### 

|  |  |
| --- | --- |
| |  | | --- | |  | |

**Parkinsons Data Set**

**Final Project**

**ECE 471**

**Niloo Ranjan**

Cody Mitchell

**December 3, 2015**

[**nranjan@vols.utk.edu**](mailto:nranjan@vols.utk.edu)

[cmitch49@vols.utk.edu](mailto:cmitch49@vols.utk.edu)

**Abstract**

**Introduction**

For the final project, we chose the Parkinson’s data set from the UCI Machine Learning repository. After the work we did with the diabetes in Pima Indian dataset, we thought that it would be interesting to continue exploring the different applications of pattern recognition in the medical field. For this project, we will use the data to compare different classification techniques and their accuracies, as well as learn how to make each classification algorithm run more efficiently. The data that we are using for this project is the recorded speech signals of people who are either positive or negative for Parkinson’s. Our goal is to use this data to derive a decision rule and find a connection between vocal patterns and Parkinson’s.

This dataset was uploaded to the UCI learning repository by Max Little of the University of Oxford on June 6, 2008. The primary focus of the data is to discriminate healthy people from those with PD. The University did this by using quadratic discriminant analysis on the speech sample data. They also introduced recurrence and fractal scaling of the speech patterns, which overcame existing range limitations. They were able to use two tools to create a “hoarseness” diagram for the vocal samples. They then developed a “simple bootstrapped classifier” that used these two features to distinguish normal voices from disordered voices. On a large database of subjects, using quadratic discriminant analysis, the University of Oxford was able to distinguish normal cases from disordered cases to overall correct classification performance of 91.8 ± 2.0%, the true positive classification performance of 95.4 ± 3.2%, and the true negative performance of 91.5 ± 2.3% (95% confidence). However, we will not be using the same methods as the University of Oxford did when they first published their data. Instead, by using the methods that we discussed in class, we hope that we will be able to achieve that high, if not higher, of a level of accuracy. If we can use new techniques and can achieve a high level of accuracy, we will easily be able to see the role of pattern classification in the clinical world of medicine.

Tracking Parkinson’s disease symptom progression often uses the unified Parkinson’s disease rating scale, or UPDRS, which requires the patient’s physical presence in the clinic, and time-consuming physical examination by trained medical practitioners. Thus, symptom monitoring is costly and logistically inconvenient for both patient and clinical staff. It also hinders recruitment for any future large-scale clinical trials. In this dataset, for the first time, the University demonstrated remote replication of UPDRS with useful accuracy using one simple and noninvasive speech test for each patient. They then ran their classification over 6000 recordings from 42 Parkinson’s patients and acquired accurate results. This data helped support the feasibility of frequent, remote, and accurate UPDRS tracking. These findings played a key role in enabling large-scale clinical trials into novel Parkinson’s treatments and removed the burden on clinical staff and those who were diagnosed with Parkinson’s.

**Technical Approach**

**Normalization**

To normalize the data set, we first calculated the mean and the standard deviation of each feature in the training set as a whole without diving the data set into classes. We used the same parameters to normalization both the training and the testing set. We used following equation to perform the normalization process,

(x-m\_i) / sigma\_i

Where, x is a sample vector, m\_i is the mean of each feature i, and sigma\_i is the standard deviation of each feature i.

**Principal Component Analysis (PCA)**

We used the PCA method to transform the normalized data set from higher dimensional space to lower dimensional space. The purpose is to reduce the complexity of the classification of the data set without loosing the aspects of the features in the data set. So, we are trying to find a new feature space that best represent the original feature space. The goal here is to find a set of basis vector that can be used to linearly represent the sample vector using principal components. The sample vector X can be represented as,



Where X is the sample vector and bi is the set of basis vectors, and yi is the principal components of X vector.

Also, it is required that the basis vectors are linearly independent and orthonormal to each other. That means,



Also we find m principal components of y (m<d) so that we can represent X using this m component with some scarify to error. Thus, we calculate the first m elements of **y** and replace the rest of element with some constant.



The error of discarding the less important features is,



The error that we can be tolerated in the transformation process is calculated using mean-square, which is shown as follows,



The optimal that will minimize the error  is calculated by taking partial derivative of with respect to  and the process is shown as,



Therefore, the mean square error now is given by,



The optimal choice of the basis vectors is the Eigen vector of the covariance of the data set. So, the covariance of the normalized training data set is calculated and then the Eigen values and vectors of the calculated covariance is derived. Then, without the loss of generality the set of eigen vectors are sorted in terms of ordered eigen values in decreasing order then the set of eigen vectors are kept as major eigen vectors or “principal components” with the error rate not greater than 0.10.

**Fisher Linear Discriminant**

This technique tries to find the optimal projection direction W so that the projected data can be best separated. For the two-class, it is the projection of data from d-dimensional space onto a line. The projection of the sample on to the optimal projection direction, is found using equation,



Where, W is the projection direction line, x is the sample vector, and y is the projected data.

The optimal projection direction W is calculated using formula,



Where, W is the projection direction,

 is the inverse of the within-class scatter matrix, given by

, where  and  are the between-class scatter matrix.

m1 and m2 are the d-dimensional sample mean of class 1 and class 2 given by,



In this case the  is responsible for rotation effect for multiplication and it determines the angle of rotation for projection direction W.

**Gaussian Parameter Estimation:**

The parameters of the Gaussian are estimated using maximum likelihood estimation technique. We calculated  and  for both the class in the training set using following equations:





where is the mean of the mean of one class i and

 is the covariance matrix of a class i.

**Decision Rule Derivation**

This project is the problem of pattern classification using two features, which means the Gaussian to model the training set, is 2-dimentinal (d =2) multivariate Gaussian.

The general multivariate normal density in d dimension is represented as:



Where,









In this project, d =2, therefore, the PDF for each of the class of training data set is of the form of,

 (3)

**Discriminant Function**

There are many ways to represent pattern classifiers. One way to represent pattern classifier is using discriminant functions gi(x), i = 1,….c to model the data and the discriminant function for Gaussian PDF is defined as,

(4)

where ln denotes natural algorithm.



Now, we derive the decision rule using three special cases based on some assumptions.

**Case 1: i =I**

In this case, we assumed that the features are statistically independent, and have the same variance, . Also, the covariance matrix is diagonal and it is just the  times the identity matrix **I**. Geometrically, this is the situation when the samples fall in equal-size hyperspherical clusters, and the cluster for the ith class is centered about the mean vector . And the components of the determinant and the inverse of covariance is given by,



Since, both the |Ei| and the ln 2pi term in equation 4 are independent of i, they are unimportant additive constant that can be ingnored. Thus the simple discriminant function in this case is,

 where 

This is the Minimum-Distance Classifier. When P(wi) are the same for all c classes, the discriminant function is actually measuring the minimum distance from each x to each of the c mean vectors

 (5)

This is the decision rule for discriminant function derived for Case I.

**Decision Boundary:** Therefore, for case I the decision boundary is derived from decision rule as follows,





 (6)

Thus, the decision boundary for case one is hyperplane of d-1 dimension.

**Case 2: i =**

In this case, the covariance matrices for all the classes are identical but not a scalar of identity matrix. This is the situation when geometrically all of the samples fall in hyperellipsoidal cluster of equal size and shape, and the cluster of the ith class is centered about the mean vector. Since, both the determinant and the ln 2pi term in equation 4 are independent of i, they can be ignored as constant. Thus the discriminant function in this case is simplified to,



where  is the Squared Mahalanobis distance

Therefore, in case 2 the decision boundary would be derived from decision rule

 (7)

**Decision boundary**: It is derived from the decision rule as follows,



 (8)

The decision boundary in Case II is hyperplane of d-1 dimension.

**Case 3:** **i = arbitrary**

In this general multivariate normal case, the covariance matrices are different for each category. Therefore, the only term can be neglected in equation 4 is ln 2pi term, and the resulting discriminant function is quadratic derived as follows,

 (9)

Therefore, the decision rule for Case III is quadratic classifier.

**Decision boundary**: The decision boundary is derived as follows,

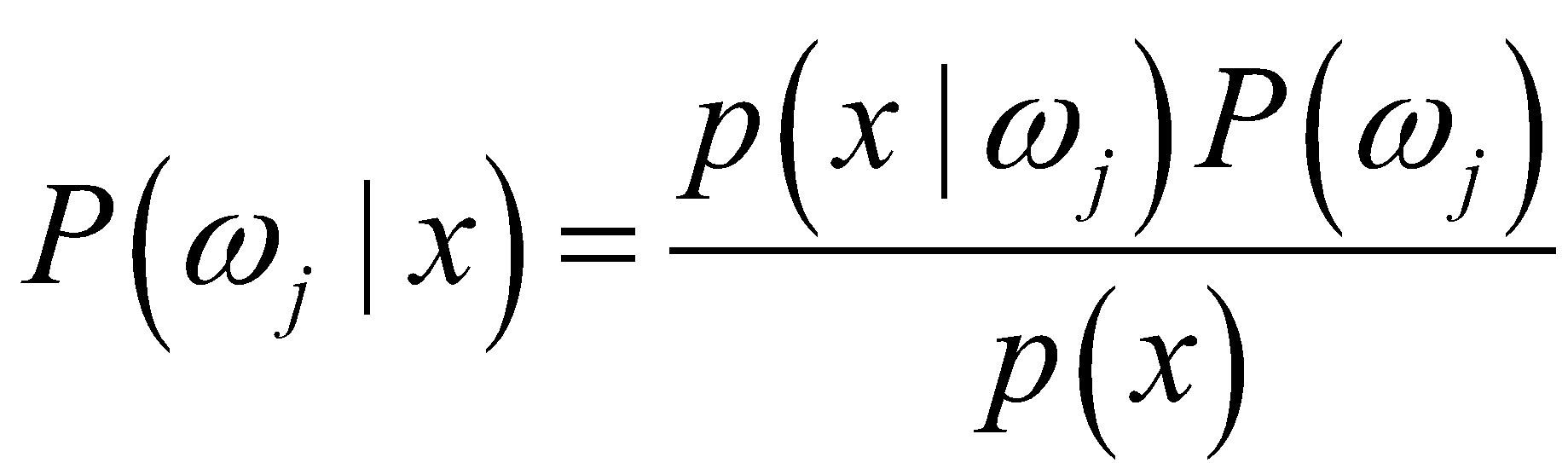




Thus, the decision boundary for Case III is hyperquadratic for 2-D Gaussian.

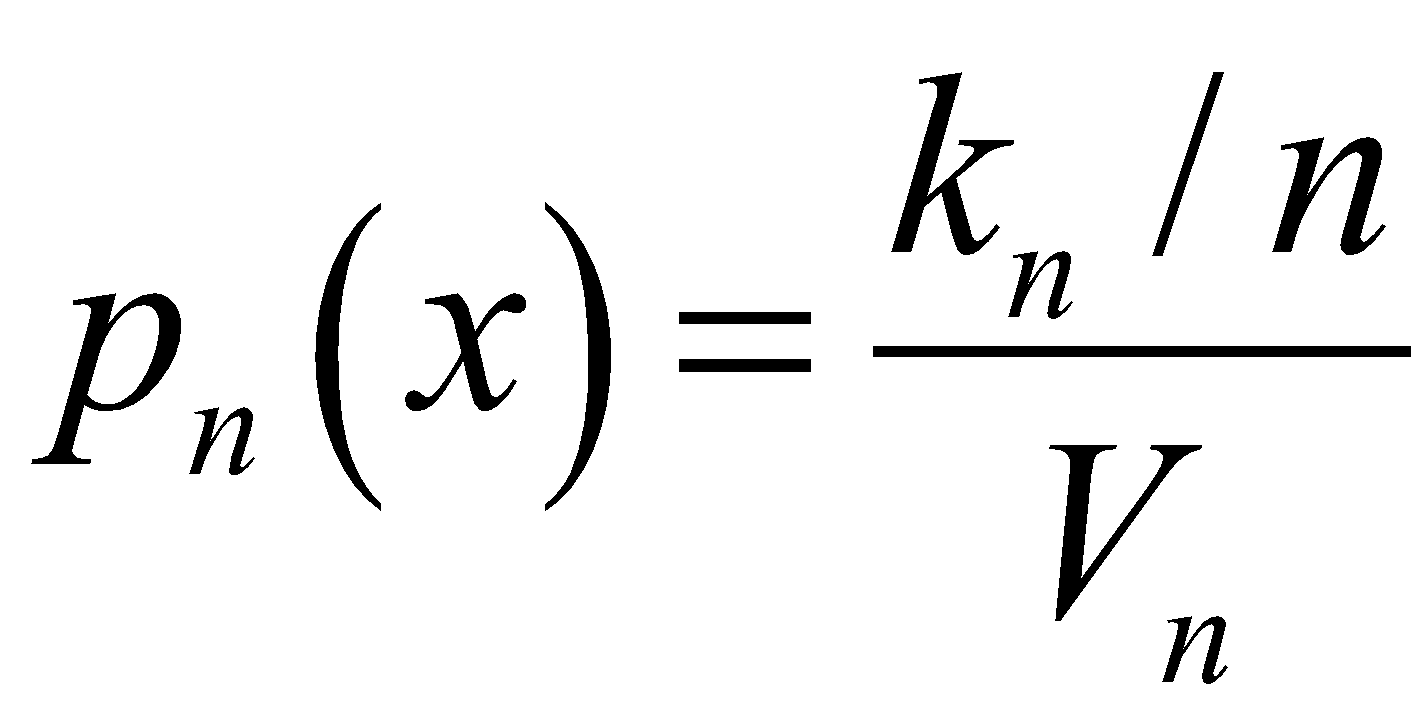
**K-Nearest Neighbor (KNN) Estimation**

This method estimates the density functions of the given data set without the assumption that the pdf has a particular form. This method follows the procedure of directly estimating the decision function posteriori probability,

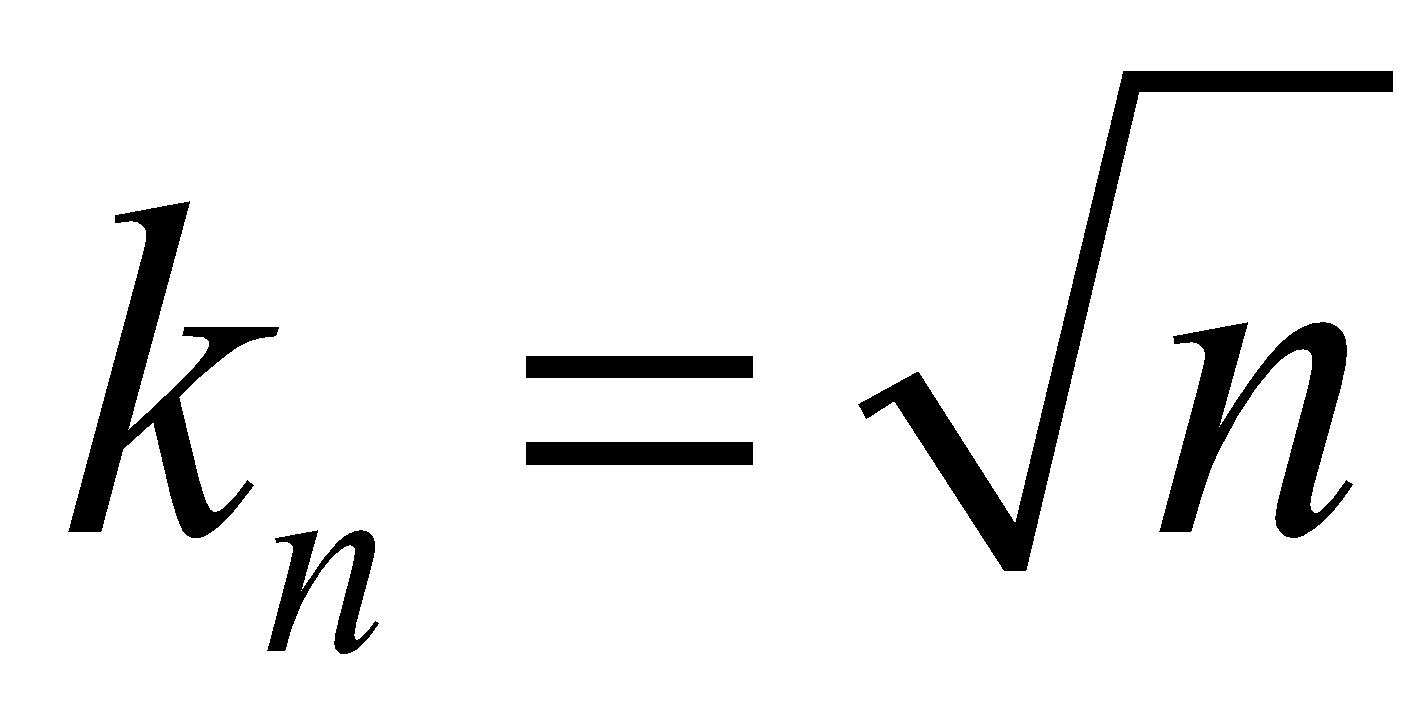


from the k nearest samples found in the training set.

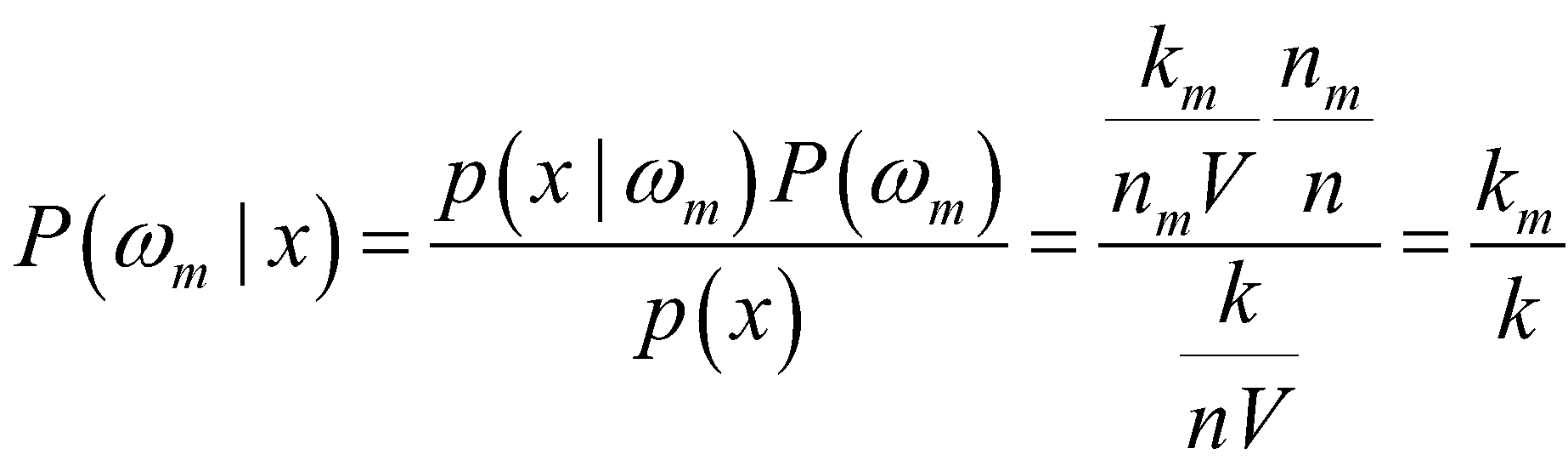
To implement this method, we try to estimate the value of *p*(*x*) for a particular testing sample from *n* samples in the training set. To do this, we center a cell at *x* and let it grow until it contains *kn* samples,



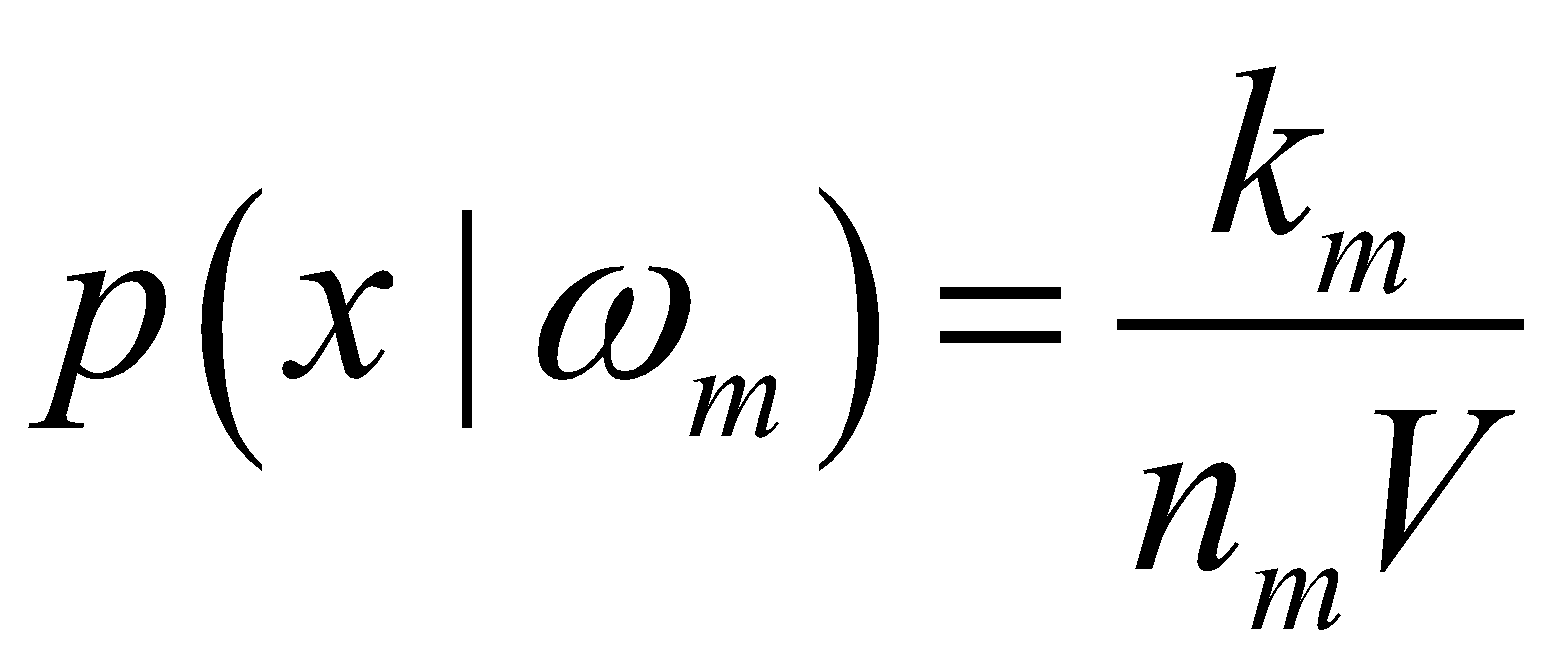
where *kn* can be some function of *n.* The commonly used value of k is given by,

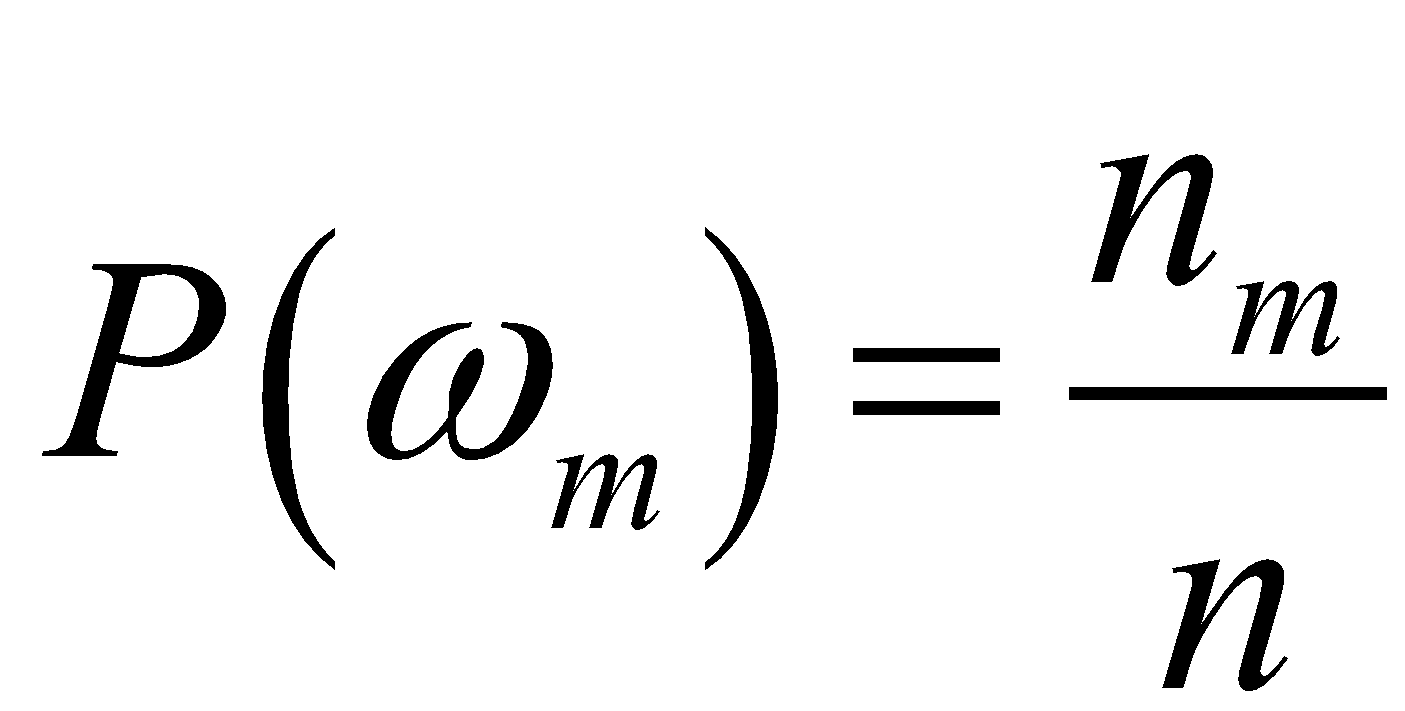


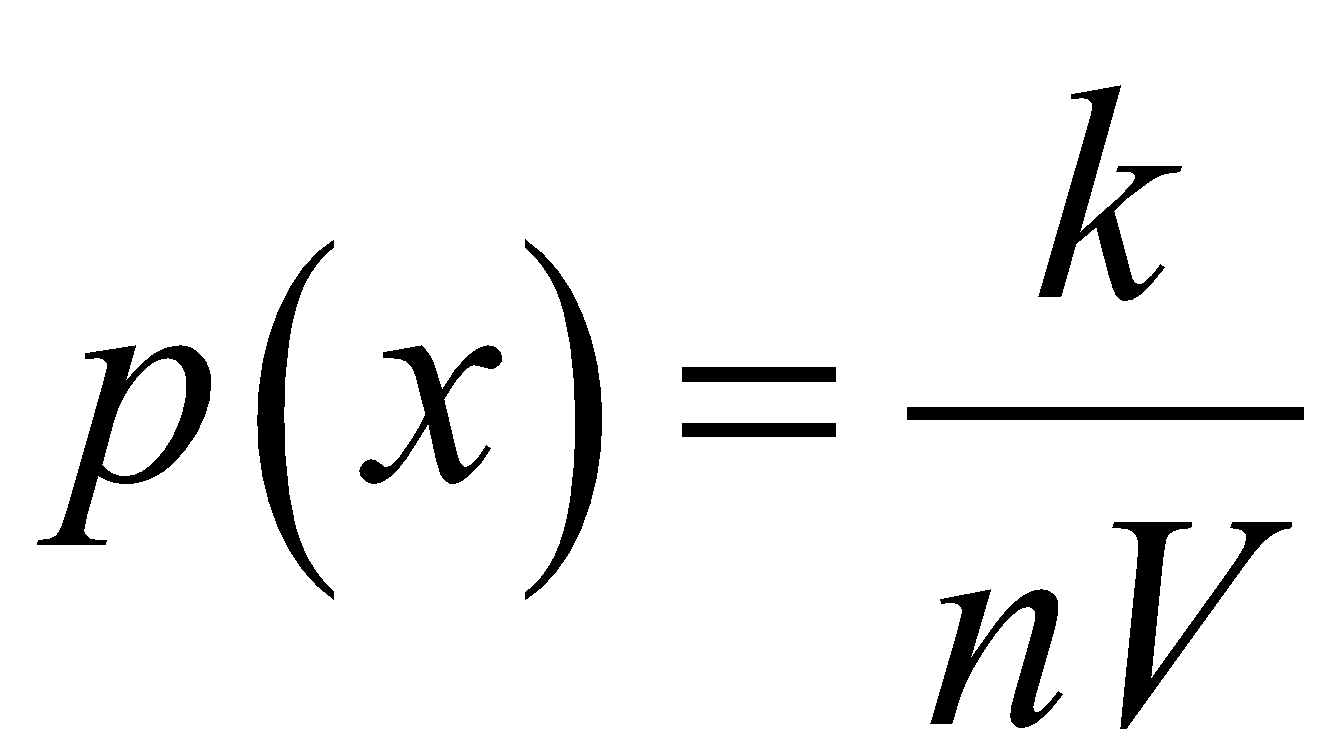
The needed posterior probability function is formulated as follows. For a given testing sample point **x** at which we are to calculate the density function, first we try to find the hypersphere of volume ***V*** which can just enclose *k* points from the entire training set. Then we check the class label information of those contained points. Then we assign the class label as m if *km* of those points belongs to class m. So, the estimation of the posteriori probability for class *m* is given by,



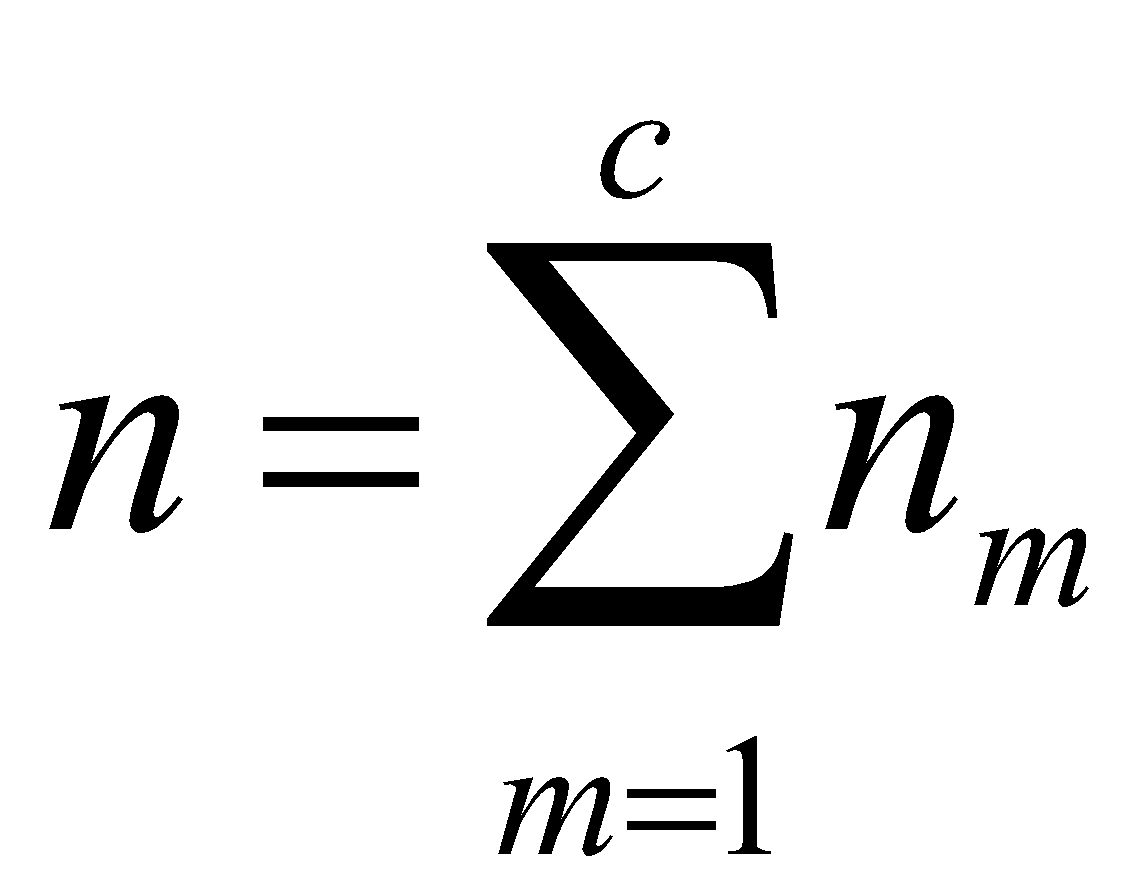
Where,

 signifies the probability density function,

 corresponds to class m prior probability information, and

 denotes the normalization constant for MAP calculation.

For a given *c* training sets from *c* classes, the total number of samples is given by,



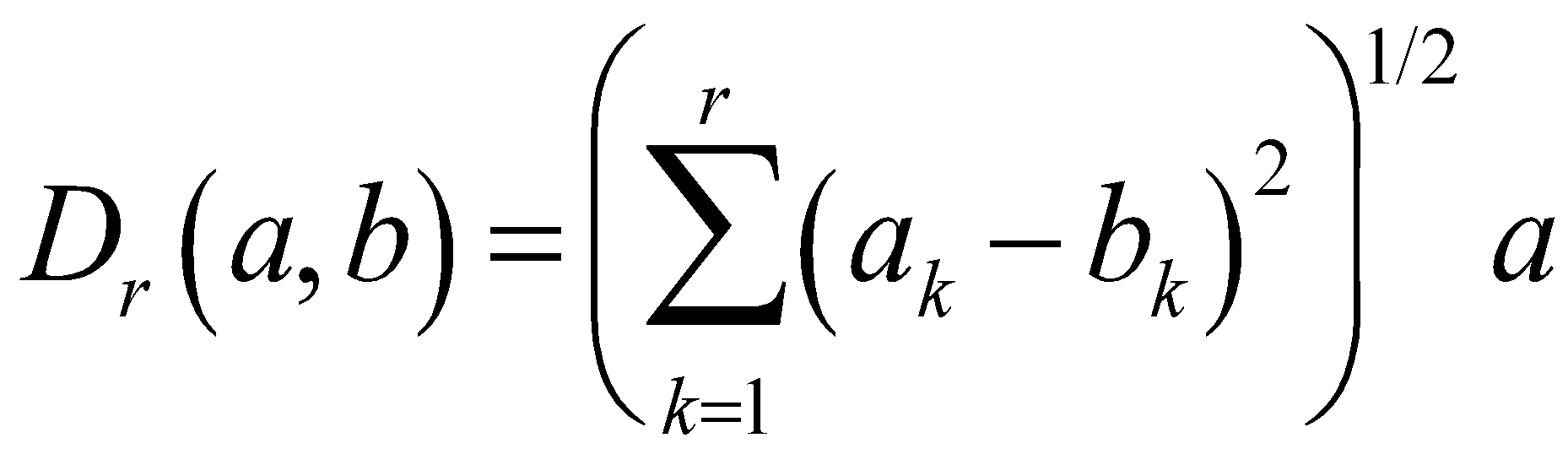
Therefore, the KNN decision rule specifies to “look in a neighborhood of the unknown feature vector for *k* samples” (1). In that neighborhood range, if more of the samples belongs to class i compared to other classes in the data set, the rule says to assign the class of i to that unknown

However, there are some problems associated with using KNN as classifier. One of the problems is regarding type of distance metric to use for calculation of k “nearest” neighbors. The use of Euclidean metric for distance measure seems reasonable, but it brings the “computation burden and massive storage burden” to the application process (1). The computational time and storage space problems are due to the need of computing the distance from the unknown to all the neighbors. As stated in the lecture’s slide, the methods such as, “computing partial distances, prestructuring, and editing the stored prototypes” could be used to reduce the computational burden by significant amount (1). This project used partial distance calculation method along with original distance calculation method to find the k nearest neighbor for the given unknown sample.

**Partial Distance Calculation**

The idea here is to calculate the distance by adding one feature dimension at a time to the subset of r of the full d dimension. Then, we check the value of the calculated partial distance if it is too large, we leave the further calculation process and we don’t include that sample in consideration of k nearest neighbor. This means that the knowledge about the “subspace is indicative of the full space” (1).

The Euclidian distance metric is given by,



**Distance Metrics and KNN**

The KNN classifier is based on “a metric or “distance” function between patterns” (2). The most commonly used distance metric is Euclidian distance in d dimensions. However, the other distance metric can be used to solve the problems associated with KNN as a classifier. “A metric D(·, ·) is merely a function that gives a generalized scalar distance between two argument patterns” (2). And, all of the metric follows the following four properties,

Non-negativity: D(a,b) >=0 i,e distance between two vector is always a positive number

Reflexivity: (D(a,b) = 0 if and only if a=b i,e distance between two vectors is zero

Only if the two vectors are identical

Symmetry: (D(a,b) = D(b,a)) i,e the order of the vector doesn’t matter in

Calculating the distance between the two vectors

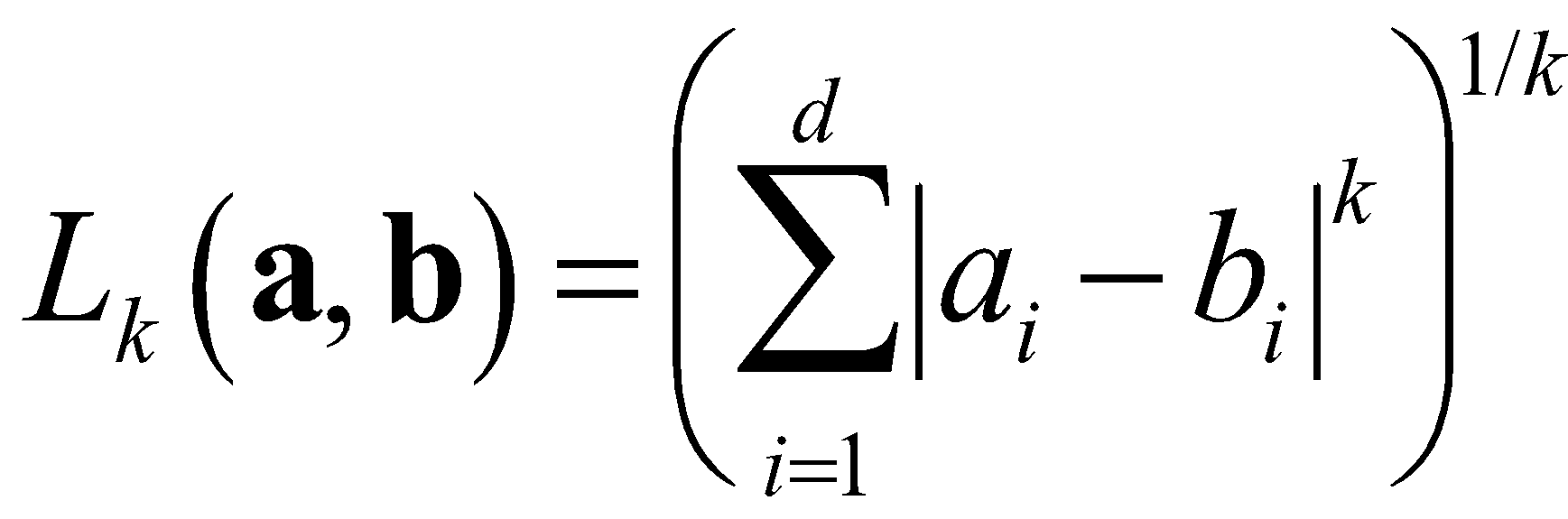
Triangle inequality (D(a,b) + D(b,c) >= D(a,c)) i,e sum of the distance between

the two pair of the vectors is greater than the distance between the one of those vectors pairs

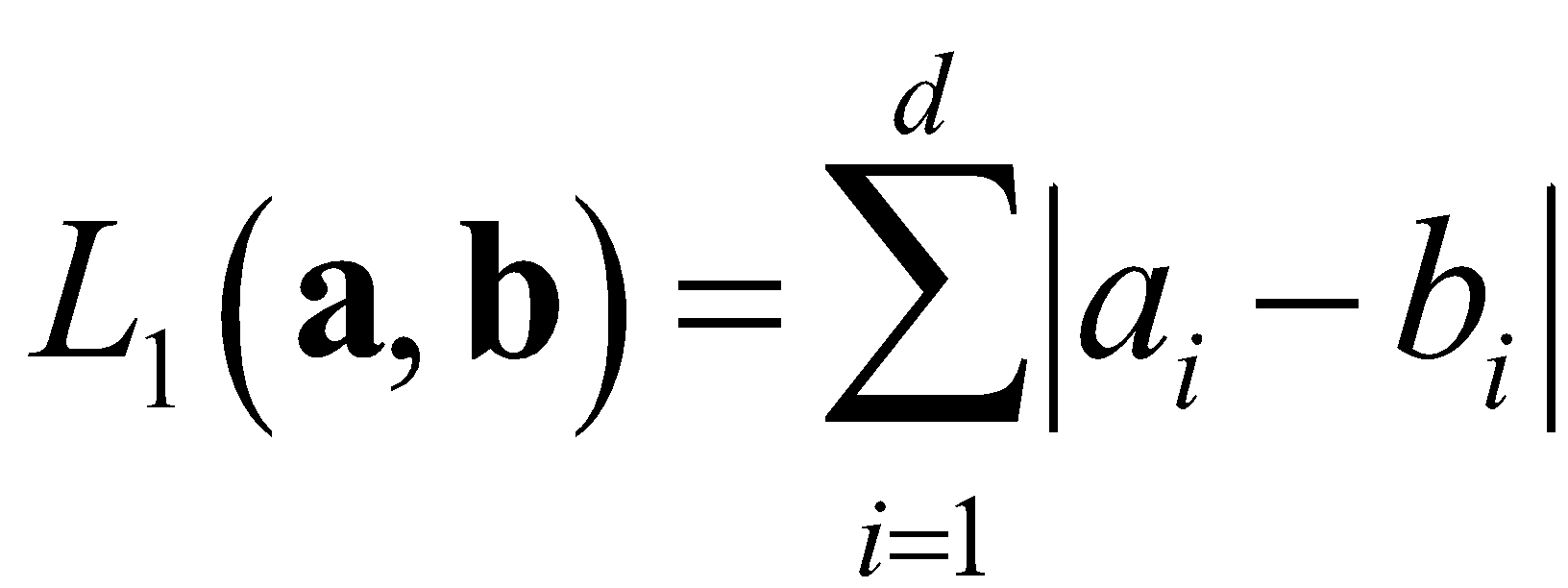
**Minkowski Distance Metric**

We also explored the option of calculating both the original and the partial distance using Minkowski distance metric for d-dimension patterns.

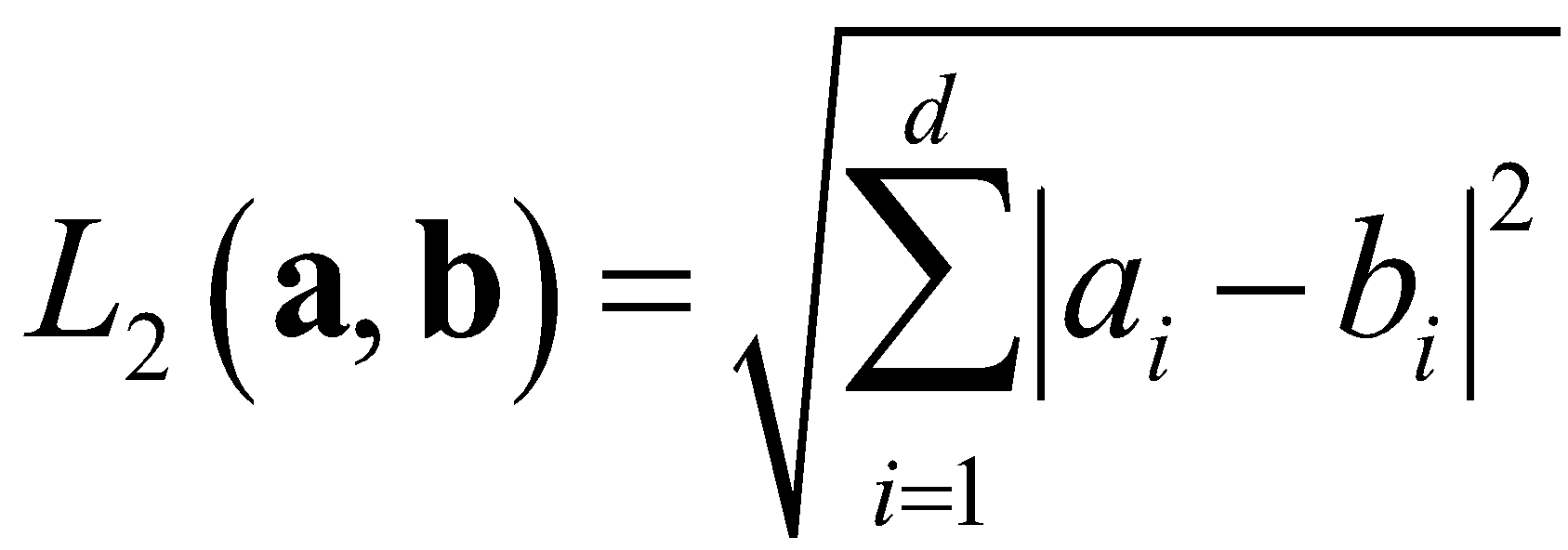
The Minkowski distance metric is given by,

* 1. 

The distance metric corresponding to k=1 is known as Manhattan distance (city block distance), and is given by,

* + 1. 

Similarly, for k=2, the distance is called Euclidean distance, and is given by,

* + 1. 

We tested different values of k values for Minkowski distance in the program with different k values for KNN as classifier.

**Performance Evaluation**

For the performance evaluation of the kNN as classifier, we used both the computational time calculation and the accuracy measurement. We calculated the performance metric such as true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Then we used these four values to calculate the Accuracy metric, which is the probability of true-decision, and is given by,

Accuracy = (TP + TN) / (TP + TN + FP + FN)

**Cross Validation**

The project also used cross-validation method to evaluate the performance of the KNN as the classifier. We applied this method along with KNN on “fglass.dat” synthetic data set with nine features and seven class categories. We used “flass.grp” that has already defined groups for 10-fold cross-validation for the “fglass.dat” data set. Then, from ten-divided data set, one set is used as validating set and all other sets are used as training set to build kNN as classifier. So, the whole process is repeated 10 times with each time different data set as validation set and different set as training set. In each run the error rate is calculated by calculating the number of misclassified samples in the validating set. Then, at the last we also calculated the average error of the 10 fold cross-validation evaluation process.

In this project, we also yy yg yy used clustering the unsupervised learning technique to find the best representation of the given *true* full-color image using *indexed* less number of colors. We used K-means and winner-take-all algorithms for the clustering purpose and for the compression of the full-color image. In this project the number of cluster to have in the data set is predetermined. Also the Euclidean distance metric is used for the calculation of the nearest cluster center for each sample in the data set.

Euclidean distance:



Both the K-mean and the winner-take-all algorithms begin with the random selection of the initial cluster centers or centroids from the given data set according to the cluster number being tested for.

**K-Means**

After the random selection of the cluster centers, for each sample in the data set it calculates the Euclidean distance between the sample and each of the selected cluster centers. Then based on the nearest cluster center found it assigns samples to that nearest clusters. After processing each sample in the data set, it computes the sample mean of each cluster and then used that cluster centers for the reassignment of each sample to the cluster with the nearest mean. The classification process stops if the classification of all samples has not changed in the current iteration through the data set. Otherwise, it continues the calculation of cluster mean for new cluster centers and then reassignment of the samples for epochs.

**Winner-Take-All (Online update)**

This algorithm also begins with an arbitrary set of cluster centers ωi. Then for each sample **x**, in the data set it calculates the Euclidean distance between that sample and each cluster centers in the set of cluster centers. Then it finds the nearest cluster center ωα, which is called the winner. As soon as the winner is identified the cluster center is updated using the formula,

ωαnew = ωαold + ε(**x**- ωαold)

where ε is known as a “learning parameter”.

Typical values of this parameter are small, on the order of 0.01.

These updated cluster centers are then used for the next iteration through the data set. Also the learning rate is decreased at the end of the iteration through the data set to guarantee the convergence of the algorithm. The processing of the assignment of the sample to the cluster center is continued until some consecutive epoch when the assignment to the cluster center doesn’t change.

**Experiments and results**

**Discussion**

**References**

Hairong Qi “ECE 471/571 – Pattern Recognition Fall 2015”. Lecture Slides. December. 3. 2015.

<http://web.eecs.utk.edu/~qi/ece471-571/syllabus.htm>

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

<http://archive.ics.uci.edu/ml/datasets/Parkinsons>

Little, M. A., McSharry, P. E., Roberts, S. J., Costello, D. A., & Moroz, I. M. (2007). Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. *BioMedical Engineering OnLine*, *6*(1), 23.

Little, M. A., McSharry, P. E., Hunter, E. J., Spielman, J., & Ramig, L. O. (2009). Suitability of dysphonia measurements for telemonitoring of Parkinson's disease. *Biomedical Engineering, IEEE Transactions on*, *56*(4), 1015-1022.

Richard O. Duda, Peter E. Hart, David G. Stork “Pattern Classification.” 2nd ed. New York: Wiley, 2001. Print.

**Appendix**

**1. Source code:** testMpp.cpp to perform the tasks of project 4

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* ECE 471 : Project 4

\*

\* testMpp.cpp - file to performs the task of Project 4

\* Last Modified by/Date: Niloo Ranjan, 11/23/2015

\* Added the code for implementation of K-means and Winner-take-all unsupervised learning methods

\* case 1: for K-means algorithm implementation

\* case 2: for winner-take-all algorith implementation

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include <string>

#include "Matrix.h"

#include "Pr.h"

#include <ctime>

using namespace std;

#define Usage "Usage: ./testMpp flowers.ppm cases \n\t flowers.ppm: the original image file\n\t cases can be 1 or 2\n\n"

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

{

Matrix XTr; // to read original image into a matrix

// check to see if the number of argument is correct

if (argc < 3) {

cout << Usage;

exit(1);

}

int cases;

int nrow = -1; // for number of rows in the original image

int ncol = -1; // for number of column in the original image

int row; // for rows in image matrix

int col; // for column in image matrix

// number of clunters center to test with

// int numclusters = 256;

// int numclusters = 128;

//int numclusters = 64;

int numclusters = 32;

cases = atoi(argv[2]);

// read the image into a matrix

XTr = readImage(argv[1], &nrow, &ncol);

row = XTr.getRow();

col = XTr.getCol();

//writeData(XTr,"image.tr");

writeImage("fig1.ppm", XTr,nrow, ncol);

Matrix centers(numclusters, col); // for random centroids to start with

Matrix newcenters(numclusters, col); // for updated centroids in case of WTA

// Matrix newData;

Matrix clusterAssign(row, 1); // assigned clusters to the test samples

// time the convergence of the algorithms

clock\_t start = clock();

// get the random cluster centers from the test data corresponponding

// to the number of cluster tested in the program

for ( int i=0; i<numclusters; i++)

{

int randSamp = (rand ( ) % row ) ;

for (int j=0; j < col; j++)

{

centers(i,j) = XTr(randSamp, j);

}

}

//

int done = 0;

int epoc = 0;

// for the clustering using K-means

if(cases ==1)

{

Matrix samp( 1 , col);

Matrix samp1(1, col);

clusterAssign.initMatrix(-1);

// stopping creteria when there is not change in the cluster center assigment to all of the test

// data set

while ( !done)

{

cout << "epoc: " << epoc << endl;

epoc++;

done = 1;

int index;

// for each test sample in the data set compare its Euclidean distance to

// each of the cluster center and find the cluster center with the minimum

// distance

for ( int i = 0; i < XTr.getRow(); i++ )

{

double sumDistance = 0.0;

double min;

int j;

for ( int h =0; h < col; h++)

{

samp(0,h) = XTr(i,h);

}

for ( j =0 ;j < centers.getRow(); j++)

{

for ( int h1 =0; h1 < col; h1++)

{

samp1(0,h1) = centers(j,h1);

}

for ( int k =0; k < samp.getCol(); k++)

{

sumDistance = sumDistance + pow((samp(0,k) - samp1(0,k)), 2);

}

sumDistance = sqrt(sumDistance);

if ( j == 0)

{

min = sumDistance;

index = j;

}

else

{

if ( sumDistance < min )

{

min = sumDistance;

//l = j;

index = j;

}

}

}

// if the found minimum distance cluster center if not the same as what is already

// assigned cluster center to the test sample then save that cluster as the nearest cluster

// center for the sample

if ((int)clusterAssign(i, 0) != index)

{

// set the flag that the change has been made in the cluster assignment

done = 0;

clusterAssign(i, 0) = index;

}

}

// exit the nearest cluster finding loop when there is

// no change in cluster assignment

if(done)

{

done = 1;

break;

}

int val;

int num =0;

int col = 3;

Matrix sum(1, col);

// Compute the sample mean of each cluster

// and that will be the cluster centers to compare with

// for the reassignment of each sample to the cluster with the nearest mean

for ( int i = 0; i < numclusters; i++)

{

sum.initMatrix(0.0);

num = 0;

for ( int j = 0; j < clusterAssign.getRow(); j++ )

{

if ((int)clusterAssign(j,0) == i )

{

for (int k=0; k< col; k++)

{

sum(0, k) = sum(0,k) + XTr(j,k);

}

num++;

}

}

if(num !=0)

{

for (int k = 0; k < col; k++)

{

centers(i,k) = sum(0, k) / num;

}

}

}

}

Matrix newData1(row,col);

// change the value of the data set to corresponding to the value

// of the assigned cluster centers that they belong to

for (int i=0; i < row; i++)

{

int r;

r = clusterAssign(i,0);

for (int j =0; j < col; j++ )

{

newData1(i,j) = centers(r, j);

}

}

clock\_t end = clock();

cout << "Running Time: " << (double) (end-start)/ 1000000 << " seconds" << endl;

// writeData(newData1, "data32NewKM.tr");

writeData(newData1, "data64NewKM.tr");

// writeData(newData1, "data128NewKM.tr");

// writeData(newData1, "data256NewKM.tr");

writeData(XTr, "originalData.tr");

// writeImage("figout32Kmean.ppm", newData1, nrow, ncol);

writeImage("figout64Kmean.ppm", newData1, nrow, ncol);

// writeImage("figout128Kmean.ppm", newData1, nrow, ncol);

// writeImage("figout256Kmean.ppm", newData1, nrow, ncol);

}

// for the implementation of the winner-take-all algorithm

else if (cases == 2)

{

Matrix samp( 1 , col);

Matrix samp1(1, col);

double LR = 0.1;

int g = -1;

int p = 0;

int valuetobepushed = -1;

int s = 0;

int success = 0;

Matrix tenResults(1,10);

clusterAssign.initMatrix(-1);

// if there is no change in the cluster center

// assignment for 10 epoch although the learning rate has been

// decreasing then the clustering is all done

while ( !success)

{

epoc++;

cout << "epoc: " << epoc << endl;

cout << "learning rate: " << LR << endl;

done = 1;

int index;

// for each test sample in the data set compare its Euclidean distance to

// each of the cluster center and find the cluster center with the minimum

// distance

for ( int i = 0; i < XTr.getRow(); i++ )

{

double sumDistance = 0.0;

double min;

int j;

for ( int h =0; h < col; h++)

{

samp(0,h) = XTr(i,h);

}

for ( j =0 ;j < centers.getRow(); j++)

{

for ( int h1 =0; h1 < col; h1++)

{

samp1(0,h1) = centers(j,h1);

}

for ( int k =0; k < samp.getCol(); k++)

{

sumDistance = sumDistance + pow((samp(0,k) - samp1(0,k)), 2);

}

sumDistance = sqrt(sumDistance);

if ( j == 0)

{

min = sumDistance;

index = j;

}

else

{

if ( sumDistance < min )

{

min = sumDistance;

index = j;

}

}

}

// if the found minimum distance cluster center if not the same as what is already

// assigned cluster center to the test sample then save that cluster as the nearest cluster

// center for the sample

if (((int)clusterAssign(i, 0) != index))

{

g++;

// set the flag that the change has been made in the cluster assignment

done = 0;

clusterAssign(i, 0) = index;

// the winner has identified so update the winner cluster center value

for ( int l =0; l<col; l++)

{

newcenters(index,l) = centers(index,l) + (LR \* ((XTr(i,l) - centers(index,l))));

}

}

}

// at the end of the one epoch the found updated winner cluster centers

// are the cluster centers to be compared with for next epoch

centers = newcenters;

// decrease the learning rate to guarantee the convergence of the algorithm

LR = 0.9 \* LR;

// keep track of the ten consecutive epoch for which the

// cluster assignment has not changed even though the

// learnig rate has been decreasing

if ( done != 0)

{

valuetobepushed=1;

}

else {

valuetobepushed=0;

}

for (int x=0; x <9; x++){

tenResults(0,x) = tenResults(0,x+1);

}

tenResults(0,9) = valuetobepushed;

s =0;

for ( int h =0; h < 10; h++)

{

s += tenResults(0,h);

}

if ( s==10){

success = 1;

}

}

// change the value of the data set to corresponding to the value

// of the assigned cluster centers that they belong to

Matrix newData1(row,col);

for (int i=0; i < row; i++)

{

int r;

r = clusterAssign(i,0);

for (int j =0; j < col; j++ )

{

newData1(i,j) = centers(r, j);

}

}

clock\_t end = clock();

cout << "Running Time: " << (double) (end-start)/ 1000000 << " seconds" << endl;

writeData(newData1, "data32NewWTA.tr");

// writeData(newData1, "data64NewWTA.tr");

// writeData(newData1, "data128NewWTA.tr");

// writeData(newData1, "data256NewWTA.tr");

writeData(XTr, "originalData.tr");

writeImage("figout32WTA.ppm", newData1, nrow, ncol);

// writeImage("figout64WTA.ppm", newData1, nrow, ncol);

// writeImage("figout128WTA.ppm", newData1, nrow, ncol);

// writeImage("figout256WTA.ppm", newData1, nrow, ncol);

}

else

{

cout << "Please enter case 1 or 2 to run either K-means or WTA" << endl;

}

return 0;

}

2. Matlab code : for the calculation of mean-squared error and the difference between the image

% ECE471/571 project 4

% Niloo Ranjan

% 11/23/2015

% to calculate mean-squared error

% and difference between the original

% and the compressed image

% also to display the compressed image

% clear the figure

clf;

%load data32NewWTA.tr;

load originalData.tr;

A1 = originalData;

load data32NewWTA.tr;

B1 = data32NewWTA;

%load data64NewWTA.tr;

%B1 = data64NewWTA;

%load data128NewWTA.tr;

%B1 = data128NewWTA;

%load data256NewWTA.tr;

%B1 = data256NewWTA;

%load data32NewKM.tr;

%B1 = data32NewKM;

%load data64NewKM.tr;

%B1 = data64NewKM;

%load data128NewKM.tr;

%B1 = data128NewKM;

%load data256NewKM.tr;

%B1 = data256NewKM;

A = imread('flowers.ppm');

% = imread('figout256WTA.ppm');

%B = imread('figout128WTA.ppm');

%B = imread('figout64WTA.ppm');

B = imread('figout32WTA.ppm');

%B = imread('figout32Kmean.ppm');

%B = imread('figout64Kmean.ppm');

%B = imread('figout128Kmean.ppm');

% = imread('figout256Kmean.ppm');

imshowpair(A, B, 'diff')

%imshow(A);

%imshow(B);

err = immse(A1, B1);

fprintf('\n The mean-squared error is %0.4f\n', err);