CPSC 540: Assignment 3

Yiwei Hou(84435156)
Xiaomeng Ju(86475150)
Tingting Yu(74439118)

1 Discrete and Gaussian Variables

1.1 MLE for General Discrete Distribution

1. Given n training samples, the joint probability could be written as

$$P(x^1, x^2, \cdots, x^n) = \prod_{c_1,c_2=1}^{k} \theta_{c_1,c_2}^{\sum_{i=1}^n \mathbf{1}(x_1^i=c_1, x_2^i=c_2)},$$

where $\mathbf{1}(x_1^i = c_1, x_2^i = c_2)$ is an indicator and equals one when $x_1^i = c_1, x_2^i = c_2$ and zero otherwise, for $i = 1, 2, \cdots, n$. The log-likelihood is

$$l = \sum_{c_1,c_2=1}^{k} \sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2)\log(\theta_{c_1,c_2}),$$

with the constraints $\theta_{c_1,c_2} \geq 0$ for all $c_1, c_2 \in \{1, 2, \cdots, k\}$ and $\sum_{c_1=1,c_2=1}^{k} \theta_{c_1,c_2} = 1$. By Lagrange multiplier, to maximize the log-likelihood function, we can equivalently maximize the following equation:

$$L(\boldsymbol{\theta}, \lambda) = \sum_{c_1,c_2=1}^{k} \sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2)\log(\theta_{c_1,c_2}) - \lambda(\sum_{c_1=1,c_2=1}^{k} \theta_{c_1,c_2} - 1).$$

Taking derivatives with respect to θ_{c_1,c_2} and λ and setting them to zeros, we have

$$\frac{\partial L}{\partial \theta_{c_1,c_2}} = \sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2)\frac{1}{\theta_{c_1,c_2}} - \lambda = 0$$

$$\frac{\partial L}{\partial \lambda} = \sum_{c_1=1,c_2=1}^{k} \theta_{c_1,c_2} - 1 = 0$$

The MLE for k^2 elements of $\boldsymbol{\theta}$ is $\hat{\lambda}_{c_1,c_2} = \frac{\sum_{i=1}^n \mathbf{1}(x_1^i=c_1, x_2^i=c_2)}{n}$.

2. With the re-parameterization, the joint probability is

$$P(x^1, x^2, \cdots, x^n) = \prod_{c_1,c_2 \neq k} \theta_{c_1,c_2}^{\sum_{i=1}^n \mathbf{1}(x_1^i=c_1, x_2^i=c_2)} \cdot (1 - \sum_{c_1,c_2 \neq k} \theta_{c_1,c_2})^{\sum_{i=1}^n \mathbf{1}(x_1^i=x_2^i=k)}.$$

The log-likelihood is

$$l = \sum_{c_1,c_2 \neq k} \sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2)\log(\theta_{c_1,c_2}) + \sum_{i=1}^{n} \mathbf{1}(x_1^i = x_2^i = k)\log(1 - \sum_{c_1,c_2 \neq k} \theta_{c_1,c_2}).$$

Taking derivative with respect to $\theta_{c_1,c_2}, c_1, c_2 \neq k$, we have

$$\frac{\partial l}{\partial \theta_{c_1,c_2}} = \sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2, c_1, c_2 \neq k)\frac{1}{\theta_{c_1,c_2}} - \sum_{i=1}^{n} \mathbf{1}(x_1^i = x_2^i = k)\frac{1}{1 - \sum_{c_1,c_2 \neq k} \theta_{c_1,c_2}}.$$

Solving for θ_{c_1,c_2}, we have the MLE for the k^2-1 elements of the $\boldsymbol{\theta}$ is $\hat{\theta}_{c_1,c_2} = \frac{\sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2)}{n}, c_1, c_2 \neq k$.

3. With the specified prior, the posterior probability can be written as

$$P(x^1, x^2, \cdots, x^n) = \prod_{c_1,c_2=1}^{k} \theta_{c_1,c_2}^{\sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2) + \alpha_{c_1} + \alpha_{c_2} - 2}.$$

Following the same procedure in (1), we have the MAP under the prior as $\hat{\theta}_{c_1,c_2} = \frac{\sum_{i=1}^{n} \mathbf{1}(x_1^i = c_1, x_2^i = c_2) + \alpha_{c_1} + \alpha_{c_2} - 2}{n + \sum_{c_1=1}^{k} \sum_{c_2=1}^{k} (\alpha_{c_1} + \alpha_{c_2} - 2)}$.

1.2 Generative Classifiers with Gaussian Assumption

1. We define $n_c = \sum_{i=1}^{n} \mathbf{1}(y^i = c), c = 1, 2, \cdots, k$. Under the assumption of common diagonal covariance matrices, we further define s_j^2 as the j-th diagonal entry of $\Sigma_c = D, j = 1, 2, \cdots, d$. The determinant of the diagonal matrix is $|D| = \prod_{j=1}^{d} s_j^2$ and D^{-1} is a diagonal matrix with $\frac{1}{s_j^2}$ along the diagonal. Taking derivative with respect to μ_c and s_j^2, we have

$$\frac{\partial \log P}{\partial \mu_c} = \sum_{i=1}^{n} D^{-1}(x^i - \mu_{y_i})1_{(y_i = c)} = 0$$

$$\frac{\partial \log P}{\partial s_j^2} = \frac{-n}{2s_j^2} + \sum_{i=1}^{n} (x_j^i - \mu_{y_i,j})^2 \frac{1}{2s_j^4} = 0$$

Solving for μ_c and s_j^2, we have the MLE for GDA model is $\hat{\mu}_c = \frac{\sum_{i=1}^{n} x^i \mathbf{1}_{(y_i = c)}}{n_c}$, $\hat{s_j^2} = \frac{1}{n} \sum_{i=1}^{n} (x_j^i - \hat{\mu}_{y^i,j})^2$, where $\hat{\mu}_{y^i} = \hat{\mu}_c$ with $y^i = c$.

2. When $\Sigma_c = \sigma_c^2 I$, we re-write the joint probability as follows:

$$P \propto \prod_{i=1}^{n} |\Sigma_{y^i}|^{-0.5} \exp\{\sum_{i=1}^{n} \frac{-1}{2}(x^i - \mu_{y^i})^T \Sigma_{y^i}^{-1} (x^i - \mu_{y^i})\}$$

The log-likelihood is thus

$$\log(P) \propto -\frac{1}{2}\sum_{i=1}^{n}(x^i-\mu_{y_i})^T\sigma_{y_i}^{-2}(x^i-\mu_{y_i}) - \frac{1}{2}\sum_{i=1}^{n}\log|\sigma_{y_i}^2 I|.$$

Taking derivative with respect to μ_c and σ_c where $c = 1, 2, \cdots, k$, we have

$$\frac{\partial \log P}{\partial \mu_c} = \sum_{i=1}^{n}\sigma_{y_i}^{-2}(x^i-\mu_{y_i})1_{(y_i=c)} = 0$$
$$\frac{\partial \log P}{\partial \sigma_c} = \sum_{i=1}^{n}\sigma_{y_i}^{-3}(x^i-\mu_{y_i})^T(x^i-\mu_{y_i})1_{(y_i=c)} - \frac{dn_c}{\sigma_c} = 0$$

Solving the two equations above, we have

$$\hat{\mu}_c = \frac{\sum_{i=1}^{n} x^i 1_{(y_i=c)}}{n_c}$$
$$\sigma_c^2 = \frac{1}{dn_c}\sum_{i=1}^{n}(x^i-\mu_{y_i})^T(x^i-\mu_{y_i})1_{(y_i=c)}$$

3. For the case of individual full covariance matrices, we have

$$\hat{\mu}_c = \frac{\sum_{i=1}^{n} x^i 1_{(y_i=c)}}{n_c}$$
$$\Sigma_c = \frac{1}{n_c}\sum_{i=1}^{n}(x^i-\mu_{y_i})(x^i-\mu_{y_i})^T 1_{(y_i=c)}$$

4. The function is given below. The test set accuracy is 0.63.

```
function [ model ] = generativeGaussian( Xtrain, Ytrain, k )
  n = size(Xtrain,1);
  classes = unique(Ytrain);
  mu = cell(k, 1);
  Sigma = cell(k,1);

  for i=1:k
     idx = (Ytrain==classes(i) );
     subX = Xtrain(idx,:);
     subY = Ytrain(idx);
     ni = size(subX,1);
     mu{i} = mean(subX);
     Z = subX-mu{i};
     Sigma{i} = Z'*Z/ni;
  end

  model.Xtrain = Xtrain;
```

```
  model.Ytrain = Ytrain;
  model.K = k;
  model.mu = mu;
  model.Sigma = Sigma;
  model.predict = @(model, Xtest) predict(model, Xtest);
end

function yhat = predict(model, Xtest)
  nTest = size(Xtest, 1);
  yhat = zeros(nTest, 1);
  for i=1:nTest
      p=zeros(model.K, 1);
      for j=1:model.K
        p(j) = mvnpdf(Xtest(i,:), model.mu{j}, model.Sigma{j});
      end
      [v,idx] = max(p);
      yhat(i) = idx;
  end
end
```

5. The function is given below. The test set accuracy is 0.79.

```
function [ model ] = generativeStudent( Xtrain, Ytrain, k )
  n = size(Xtrain,1);
  classes = unique(Ytrain);
  models = cell(k, 1);

  for i=1:k
     idx = (Ytrain==classes(i) );
     subX = Xtrain(idx,:);
     models{i} = multivariateT(subX);
  end

  model.Xtrain = Xtrain;
  model.Ytrain = Ytrain;
  model.K = k;
  model.models=models;
  model.predict = @(model, Xtest) predict(model, Xtest);
end

function yhat = predict(model, Xtest)
  nTest = size(Xtest, 1);
  yhat = zeros(nTest, 1);

  for i=1:nTest
      p=zeros(model.K, 1);
```

```
      for j=1:model.K
        p(j) = model.models{j}.pdf(model.models{j}, Xtest(i,:));
      end
      [v,idx] = max(p);
      yhat(i) = idx;
    end
  end
```

1.3 Self-Conjugacy for the Mean Parameter

Let $\lambda = 1/\sigma^2, \lambda_0 = 1/\gamma^2$. We have

$$
\begin{aligned}
p(\mu|x,\alpha,\sigma^2,\gamma^2) &\propto p(x|\mu,\sigma^2)p(\mu|\alpha,\gamma^2) \\
&\propto \exp(-0.5\lambda(x-\mu)^2)\exp(-0.5\lambda_0(\mu-\alpha)^2) \\
&= \exp(-0.5\lambda(x-\mu)^2 - 0.5\lambda_0(\mu-\alpha)^2) \\
&= \exp\Bigg\{-0.5(\lambda+\lambda_0)\Bigg[\Big(\mu - \frac{\lambda x + \lambda_0\alpha}{\lambda+\lambda_0}\Big)^2 - \Big(\frac{\lambda x + \lambda_0\alpha}{\lambda+\lambda_0}\Big)^2 + \frac{\lambda x^2 + \lambda_0\alpha^2}{\lambda+\lambda_0}\Bigg]\Bigg\} \\
&\propto \exp\Bigg\{-0.5(\lambda+\lambda_0)\Big(\mu - \frac{\lambda x + \lambda_0\alpha}{\lambda+\lambda_0}\Big)^2\Bigg\}
\end{aligned}
$$

Therefore,

$$\mu|x,\alpha,\sigma^2,\gamma^2 \sim N\Big(\frac{\lambda x + \lambda_0\alpha}{\lambda+\lambda_0}, \frac{1}{\lambda+\lambda_0}\Big)$$

2 Mixture Models and Expectation Maximization

2.1 Semi-Supervised Gaussian Discriminant Analysis

1. Let $(x^i, y^i), i = 1, \cdots, n$, be the n labeled examples and $(\bar{x}^i), i = 1, \cdots, t$, be the t unlabeled examples. According to the notes we have

$$Q(\Theta|\Theta^k) = \sum_{i=1}^{n} \log p(y^i, x^i|\Theta) + \sum_{i=1}^{t} \sum_{\tilde{y}^i \in \{1,..,k\}} r^i_{\tilde{y}^i} \log p(\tilde{y}^i, \tilde{x}^i|\Theta), \tag{1}$$

   where

$$r^i_{\tilde{y}^i} = p(\tilde{y}^i = \tilde{y}^i|\tilde{x}^i, \Theta^k) = \frac{p(\tilde{y}^i, \tilde{x}^i|\Theta^k)}{\sum_{y\in\{1,\ldots,k\}} p(y, \tilde{x}^i|\Theta^k)}.$$

   To find the update at time $k+1$,

$$\Theta^{k+1} = \text{argmax}_\Theta Q(\Theta|\Theta^k).$$

   In the notes,

$$\pi_c^{k+1} = \frac{n_c + \sum_{i=1}^{t} r_c^i}{n+t}.$$

Next we derive μ_c^{k+1} and Σ_c^{k+1} by solving $\partial Q(\Theta|\Theta^k)/\partial\mu_c = 0$ and $\partial Q(\Theta|\Theta^k)/\partial\Sigma_c = 0$.

Following from Equation 1

$$
\begin{aligned}
Q(\Theta|\Theta^k) &= \sum_{i=1}^{n} \log p(x^i|y^i,\Theta) + \sum_{i=1}^{n} \log p(y^i|\Theta) + \sum_{i=1}^{t} \sum_{\tilde{y}\in\{1,..,k\}} r^i_{\tilde{y}^i} \log p(\tilde{x}^i|\tilde{y}^i,\Theta) + \\
&\quad \sum_{i=1}^{t} \sum_{\tilde{y}\in\{1,..,k\}} r^i_{\tilde{y}^i} \log p(\tilde{y}^i|\Theta) \\
\frac{\partial Q(\Theta|\Theta^k)}{\partial \mu_c} &= \sum_{i=1}^{n} \frac{\partial \log p(x^i|y^i,\Theta)}{\partial \mu_c} + \sum_{i=1}^{t} \sum_{\tilde{y}\in\{1,..,k\}} r^i_{\tilde{y}^i} \frac{\partial \log p(\tilde{x}^i|\tilde{y}^i,\Theta)}{\partial \mu_c} \\
&= \sum_{i=1}^{n} I(y^i = c)\Sigma_c^{-1}(\mu_c - x^i) + \sum_{i=1}^{t} r^i_c \Sigma_c^{-1}(\mu_c - \tilde{x}^i) \\
\mu_c^{k+1} &= \{\mu_c : \frac{\partial \log p(\tilde{x}^i|\tilde{y}^i,\Theta)}{\partial \mu_c} = 0\} \\
&= \frac{\sum_{i=1}^{n} I(y^i = c)x^i + \sum_{i=1}^{t} r^i_c \tilde{x}^i}{n_c + \sum_{i=1}^{t} r^i_c} \\
\frac{\partial Q(\Theta|\Theta^k)}{\partial \Sigma_c} &= \sum_{i=1}^{n} \frac{\partial \log p(x^i|y^i,\Theta)}{\partial \Sigma_c} + \sum_{i=1}^{t} \sum_{\tilde{y}\in\{1,..,k\}} r^i_{\tilde{y}^i} \frac{\partial \log p(\tilde{x}^i|\tilde{y}^i,\Theta)}{\partial \Sigma_c} \\
&= \sum_{i=1}^{n} I(y^i = c)\left[\frac{1}{2}\Sigma_c^{-1}(x^i - \mu_c)(x^i - \mu_c)^T\Sigma_c^{-1} - \frac{1}{2}\Sigma_c^{-1}\right] \\
&\quad + \sum_{i=1}^{t} r^i_c \left[\frac{1}{2}\Sigma_c^{-1}(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T\Sigma_c^{-1} - \frac{1}{2}\Sigma_c^{-1}\right] \\
\Sigma_c^{k+1} &= \{\Sigma_c : \frac{\partial \log p(\tilde{x}^i|\tilde{y}^i,\Theta)}{\partial \Sigma_c} = 0\} \\
&= \{\Sigma_c : \Sigma_c^{-1}\left\{\sum_{i=1}^{n} I(y^i = c)\left[(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T - \Sigma_c\right] + \sum_{i=1}^{t} r^i_c\left[(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T - \Sigma_c\right]\right\} \\
&= \frac{\sum_{i=1}^{n} I(y^i = c)(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T + \sum_{i=1}^{t} r^i_c(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T}{n_c + \sum_{i=1}^{t} r^i_c}
\end{aligned}
$$

2. With the threshold `tol` of difference of Θ^k and Θ^{k-1} set to be 0.1, the algorithm converges in 40 iterations with the test error being 1-75% = 25%.

```
function [model] = generativeGaussianSSL( Xtrain, Ytrain, Xtilde)

model.Xtrain = Xtrain;
model.Ytrain = Ytrain;
model.Xtilde = Xtilde;
model.predict = @(model, Xtest) predict(model, Xtest);
end

```

```
function yhat = predict(model, Xtest)
Ytrain = model.Ytrain; Xtrain = model.Xtrain;nTest = size(Xtest,1);
Xtilde = model.Xtilde; yhat = zeros(nTest, 1);
tol= 0.1; nMaxIter = 200;diff = Inf;
k = max(Ytrain); %number of classes notations consistent with the notes
t = size(Xtilde,1); [n,p] = size(Xtrain); % p is the number of variables
%initialize nc
nc = zeros(k,1);
for i = 1:k
    nc(i) = sum(Ytrain==i);
end
%initialize r
r =  repmat(1/k,t,k); %initialize r
%initialize theta_c
theta_c = repmat(1/k,1,k);

%initialize mu and sigma
for i = 1:k
    indice = find(Ytrain==i);
    mu_est(:,:,i) = zeros(1,p);
    sigma_est(:,:,i) = eye(p);
end

for i = 1: nMaxIter
    mu_pre = mu_est;
    sigma_pre = sigma_est;

    for j = 1:k
        theta_c(j) = (nc(j) + sum(r(:,j)))/(n+t);
        indice = find(Ytrain==j);
        mu_est(:,:,j) = (sum(Xtrain(indice,:),1) +sum(diag(r(:,j))*Xtilde,1))/(length(indice)+sum(r(:,j)));

        Xtmp= Xtrain(indice,:)-ones(length(indice),1)*mu_est(:,:,j);
        Xtmp_miss = Xtilde-ones(t,1)*mu_est(:,:,j);
        sigma_est(:,:,j) = (Xtmp'*Xtmp + Xtmp_miss'*diag(r(:,j))*Xtmp_miss)/(length(indice)+sum(r(:,j)));
        r(:,j) = calculate_pdf(Xtilde,j,mu_est,sigma_est,theta_c); % Have not been normalized yet
    end
    r = r./sum(r,2);
    diff = sum(sum(squeeze(mu_pre - mu_est).^2))+ sum(sum(squeeze(sum(squeeze(sigma_pre - sigma_est).^2))))
    if diff<tol
        fprintf('Converges')
        i
        break;
    end
end


for i=1:nTest
    tmp = zeros(0, k,1);
    for j = 1:k
        tmp(j) = mvnpdf(Xtest(i,:),mu_est(:,:,j),sigma_est(:,:,j))*theta_c(j)
```

```
              ;
    end
    yhat(i)=find(tmp == max(tmp));
end

end
```

3. The hard-EM algorithm converges faster than EM algorithm. With the same `tol`, the algorithm converges in 27 iterations with the same test error (25%)

2.2 Mixture of Bernoullis

1. The average NLL with $\alpha = 1$ is 205.8973.

```
function [ model ] = densityBernoulli(X,alpha)

[n,d] = size(X);

%theta = mean(X);
theta = (sum(X,1)+ alpha)/(size(X,1) + size(X,2)*alpha);
model.theta = theta;
model.predict = @predict;
model.sample = @sample;
end

function nlls = predict(model, Xhat)
[t,d] = size(Xhat);
theta = model.theta;

nlls = -sum(prod0(Xhat,repmat(log(theta),[t 1])) + prod0(1-Xhat,repmat(log(1-theta),[t 1])),2);
end

function samples = sample(model,t)
theta = model.theta;
d = length(theta);

samples = zeros(t,d);
for i = 1:t
    samples(i,:) = rand(1,d) < theta;
end
end
```

2. The average NLL on testing data is 166.7085. The algorithm converges in 30 iterations.

```
%% new code (mixture distribution)
model = mixtureBernoulli(Xtrain,1);
nlls = model.predict(model,Xtest);
averageNLL = sum(nlls)/size(Xtest,1)

samples = model.sample(model,4);
figure(1);

```

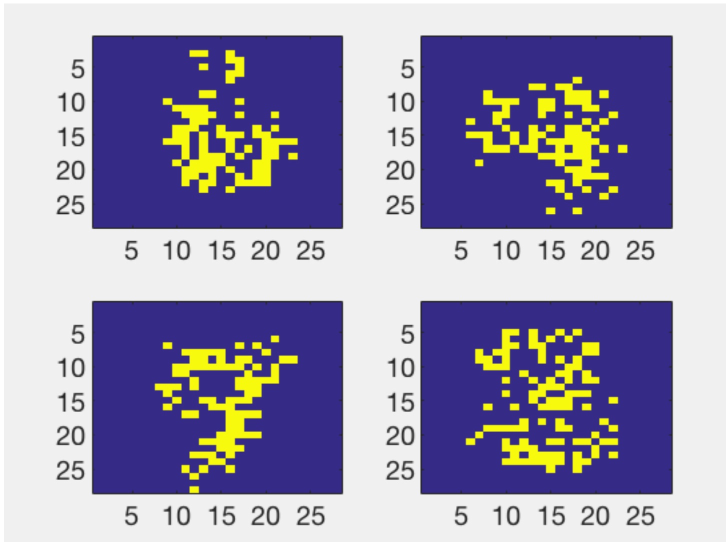

Figure 1: Four figures generated by the model

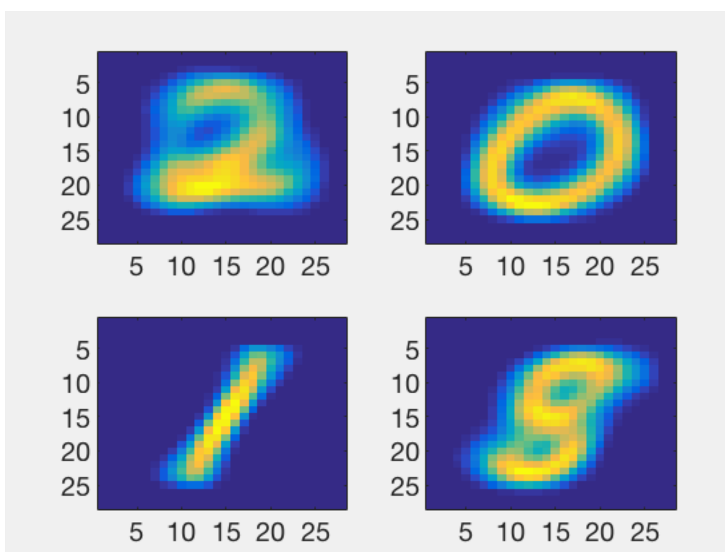

Figure 2: Four cluster figures from the estimated parameters

```matlab
%four samples generated by the model
for i = 1:4
    subplot(2,2,i);
    imagesc(reshape(samples(i,:),[28 28])');
end

figure(1);
% four cluster samples
for i = 1:4
    subplot(2,2,i);
    %class_k = find(mnrnd(1,pi_est')==1);
    class_k = i;
    imagesc(reshape(model.theta(:,class_k),[28 28])');
end

function [ model ] = mixtureBernoulli(X,alpha)

[n,d] = size(X);
theta = (sum(X,1)+ alpha)/(size(X,1) + size(X,2)*alpha);
Xtrain = X;
[n,d] = size(Xtrain);
k = 10;
theta_est = 0.5*ones(d,k); pi_est = repmat(1/k,1,k);
tol = 0.1; nMaxIter = 100; diff = Inf;
z = repmat(1/k,n,k);
for j = 1:k
    z(:,j) = rand(n,1);
end
z = z./sum(z,2);

for i = 1: nMaxIter
    theta_pre = theta_est; pi_pre =pi_est;
    nm = sum(z,1); %number of images assigned to each cluster

    for j = 1:k
       theta_est(:,j) = (z(:,j)'*Xtrain+alpha)./(nm(j)+alpha*d);
    end

    for j = 1:k
        nlls = -sum(prod0(Xtrain,repmat(log(theta_est(:,j)'),[n 1])) + prod0
            (1-Xtrain,repmat(log(1-theta_est(:,j)'),[n 1])),2);
        z(:,j)= exp(-nlls).*pi_est(j);
    end

    z = z./sum(z,2);
    pi_est = sum(z,1)/n;

    diff = sum(sum(theta_pre - theta_est).^2)+ sum((pi_pre - pi_est).^2);

    [i,diff]
    if diff<tol
       fprintf('Converges')
       i
       break;
    end
```

```
    pi_est
end

 model.z = z;
 model.theta = theta_est;
 model.pi = pi_est;
 model.predict = @(model, Xtest) predict(model, Xtest);
 model.sample = @sample;
end

function nlls = predict(model, Xhat)
[t,d] = size(Xhat);
theta = model.theta;
nlls = 0;
pi_est = model.pi;
k = size(theta,2);
pi_est(10) = 1- sum(pi_est(1:9))
for j = 1:k
   nll_j = -sum(prod0(Xhat,repmat(log(theta(:,j)'),[t 1])) + prod0(1-Xhat,
       repmat(log(1-theta(:,j)'),[t 1])),2);
   nlls = nlls+ nll_j-log(pi_est(j));
end

end


function samples = sample(model,t)
theta = model.theta;
d = size(theta,1);

pi_est = model.pi;

samples = zeros(t,d);
pi_est(10) = 1- sum(pi_est(1:9))
for i = 1:t
    class_k = find(mnrnd(1,pi_est')==1)
    samples(i,:) = rand(1,d) < theta(:,class_k)';
end

end
```

3 Project Proposal

Large-scale Online Matrix Completion

Problem Description

The goal of this project is to extend the practice of online matrix completion algorithms to large-scale recommender systems. Given a set of user ratings on items, we wish to predict what individual users will rate the items they have not yet rated. Essentially, the problem becomes a matrix completion problem which involves predicting missing entries in data matrices using the assumption that the fully observed matrix is low-rank. Our focus is on large-scale problems, where the matrices have many rows and columns. To the best of our knowledge, there exist matrix completion algorithms with online data and large-scale data individually. We aim to study these algorithms and hopefully to come up with one that deals with online and large-scale problems simultaneously. The performance of the algorithms will be evaluated with the MovieLens dataset [1].

Methodology

The matrix completion has two general frameworks:
(1) Non-robust version

$$\min \frac{1}{2}\|X - A\|_{ob}^2 \quad \text{subj } \|A\|_* \leq k,$$

where $\|\cdot\|_{ob}$ is the Euclidean norm only on the observed entries of X and $\|A\|_*$ is the nuclear norm defined as the L_1 norm of A's singular values. Based on preliminary literature review, we found that conditional gradient methods has been applied to solve (1). We consider an online Frank-Wolfe (OFW) algorithm that is a projection free algorithm described in [2]. We also hope to include a comparison of OFW to other gradient-based online convex optimization methods.

(2) Robust version

$$\min \frac{1}{2}\|X - A\|_1 \quad \text{subj } \|A\|_* \leq k.$$

The robust principle component analysis (RPCA) has been applied to solve (2). The RPCA takes a data matrix and decompose it into the sum of a low-rank matrix and a sparse matrix [3][4]. We hope to find a variant of RPCA that can be solved in a recursive fashion for online data.

Real Data

We plan to validate the algorithms on the MovieLens 10K dataset [1]. This standard dataset consists of 100,000 ratings (1-5) from 943 users on 1682 movies. The dataset has been cleaned up and each user has rated at least 20 movies. Each rating record has a tab that contains a user id, item id, rating, and a timestamp that can be used to specify the sequence data in our online algorithm.

Possible Extensions

If time permitted, we consider to explore several possible extensions to the problem.
(1) Partial feedback problem that assumes that some users will provide feedback to the recommended items. We may try to explore how to adjust the system according to the feedback.
(2) Bayesian optimization for selecting hyperparameters in the optimization problems.

References

[1] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[2] Elad Hazan and Satyen Kale. Projection-free online learning. *arXiv preprint arXiv:1206.4657*, 2012.

[3] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[4] John Wright, Arvind Ganesh, Shankar Rao, Yigang Peng, and Yi Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In *Advances in neural information processing systems*, pages 2080–2088, 2009.