these are replaced.

The PSO algorithm consists of three basic steps, which are iterated until some stopping criteria is fulfilled [8]:

1. Compute the fitness of each particle against the objective function.
2. Update personal best and global best fitness and positions of particles.
3. Update velocities and positions of particles in the swarm.

In the first step the fitness evaluation of each particle is conducted ageing the objective function, and in second step the newly computed fitness are compared with the previously computed fitness, and satisfying the objective function older personal best and global best values are replaced with the new ones.

The optimization ability of PSO highly depends upon the updates of velocities and positions of the particles. The velocity of a particle is updated using equation (2.1).

$$\mathrm{Vi}(t+1) = W * \mathrm{Vi}(t) + C1 * r1 * [p(t) - \mathrm{Xi}(t)] + C2 * r2 * [g(t) - \mathrm{Xi}(t)] \qquad (2.1)$$

- ✓ ‘i’ is the index of the particle whose velocity is to be updated.
- ✓ Vi(t) is the velocity of particle ‘i’ at time t.
- ✓ Xi(t) is the position of particle ‘i’ at time t.
- ✓ The variable W is a user specified value mostly between ($0 \leq W \leq 1.2$).
- ✓ The variables C1 and C2 are also user specified in between ($0 \leq C1 \leq 2$ and $0 \leq C2 \leq 2$).
- ✓ G(t) is the swarm’s global best value at time t.
- ✓ P(t) is the personal best value of a particle at time t.

There are a total of three terms involved in the update of velocity of particles and all of these three terms have different roles in PSO algorithm. The first term *‘W*Vi(t)’* is responsible to keep the direction of a particle same as it was originally moving. This term is known as the inertia component. The value of the inertia constant *W* is responsible to dampen the particle or to accelerate it in its original direction [12].
“Generally, lower values of the inertial coefficient speed up the convergence of the swarm to optima, and higher values of the inertial coefficient encourage exploration of the entire search space” [11].

The second part of equation is known as the ‘cognitive component’ *(C1*r1*[p(t)−Xi(t)]),* this is actually the memory of a particle. The regions in which the particle experienced personal best fitness are memorized and the particle is moved towards this region. *C1* is the ‘cognitive coefficient’ which is responsible for the stepsize the particle takes to move towards the personal best fitness solution. Usually the value of *C1* is kept close to 2.

The third part of equation is known as the ‘social component’ *(C2*r2*[g(t)−Xi(t)])*. The regions in which the swarm experienced global best fitness are memorized and the particle is moved towards this region. *C2* is the ‘social coefficient’ which is responsible for the step size the particle takes to move towards the global best fitness solution. Usually the value of *C2* is also kept close to 2.

A stochastic influence is generally induced by the two random values *r1* and *r2* in cognitive and social components. These two random values will cause the particle to move in a manner which is semi-random in nature, it will be influenced by the directions of both the personal best fitness value of a particle and the global best fitness value of swarm.

If it is desired that the particles must not move beyond the region of search space a technique known as 'velocity clamping' is used so that the maximum velocity of a particle is limited[8]. If the search space is limited to a range of *[-Xmax , Xmax]* then the velocity is limited to a range *[-k*Xmax , k*Xmax]*. The value of constant k is kept in-between $0 < k \leq 1$. In most of the cases the search space is likely to bound between *[Xmin, Xmax]* in such a scenario the maximum velocity is limited to $Vmax = k \times (Xmax - Xmin)/2$.

After updating the velocity of a particle next is the update of its position, this can be done using the following equation:

$$Xi(t + 1) = Xi(t) + Vi(t + 1) \qquad (2.2)$$

There are certain stopping criterion which when fulfilled the PSO algorithm stop its execution until then this task of updating velocities and positions of particles is repeated. The stopping criterion may include the total number of iterations or the achievement of targeted fitness value.

2.3 Basic Principle of Discrete Binary PSO

There are basically two kinds of optimization problems one is the continuous optimization problem which can be solved using the standard PSO technique, other is the combination optimization problem this is solved using the discrete binary version of PSO technique [19]. The discrete binary version of PSO also makes use of population of individuals and tries to find the best position by searching the entire swarm. For finding the best position it makes use of the individual's best position and the swarm's best position. [19]

Suppose a swarm of size N is created and the dimension of the search space is M, then the position of particle is described as $Xi(x_{i1}, x_{i2} \cdots, x_{iM})$ where x_{iM} is one bit binary number either 0 or 1. If x_{iM} is 1 it means that the feature at this location is selected, if it is zero than the feature is not selected. The personal best position of the particle is denoted by $\mathrm{P}i(p_{i1}, p_{i2} \cdots, p_{iM})$ and the globle best position of the swarm is dented by$Gi(g_{i1}, g_{i2} \cdots, g_{iM})$. The particles velocity is updated according to equation (2.1) and position of the particles is updated according to equation (2.3) and (2.4).

```
If (rand() < S( Vi(t) ) )
    $x_{iM} = 1$
Else
    $x_{iM} = 0$                                    (2.3)
```

$$S(Vi(t)) = 1/(1+exp(-Vi(t))) \qquad (2.4)$$

rand() is a random number with uniform distribution *U(0,1)*. [19]

The fitness function for PSO is the evaluation of position of the particle which in this case is the measure of testing error achieved by the classifier after using the position of particle for classification. This is defined in equation (2.5).

$$Fitness = TE\ (x_{i1}, x_{i2} \cdots, x_{iM}) \tag{2.5}$$

The entire procedure followed to use PSO as a feature selection technique (Flow Chart) is discussed in the implementation section latter.

CHAPTER 3

Classifiers

Support Vector Machine, Naïve Bayes and K-Nearest Neighbor

This particular chapter is about the three different classifiers used to serve the purpose of fitness function for PSO. All the three classifiers SVM, NBC and KNN are separately discussed in detail, their working and use as a classifier is discussed. It also discusses feature selection and classification in machine learning briefly.

3.1 Feature Selection:

When we have a number of features to discriminate between different classes the system becomes computationally intensive, so we must reduce the number of features. Feature selection is a process of selecting a subset of features from a large number of features. The subset of features selected is such that it is powerful to discriminate among different classes. Feature selection is one of most challenging task in machine learning or pattern recognition. It is actually the process of selecting most efficient and powerful subset of features which can easily discriminate different classes. The features from different classes should maintain ideally a large distance. By following the feature selection process we actually omit the irrelevant and redundant features, the main aim is actually to left with most useful features.

Feature selection is a search technique that searches for the best features among a number of features, best features are selected on the basis of an evaluation measure. Features are evaluated against an objective function and are selected based on their performance. The objective is to find the minimum number of features with high efficiency, the number of features selected should be minimum and the performance should be maximum.

3.2 Classification in Machine Learning:

The term classification is applied to identification, recognition, and discrimination of objects or patterns on the basis of their attributes. In machine learning classification 'classifies' an observation based on the previous knowledge. The system is trained using the attributes (features) of the training data, and is tested using both the training data and unseen testing data. The error due to training data is known as training error, and error due to testing data is known as testing error. The training error of a classifier is usually low because the classifier is trained using the same data, aim is to reduce the testing error because this data is unseen data for the classifier.

3.3 Classifiers:

Three classifiers are used here which are, support vector machine (SVM), naïve bayes classifier (NBC) and k nearest neighbor (KNN) classifier. These are discussed in detail below:

3.4 Support Vector Machine

Support Vector Machine (SVM) is an arguably most efficient classification method in machine learning. A very nice package of optimization techniques are used to derive the solution which has a very intuitive interpretation. Main theme of SVM derives from the maximization of margin and finding the best possible margin.

Let us suppose that we have a linearly separable dataset consisting of four points shown in figure 3.1, there are different lines that will be separating these four data points in to two groups.

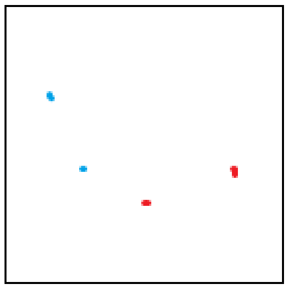


Figure 3.1 four different data points

The addition to the problem is to find the line that has some advantage over all the other possible lines separating these points. For example if we choose the line shown in figure 3.2, indeed this is not the perfect line for separation because the margin of error (yellow color) is very small.

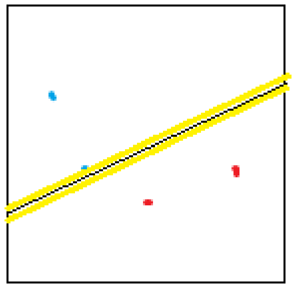


Figure 3.2 line separating data points

Few other possible lines that can be used for separation are shown in figure 3.3 with their margins of error.

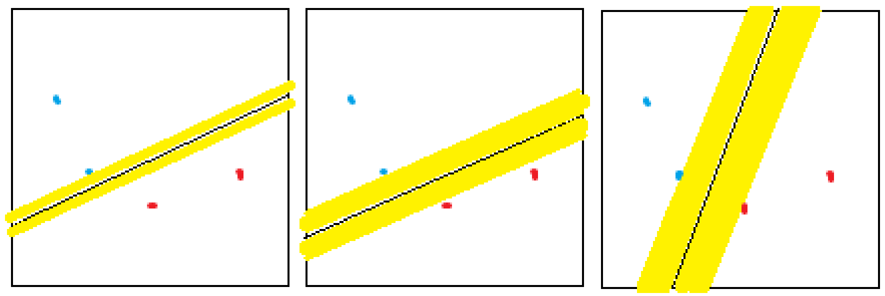


Figure 3.3 different lines separating four data points

As far as the training error is concerned it will be zero either we use any of the lines for separation of the data points, but the testing error using these lines will be different and other than zero. The logic behind SVM is to choose the line with widest (fat) margin, because chances are high that the new points will be on the correct side of the line if margin of error is kept wide.

So the first step to find a line with fat margin is to find a 'w' which can maximize the margin and classify the points correctly. For this let's assume that we have three points' dichotomies (2^3=8 dichotomies from growth function) with fat margins as in figure 3.4:

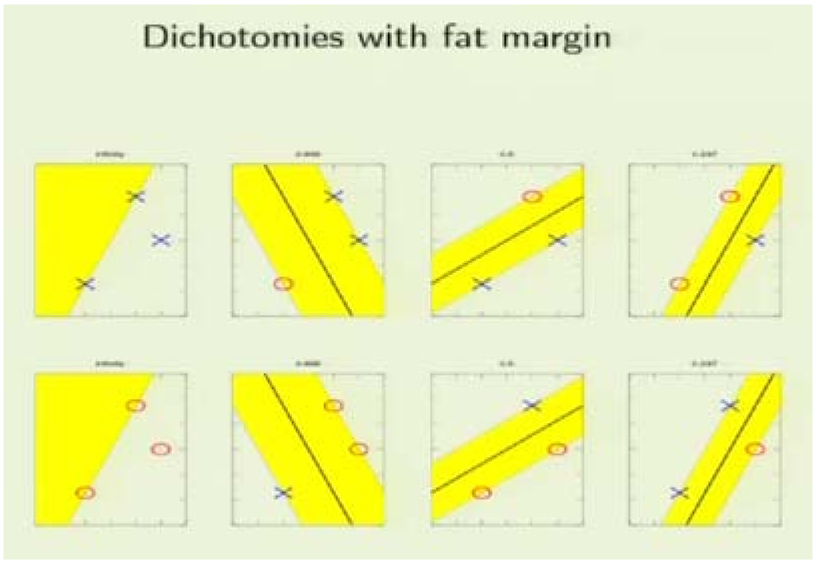


Figure 3.4 dichotomies with fat margin

If we have to use a classifier that has to choose a fat margin i-e a margin which should be at least of a certain width, so the dichotomies with lesser width than the required are not allowed. By requiring the margin to be at least something a restriction is made on the growth function. Fat margin apply fewer dichotomies possible, so we can say that if we are able to classify using fat margin we will require small VC dimension.

Margin is the distance from a plane to a point, let *xn* is the nearest data point to the line, and the line is given by the linear equation (3.1)

$$w^T x = 0 \qquad (3.1)$$

If the dimensions are high this line is actually referred to as plane. Now we require estimating the distance between the plane and the nearest point as this distance will be actually the margin.

Two preliminary technicalities are invoked here which will simplify the analysis latter on:

1). Normalize w.

$$|w^T x_n| = 1 \qquad (3.2)$$

2). Pull out *w0*.

Now the equation for the plane is:

$$w^T x + b = 0 \qquad (3.4)$$

Where$|w^T x + b| = 1$, and we have to estimate the distance between the point and the plane, the geometry for this is shown in figure 3.5:

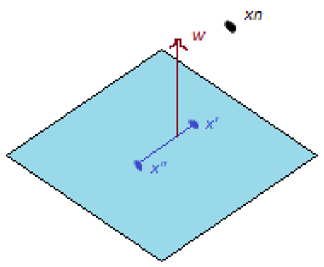


Figure 3.5 geometry of plane and line

The vector w is perpendicular to the plane in *X* space. If we take two points x' and x" on the plane these will satisfy the equation (3.4) of the plane and equations (3.5) and (3.6) are obtained.

$$w^T x' + b = 0 \qquad (3.5)$$

and

$$w^T x'' + b = 0 \qquad (3.6)$$

The difference between equations (3.5) and (3.6) provides:

$$w^T(x' - x'') = 0 \qquad (3.7)$$

So the vector *w* is orthogonal to the vector *(x' – x").*

If we take a single point *x* on the plane the distance between the point x_n and the plane *w* is the projection of x_n – *x* on *w*. For projection the unit vector in the direction of *w* is $\hat{w}$:

$$\hat{w} = \frac{w}{||w||} \qquad (3.8)$$

Now simply we have the dot product with unit vector which is the *distance* = | $\hat{w}^T$ $(x_n - x)$ |.

$$distance = \frac{1}{||w||} |w^T x_n - w^T x| = \frac{1}{||w||} |w^T x_n + b - w^T x - b| = \frac{1}{||w||} \qquad (3.9)$$

This distance is the margin and the optimization problem is to maximize the margin.

Maximize $\frac{1}{||w||}$ subject to $\min_{n=1,2,\ldots.N} |w^T x_n + b| = 1$

If all the points are classified correctly we can replace $|w^T x_n + b|$ by $y_n (w^T x_n + b)$ where y_n are the true labels. So our optimization problem now changes to:

Minimize $\frac{1}{2} w^T w$ subject to $y_n (w^T x_n + b) \geq 1$ for *n=1,2,……..,N* (3.10)

Until now the problem is constrained and it is the inequality constraint not the equality constraint. Lagrange is introduced here under the inequality constraint which is referred as KKT (Karush-Kuhn-Tucker) conditions.

3.4.1 Lagrange Formulation:

Minimize $L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{n=1}^{N} \alpha_n (y_n (w^T x_n + b) - 1)$ (3.11)

w.r.t *w* and *b* and maximize it w.r.t $\alpha_n \geq 0$

Differentiate equation (3.11) w.r.t to w and equate it to zero we get equation (3.12)

$$\nabla wL = w - \sum_{n=1}^{N} \alpha_n y_n x_n = 0 \qquad (3.12)$$

Similarly taking partial derivative w.r.t *b* we have:

$$\frac{\partial L}{\partial b} = - \sum_{n=1}^{N} \alpha_n y_n = 0 \qquad (3.13)$$

Substituting equations (3.12) and (3.13) in original Lagrange equation (3.11) we will get rid of *w* and *b* and left only with α.

$w \ = \ \sum_{n=1}^{N} \alpha_n \, y_n \, x_n$ From equation (3.12) and $\sum_{n=1}^{N} \alpha_n \, y_n \ = 0$ from equation (3.13) putting in equation (3.11):

$$L(\alpha) = \ \sum_{n=1}^{N} \alpha_n \ -\tfrac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m \, y_n \, y_m \, x_n x_m \qquad (3.14)$$

We are left with a simple quadratic from in terms of α. If we put back the constraints:

Maximize w.r.t α: subject to α ≥ 0 for *n=1,2,……..,N and*

$$\sum_{n=1}^{N} \alpha_n \, y_n \ = 0$$

3.4.2 The Solution: Quadratic Programming

The aim is to translate the objectives and constraints discussed above into the coefficients which can be passed to the quadratic programming easily and solved. If we take negative of the maximization problem defined in equation (3.14) we get the minimization problem for quadratic programming, so:

$$\underset{\alpha}{min} = \ \tfrac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m \, y_n \, y_m \, x_n x_m - \sum_{n=1}^{N} \alpha_n \qquad (3.15)$$

If we transform equation (3.15) in matrix form we get the following:

$$\underset{\alpha}{min} \tfrac{1}{2} \alpha^T \begin{bmatrix} y1y1x1x1 & y1y2x1x2 \cdots y1ynx1xn \\ y2y1x2x1 & y2y2x2x2 \cdots y2ynx2xn \\ & \cdots \\ yny1xnx1 & yny2xnx2 \cdots ynynxnxn \end{bmatrix} \alpha + (-1^T)\alpha \qquad (3.16)$$

Subject to $\quad y^T \alpha = 0 \quad$ (linear constraint)

(Lower bound) $0 \leq \alpha \geq \infty$ (Upper bound)

Quadratic programming returns α :

$$\alpha = \alpha 1, \ldots\ldots \alpha n \ ; \quad w \ = \ \sum_{n=1}^{N} \alpha_n \, y_n \, x_n$$

3.4.3 KKT Condition:

$$\alpha_n (y_n (w^T x_n + b) - 1) = 0 \quad (3.17)$$

$$\alpha_n \geq 0 \Rightarrow x_n \; is \; \boldsymbol{support\ vector}$$

3.5 Naïve Bayes Classifier:

A naïve bayes classifier (NBC) is a probability based classifier which applies the “Bayes theorem” and strong independence assumption for classification. NBC makes a strong independence assumption about the features which states that the presence of a certain feature cannot influence or is unrelated to the presence of another feature or vice versa. For example a fruit having ‘green colour’, ‘round shape’ and ‘10” diameter’ may be considered a watermelon. NBC takes each of these features independently to contribute to the probability that this fruit belongs to the class ‘watermelon’.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. “In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers.”[13] Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests. [14]

3.5.1 Bayes Decision Rule:

$$Posterior = likelihood * prior \div Evidence$$

$$P(W \mid A) = P(A \mid W) * P(W) \div P(A) \quad (3.18)$$

3.5.2 Probabilistic Classification:

MAP **(**Maximum A Posterior**)** Classification Rule:

Assign a to w^* if:

$$P(W = w^* \mid A = a) > P(W = w \mid A = a) \;\; and \, w^* \neq w, \, w = w_1, w_2 \cdots w_n \quad (3.19)$$

Apply Bayesian rule to convert**:**

$$P(W \mid A) = P(A \mid W) * P(W) \div P(A) \propto P(A \mid W) * P(W)$$
$$P(W \mid A) \propto P(A \mid W) * P(W) = P\left(A_{1,} A_{2,} \cdots \cdots A_n \mid W\right) * P(W) \quad (3.20)$$

Difficulty is learning the joint probability $P\left(A_{1,} A_{2,} \cdots \cdots A_n \mid W\right)$ so naïve bayes comes to rescue :

3.5.3 Naïve Bayes Classification:

NBC makes the assumption that all input features are independent so we have:

$$P(A_1, A_2, \cdots\cdots A_n \mid W) = P(A_1 \mid W)P(A_2 \mid W)\cdots\cdots\cdots P(A_n \mid W) \qquad (3.21)$$

When we blend the MAP classification rule equation (3.19) in equation (3.21) we get:

$$[P(a_1 \mid w^*)\cdots P(a_n \mid w^*)]P(w^*) > [P(a_1 \mid w)\cdots P(a_n \mid w)]P(w), \ldots\ldots$$

$$\ldots\ldots w^* \neq w,\, w = w_1, w_2 \cdots w_n \qquad (3.22)$$

Which tells that we assign a label by independently calculating all the probabilities of features provided, combining them and finding the greater value.

NBC Algorithm has a training phase, input to which is a training data and output of this phase is conditional probability tables. During testing phase an unknown instance is given and this is assigned a class label using the lookup tables generated during the training phase.

3.6 K-Nearest Neighbor Classifier:

KNN classifier is one of the simplest classifier in machine learning, yet it is widely used for classification in many problems. KNN classifies the new observation based on the 'majority voting' of its neighbors. The variable term K is the number of neighbors which are considered for classification and these are kept odd so that the decision can be made on majority votes and a confusing condition of equal votes does not occur. KNN Algorithm is described below:

Let's we have two classes and we have to classify a new observation as shown in figure 3.6:

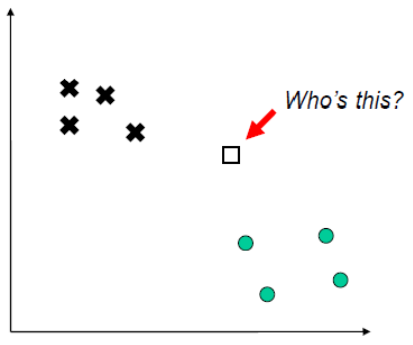


Figure 3.6 Two different classes

KNN algo will simply measure all the distances between the new observation and the sample space, then it will consider only K closest neighbors and will assign the label based on the majority, for example if k=3, and among the closest three neighbors two belong to class 'X' and one belong to class 'Y' the new observation will be labeled 'X'.

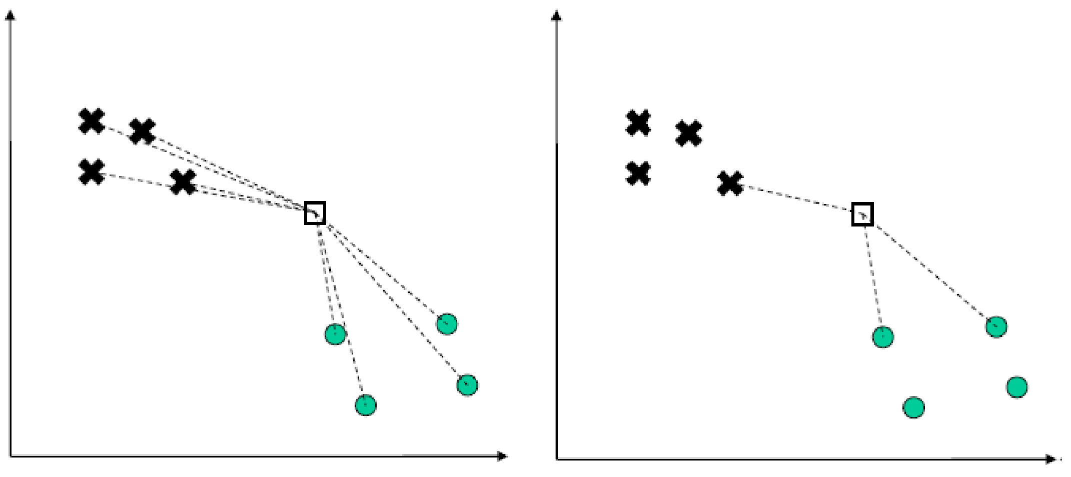


Figure 3.7 (a) *Figure 3.7 (b)*

Figure 3.7 (a) Shows calculation of distance from all the samples (b) shows comparison of distance to the closest k (3) points

The distance measured is the Euclidean distance, if we have two features *a* and *b,* new observation will have features *a'* and *b'* the Euclidean distance measured will be as follows:

$$euclidean\ distance = \sqrt{(a'-a)^2 + (b'-b)^2}$$

3.6.1 KNN Algorithm:

```
For every testing sample
    Calculate the Euclidean distance
    Extract the K closest samples
    Find common class label
    Assign class label
End
```

CHAPTER 4

Texture Features

Chapter 4 is all about the textural features based on gray level Coocurrence matrix (GLCM) used for classification of Brain MR Images. It contains all the mathematical equations for all the features used and a brief introduction about the GLCM.

4.1 Gray Level Coocurrence Matrix

“In statistical texture analysis, texture features are computed from the statistical distribution of observed combinations of intensities at specified positions relative to each other in the image. According to the number of intensity points (pixels) in each combination, statistics are classified into first-order, second-order and higher-order statistics.”[16]

The Gray Level Coocurrence Matrix (GLCM) method is a way of extracting second order statistical texture features. [16]

“A GLCM is a matrix where the number of rows and columns is equal to the number of gray levels, G, in the image. The matrix element $P(i, j \mid \Delta x, \Delta y)$ is the relative frequency with which two pixels, separated by a pixel distance $(\Delta x, \Delta y)$, occur within a given neighborhood, one with intensity *i* and the other with intensity *j*. One may also say that the matrix element $P(i, j \mid d, \theta)$ contains the second order statistical probability values for changes between gray levels *i* and *j* at a particular displacement distance *d* and at a particular angle (θ).”[16]

“Because a $G \times G$ matrix (or histogram array) must be accumulated for each sub-image/window and for each separation parameter set (d, θ), it is usually computationally necessary to restrict the (d, θ)-values to be tested to a limited number of values. Figure 4.1 illustrates the geometrical relationships of GLCM measurements made for four distances d ($d = \max\{|\Delta x|, |\Delta y|\}$) and angles of $\theta = 0, \pi/4, \pi/2$ and $3\pi/4$ radians under the assumption of angular symmetry.”[16]

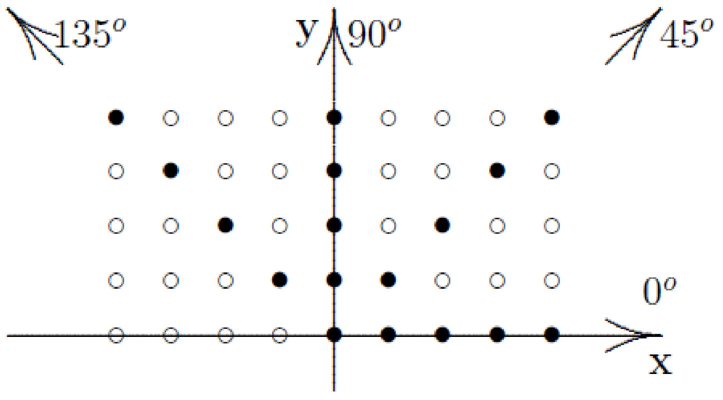


Figure 4.1 geometrical relationships of GLCM measurements made for four distances

A number of texture features may be extracted from the GLCM (see Haralick et al. 1973 [15], Conners et al. 1984 [17]). We use the following notation:

- ✓ G is the number of gray levels used.
- ✓ μ is the mean value of P.
- ✓ μx, μy, _x and _y are the means and standard deviations of Px and Py.
- ✓ Px(i) is the ith entry in the marginal-probability matrix obtained by summing the rows of P (i,j): [16]

P(i,j):

$$P_{x(i)=\sum_{j=0}^{G-1} P(i,j)}$$

$$P_{y(j)=\sum_{i=0}^{G-1} P(i,j)}$$

$$\mu_x = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} iP(i,j) = \sum_{i=0}^{G-1} iPx(i)$$

$$\mu_y = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} jP(i,j) = \sum_{i=0}^{G-1} jPy(j)$$

$$\sigma_x^2 = \sum_{j=0}^{G-1}(i-\mu_x)^2 \sum_{j=0}^{G-1} P(i,j) = \sum_{i=0}^{G-1}\left(P_x(i)-\mu_x(i)\right)^2$$

$$\sigma_y^2 = \sum_{j=0}^{G-1}\left(j-\mu_y\right)^2 \sum_{j=0}^{G-1} P(i,j) = \sum_{j=0}^{G-1}\left(P_y(j)-\mu_y(j)\right)^2$$

And

$$P_{x+y}(k) = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) \cdots\cdots\cdots i+j=k$$

For k=0,1,2…………2(G-1)

$$P_{x\text{-}y}(k) = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) \cdots\cdots\cdots \mid i+j \mid = k$$

For k=0,1,2…………(G-1)

4.2 Features computed:

The following features are used:

Homogeneity, Angular Second Moment (ASM):

$$ASM = \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} \{P(i,j)\}^2$$

ASM is a measure of homogeneity of an image. A homogeneous scene will contain only a few gray levels, giving a GLCM with only a few but relatively high values of P(i, j). Thus, the sum of squares will be high.

Contrast:

$$Contrast = \sum_{n=0}^{G-1} n^2 \left\{ \sum_{i=1}^{G} \sum_{j=1}^{G} P(i,j) \right\}, \qquad |i-j| = n$$

This measure of contrast or local intensity variation will favour contributions from P(i, j) away from the diagonal, i.e. i≠ j. Returns a measure of the intensity contrast between a pixel and its neighbor over the whole image.

Range = [0 (size (GLCM, 1)-1) ^2]

Contrast is 0 for a constant image.

Local Homogeneity, Inverse Difference Moment (IDM):

$$IDM = \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} \frac{1}{1+(i-j)^2} P(i,j)$$

IDM is also influenced by the homogeneity of the image. Because of the weighting factor (1+(i−j)2)−1 IDM will get small contributions from inhomogeneous areas (i≠ j). The result is a low IDM value for inhomogeneous images, and a relatively higher value for homogeneous images. Returns a value that measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal.

Range = [0 1]

Homogeneity is 1 for a diagonal GLCM.

Entropy:

$$Entropy = -\sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) \times log\big(P(i,j)\big)$$

Inhomogeneous scenes have low first order entropy, while a homogeneous scene has high entropy.

Correlation:

$$Correlation = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \frac{\{i \times j\} \times P(i,j) - \{\mu_x \times \mu_y\}}{\sigma_x \times \sigma_y}$$

Correlation is a measure of gray level linear dependence between the pixels at the specified positions relative to each other. Returns a measure of how correlated a pixel is to its neighbor over the whole image.
Range = [-1 1]

Correlation is 1 or -1 for a perfectly positively or negatively correlated image. Correlation is Nan for a constant image.

Energy:

$$Energy = \sum_{i,j} p(i,j)^2$$

Returns the sum of squared elements in the GLCM.

Range = [0 1]

Energy is 1 for a constant image.

Sum of Squares, Variance:

$$Variance = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} (i-\mu)^2 P(i,j)$$

This feature puts relatively high weights on the elements that differ from the average value of P (i, j).

Sum Average:

$$Average = \sum_{i=0}^{2G-2} iP_{x+y}\,(i)$$

Sum Entropy:

$$sum\ entropy = -\sum_{i=0}^{2G-2} P_{x+y}\,(i) log\left(P_{x+y}(i)\right)$$

Difference Entropy:

$$difference\ entropy = -\sum_{i=0}^{G-1} P_{x+y}\,(i) log\left(P_{x+y}(i)\right)$$

Inertia:

$$Inertia = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1}\{i-j\}^2 \times P(i,j)$$

Cluster Shade:

$$Shade = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1}\left\{i+j-\mu_x-\mu_y\right\}^3 \times P(i,j)$$

Cluster Prominence:

$$Prominence = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1}\left\{i+j-\mu_x-\mu_y\right\}^4 \times P(i,j)$$

Autocorrelation:

If we take the cross-correlation (measure of similarity of two waveforms) of a signal with itself we get the autocorrelation. “It is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal which has been buried under noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies”.[18]

Information measure of correlation

$$Information\,measure\,of\,correlation\;1(IMC1) = \frac{HXY-HXY1}{max\{HX,HY\}}$$

$$Information\,measure\,of\,correlation\;2(IMC2) = \sqrt{(1-exp[-2.0(HXY2-HXY)])}$$

$$HXY = -\sum_i\sum_j g_{ij}log_2 g_{ij} \quad where\,and\,HY\,are\,entropies\,of\;g_x and g_y$$

$$HXY1 = -\sum_i\sum_j g_{ij}log_2\{g_x(i)\,g_y(j)\}$$

$$HXY2 = -\sum_i\sum_j g_x(i)g_y(j)\}log_2\{g_x(i)\,g_y(j)\}$$

Maximum probability

$$Maximum\,probability = {}^{MAX}_{i,j}P(i,j)$$

Inverse difference normalized
Inverse difference moment normalized

These are all the twenty two textural features which were worked out and among all these few proved to be the best features for classification of Brain MR Images. PSO selected among these features the best features, so that the complexity and computational time to classify the images is reduced.

CHAPTER 5

Implementation and Evaluation

This chapter describes the implementation details and performance evaluation of the proposed algorithms. Previous Chapters contain the basic details of the hybrid techniques which are implemented and evaluated. All the hybrid techniques PSO-SVM, PSO-NB and PSO-KNN are evaluated against the achieved classification accuracy for unseen data using least number of textural features. In our proposed techniques PSO serves the purpose of reducing the number of features i –e feature selection whereas different classifiers i-e SVM, NB and KNN server as the fitness function for PSO.

Selected Features are passed to the different classifiers for classification of T1 Brain MRI and the classifiers classify normal images from the tumor images. Testing error is computed which is passed to the PSO and it uses the testing error as its fitness function.

An overall system used is shown in figure 5.1:

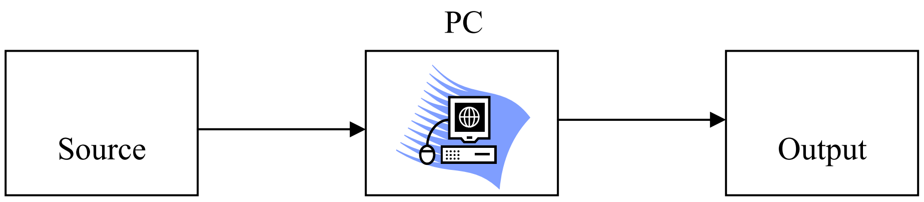


Figure 5.1 overall system

The source is the dataset of T1 Brain MRI, a total of 362 images (217 x 181) are used among which 181 are the normal brain images and 181 are the tumor images. This dataset is taken from the Internet Brain Segmentation Repository (IBSR) which is quiet reliable data source for our purpose. There are hundreds of publications using data provided by IBSR (listed on their official website).

All the work is carried on Intel(R) (Core(TM) i5-2410M CPU @ 2.30 GHz, with 4.00 GB of RAM) PC. MATLAB 7.8.0 (R2009a) is used for simulation purpose.

Output of our system is the reduced feature subset with high classification accuracy. Different feature subsets with different classification accuracy are obtained.

Detailed flow chart of all the procedure followed is shown in figure 5.2.

Details of different constants initialized in PSO are given in last section (Results).

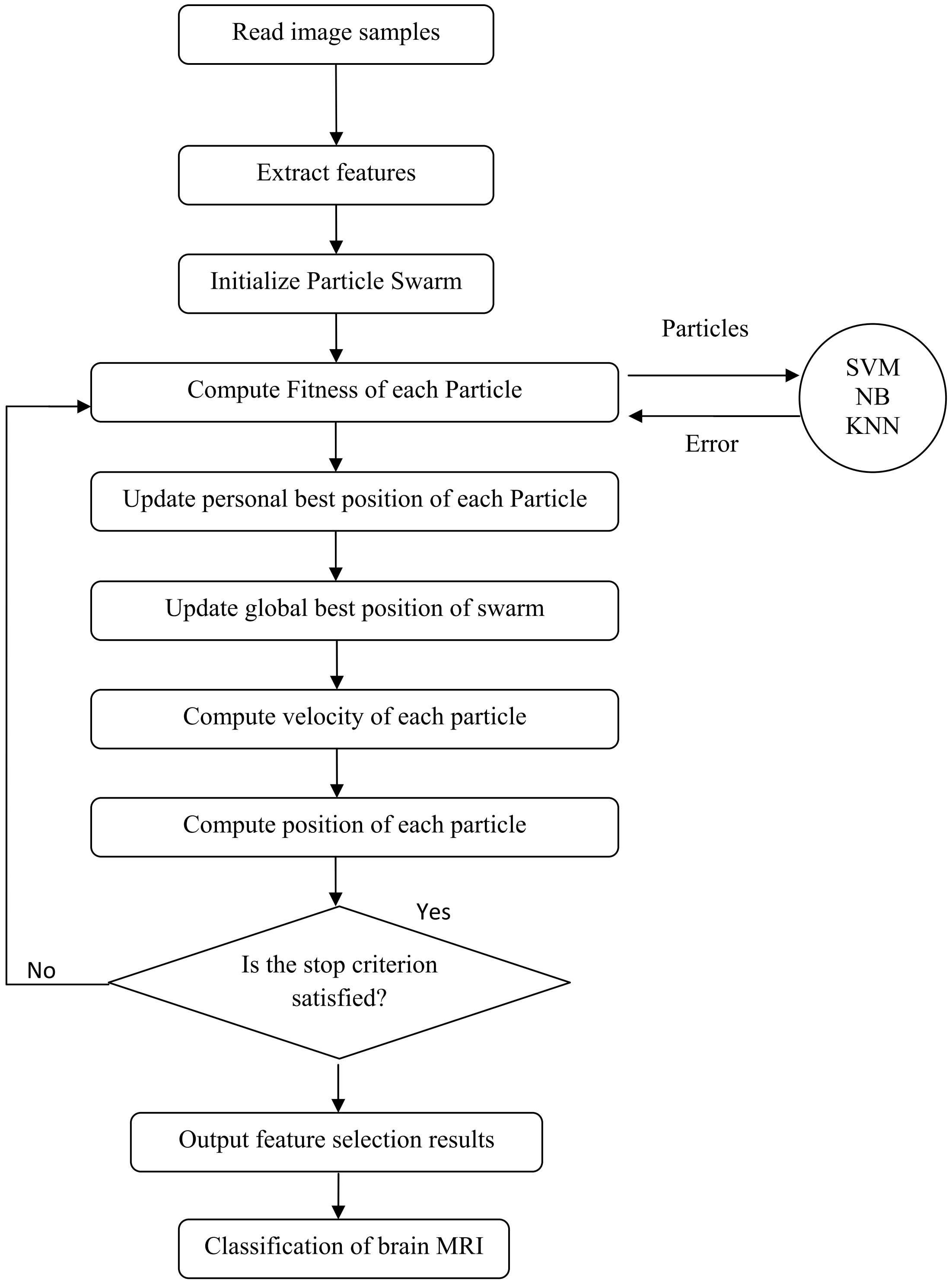

Read image samples
Extract features
Initialize Particle Swarm
Particles
Compute Fitness of each Particle
SVM
NB
KNN
Error
Update personal best position of each Particle
Update global best position of swarm
Compute velocity of each particle
Compute position of each particle
Yes
No
Is the stop criterion satisfied?
Output feature selection results
Classification of brain MRI

Figure 5.2 Flow chart of feature selection for Brain MRI scan

5.1 Working of Discrete Binary PSO:

The procedure is described as follows:

1) Initialize all the PSO constants as well as the position and velocity matrices of the swarm.
2) Evaluate the fitness of each particle by sending it to a classifier for classification.
3) After evaluation compare the error obtained by the particular particle by it's previously achieved error, if the new error is less than the older than replace the position of the particle with it's new position(update personal best position).
4) Now compare the fitness of the particle with entire swarm, if the achieved error is minimum then replace the global best position with this position of the particle.
5) Update the velocity of each particle.
6) Update position of each particle.
7) Exit if the desired efficiency with minimum number of features is achieved otherwise goto step 2.

5.2 PSO-SVM

In PSO-SVM (particle swarm optimization- support vector machine) the PSO is used for feature selection of brain images and SVM is used as an evaluation of the features. Features are selected based on the classification accuracy achieved by SVM.

The block diagram of the process is shown below in figure 5.3:

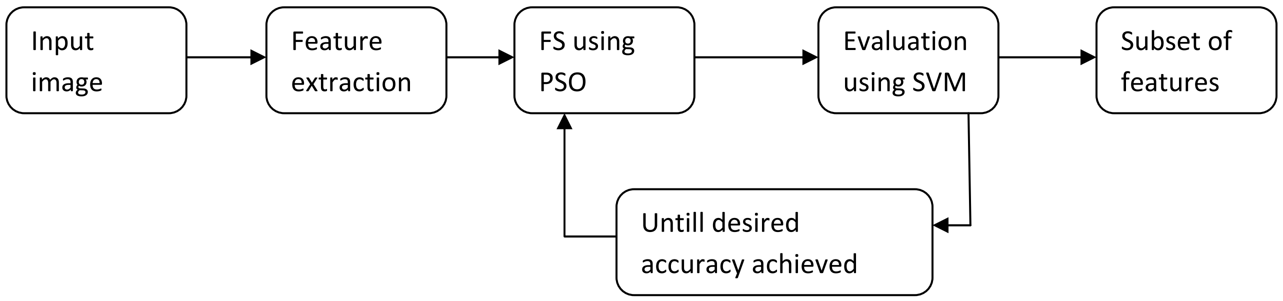


Figure 5.3: PSO-SVM

The first step is to input MR brain image to the system, next different features are extracted from the image. These all extracted features are passed on to PSO which selects a subset of features and passes it to SVM for evaluation. SVM evaluates the provided subset of features by classifying the test brain images in to two classes' i-e 'normal image' or 'tumor image'. Until a desired level of accuracy is achieved the process of selecting new subset of features and evaluating these using SVM is repeated.

A total of ten images are used to train the SVM, among these ten images five are normal images and five are tumor images. The features used to train SVM are only those selected by PSO. After training SVM a set of ten unseen images are classified using SVM, again among these unseen test images five are normal images and five are tumor images. The testing accuracy is computed on the basis of classification done by SVM (using features provided by PSO). This process of selecting new subset of features and evaluating these on classification performance by SVM is repeated until a desired accuracy is achieved. When the desired level of accuracy is achieved with minimum number of features the process stops. After performing classification testing error is computed and this error is sent back to PSO which uses it as an objective function. So the objective function of PSO is to minimize the testing error of the classifier.

5.3 PSO-NBC

This process makes use of PSO-NBC(particle swarm optimization- Naïve Bayes Classifier), here again PSO is used to select features and NBC servers as an evaluator by classifying the images using subset of features provided by PSO.

Block diagram of PSO-NBC is shown below in figure 5.4:

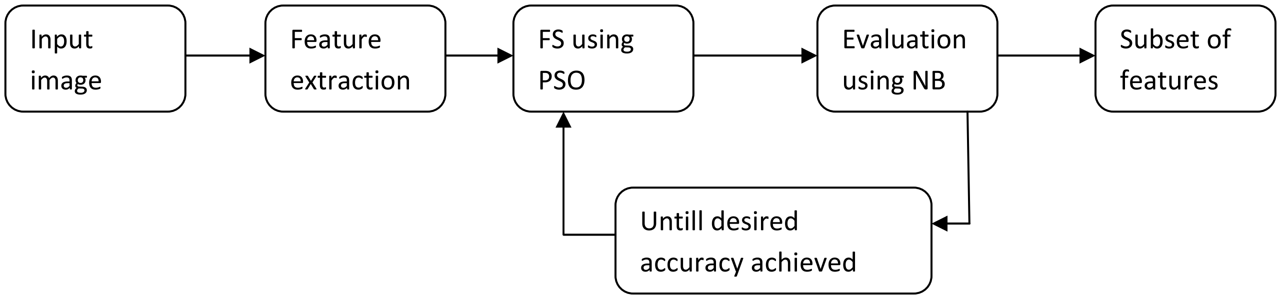


Figure 5.4: PSO-NBC

The whole process of PSO-NBC is executed in a similar manner as PSO-SVM, the only difference is the use of NBC as an evaluator instead of PSO. Classification is performed using NBC and testing error computed is feeded as an objective function to be optimized by PSO.

5.4 PSO-KNN

PSO-KNN (particle swarm optimization- k nearest neighbor) also makes use of PSO for feature selection and KNN is used as a classifier. Testing accuracy is computed and error is given as a feedback to PSO.

The block diagram of PSO-KNN is shown below in figure 5.5:

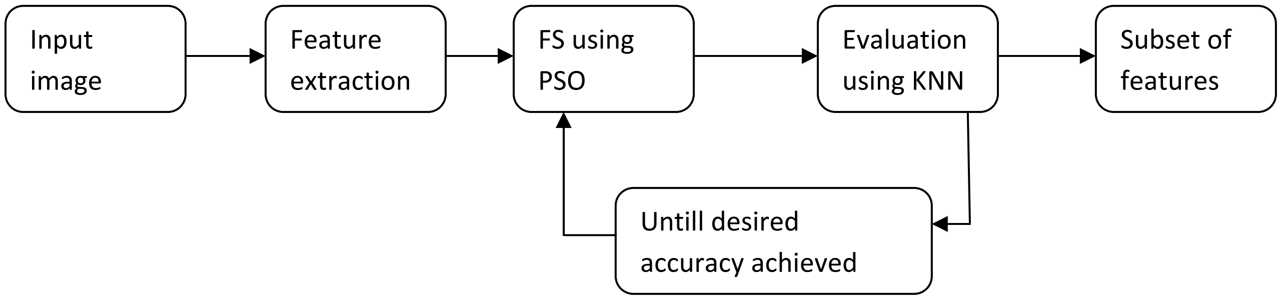


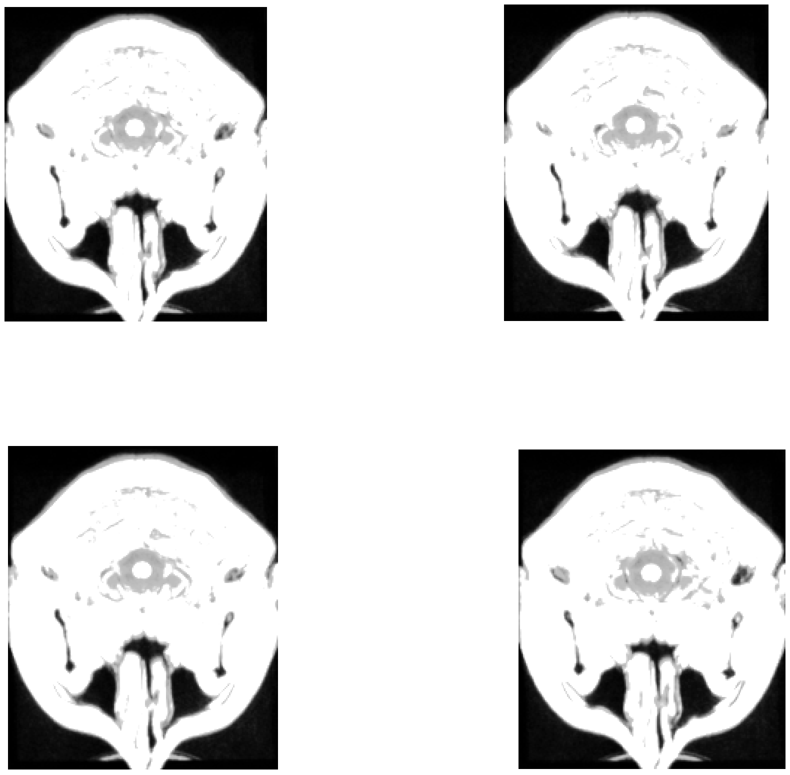
Figure 5.5: PSO-KNN

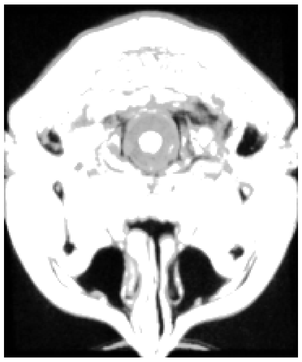
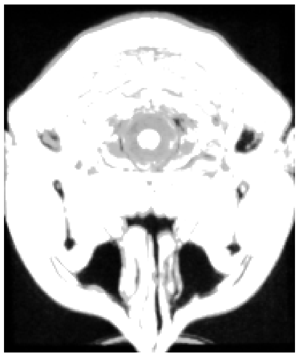
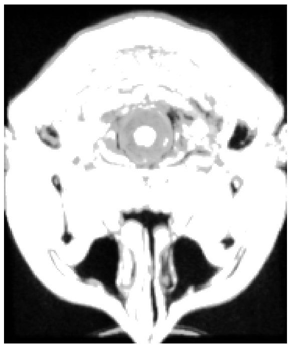
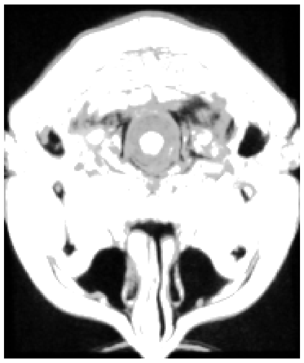
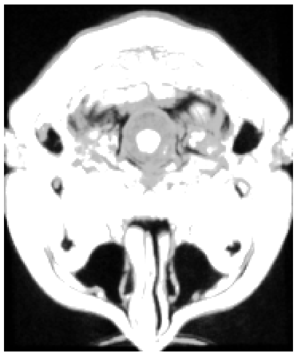
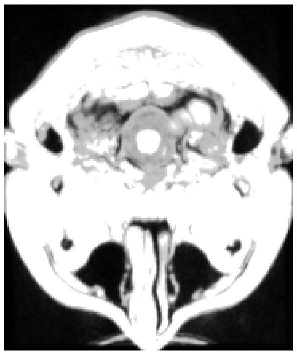
Rest of all the process in PSO-KNN is similar to PSO-SVM and PSO-NBC.

5.5 Results:

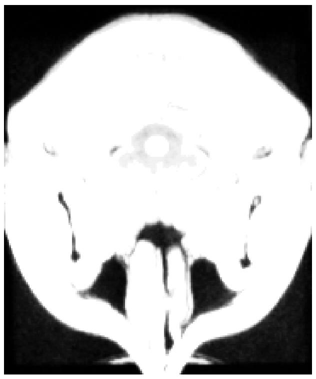
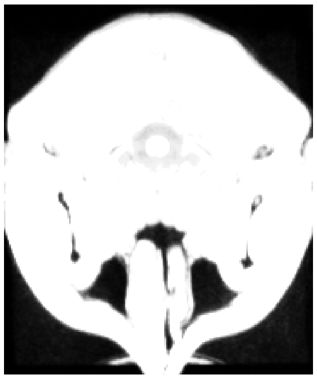
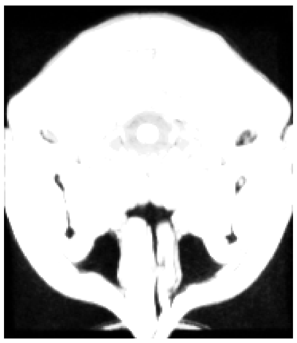
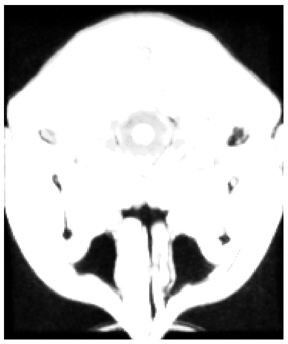
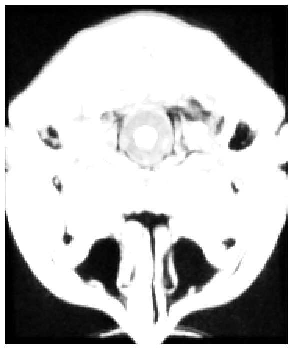
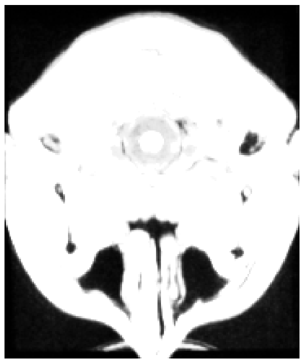
Few Brain MRI snapshots used for evaluation are shown below:

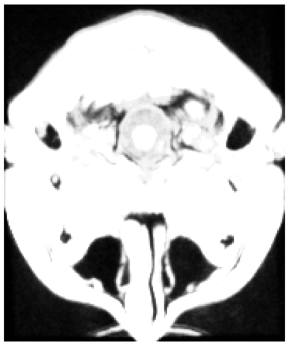
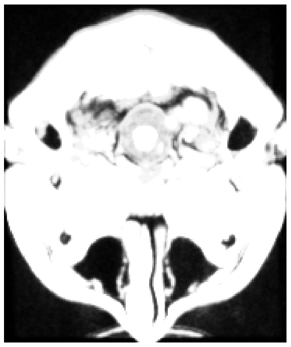
5.5.1 Normal Brain MRI (T1 weighted):



5.5.2 Brain with Lesion (Tumor) MRI (T1 weighted):

*****_Data from Internet Brain Segmentation Repository_

PSO initialization Constants:

C1= 2

C2=2

W=0.5 (inertial constant)

Swarm size=1000

PSO-SVM

Sr. No.	Efficiency	Number of features	Features
1	100 % Unseen Data	six	Cluster prominence, Dissimilarity, Entropy, Difference variance, Inverse difference normalized , Inverse difference moment normalized
2	100 % Unseen Data	six	Contrast, Energy, Sum average, Difference variance, Information measure of Correlation1, Information measure of Correlation2, Inverse difference normalized.
3	90 % Unseen Data	six	Autocorrelation, Dissimilarity, Sum of squares: Variance, Sum entropy, Information measure of correlation1, Inverse difference moment normalized
4	90 % Unseen Data	seven	Correlation, Dissimilarity, Maximum probability, Information measure of correlation1, Information measure of correlation2, Inverse difference normalized, Inverse difference moment normalized

Table 5.1 PSO-SVM Results

PSO-KNN

Sr. No.	Efficiency	Number of features	Features
1	100 % Unseen Data	five	Contrast, Energy, Difference Entropy, Homogeneity, Inverse difference normalized.
2	90 % Unseen Data	five	Contrast ,Correlation matlab, Difference entropy, Information measure of correlation1, Inverse difference moment normalized
3	80 % Unseen Data	five	Correlation matlab, Correlation paper, Energy Difference variance, Inverse difference moment normalized
4	80 % Unseen Data	six	Correlation matlab, Correlation paper, Dissimilarity, Maximum probability, Difference variance, Difference entropy

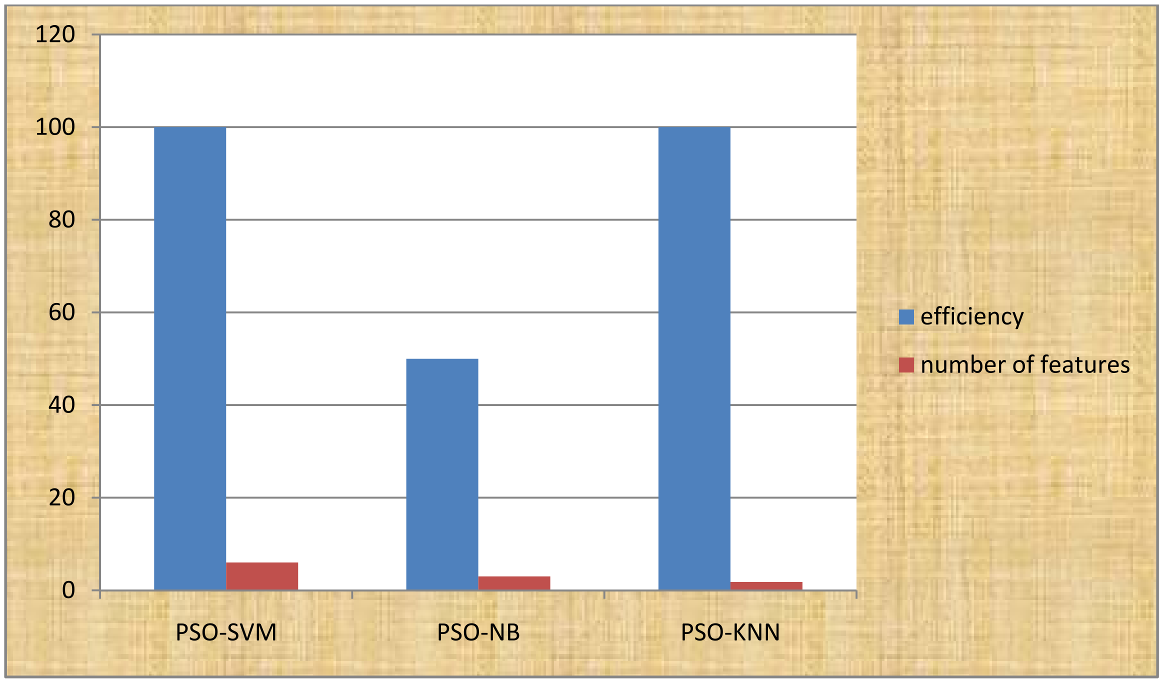
Table 5.2 PSO-KNN Results

PSO-NBC

Sr. No.	**Efficiency**	**Number of features**	**Features**
1	50% Unseen Data	three	Homogeneity, Sum of squares: Variance, Difference entropy.
2	90% Seen Data	four	Homogeneity, Sum of squares: Variance, Difference entropy, Inverse difference normalized.

Table 5.1 PSO-NBC Results

Number Of Features VS maximum Efficiency for unseen data



References

1) *Central Brain Tumor Registry of the United States (CBTRUS). Fact Sheet. Available at: http://www.cbtrus.org. Accessed April 4, 2011.*
2) *Viner J. Brain Tumors. University of California, San Francisco Department of Neurosurgery. Available at: http://nursing.ucsfmedicalcenter.org/education/classMaterial/34_2.pdf. Accessed April 6, 2011.*
3) *Andreasen NC, ed. Brain Imaging: Applications in Psychiatry. Washington, DC: American Psychiatric Press; 1989.*
4) *Debatin JF, McKinnon GC, eds. Ultrafast MRI: Techniques and Applications. Berlin, Germany: Springer-Verlag; 1998.*
5) *Westbrook C, Roth CK, Talbot J, eds. MRI in Practice. 3rd ed. Oxford, United Kingdom: Wiley-Blackwell; 2005.*
6) *Westbrook C. MRI at a Glance. Hoboken, NJ: Wiley-Blackwell; 2010.*
7) *Kennedy, J. and R.C. Eberhart, 1995. Particle Swarm Optimization. Proc. IEEE, International Conference on Neural Networks. Piscataway.*
8) *Frans van den Bergh. An Analysis of Particle Swarm Optimizers. PhD thesis, University of Pretoria,2001.*
9) *James Kennedy, Russell Eberhart, and Yuhui Shi. Swarm Intelligence. Morgan Kaufmann, 2001.*
10) *E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization, 2002.*
11) *James Blondin, Particle Swarm Optimization: A Tutorial, September 4, 2009*
12) *Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. Proceedings of the IEEE International Conference on Evolutionary Computation, pages 69–73, 1998.*
13) *^ Zhang, Harry. "The Optimality of Naive Bayes". FLAIRS2004 conference.*
14) *^ Caruana, R.; Niculescu-Mizil, A. (2006). "An empirical comparison of supervised learning algorithms". Proceedings of the 23rd international conference on Machine learning. CiteSeerX:10.1.1.122.5901.*
15) *1. R. M. Haralick, K. Shanmugam, and I. Dinstein, Textural Features of Image Classification, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-3, no. 6, Nov. 1973*
16) *Fritz Albregtsen, Statistical Texture Measures Computed from Gray Level Coocurrence Matrices, Image Processing Laboratory Department of Informatics University of Oslo,November 5, 2008.*
17) *R.W. Conners, M.M. Trivedi, and C.A. Harlow,"Segmentation of a High-Resolution Urban Scene Using Texture Operators",Computer Vision, Graphics, and Image Processing, Vol. 25, pp. 273-310, 1984.*
18) *http://en.wikipedia.org/wiki/Autocorrelation*
19) *Linyi Li,"feature selection for residential area recognition in high resolution images based on particle swarm optimization",school of remote sensing and information engineering Wuhan university Wuhan, China*

Annex I:

MATLAB CODE :

```
clear all
close all
clc

%% Load brain images with tumour/lesion (MRI T1)
[imaVOL,scaninfo] = loadminc('G:\Ms-69\thesis\New folder\lesion
subjects\t1_icbm_normal_1mm_pn0_rf20.mnc');

%% Load normal brain images (MRI T1)
[imaVOLL,scaninfoo] = loadminc('G:\Ms-69\thesis\New folder\lesion
subjects\t1_ai_msles2_1mm_pn0_rf20.mnc');

x1=imaVOL(:,:,1:10); %read first 10 normal images
x2=imaVOLL(:,:,1:10);%read first 10 abnormal images
x(:,:,1:10)=x2; %concatinate both type of images in one matrix x
x(:,:,11:20)=x1;

s1='tumour';%labeling
s2='normal';
S1={s1;s1;s1;s1;s1;s1;s1;s1;s1;s1;s2;s2;s2;s2;s2;s2;s2;s2;s2;s2};% true
labels

f_mat=zeros(20,22); % initialize feature matrix (20 data samples 22 features)

for v=1:20

glcm = graycomatrix(x(:,:,v),'Offset',[1 0]);% calculate gray level co
occurance matrix for all images
[out(v)] = GLCM_Features1(glcm,0);%computing features
end

for v=1:20 %extracting all the features in 20 * 22 matrix(20 images,22
features (all data))
f_mat(v,1) = getfield(out,{1,v}, 'autoc');
f_mat(v,2) = getfield(out,{1,v}, 'contr');
f_mat(v,3) = getfield(out, {1,v},'corrm');
f_mat(v,4) = getfield(out,{1,v}, 'corrp');
f_mat(v,5) = getfield(out,{1,v}, 'cprom');
f_mat(v,6) = getfield(out,{1,v}, 'cshad');
f_mat(v,7) = getfield(out,{1,v}, 'dissi');
f_mat(v,8) = getfield(out,{1,v}, 'energ');
f_mat(v,9) = getfield(out,{1,v}, 'entro');
f_mat(v,10) = getfield(out,{1,v}, 'homom');
f_mat(v,11) = getfield(out, {1,v},'homop');
f_mat(v,12) = getfield(out, {1,v},'maxpr');
f_mat(v,13) = getfield(out,{1,v}, 'sosvh');
f_mat(v,14) = getfield(out,{1,v}, 'savgh');
f_mat(v,15) = getfield(out,{1,v}, 'svarh');
f_mat(v,16) = getfield(out,{1,v}, 'senth');
f_mat(v,17) = getfield(out,{1,v}, 'dvarh');
f_mat(v,18) = getfield(out,{1,v}, 'denth');
f_mat(v,19) = getfield(out,{1,v}, 'inf1h');
f_mat(v,20) = getfield(out,{1,v}, 'inf2h');
f_mat(v,21) = getfield(out,{1,v}, 'indnc');
f_mat(v,22) = getfield(out,{1,v}, 'idmnc');
end
energy=zeros(1,20);
for h=1:20
[cll,clh,chl,chh]=dwt2(x(:,:,h),'db1');
  energy(h) = sum(abs(cll(:)).^2);
end
f_mat=[f_mat energy'];
%% PSO

 w  = 0.5;    % Inertial Constant
    C1 = 2;    % Constant 1
    C2 = 2;    % Constant 2

Np=1000; %number of particles



posi=round(rand(Np,23)); %generating binary numbers (particles)
pbest=posi;%initializing personal best values randomly



V=rand(Np,23);%initialize velocities of particles
Vmax=1;%maximum velocity

e=ones(1,Np);
ee=ones(1,Np);

for ii=1:100 %total number of iterations
for j=1:Np
    E=my_svm(posi(j,:),f_mat);
    %E=my_naive(posi(j,:),f_mat);
    %E=my_knn(posi(j,:),f_mat);
    ee(j)=E;
     if ee(j)<e(j)%if error improved update personal best
         pbest(j,:)=posi(j,:);
     else
         pbest(j,:)=pbest(j,:);
     end
end
e=ee;

indx=0;

for k=1:Np %finding the global best among all the particles
    if ee(k)==min(ee)
        indx=k;
        break
%     else
%         indx=indx;
    end
end
```

```
gbest=posi(indx,:);%saving global best in gbest

if ee(indx)==0 && sum(gbest)<=3 %if error less than 10% with only 4 features
STOP
    break
end



for p=1:Np
            for i=1:23
V(p,i)  = w*(V(p,i) + rand * C1 * (pbest(p,i)-posi(p,i)) + rand * C2 *
(gbest(i) - posi(p,i)));%update velocities of particles

if V(p,i)<=Vmax && V(p,i)>=-Vmax %velocity clamping
    V(p,i)=V(p,i);
elseif V(p,i)<=-Vmax
    V(p,i)=-Vmax;
else
    V(p,i)=Vmax;
end

SV=1/(1+exp(-V(p,i)));%update position of particles
if rand<SV
    posi(p,i)=1;
else
    posi(p,i)=0;
end

            end
end
end


--------------------------------------------------------------------------------


  function error=my_svm(data,f_mat)
X=zeros(20,1);
for i=1:23
    if data(1,i)==0
        X=X;
    else
        X=[X f_mat(:,i)];
    end
end
[R C]=size(X);
X=X(:,2:C);% left only with features with particle position '1'

xx=[X(1:5,:);X(11:15,:)];
s1='tumour';
s2='normal';
S={s1;s1;s1;s1;s1;s2;s2;s2;s2;s2};
%S1={s1;s1;s1;s2;s2;s2};
```

```
svmstruct=svmtrain(xx,S);% training SVM with 10 images 5 normal and 5
abnormal
clas=svmclassify(svmstruct,[X(6:10,:);X(16:20,:)]);% testing SVM for unseen
data
e=strcmp(clas,S);% finding errors
error=(10-sum(e))/10;%computing error
end
```

```
function error=my_naive(data,f_mat)
X=zeros(20,1);
for i=1:23
    if data(1,i)==0
        X=X;
    else
        X=[X f_mat(:,i)];
    end
end
[R C]=size(X);
X=X(:,2:C);

xx=[X(1:5,:);X(11:15,:)];
s1='tumour';
s2='normal';
S={s1;s1;s1;s1;s1;s2;s2;s2;s2;s2};

O1 = NaiveBayes.fit(xx,S);
       C1 = O1.predict([X(6:10,:);X(16:20,:)]);
       e=strcmp(C1,S);
error=(10-sum(e))/10;
end
```

```
function error=my_knn(data,f_mat)

X=zeros(20,1);
k=5;
for i=1:23
    if data(1,i)==0
        X=X;
    else
        X=[X f_mat(:,i)];
    end
end
[R C]=size(X);
X=X(:,2:C);

xx=[X(1:5,:);X(11:15,:)];
s1='tumour';
s2='normal';
S={s1;s1;s1;s1;s1;s2;s2;s2;s2;s2};
Class = knnclassify([X(6:10,:);X(16:20,:)], xx, S, k);
 e=strcmp(Class,S);
error=(10-sum(e))/10;
```

```
end
------------------------------------------------------------------------------
```

```
function [out] = GLCM_Features1(glcmin,pairs)
%
% GLCM_Features1 helps to calculate the features from the different GLCMs
% that are input to the function. The GLCMs are stored in a i x j x n
% matrix, where n is the number of GLCMs calculated usually due to the
% different orientation and displacements used in the algorithm. Usually
% the values i and j are equal to 'NumLevels' parameter of the GLCM
% computing function graycomatrix(). Note that matlab quantization values
% belong to the set {1,..., NumLevels} and not from {0,...,(NumLevels-1)}
% as provided in some references
% http://www.mathworks.com/access/helpdesk/help/toolbox/images/graycomatrix
% .html
%
% Although there is a function graycoprops() in Matlab Image Processing
% Toolbox that computes four parameters Contrast, Correlation, Energy,
% and Homogeneity. The paper by Haralick suggests a few more parameters
% that are also computed here. The code is not fully vectorized and hence
% is not an efficient implementation but it is easy to add new features
% based on the GLCM using this code. Takes care of 3 dimensional glcms
% (multiple glcms in a single 3D array)
%
% If you find that the values obtained are different from what you expect
% or if you think there is a different formula that needs to be used
% from the ones used in this code please let me know.
% A few questions which I have are listed in the link
% http://www.mathworks.com/matlabcentral/newsreader/view_thread/239608
%
% I plan to submit a vectorized version of the code later and provide
% updates based on replies to the above link and this initial code.
%
% Features computed
% Autocorrelation: [2]                      (out.autoc)
% Contrast: matlab/[1,2]                    (out.contr)
% Correlation: matlab                       (out.corrm)
% Correlation: [1,2]                        (out.corrp)
% Cluster Prominence: [2]                   (out.cprom)
% Cluster Shade: [2]                        (out.cshad)
% Dissimilarity: [2]                        (out.dissi)
% Energy: matlab / [1,2]                    (out.energ)
% Entropy: [2]                              (out.entro)
% Homogeneity: matlab                       (out.homom)
% Homogeneity: [2]                          (out.homop)
% Maximum probability: [2]                  (out.maxpr)
% Sum of sqaures: Variance [1]              (out.sosvh)
% Sum average [1]                           (out.savgh)
% Sum variance [1]                          (out.svarh)
% Sum entropy [1]                           (out.senth)
% Difference variance [1]                   (out.dvarh)
% Difference entropy [1]                    (out.denth)
% Information measure of correlation1 [1]   (out.inf1h)
% Informaiton measure of correlation2 [1]   (out.inf2h)
% Inverse difference (INV) is homom [3]     (out.homom)
% Inverse difference normalized (INN) [3]   (out.indnc)
% Inverse difference moment normalized [3]  (out.idmnc)
%
% The maximal correlation coefficient was not calculated due to
% computational instability
% http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
%
% Formulae from MATLAB site (some look different from
% the paper by Haralick but are equivalent and give same results)
% Example formulae:
% Contrast = sum_i(sum_j(  (i-j)^2 * p(i,j) ) ) (same in matlab/paper)
% Correlation = sum_i( sum_j( (i - u_i)(j - u_j)p(i,j)/(s_i.s_j) ) ) (m)
% Correlation = sum_i( sum_j( ((ij)p(i,j) - u_x.u_y) / (s_x.s_y) ) ) (p[2])
% Energy = sum_i( sum_j( p(i,j)^2 ) )           (same in matlab/paper)
% Homogeneity = sum_i( sum_j( p(i,j) / (1 + |i-j|) ) ) (as in matlab)
% Homogeneity = sum_i( sum_j( p(i,j) / (1 + (i-j)^2) ) ) (as in paper)
%
% Where:
% u_i = u_x = sum_i( sum_j( i.p(i,j) ) ) (in paper [2])
% u_j = u_y = sum_i( sum_j( j.p(i,j) ) ) (in paper [2])
% s_i = s_x = sum_i( sum_j( (i - u_x)^2.p(i,j) ) ) (in paper [2])
% s_j = s_y = sum_i( sum_j( (j - u_y)^2.p(i,j) ) ) (in paper [2])
%
%
% Normalize the glcm:
% Compute the sum of all the values in each glcm in the array and divide
% each element by it sum
%
% Haralick uses 'Symmetric' = true in computing the glcm
% There is no Symmetric flag in the Matlab version I use hence
% I add the diagonally opposite pairs to obtain the Haralick glcm
% Here it is assumed that the diagonally opposite orientations are paired
% one after the other in the matrix
% If the above assumption is true with respect to the input glcm then
% setting the flag 'pairs' to 1 will compute the final glcms that would
result
% by setting 'Symmetric' to true. If your glcm is computed using the
% Matlab version with 'Symmetric' flag you can set the flag 'pairs' to 0
%
% References:
% 1. R. M. Haralick, K. Shanmugam, and I. Dinstein, Textural Features of
% Image Classification, IEEE Transactions on Systems, Man and Cybernetics,
% vol. SMC-3, no. 6, Nov. 1973
% 2. L. Soh and C. Tsatsoulis, Texture Analysis of SAR Sea Ice Imagery
% Using Gray Level Co-Occurrence Matrices, IEEE Transactions on Geoscience
% and Remote Sensing, vol. 37, no. 2, March 1999.
% 3. D A. Clausi, An analysis of co-occurrence texture statistics as a
% function of grey level quantization, Can. J. Remote Sensing, vol. 28, no.
% 1, pp. 45-62, 2002
% 4. http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
%
%
% Example:
%
```

```
% Usage is similar to graycoprops() but needs extra parameter 'pairs' apart
% from the GLCM as input
% I = imread('circuit.tif');
% GLCM2 = graycomatrix(I,'Offset',[2 0;0 2]);
% stats = GLCM_features1(GLCM2,0)
% The output is a structure containing all the parameters for the different
% GLCMs
%
% [Avinash Uppuluri: avinash_uv@yahoo.com: Last modified: 11/20/08]

% If 'pairs' not entered: set pairs to 0
if ((nargin > 2) || (nargin == 0))
   error('Too many or too few input arguments. Enter GLCM and pairs.');
elseif ( (nargin == 2) )
    if ((size(glcmin,1) <= 1) || (size(glcmin,2) <= 1))
       error('The GLCM should be a 2-D or 3-D matrix.');
    elseif ( size(glcmin,1) ~= size(glcmin,2) )
        error('Each GLCM should be square with NumLevels rows and NumLevels
cols');
    end
elseif (nargin == 1) % only GLCM is entered
    pairs = 0; % default is numbers and input 1 for percentage
    if ((size(glcmin,1) <= 1) || (size(glcmin,2) <= 1))
       error('The GLCM should be a 2-D or 3-D matrix.');
    elseif ( size(glcmin,1) ~= size(glcmin,2) )
       error('Each GLCM should be square with NumLevels rows and NumLevels
cols');
    end
end



format long e
if (pairs == 1)
    newn = 1;
    for nglcm = 1:2:size(glcmin,3)
        glcm(:,:,newn)  = glcmin(:,:,nglcm) + glcmin(:,:,nglcm+1);
        newn = newn + 1;
    end
elseif (pairs == 0)
    glcm = glcmin;
end

size_glcm_1 = size(glcm,1);
size_glcm_2 = size(glcm,2);
size_glcm_3 = size(glcm,3);

% checked
out.autoc = zeros(1,size_glcm_3); % Autocorrelation: [2]
out.contr = zeros(1,size_glcm_3); % Contrast: matlab/[1,2]
out.corrm = zeros(1,size_glcm_3); % Correlation: matlab
out.corrp = zeros(1,size_glcm_3); % Correlation: [1,2]
out.cprom = zeros(1,size_glcm_3); % Cluster Prominence: [2]
out.cshad = zeros(1,size_glcm_3); % Cluster Shade: [2]
out.dissi = zeros(1,size_glcm_3); % Dissimilarity: [2]
out.energ = zeros(1,size_glcm_3); % Energy: matlab / [1,2]
out.entro = zeros(1,size_glcm_3); % Entropy: [2]
out.homom = zeros(1,size_glcm_3); % Homogeneity: matlab
out.homop = zeros(1,size_glcm_3); % Homogeneity: [2]
out.maxpr = zeros(1,size_glcm_3); % Maximum probability: [2]

out.sosvh = zeros(1,size_glcm_3); % Sum of sqaures: Variance [1]
out.savgh = zeros(1,size_glcm_3); % Sum average [1]
out.svarh = zeros(1,size_glcm_3); % Sum variance [1]
out.senth = zeros(1,size_glcm_3); % Sum entropy [1]
out.dvarh = zeros(1,size_glcm_3); % Difference variance [4]
%out.dvarh2 = zeros(1,size_glcm_3); % Difference variance [1]
out.denth = zeros(1,size_glcm_3); % Difference entropy [1]
out.inf1h = zeros(1,size_glcm_3); % Information measure of correlation1 [1]
out.inf2h = zeros(1,size_glcm_3); % Informaiton measure of correlation2 [1]
%out.mxcch = zeros(1,size_glcm_3);% maximal correlation coefficient [1]
%out.invdc = zeros(1,size_glcm_3);% Inverse difference (INV) is homom [3]
out.indnc = zeros(1,size_glcm_3); % Inverse difference normalized (INN) [3]
out.idmnc = zeros(1,size_glcm_3); % Inverse difference moment normalized [3]

% correlation with alternate definition of u and s
%out.corrm2 = zeros(1,size_glcm_3); % Correlation: matlab
%out.corrp2 = zeros(1,size_glcm_3); % Correlation: [1,2]

glcm_sum  = zeros(size_glcm_3,1);
glcm_mean = zeros(size_glcm_3,1);
glcm_var  = zeros(size_glcm_3,1);

% http://www.fp.ucalgary.ca/mhallbey/glcm_mean.htm confuses the range of
% i and j used in calculating the means and standard deviations.
% As of now I am not sure if the range of i and j should be [1:Ng] or
% [0:Ng-1]. I am working on obtaining the values of mean and std that get
% the values of correlation that are provided by matlab.
u_x = zeros(size_glcm_3,1);
u_y = zeros(size_glcm_3,1);
s_x = zeros(size_glcm_3,1);
s_y = zeros(size_glcm_3,1);

% % alternate values of u and s
% u_x2 = zeros(size_glcm_3,1);
% u_y2 = zeros(size_glcm_3,1);
% s_x2 = zeros(size_glcm_3,1);
% s_y2 = zeros(size_glcm_3,1);

% checked p_x p_y p_xplusy p_xminusy
p_x = zeros(size_glcm_1,size_glcm_3); % Ng x #glcms[1]
p_y = zeros(size_glcm_2,size_glcm_3); % Ng x #glcms[1]
p_xplusy = zeros((size_glcm_1*2 - 1),size_glcm_3); %[1]
p_xminusy = zeros((size_glcm_1),size_glcm_3); %[1]
% checked hxy hxy1 hxy2 hx hy
hxy  = zeros(size_glcm_3,1);
hxy1 = zeros(size_glcm_3,1);
hx   = zeros(size_glcm_3,1);
hy   = zeros(size_glcm_3,1);
hxy2 = zeros(size_glcm_3,1);

%Q    = zeros(size(glcm));
```

```
for k = 1:size_glcm_3 % number glcms

    glcm_sum(k) = sum(sum(glcm(:,:,k)));
    glcm(:,:,k) = glcm(:,:,k)./glcm_sum(k); % Normalize each glcm
    glcm_mean(k) = mean2(glcm(:,:,k)); % compute mean after norm
    glcm_var(k)  = (std2(glcm(:,:,k)))^2;

    for i = 1:size_glcm_1

        for j = 1:size_glcm_2

            out.contr(k) = out.contr(k) + (abs(i - j))^2.*glcm(i,j,k);
            out.dissi(k) = out.dissi(k) + (abs(i - j)*glcm(i,j,k));
            out.energ(k) = out.energ(k) + (glcm(i,j,k).^2);
            out.entro(k) = out.entro(k) - (glcm(i,j,k)*log(glcm(i,j,k) +
eps));
            out.homom(k) = out.homom(k) + (glcm(i,j,k)/( 1 + abs(i-j) ));
            out.homop(k) = out.homop(k) + (glcm(i,j,k)/( 1 + (i - j)^2));
            % [1] explains sum of squares variance with a mean value;
            % the exact definition for mean has not been provided in
            % the reference: I use the mean of the entire normalized glcm
            out.sosvh(k) = out.sosvh(k) + glcm(i,j,k)*((i - glcm_mean(k))^2);

            %out.invdc(k) = out.homom(k);
            out.indnc(k) = out.indnc(k) + (glcm(i,j,k)/( 1 + (abs(i-
j)/size_glcm_1) ));
            out.idmnc(k) = out.idmnc(k) + (glcm(i,j,k)/( 1 + ((i -
j)/size_glcm_1)^2));
            u_x(k)          = u_x(k) + (i)*glcm(i,j,k); % changed 10/26/08
            u_y(k)          = u_y(k) + (j)*glcm(i,j,k); % changed 10/26/08
            % code requires that Nx = Ny
            % the values of the grey levels range from 1 to (Ng)
        end

    end
    out.maxpr(k) = max(max(glcm(:,:,k)));
end
% glcms have been normalized:
% The contrast has been computed for each glcm in the 3D matrix
% (tested) gives similar results to the matlab function

for k = 1:size_glcm_3

    for i = 1:size_glcm_1

        for j = 1:size_glcm_2
            p_x(i,k) = p_x(i,k) + glcm(i,j,k);
            p_y(i,k) = p_y(i,k) + glcm(j,i,k); % taking i for j and j for i
            if (ismember((i + j),[2:2*size_glcm_1]))
                p_xplusy((i+j)-1,k) = p_xplusy((i+j)-1,k) + glcm(i,j,k);
            end
            if (ismember(abs(i-j),[0:(size_glcm_1-1)]))
                p_xminusy((abs(i-j))+1,k) = p_xminusy((abs(i-j))+1,k) +...
                    glcm(i,j,k);
            end
        end
    end

%     % consider u_x and u_y and s_x and s_y as means and standard deviations
%     % of p_x and p_y
%     u_x2(k) = mean(p_x(:,k));
%     u_y2(k) = mean(p_y(:,k));
%     s_x2(k) = std(p_x(:,k));
%     s_y2(k) = std(p_y(:,k));

end

% marginal probabilities are now available [1]
% p_xminusy has +1 in index for matlab (no 0 index)
% computing sum average, sum variance and sum entropy:
for k = 1:(size_glcm_3)

    for i = 1:(2*(size_glcm_1)-1)
        out.savgh(k) = out.savgh(k) + (i+1)*p_xplusy(i,k);
        % the summation for savgh is for i from 2 to 2*Ng hence (i+1)
        out.senth(k) = out.senth(k) - (p_xplusy(i,k)*log(p_xplusy(i,k) +
eps));
    end

end
% compute sum variance with the help of sum entropy
for k = 1:(size_glcm_3)

    for i = 1:(2*(size_glcm_1)-1)
        out.svarh(k) = out.svarh(k) + (((i+1) -
out.senth(k))^2)*p_xplusy(i,k);
        % the summation for savgh is for i from 2 to 2*Ng hence (i+1)
    end

end
% compute difference variance, difference entropy,
for k = 1:size_glcm_3
% out.dvarh2(k) = var(p_xminusy(:,k));
% but using the formula in
% http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
% we have for dvarh
    for i = 0:(size_glcm_1-1)
        out.denth(k) = out.denth(k) - (p_xminusy(i+1,k)*log(p_xminusy(i+1,k)
+ eps));
        out.dvarh(k) = out.dvarh(k) + (i^2)*p_xminusy(i+1,k);
    end
end

% compute information measure of correlation(1,2) [1]
for k = 1:size_glcm_3
    hxy(k) = out.entro(k);
    for i = 1:size_glcm_1

        for j = 1:size_glcm_2
```

```
            hxy1(k) = hxy1(k) - (glcm(i,j,k)*log(p_x(i,k)*p_y(j,k) + eps));
            hxy2(k) = hxy2(k) - (p_x(i,k)*p_y(j,k)*log(p_x(i,k)*p_y(j,k) +
eps));
%             for Qind = 1:(size_glcm_1)
%                 Q(i,j,k) = Q(i,j,k) +...
%                     ( glcm(i,Qind,k)*glcm(j,Qind,k) /
(p_x(i,k)*p_y(Qind,k)) );
%             end
        end
        hx(k) = hx(k) - (p_x(i,k)*log(p_x(i,k) + eps));
        hy(k) = hy(k) - (p_y(i,k)*log(p_y(i,k) + eps));
    end
    out.inf1h(k) = ( hxy(k) - hxy1(k) ) / ( max([hx(k),hy(k)]) );
    out.inf2h(k) = ( 1 - exp( -2*( hxy2(k) - hxy(k) ) ) )^0.5;
%     eig_Q(k,:)   = eig(Q(:,:,k));
%     sort_eig(k,:)= sort(eig_Q(k,:),'descend');
%     out.mxcch(k) = sort_eig(k,2)^0.5;
% The maximal correlation coefficient was not calculated due to
% computational instability
% http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
end

corm = zeros(size_glcm_3,1);
corp = zeros(size_glcm_3,1);
% using http://www.fp.ucalgary.ca/mhallbey/glcm_variance.htm for s_x s_y
for k = 1:size_glcm_3
    for i = 1:size_glcm_1
        for j = 1:size_glcm_2
            s_x(k)  = s_x(k)  + (((i) - u_x(k))^2)*glcm(i,j,k);
            s_y(k)  = s_y(k)  + (((j) - u_y(k))^2)*glcm(i,j,k);
            corp(k) = corp(k) + ((i)*(j)*glcm(i,j,k));
            corm(k) = corm(k) + (((i) - u_x(k))*((j) - u_y(k))*glcm(i,j,k));
            out.cprom(k) = out.cprom(k) + (((i + j - u_x(k) - u_y(k))^4)*...
                glcm(i,j,k));
            out.cshad(k) = out.cshad(k) + (((i + j - u_x(k) - u_y(k))^3)*...
                glcm(i,j,k));
        end
    end
    % using http://www.fp.ucalgary.ca/mhallbey/glcm_variance.htm for s_x
    % s_y : This solves the difference in value of correlation and might be
    % the right value of standard deviations required
    % According to this website there is a typo in [2] which provides
    % values of variance instead of the standard deviation hence a square
    % root is required as done below:
    s_x(k) = s_x(k) ^ 0.5;
    s_y(k) = s_y(k) ^ 0.5;
    out.autoc(k) = corp(k);
    out.corrp(k) = (corp(k) - u_x(k)*u_y(k))/(s_x(k)*s_y(k));
    out.corrm(k) = corm(k) / (s_x(k)*s_y(k));
%     % alternate values of u and s
%     out.corrp2(k) = (corp(k) - u_x2(k)*u_y2(k))/(s_x2(k)*s_y2(k));
%     out.corrm2(k) = corm(k) / (s_x2(k)*s_y2(k));
end
```