Report for HW4

Yang, Yunfan **Professor Tu**

yuy130@ucsd.edu

Abstract

Facial recognition is an important task to study. A lot of approaches have been developed over the past years, such as PCA (Principal Component Analysis). The traditional PCA approach has its limitations such as it cannot handle errors well. A new approach called RPCA (Robust Component Analysis) shows its superior than its predecessor PCA. This report compares RPCA with PCA.

1 Introduction

This paper compares the effectiveness of PCA and RPCA on facial recognition task under different light settings. The traditional approach of PCA seems not able to handle errors well, such as different light settings, sunglasses. This paper explores whether RPCA can handle error better than traditional PCA approach.

2 Methodology

I study PCA and RPCA on yale dataset of 38 different people under different light settings.

PCA analysis

PCA analysis can be formulated as:
Given a set of data $X = \{x_1, x_2, ..., x_n\}$.
We want to minimize the reconstruction error: minimize $|X - A|$, s.t. $rank(A) \leq K$

To minimize the reconstruction error, we can calculate the covariance matrix $\hat{X}\hat{X}^T$, where $\hat{X} = X - \bar{X}$ by the definition of covariance matrix. Eigen decomposition can be used to decompose a matrix into components that are orthogonal to each other.

Since covariance matrix is symmetric, it is diagonalizable. So the covariance matrix can be eigen decomposed. To find the eigen decomposition, we need to find $\hat{X}\hat{X}^T = UDU^T$.

The naive approach is to apply eigen decomposition, and there are a lot of exisiting algorithm that can perform eigen decomposition. After eigen decomposition, each row becomes linearly independent. Then we just need to sort each component or each row based on the value of its diagonal matrix because the singular values on its diagonal represents the variance in each direction.

An alternative approach is to perform SVD (singular value decomposition). This is because by applying SVD on $\hat{X}$, we can get $\hat{X} = U\Sigma V^T$. $\hat{X}\hat{X}^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T)((\Sigma V^T)^T U^T) = (U\Sigma V^T)(V\Sigma^T U^T) = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma^2 U^T$. As we can see here, this expression is same as eigen decomposition's. Thus, we can use SVD to find PCA principle components. In practice, using SVD to find principle components is a lot faster than using eigen decomposition. In this assignment, I use SVD to find

principle components.

In the facial recognition task, the first step is to load everyone's face into a big matrix where each row of the matrix representing one face image. After performing PCA analysis, the principle components are extracted. In this task, the principle components are the eigenfaces. Eigenfaces means the faces that is deviated from the mean face from all the people. Then, we just need to use most important eigenfaces to represent the training faces through projection. To classify, I can project a face to eigen spaces and use a linear combination of eigenfaces to represent this face. Then, comparing the coefficients of the linear combination of eigenfaces with the training coefficients gives us the closet prediction. I use a multiclass classifier 1-NN (1 nearest neighbor) in the prediction process because it is simple and yields better result than other classifiers though taking slightly more time to compute.

RPCA analysis

RPCA analysis can be formulated as:
$D = P_\Omega[A + E + Z]$, where D is the original corrupted data, A is the recovered data, E is the error and Z is noise. Using convex optimization, we can actually separate noise and error. In the RPCA, I used $inexact_alm_rpca$ algorithm which is the fastest RPCA algorithm so far. After separating data and error, I can just apply SVD on recovered data as I did above. The rest of the process is exactly the same as PCA's.

3 Experiment

I designed six groups of tests including shuffled images and unshuffled images on PCA and RPCA. First group includes 35 testing faces per person and 40 training faces per person. Second group includes 45 testing faces per person and 30 training faces per person. Third group includes 55 testing faces per person and 20 training faces per person. Each experiment uses at most 200 eigenfaces. The results are shown in the result section.

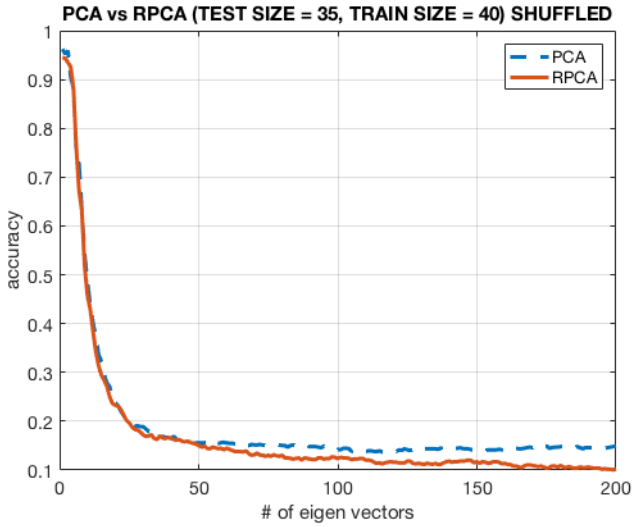
4 Conclusion

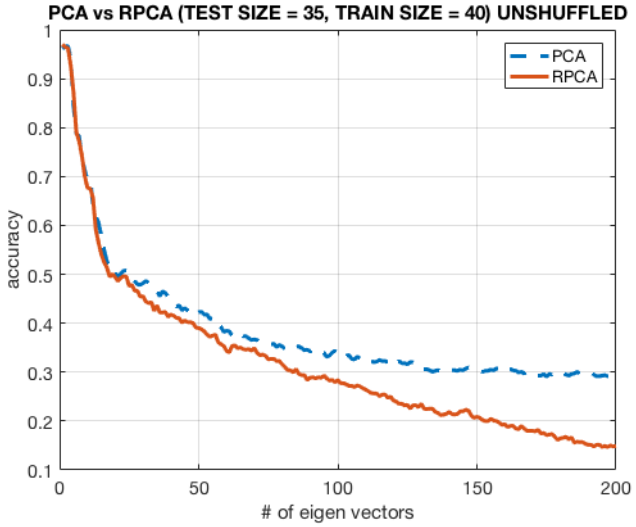
From the above experiment, RPCA outperforms PCA by upto 15%. In the unshuffled dataset, RPCA outperforms PCA by a significant margin. On the other hand, in the shuffled dataset, RPCA still outperforms PCA by 5%. So, I can conclude RPCA does provide a significant advantage over PCA. Although, performing RPCA analysis takes significant longer than PCA.

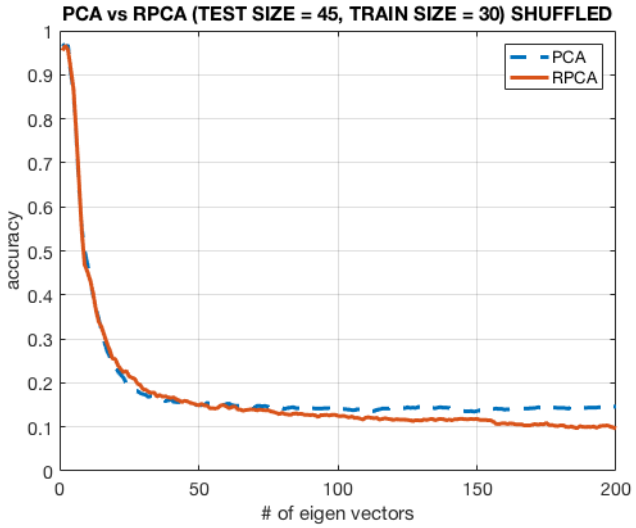
Acknowledgments

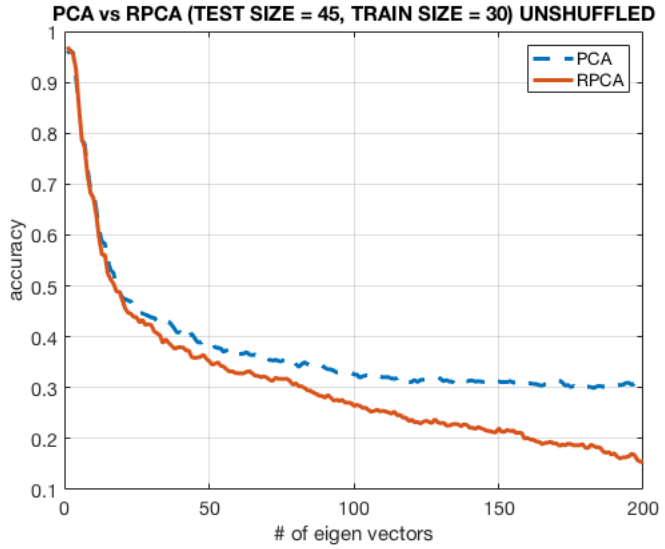
I want to expressly acknowledge the teaching and instruction provided by Prof. Zhuowen Tu. Additionally we want to highlight our exceptional TA, Saining Xie and Daniel Maryanovsky, whose mentoring and patience was instrumental for our success throughout this project and the course at large.

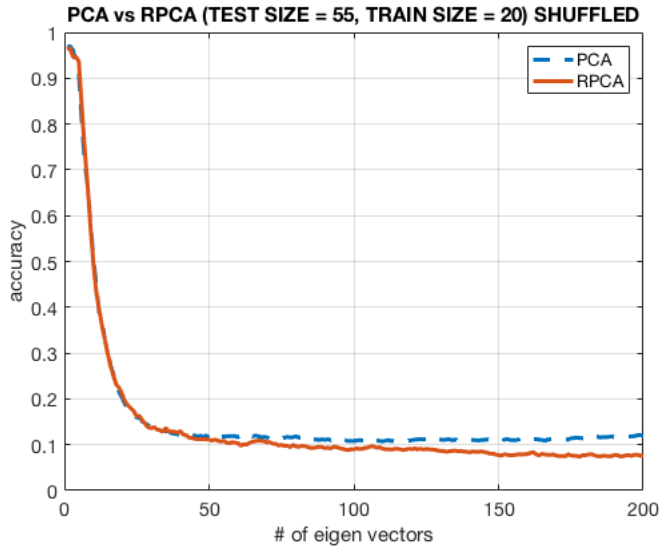
5 Results

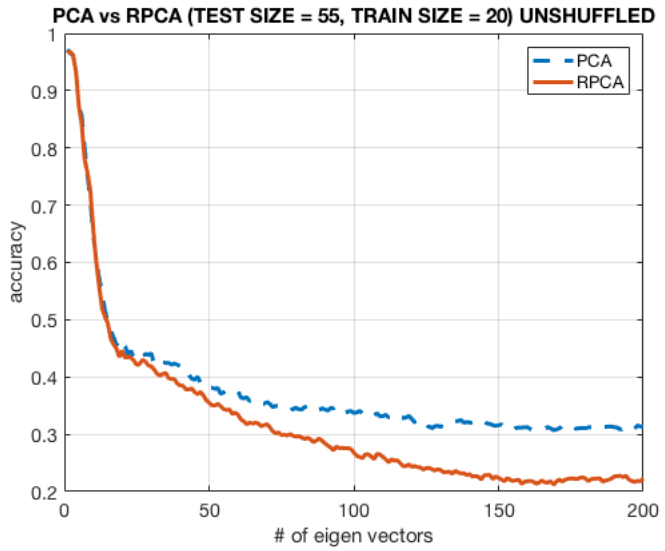

PCA vs RPCA (TEST SIZE = 35, TRAIN SIZE = 40) SHUFFLED
PCA
RPCA
accuracy
# of eigen vectors


PCA vs RPCA (TEST SIZE = 35, TRAIN SIZE = 40) UNSHUFFLED
PCA
RPCA
accuracy
# of eigen vectors


PCA vs RPCA (TEST SIZE = 45, TRAIN SIZE = 30) SHUFFLED
PCA
RPCA
accuracy
# of eigen vectors


PCA vs RPCA (TEST SIZE = 45, TRAIN SIZE = 30) UNSHUFFLED
PCA
RPCA
accuracy
# of eigen vectors
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200


PCA vs RPCA (TEST SIZE = 55, TRAIN SIZE = 20) SHUFFLED
PCA
RPCA
accuracy
# of eigen vectors
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200


PCA vs RPCA (TEST SIZE = 55, TRAIN SIZE = 20) UNSHUFFLED
PCA
RPCA
accuracy
# of eigen vectors
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
50
100
150
200

Source Code

PCA.m

```
BASEDIR = '/Users/yunfanyang/Downloads/CroppedYale/';
% # of face classes to load
CLASSES = 39;
% test size per class
TEST_SZ = 35;
% train size per class
TRAIN_SZ = 40;
% set rnd seed
SEED = 13;
% valid image size
IMG_SIZE = [192, 168];

X_train = {};
Y_train = [];
X_test = {};
Y_test = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%     load data   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Loading data..');
% load all the files
dirs = dir(BASEDIR);
% counter for # dirs loaded
ii = 1;
for i = 1 : size(dirs,1)
    PREFIX = 'yaleB';
    % match folder prefix
    if(strncmpi(dirs(i).name, PREFIX, size(PREFIX,2)))
        disp(['Loading ' dirs(i).name '..']);
        files = dir([BASEDIR dirs(i).name]);
        % shuffle files
         rng(SEED);
         files = files(randperm(size(files, 1)));

        % counter for # imgs loaded in the subdir
        jj = 1;
        % load images from subfolders
        for j = 1: size(files,1)
            % match pgm files
            if(any(regexp(files(j).name ,'.pgm$')))
                [pic, maxgray] = getpgmraw([BASEDIR '/' dirs(i).name '/' files(j).name]);

               % validate image size
                if((size(pic) ~= IMG_SIZE))
                    continue;
                end

                if(jj <= TRAIN_SZ)
                    % reshape 2D array to 1D
                    X_train = [X_train;reshape(pic,1,size(pic,1) * size(pic,2))];
                    Y_train = [Y_train;i];
                else
                    % reshape 2D array to 1D
                    X_test = [X_test;reshape(pic,1,size(pic,1) * size(pic,2))];
                    Y_test = [Y_test;i];
                end
```

```
                jj = jj + 1;
            end

            % limit % of imgs per class to load
            if(jj > TEST_SZ + TRAIN_SZ)
                break;
            end
        end
        disp(['# of images loaded ' num2str(jj)]);
        ii = ii + 1;
    end
    % limit # of classes to load
    if(ii > CLASSES)
        break;
    end
end
disp(['# of classes loaded: ' num2str(ii)]);
disp(['# of training size: ' num2str(size(X_train,1))]);
disp(['# of testing size: ' num2str(size(X_test,1))]);
% unpack cell array to array
X_train = vertcat(X_train{:});
X_test = vertcat(X_test{:});
%%%%%%%%%%%%%%%%%%%%%%%%%%%%     PCA anaylsis   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[COEFF, SCORE, latent, tsquared, explained] = pca(X_train, 'Algorithm', 'svd');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%     Train Classifier   %%%%%%%%%%%%%%%%%%%%%%%%
disp('Benchmark starts..');
ERR_TEST = [];
ERR_TRAIN = [];
ERR_TEST2 = [];
ERR_TRAIN2 = [];
for i = 1: 200
    disp(['# of Eigenvectors used ' num2str(i)]);

    meData = X_train-(ones(size(X_train,1), 1) * mean(X_train));
    X_train_prj = meData * COEFF(:, 1:i);

    meData = X_test-(ones(size(X_test,1), 1) * mean(X_test));
    X_test_prj = meData * COEFF(:, 1:i);

    % Multiclass SVM
    % Mdl = fitcecoc(X_train_prj, Y_train);

    % KNN
     Mdl = fitcknn(X_train_prj, Y_train, 'NumNeighbors',1,'Standardize',1);

    %disp('Training error rate: ')
    %ERR_TRAIN(i) = sum(predict(Mdl, X_train_prj) ~= Y_train) / size(Y_train, 1);
    %disp(ERR_TRAIN(i));

    disp('Testing error rate: ')
    ERR_TEST(i) = sum(predict(Mdl, X_test_prj) ~= Y_test) / size(Y_test, 1);
    disp(ERR_TEST(i));
end

%plot(ERR_TRAIN, 'LineWidth',3);
hold on;
plot(ERR_TEST, 'LineWidth',3);
```

5.1 RPCA.m

```
BASEDIR = '/Users/yunfanyang/Downloads/CroppedYale/';
% # of face classes to load
CLASSES = 39;
% test size per class
TEST_SZ = 45;
% train size per class
TRAIN_SZ = 30;
% set rnd seed
SEED = 13;
% valid image size
IMG_SIZE = [192, 168];

X_train = {};
Y_train = [];
X_test = {};
Y_test = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%     load data   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Loading data..');
% load all the files
dirs = dir(BASEDIR);
% counter for # dirs loaded
ii = 1;
for i = 1 : size(dirs,1)
    PREFIX = 'yaleB';
    % match folder prefix
    if(strncmpi(dirs(i).name, PREFIX, size(PREFIX,2)))
        disp(['Loading ' dirs(i).name '..']);
        files = dir([BASEDIR dirs(i).name]);
        % shuffle files
         rng(SEED);
         files = files(randperm(size(files, 1)));
        % counter for # imgs loaded in the subdir
        jj = 1;
        % load images from subfolders
        for j = 1: size(files,1)
            % match pgm files
            if(any(regexp(files(j).name ,'.pgm$')))
                [pic, maxgray] = getpgmraw([BASEDIR '/' dirs(i).name '/' files(j).name]);

                % validate image size
                if((size(pic) ~= IMG_SIZE))
                    continue;
                end

                if(jj <= TRAIN_SZ)
                    % reshape 2D array to 1D
                    X_train = [X_train;reshape(pic,1,size(pic,1) * size(pic,2))];
                    Y_train = [Y_train;i];
                else
                    % reshape 2D array to 1D
                    X_test = [X_test;reshape(pic,1,size(pic,1) * size(pic,2))];
                    Y_test = [Y_test;i];
                end
                jj = jj + 1;
            end
```

```
            % limit % of imgs per class to load
            if(jj > TEST_SZ + TRAIN_SZ)
                break;
            end
        end
        disp(['# of images loaded ' num2str(jj)]);
        ii = ii + 1;
    end
    % limit # of classes to load
    if(ii > CLASSES)
        break;
    end
end
%%%
disp(['# of classes loaded: ' num2str(ii)]);
disp(['# of training size: ' num2str(size(X_train,1))]);
disp(['# of testing size: ' num2str(size(X_test,1))]);
% unpack cell array to array
X_train = vertcat(X_train{:});
X_test = vertcat(X_test{:});
%%%%%%%%%%%%%%%%%%%%%%%%%%%%    PCA anaylsis   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X_train_mean = mean(X_train);
%X_train_new = X_train - (ones(size(X_train,1),1) * mean(X_train));
X_train_new = X_train;

[A_hat, E_hat, iter] = inexact_alm_rpca(X_train_new');
%[A_hat2, E_hat2, iter2] = inexact_alm_rpca(X_test');
%X_test = A_hat2';
X_train = A_hat';

%%
%X_test_new = X_test - (ones(100,1) * mean(X_test));
%[AA_hat, EE_hat, iiter] = inexact_alm_rpca(X_test_new');
%X_test = AA_hat';
%%
[U, S, V] = svd(A_hat, 'econ');
COEFF = U * S;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%    Train Classifier   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Benchmark starts..');
ERR_TEST = [];
ERR_TRAIN = [];
ERR_TEST2 = [];
ERR_TRAIN2 = [];
for i = 1: 200
    disp(['# of Eigenvectos used ' num2str(i)]);

    meData = X_train-(ones(size(X_train,1), 1) * mean(X_train));
    X_train_prj = meData * COEFF(:, 1:i);

    meData = X_test-(ones(size(X_test,1), 1) * mean(X_test));
    X_test_prj = meData * COEFF(:, 1:i);

    % Multiclass SVM
    %Mdl = fitcecoc(X_train_prj, Y_train);

    % KNN
    Mdl = fitcknn(X_train_prj, Y_train, 'NumNeighbors',1,'Standardize',1);
```

```
    %disp('Training error rate: ')
    %ERR_TRAIN(i) = sum(predict(Mdl, X_train_prj) ~= Y_train) / size(Y_train, 1);
    %disp(ERR_TRAIN(i));

    disp('Testing error rate: ')
    ERR_TEST(i) = sum(predict(Mdl, X_test_prj) ~= Y_test) / size(Y_test, 1);
    disp(ERR_TEST(i));
end

%plot(ERR_TRAIN, '--', 'LineWidth',3);
hold on;
plot(ERR_TEST, '--', 'LineWidth',3);
% xlabel('# of eigen vectors'); ylabel('accuracy'); title('PCA vs RPCA (TEST SIZE = 55, TRAIN
```