CPSC 340: Assignment5

Yiwei Hou(84435156)
Qiong Zhang(85896158)

1 Principal Component Analysis

1.1 PCA by Hand

1. The original data set is $X = \begin{pmatrix} -2 & -1 \\ -1 & 0 \\ 0 & 1 \\ 1 & 2 \\ 2 & 3 \end{pmatrix}$. After centralization, $\tilde{X} = \begin{pmatrix} -2 & -2 \\ -1 & -1 \\ 0 & 0 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}$, which means after centralization, the observation for the first feature and the second feature are the same. When plot the observations where x_1 is the x-axis and x_2 is the y-axis, then the observations fall on the line $x_2 = x_1$. Thus, we have $w_1 = w_2$ and the first principal component is $W = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$ since $||W|| = 1$.
2. Since point $(3, 3)$ also falls on the line $x_2 = x_1$, thus the reconstruction error is 0.
3. The reconstruction error for point $(3, 4)$ is the distance of point $(3, 4)$ to the line $x_2 = x_1$, thus the reconstruction error equals $\frac{\sqrt{2}}{2}$.

1.2 Data Visualization

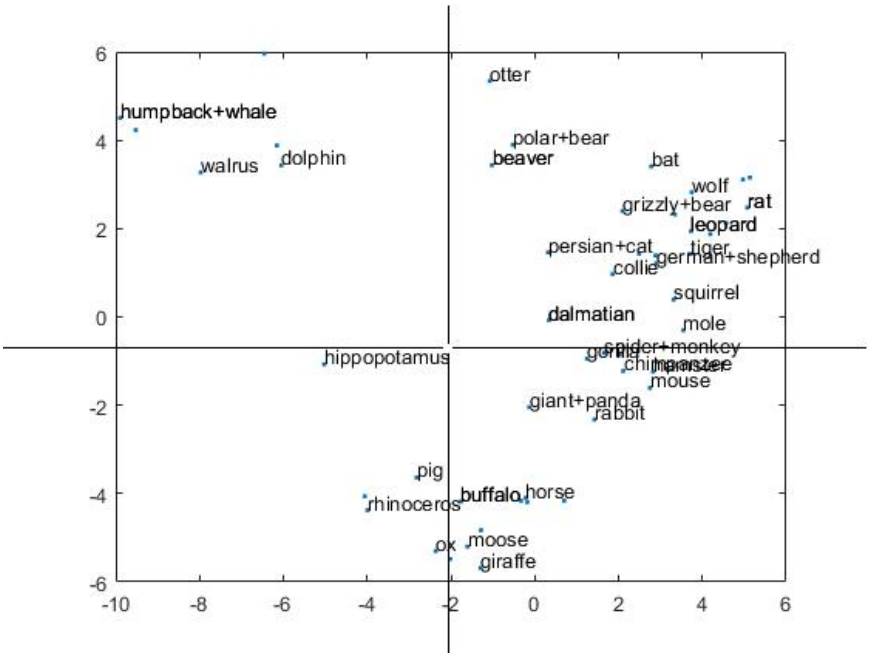


Figure 1: Scatter plot for the weight on the first two principal components

The demo:

```
load animals.mat
[n,d] = size(X);
X = standardizeCols(X);
model=dimRedPCA(X,2);
Z=model.compress(model,X);
plot(Z(:,1),Z(:,2),'.');
gname(animals);
```

1.3 Data Compression

1. The proportion of the variances can be explained by two-dimensional representation is calculated by $1 - \frac{||ZW-X||_F^2}{||X||_F^2} = 16.45\%$.

2. By taking different values for k, we find out 14 PCS are required to explain 50% of the variance in the data.

2 PCA Generalizations

2.1 Robust PCA

```
function [model] = dimRedRPCA(X,k)

[n,d] = size(X);
epsilon=0.0001;

% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);

% Initialize W and Z
W = randn(k,d);
Z = randn(n,k);

R = Z*W-X;
f = sum(sum(sqrt(R.^2+epsilon)));
for iter = 1:50
    fOld = f;

    % Update Z
    Z(:) = findMin(@funObjZ,Z(:),10,0,X,W);

    % Update W
    W(:) = findMin(@funObjW,W(:),10,0,X,Z);

    R = Z*W-X;
    f = sum(sum(sqrt(R.^2+epsilon)));
    fprintf('Iteration %d, loss = %.5e\n',iter,f);

    if fOld - f < 1e-4
        break;
    end
```

```
end

model.mu = mu;
model.W = W;
model.compress = @compress;
model.expand = @expand;
end

function [Z] = compress(model,X)
[t,d] = size(X);
mu = model.mu;
W = model.W;
k = size(W,1);

X = X - repmat(mu,[t 1]);
% We didn't enforce that W was orthogonal so we need to optimize to find Z
Z = zeros(t,k);
Z(:) = findMin(@funObjZ,Z(:),100,0,X,W);
end

function [X] = expand(model,Z)
[t,d] = size(Z);
mu = model.mu;
W = model.W;

X = Z*W + repmat(mu,[t 1]);
end

function [f,g] = funObjW(W,X,Z)
% Resize vector of parameters into matrix
d = size(X,2);
k = size(Z,2);
W = reshape(W,[k d]);
epsilon=0.0001;
% Compute function value
R = Z*W-X;
f = sum(sum(sqrt(R.^2+epsilon)));

% Compute derivative with respect to each residual
dR = R./sqrt(R.^2+epsilon);

% Multiply by Z' to get elemetns of gradient
g = Z'*dR;

% Return a vector
g = g(:);
end

function [f,g] = funObjZ(Z,X,W)
% Resize vector of parameters into matrix
n = size(X,1);
k = size(W,1);
epsilon=0.0001;
Z = reshape(Z,[n k]);

```

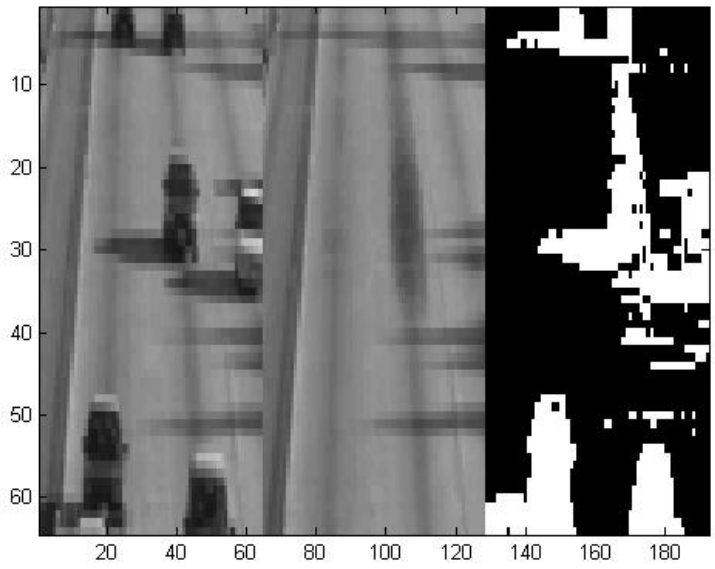
```
% Compute function value
R = Z*W-X;
f = sum(sum(sqrt(R.^2+epsilon)));

% Compute derivative with respect to each residual
dR = R./sqrt(R.^2+epsilon);

% Multiply by W' to get elements of gradient
g = dR*W';

% Return a vector
g = g(:);
end
```

The ouput of the highway example by robust PCA is given by



2.2 L1-Regularized and Binary Latent-Factor Models

1. 1.Higher value of $_W$ will encourage the sparsity of W and does not directly affect the sparsity of Z. λ_Z controls the shrinkage of Z but does not directly affect the sparsity of W.

2. The fundamental trade-off in machine learning is a trade-off between how small you can make the training error and how well training error approximates the test error. As mentioned in 1, the higher the λ_z, the result is more robust which means the training error approximates the test error well, but the training error becomes larger. In terms of the effect of the number of principal component k on fundamental trade, the higher the k value, the smaller the training error. When $k = d$, the error is smallest. However, the large k values may lead to over-fitting which means the test error would be high.

3. If λ_W is zero, the answer in (2) will not change as λ_W only affect the sparsity of W and with zero $_W$, we don't penalize large coefficients. But the model fit is determined by the product of Z and W. It would not directly affect the training and test error.

4.

$$f(Z,W) = \sum_{(i,j)\in\mathcal{R}} \frac{1}{1+\exp(-x_{ij}w_j^T z_i)} + \lambda_W \sum_{j=1}^{d} ||w_j||_1 + \frac{\lambda_Z}{2}\sum_{i=1}^{n} ||z_i||^2$$

3 Multi-Dimensional Scaling

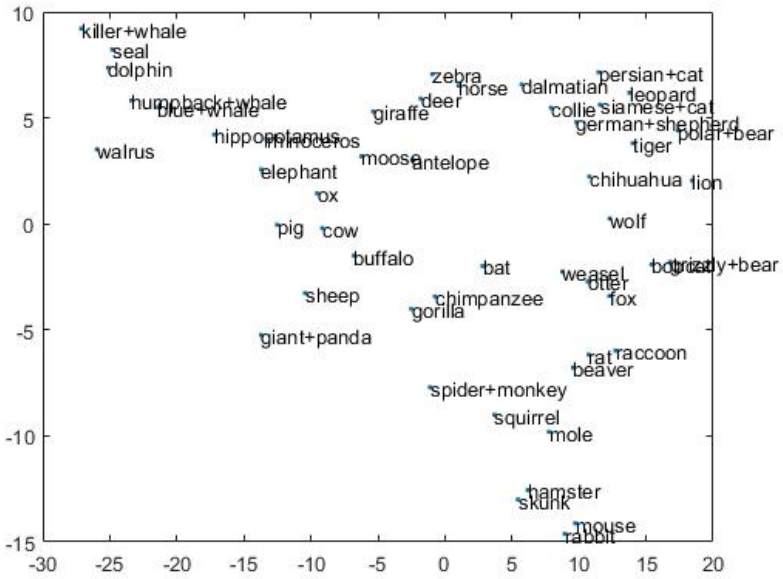
3.1 ISOMAP

We can implement the *visualizeMDS* function by a different distance function given by

```
% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));

% Intialize the KNN graph
G=zeros(n,n);
% Find the 3 nearest neighbours
[~,pos]=sort(D,1);
neighbour=pos(2:4,:);
% include an edge i to j if i is a KNN of j or if j is a KNN of i
for i=1:n;
    for j=neighbour(:,i);
            G(i,j)=D(i,j);
            G(j,i)=D(j,i);
    end
end
%computes the approximate geodesic distance
D=zeros(n);
for i=1:n;
    for j=1:n;
        D(i,j)=dijkstra(G,i,j);
    end
end
```

The final result is shown below.



3.2 ISOMAP with Disconnected Graph

```
%computes the approximate geodesic distance
D=zeros(n);
for i=1:n;
    for j=1:n;
        D(i,j)=dijkstra(G,i,j);
    end
end
D(isinf(D))=max(max(~isinf(D)));
```

The final result is shown below.