EECS 545 - Homework 5

Ding Ding

April 4, 2016

EECS 545 HW5.

Q1

(a). $D_{KL}(p\|q) = D_{KL}(p(x,y)\|q(x,y)) = \sum_x\sum_y p(x,y)\log\frac{p(x,y)}{q_1(x)q_2(y)} = \sum_x\sum_y p(x,y)\log p(x,y) - \sum_x\sum_y p(x,y)\log q_1(x)q_2(y)$

To minimize $D_{KL}(p\|q)$, we want to minimize $-\sum_x\sum_y(p(x,y)\log q_1(x) + p(x,y)\log q_2(y))$

$-\sum_x\sum_y p(x,y)\log q_1(x) = -\sum_x\left[\left(\sum_y p(x,y)\right)\log q_1(x)\right] = -\sum_x p(x)\log q_1(x) = H(p,q)$

since $H(p,q)\geq H(p) \Rightarrow \underset{q_1(x)}{\arg\min} -\sum_x\sum_y p(x,y)\log q_1(x) = p(x)$

Similarly $-\sum_x\sum_y p(x,y)\log q_2(y) = -\sum_y\left[\left(\sum_x p(x,y)\right)\log q_2(y)\right] = -\sum_y p(y)\log q_2(y) = H(p,q)$

$\Rightarrow \underset{q_2(y)}{\arg\min} -\sum_x\sum_y p(x,y)\log q_2(y) = p(y)$.

The optimal approximation is a product of marginals.

(b). Still considering the factored approximation $q(x,y) = q_1(x)q_2(y)$

Since for $p(x_i,y_j)=0$, to avoid ∞, we have to let $q_1(x_i)q_2(y_j)=0$ since $\lim_{x\to 0} x\log x = 0$

$\therefore$ with the Joint probability table for $p(x,y)$, there is only three condition that can let $q_1(x_i)q_2(y_j)\log\frac{q_1(x_i)q_2(y_j)}{p(x_i,y_j)}$ all $\neq\infty$.

<1> only $q_1(x_3)$, $q_2(y_3)\neq 0$

<2> only $q_1(x_4)$, $q_2(y_4)\neq 0$

<3> $q_1(x_1), q_1(x_2), q_2(y_1), q_2(y_2)=0$

First, let us consider the general situation: to minimize $\underset{q_1(x),q_2(y)}{} D_{KL}(q\|p) = \sum_{i,j} q_1(x_i)q_2(y_j)\log\frac{q_1(x_i)q_2(y_j)}{p(x_i,y_j)}$

s.t $\sum_i q_1(x_i)=1$ $\quad \sum_j q_2(y_j)=1$

$\therefore \mathcal{L} = \sum_{i,j} q_1(x_i)q_2(y_j)\log\frac{q_1(x_i)q_2(y_j)}{p(x_i,y_j)} + \lambda(\sum_i q_1(x_i)-1) + \beta(\sum_j q_2(y_j)-1)$

$\forall i \quad \frac{\partial\mathcal{L}}{\partial q_1(x_i)} = \log q_1(x_i) + \sum_j q_2(y_j)\log q_2(y_j) - \sum_j q_2(y_j)\log p(x_i,y_j) + 1 + \lambda = 0$

$\forall j \quad \frac{\partial\mathcal{L}}{\partial q_2(y_j)} = \log q_2(y_j) + \sum_i q_1(x_i)\log q_1(x_i) - \sum_i q_1(x_i)\log p(x_i,y_j) + 1 + \beta = 0$

$\frac{\partial\mathcal{L}}{\partial\lambda} = \sum_i q_1(x_i) - 1 = 0$

$\frac{\partial\mathcal{L}}{\partial\beta} = \sum_j q_2(y_j) - 1 = 0$

For $\langle 1 \rangle$, $q_1(x_3) = q_2(y_3) = 1$.

we have $D_{KL}(q \| p) = 1 \times \log \frac{1}{p(x_3, y_3)} = \log 4$.

For $\langle 2 \rangle$ $q_1(x_4) = q_2(y_4) = 1$

we have $D_{KL}(q \| p) = 1 \times \log \frac{1}{p(x_4, y_4)} = \log 4$

For $\langle 3 \rangle$,

$$\left\{\begin{array}{l} \log q_1(x_1) + H(q_2) - \log 1/8 + 1 + \eta = 0 \\ \log q_1(x_2) + H(q_2) - \log 1/8 + 1 + \eta = 0 \\ \log q_2(y_1) + H(q_1) - \log 1/8 + 1 + \beta = 0 \\ \log q_2(y_2) + H(q_1) - \log 1/8 + 1 + \beta = 0 \\ q_2(y_1) + q_2(y_2) = 1 \\ q_1(x_1) + q_1(x_2) = 1 \end{array}\right. \Rightarrow \quad q_1(x_1) = q_1(x_2) = q_2(y_1) = q_2(y_2) = 0.5$$

$$D_{KL}(q \| p) = 1/4 \times \log \frac{1/4}{1/8} \times 4 = \log 2$$

(c) If we set $q(x, y) = p(x)p(y)$

due from the table 1, the marginals are: $p(x_1) = p(x_2) = p(x_3) = p(x_4) = 1/4$

$p(y_1) = p(y_2) = p(y_3) = p(y_4) = 1/4$.

$$D_{KL}(q \| p) = \sum_{i,j} p(x_i)p(y_j) \log \frac{p(x_i)p(y_j)}{p(x_i, y_j)}$$

$D_{KL}(q \| p) = \infty$. Since $p(x_i)p(y_j) \neq 0$ when $p(x_i, y_j) = 0$.

still (b)

Consider marginal condition: If $q_1(x_1)$ or $q_1(x_2) = 0 \Rightarrow \left\{\begin{array}{l} \text{if both } q_2(y_1), q_2(y_2) \neq 0 \quad D_{KL}(q \| p) = \log 4 \\ \text{if } q_2(y_1) \text{ or } q_2(y_2) = 0. \quad D_{KL}(q \| p) = \log 8. \end{array}\right.$

Similarly, if $q_2(y_1)$ or $q_2(y_2) = 0 \Rightarrow \left\{\begin{array}{l} \text{if both } q_1(x_1), q_1(x_2) \neq 0, \quad D_{KL}(q \| p) = \log 4 \\ \text{if } q_1(x_1) \text{ or } q_1(x_2) = 0 \quad D_{KL}(q \| p) = \log 8 \end{array}\right.$

$\therefore$ $\min D_{KL}(q \| p) = \log 2$ in $\langle 3 \rangle$

Q2 Gibbs Sampling from a 2D Gaussian

(a). $P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)}$; $P(x_2|x_1) = \frac{P(x_2, x_1)}{P(x_1)}$

$$P(x_1,x_2) = \frac{1}{(\sqrt{2\pi})^2 \sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{1}{2(1-\rho^2)}\left(\frac{(x_1-\mu_1)^2}{\sigma_1^2} - 2\rho\frac{x_1-\mu_1}{\sigma_1}\cdot\frac{x_2-\mu_2}{\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2}\right)\right]$$

$$= \frac{1}{\sqrt{3}\pi} \exp\left[-\frac{2}{3}\left[(x_1-1)^2 - (x_1-1)(x_2-1)) + (x_2-1)^2\right]\right]$$

$$P(x_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(x_2-1)^2\right) \qquad P(x_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(x_1-1)^2\right)$$

$$P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)} = \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}\left[(x_1-1)^2-(x_1-1)(x_2-1) + (x_2-1)^2\right] + \frac{1}{2}(x_2-1)^2\right)$$

$$= \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}(x_1-1)^2 + \frac{2}{3}(x_1-1)(x_2-1) - \frac{1}{6}(x_2-1)^2\right)$$

$$= \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}x_1^2 + \frac{4}{3}x_1 - \frac{2}{3} + \frac{2}{3}x_1x_2 - \frac{2}{3}x_2 - \frac{2}{3}x_1 + \frac{2}{3} - \frac{1}{6}x_2^2 - \frac{1}{6} + \frac{1}{3}x_2\right)$$

$$= \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}x_1^2 - \frac{1}{6}x_2^2 + \frac{2}{3}x_1 - \frac{1}{3}x_2 + \frac{2}{3}x_1x_2 - \frac{1}{6}\right)$$

$$= \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\frac{1}{6}\left(-4x_1^2 - x_2^2 + 4x_1 - 2x_2 + 4x_1x_2 - 1\right)$$

$$= \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp-\frac{1}{6}(x_2-2x_1+1)^2 = \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}\left(\frac{x_2}{2} - x_1 + \frac{1}{2}\right)^2\right) = \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}\left(x_1 - \frac{x_2}{2} - \frac{1}{2}\right)^2\right)$$

similarly

$$P(x_2|x_1) = \frac{1}{\sqrt{\frac{3\pi}{2}}} \exp\left(-\frac{2}{3}\left(x_2 - \frac{x_1}{2} - \frac{1}{2}\right)^2\right)$$

$\therefore$ $x_1|x_2 \sim N\left(\frac{x_2}{2}+\frac{1}{2}, \frac{3}{4}\right)$

$x_2|x_1 \sim N\left(\frac{x_1}{2}+\frac{1}{2}, \frac{3}{4}\right)$

2.b

```python
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
x1=0
x2=np.sqrt(0.75)*np.random.randn(1,1)+x1/2+0.5
for i in range(5000):
    x1=np.append(x1,np.sqrt(0.75)*np.random.randn()+x2[i]/2+0.5)
    x2=np.append(x2,np.sqrt(0.75)*np.random.randn()+x1[i+1]/2+0.5)
cont,bins,ignored=plt.hist(x1,40,normed=True)
plt.plot(bins,1/np.sqrt(2*np.pi)*np.exp(-(bins-1)**2/2),linewidth=2,color='r')
cont,bins,ignored=plt.hist(x2,40,normed=True)
plt.plot(bins,1/np.sqrt(2*np.pi)*np.exp(-(bins-1)**2/2),linewidth=2,color='r')
```

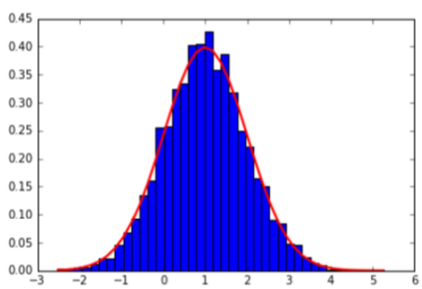

Figure 1: p(x1)

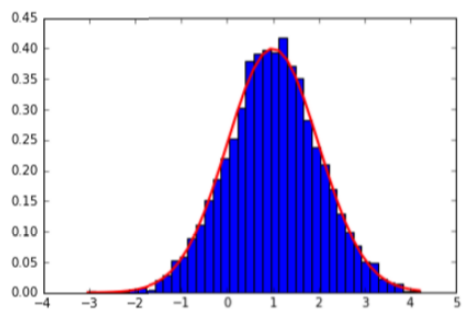

Figure 2: p(x2)

3.a HMM

```python
import numpy as np
#import matrixs
A=np.matrix([[0.5,0.2,0.3],[0.2,0.4,0.4],[0.4,0.1,0.5]])
B=np.matrix([[0.8,0.2],[0.1,0.9],[0.5,0.5]])
pi=np.array([0.5,0.3,0.2])
s=np.array([0,1,0,1])
#set up all possible sequence
P=np.zeros(81)
Z=np.zeros((81,4))
m=0
for i in range(3):
    for j in range(3):
        for l in range(3):
            for k in range(3):
                Z[m,:]=np.matrix([i,j,l,k])
                m=m+1
for i in range(81):
    P[i]=pi[Z[i,0]]*B[Z[i,0],s[0]]*A[Z[i,0],Z[i,1]]*B[Z[i,1],s[1]]*A[Z[i,1],Z[i,2]]*B[Z[i,2],s[2]]*A[Z[i
     ,2],Z[i,3]]*B[Z[i,3],s[3]]
N1=np.argmax(P)
First=Z[N1,:]
P1=np.append(np.append(P[0:N1],0),P[N1+1:])
N2=np.argmax(P1)
```

```
Second=Z[N2,:]
P2=np.append(np.append(P1[0:N2],0),P1[N2+1:])
N3=np.argmax(P2)
Third=Z[N3,:]
#Qa1
Firstpp=pi[First[0]]*A[First[0],First[1]]*A[First[1],First[2]]*A[First[2],First[3]]
Secondpp=pi[Second[0]]*A[Second[0],Second[1]]*A[Second[1],Second[2]]*A[Second[2],Second[3]]
Thirdpp=pi[Third[0]]*A[Third[0],Third[1]]*A[Third[1],Third[2]]*A[Third[2],Third[3]]
print Firstpp,Secondpp,Thirdpp
#Qa2
Firstll=B[First[0],s[0]]*B[First[1],s[1]]*B[First[2],s[2]]*B[First[3],s[3]]
Secondll=B[Second[0],s[0]]*B[Second[1],s[1]]*B[Second[2],s[2]]*B[Second[3],s[3]]
Thirdll=B[Third[0],s[0]]*B[Third[1],s[1]]*B[Third[2],s[2]]*B[Third[3],s[3]]
print Firstll,Secondll,Thirdll
Firstpop=P[N1]/np.sum(P)
Secondpop=P[N2]/np.sum(P)
Thirdpop=P[N3]/np.sum(P)
print Firstpop,Secondpop,Thirdpop
```

We get the result:

Most Probable State Sequences	Prior Probability	Likelihood	Posterior Probability
0222	0.0375	0.1	0.074
0122	0.02	0.18	0.071
0201	0.012	0.288	0.068

3.b

```
from __future__ import division
import numpy as np
# Generate the data according to the specification in the homework description
# for part (b)
A1 = np.array([[0.5, 0.2, 0.3], [0.2, 0.4, 0.4], [0.4, 0.1, 0.5]])
phi = np.array([[0.8, 0.2], [0.1, 0.9], [0.5, 0.5]])
pi0 = np.array([0.5, 0.3, 0.2])
X = []
for _ in xrange(5000):
    z = [np.random.choice([0,1,2], p=pi0)]
    for _ in range(3):
        z.append(np.random.choice([0,1,2], p=A1[z[-1]]))
    x = [np.random.choice([0,1], p=phi[zi]) for zi in z]
    X.append(x)
X=np.array(X)
#get beta
def b(N,A,B):
    Betat=np.zeros([1,12])
    Beta1=np.zeros([1,12])
    for i in range(N):
        Beta=np.zeros([1,12])
        Beta[0,0:3]=1
        for j in range(3):
            for k in range(3):
                Beta[0,3+j]=Beta[0,3+j]+A[j,k]*B[k,X[i,3]]*Beta[0,k]
        for j in range(3):
            for k in range(3):
```

```
                Beta[0,6+j]=Beta[0,6+j]+A[j,k]*B[k,X[i,2]]*Beta[0,3+k]
        for j in range(3):
            for k in range(3):
                Beta[0,9+j]=Beta[0,9+j]+A[j,k]*B[k,X[i,1]]*Beta[0,6+k]
        for j in range(3):
            Beta1[0,j]=Beta[0,9+j]
            Beta1[0,j+3]=Beta[0,6+j]
            Beta1[0,j+6]=Beta[0,3+j]
            Beta1[0,j+9]=Beta[0,j]
        Betat=np.append(Betat,Beta1,axis=0)
    Betat=Betat[1:,:]
    return Betat
#get alpha
def a(N,pi,A,B,X):
    Alphat=np.zeros([1,12])
    for i in range(N):
        alpha=np.zeros([1,12])
        for j in range(3):
            alpha[0,j]=pi[j]*B[j,X[i,0]]
        for j in range(3):
            for k in range(3):
                alpha[0,3+j]=alpha[0,3+j]+alpha[0,k]*A[k,j]
            alpha[0,3+j]=alpha[0,3+j]*B[j,X[i,1]]
        for j in range(3):
            for k in range(3):
                alpha[0,6+j]=alpha[0,6+j]+alpha[0,k+3]*A[k,j]
            alpha[0,6+j]=alpha[0,6+j]*B[j,X[i,2]]
        for j in range(3):
            for k in range(3):
                alpha[0,9+j]=alpha[0,9+j]+alpha[0,k+6]*A[k,j]
            alpha[0,9+j]=alpha[0,9+j]*B[j,X[i,3]]
        Alphat=np.append(Alphat,alpha,axis=0)
    Alphat=Alphat[1:,:]
    return Alphat
#initial the parameter
#random number
R=np.random.uniform(0,1,18)
A=np.matrix([[R[0]/(R[1]+R[2]+R[0]),R[1]/(R[1]+R[2]+R[0]),R[2]/(R[1]+R[2]+R[0])],[R[3]/(R[4]+R[5]+R[3]),R
    [4]/(R[4]+R[5]+R[3]),R[5]/(R[4]+R[5]+R[3])],[R[6]/(R[7]+R[8]+R[6]),R[7]/(R[7]+R[8]+R[6]),R[8]/(R[7]+R
    [8]+R[6])]])
B=np.matrix([[R[9]/(R[10]+R[9]),R[10]/(R[10]+R[9])],[R[11]/(R[12]+R[11]),R[12]/(R[12]+R[11])],[R[13]/(R
    [14]+R[13]),R[14]/(R[14]+R[13])]])
pi=np.array([R[15]/(R[16]+R[17]+R[15]),R[16]/(R[16]+R[17]+R[15]),R[17]/(R[16]+R[17]+R[15])])

def g(N,al,be):
    gamma=np.zeros((al.shape))
    for i in range(N):
        for j in range(12):
            gamma[i,j]=al[i,j]*be[i,j]
    sum1=np.sum(gamma[:,0:3],axis=1)
    sum2=np.sum(gamma[:,3:6],axis=1)
    sum3=np.sum(gamma[:,6:9],axis=1)
    sum4=np.sum(gamma[:,9:12],axis=1)
    for i in range(N):
```

```
        gamma[i,0:3]=gamma[i,0:3]/sum1[i]
        gamma[i,3:6]=gamma[i,3:6]/sum2[i]
        gamma[i,6:9]=gamma[i,6:9]/sum3[i]
        gamma[i,9:12]=gamma[i,9:12]/sum4[i]
    return gamma

def y(k,al,be,A,B):
    yy=np.zeros((3,3))
    a=np.multiply(np.matrix(al[k,0:3]).T*np.matrix(be[k,3:6]),A)
    for l in range(3):
        a[:,l]=a[:,l]*B[l,X[k,1]]
    a=a/np.sum(al[k,3:6]*be[k,3:6])
    b=np.multiply(np.matrix(al[k,3:6]).T*np.matrix(be[k,6:9]),A)
    for l in range(3):
        b[:,l]=b[:,l]*B[l,X[k,2]]
    b=b/np.sum(al[k,6:9]*be[k,6:9])
    c=np.multiply(np.matrix(al[k,6:9]).T*np.matrix(be[k,9:12]),A)
    for l in range(3):
        c[:,l]=c[:,l]*B[l,X[k,3]]
    c=c/np.sum(al[k,9:12]*be[k,9:12])
    yy=a+b+c
    return yy
Z=np.zeros((16,4))
m=0
for i in range(2):
    for j in range(2):
        for l in range(2):
            for k in range(2):
                Z[m,:]=np.matrix([i,j,l,k])
                m=m+1
res=a(16,pi0,A1,phi,Z)
Pt=np.sum(res[:,9:12],axis=1)
N=500
I=50
error=0
loglike1=0
for o in range(I):
    alpha1=a(N,pi,A,B,X)
    beta1=b(N,A,B)
    gamma1=g(N,alpha1,beta1)
    An=np.zeros((3,3))
    for i in range(3):
        pi[i]=np.sum(gamma1[:,i])/np.sum(gamma1[:,0:3])
    for i in range(N):
        An=An+y(i,alpha1,beta1,A,B)
    An=An/np.sum(An,axis=1)
    A=An
    d=np.zeros((N,3))
    for i in range(3):
        for j in range(N):
            d[j,i]=gamma1[j,i]*X[j,0]+gamma1[j,i+3]*X[j,1]+gamma1[j,i+6]*X[j,2]+gamma1[j,i+9]*X[j,3]
        B[i,1]=np.sum(d[:,i])/(np.sum(gamma1[:,i])+np.sum(gamma1[:,i+3])+np.sum(gamma1[:,i+6])+np.sum(
    gamma1[:,i+9]))
    for i in range(3):
```

```
        B[i,0]=1-B[i,1]
    ress=a(16,pi,A,B,Z)
    P=np.sum(ress[:,9:12],axis=1)
    error=np.append(error,np.sum(np.abs(Pt-P))/2)
    ress=a(N,pi,A,B,X)
    P=np.sum(ress[:,9:12],axis=1)
    loglike=1
    for i in range(16):
        loglike=loglike*P[i]
    loglike1=np.append(loglike1,loglike)
error500=error[1:]
#similarly get error1000,2000,5000.
loglike1=loglike1[1:]
#similarly get loglike2,3,4.
```

The distance:

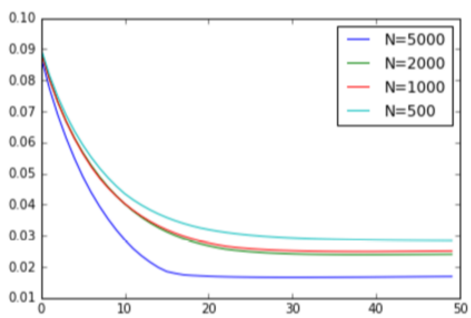

Figure 3: Distance VS iterations

The loglikelihood for 4 choices of N:

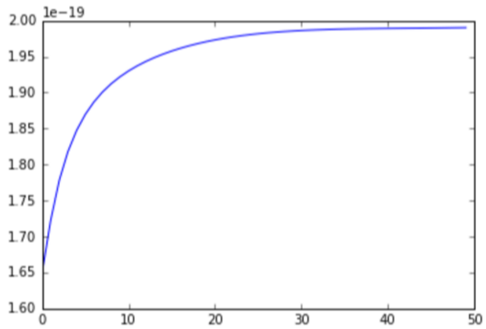

Figure 4: N=500

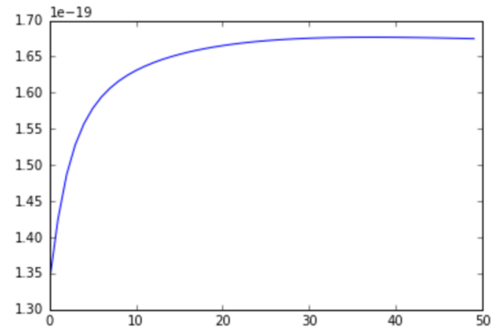

Figure 5: N=1000

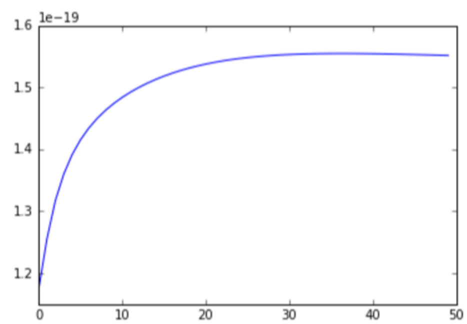

Figure 6: N=2000

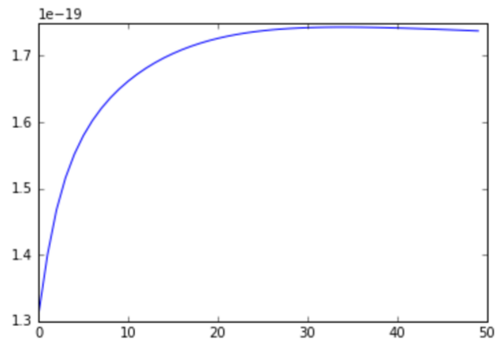

Figure 7: N=5000

Kaggle

```
import numpy as np
X=np.loadtxt("train_noised.csv",delimiter=",", skiprows=1)[:, 1:]
Y=np.loadtxt("train_clean.csv",delimiter=",", skiprows=1)[:, 1:]
t=np.loadtxt("test_noised.csv",delimiter=",", skiprows=1)[:, 1:]
import numpy as np
def f_get_patches(X):
    m,n = X.shape
    X = np.pad(X, ((2, 2), (2, 2)), 'constant')
    patches = np.zeros((m*n, 25))
    for i in range(m):
        for j in range(n):
            patches[i*n+j] = X[i:i+5,j:j+5].reshape(25)
    return patches
X=f_get_patches(X)
Y=Y.reshape(500*784)
t=f_get_patches(t)
from sklearn.ensemble import BaggingRegressor
tc=BaggingRegressor()
tc.fit(X1,Y)
tcr=tc.predict(t)
np.savetxt('result1.csv',tcr,delimiter=",")
```

Q5. (a) For $X \in R^{d \times N}$, Consider $X = U\Sigma V^T$ (the singular value decomposition).

with $U \in R^{d \times d}$ $\Sigma \in R^{d \times N}$ $V \in R^{N \times N}$

Then $XX^T = U\Sigma V^T V \Sigma^T U^T = U \Sigma \Sigma^T U^T$ let $A = \Sigma\Sigma^T$

Choose $D = \sqrt{N}(A^{1/2})^{-1} U^T$ $\tilde{X} = DX$

where $\frac{1}{N}\tilde{X}\tilde{X}^T = \frac{1}{N}\sqrt{N} A^{-\frac{1}{2}} U^T U A U^T U A^{-\frac{1}{2}} \sqrt{N} = I$

5.b ICA

```
from __future__ import division
import numpy as np
# Generate the data according to the specification in the homework description
N = 10000
# Here's an estimate of gamma for you
G = lambda x: np.log(np.cosh(x))
gamma = np.mean(G(np.random.randn(10**6)))
s1 = np.sin((np.arange(N)+1)/200)
s2 = np.mod((np.arange(N)+1)/200, 2) - 1
S = np.concatenate((s1.reshape((1,N)), s2.reshape((1,N))), 0)
A = np.array([[1,2],[-2,1]])
X = A.dot(S)
# TODO: Implement ICA using a 2x2 rotation matrix on a whitened version of X
def w(theta):
    w=np.matrix([[np.cos(theta),-np.sin(theta)],[np.sin(theta),np.cos(theta)]])
    return w
(U,V,A)=np.linalg.svd(np.dot(X,X.T))
X1=np.dot(np.dot(np.linalg.inv(A),U.T)*np.sqrt(N),X)
def J(theta,X):
    a=np.dot(w(theta),X)
    J=(np.mean(G(a[0,:]))-gamma)**2+(np.mean(G(a[1,:]))-gamma)**2
    return J
the=np.linspace(0,np.pi/2,200)
Error=np.zeros(200)
for i in range(200):
    Error[i]=J(the[i],X1)
from matplotlib import pyplot as plt
%matplotlib inline
plt.plot(Error)
ww=np.argmax(Error)
print ww
y=np.dot(w(the[145]),X1)
y1=y[0,:].T
plt.plot(y1)
y2=y[1,:].T
plt.plot(y2)
```

The plot of J vs θ:

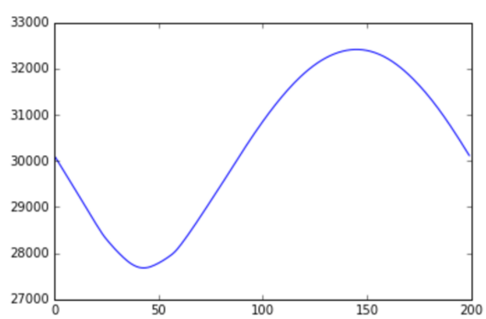

Figure 8: J vs θ

The plot of each row of recovered Y

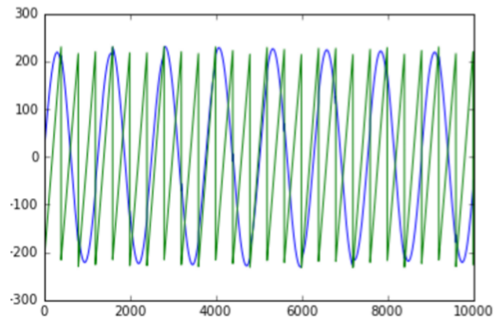

Figure 9: two rows of recovered Y