CPSC340A4

Qinglan Huang n6v9a   Liu Yang v4d9

November 29, 2017

1 PCA Generalizations

1.1 Robust PCA

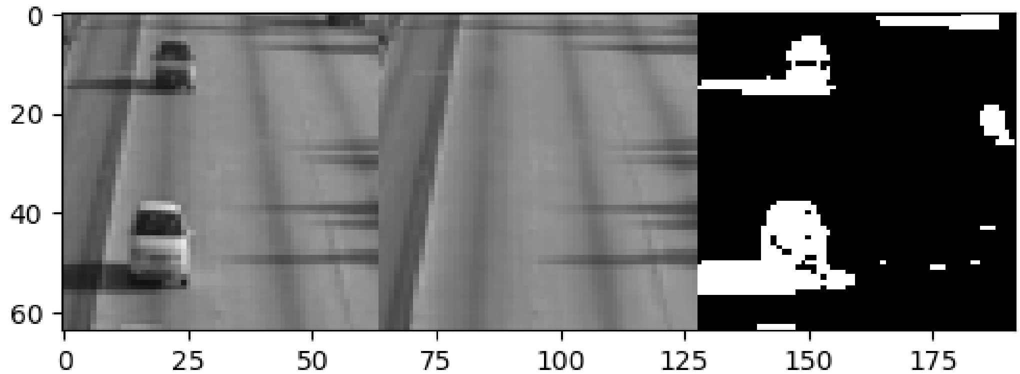
The code:

```
function RPCA(X,k)
    (n,d) = size(X)

    # Subtract mean
    mu = mean(X,1)
    X -= repmat(mu,n,1)

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

    R = Z*W - X
    ep = 0.0001
    f = sum(sqrt.(R.^2 + ep))
    funObjZ(z) = rpcaObjZ(z,X,W)
    funObjW(w) = rpcaObjW(w,X,Z)
    for iter in 1:50
        fOld = f

        # Update Z
        Z[:] = findMin(funObjZ,Z[:],verbose=false,maxIter=10)

        # Update W
        W[:] = findMin(funObjW,W[:],verbose=false,maxIter=10)

        R = Z*W - X
        f = sum(sqrt.(R.^2 + ep))
        @printf("Iteration %d, loss = %f\n",iter,f/length(X))

```

```
        if (fOld - f)/length(X) < 1e-2
            break
        end
    end


    # We didn't enforce that W was orthogonal so we need to optimize to find Z
    compress(Xhat) = rcompress_gradientDescent(Xhat,W,mu)
    expand(Z) = expandFunc(Z,W,mu)

    return CompressModel(compress,expand,W)
end

function rcompress_gradientDescent(Xhat,W,mu)
    (t,d) = size(Xhat)
    Xcentered = Xhat - repmat(mu,t,1)
    Z = zeros(t,k)

    funObj(z) = rpcaObjZ(z,Xcentered,W)
    Z[:] = findMin(funObj,Z[:],verbose=false)
    return Z
end



function rpcaObjZ(z,X,W)
    # Rezie vector of parameters into matrix
    n = size(X,1)
    k = size(W,1)
```

```
    Z = reshape(z,n,k)

    # Comptue function value
    ep = 0.0001
    R = Z*W - X
    f = sum(sqrt.(R.^2 + ep))

    # Comptue derivative with respect to each residual
    dR = R./sqrt.(R.^2 + ep)

    # Multiply by W' to get elements of gradient
    G = dR*W'

    # Return function and gradient vector
    return (f,G[:])
end

function rpcaObjW(w,X,Z)
    # Rezie vector of parameters into matrix
    d = size(X,2)
    k = size(Z,2)
    W = reshape(w,k,d)

    # Comptue function value
    ep = 0.0001
    R = Z*W - X
    f = sum(sqrt.(R.^2 +ep))

    # Comptue derivative with respect to each residual
    dR = R./sqrt.(R.^2 +ep)

    # Multiply by Z' to get elements of gradient
    G = Z'dR

    # Return function and gradient vector
    return (f,G[:])
end
```

The final graph is:



1.2 L1-Regularized and Binary Latent-Factor Models

1. L1-regularization will lead sparsity. therefore as penalty λ_w increase, it will lead more sparsity in W.

2. There is no effect to lead sparsity of W or Z, because of L2-regularization has no effect in sparisty. Thus, no matter λ_z increase or decrease, penality λ_z will not lead sparisty.

3.L2-regularization in fundamental trade-off, as we only regularize Z and add penalty λ_z, training error will increase and approximation error decrease as λ_z increase.

4.When k = 0, f(Z,W) = x_{ij}; When k increases, the training error will decrease, and approximation error increase.

5.As $\lambda_w = 0$, there is no effect to result for the question 4.

When we increase λ_z,we anticipate a decrease in approximation error as we said n q3. But when $\lambda_w$is 0,we can set z to be extremely small such that regularization penalty over z can be ignored but set $|W|$ to be large and the value of $W^T Z$ (the value of loss, f(W,Z))remains the same, no effect on fundamental tradeoff.

6.Objective function:

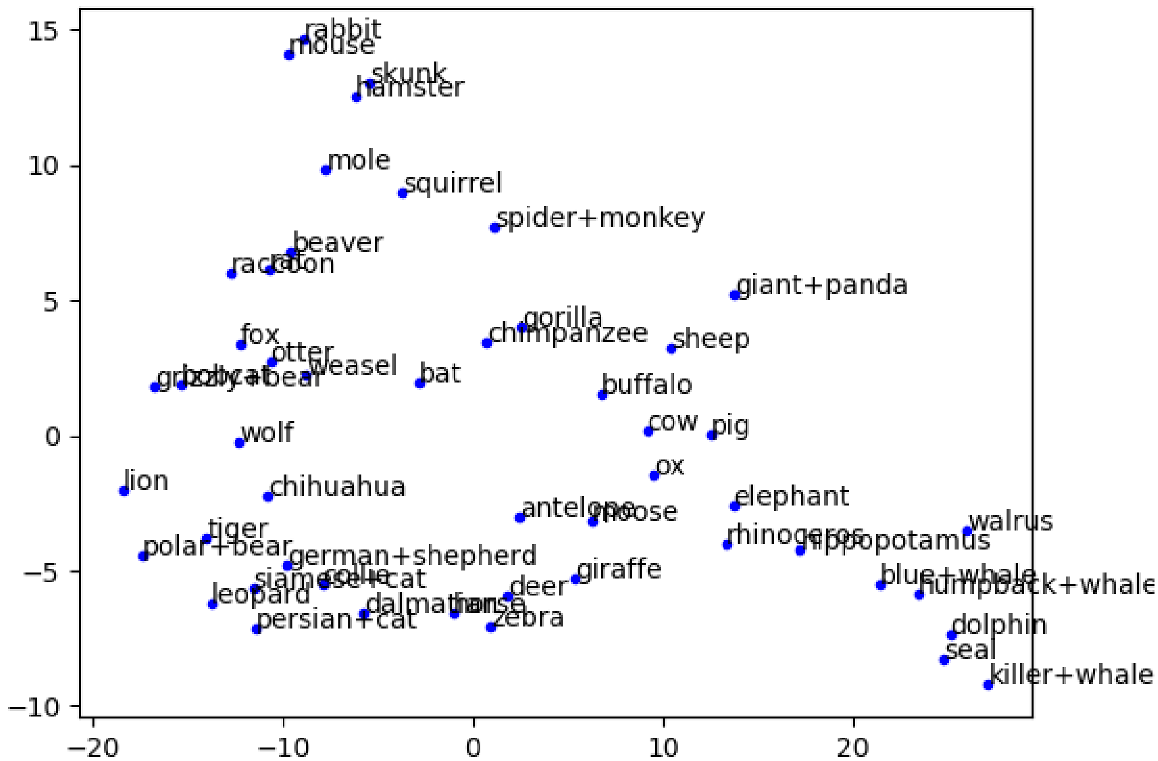
$f(Z, W) = \sum_{ij=1}^{n} \max\{0, 1 - x_{ij} w_j\} + \lambda_w \sum_{j=1}^{d} |w_j|_1 + \frac{\lambda_2}{2} \sum_{i=1}^{d} |z_i|_2$

2 Multi-Dimensional Scaling

2.1 ISOMAP

The code:

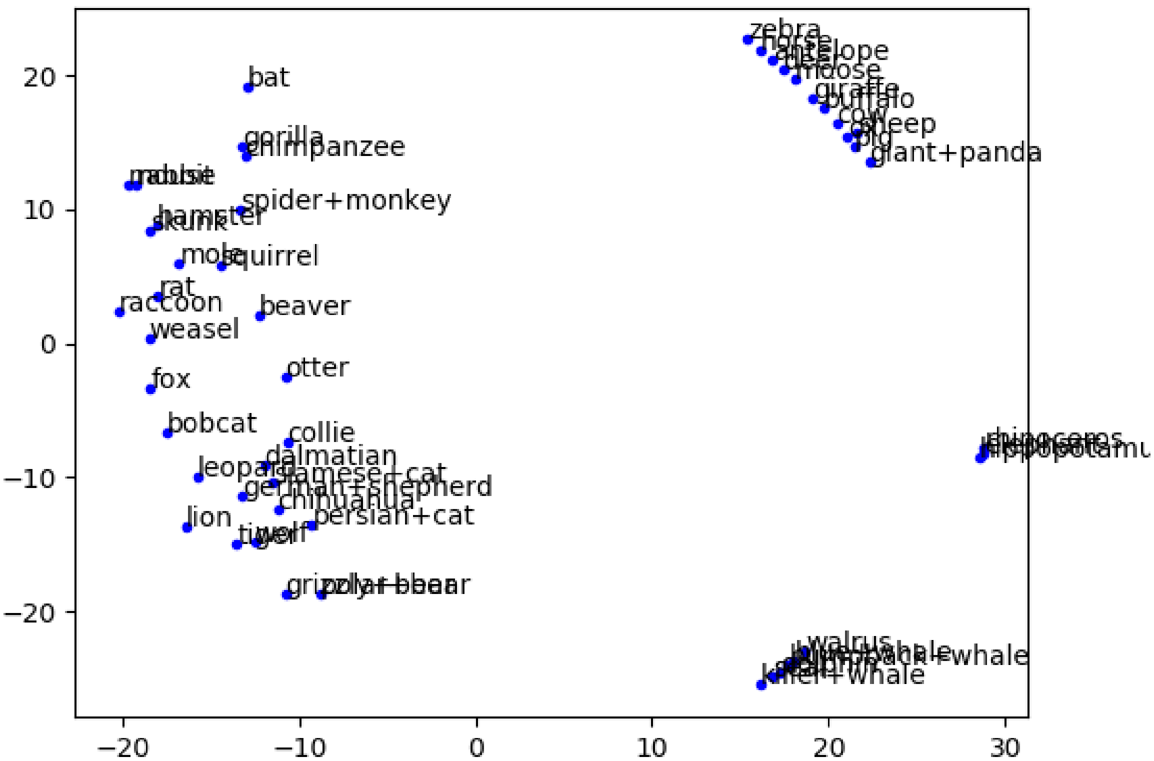
```
function ISOMAP(X,k)
    (n,d) = size(X)

    temp = distancesSquared(X,X)
    temp = sqrt.(abs.(temp))

    weight = fill(Inf, n,n)

    for i in 1:n
        v = sortperm(temp[i,:])
        weight[i,v[1:k]] = temp[i,v[1:k]]
        weight[v[1:k],i] = temp[i,v[1:k]]
    end

    D = zeros(n,n)

    for i in 1:n
        for j in 1:n
            D[i,j] = dijkstra(weight,i,j)
        end
    end

    model = PCA(X,2)
    Z = model.compress(X)

    funObj(z) = stress(z,D)

    Z[:] = findMin(funObj,Z[:])

    return Z
end
```



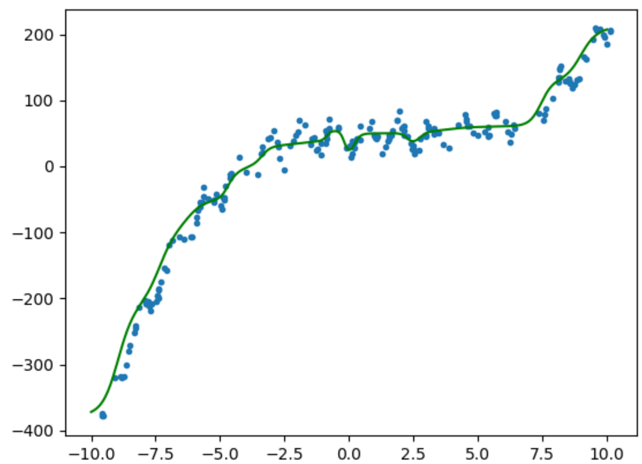
rabbit
mouse
skunk
hamster
mole
squirrel
spider+monkey
beaver
raccoon
rat
giant+panda
gorilla
chimpanzee
sheep
fox
otter
weasel
bat
grizzly+bear
bobcat
buffalo
cow
pig
wolf
ox
lion
chihuahua
elephant
antelope
moose
walrus
tiger
polar+bear
rhinoceros
hippopotamus
german+shepherd
giraffe
collie
siamese+cat
blue+whale
humpback+whale
deer
leopard
dalmatian
horse
zebra
persian+cat
dolphin
seal
killer+whale

2.2 ISOMAP with Disconnected Graph

```
function ISOMAP2(X,k)
    (n,d) = size(X)

    temp = distancesSquared(X,X)
    temp = sqrt.(abs.(temp))

    weight = fill(Inf,n,n)

    for i in 1:n
        v = sortperm(temp[i,:])
        weight[i,v[1:k]] = temp[i,v[1:k]]
        weight[v[1:k],i] = temp[i,v[1:k]]
    end

    D = zeros(n,n)

    for i in 1:n
        for j in 1:n
            D[i,j] = dijkstra(weight,i,j)
        end
    end

    a = findmax(filter(!isinf, D))

    D[isinf(D)] = a[1];

    model = PCA(X,2)
    Z = model.compress(X)

    funObj(z) = stress(z,D)

    Z[:] = findMin(funObj,Z[:])
```


zebra
antelope
moose
giraffe
buffalo
cow
sheep
pig
giant+panda
bat
gorilla
chimpanzee
spider+monkey
hamster
skunk
mole
squirrel
rat
raccoon
beaver
weasel
otter
fox
bobcat
collie
dalmatian
siamese+cat
german+shepherd
chihuahua
lion
persian+cat
wolf
grizzly+bear
walrus
humpback+whale
killer+whale
rhinoceros
20
10
0
−10
−20
−20
−10
0
10
20
30

3 Neural Networks



We increase the value for maxIter. We increase the value for nHidden.

4 Very-Short Answer Questions

1. NMP is non-convex loss function. we can use projected algorithm(projected gradient + random initialization to minimize the loss function.

2. Collaborative filtering predicts ($y_{ij} = w^T z_i$). But adding a new item means adding a column with all question marks in X. And we cannot calculate corresponding Wj. Therefore we cannot make appropriate and meaningful prediction about this new item.

3. Yes,MDS algorithm with k =2 guranteed to find representation with error of zero. But for PCA, we cannot find error with zero.

4.Euclidean distance is distance between two points. Geodesic distance is shortest path between two vertice in a graph (edge in this graph is euclidean or other distance between any two vertices).
In high dimensional space, images are on a manifold. In Euclidean distance, it doesn't reflect manifold structure. In geodesic distance, distance through space of rotations.

5.Because sigmoid is smooth and non=linear.

6. As depth increases, training error decreases.

7. L2-regularization, L1-regularization, dropout.
8. As width increases , the training error decreases