Chapter IV

MODIFICATION IN EXISTING METHODOLOGY AND PROPOSED ALGORITHMS

4.1 Introduction

In chapter 3 basics of image preprocessing, segmentation, feature extraction and classification is explained. This chapter briefly explains existing, modified and new techniques suggested for, Pre-processing, segmentation, feature extraction and classification of Devanagari OCR. Main objective of this work is to develop Devanagari OCR so it is essential to study the composition of the characters.

4.2 Database Creation

Extensive Experimentation is performed on the database specially created for this work. Database is collected approximately from 470 different persons and collected in four different phases named as, Database I - Database of Devanagari characters with Shirorekha, Database II - Database of Devanagari characters without Shirorekha, Database III- Unconstrained Numerals. Database-IV is collection of unconstrained sentences written by different authors. For creation of Database -I a special form is designed for collecting the characters which are written on plain white paper by using black pen. Database IV is as shown in fig. 1.8.

4.3 Image Acquired

Image is acquired either by using scanner or by using camera. 8 bit Gray scale BMP, JPG, PNG image with 300 dpi is acquired by using Hp 3010 scanner and RGB24 bit image is acquired using 20 MP Xpro camera. Camera captured image is converted to 8 bit Gray scale BMP, JPG, PNG image.
Database -I without and with Shirorekha acquired by using scanner

Figure 4.1 - Database -I with Shirorekha acquired by using scanner

Figure 4.2 Database -II Without Shirorekha, acquired by using scanner

Figure 4.3 - Database -I with Shirorekha acquired by using Camera
4.4 Pre-processing Module

Pre-processing brings the document in suitable form which is required for feature extraction. It includes page, word segmentation and character segmentation. Each of the above technique is described in the following manner.

- Purpose of the technique
- Usefulness for Devanagari OCR
- Modifications and new algorithm

Various pre-processing and feature extraction techniques are experimented for optical character recognition system. Main pre-processing modules are

1. Filtering (existing)
2. Binarisation (existing)
3. Skew Removal using Hough Transform (existing)
4. Projection (existing)
5. Segmentation of line, word, character using bounding box by calculating Centroid (modified version of existing technique).
6. Skeletonization and thinning.
7. Shiorekha Removal for plain characters and character validation is developed in this work.

Extensive experimentation is carried out on the database of 5000 numerals and 5000 characters, containing approximately 200 examples of every digit and 150 examples of every character. For every numeral and character 100 images are placed in the training phase and 100 are placed in the testing phase. In addition to the above database approximately 2000 Devanagari worlds, written in three different writing styles are collected.

4.4.1 Page Level Preprocessing

Scanned document may have isolated small blobs of ink or stains and spots on the paper. Median filter is experimented with. Binarisation reduces the noise due to scanning Iterative Global Thresholding is used for the binarisation. Skew may be detected due to improper scanning. The skew have to be removed to facilitate the segmentation of the documents into lines. Hough transform is applied for removal of skew and needs binary images. Skeletonization is also need binary images. Vertical Projection Histogram and Horizontal (VPH ) and Horizontal Projection Histogram (HPH) needs counting of black pixels from given image so binarisation is an important requirement of the system, VPH and HPH is used for segmentation of unconstrained Devanagari script.

\[
I(x,y) = \begin{cases} 
0 & I(X,Y) < t \\
1 & I(X,Y) > t 
\end{cases} \quad (4.1)
\]

\[
I(x,y) = \begin{cases} 
\alpha_{p,q} & \text{for black pixel} \\
0 & \text{for white pixel} 
\end{cases} \quad (4.3)
\]

4.4.2 Horizontal Projection Histogram (HPH)

In each row black and white pixel has to be checked. Number of black pixels in each row are counted and summed up. Horizontal Projection histogram checks (HPH) presence of Shiorekha, number of pixels above Shiorekha( presence of top modifiers). p\textsuperscript{th} row and q\textsuperscript{th} column in the character matrix is given by function \(f(p,q)\) in equation (4.3)

\[f(p, q) = \alpha_{p,q} \quad (4.3)\]

and \(\alpha_{p,q}\) is a binary value (i.e. 0 for white pixel and 1 for black pixel).

The HPH i.e. Hh of character A is the sum of black pixel in a row given by equation (4.4)
\[ H_h = \sum_{p} f(p, q) \quad (4.4) \]

### 4.4.3 Vertical Projection Histogram (VPH)

In each row black and white pixel has to be checked. Number of black pixels in each column are counted and summed up. VPH decides vertical bar in the character and gap between the lines and broken character.

\[ H_n = \sum_{q} f(p, q) \quad (4.5) \]

For character A - अ, VPH and HPH are as shown in fig. 4.6

![VPH and HPH for letter A](image)

a) Devanagari Character A - अ  b) VPH  c) HPH

**Figure 4.6 VPH and HPH for letter A - अ**

VPH helps to classify vertical bar and non vertical bar characters.

### 4.4.4 Fixed Zoning

![Fixed Zoning](image)

a) Horizontal Zoning  b) Vertical Zoning

**Figure 4.7 Fixed Zoning**
HZ1 indicates the Shirorekha if available and presence and absence of top modifiers.

![Image](image_url)

Figure 4.8 Characters with Top Modifiers

A peak value above threshold in VZ2 indicates middle bar and in VZ3 indicates end bar

![Image](image_url)

a) Characters with Middle Vertical Bar   b) Characters with End Vertical Bar

![Image](image_url)

c) No Bar Characters

Figure 4.9 Vertical and Non Vertical Bar Characters

4.4.5 Character Level Preprocessing

Characters may have some background spots and strains. Median filter is used to remove the noise. Character is preprocessed and Shirorekha (header line) is detected and removed. Skeletonization of a character facilitates feature extraction. Skeletonization is useful in finding end points, junction points, Zero crossings etc. For Median Filter and Shirorekha removal following algorithm is used [36],[86].

Median Filter

- Input the image of M X N size.
- From equation (3.1) arrange the window pixels either increasing or decreasing.
- Choose the window size \( N_w \) odd.
- If it is even take the average of two middle values.

Shirorekha Removal only used in plain character preprocessing
- Plot HPH
- Apply the fixed zone as per diagram 4.7 a).
- HZ1 indicates Shirorekha Presence or absence of Shirorekha.

**Character Level Pre-processing.**

- Input the character of size (M,N)
- Calculate the Centroid of input character.
- Apply the equation (3.10) , (3.11) & (3.12) , Calculate the bounding box.
- Segment the character based on bounding box method using equation (3.12) & (3.13)

**4.5 Devanagari Numeral Pre-Processing**

Numerals are the integrated part of Devanagari script recognition. Basic ten numerals written on tabular application form are collected as a database. Numerals are extracted from data sheets and from application form. Numerals are pre-processed and brought in the suitable form for feature extraction. Ten different classes are defined based on the basic shape and profile of the numeral[32],[49].

![Figure 4.10 Numeral Database Collected On Plain Sheet](image)
Modified new algorithm for Pre-processing

- Filter the input image and binaries it by using Otsu's algorithm
- Detect the skew by applying Hough Transform.
- Rotate the document by detected skew angle.
- Plot HPH by using equation (4.4)
- Find the minimum and maximum value from the histogram.
- If minimum value is below threshold 0.02 x peak, segment the line (0.02 value decided after extensive experimentation).
- Store the lines in line arrays.
- Plot HPH and segment the word.
- Store the word in word array.
- Find the vertical angle by plotting vertical projection between -5 degree to + 5 degree.
- Remove the slat by using Shear algorithm.
- Thin the word.
- Find VPH of a thin word
- Segment the character. Apply Hough transform and remove the Shirorekha
- Find VPH and Segment the character.
- Store the character in character array.

For numerals it is not needed to remove Shirorekha. Numerals are treated isolated from each other. All stages of document level preprocessing are illustrated in figure 4.11
4.6 Feature Extraction Module

The following methods are used for feature extraction

1. Principal Component Analysis (existing).
2. Discrete Wavelet Transform (existing). Multiclass unconstrained numeral recognition is the modification in the existing technique.
3. Structural and statistical feature extraction with minimum distance classifier for similar type of character pairs is newly developed method in this work. Two algorithms developed in this work are
   i) Novel feature extraction algorithm for non confusing characters.
   ii) Robust feature extraction algorithm for confusing characters.
Principal Component Analysis (PCA)

Principal component analysis is a linear transform-based method. It is widely used for large data analysis and compression. Principal component analysis is based on the statistical representation of a random variable.

Following steps are used for calculation of PCA.
1. Calculate the mean of data set \(X\) of dimension \(M \times N\).
2. Find the mean along \(m = 1, \ldots, M\) of each column.
3. Vector \(p\) of dimension \(M \times 1\) is having mean value calculated in step 2.
   \[
   p[m] = \left(\frac{1}{N}\right) \sum X[m, n] \quad (4.6)
   \]
4. Subtract the empirical mean vector \(p\) from each column of the data matrix \(X\).
5. Store mean-subtracted data in the \(M \times N\) matrix \(B\).
   \[
   B = X - [pxq] \quad (4.7)
   \]
   [Where \(q\) is a \(1 \times N\) row vector of all 1's : \(q[n]=1\) for \(n=1\ldots N\)]
6. Find covariance matrix \(C\) of dimension \(M \times M\) from the outer product of matrix \(B\) with itself:
   \[
   C = E[BB^*] = \left(\frac{1}{N}\right) \sum B.B^* \quad (4.8)
   \]
7. Compute the matrix \(V\) of eigenvectors which diagonalize the covariance matrix \(C\),
   \[
   R = V^{-1}CV \quad (4.9)
   \]
   where \(R\) is the diagonal matrix of Eigen values of \(C\).
8. Matrix \(R\) will take the form of an \(M \times M\) diagonal matrix,
   where \(R[i, j]=\lambda_m\) for \(i = j = m\) is the \(m\)th eigen value of the covariance matrix \(C\),
   and \(R[i, j]=0\) for \(i \neq j\)
   Matrix \(V = M \times M\) dimensions, \(M\) column vectors, each of length \(M\),
   and represents \(M\) eigenvectors of the covariance matrix \(C\).
9. Arrange variance in decreasing order
10. Compute the feature vector \((E_1, E_2, E_3, \ldots, E_n)\)
    For PCA Image size of 100 x 100 is chosen which results in feature set size of 10000 x 1 which is further reduced to 200 features.
4.6.2 Discrete Wavelet Transform

In DWT following features are calculated[29],[48],[86].

**Homogeneity** - Homogeneity measures the spatial closeness of the distribution of the co-occurrence matrix.

\[
\text{Homogeneity} = \sum_{i=1}^{K} \sum_{j=1}^{K} \frac{P_{ij}}{1 + |i - j|}
\]  
(4.10)

**Correlation** - Correlation measures the linear dependency of grey levels of neighboring pixels.

\[
\text{Correlation} = \sum_{i,j} \frac{(i - \mu_i)(j - \mu_j)p(i, j)}{\sigma_i \sigma_j}
\]  
(4.11)

**Contrast** - Contrast is a measure of intensity contrast between a pixel and its neighbor over the entire image.

\[
\text{Con} = \sum_{i=1}^{K} \sum_{j=1}^{K} (i - j)^2 p_{ij}
\]  
(4.12)

**Energy** - Energy is a measure of uniformity where energy is maximum when the image is constant.

\[
\text{Ene} = \sum_{i=1}^{K} \sum_{j=1}^{K} p_{ij}^2
\]  
(4.13)

For DWT if image size of 100 x 100 is selected then it results in 2500 x 1 features set. Feature Vector For Ψ by using DWT is as shown in table 4.1.

4.6.3 Proposed Novel Method for Feature Extraction

- **For Non - Confusing Characters.**

Mathematically associated parameters with character defines statistical features, aspect ratio, Centroid etc. are considered as statistical features or geometrical features. Structural feature are mostly related to the shape profile of the character. It includes
number of loops, Number of junctions, number of end points, zero crossings etc. as mentioned in feature vector summary table 3.1. For Noun confusing characters set of 75 features is used and listed in table 4.1

Table -4.1 Features for Non -Confusing Characters

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>No. of features in Feature Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Ratio</td>
<td>1</td>
</tr>
<tr>
<td>Number of end points</td>
<td>1</td>
</tr>
<tr>
<td>Number of Junctions</td>
<td>1</td>
</tr>
<tr>
<td>Number of Loops</td>
<td>1</td>
</tr>
<tr>
<td>Zero Crossings</td>
<td>16</td>
</tr>
<tr>
<td>Pixel Density</td>
<td>9</td>
</tr>
<tr>
<td>Fixed Point Distance</td>
<td>23</td>
</tr>
<tr>
<td>Centre of Gravity</td>
<td>23</td>
</tr>
<tr>
<td><strong>Total No of Features</strong></td>
<td><strong>75</strong></td>
</tr>
</tbody>
</table>

For Dissimilar characters number of features considered are 75. Minimum distance classifier is used to calculate the distance between two classes. Figure 4.12 depicts the comparison between two dissimilar characters 产地 and 中文.

![Figure 4.12 Comparison between dissimilar characters 产地 and 中文 (feature set of 75)](image)
For Confusing Characters.

For similar characters ஐ and ச feature set of 199 features is considered and compared with each other.

**Figure 4.13 Comparison between similar characters ஐ and ச (feature set of 199)**

**Table -4.2 Features for Confusing Characters**

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>No. of features in a Feature Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robust Feature</td>
<td>124</td>
</tr>
<tr>
<td>Aspect Ratio</td>
<td>1</td>
</tr>
<tr>
<td>Number of end points</td>
<td>1</td>
</tr>
<tr>
<td>Number of Junctions</td>
<td>1</td>
</tr>
<tr>
<td>Number of Loops</td>
<td>1</td>
</tr>
<tr>
<td>Zero Crossings</td>
<td>16</td>
</tr>
<tr>
<td>Pixel Density</td>
<td>9</td>
</tr>
<tr>
<td>Fixed Point Distance</td>
<td>23</td>
</tr>
<tr>
<td>Centre of Gravity</td>
<td>23</td>
</tr>
<tr>
<td><strong>Total No of Features</strong></td>
<td><strong>199</strong></td>
</tr>
</tbody>
</table>
For confusing characters (similar character) only a set of 199 features is considered. It includes number of loops, Number of junctions, number of end points, zero crossings etc. as shown in table 4.2. Distance is calculated using minimum distance classifier.

4.7 Classification

Feature vector is applied to the classifier. Researchers had tried many classifiers for Devanagari OCR. Some of them are[35]

- Bayes Classifier
- Pattern Matching
- Correlation between the images
- Hidden Markov Model[34],[62]
- Principal Component Analysis

Few Artificial Neural Network Based classifiers are

- Multilayer Perceptron
- Radial Basis Function Network
- Self Organizing Feature Map (SOM)
- Learning Vector Quantization (LVQ)
- Self organizing Map + Learning Vector Quantization
- Support Vector Machine (SVM)

Experimentation is carried out by using SOM + LVQ, SVM and pattern matching classifiers. SOM + LVQ , SVM are very powerful classifiers if efficiently used[57],[79],[81],[82].

4.7.1 Self Organizing Feature Map.

Classification and recognition is based on Artificial Neural Network. Kohonens Self-organizing map is a two-layered network that can organize a topological map from a random starting point. The resulting map shows the natural relationship among the patterns that are given to the network. Input and output layers are fully interconnected[24].

57
When an input pattern (feature vector) is applied, each unit in the first layer takes the values of the corresponding entry, from an input pattern. The second layer units then sums the inputs and compute to find a single winning unit. Every unit is associated with its weight value which is initially random. The overall operation of the Kohonens network is similar to the competitive learning paradigm [24]. Each input pattern vector is uniformly distributed between 0 and 1. We can apply input pattern vector with n entries. As a result the input pattern is uniformly spread over d dimensional hypercube. If n = 2, then the input pattern are uniformly spread over a square [24],[84].

Let n denotes the dimension of an input space. An input pattern (vector) selected at random from input space is denoted as [24]

\[ X = [x_1, x_2, x_3, x_4, x_5, \ldots, x_n]^T \]  

(4.14)

The synaptic weight vector of each neuron in the network has same dimensions as the input space. Let the weight vector of neuron j be denoted by [24].

\[ W = [w_{j1}, w_{j2}, w_{j3}, w_{j4}, w_{j5}, \ldots, \ldots, w_{jn}]^T \quad j = 1, 2, \ldots, l \]  

(4.15)

Where l is the length of neuron in the network. The first step in the operation of Kohonens network is to compute a matching value for unit i [24]

\[ i(x) = \arg\min_j \|x - w_j\| \quad j \in \mathcal{A} \]  

(4.16)

Where \( \mathcal{A} \) is the lattice of neurons. Distance between vector x and \( w_j \) is denoted by d and computed by
\[ d = \sqrt{\sum (x_j - w_{ij})^2} \]  

(4.17)

The unit with the best match wins the competition. The winner unit with the best match is chosen such that

\[ \|x - w_c\| = \min_i \{ \|x - w_i\| \} \]  

(4.18)

After winning unit is identified, the next step is to identify the neighborhood around it. The neighborhood in this case consists of the units that are within the square having center winning unit c as shown in figure 4.15. Weights are updated for all neurons that are in the neighborhood of the winning unit.

\[ W_{ij} = \{ \sum (x_j (t) - w_{ij} (t)) \} \]  

(4.19)

If the unit i is the neighborhood \( N_c \) otherwise

\[ W_{ij\new} = (w_{ij\old} + \alpha(t).w_{ij}) \]  

(4.20)

Where \( \alpha \) is the learning rate initially it is set to 1. Two parameters that must be specified are, the learning rate \( \alpha_i \), the neighborhood size \( N_c \), learning rate is given by

\[ \alpha = \alpha_0 \left\{ 1 - \left( \frac{t}{T} \right) \right\} \]  

(4.21)

Where \( t = \) the current training iteration, \( T = \) Total no. of training iterations to be done. Thus \( \alpha_0 \) is decreased until it reaches 0. The decrease is linear with number of training iterations completed. Consider the neighborhood as shown in figure 4.15 centered on winning unit c, at position \((x_c, y_c)\). Let \( d \) be the distance from \( c \) to the edge of the neighborhood. The neighborhood is then all \((x, y)\) Such that [24]

\[ c - d < x < c + d \quad \text{and} \quad c - d < y < c + d \]  

(4.22)
This defines a square neighborhood about C. Since the width of the neighborhood decreases over the training iterations, the value of d decreases. Initially d is set at a chosen value denoted by d₀ may be chosen at a half or a third of the width of the competitive layer of the processing units. The value of d is then made to decrease according to the equation.

\[ d = d_0 \left\{ 1 - \left( \frac{t}{T} \right) \right\} \]  \hspace{1cm} (4.23)

Where \( t = \text{the current training iteration} \)

\( T = \text{Total no. of training iterations to be done.} \)

### 4.7.2 Algorithm For Self Organizing Feature Map

1. Initialize the weights from M inputs to the N output units to Small random Values. Initialize the size of neighborhood region \( N_c(0) \).
2. Present a new input x.
3. Compute the distance \( d_i \) between the inputs and weight on each output unit i:

\[ d_i = \sqrt{\sum_{j=1}^{N} \left( x_j(t) - w_{ij}(t) \right)^2} \quad i = 1,2, \ldots \ldots \ldots N \]  \hspace{1cm} (4.24)

Where \( x_j(t) \) is the input unit and \( w_{ij}(t) \) is the weight vector from j\(^{th}\) input unit

4. Select the output unit d with minimum distance

\[ d = \text{index} \left[ \min(d_i) \right] \]  \hspace{1cm} (4.25)

5. Update weights to node d and its neighbors

\[ W_{ij}(t+1) = (W_{ij}(t) + \alpha(t) \left( x_j(t) - w_{ij}(t) \right) \]  \hspace{1cm} (4.26)

For \( i \in N_c(t) \) and \( j=1,2,3, \ldots \ldots M \) where \( \alpha(t) \) is the learning rate parameter \( (0 < \alpha(t) < 1) \)

that decreases with time \( N_c(t) \) gives the neighborhood region around node d at time t

6. Repeat the 2 to 5 several times.

The output of SOM is fed to LVQ for fine tuning.
4.7.3 Vector Quantization.

Vector quantization is a classical method that produces an approximation to continuous probability density function \( \rho(x) \) of the vectorial input variable \( x \) using a finite number of codebook vector[13].

\[
Rate \ R = \log_2 M \quad \text{per vector}
\]
\[
Rate \ R = (\log_2 M)/k \quad \text{per symbol} \quad (4.27)
\]

Output of Self Organizing Map is fed to Learning Vector Quantizer. Learning vector Quantizer follows supervised learning.

- **Learning Vector Quantization.**

Self Organizing Map is to be used as a pattern classifier in which the cells or their responses are grouped into subsets, each of which corresponds to a discrete class of patterns, then the problem becomes a decision process and must be handled differently. The original Map, like the classical Vector Quantization method is mainly intended to approximate input signal values, or their probability density function, by quantized “codebook” vectors that are localized in the input space to minimize a quantization error functional. On the other hand, if the signal sets are to be classified into a finite number of categories, then several codebook vectors are usually made to represent each class, and their identity within the classes is no longer important. In fact, only decisions made at class borders count. It is possible to define effective values for the codebook vectors such that they directly define near-optimal decision borders between the classes, even in the sense of classical Bayesian decision theory. These strategies and learning algorithms are known as Learning Vector Quantization(LVQ)[24],[103].
• **Type One Learning Vector Quantization (LVQ1).**

Several codebook vectors $m_i$ are assigned to each class and each of them is labeled with the corresponding class symbol, the class regions in the $x$ space are defined by simple nearest-neighbor comparison of $x$ with the $m_i$; the label of the closest $m_i$ defines the classification of $x$. To define the optimal placement of $m_i$ in an iterative learning process, initial values for them must first be set using any classical VQ method or by the self-Organizing Map algorithm. The initial values for both cases roughly correspond to overall statistical density function $p(x)$ of the input. The next phase is to determine the labels of the codebook vectors by presenting a number of input vectors with known classification, and assigning the cells to different classes by majority voting, according to the frequency with which each $m_i$ is closest to the calibration vectors of a particular class[24].

Update the $m_i = m_i(t)$ as follows:

$$m_c(t + 1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$$  \hspace{1cm} (4.28)

If $x$ is classified correctly,

$$m_c(t + 1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$$  \hspace{1cm} (4.29)

If the classification of $x$ is incorrect,

$$m_c(t + 1) = m_i(t) \text{ for } i \neq c$$  \hspace{1cm} (4.30)

Here $\alpha(t)$ is scalar gain ($0 < \alpha(t) < 1$), which is decreasing monotonically in time, as in earlier formulas. Since a fine-tuning method, one should start with a fairly small value, say $\alpha(0) = 0.01$ or 0.02 and let it decrease to zero, say in **100 000 steps.**
This algorithm tends to reduce the point density of the \( m_i \) around the Bayesian decision surfaces. The decision surface defined by this classifier seems to be near optimal, although piecewise linear, and the classification accuracy in this rather difficult example is within a fraction of a percent of that achieved with the Bayes classifier\[24].

- **Type Two Learning Vector Quantization (LVQ2)**

  The previous algorithm can easily be modified to comply even better with Bayes’ decision. Assume that two codebook vectors \( m_i \) and \( m_j \) belong to different classes and are closest neighbors in the vector space are initially in a wrong position then,

  \[
  m_i(t + 1) = m_i(t) + \alpha(t) [x(t) - m_i(t)] \quad (4.31)
  \]

  \[
  m_j(t + 1) = m_j(t) + \alpha(t) [x(t) - m_j(t)] \quad (4.32)
  \]

  if \( C_i \) is the nearest class, but \( x \) belongs to \( C_j \neq C_i \) where \( C_j \) is the next-to-nearest class ("runner-up"); furthermore \( x \) must fall into the "window". In all the other case

  \[
  m_k(t + 1) = m_k(t) \quad (4.33)
  \]

  The optimal width of the window must be determine experimentally, and it depends on the number of available samples. The classification accuracy of the LVQ2 is first improved when the decision surface is shifted towards the Bayes limit; after that, however, the \( m_i \) continue “drifting away”. Therefore LVQ2 is applied for relatively short time.

- **Type Three Learning Vector Quantization (LVQ3)**

  Combining above two ideas i.e. LVQ1+LVQ2 an improved algorithm is obtained which is called LVQ3:

  \[
  m_i(t + 1) = m_i(t)
  - \alpha(t)[x(t) - m_i(t)] \quad (4.34)
  \]

  \[
  m_j(t + 1) = m_j(t) + \alpha(t)[x(t) - m_j(t)] \quad (4.35)
  \]

  where \( m_i \) and \( m_j \) are the two closest codebook vectors to \( x \), and \( x \) and \( m_j \) belong to the same class, while \( x \) and \( m_i \) belong to different classes\[24].

- **Learning Vector Quantization Algorithm**

  **Step 0** Initialize the reference vectors, Initialize learning rate \( \alpha(0) \).

  **Step 1** While stopping condition is false do steps 2 – 6

  **Step 2** for each training input vector \( x \) do steps 3-4

  **Step 3** Find \( J \) so that \( \| X - W_j \| \) is minimum

  63
### Step 4
Update $W_j$ as follows
If $w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha [(x-w_j(\text{old})]$  
If $T \neq C_j$, Then  
$w_{ij}(\text{new}) = w_{ij}(\text{old}) - \alpha [(x-w_j(\text{old})] t)]$

### Step 5
Reduce the learning rate.

### Step 6
Test stopping condition (Stopping condition is fixed no. of iterations).

Thus we conclude that in LVQ1 only one of the $m_i$ values is changed at each step, LVQ2 and LVQ3 changes two codebook vectors simultaneously.

### 4.7.4 Support Vector Machine (SVM)

Support vector machine is one of the most popular classifier. Basically SVM is a binary learning machine with highly elegant properties. The main ideas behind use of support vector machine are[102],

- Apply training samples to SVM which defines a hyper plane as a decision surface in such a way that margin of separation between positive and negative class is maximized.
- Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications.
- Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space.

Multiclass SVM with Majority voting is used in the present work.

![Support Vector Machine using Radial Basis Function Kernel](image)

**Figure 4.18** Support Vector Machine using Radial Basis Function Kernel
4.8 Concluding Remarks

Basics of image preprocessing, segmentation, feature extraction and classification is explained. Existing techniques are briefly studied and pre-processing, segmentation techniques are modified. New techniques are suggested mainly for feature extraction and classification of Devanagari OCR. Next chapter provides the details of the experimentation and performance analysis.