**EECS**

**University of Tennessee**

**Pattern Recognition – ECE 571**

**Final Project – Vehicle Recognition Using Vehicle Silhouettes**

**Submitted by:**

Yazeed Alkhrijah                         Mahendra Duwal Shrestha

yalkhrij@vols.utk.edu         mduwalsh@vols.utk.edu

Abstract

This project uses pattern recognition procedures for vehicle recognition using vehicle silhouettes. The objective is to integrate different classification techniques to get best accuracy of classification.

Specific properties of shape features measures of 2D image can be studied to gain knowledge about which vehicle the image belongs to. The dataset was collected by JP Siebert at the Turing Institute, Glasgow, Scotland in 1986-87. The original purpose was to find a method of identifying 3D objects within a 2D image by application of combination of shape feature extractors to the 2D silhouettes of the objects.

Four types of vehicle were chosen for this project and images of painted black vehicles were captured with different angles, and used for feature extraction. The dataset was already provided for our project. The given dataset was then normalized and introduced to dimensionality reduction technique. The effects of dimensionality reduction and not using dimensionality reduction technique were measured using different types of classifiers on those datasets. Both supervised and unsupervised learning were used for classification purpose. Since just a single dataset was provided 3 fold cross validation evaluation technique was implemented for all classifiers using supervised learning. Best three high performance giving classifiers were selected for classifier fusion.

Three types of classifier fusion methods were implemented which were plurality, Naive Bayes and Behavior Knowledge Space. After implementing all three fusions, classification accuracy were measured for each fused classifier.

Introduction

The dataset we've chosen for this the project is vehicle silhouettes. Vehicle recognition has been used for decades now and been used in various fields in real world. Since shape and design of each brand of vehicle are different than others, silhouettes formed by different types of vehicle in 2D image have different properties depending upon the type of vehicle. Specific properties of shape features measures of the image can be studied to gain knowledge about which vehicle the image belongs to. So identifying vehicle type which is 3D based on the shape features extracted from 2D image with all possible angles forms a very good pattern classification problem.

Vehicle recognition has been used in different areas. It can be used for solving criminal cases. It can be used as tool for data mining or statistics of vehicle type in specific location. This can also be used in advertising market.

**Data set**

The data was originally collected by JP Siebert at the Turing Institute, Glasgow, Scotland in 1986-87. The original purpose was to find a method of identifying 3D objects within a 2D image by application of combination of shape feature extractors to the 2D silhouettes of the objects.

For the experiment, four "Corgie" model vehicles were chosen: a double decker bus, Chevrolet van, Saab 9000 and an Opel Manta 400, with the expectation that the bus, van and either one of the cars would be readily identifiable but would be more difficult to distinguish between the cars. The vehicles were painted black to minimize highlights and the images were captured by a camera looking downwards at model vehicle from fixed angle of elevation. The images were then threshold to produce binary vehicle silhouettes. The images of a vehicle were captured with different orientations of vehicle facing head on, rear on and rotating.

The data set consist of 18 different features for each sample extracted by the HIPS (Hierarchical Image Processing System) extension BINATTS from the silhouettes. This extracts a combination of scale independent features utilizing classical moments based measures and heuristic measures.

|  |  |
| --- | --- |
| Feature | How it is extracted |
| Compactness | (average perim)^2/area |
| Circularity | (average radius)^2/area |
| Distance circularity | area/(av.distance from border)^2 |
| Radius ratio | (max.radius-min.radius)/average radius |
| Pr.axis aspect ratio | (minor axis)/(major axis) |
| Max.length aspect ratio | (length perp. max length)/(max length) |
| Scatter ratio | (inertia about minor axis)/(inertia about major axis) |
| Elongatedness | area/(shrink width)^2 |
| Pr.axis rectangularity | area/(primary axis length\*primary axis width) |
| Max.length rectangularity | area/(max.length\*length) |
| Scaled variance along major axis | (2nd order moment about minor axis)/area |
| Scaled variance along minor axis | (2nd order moment about major axis)/area |
| Scaled radius of gyration | (major var+minor var)/area |
| Skewness about major axis | (3rd order moment about major axis)/sigma\_min^3 |
| Skewness about minor axis | (3rd order moment about minor axis)/sigma\_maj^3 |
| Kurtosis about major axis | 4th order moment about major axis)/sigma\_min^4 |
| Kurtosis about minor axis | 4th order moment about minor axis)/sigma\_min^4 |
| Hollows ratio | (area of hollows)/(area of bounding polygon) |

Table1: explanation for feature extraction

Technical Approach

The dataset chosen had 18 features. The data was normalized before using. Though it’s not quite large number of dimension, the dataset was introduced to dimension reduction techniques FLD and PCA which reduced number of dimensions to 3 and 6 respectively. Since only one dataset was provided, for cross validation purpose the dataset was divided into three folds and different classification techniques were used on three folds datasets following supervised learning while unsupervised learning used the dataset as a whole. The best performing three classifiers were then chosen to integrate and generate fused result lookup table. Three different techniques of fusion were implemented to build fused classifier. The dataset was classified using the fused lookup table and accuracy of classification was measured.

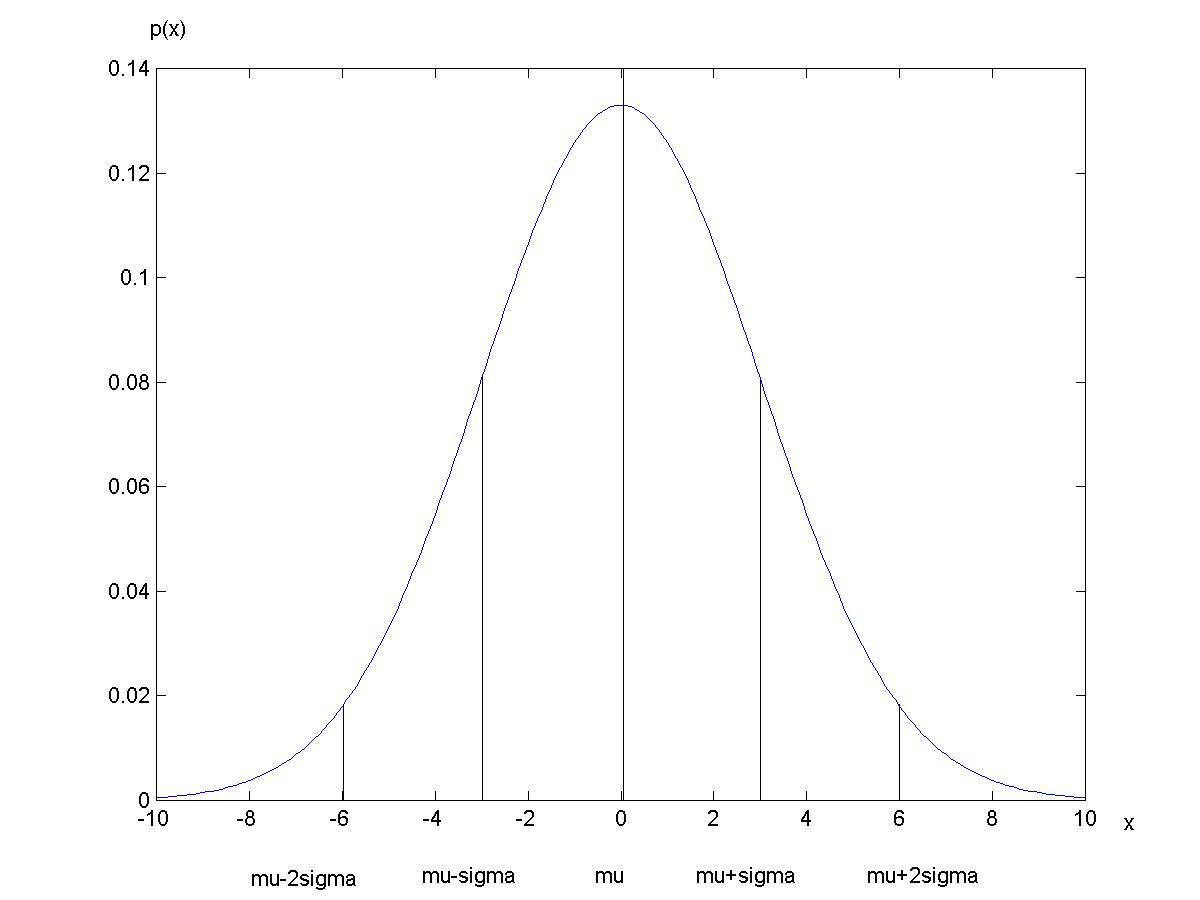
**Gaussian Density Distribution:**

Figure1: Gaussian distribution

Multivariate Gaussian density is given by:



When d=1,

**Fisher’s Liner Discriminant (FLD) method**

It is one of commonly used dimension reduction methods which tries to discriminate transformed data best possible way. This method tries to project mean vector of each class as further as possible and minimize scatterings of sample data within each class as best as possible. Since this uses class information of training samples, it is supervised learning. Using this method, number of dimensions d can be reduced to c-1 where c is number of classes and is supposed less than d. If we consider two class case, then data of d dimension is projected onto a line. So we want to find projection vector ‘**w**’ such that the data can be best separated. And projected data points are given by,



And



is within class scatter matrix and given by



And **m1** and **m2** are mean of each class.

For more than two classes case,

Between mean scatter matrix is given by,



And solving v = λ v (generalized eigenvalue problem),

will at most give c-1 distinct eigenvalues and thus we can obtain projection vector.

**Principal Component Analysis (PCA) method**

It is also known as K-L transform. This method tries to represent the data in best possible way. It finds a new feature space (m- dimensional) that is sufficient to describe data in original feature space where m<d.

A vector **x**described in terms of a set of basis vectors **b***i*.



The basis vectors (**b***i*) should be linearly independent and orthonormal, that is,



If we want to consider m (m<d) components of **y** and still represent **x** though with some error, we will calculate the first *m* elements of **y** and replace the others with constants.



Error:

Instead of plain error, mean square of error is used to quantify error and can be expressed as

Eigenvectors of covariance matrices of all classes are optimal basis vectors. Eigenvectors can be sorted corresponding to its eigenvalues in descending order and eigenvector with largest eigenvalue is principal component. Dimensions of data can be reduced to m from d by omitting eigenvectors corresponding to eigenvalues whose total sum (summing from smallest to larger eigenvalue) is not than error tolerance.

**CrossValidation**

Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give an indication of how well the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on new data. This is the basic idea for a whole class of model evaluation methods called cross validation.

**n-fold cross validation**

The dataset is divided into n different sets and training and testing method is carried out for n times where one set will be testing set and other n-1 set will be training. This is carried out for n different sets of data as testing set and performance accuracy is averaged.

**Maximum Posterior Probability (MPP):**

Bayesian decision formula for MPP is given by:

Where P(ωj|x) is posterior probability; p(x| ωj) is conditional probability density function or likelihood; p(x) is normalization constant and is given by:

P(ωj) is prior probability.

Discriminant function for multivariate Gaussian density can be represented as:



**Case I:**

If features of sample data are statistically independent and have same variance, then we’ll have



where

The Euclidean distance is also called minimum distance because it is the distance of x from each of the c mean vectors. And when  is same for all classes the discriminant function is measuring minimum distance.



**Case II:**

If covariance matrices of all classes are same and not equal to scalar multiplication of identity matrix, then and discriminant function will be:



**Case III:**

If covariance matrices of all classes are arbitrary, then discrimanant function will be:



**kNN(k- Nearest Neighbors)**

To estimate *p*(*x*) from *n* samples, we can center a cell at *x* and let it grow until it contains *kn* samples, and *kn* can be some function of *n*

Normally, we let

Given *c* training sets from *c* classes, the total number of samples is

Given a point **x** at which we wish to determine the statistics, we find the hypersphere of volume ***V*** which just encloses *k* points from the combined set. If within that volume, *km* of those points belongs to class *m*, then we estimate the density for class *m* by





Using Bayesian decision rule and posterior probability,



So from the above posterior probability expression, we can derive a decision rule which tells us to look in a neighborhood of the unknown feature vector for *k* samples. If within that neighborhood, more samples lay in class *i* than any other class, we assign the unknown as belonging to class *i*. This is using majority voting principle.

For distance calculation, most commonly used is Euclidean distance metric while we can also use Minkowski distance.

In the project, distance from a sample data to all data in training set was calculated and sorted in ascending order. Then picking only k of the nearest distances, majority of class of training data in k samples was identified.

**BackPropagation**

Back propagation is feed forward neural network algorithm where feedback is only used during training of network. It is a method of trainingneural network used in conjunction with an optimization method, which is least mean square in our case.

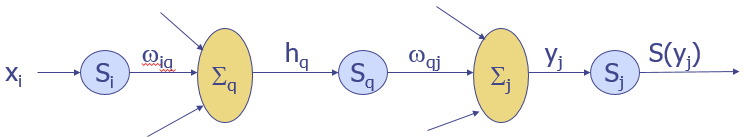


Figure2: backpropagation three layer network

Where the objective function is ,

The backpropagation learning rule is based on gradient descent where the weights are initialized with random values, and changed in a direction that will reduce the error to find weight values.



Back propagation uses sigmoid function as threshold function which is continuous and derivable.



**Decision Tree**

Decision tree is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees and tree models where target variable can take continuous values are called regression trees. The term Classification And Regression Tree (CART) analysis is a term used to refer to both classification and regression procedures.

Decision tree learning constructs a decision tree from class labeled training samples. In a decision tree, each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.

Since it is more expressive and simple in training, most of the decision trees are binary tree. We seek property query at each node that makes data reaching the immediate leaf nodes contain as less impurity as possible.

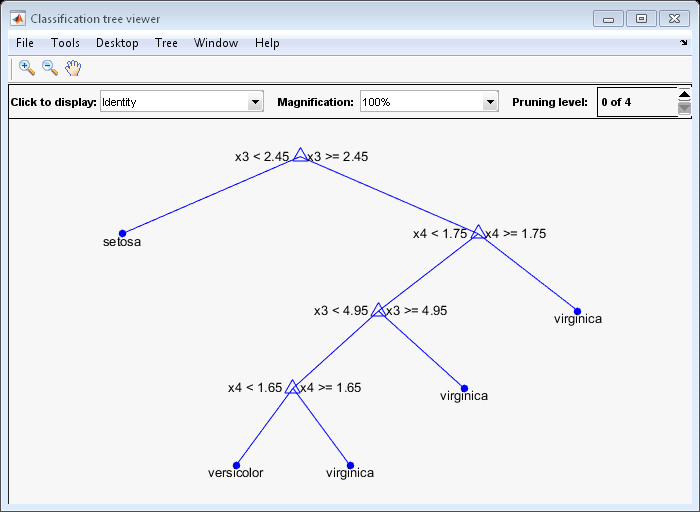


Figure3: Decision tree example

**Support Vector machine (SVM)**

Support Vector Machines are motivated by training the linear machines with margins, but rely on preprocessing the data to represent patterns in a high dimension — typically much higher than the original feature space. In this project, we used predefined function to apply classification by using SVM for multiclass data set.



To apply SVM:

For Non-separable Cases we can use:

1. SVM with soft margin
2. Kernel trick

**k-means**

k-means is one of the clustering algorithms which assumes there are k clusters and starts by initializing cluster centers to arbitrary values.

Step 1: Begin with an arbitrary set of cluster centers and assign samples to nearest clusters

Step 2: Calculate sample mean of each cluster using samples in each cluster

Step 3: Reassign each sample to the cluster with the nearest cluster mean

Step 4: If assignment of sample to cluster does not change cluster, then stop; else repeat from step 2

**Winner takes all**

Winner takes all is improvement in k-means algorithm to increase speed of convergence. The winner (cluster center which is nearest to a sample) is updated on the fly immediately after a sample is classified to the cluster by pulling cluster towards the sample.

Step 1: Begin with an arbitrary set of cluster centers*i*

Step 2: Calculate sample mean of each cluster using samples in each cluster

Step 3: For each sample **x**, find the nearest cluster center, which is called the winner and assign the sample to the cluster

Step 4: Modifyusingnew = old + (**x**- old) whereis learning parameter and typically small value in order of 0.001

Step 5: If assignment of sample to cluster does not change cluster, then stop; else repeat from step 2



x

x-

**Kohonen Feature Map**

It is an extension of the winner-take-all algorithm and is also called self-organizing feature maps. In this algorithm, a problem dependent topological distance is assumed to exist between each pair of cluster centers. When the winner cluster center is updated so are other cluster centers as its neighbors in the sense of topological distance.



x

x-









The winning cluster center and its neighbors are trained based on the following formula:

are cluster centers

As k increases, decreases. (However, for this project it is chosen small value constant)

are coordinates of cluster centers

are coordinates of winning cluster center

The last expression tells that closer the neighbors are in topology, more affected they are.

**Classifier fusion**

Classifier fusion is integration of different classifiers for classifying a dataset. A classifier fusion is applied to get better accuracy, which means getting the classification result from multiple classifiers instead of only one and use all the results to generate one. In this project, we applied three classification fusion algorithms, which are majority voting, Naïve Bayes and behavior knowledge space.

**Majority voting**

There are three methods for applying majority voting:

1. Unanimity (100%)

To classify a sample to a certain class we have to have an agreement between different classifiers used for fusion.

1. Simple majority (50%+1)

To classify a sample to a certain class we have to have more than 50%+1 of the classifiers are agree to this class.

1. Plurality (most votes)

We pick the highest selected class as classification results.

**Naïve Bayes**

In Naïve Bayes,it is assumed classifiers are independent of each other.Confusion matrix can be generated based on known label of samples and to which class label it was classified by classifier. Then Bayesian decision rule can be applied as a decision rule to classify each sample based on confusion matrix:



Where

L classifiers, i=1,..,L

c classes, k=1,…,c

si: class label given by the ith classifier, i=1,…,L, s={s1,…,sL}.

**Behavior knowledge space**

In this algorithm,for a particular combination of output labels from each classifier, number of samples from actual class of each sample is calculated. Then the voting goes to the class containing maximum number of samples for that combination of output labels from different classifiers.And a lookup table can be generated for fused result of classifiers.

Experiments and Results

The dataset was first normalized and then FLD was used to reduce dimensions from 18 to only 3. PCA was also performed on normalized with tolerance of 0.1 which resulted to reduced dimensions of 6.

All three datasets from normalization, FLD and PCA were each divided into three folds for better evaluation of performance of classifiers.

**Maximum posterior probability(MPP)**

After applying MPP for the three folds consist of normalized data set, we get the following error rate:

1. Case I

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.475177 |
| 2 | 0.507092 |
| 3 | 0.453901 |
| Average error rate=0.4787 | |

Table 2: Error rate from mpp case I

1. Case II

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.230469 |
| 2 | 0.191489 |
| 3 | 0.212766 |
| Average error rate=0.21157 | |

Table 3: Error rate from mpp case II

**Comments**:

We couldn’t perform mpp case III for normalized data that because the dimensions of the data set are 18 and the code couldn’t find the determinant of 18 by 18 covariance matrix. So, we chose case II as best case, and we performed mpp for reduced dimension data set by using case 2.

MPP for reduced dimension data set:

1. Case II for dimension reduced data set by using PCA

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.47447 |
| 2 | 0.482269 |
| 3 | 0.471631 |
| Average error rate=0.4761 | |

Table 4: Error rate from mpp case II for reduced data set by using PCA

1. Case II for dimension reduced data set by using FLD

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.3014 |
| 2 | 0.3085 |
| 3 | 0.33333 |
| Average error rate=0.3144 | |

Table 5: Error rate from mpp case II for reduced data set by using FLD

**Comments**:

After performing MPP for normalized data set and reduced dimension data set, we found the normalized data set gave us the best accuracy.

**KNN**

To accelerate the performance we used KNN with partial distance. After applying KNN for deferent K we got the following results:

1. Normalized data set fold1:

|  |  |
| --- | --- |
| K value | Error rate |
| 1 | 0.28 |
| 3 | 0.28 |
| 10 | 0.32 |
| 15 | 0.29 |
| 23 (the square root of the data set) | 0.32 |

Table 6: the error rate for fold1 after applying KNN

1. Normalized data set fold2:

|  |  |
| --- | --- |
| K value | Error rate |
| 1 | 0.28 |
| 3 | 0.30 |
| 10 | 0.28 |
| 15 | 0.30 |
| 23 (the square root of the data set) | 0.30 |

Table 7: the error rate for fold1 after applying KNN

1. Normalized data set fold3:

|  |  |
| --- | --- |
| K value | Error rate |
| 1 | 0.26 |
| 3 | 0.28 |
| 10 | 0.28 |
| 15 | 028 |
| 23 (the square root of the data set) | 0.31 |

Table 8: the error rate for fold1 after applying KNN

**Comments**:

The result from different K’s are close. We chose 3 as a best value for K, so we decided to apply KNN with K=3 for reduced dimension data set.

For Reduced dimension data set we get the following results:

1. KNN with K=3 for dimension reduced data set by using PCA

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.361 |
| 2 | 0.361 |
| 3 | 0.368 |
| Average error rate=0.363 | |

Table 9: Error rate from KNN with K=3 for reduced data set by using PCA

1. KNN with K=3 for dimension reduced data set by using FLD

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.28 |
| 2 | 0.30 |
| 3 | 0.28 |
| Average error rate=0.363 | |

Table 10: Error rate from KNN with K=3 for reduced data set by using FLD

**Comments**:

After applying KNN for reduced dimension data set by using PCA, the accuracy for the classifier was reduced and the error rate was increased compared to results for KNN for normalized data set. However, the results from reduced dimension data set by using FLD were almost the same with normalized data set.

**Back Propagation**

For normalized data set:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.2092 |
| 2 | 0.2021 |
| 3 | 0.2128 |
| Average error rate=0.2080 | |

Table 11: Error rate from PB using normalized data set

For PCA:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.4539 |
| 2 | 0.4291 |
| 3 | 0. 5177 |
| Average error rate=0.4669 | |

Table 12: Error rate from PB for reduced data set by using PCA

For FLD:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.2553 |
| 2 | 0.2695 |
| 3 | 0.2624 |
| Average error rate=0.2624 | |

Table 13: Error rate from PB for reduced data set by using FLD

**Comments**:

From the classification accuracy results, back propagation seemed to have higher performance for normalized dataset.

**Decision Tree**

For normalized data set:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.2872 |
| 2 | 0. 2943 |
| 3 | 0. 3050 |
| Average error rate=0.2955 | |

Table 14: Error rate from Decision tree using normalized data set

For PCA:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.3865 |
| 2 | 0.3865 |
| 3 | 0.3794 |
| Average error rate=0.3841 | |

Table 15: Error rate from Decision tree for reduced data set by using PCA

For FLD:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.3298 |
| 2 | 0.3546 |
| 3 | 0.2908 |
| Average error rate=0.3251 | |

Table 16: Error rate from Decision tree for reduced data set by using FLD

**Comments**:

For decision tree, the best accuracy is 0.2955 and is for normalized dataset.

**SVM**

After applying SVM for normalized data set and reduced dimension data set we get the following results:

1. SVM for normalized data set:

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.2589 |
| 2 | 0.2943 |
| 3 | 0.2518 |
| Average error rate=0.26 | |

Table 17: Error rate from SVM

1. SVM for dimension reduced data set by using PCA

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.44 |
| 2 | 0.42 |
| 3 | 0.41 |
| Average error rate=0.42 | |

Table 18: Error rate from SVMfor reduced data set by using PCA

1. SVM for dimension reduced data set by using FLD

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.27 |
| 2 | 0.33 |
| 3 | 0.29 |
| Average error rate=0.29 | |

Table 19: Error rate from SVM for reduced data set by using FLD

**Comments**:

We got the best result for SVM from normalized data set by using FLD, and it’s high accuracy compared to other classifiers.

**k-means:**

It was tested on a single whole sample set.

For normalized data set:

Accuracy=0.6501

For FLD:

Accuracy=0.6478

For PCA:

Accuracy=0.6158

**Winner Takes All**

For normalized data set:

Accuracy=0.6454

For FLD:

Accuracy=0.6477

For PCA:

Accuracy=0. 6158

**Kohonen Map**

For normalized data set:

Accuracy= 0.6489

For FLD:

Accuracy=0. 5284

For PCA:

Accuracy=0. 6478

**For all unsupervised clustering methods, performance error rate was found way higher.**

**Classifier Fusion**

After experimenting and classifying with all classifiers, the best three classifiers for the project were MPP case II, back propagation and kNN with k=3, and for normalized dataset. So we opted to do fusion using output labels from those classifiers on normalized datasets.

**Majority Voting (Plurality)**

It used only simple algorithm that chooses label with highest occurrence in out of three classifier output labels. Using this, accuracy of integrated classifier was measured to be 0.2399.

**Naïve Bayes**

After we selected the best classifier, we constructed the confusion matrix for each of them:

1. Confusion Matrix from MPP case 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | 0 | 1 | 2 | 3 |
| 0 | 200 | 2 | 7 | 4 |
| 1 | 3 | 203 | 1 | 5 |
| 2 | 12 | 10 | 129 | 62 |
| 3 | 11 | 4 | 58 | 135 |

Table 20:MPP Confusion

1. Confusion Matrix from KNN with K=3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | 0 | 1 | 2 | 3 |
| 0 | 195 | 10 | 3 | 5 |
| 1 | 5 | 204 | 0 | 3 |
| 2 | 21 | 11 | 109 | 72 |
| 3 | 21 | 6 | 90 | 91 |

Table 21:KNN Confusion matrix

1. Confusion Matrix from Back Propagation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | 0 | 1 | 2 | 3 |
| 0 | 192 | 2 | 3 | 2 |
| 1 | 3 | 210 | 2 | 3 |
| 2 | 6 | 8 | 150 | 53 |
| 3 | 8 | 10 | 76 | 118 |

Table 22:BP Confusion matrix

Based on these results, we constructed the following lookup table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| MPP | KNN | PB | Results | MPP 1 | KNN | PB | Results |
| 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 0 | 1 | 2 |
| 0 | 0 | 2 | 0 | 2 | 0 | 2 | 2 |
| 0 | 0 | 3 | 0 | 2 | 0 | 3 | 3 |
| 0 | 1 | 0 | 0 | 2 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 2 | 1 | 1 | 1 |
| 0 | 1 | 2 | 2 | 2 | 1 | 2 | 2 |
| 0 | 1 | 3 | 3 | 2 | 1 | 3 | 2 |
| 0 | 2 | 0 | 0 | 2 | 2 | 0 | 2 |
| 0 | 2 | 1 | 2 | 2 | 2 | 1 | 2 |
| 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 0 | 2 | 3 | 3 | 2 | 2 | 3 | 2 |
| 0 | 3 | 0 | 0 | 2 | 3 | 0 | 2 |
| 0 | 3 | 1 | 3 | 2 | 3 | 1 | 2 |
| 0 | 3 | 2 | 2 | 2 | 3 | 2 | 2 |
| 0 | 3 | 3 | 3 | 2 | 3 | 3 | 3 |
| 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 3 | 0 | 1 | 3 |
| 1 | 0 | 2 | 2 | 3 | 0 | 2 | 3 |
| 1 | 0 | 3 | 2 | 3 | 0 | 3 | 3 |
| 1 | 1 | 0 | 1 | 3 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 |
| 1 | 1 | 2 | 1 | 3 | 1 | 2 | 2 |
| 1 | 1 | 3 | 1 | 3 | 1 | 3 | 3 |
| 1 | 2 | 0 | 2 | 3 | 2 | 0 | 3 |
| 1 | 2 | 1 | 2 | 3 | 2 | 1 | 3 |
| 1 | 2 | 2 | 2 | 3 | 2 | 2 | 2 |
| 1 | 2 | 3 | 2 | 3 | 2 | 3 | 3 |
| 1 | 3 | 0 | 2 | 3 | 3 | 0 | 3 |
| 1 | 3 | 1 | 1 | 3 | 3 | 1 | 3 |
| 1 | 3 | 2 | 2 | 3 | 3 | 2 | 3 |
| 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Table 23:Naïve Bayes lookup table

By using this lookup table we tested the Naïve Bayes performance and got the following results

|  |  |
| --- | --- |
| Fold | Error rate |
| 1 | 0.24822 |
| 2 | 0.2375 |
| 3 | 0.2269 |
| Average error rate=0.237 | |

Table 24:Error rate after using Naïve Bayes

**Comments**:

From the previous table, we can see the overall performance is improved that compared by using only one classifier.

**Behavior Knowledge Space (BKS)**

Classification output labels from all BP, MPP case II and KNN were used to generate lookup table for BKS.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PB | MPP | KNN | t1(0) | t2(1) | t3(2) | t4(3) | result |
| 0 | 0 | 0 | 166 | 0 | 7 | 5 | 0 |
| 0 | 0 | 1 | 6 | 1 | 0 | 0 | 0 |
| 0 | 0 | 2 | 3 | 0 | 1 | 1 | 0 |
| 0 | 0 | 3 | 3 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 3 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 34 | 1 | 0 | 1 |
| 0 | 1 | 2 | 0 | 0 | 1 | 0 | 2 |
| 0 | 1 | 3 | 0 | 0 | 0 | 0 | 3 |
| 0 | 2 | 0 | 4 | 0 | 4 | 3 | 2 |
| 0 | 2 | 1 | 1 | 0 | 1 | 0 | 2 |
| 0 | 2 | 2 | 0 | 0 | 3 | 3 | 3 |
| 0 | 2 | 3 | 1 | 0 | 3 | 3 | 3 |
| 0 | 3 | 0 | 3 | 1 | 2 | 2 | 0 |
| 0 | 3 | 1 | 1 | 1 | 2 | 1 | 2 |
| 0 | 3 | 2 | 0 | 0 | 3 | 6 | 3 |
| 0 | 3 | 3 | 0 | 0 | 0 | 2 | 3 |
| 1 | 0 | 0 | 18 | 0 | 0 | 2 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 2 | 0 | 0 | 0 | 0 | 3 |
| 1 | 0 | 3 | 1 | 0 | 3 | 1 | 2 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 2 |
| 1 | 1 | 1 | 0 | 92 | 3 | 1 | 1 |
| 1 | 1 | 2 | 0 | 0 | 2 | 1 | 2 |
| 1 | 1 | 3 | 0 | 0 | 1 | 2 | 3 |
| 1 | 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 0 | 1 | 3 |
| 1 | 2 | 2 | 0 | 0 | 3 | 5 | 3 |
| 1 | 2 | 3 | 0 | 0 | 3 | 3 | 3 |
| 1 | 3 | 0 | 0 | 0 | 2 | 5 | 3 |
| 1 | 3 | 1 | 0 | 2 | 2 | 2 | 3 |
| 1 | 3 | 2 | 0 | 0 | 5 | 8 | 3 |
| 1 | 3 | 3 | 0 | 0 | 2 | 20 | 3 |
| 2 | 0 | 0 | 2 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 3 |
| 2 | 0 | 2 | 0 | 0 | 0 | 0 | 3 |
| 2 | 0 | 3 | 0 | 0 | 0 | 0 | 3 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 3 |
| 2 | 1 | 1 | 0 | 32 | 0 | 0 | 1 |
| 2 | 1 | 2 | 0 | 0 | 0 | 0 | 3 |
| 2 | 1 | 3 | 0 | 2 | 0 | 0 | 1 |
| 2 | 2 | 0 | 0 | 0 | 1 | 2 | 3 |
| 2 | 2 | 1 | 0 | 1 | 1 | 0 | 2 |
| 2 | 2 | 2 | 0 | 0 | 52 | 15 | 2 |
| 2 | 2 | 3 | 0 | 0 | 23 | 6 | 2 |
| 2 | 3 | 0 | 0 | 0 | 1 | 0 | 2 |
| 2 | 3 | 1 | 0 | 0 | 1 | 1 | 3 |
| 2 | 3 | 2 | 0 | 0 | 9 | 24 | 3 |
| 2 | 3 | 3 | 0 | 0 | 14 | 20 | 3 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 3 |
| 3 | 0 | 1 | 0 | 0 | 0 | 0 | 3 |
| 3 | 0 | 2 | 0 | 0 | 0 | 0 | 3 |
| 3 | 0 | 3 | 0 | 0 | 0 | 0 | 3 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 3 |
| 3 | 1 | 1 | 0 | 39 | 0 | 0 | 1 |
| 3 | 1 | 2 | 0 | 0 | 0 | 0 | 3 |
| 3 | 1 | 3 | 0 | 0 | 1 | 0 | 2 |
| 3 | 2 | 0 | 0 | 0 | 1 | 1 | 3 |
| 3 | 2 | 1 | 0 | 0 | 0 | 0 | 3 |
| 3 | 2 | 2 | 0 | 0 | 20 | 9 | 2 |
| 3 | 2 | 3 | 0 | 0 | 14 | 7 | 2 |
| 3 | 3 | 0 | 0 | 0 | 1 | 0 | 2 |
| 3 | 3 | 1 | 0 | 1 | 0 | 0 | 1 |
| 3 | 3 | 2 | 0 | 0 | 10 | 18 | 3 |
| 3 | 3 | 3 | 0 | 0 | 8 | 26 | 3 |

Table 25:BKS lookup table

The accuracy using this lookup table was 0.1891. This was better than any of the performances noted for any classifier.

Discussion

In the project, we tried to classify vehicle silhouettes as different vehicles using different classification techniques. Supervised classification techniques were found to give better performance than unsupervised classifiers. And for improving performance of classification, we did experiment on fusion of best three classifiers. We used three different fusion algorithms plurality, Naïve Bayes and behavior knowledge space to test on fused results. The accuracy of fused classifiers on overall was found better than individual classifiers and among them behavior knowledge space fused classifier gave highest accuracy for the dataset.

References

1. Vehicle silhouettes, Turing Institute, Glasgow, Scotland,<http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/vehicle/>
2. <http://www.mathworks.com/matlabcentral/fileexchange/39352-multi-class-svm>
3. <http://en.wikipedia.org/>
4. Lecture notes ece471-571 from Dr. Qi
5. Pattern Classification , second edition, Richard O. Duda, Peter E. Har, David G. Stork
6. [Results and Codes](https://drive.google.com/a/vols.utk.edu/?tab=mo#folders/0B2A-tWBTKV-6T0pDcGhXblZ2VVU) http://goo.gl/dnkQVU

Appendixes

**Appendix A: Preprocessing codes:**

1. Normalizing data set(norm.cpp):
2. #include <iostream>
3. #include <fstream>
4. #include <cmath>
5. #include <cstdlib>
6. #include "Matrix.h"
7. #include "Pr.h"
8. #define usage "Usage: ./norm traning\_set testing\_set\n\n"
9. void normalize**(**Matrix **&**tr**,** Matrix **&**te**,** const int nf**,** const int flag**)**
10. **{**
11. Matrix mu**,** Sigma**,** sigma**;**
12. // get the statistics from the training set
13. mu **=** mean**(**tr**,** nf**);**
14. Sigma **=** cov**(**tr**,** nf**);**
15. sigma**.**createMatrix**(**nf**,**1**);**
16. **for** **(**int j**=**0**;** j**<**nf**;** j**++)**
17. sigma**(**j**,**0**)** **=** sqrt**(**Sigma**(**j**,**j**));**
18. // normalize the training set
19. **for** **(**int i**=**0**;** i**<**tr**.**getRow**();** i**++)** **{**
20. **for** **(**int j**=**0**;** j**<**nf**;** j**++)**
21. tr**(**i**,**j**)** **=** **(**tr**(**i**,**j**)-**mu**(**j**,**0**))** **/** sigma**(**j**,**0**);**
22. **}**
23. // normalize the test set
24. **if** **(**flag**)** **{**
25. **for** **(**int i**=**0**;** i**<**te**.**getRow**();** i**++)** **{**
26. **for** **(**int j**=**0**;** j**<**nf**;** j**++)**
27. te**(**i**,**j**)** **=** **(**te**(**i**,**j**)-**mu**(**j**,**0**))** **/** sigma**(**j**,**0**);**
28. **}**
29. **}**
30. **}**
31. **using** **namespace** std**;**
32. int main **(**int argc**,** char **\*\***Set**)**
33. **{**
34. **if** **(**argc **<** 3**)** **{**
35. cout **<<** usage**;**
36. exit**(**1**);**
37. **}**
38. int i**,** j**,**k**;**
39. static Matrix **\***meanstr**,\***sigma**,** **\***norm**,** **\***normt**;**
40. int nc**,**nr**,**nf**,**tenr**,**tmpnr**,**tmptnr**;**
41. float sum**,**sumt**;**
42. nf**=**atoi**(**Set**[**3**]);**
43. Matrix tmp2**,**tmp1**,**data**,** te**;**
44. data **=** readData**(**Set**[**1**],**nf**);**
45. nc **=** data**.**getCol**();**
46. nr **=** data**.**getRow**();**
47. te **=** readData**(**Set**[**2**],**nf**);**
48. tenr**=** te**.**getRow**();**
49. normalize**(**data**,**te**,** nf**,** 1**);**
50. writeData**(**data **,**"nX.tr"**);**
51. writeData**(**te**,**"nX.te"**);**
52. **return** 0**;**
53. **}**
54. Dimensionality reduction using Fisher discriminant function (fld.m)

% extract the samples belonging to different classes

nf**=**18**;**

c**=**4**;**

load dataset**;**

Tr **=** dataset**;**

I0 **=** find**(**Tr**(:,**nf**+**1**)** **==** 0**);** % find the row indices for the 1st class, labeled as 0

Tr0 **=** Tr**(**I0**,**1**:**nf**);**

I1 **=** find**(**Tr**(:,**nf**+**1**)** **==** 1**);** % find the row indices for the 1st class, labeled as 1

Tr1 **=** Tr**(**I1**,**1**:**nf**);**

I2 **=** find**(**Tr**(:,**nf**+**1**)** **==** 2**);** % find the row indices for the 1st class, labeled as 2

Tr2 **=** Tr**(**I2**,**1**:**nf**);**

I3 **=** find**(**Tr**(:,**nf**+**1**)** **==** 3**);** % find the row indices for the 1st class, labeled as 3

Tr3 **=** Tr**(**I3**,**1**:**nf**);**

C0**=**cov**(**Tr0**);**

M0**=**mean**(**Tr0**);**

n0**=**size**(**Tr0**,**1**);**

S0**=(**n0**-**1**)\***C0**;**

C1**=**cov**(**Tr1**);**

M1**=**mean**(**Tr1**);**

n1**=**size**(**Tr1**,**1**);**

S1**=(**n1**-**1**)\***C0**;**

C2**=**cov**(**Tr2**);**

M2**=**mean**(**Tr2**);**

n2**=**size**(**Tr2**,**1**);**

S2**=(**n2**-**1**)\***C0**;**

C3**=**cov**(**Tr3**);**

M3**=**mean**(**Tr3**);**

n3**=**size**(**Tr3**,**1**);**

S3**=(**n3**-**1**)\***C0**;**

M**=(**n0**\***M0**+**n1**\***M1**+**n2**\***M2**+**n3**\***M2**)/(**n0**+**n1**+**n2**+**n3**);**

Sw**=**S0**+**S1**+**S2**+**S3**;**

sb0**=**n0**\***transpose**(**M0**-**M**)\*(**M0**-**M**);**

sb1**=**n1**\***transpose**(**M1**-**M**)\*(**M1**-**M**);**

sb2**=**n2**\***transpose**(**M2**-**M**)\*(**M2**-**M**);**

sb3**=**n3**\***transpose**(**M3**-**M**)\*(**M3**-**M**);**

Sb**=**sb0**+**sb1**+**sb2**+**sb3**;**

**[**V**,**D**]=**eig**(**Sb**,**Sw**);**

W**=**V**(:,**1**:**c**-**1**);**

fX**=**Tr**(:,**1**:**nf**)\***W**;**

lX**=**Tr**(:,**nf**+**1**);**

size**(**lX**)**

size**(**fX**)**

fTr**=[**fX lX**];**

%writing the output file

dlmwrite**(**'nfX'**,**fTr**,** ' '**);**

dlmwrite**(**'nfX1'**,** fTr**(**1**:**282**,:),**' '**);**

dlmwrite**(**'nfX2'**,**fTr**(**283**:**282**\***2**,:),** ' '**);**

dlmwrite**(**'nfX3'**,**fTr**(**282**\***2**+**1**:**282**\***3**,:),** ' '**);**

dlmwrite**(**'nfX12'**,[**fTr**(**1**:**282**,:);**fTr**(**283**:**282**\***2**,:)],** ' '**);**

dlmwrite**(**'nfX13'**,** **[**fTr**(**1**:**282**,:);**fTr**(**282**\***2**+**1**:**282**\***3**,:)],**' '**);**

dlmwrite**(**'nfX23'**,[**fTr**(**283**:**282**\***2**,:);**fTr**(**282**\***2**+**1**:**282**\***3**,:)],** ' '**);**

1. Principal component analysis(PCA.cpp)

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

#define usage "Usage: ./pca traning\_set testing\_set nf\n\n"

int pca**(**Matrix **&**tr**,** Matrix **&**te**,** const int nf**,** const float err**,** const int flag**)**

**{**

Matrix Sigma**,** temp**;**

Matrix d**(**1**,**nf**),** // eigenvalue (a row vector)

V**(**nf**,**nf**),** // eigenvector with each col an eigenvector

pV**;** // eigenvectors selected based on "err"

int p**,** pnf**;**

float psum**,** sum**;**

Matrix trpca**,**tepca**;** //output matrices after pca

int nr**,**nrte**;**

nr **=** tr**.**getRow**();**

nrte**=**te**.**getRow**();**

Sigma **=** cov**(**tr**,** nf**);**

jacobi**(**Sigma**,** d**,** V**);**

eigsrt**(**d**,** V**);** // sort the eigenvalue in the ascending order

// determine the number of principal components to keep based on "err" given

sum **=** 0.0**;**

**for** **(**int i**=**0**;** i**<**nf**;** i**++)**

sum **+=** d**(**0**,**i**);**

p **=** 0**;**

psum **=** 0.0**;**

**while** **(**psum**/**sum **<** err **&&** p **<** nf**)**

**{**

psum **+=** d**(**0**,**p**);**

p**++;**

**}**

pnf **=** nf **-** **)**p**-**1**);**

cout**<<** "d= "**<<**pnf**<<**endl**;**

trpca**.**createMatrix**(**nr**,**pnf**+**1**);**

pV **=** subMatrix**(**V**,**0**,(**p**-**1**),**nf**-**1**,**nf**-**1**);**

// perform the transformation

**for** **(**int i**=**0**;** i**<**tr**.**getRow**();** i**++)** **{** // for training set

temp **=** subMatrix**(**tr**,**i**,**0**,**i**,**nf**-**1**);**

temp **=** temp**->\***pV**;**

**for** **(**int j**=**0**;** j**<**pnf**;** j**++){**

tr**(**i**,**j**)** **=** temp**(**0**,**j**);**

trpca**(**i**,**j**)=**temp**(**0**,**j**);**

**}**

trpca**(**i**,**pnf**)=**tr**(**i**,**nf**);**

**}**

tepca**.**createMatrix**(**nrte**,**pnf**+**1**);**

**if** **(**flag**)** **{**

**for** **(**int i**=**0**;** i**<**te**.**getRow**();** i**++)** **{** // for test set

temp **=** subMatrix**(**te**,**i**,**0**,**i**,**nf**-**1**);**

temp **=** temp**->\***pV**;**

**for** **(**int j**=**0**;** j**<**pnf**;** j**++){**

te**(**i**,**j**)** **=** temp**(**0**,**j**);**

tepca**(**i**,**j**)=**temp**(**0**,**j**);**

**}**

tepca**(**i**,**pnf**)=**te**(**i**,**nf**);**

**}**

**}**

writeData**(**trpca**,**"tX.tr"**);**

writeData**(**tepca**,**"tX.te"**);**

**return** pnf**;**

**}**

**using** **namespace** std**;**

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

**{**

**if** **(**argc **<** 3**)** **{**

cout **<<** usage**;**

exit**(**1**);**

**}**

int i**,**j**;**

int nf**,**nc**,**p**;**

Matrix data**,** te**;**

nf**=**atoi**(**Set**[**3**]);**

data **=** readData**(**Set**[**1**],**nf**);**

nc **=** data**.**getCol**();**

te **=** readData**(**Set**[**2**],**nf**);**

p**=**pca**(**data**,** te**,** nf**,** 0.1**,**1**);**

cout**<<** "Number of new feature="**<<**p**<<**endl**;**

**return** 0**;**

**}**

**Appendix B: classifiers code**

MPP (mpp.cpp)

1. /\*
2. \* MPP.cpp - test routine to use MPP to process the synthetic dataset and generate CM
3. \*
4. \*/
5. #include <iostream>
6. #include <fstream>
7. #include <cmath>
8. #include <cstdlib>
9. #include "Matrix.h"
10. #include "Pr.h"
11. **using** **namespace** std**;**
12. #define Usage "Usage: ./MppCM training\_set test\_set classes features cases \n\t training\_set: the file name for training set\n\t test\_set: the file name for test set\n\t classes: number of classes\n\t features: number of features (dimension)\n\t cases: used for MPP, cases can only be 1, 2, or 3 \n\toutputfilename\n\t BKS\_file\_name\n"
13. int main**(**int argc**,** char **\*\***argv**)**
14. **{**
15. int nrTr**,** nrTe**,** // number of rows in the training and test set
16. nc**;** // number of columns in the data set; both the
17. // training and test set should have the same
18. // column number
19. Matrix Tr**,** Te**,**bks**;**
20. // check to see if the number of argument is correct
21. **if** **(**argc **<** 7**)** **{**
22. cout **<<** Usage**;**
23. exit**(**1**);**
24. **}**
25. int c**=** atoi**(**argv**[**3**]);** // number of classes
26. int nf **=** atoi**(**argv**[**4**]);** // number of features (dimension)
27. int cases **=** atoi**(**argv**[**5**]);** // used by MPP
29. // read in data from the data file
30. nc **=** nf**+**1**;** // the data dimension; plus the one label column
31. Tr **=** readData**(**argv**[**1**],** nc**);**
32. nrTr **=** Tr**.**getRow**();** // get the number of rows
33. Te **=** readData**(**argv**[**2**],** nc**);**
34. nrTe **=** Te**.**getRow**();** // get the number of columns
36. //truth results and classfied result matrix
37. bks**.**createMatrix**(**nrTe**,**2**);**
38. // prepare the labels and error count
39. Matrix labelMPP**(**nrTe**,** 1**);** // a col vector to hold result for MPP
40. int errCountMPP **=** 0**;** // calcualte error rate for MPP
41. // assign prior probability
42. Matrix Pw**(**c**,** 1**);**
43. **for** **(**int i**=**0**;** i**<**c**;** i**++)**
44. Pw**(**i**,**0**)** **=** **(**float**)**1**/**c**;** // if assuming equal prior probability
45. Matrix output**,**cm**;**
46. output**.**createMatrix**(**nrTe**,**3**);**
47. cm**.**createMatrix**(**c**+**1**,**c**+**1**);**
48. // perform classification
49. **for** **(**int i**=**0**;** i**<**nrTe**;** i**++)** **{**
50. // classify one test sample at a time, get one sample from the test data
51. Matrix sample **=** transpose**(**subMatrix**(**Te**,** i**,** 0**,** i**,** nf**-**1**));**
52. // call MPP to perform classification
53. labelMPP**(**i**,**0**)** **=** mpp**(**Tr**,** sample**,** c**,** cases**,** Pw**);**
55. // check if the classification result is correct or not
56. **if** **(**labelMPP**(**i**,**0**)** **!=** Te**(**i**,**nf**))**
57. errCountMPP**++;**
58. //generating confustion matrix element
59. output**(**i**,**0**)=**i**;**//sample number
60. output**(**i**,**1**)=**Te**(**i**,**nf**);**//actual result
61. output**(**i**,**2**)=**labelMPP**(**i**,**0**);**//classification result
62. **}**
63. bks**=**subMatrix**(**output**,** // crop a matrix
64. 0**,** // starting row index
65. 1**,** // starting column index
66. nrTe**-**1**,** // ending row index
67. 2**);** // ending column index
68. //generate CM
69. cm**(**0**,**0**)=**0**;**
70. **for** **(**int i**=**0**;**i**<**c**;**i**++)**
71. **{** cm**(**0**,**i**+**1**)=**i**;**
72. cm**(**i**+**1**,**0**)=**i**;**
73. **}**
74. **for(**int r**=**0**;**r**<**c**;**r**++){**
75. **for(**int i**=**0**;**i**<**c**;**i**++)**
76. **{**
77. int w**=**0**;**
78. **for(**int j**=**0**;**j**<**nrTe**;**j**++)**
79. **{**
80. **if** **((**output**(**j**,**2**)==**r**)&&(**output**(**j**,**1**)==**i**))**
81. w**++;**
82. **}**
83. cm**(**i**+**1**,**r**+**1**)=**w**;**
84. **}**
86. **}**
87. //writeData(output,"output");
88. writeData**(**cm**,**argv**[**6**]);**
89. writeData**(**bks**,**argv**[**7**]);**
90. // calculate accuracy
91. cout **<<** "The error rate using MPP is = " **<<** **(**float**)**errCountMPP**/**nrTe **<<** endl**;**
93. **return** 0**;**
94. **}**

KNN (knn.cpp)

#include <iostream>

#include <cstdlib>

#include <cmath>

#include "time.h"

#include "Pr.h"

#define usage "Usage: ./knnpdCM traning\_set testing\_set k c nf CMoutputfile BKS\_file\_name\n\n"

**using** **namespace** std**;**

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

**{**

clock\_t t0**=**clock**(),**t1**;**

**if** **(**argc **<** 8**)** **{**

cout **<<** usage**;**

exit**(**1**);**

**}**

int i**,** j**,**w**,**f**;**

int nc**,**nr**,**nf**,**nrte**;**

int c**,**k**,**r**,**Result**,**error**=**0**;**

char **\***h**=**Set**[**3**],\***h1**=**Set**[**4**],\***h2**=**Set**[**5**];**

k**=**atoi**(**h**);**//for k

c**=**atoi**(**h1**);**//for number of class

nf**=**atoi**(**h2**);**// for number of feature

Matrix data**,**subt**,**sub**,**te**,**tmp**,**Mx**,**output**,**cm**,**bks**;**

double a**,**b**,**n**,**m**=**0**;**

data **=** readData**(**Set**[**1**],**nf**+**1**);**

te **=** readData**(**Set**[**2**],**nf**+**1**);**

nc **=** data**.**getCol**();**

nr **=** data**.**getRow**();**

nrte**=** te**.**getRow**();**

tmp**.**createMatrix**(**k**,**2**);**

Mx**.**createMatrix**(**c**,**2**);**

output**.**createMatrix**(**nrte**,**3**);**

bks**.**createMatrix**(**nrte**,**2**);**

cm**.**createMatrix**(**c**+**1**,**c**+**1**);**

**for** **(**f**=**0**;**f**<**nrte**;**f**++){**

subt**=**subMatrix**(**te**,**f**,**0**,**f**,**nf**-**1**);** //testing sample

**for** **(**i**=**0**;**i**<**k**;**i**++)**//finding euc dist bewteen first k sample in training set and testing sample

**{**

sub**=**subMatrix**(**data**,**i**,**0**,**i**,**nf**-**1**);**

tmp**(**i**,**0**)=**euc**(**subt**,**sub**);**

tmp**(**i**,**1**)=**data**(**i**,**nf**);**

**}**

**for(**j**=**k**;**j**<**nr**;**j**++)**

**{**

**for** **(**w**=**0**;**w**<**k**;**w**++){** //bubble sorting for partial set

**for** **(**i**=**0**;**i**<**k**-**1**;**i**++){**

**if** **(**tmp**(**i**,**0**)<**tmp**(**i**+**1**,**0**)){**

a**=** tmp**(**i**+**1**,**0**);**

b**=** tmp**(**i**+**1**,**1**);**

tmp**(**i**+**1**,**0**)=**tmp**(**i**,**0**);**

tmp**(**i**+**1**,**1**)=**tmp**(**i**,**1**);**

tmp**(**i**,**0**)=**a**;**

tmp**(**i**,**1**)=**b**;**

**}**

**}**

**}**

sub**=**subMatrix**(**data**,**j**,**0**,**j**,**nf**-**1**);**

n**=**euc**(**subt**,**sub**);**

**if** **(**tmp**(**0**,**0**)>**n**){**

tmp**(**0**,**0**)=**n**;**

tmp**(**0**,**1**)=**data**(**j**,**nf**);}**

**}**

**for** **(**i**=**0**;**i**<**c**;**i**++)**//generate labels

Mx**(**i**,**0**)=**i**;**

**for** **(**j**=**0**;**j**<**c**;**j**++)**//counting how many sample for each lable

**{**

r**=**0**;**

**for** **(**i**=**0**;** i**<**k**;** i**++)**

**{**

**if** **((**tmp**(**i**,**1**)==**j**))**

r**++;**

**}**

Mx**(**j**,**1**)=**r**;**

**}**

**for** **(**i**=**0**;**i**<**c**;**i**++){**

**if** **(**Mx**(**0**,**1**)<**Mx**(**i**+**1**,**1**)){**

a**=** Mx**(**i**+**1**,**0**);**

b**=** Mx**(**i**+**1**,**1**);**

Mx**(**i**+**1**,**0**)=**Mx**(**0**,**0**);**

Mx**(**i**+**1**,**1**)=**Mx**(**0**,**1**);**

Mx**(**0**,**0**)=**a**;**

Mx**(**0**,**1**)=**b**;**

**}**

**}**

//Mx(0,0) is the result of classification

Result**=**Mx**(**0**,**0**);**

//generating confustion matrix element

output**(**f**,**0**)=**f**;**//sample number

output**(**f**,**1**)=**te**(**f**,**nf**);**//actual result

output**(**f**,**2**)=**Result**;**//classification result

**if** **(**Result **!=** te**(**f**,**nf**))**

error**++;**

**}**

//generate CM

cm**(**0**,**0**)=**0**;**

**for** **(**i**=**0**;**i**<**c**;**i**++)**

**{** cm**(**0**,**i**+**1**)=**i**;**

cm**(**i**+**1**,**0**)=**i**;**

**}**

**for(**r**=**0**;**r**<**c**;**r**++){**

cout**<<**output**;**

**for(**i**=**0**;**i**<**c**;**i**++)**

**{**

w**=**0**;**

**for(**j**=**0**;**j**<**nrte**;**j**++)**

**{**

**if** **((**output**(**j**,**2**)==**r**)&&(**output**(**j**,**1**)==**i**))**

//if(output(j,1)=i)

w**++;**

**}**

cm**(**i**+**1**,**r**+**1**)=**w**;**

**}**

**}**

bks**=**subMatrix**(**output**,** // crop a matrix

0**,** // starting row index

1**,** // starting column index

nrte**-**1**,** // ending row index

2**);** // ending column index

//writeData(output,"output");

writeData**(**cm**,**Set**[**6**]);**

writeData**(**bks**,**Set**[**7**]);**

cout**<<**"error Rate"**<<(**double**)**error**/**nrte**<<**endl**;**

t1**=**clock**()-**t0**;**

cout**<<**"time"**<<**t1**<<**endl**;**

**return** 0**;**

**}**

Back propagation (bp.m)

% set parameters

nf**=**6**;**

c**=**4**;**

%load training set

load ndf23**;**

Tr**=**ndf23**;**

TrX**=**Tr**(:,**1**:**nf**);**

TrL**=**Tr**(:,**nf**+**1**);**

% load testing set

load ndf1**;**

Te**=**ndf1**;**

TeX**=**Te**(:,**1**:**nf**);**

TeL**=**Te**(:,**nf**+**1**);**

% prepare data for bp

% transpose inputs

iTr**=**transpose**(**TrX**);**

iTe**=**transpose**(**TeX**);**

% create target vectors

tTr**=**zeros**(**size**(**TrL**,**1**),**c**);**

tTe**=**zeros**(**size**(**TeL**,**1**),**c**);**

**for** i**=**1**:**size**(**TrL**,**1**)**

tTr**(**i**,**TrL**(**i**)+**1**)=**1**;**

**end**

**for** i**=**1**:**size**(**TeL**,**1**)**

tTe**(**i**,**TeL**(**i**)+**1**)=**1**;**

**end**

tTr**=**transpose**(**tTr**);**

tTe**=**transpose**(**tTe**);**

% create NN

net**=**patternnet**(**10**);**

% train the network

net**=**train**(**net**,**iTr**,**tTr**);**

% test using the network

oTe**=**net**(**iTe**);**

% get back class labels from output target vectors

y**=**vec2ind**(**oTe**);**

y**=**transpose**(**y**);**

y**=**y**-**1**;** % since class starts from 1 target output vectors

%performance calculation

k**=**0**;**

**for** i**=**1**:**size**(**TeL**,**1**)**

**if(**y**(**i**,**1**)~=**TeL**(**i**,**1**))**

k**=**k**+**1**;**

**end**

**end**

disp**(**'peformance = '**);**

disp**(**k**/**size**(**TeL**,**1**));**

% calculate confusion matrix

out**=**zeros**(**size**(**y**,**1**),**2**);**

out**(:,**1**)=**TeL**;**

out**(:,**2**)=**y**;**

cm**=**zeros**(**c**+**1**,**c**+**1**);**

cm**(**1**,**1**)=**0**;**

**for** i**=**1**:**c

cm**(**1**,**i**+**1**)=**i**;**

cm**(**i**+**1**,**1**)=**i**;**

**end**

**for** r**=**1**:**c

**for** i**=**1**:**c

w**=**0**;**

**for** j**=**1**:**size**(**TeL**,**1**)**

**if** **((**out**(**j**,**2**)==**r**-**1**)&&(**out**(**j**,**1**)==**i**-**1**))**

w**=**w**+**1**;**

**end**

**end**

cm**(**i**+**1**,**r**+**1**)=**w**;**

**end**

**end**

disp**(**cm**);**

% write to a file

dlmwrite**(**'bpCMndf23.txt'**,**cm**,**' '**);**

**Decision Tree (DT.m)**

% set parameters

nf**=**6**;**

c**=**4**;**

%load training set

load **(**'ndf23'**,**'-ascii'**);**

Tr**=**ndf23**;**

TrX**=**Tr**(:,**1**:**nf**);**

TrL**=**Tr**(:,**nf**+**1**);**

save Trt**;**

%load testing set

load **(**'ndf1'**,**'-ascii'**);**

Te**=**ndf1**;**

TeX**=**Te**(:,**1**:**nf**);**

TeL**=**Te**(:,**nf**+**1**);**

save Tet**;**

% fit /train decision tree

ctree**=** fitctree**(**TrX**,**TrL**);**

% predict using the trained tree

y**=**predict**(**ctree**,**TeX**);**

%performance calculation

k**=**0**;**

**for** i**=**1**:**size**(**TeL**,**1**)**

**if(**y**(**i**,**1**)~=**TeL**(**i**,**1**))**

k**=**k**+**1**;**

**end**

**end**

disp**(**'peformance = '**);**

disp**(**k**/**size**(**TeL**,**1**));**

% calculate confusion matrix

out=zeros(size(y,1),2);

out(:,1)=TeL;

out(:,2)=y;

cm=zeros(c+1,c+1);

cm(1,1)=0;

for i=1:c

cm(1,i+1)=i;

cm(i+1,1)=i;

end

for r=1:c

for i=1:c

w=0;

for j=1:size(TeL,1)

if ((out(j,2)==r-1)&&(out(j,1)==i-1))

w=w+1;

end

end

cm(i+1,r+1)=w;

end

end

disp(cm);

% write to a file

dlmwrite('decisionCMndf23.txt',cm,' ');

**SVM: Classifying function(multisvm.m)**

1. **function** **[**result**]** **=** multisvm**(**TrainingSet**,**GroupTrain**,**TestSet**)**
2. %Models a given training set with a corresponding group vector and
3. %classifies a given test set using an SVM classifier according to a
4. %one vs. all relation.
5. %
6. %This code was written by Cody Neuburger cneuburg@fau.edu
7. %Florida Atlantic University, Florida USA
8. %This code was adapted and cleaned from Anand Mishra's multisvm function
9. %found at http://www.mathworks.com/matlabcentral/fileexchange/33170-multi-class-support-vector-machine/
10. u**=**unique**(**GroupTrain**);**
11. numClasses**=**length**(**u**);**
12. result **=** zeros**(**length**(**TestSet**(:,**1**)),**1**);**
13. %build models
14. **for** k**=**1**:**numClasses
15. %Vectorized statement that binarizes Group
16. %where 1 is the current class and 0 is all other classes
17. G1vAll**=(**GroupTrain**==**u**(**k**));**
18. models**(**k**)** **=** svmtrain**(**TrainingSet**,**G1vAll**);**
19. **end**
20. %classify test cases
21. **for** j**=**1**:**size**(**TestSet**,**1**)**
22. **for** k**=**1**:**numClasses
23. **if(**svmclassify**(**models**(**k**),**TestSet**(**j**,:)))**
24. **break;**
25. **end**
26. **end**
27. result**(**j**)** **=** k**;**
28. **End**

**SVM:Testing code(svm.m)**

% set parameters

nf**=**18**;**

c**=**4**;**

%load training set

load n23**;**

Tr**=**n23**;**

TrX**=**Tr**(:,**1**:**nf**);**

TrL**=**Tr**(:,**nf**+**1**);**

% load testing set

load n1**;**

Te**=**n1**;**

TeX**=**Te**(:,**1**:**nf**);**

TeL**=**Te**(:,**nf**+**1**);**

resutls**=** multisvm**(**TrX**,**transpose**(**TrL**),**TeX**);**

y**=**resutls**;**

%performance calculation

k**=**0**;**

**for** i**=**1**:**size**(**TeL**,**1**)**

**if(**resutls**(**i**)~=**TeL**(**i**,**1**))**

k**=**k**+**1**;**

**end**

**end**

disp**(**'performance = '**);**

disp**(**k**/**size**(**TeL**,**1**));**

**Appendix C: Unsupervised learning**

1. **Kmeans (kmeans.cpp)**
2. #include "Matrix.h"
3. #include "Pr.h"
4. #include <iostream>
5. #include <cstdlib>
6. #include <cmath>
7. **using** **namespace** std**;**
8. /\*\*
9. \* @param S: data matrix
10. \* @param C: cluster center matrix
11. \*\*/
12. Matrix kmeans**(**Matrix **&**S**,** Matrix **&**C**,** int max\_val**)**
13. **{**
14. int nrS**=**S**.**getRow**();**
15. int ncS**=**S**.**getCol**();**
16. int nrC**=**C**.**getRow**();**
17. int ncC**=**C**.**getCol**();**
18. int d**=**0**;**
19. Matrix dist**(**1**,**nrC**);**
20. Matrix sdist**(**1**,**nrC**);**
21. int pos**;**
22. double minD**;**
23. Matrix lookup**(**nrS**,**1**);** // lookup table for sample belonging to which cluster
24. Matrix count**(**nrC**,**1**);** // to store count for each cluster center
25. Matrix sum**(**nrC**,**ncC**);**
26. // initialize cluster means
27. **for(**int i**=**0**;**i**<**nrC**;**i**++)**
28. **for(**int j**=**0**;**j**<**ncC**;**j**++)**
29. **{**C**(**i**,**j**)=**rand**()%(**max\_val**+**1**);}**
30. //Matrix x1,y1; // for storing points to be used calculate distance
31. **for(**int i**=**0**;**i**<**nrS**;**i**++)**
32. **{**
33. lookup**(**i**,**0**)=**0**;**
34. **}**
35. bool change**;**
36. // while change of assignment to centers occurs
37. **do{**
38. cout**<<**d**++<<**endl**;**
39. change**=false;**
40. // clear values in count and cluster mean
41. **for(**int i**=**0**;**i**<**nrC**;**i**++)**
42. **{**
43. **for(**int j**=**0**;**j**<**ncC**;**j**++)**
44. **{**
45. sum**(**i**,**j**)=**0**;**
46. count**(**i**,**0**)=**0**;**
47. **}**
49. **}**
50. **for(**int i**=**0**;**i**<**nrS**;**i**++)**
51. **{**
53. // calculate distance between sample data and each cluster mean
54. minD**=**0**;**
55. pos**=**0**;**
56. **for(**int k**=**0**;**k**<**ncC**;**k**++)**
57. minD**+=(**S**(**i**,**k**)-**C**(**0**,**k**))\*(**S**(**i**,**k**)-**C**(**0**,**k**));**
58. **for(**int j**=**0**;**j**<**nrC**;**j**++)**
59. **{**
60. dist**(**0**,**j**)=**0**;**
61. **for(**int k**=**0**;**k**<**ncC**;**k**++)**
62. dist**(**0**,**j**)+=(**S**(**i**,**k**)-**C**(**j**,**k**))\*(**S**(**i**,**k**)-**C**(**j**,**k**));**
63. // identify positional value of cluster mean that will have minimum distance
64. **if(**dist**(**0**,**j**)<**minD**)**
65. **{**
66. pos**=**j**;**
67. minD**=**dist**(**0**,**j**);**
68. **}**
69. **}**
71. // if lookup table value for this doesn't matches cluster mean's position then update
72. **if(((**int**)**lookup**(**i**,**0**))!=**pos**)**
73. **{**
74. change**=true;**
75. lookup**(**i**,**0**)=**pos**;**
76. **}**
77. // calculate sum of values in each dimension belonging to each cluster
78. **for(**int j**=**0**;**j**<**ncC**;**j**++)**
79. **{**
80. sum**((**int**)**lookup**(**i**,**0**),**j**)+=**S**(**i**,**j**);**
81. **}**
82. // increase count for the cluster mean by 1
83. count**((**int**)**lookup**(**i**,**0**),**0**)++;**
84. **}**
86. // update mean for each cluster
87. **if(**change**==true)**
88. **{**
90. // divide sum by count and update mean
91. **for(**int i**=**0**;**i**<**nrC**;**i**++)**
92. **{**
93. **if(**count**(**i**,**0**)!=**0**)**
94. **{**
95. **for(**int j**=**0**;**j**<**ncC**;**j**++)**
96. **{**
98. C**(**i**,**j**)=(**sum**(**i**,**j**)/**count**(**i**,**0**));**
99. **}**
100. **}**
101. **}**
102. **}**
103. **}** **while(**change**==true);**
105. **return** lookup**;**
106. **}**
107. **Testing Kmeans (testKmeans.cpp)**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

**using** **namespace** std**;**

#define Usage "Usage: ./testkmeans data\_file nf k CM\_file cl\_file \n"

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

**{**

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

**if** **(**argc **^** 6**)** **{**

cout **<<** Usage**;**

exit**(**1**);**

**}**

int nf**=**atoi**(**argv**[**2**]);**

int nc**=**nf**+**1**;**

Matrix Tr**=**readData**(**argv**[**1**],**nc**);**

char **\***cm\_file**=**argv**[**4**];**

char **\***cl\_file**=**argv**[**5**];**

int k**=**atoi**(**argv**[**3**]);**

int nrTr**=**Tr**.**getRow**();**

Matrix mTr**=**subMatrix**(**Tr**,**0**,**0**,**nrTr**-**1**,**nf**-**1**);**

Matrix oTrL**(**nrTr**,**1**);**

// define arbitrary clr cluster means

Matrix C**(**k**,**nf**);**

// call kmeans to get lookup table that refers each pixel in original pic to updated cluster mean pixel

Matrix lookup**;**

lookup**=**kmeans**(**mTr**,**C**,** 1**);**

// find which cluster mean belongs to which class in supervised method

Matrix x**(**k**,**k**);**

**for(**int i**=**0**;**i**<**nrTr**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**k**;**j**++)**

**{**

**if(**lookup**(**i**,**0**)==**j**)**

**{**

x**(**j**,**Tr**(**i**,**nf**))+=**1**;**

**}**

**}**

**}**

Matrix s**(**1**,**k**);**

Matrix pos**(**1**,**k**);**

Matrix clookup**(**k**,**1**);**

Matrix d**;**

Matrix output**,** cm**;**

int error**=**0**;**

output**.**createMatrix**(**nrTr**,**2**);**

cm**.**createMatrix**(**k**+**1**,**k**+**1**);**

**for(**int j**=**0**;**j**<**k**;**j**++)**

**{**

d**=**subMatrix**(**x**,**j**,**0**,**j**,**k**-**1**);**

insertsort**(**d**,**s**,**pos**);**

clookup**(**j**,**0**)=**pos**(**0**,**k**-**1**);**

**}**

// assign class according to lookup content using clookup mapping

**for(**int i**=**0**;**i**<**nrTr**;**i**++)**

**{**

oTrL**(**i**,**0**)=**clookup**(**lookup**(**i**,**0**),**0**);**

output**(**i**,**0**)=**Tr**(**i**,**nf**);**//actual result

output**(**i**,**1**)=**oTrL**(**i**,**0**);**//classification result

**if(**oTrL**(**i**,**0**)!=**Tr**(**i**,**nf**))**

error**++;**

**}**

// create confusion matrix

//generate CM

cm**(**0**,**0**)=**0**;**

**for** **(**int i**=**0**;**i**<**k**;**i**++)**

**{** cm**(**0**,**i**+**1**)=**i**;**

cm**(**i**+**1**,**0**)=**i**;**

**}**

**for(**int r**=**0**;**r**<**k**;**r**++){**

**for(**int i**=**0**;**i**<**k**;**i**++)**

**{**

int w**=**0**;**

**for(**int j**=**0**;**j**<**nrTr**;**j**++)**

**{**

**if** **((**output**(**j**,**1**)==**r**)&&(**output**(**j**,**0**)==**i**))**

w**++;**

**}**

cm**(**i**+**1**,**r**+**1**)=**w**;**

**}**

**}**

cout**<<**x**<<**endl**;**

cout**<<**cm**<<**endl**;**

cout**<<**clookup**<<**endl**;**

cout**<<**C**<<**endl**;**

//writeData(output,"output");

writeData**(**cm**,**cm\_file**);**

writeData**(**output**,**cl\_file**);**

cout**<<**"error rate = "**<<(**double**)**error**/**nrTr**<<**endl**;**

**return** 0**;**

**}**

1. **WTA (wta.cpp)**

#include "Matrix.h"

#include "Pr.h"

#include <iostream>

#include <cstdlib>

#include <cmath>

**using** **namespace** std**;**

/\*\*

\* @param S: data matrix

\* @param C: cluster center matrix

\*\*/

Matrix winnerTakesAll**(**Matrix **&**S**,** Matrix **&**C**,** int max\_val**,** double epsilon**)**

**{**

int nrS**=**S**.**getRow**();**

int ncS**=**S**.**getCol**();**

int nrC**=**C**.**getRow**();**

int ncC**=**C**.**getCol**();**

int d**=**0**;**

Matrix dist**(**1**,**nrC**);**

int pos**;**

double minD**;**

Matrix lookup**(**nrS**,**1**);** // lookup table for sample belonging to which cluster

Matrix count**(**nrC**,**1**);** // to store count for each cluster center

Matrix sum**(**nrC**,**ncC**);**

// initialize cluster means

**for(**int i**=**0**;**i**<**nrC**;**i**++)**

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**C**(**i**,**j**)=**rand**()%(**max\_val**+**1**);}**

**for(**int i**=**0**;**i**<**nrS**;**i**++)**

**{**

lookup**(**i**,**0**)=**0**;**

**}**

bool change**;**

// while change of assignment to centers occurs

**do{**

cout**<<**d**++<<**endl**;**

change**=false;**

// clear values in count and cluster mean

**for(**int i**=**0**;**i**<**nrC**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**

sum**(**i**,**j**)=**0**;**

count**(**i**,**0**)=**0**;**

**}**

**}**

**for(**int i**=**0**;**i**<**nrS**;**i**++)**

**{**

// calculate distance between sample data and each cluster mean

minD**=**0**;**

pos**=**0**;**

**for(**int k**=**0**;**k**<**ncC**;**k**++)**

minD**+=(**S**(**i**,**k**)-**C**(**0**,**k**))\*(**S**(**i**,**k**)-**C**(**0**,**k**));**

**for(**int j**=**0**;**j**<**nrC**;**j**++)**

**{**

dist**(**0**,**j**)=**0**;**

**for(**int k**=**0**;**k**<**ncC**;**k**++)**

dist**(**0**,**j**)+=(**S**(**i**,**k**)-**C**(**j**,**k**))\*(**S**(**i**,**k**)-**C**(**j**,**k**));**

// identify positional value of cluster mean that will have minimum distance

**if(**dist**(**0**,**j**)<**minD**)**

**{**

pos**=**j**;**

minD**=**dist**(**0**,**j**);**

**}**

**}**

// if lookup table value for this doesn't matches cluster mean's position then update

**if(((**int**)**lookup**(**i**,**0**))!=**pos**)**

**{**

change**=true;**

lookup**(**i**,**0**)=**pos**;**

//cout<<lookup(i,0);

**}**

// update value of that cluster mean

**for(**int k**=**0**;**k**<**ncC**;**k**++)**

**{**

C**(**pos**,**k**)+=**epsilon**\*(**S**(**i**,**k**)-**C**(**pos**,**k**));**

**}**

// calculate sum of values in each dimension belonging to each cluster

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**

sum**((**int**)**lookup**(**i**,**0**),**j**)+=**S**(**i**,**j**);**

**}**

// increase count for the cluster mean by 1

count**((**int**)**lookup**(**i**,**0**),**0**)++;**

**}**

// update mean for each cluster

**if(**change**==true)**

**{**

// divide sum by count and update mean

**for(**int i**=**0**;**i**<**nrC**;**i**++)**

**{**

**if(**count**(**i**,**0**)!=**0**)**

**{**

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**

C**(**i**,**j**)=(**sum**(**i**,**j**)/**count**(**i**,**0**));**

**}**

**}**

**}**

**}**

**}** **while(**change**==true);**

**return** lookup**;**

**}**

1. **Testing WTA (WTA.cpp)**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

**using** **namespace** std**;**

#define Usage "Usage: ./testWTA data\_file nf k CM\_file cl\_file \n"

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

**{**

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

**if** **(**argc **^** 6**)** **{**

cout **<<** Usage**;**

exit**(**1**);**

**}**

int nf**=**atoi**(**argv**[**2**]);**

int nc**=**nf**+**1**;**

Matrix Tr**=**readData**(**argv**[**1**],**nc**);**

char **\***cm\_file**=**argv**[**4**];**

char **\***cl\_file**=**argv**[**5**];**

int k**=**atoi**(**argv**[**3**]);**

int nrTr**=**Tr**.**getRow**();**

Matrix mTr**=**subMatrix**(**Tr**,**0**,**0**,**nrTr**-**1**,**nf**-**1**);**

Matrix oTrL**(**nrTr**,**1**);**

// define arbitrary clr cluster means

Matrix C**(**k**,**nf**);**

// call kmeans to get lookup table that refers each pixel in original pic to updated cluster mean pixel

Matrix lookup**;**

lookup**=**winnerTakesAll**(**mTr**,**C**,**1**,**0.0001**);**

// find which cluster mean belongs to which class in supervised method

Matrix x**(**k**,**k**);**

**for(**int i**=**0**;**i**<**nrTr**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**k**;**j**++)**

**{**

**if(**lookup**(**i**,**0**)==**j**)**

**{**

x**(**j**,**Tr**(**i**,**nf**))+=**1**;**

**}**

**}**

**}**

Matrix s**(**1**,**k**);**

Matrix pos**(**1**,**k**);**

Matrix clookup**(**k**,**1**);**

Matrix d**;**

Matrix output**,** cm**;**

int error**=**0**;**

output**.**createMatrix**(**nrTr**,**2**);**

cm**.**createMatrix**(**k**+**1**,**k**+**1**);**

**for(**int j**=**0**;**j**<**k**;**j**++)**

**{**

d**=**subMatrix**(**x**,**j**,**0**,**j**,**k**-**1**);**

insertsort**(**d**,**s**,**pos**);**

clookup**(**j**,**0**)=**pos**(**0**,**k**-**1**);**

**}**

// assign class according to lookup content using clookup mapping

**for(**int i**=**0**;**i**<**nrTr**;**i**++)**

**{**

oTrL**(**i**,**0**)=**clookup**(**lookup**(**i**,**0**),**0**);**

output**(**i**,**0**)=**Tr**(**i**,**nf**);**//actual result

output**(**i**,**1**)=**oTrL**(**i**,**0**);**//classification result

**if(**oTrL**(**i**,**0**)!=**Tr**(**i**,**nf**))**

error**++;**

**}**

// create confusion matrix

//generate CM

cm**(**0**,**0**)=**0**;**

**for** **(**int i**=**0**;**i**<**k**;**i**++)**

**{** cm**(**0**,**i**+**1**)=**i**;**

cm**(**i**+**1**,**0**)=**i**;**

**}**

**for(**int r**=**0**;**r**<**k**;**r**++){**

**for(**int i**=**0**;**i**<**k**;**i**++)**

**{**

int w**=**0**;**

**for(**int j**=**0**;**j**<**nrTr**;**j**++)**

**{**

**if** **((**output**(**j**,**1**)==**r**)&&(**output**(**j**,**0**)==**i**))**

w**++;**

**}**

cm**(**i**+**1**,**r**+**1**)=**w**;**

**}**

**}**

//writeData(output,"output");

writeData**(**cm**,**cm\_file**);**

writeData**(**output**,**cl\_file**);**

cout**<<**"error rate = "**<<(**double**)**error**/**nrTr**<<**endl**;**

**return** 0**;**

**}**

1. **Kohenan Maps (kmaps.cpp)**

#include "Matrix.h"

#include "Pr.h"

#include <iostream>

#include <cstdlib>

#include <cmath>

**using** **namespace** std**;**

/\*\*

\* @param S: data matrix

\* @param C: cluster center matrix

\*\*/

Matrix kohonenMap**(**Matrix **&**S**,** Matrix **&**C**,** int max\_val**,** int kmax**,** int map\_width**,** double sig**,** double e**)**

**{**

int nrS**=**S**.**getRow**();**

int ncS**=**S**.**getCol**();**

int nrC**=**C**.**getRow**();**

int ncC**=**C**.**getCol**();**

int d**=**0**;**

Matrix dist**(**1**,**nrC**);**

Matrix sdist**(**1**,**nrC**);**

int pos**;**

double minD**;**

Matrix lookup**(**nrS**,**1**);** // lookup table for sample belonging to which cluster

Matrix count**(**nrC**,**1**);** // to store count for each cluster center

Matrix sum**(**nrC**,**ncC**);**

Matrix g**(**nrC**,**ncC**);**

int x**=**0**;**

double sigma**=**1**;**

double phi**=**0**;**

// initialize cluster means and map co-ordinates

**for(**int i**=**0**;**i**<**nrC**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**

C**(**i**,**j**)=**rand**()%(**max\_val**+**1**);**

**}**

**}**

int y**=**0**;**

**do{**

**for(**int k**=**0**;**k**<**map\_width**;**k**++)**

**{**

g**(**y**,**0**)=**y**;**

g**(**y**,**1**)=**x**;**

g**(**y**,**2**)=**k**;**

y**++;**

**if(**y**>=**nrC**)**

**{**

**break;**

**}**

**}**

**if(**y**<**nrC**)**

x**++;**

**}while(**y**<**nrC**);**

// calculate sigma

/\*double maxD=0;double gdist;

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

{

gdist=pow(g(i,1)-g(0,1),2)+pow(g(i,2)-g(0,2),2);

if(gdist>maxD)

{

maxD=gdist;

}

}\*/

sigma**=**sig**;**

//Matrix x1,y1; // for storing points to be use calculate distance

**for(**int i**=**0**;**i**<**nrS**;**i**++)**

**{**

lookup**(**i**,**0**)=**0**;**

**}**

bool change**;**

// while change of assignment to centers occurs

**do{**

change**=false;**

cout**<<**d**++<<**endl**;**

// clear values in count and cluster mean

**for(**int i**=**0**;**i**<**nrC**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**

sum**(**i**,**j**)=**0**;**

count**(**i**,**0**)=**0**;**

**}**

**}**

**for(**int i**=**0**;**i**<**nrS**;**i**++)**

**{**

// calculate distance between sample data and each cluster mean

minD**=**0**;**

pos**=**0**;**

**for(**int k**=**0**;**k**<**ncC**;**k**++)**

minD**+=(**S**(**i**,**k**)-**C**(**0**,**k**))\*(**S**(**i**,**k**)-**C**(**0**,**k**));**

**for(**int j**=**0**;**j**<**nrC**;**j**++)**

**{**

dist**(**0**,**j**)=**0**;**

**for(**int k**=**0**;**k**<**ncC**;**k**++)**

dist**(**0**,**j**)+=(**S**(**i**,**k**)-**C**(**j**,**k**))\*(**S**(**i**,**k**)-**C**(**j**,**k**));**

// identify positional value of cluster mean that will have minimum distance

**if(**dist**(**0**,**j**)<**minD**)**

**{**

pos**=**j**;**

minD**=**dist**(**0**,**j**);**

**}**

**}**

// if lookup table value for this doesn't matches cluster mean's position then update

**if(((**int**)**lookup**(**i**,**0**))!=**pos**)**

**{**

change**=true;**

lookup**(**i**,**0**)=**pos**;**

**}**

**for(**int j**=**0**;**j**<**nrC**;**j**++)**

**{**

phi**=**exp**(-(**pow**(**g**(**pos**,**1**)-**g**(**j**,**1**),**2**)+**pow**(**g**(**pos**,**2**)-**g**(**j**,**2**),**2**))/(**2**\***pow**(**sigma**,**2**)));**

// update value all cluster means

**for(**int k**=**0**;**k**<**ncC**;**k**++)**

**{**

C**(**j**,**k**)+=**e**\***phi**\*(**S**(**i**,**k**)-**C**(**j**,**k**));**

**}**

**}**

// calculate sum of values in each dimension belonging to each cluster

**for(**int j**=**0**;**j**<**ncC**;**j**++)**

**{**

sum**((**int**)**lookup**(**i**,**0**),**j**)+=**S**(**i**,**j**);**

**}**

// increase count for the cluster mean by 1

count**((**int**)**lookup**(**i**,**0**),**0**)++;**

**}**

// update mean for each cluster

// update epsilon and sigma

e**=**e**\***0.9**;**

sigma**=**sigma**\***0.9**;**

**}** **while(**d**<**kmax **&&** change**==true);**

**return** lookup**;**

**}**

1. **Testing Kohenan Maps (testKmaps.cpp)**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

**using** **namespace** std**;**

#define Usage "Usage: ./testKohonen data\_file nf k CM\_file cl\_file \n"

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

**{**

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

**if** **(**argc **^** 6**)** **{**

cout **<<** Usage**;**

exit**(**1**);**

**}**

int nf**=**atoi**(**argv**[**2**]);**

int nc**=**nf**+**1**;**

Matrix Tr**=**readData**(**argv**[**1**],**nc**);**

char **\***cm\_file**=**argv**[**4**];**

char **\***cl\_file**=**argv**[**5**];**

int k**=**atoi**(**argv**[**3**]);**

int nrTr**=**Tr**.**getRow**();**

Matrix mTr**=**subMatrix**(**Tr**,**0**,**0**,**nrTr**-**1**,**nf**-**1**);**

Matrix oTrL**(**nrTr**,**1**);**

// define arbitrary clr cluster means

Matrix C**(**k**,**nf**);**

// call kmeans to get lookup table that refers each pixel in original pic to updated cluster mean pixel

Matrix lookup**;**

lookup**=**kohonenMap**(**mTr**,**C**,** 1**,** 200**,** 4**,** 1**,**0.01**);**

// find which cluster mean belongs to which class in supervised method

Matrix x**(**k**,**k**);**

**for(**int i**=**0**;**i**<**nrTr**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**k**;**j**++)**

**{**

**if(**lookup**(**i**,**0**)==**j**)**

**{**

x**(**j**,**Tr**(**i**,**nf**))+=**1**;**

**}**

**}**

**}**

Matrix s**(**1**,**k**);**

Matrix pos**(**1**,**k**);**

Matrix clookup**(**k**,**1**);**

Matrix d**;**

Matrix output**,** cm**;**

int error**=**0**;**

output**.**createMatrix**(**nrTr**,**2**);**

cm**.**createMatrix**(**k**+**1**,**k**+**1**);**

**for(**int j**=**0**;**j**<**k**;**j**++)**

**{**

d**=**subMatrix**(**x**,**j**,**0**,**j**,**k**-**1**);**

insertsort**(**d**,**s**,**pos**);**

clookup**(**j**,**0**)=**pos**(**0**,**k**-**1**);**

**}**

// assign class according to lookup content using clookup mapping

**for(**int i**=**0**;**i**<**nrTr**;**i**++)**

**{**

oTrL**(**i**,**0**)=**clookup**(**lookup**(**i**,**0**),**0**);**

output**(**i**,**0**)=**Tr**(**i**,**nf**);**//actual result

output**(**i**,**1**)=**oTrL**(**i**,**0**);**//classification result

**if(**oTrL**(**i**,**0**)!=**Tr**(**i**,**nf**))**

error**++;**

**}**

// create confusion matrix

//generate CM

cm**(**0**,**0**)=**0**;**

**for** **(**int i**=**0**;**i**<**k**;**i**++)**

**{** cm**(**0**,**i**+**1**)=**i**;**

cm**(**i**+**1**,**0**)=**i**;**

**}**

**for(**int r**=**0**;**r**<**k**;**r**++){**

**for(**int i**=**0**;**i**<**k**;**i**++)**

**{**

int w**=**0**;**

**for(**int j**=**0**;**j**<**nrTr**;**j**++)**

**{**

**if** **((**output**(**j**,**1**)==**r**)&&(**output**(**j**,**0**)==**i**))**

w**++;**

**}**

cm**(**i**+**1**,**r**+**1**)=**w**;**

**}**

**}**

//writeData(output,"output");

writeData**(**cm**,**cm\_file**);**

writeData**(**output**,**cl\_file**);**

cout**<<**x**<<**endl**;**

cout**<<**cm**<<**endl**;**

cout**<<**"error rate = "**<<(**double**)**error**/**nrTr**<<**endl**;**

**return** 0**;**

**}**

**Appendix D: Fusion codes:**

1. **Majority Voting (plurality.cpp)**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

**using** **namespace** std**;**

#define Usage "Usage: ./testPlurality cl1\_output\_file cl2\_output\_file cl3\_output\_file c \n"

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

**{**

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

**if** **(**argc **^** 5**)** **{**

cout **<<** Usage**;**

exit**(**1**);**

**}**

// read data from from source in command arguments

int c**=**atoi**(**argv**[**4**]);**

Matrix cl1**=**readData**(**argv**[**1**],**2**);**

Matrix cl2**=**readData**(**argv**[**2**],**2**);**

Matrix cl3**=**readData**(**argv**[**3**],**2**);**

int nr**=**cl1**.**getRow**();**

// prepare data for plurality fusion

Matrix T**=**subMatrix**(**cl1**,**0**,**0**,**nr**-**1**,**0**);**

Matrix S1**=**subMatrix**(**cl1**,**0**,**1**,**nr**-**1**,**1**);**

Matrix S2**=**subMatrix**(**cl2**,**0**,**1**,**nr**-**1**,**1**);**

Matrix S3**=**subMatrix**(**cl3**,**0**,**1**,**nr**-**1**,**1**);**

// test the lookup table

Matrix d**(**1**,**c**);**

Matrix init**(**1**,**c**);**

Matrix s**(**1**,**c**);**

Matrix pos**(**1**,**c**);**

int error**=**0**;**

**for(**int i**=**0**;**i**<**nr**;**i**++)**

**{**

d**=**init**;**

d**(**0**,**S1**(**i**,**0**))+=**1**;**

d**(**0**,**S2**(**i**,**0**))+=**1**;**

d**(**0**,**S3**(**i**,**0**))+=**1**;**

insertsort**(**d**,**s**,**pos**);**

**if(**pos**(**0**,**c**-**1**)!=**T**(**i**,**0**))**

**{**

error**++;**

**}**

cout**<<**S1**(**i**,**0**)<<**" "**<<**S1**(**i**,**0**)<<**" "**<<**S1**(**i**,**0**)<<**" "**<<**T**(**i**,**0**)<<**" "**<<**pos**(**0**,**c**-**1**)<<**endl**;**

**}**

cout**<<**"The error rate is: "**<<(**double**)**error**/**nr**<<**endl**;**

**return** 0**;**

**}**

1. **Naïve Bayes (NB.cpp)**

#include <iostream>

#include <cstdlib>

#include <cmath>

#include "time.h"

#include "Pr.h"

#define usage "Usage: ./NB CM1 CM2 CM3 io1 io2 io3 \n\n"

**using** **namespace** std**;**

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

**{**

**if** **(**argc **<** 7**)** **{**

cout **<<** usage**;**

exit**(**1**);**

**}**

Matrix cm1**,**cm2**,**cm3**,**lookup**,**sub1**,**sub2**,**sub3**,**fusion**,**rank**,**io1**,**io2**,**io3**;**

rank**.**createMatrix**(**4**,**2**);**

int i**,**j**=**0**,**k**=**0**,**h**=**0**,**nr**,**result**,**error**=**0**;**

double a**,**b**;**

cm1 **=** readData**(**Set**[**1**],**5**);**

cm2 **=** readData**(**Set**[**2**],**5**);**

cm3 **=** readData**(**Set**[**3**],**5**);**

io1**=**readData**(**Set**[**4**],**2**);**

io2**=**readData**(**Set**[**5**],**2**);**

io3**=**readData**(**Set**[**6**],**2**);**

nr**=**io1**.**getRow**();**

//crating lookup table

lookup**.**createMatrix**(**64**,**4**);**

**for(**i**=**0**;**i**<**64**;**i**++)**

**{**

**if(**i**%**4**==**0**)**

k**=**0**;**

**if((**i**!=**0**)&&**i**%**4**==**0**)**

j**++;**

**if(**j**==**4**)**

j**=**0**;**

**if((**i**!=**0**)&&(**i**%**16**)==**0**)**

h**++;**

lookup**(**i**,**0**)=**h**;**

lookup**(**i**,**1**)=**j**;**

lookup**(**i**,**2**)=**k**;**

k**++;**

**}**

//fusion calcuation

**for(**i**=**0**;**i**<**64**;**i**++)**

**{**

sub1**=**subMatrix**(**cm1**,**1**,**lookup**(**i**,**0**)+**1**,**4**,**lookup**(**i**,**0**)+**1**);**

sub3**=**subMatrix**(**cm2**,**1**,**lookup**(**i**,**1**)+**1**,** 4**,**lookup**(**i**,**1**)+**1**);**

sub2**=**subMatrix**(**cm3**,**1**,** lookup**(**i**,**2**)+**1**,**4**,**lookup**(**i**,**2**)+**1**);**

fusion**=**sub1**\***sub2**\***sub3**;**

**for(**j**=**0**;**j**<**4**;**j**++)**

**{**

rank**(**j**,**0**)=**j**;**

rank**(**j**,**1**)=**fusion**(**0**,**j**);**

**}**

**for(**j**=**1**;**j**<**4**;**j**++)**

**if(**rank**(**0**,**1**)<**rank**(**j**,**1**))**

**{**

a**=**rank**(**0**,**1**);**

b**=**rank**(**0**,**0**);**

rank**(**0**,**1**)=**rank**(**j**,**1**);**

rank**(**0**,**0**)=**rank**(**j**,**0**);**

rank**(**j**,**0**)=**b**;**

rank**(**j**,**1**)=**a**;**

**}**

lookup**(**i**,**3**)=**rank**(**0**,**0**);**

**}**

cout**<<**lookup**;**

**for(**int l**=**0**;**l**<**nr**;**l**++){**

**for** **(**i**=**0**;**i**<**64**;**i**++)**

**if((**io1**(**l**,**1**)==**lookup**(**i**,**0**))&&(**io2**(**l**,**1**)==**lookup**(**i**,**1**))&&(**io3**(**l**,**1**)==**lookup**(**i**,**2**)))**

result**=**lookup**(**i**,**3**);**

**if(**result **!=**io3**(**l**,**0**))**

error**++;**

**}**

cout**<<**error**/(**double**)**nr**<<**endl**;**

**return** 0**;**

**}**

1. **Behavior knowledge space (BKS.cpp)**

#include "Matrix.h"

#include "Pr.h"

#include <iostream>

#include <cstdlib>

#include <cmath>

**using** **namespace** std**;**

/\*\*

\* @param

\* @param

\*\*/

// works for three classifiers only

Matrix bks**(**Matrix **&**T**,** Matrix **&**S1**,**Matrix **&**S2**,**Matrix **&**S3**,** int c**)**

**{**

int nr**=**T**.**getRow**();**

Matrix F**(**pow**(**c**,**3**),**3**+**c**+**1**);** // 3 cols for classifier result combination; c cols for no. of samples of one of c classes as truth; 1 col for fused label

Matrix d**(**1**,**c**);**

Matrix s**(**1**,**c**);**

Matrix pos**(**1**,**c**);**

int r**=**0**;**

**for(**int i**=**0**;**i**<**c**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**c**;**j**++)**

**{**

**for(**int k**=**0**;**k**<**c**;**k**++)**

**{**

F**(**r**,**0**)=**i**;**F**(**r**,**1**)=**j**;**F**(**r**,**2**)=**k**;**

**for(**int l**=**0**;**l**<**nr**;**l**++)**

**{**

**if(**S1**(**l**,**0**)==**i **&&** S2**(**l**,**0**)==**j **&&** S3**(**l**,**0**)==**k**)**

**{**

F**(**r**,**T**(**l**,**0**)+**3**)+=**1**;**

**}**

d**=**subMatrix**(**F**,**r**,**3**,**r**,**3**+**c**-**1**);**

insertsort**(**d**,**s**,**pos**);**

F**(**r**,**3**+**c**)=**pos**(**0**,**c**-**1**);**

**}**

r**++;**

**}**

**}**

**}**

**return** F**;**

**}**

1. **Testing BKS(testBKS.cpp)**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

**using** **namespace** std**;**

#define Usage "Usage: ./testBKS cl1\_output\_file cl2\_output\_file cl3\_output\_file c \n"

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

**{**

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

**if** **(**argc **^** 5**)** **{**

cout **<<** Usage**;**

exit**(**1**);**

**}**

// read data from from source in command arguments

int c**=**atoi**(**argv**[**4**]);**

Matrix cl1**=**readData**(**argv**[**1**],**2**);**

Matrix cl2**=**readData**(**argv**[**2**],**2**);**

Matrix cl3**=**readData**(**argv**[**3**],**2**);**

int nr**=**cl1**.**getRow**();**

// prepare data for bks

Matrix T**=**subMatrix**(**cl1**,**0**,**0**,**nr**-**1**,**0**);**

Matrix S1**=**subMatrix**(**cl1**,**0**,**1**,**nr**-**1**,**1**);**

Matrix S2**=**subMatrix**(**cl2**,**0**,**1**,**nr**-**1**,**1**);**

Matrix S3**=**subMatrix**(**cl3**,**0**,**1**,**nr**-**1**,**1**);**

// call bks

Matrix F**=**bks**(**T**,**S1**,**S2**,**S3**,**c**);**

writeData**(**F**,**"lookupBKS.txt"**);**

cout**<<**F**<<**endl**;**

// test the lookup table

int error**=**0**;**

**for(**int i**=**0**;**i**<**nr**;**i**++)**

**{**

**for(**int j**=**0**;**j**<**pow**(**c**,**3**);**j**++)**

**{**

**if(**S1**(**i**,**0**)==**F**(**j**,**0**)** **&&** S2**(**i**,**0**)==**F**(**j**,**1**)** **&&** S3**(**i**,**0**)==**F**(**j**,**2**))**

**{**

**if(**F**(**j**,**3**+**c**)!=**T**(**i**,**0**))**

**{**

error**++;**

**}**

**break;**

**}**

**}**

**}**

cout**<<**"The error rate is: "**<<(**double**)**error**/**nr**<<**endl**;**

**return** 0**;**

**}**