Seminar research methodology for data science
CS4125

Delft University of technology

COURSEWORK C

Nathan Buskulic, 4947916, n.m.m.buskulic@student.tudelft.nl
Mitchell Deen, 4396340, m.r.deen@student.tudelft.nl
Þórunn Arna Ómarsdóttir, 4917499, omarsdottir@student.tudelft.nl
Team 5

March 25, 2019

Task 1: Principal Component Analysis (PCA)

In this task we tried out two different algorithms to compute PCA modes. The main difference between them is that one of them uses the covariance matrix and the other one is based on the Gram matrix. In the following section we will be addressing the two algorithms as the covariance algorithm and the gram algorithm.

The computational time for the covariance algorithm is much higher than for the other one. That makes sense since the covariance matrix is a $d \times d$ matrix where d stands for the number of Dimensions. While the gram matrix is$m \times m$ where m is the number of data points.

Datasets

We decided to use two different high dimensional image datasets for this task.

Celebrity faces

The first one is a dataset consisting om many images of celebrity faces [1]. We run the PCA analysis on a subset of the database, or the 200 first images. In this dataset the images consist of 38804 pixels. So when computing PCA with the covariance matrix the matrix is of size 38804x38804 but the gram matrix is only 200x200.

The code we used to read the images can be found below in the appendix. The images where already center aligned so we did not have to do anything regarding that. We turned the images to gray scale. Then we reshaped them to a vector representation so it is possible to all the images to a matrix and perform PCA on them.

PCA with Covariance Matrix

The covariance matrix yielded from the calculations is $d \times d$. Since we are working in high dimensional datasets this matrix gets very large. The calculations of eigenvectors and eigenvalues are very time and computation expensive so we only calculate the biggest eigenvector. That vector is then used for the projection of the centered, averaged image to the corresponding eigenspace.

PCA with Gram Matrix

Since we are using a dataset where the dimension of the images are bigger than the total number of image, one way is to use the Gram matrix that represents how images covary. We now have to find the eigenvectors and eigenvalues of a $m \times m$ matrix. In order to be able to project the initial points on this new space we have to transform the eigenvectors from being m-dimensional to being n-dimensional. The basis vectors of the affine space are then these lifted n-dimensional eigenvectros of the Gram matrix.

Dataset2

to be continued

Comparison

In figures 1, 2 we can see the results of the PCA algorithms on 200 images from the celebrity dataset. As can be seen they come out very similarly. The

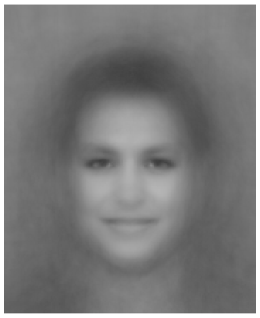

Figure 1: Results of PCA algorithm using covariance

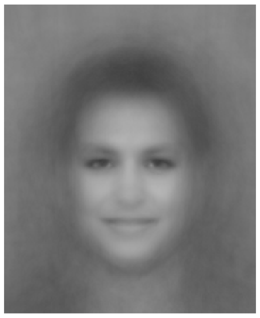

Figure 2: Results of PCA algorithm using gram matrix

The computation time for the covariance algorithm is way higher than for the gram algorithm. For the celebrity dataset the covariance takes 46.8049 seconds meanwhile the gram only takes 2.2339 seconds. If we would be calculating more than just the first eigenvector for the covariance algorithm the computation time would increase alot.

Task 2: Fast Approximation of PCA

snapshot PCA

When the dataset becomes very large, simply using the gram matrix is not a solution anymore and we need to speed the calculation in some way. This speed increase will of course cost us in term of information. The first technique that we will apply consist in reducing the number of images that we use to compute the gram matrix. These selected images are the "snapshots". Based on these snapshots we calculate the gram matrix and apply the same process that we did in the second algorithm of Task 1.
The big question is how to select these snapshots in a smart way to capture most of the variance of the data. For now we are only using a random selection of images but we want to experiment with different method to get these snapshots in a smarter way. Indeed the random selection yields quite poor results.

Task 3: Multidimensional Scaling (MDS)

Appendix

Loading images for celebrity dataset

```
initMatrix = []
imagefiles = dir('/img_align_celeba/*.jpg');
nfiles = length(imagefiles);    % Number of files found
for i=1:maxImg
    currentfilename = imagefiles(i).name
    currentimage = imread(strcat('/img_align_celeba/',currentfilename));
    currentimage = rgb2gray(currentimage);

    [h w d]=size(currentimage);
    x = double(reshape(currentimage,w*h,d))/255;
    initMatrix = [initMatrix; x'];

end
initMatrix = initMatrix';
```

Implementation of PCA using covariance matrix

```
% Get the center
center = 1/size(initMatrix,2) * sum(initMatrix,2);

% Get the centerd points
y = initMatrix - center;

% compute the Covarience matrix
covarience = y * y';

% get the eigenvalues and eigenvectors
[V,D] = eigs(covarience,1);

%find the biggest eigenvectos
em1=V(:,1);

matrixCenter = zeros(size(center,1), size(V,2));
for i = 1:size(V,2)
  matrixCenter(:,i) = center;
end
reconstructedMatrix = (y' * em1 * em1')' + matrixCenter;
reconstructedMatrix;
```

Implementation of PCA using gram matrix

```
% Get the center
center = 1/size(initMatrix,2) * sum(initMatrix,2);

% Get the centerd points
y = initMatrix - center;

% get the Gram Matrix
gram = y' * y;
```

```
% get the eigenvalues and eigenvectors
[V,D] = eig(gram);
% We change the order of the vectors
D = rot90(fliplr(D),-1);
V = flip(V,2);

% We get the basis vectors
U = zeros(size(y,1), size(D,1));
for i = 1:size(D,1)
  U(:,i) = (y * V(:,i)) * (1/(sqrt(D(i,i))));
end
U;

matrixCenter = zeros(size(center,1), size(U,2));
for i = 1:size(U,2)
  matrixCenter(:,i) = center;
end
reconstructedMatrix = (y' * U * U')' + matrixCenter;
%reconstructedMatrix = (initMatrix' * U * U')' + matrixCenter;
reconstructedMatrix;
```

Implementation of fast PCA with snapshots selected randomly

```
% Get the snapshots
% We select them randomly for now
nbSnapshots = 200;
indicesSnapshots = randperm(maxImg,nbSnapshots);
snapshotMatrix = initMatrix(:,indicesSnapshots);

% Get the center
center = mean(initMatrix,2);
centeredPoints = initMatrix - center;

% Get the centerd points
y = snapshotMatrix - center;

% get the Gram Matrix
gram = y' * y;

% get the eigenvalues and eigenvectors
[V,D] = eig(gram);
% We change order the vectors in increasing order
D = rot90(fliplr(D),-1);
V = flip(V,2);

% We get the basis vectors
U = zeros(size(initMatrix,1), size(D,1));
for i = 1:size(D,1)
  U(:,i) = (1/(sqrt(D(i,i)))) * (y * V(:,i));
end
U;

% We project on the new space
matrixCenter = zeros(size(initMatrix,1), size(initMatrix,2));
for i = 1:size(U,2)
  matrixCenter(:,i) = center;
```

```
end

%size(centeredPoints' * U * U')
reconstructedMatrix = (centeredPoints' * U * U')' + matrixCenter;
```

References

[1] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.