Tutorial 10

CPSC 340: Machine Learning and Data Mining

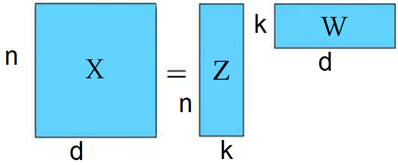
Fall 2016

Overview

1. Principal Component Analysis
   - Singular Value Decomposition (SVD)
   - Non-Negative Matrix Factorization (NMF)
2. Collaborative Filtering

Principal Component Analysis (PCA)

$$f(\mathbf{W},\mathbf{Z}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}(\mathbf{w}_j^T\mathbf{z}_i - \mathbf{x}_{ij})^2 = \frac{1}{2}\left\|\mathbf{ZW} - \mathbf{X}\right\|_F^2$$



- Can apply different constrains on $\mathbf{W}$ and $\mathbf{Z}$, e.g.,:
  - Orthogonal $\mathbf{W}$.
  - Non-negative $\mathbf{W}$ and $\mathbf{Z}$ (consequently sparse) .
  - L1-regularization on $\mathbf{W}$ and $\mathbf{Z}$ (consequently sparse).
  - ...

Solving PCA

- 3 common ways to solve this problem:
  - Singular value decomposition (SVD)classic non-iterative approach.
  - Alternating between updating $\mathbf{W}$ and updating $\mathbf{Z}$.
  - Stochastic gradient: gradient descent based on random i and j.
    - (Or just plain gradient descent).

Solving PCA: Singular Value Decomposition (SVD)

- At train: enforce orthogonality on $\mathbf{W}$ with SVD.
- At test:

$$\nabla_Z f(\mathbf{Z}) = \mathbf{ZWW}^T - \mathbf{XW}^T \rightarrow \mathbf{Z} = \mathbf{XW}^T(\mathbf{WW}^T)^{-1} = \mathbf{XW}^T$$

```
function [model] = dimRedPCA(X,k)
[n,d] = size(X);
% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);
[U,S,V] = svd(X);
W = V(:,1:k)';
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;
X = X - repmat(mu,[t 1]);
Z = 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
```

At train time, find orthogonal W given k

At test time, find optimal Z given W for new data

W is orthonormal

Solving PCA: Alternating between Updating $\mathbf{W}$ and Updating $\mathbf{Z}$

- Instead of using SVD to compute the principal components, we can alternate between updating$\mathbf{W}$ and updating $\mathbf{Z}$.
  - One way is to use a gradient method that alternates between updating $\mathbf{W}$ and $\mathbf{Z}$.
    - We get different principal components with gradient descent because we haven't set constraints on $\mathbf{W}$.
    - You would never actually use this method to fit a PCA model, but this optimization strategy generalizes to other models e.g., non-negative matrix factorization (NMF).

Solving PCA: Alternating between Updating $\mathbf{W}$ and Updating $\mathbf{Z}$

$$f(\mathbf{W}, \mathbf{Z}) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{d} (\mathbf{w}_j^T \mathbf{z}_i - \mathbf{x}_{ij})^2 = \frac{1}{2} \|\mathbf{ZW} - \mathbf{X}\|_F^2$$

- Fix $\mathbf{Z}$, find $\mathbf{W}$:

$$\nabla_W f(\mathbf{W}) = \mathbf{Z}^T \mathbf{ZW} - \mathbf{Z}^T \mathbf{X}, \mathbf{W} = (\mathbf{Z}^T \mathbf{Z})^{-1} (\mathbf{Z}^T \mathbf{X})$$

- Fix $\mathbf{W}$, find $\mathbf{Z}$:

$$\nabla_Z f(\mathbf{Z}) = \mathbf{ZWW}^T - \mathbf{XW}^T, \mathbf{Z} = \mathbf{XW}^T (\mathbf{WW}^T)^{-1}$$

Solving PCA: Alternating between Updating $\mathbf{W}$ and Updating $\mathbf{Z}$

```
function [model] = dimRedPCA_alternate(X,k)
[n,d] = size(X);
% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);
% Initialize W and Z
W = randn(k,d);
Z = randn(n,k);
f = (1/2)*sum(sum((X-Z*W).^2));
for iter = 1:50
    fOld = f;
    Z(:) = findMin(@funObjZ,Z(:),10,0,X,W);
    W(:) = findMin(@funObjW,W(:),10,0,X,Z);
    f = (1/2)*sum(sum((X-Z*W).^2));
    fprintf('Iteration %d, loss = %.5e\n',iter,f);
    if fOld - f < 1
        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;
X = X - repmat(mu,[t 1]);
% We didn't enforce that W was orthogonal so we need to solve least squares
Z = X*W'*inv(W*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
```

Initialize W and Z

findMin uses gradient descent (no constraints)

Alternatively update Z and W

Solve least squares, because orthognality isn't enforced on W

Solving PCA: Alternating between Updating $\mathbf{W}$ and Updating $\mathbf{Z}$

```
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]);
% Compute function and gradient
R = X-Z*W;
f = (1/2)*sum(sum(R.^2));
g = -Z'*R;
% 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);
Z = reshape(Z,[n k]);
% Compute function and gradient
R = X-Z*W;
f = (1/2)*sum(sum(R.^2));
g = -(R*W');
% Return a vector
g = g(:);
end
```

Non-Negative Matrix Factorization (NMF)

- NMF is solving PCA such that $\mathbf{Z}$ and $\mathbf{W}$ have non-negative terms.
- How can we minimize $f(\mathbf{W})$ with non-negative constraints?
  - Naive approach: solve least squares, set negative $\mathbf{w_{ij}}$ to 0, e.g.,

$$\mathbf{W} = (\mathbf{Z}^T\mathbf{Z})^{-1}(\mathbf{Z}^T\mathbf{X})$$

$$\mathbf{w_{ij}} = \max\{0, \mathbf{w_{ij}}\}$$

  - Generally not correct!

Non-Negative Matrix Factorization (NMF)

- How can we minimize $f(\mathbf{W})$ with non-negative constraints?
  - Correct approach: projected gradient descent.
    - Run a gradient descent iteration:

$$\mathbf{W}^{t+\frac{1}{2}} = \mathbf{W}^{T} - \alpha^{T} \triangledown f(\mathbf{W}^{t})$$

    - After each step, set negative values to 0.

$$\mathbf{w_{ij}}^{t+1} = \max\{0, \mathbf{w_{ij}}\}$$

    - Repeat.
- One way to use projected gradient:
  - Alternate between projected gradient steps on $\mathbf{W}$ and on $\mathbf{Z}$.

Exercise

- Modify the dimRedPCA_alternate function from the previous slides, to add non-negativity constraint.
  - Hint: you may use findMinNN as a black box that implements gradient descent and enforces non-negative parameters!

Exercise: Solution

```
function [model] = dimRedNMF_alternate(X,k)
[n,d] = size(X);
% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);
% Initialize W and Z
W = randn(k,d);
Z = randn(n,k);
W(W < 0) = 0;
Z(Z < 0) = 0;
f = (1/2)*sum(sum((X-Z*W).^2));
for iter = 1:50
    fOld = f;
    % Update Z
    Z(:) = findMinNN(@funObjZ,Z(:),10,0,X,W);
    % Update W
    W(:) = findMinNN(@funObjW,W(:),10,0,X,Z);
    f = (1/2)*sum(sum((X-Z*W).^2));
    fprintf('Iteration %d, loss = %.5e\n',iter,f);
    if abs(fOld - f) < 1
        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);
k = size(model.W,1);
mu = model.mu;
W = model.W;

X = X - repmat(mu,[t 1]);
Z = zeros(t,k);
Z(:) = findMinNN(@funObjZ,Z(:),500,0,X,W);
end
```

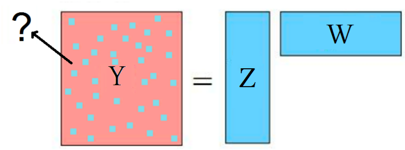
Initialize without negative values.

Use gradient descent that enforces non-negative parameters (findMinNN)!

Use gradient descent that enforces non-negative parameters (findMinNN)!

Collaborative Filtering

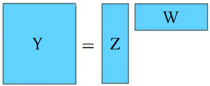
- Given a user-item interaction matrix
  - Each cell can be rating of user u for item i
  - A large number of missing values!!!
- In collaborative filtering, we are interested in filling in, or predicting, the missing values.



Collaborative Filtering

- Our standard latent-factor framework:

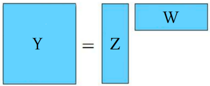
$$\underset{W,Z}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}(\mathbf{y}_{ij} - \mathbf{w}_j^T\mathbf{z}_i)^2$$



Collaborative Filtering

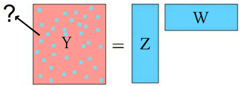
- Our standard latent-factor framework:

$$\underset{W,Z}{\text{argmin}}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}(y_{ij} - w_j^T z_i)^2$$



- But don't include missing entries in loss:

$$\underset{W,Z}{\text{argmin}}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d} I[y_{ij} \neq ?](y_{ij} - w_j^T z_i)^2$$



Collaborative Filtering

- Can predict missing rating for user i and item j:

$$\hat{\mathbf{y}}_{ij} = \mathbf{w}_j^T \mathbf{z}_i$$

Collaborative Filtering

- Can predict missing rating for user i and item j:

$$\hat{\mathbf{y}}_{ij} = \mathbf{w}_j^T \mathbf{z}_i$$

- Can add user bias $\mathbf{b}_i$ and item bias $\mathbf{b}_j$.

$$\hat{\mathbf{y}}_{ij} = \mathbf{w}_j^T \mathbf{z}_i + \mathbf{b}_i + \mathbf{b}_j$$

  - High $\mathbf{b}_i$ means user i rates higher than average.
  - High $\mathbf{b}_j$ means j is rated higher than average.

Exercise

$$f(\mathbf{b}_u, \mathbf{b}_m, \mathbf{w}_m, \mathbf{z}_u) = \frac{1}{2}(\mathbf{y}_{um} - (\mathbf{w}_m^T \mathbf{z}_u + \mathbf{b}_u + \mathbf{b}_m))^2$$

Using the notation $r_{um} = (y_{um} - (b_u + b_m + w_m^T z_u))$, derive the partial derivative of this expression with respect to (i) b_u, (ii) b_m, (iii) $(w_m)_i$ for a particular element i of w_m, and (iv) $(z_u)_i$ for a particular element i of z_u.

$$\frac{\partial f}{\partial \mathbf{b}_u} = ?$$

$$\frac{\partial f}{\partial \mathbf{b}_m} = ?$$

$$\frac{\partial f}{\partial (\mathbf{w}_m)_i} = ?$$

$$\frac{\partial f}{\partial (\mathbf{z}_u)_i} = ?$$

Exercise: Solution

$$f(\mathbf{b}_u, \mathbf{b}_m, \mathbf{w}_m, \mathbf{z}_u) = \frac{1}{2}(\mathbf{y}_{um} - (\mathbf{w}_m^T \mathbf{z}_u + \mathbf{b}_u + \mathbf{b}_m))^2$$

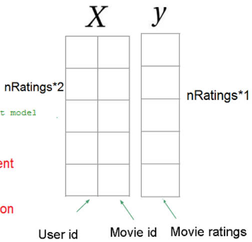
Using the notation $r_{um} = (y_{um} - (b_u + b_m + w_m^T z_u))$, derive the partial derivative of this expression with respect to (i) b_u, (ii) b_m, (iii) $(w_m)_i$ for a particular element i of w_m, and (iv) $(z_u)_i$ for a particular element i of z_u.

$$\frac{\partial f}{\partial \mathbf{b}_u} = -r_{um}$$
$$\frac{\partial f}{\partial \mathbf{b}_m} = -r_{um}$$
$$\frac{\partial f}{\partial (\mathbf{w}_m)_i} = -r_{um}\mathbf{z}_{ui}$$
$$\frac{\partial f}{\partial (\mathbf{w}_m)_i} = -r_{um}\mathbf{w}_{mi}$$

Exercise

- Using the previous question,complete the following function with gradient descent.

```
function [model] = recommendSVD(X,y,k)
n = max(X(:,1)); d = max(X(:,2)); nRatings = size(X,1);
% Initialize parameters
% - for the biases, we'll use the user/item averages  % - for the latent factors, random
subModel = recommendUserItemMean(X,y);
bu = subModel.bu/2; bm = subModel.bm/2; W = .00001*randn(k,d); Z = .00001*randn(n,k);
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter
    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings
        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        Add gradient of this prediction to overall gradient
                 ?
    end
    Take a small step in the negative gradient direction
                      ?
    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
end
model.bu = bu; model.bm = bm; model.W = W; model.Z = Z; model.predict = @predict;
end
```



Exercise: Solution

```
function [model] = recommendSVD(X,y,k)
n = max(X(:,1)); d = max(X(:,2)); nRatings = size(X,1);
% Initialize parameters
% - for the biases, we'll use the user/item averages  % - for the latent factors, random
subModel = recommendUserItemMean(X,y);
bu = subModel.bu/2; bm = subModel.bm/2; W = .00001*randn(k,d); Z = .00001*randn(n,k);
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter
    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings
        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        % Add gradient of this prediction to overall gradient
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end
    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;
    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
end
model.bu = bu; model.bm = bm; model.W = W; model.Z = Z; model.predict = @predict;
end
```