ENGG4030/ESTR4300 Fall 2016 Homework 4

Due date: Dec 5, 2016 11:59pm

The solution will be posted right after the deadline, so no late homework will be accepted!

Every Student MUST include the following statement, together with his/her signature in the submitted homework.

I declare that the assignment submitted on Elearning system is original

except for source material explicitly acknowledged, and that the same or

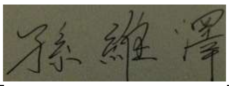
related material has not been previously submitted for another course. I also

acknowledge that I am aware of University policy and regulations on honesty

in academic work, and of the disciplinary guidelines and procedures

applicable to breaches of such policy and regulations, as contained in the

website http://www.cuhk.edu.hk/policy/academ ichonesty/.

Signed (Student______) Date: _____Dec.1st, 2016________

Name________Sun Weize__________ SID____1155062041____________

Submission notice:

- Submit your report in a single PDF document on Elearning

General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be created COMPLETELY by oneself ALONE. A student may not share ANY written work or pictures, nor may one copy answers from any source other than one's own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with** . If the answer includes content from any other source, the student **MUST STATE THE SOURCE** . Failure to do so is cheating and will result in sanctions. Copying answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a

reasonable estimate of its value, and justify any assumptions you make. You will be graded not only on whether your answer is correct, but also on whether you have done an intelligent analysis.

Q1.

MatLab code (here I adapted 99% accuracy, so my new data set contains 53 features each):

```
function [i] = estr4300pca(accuracy)
    train = dlmread('train.txt');
    test = dlmread('test.txt');
    train_labels = train(:, 1);
    test_labels = test(:, 1);
    train = train(:, 2:end);
    test = test(:, 2:end);
    %normalized = normc([train; test]);
    normalized = [train; test];
    sigma = normalized' * normalized / (size(train, 1) + size(test, 1));
    [u, s, v] = svd(sigma);
    sinvalues = diag(s);
    temp = 0;
    sumvalues = sum(sinvalues);
    for i=1:size(train, 2),
        temp = temp + sinvalues(i);
        if(temp / sumvalues >= accuracy),
            break;
        end
    end
    ured = u(:, 1:i);
    newd = [train; test] * ured;
    dlmwrite('newtrain.txt', [train_labels ,newd(1:size(train, 1), :)], '
');
    dlmwrite('newtest.txt', [test_labels, newd(size(train, 1)+1:end, :)], '
');
end
```

Result:

Cluster Number	#images	Major Label	#correctly clustered images	Accuracy%
0	801	2	710	88.6392009988%
1	1093	8	586	53.6139066789%
2	844	1	471	55.8056872038%
3	1433	7	603	42.0795533845%
4	1548	4	556	35.9173126615%
5	1313	3	693	52.779893374%
6	931	6	795	85.3920515575%
7	495	0	450	90.9090909091%
8	997	2	658	65.9979939819%

9	545	0	421	77.247706422%
Total Set	10000	N/A	5943	59.43%

My observation:

The performance of training data set after PCA is slightly worse than original one, but the implementation speed is much faster.

Q2

(a)

MatLab code:

```
function [score, sims] = q2a(x, y)
    R = [1, 1, 5, 5, 4, 0;
        0, 2, 0, 4, 5, 4;
        5, 0, 0, 1, 4, 2;
        2, 1, 4, 5, 0, 5;
        2, 4, 1, 0, 3, 1];
    R = R';
    tempR = normalize(R);
    for k=1:size(tempR, 1),
        sims(k) = tempR(x, :) * tempR(k, :)' / norm(tempR(x, :)) /
norm(tempR(k, :));
    end
    numerator = 0;
    denominator = 0;
    for k=1:size(R, 1),
        if(sims(k) >= 0.1 && R(k, y) ~= 0),
            numerator = numerator + sims(k) * R(k, y);
            denominator = denominator + sims(k);
        end
    end
    %numerator
    %denominator
    score = numerator / denominator;
end

function [x] = normalize(x)
    for i=1:size(x, 1),
        temp = sum(x(i, :)) / size(nonzeros(x(i, :)), 1);
        for j=1:size(x, 2),
            if(x(i, j) ~= 0),
                x(i, j) = x(i, j) - temp;
            end
        end
    end
```

```
end
```

output of command q2a (2,3):

score = 4.4651
sims = [-0.1887 -0.9707 1.0000 0.3022 0.5604 0.6445]

I subtracted the row mean for each nonzero element to reduce the effect of the blank elements.

(b)

MatLab code:

```
function [score, sims] = q2a(x, y)
    R = [1, 1, 5, 5, 4, 0;
        0, 2, 0, 4, 5, 4;
        5, 0, 0, 1, 4, 2;
        2, 1, 4, 5, 0, 5;
        2, 4, 1, 0, 3, 1];
    tempR = normalize(R);
    for k=1:size(tempR, 1),
        sims(k) = tempR(x, :) * tempR(k, :)' / norm(tempR(x, :)) /
norm(tempR(k, :));
    end
    numerator = 0;
    denominator = 0;
    for k=1:size(R, 1),
        if(sims(k) >= 0.1 && R(k, y) ~= 0),
            numerator = numerator + sims(k) * R(k, y);
            denominator = denominator + sims(k);
        end
    end
    %numerator
    %denominator
    score = numerator / denominator;
end

function [x] = normalize(x)
    for i=1:size(x, 1),
        temp = sum(x(i, :)) / size(nonzeros(x(i, :)), 1);
        for j=1:size(x, 2),
            if(x(i, j) ~= 0),
                x(i, j) = x(i, j) - temp;
            end
        end
    end
```

```
end
```

output of command q2b (2, 3):

score = 4.4844
sims = [0.5933 1.0000 0.0725 0.6314 -0.4311]

The predicted score of 2b is slightly larger than 2a. This is because there are more similar candidates corresponding to the target element in 2b, and all of them has a high score.

(c)

Python code:

```
import numpy


def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in xrange(steps):
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i, :], Q[:, j])
                    for k in xrange(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta *
P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta *
Q[k][j])
        eR = numpy.dot(P, Q)
        e = 0
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    e += pow(R[i][j] - numpy.dot(P[i, :], Q[:, j]), 2)
                    for k in xrange(K):
                        e += (beta / 2) * (pow(P[i][k], 2) + pow(Q[k][j], 2))
        print e
        if e < 0.001:
            break
    return P, Q.T, e


R = [[1, 1, 5, 5, 4, 0],
     [0, 2, 0, 4, 5, 4],
     [5, 0, 0, 1, 4, 2],
```

```
    [2, 1, 4, 5, 0, 5],
    [2, 4, 1, 0, 3, 1]]

R = numpy.array(R)

N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N, K)
Q = numpy.random.rand(M, K)
nP, nQ, e = matrix_factorization(R, P, Q, K)
nR = numpy.dot(nP, nQ.T)

print nR
print e
```

The approximated R I got:

1.29528196	0.63870552	4.55787999	5.03879181	4.3312096	4.88197467
2.27555484	2.24694309	3.89341079	4.11254108	4.51856694	4.23753945
4.60803144	6.29705917	1.4951525	0.9670951	4.3504474	1.84220677
1.66779517	1.21483327	4.43344463	4.83435675	4.49861802	4.77216442
2.71064791	3.67934589	0.97044398	0.67364792	2.62743517	1.17957517

The error is 4.53028218041

And the predicted score is 3.89341079

(d)

The program used in 2c did not update the intermediate parameters simultaneously.

Python code:

```
import numpy


def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    for step in xrange(steps):
        tempP = P.copy()
        tempQ = Q.copy()
        for i in xrange(R.shape[0]):
            for j in xrange(R.shape[1]):
                if R[i, j] > 0:
                    eij = R[i, j] - P[i, :]*Q[j, :].T
                    for k in xrange(K):
                        tempP[i, k] += alpha * (2 * eij * Q[j, k] - beta * P[i,
k])
```

```
                    tempQ[j, k] += alpha * (2 * eij * P[i, k] - beta * Q[j,
k])
        e = 0
        P = tempP.copy()
        Q = tempQ.copy()
        for i in xrange(R.shape[0]):
            for j in xrange(R.shape[1]):
                if R[i, j] > 0:
                    e += (R[i, j] - P[i, :]*Q[j, :].T)**2
                    for k in xrange(K):
                        e += (beta / 2) * (P[i, k]**2 + Q[j, k]**2)
        print e[0, 0]
        if e[0, 0] < 0.001:
            break
    return P, Q, e


R = [[1, 1, 5, 5, 4, 0],
     [0, 2, 0, 4, 5, 4],
     [5, 0, 0, 1, 4, 2],
     [2, 1, 4, 5, 0, 5],
     [2, 4, 1, 0, 3, 1]]

R = numpy.mat(R)

N = R.shape[0]
M = R.shape[1]
K = 2

P = numpy.mat(numpy.random.rand(N, K))
Q = numpy.mat(numpy.random.rand(M, K))
nP, nQ, E = matrix_factorization(R, P, Q, K, steps=10000, alpha=0.002,
beta=0.02)
nR = nP*nQ.T

print nR
print E[0, 0]
```

The approximated R I got is:

1.2906368	0.63316161	4.55127066	4.97576625	4.40113986	4.8007224
2.13190302	2.20518354	3.85486178	4.16812917	4.49755197	4.22335706
4.72693429	7.25516267	1.07219225	0.97516445	4.31475123	1.8003538
1.64500614	1.23541344	4.44729762	4.84499575	4.58508154	4.74914623
2.54404998	3.79464272	0.92495799	0.90958092	2.58551936	1.32099335

The error is 4.44001094678
The predicted score is 3.85486178
The program in 2d has lower objective value, and requires more iterations that that in 2c.

Q3

(a)

Suppose $R = i$, then there should be $i-1$ consecutive 1s at the end of all hash values, and there is at least one hash value whose digit i is 0, so we can get the $p.m.f$ equation:

$$P(R=i)=\begin{cases}\left(1-\left(\frac{1}{2}\right)^{N}\right)\left(\frac{1}{2}\right)^{iN}, & 0<i\le\log_2 N\\ \frac{1}{N^N}, & All\ the\ digits\ are\ 1\end{cases}$$

(b)

estr3.sh:

```
#!/bin/bash

rm rsq3b
hadoop dfs -rm -R /user/1155062041/output

hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/data -output
/user/1155062041/output

hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
mv part-00000 rsq3b
```

mapper.py:

```
#!/usr/bin/env python

import sys
import hashlib
import random

def getr(x):
    temp = x + 1
    i = 0
    while 1:
        i += 1
        if temp % 2:
            return i
        else:
            temp /= 2
```

```
m = 169

for line in sys.stdin:
    for word in line.split():
        salt = str(int(hashlib.md5(word).hexdigest(), 16) % m)
        print "{0}\t{1}".format(salt,
getr(int(hashlib.sha1(word+salt).hexdigest(), 16)))
```

reducer.py:

```
#!/usr/bin/env python

import sys

def getr(x):
    temp = x + 1
    i = 0
    while True:
        i += 1
        if temp % 2:
            return i
        else:
            temp /= 2

cur_stream = None
cur_sha = None

for line in sys.stdin:
    substream, shacode = line.split()
    substream = int(substream)
    shacode = int(shacode)
    if substream == cur_stream:
        cur_sha = max(shacode, cur_sha)
    else:
        if cur_stream:
            print "{0}\t{1}".format(cur_stream, cur_sha)
        cur_stream = substream
        cur_sha = shacode
print "{0}\t{1}".format(cur_stream, cur_sha)
```

The I copied the file rsq3b to MatLab root directory

q3b.m:

```
function [n] = q3b()
```

```
    R = dlmread('rsq3b');
    R = R(:, 2);
    m = length(R);
    alpha = (gamma(-1/m) * (1 - 2 ^(1/m)) / log(2)) ^ (-m);
    n = alpha * m * 2 ^ (sum(R) / m);
end
```

In my code, I set $m = 169$
The estimation of my q3b.m program was 64941
The true cardinality got by Hadoop wordcount was 72379
Error: 10.28%

(c)

estr3.sh: (I just changed the filename, other parts are the same as that in (a))

```
#!/bin/bash

rm rsq3c
hadoop dfs -rm -R /user/1155062041/output

hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/data -output
/user/1155062041/output

hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
mv part-00000 rsq3c
```

mapper.py: (I just changed the value of m to be 106)

```
#!/usr/bin/env python

import sys
import hashlib
import random

def getr(x):
    temp = x + 1
    i = 0
    while 1:
        i += 1
        if temp % 2:
            return i
        else:
            temp /= 2

m = 106
```

```python
for line in sys.stdin:
    for word in line.split():
        salt = str(int(hashlib.md5(word).hexdigest(), 16) % m)
        print "{0}\t{1}".format(salt,
getr(int(hashlib.sha1(word+salt).hexdigest(), 16)))
```

reducer.py:

The same as that of (b)

The I copied the file rsq3c to MatLab root directory

q3c.m:

```matlab
function [n] = q3c()
    R = dlmread('rsq3c');
    R = R(:, 2);
    m = length(R);
    n = betam(m) * m^2 / sum(2.^(-R));
end

function [betam_output] = betam(m)
    fun = @(u) (log2((2+u)./(1+u)).^m);
    betam_output = (m*integral(fun, 0, Inf))^-1;
end
```

In my code, I set $m = 106$
The estimation of my q3b.m program was 78816
The true cardinality got by Hadoop wordcount was 72379
Error: 8.89%

The running time of (b) and (c) are similar due to similar m and the convenience and power of MatLab. However, the accuracy of (c) is higher, because LogLog algorithm adapts geometric mean, while HyperLogLog uses harmonic mean, which is more stable.

(d)

estr3d.sh:

```bash
#!/bin/bash

rm shakespeare
hadoop dfs -rm -R /user/1155062041/output
hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/data -output
/user/1155062041/output
hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
```

```
mv part-00000 shakespeare

rm Gutenberg_A
hadoop dfs -rm -R /user/1155062041/output
hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/gutenberg_A -output
/user/1155062041/output
hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
mv part-00000 Gutenberg_A


rm Gutenberg_B
hadoop dfs -rm -R /user/1155062041/output
hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/gutenberg_B -output
/user/1155062041/output
hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
mv part-00000 Gutenberg_B

rm Gutenberg_C
hadoop dfs -rm -R /user/1155062041/output
hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/gutenberg_C -output
/user/1155062041/output
hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
mv part-00000 Gutenberg_C

rm Gutenberg
hadoop dfs -rm -R /user/1155062041/output
hadoop jar hadoop-streaming.jar -file mapper.py -mapper mapper.py -file
reducer.py -reducer reducer.py -input /user/1155062041/gutenberg -output
/user/1155062041/output
hadoop dfs -copyToLocal /user/1155062041/output/part-00000 .
mv part-00000 Gutenberg
```

mapper.py: (I changed m to be 424, the other parts are the same as before)

```
#!/usr/bin/env python

import sys
import hashlib
import random

def getr(x):
    temp = x + 1
```

```
    i = 0
    while 1:
        i += 1
        if temp % 2:
            return i
        else:
            temp /= 2

m = 424

for line in sys.stdin:
    for word in line.split():
        salt = str(int(hashlib.md5(word).hexdigest(), 16) % m)
        print "{0}\t{1}".format(salt,
getr(int(hashlib.sha1(word+salt).hexdigest(), 16)))
```

reducer.py:

The same as that in (b) and (c)

Then I copied file shakespeare, Gutenberg_A, Gutenberg_B, Gutenberg_C, Gutenberg_D to MatLab root directory.

q3d.m:

```
function [sug, sua, sub, suc, sig, sia, sib, sic] = q3d()
    m = 423;

    sf = dlmread('shakespeare');
    gf = dlmread('Gutenberg');
    af = dlmread('Gutenberg_A');
    bf = dlmread('Gutenberg_B');
    cf = dlmread('Gutenberg_C');

    s = zeros(m, 1);
    g = zeros(m, 1);
    a = zeros(m, 1);
    b = zeros(m, 1);
    c = zeros(m, 1);

    for i=1:length(sf),
        s(sf(i, 1)) = sf(i, 2);
    end
    for i=1:length(gf),
        g(gf(i, 1)) = gf(i, 2);
    end
```

```
    for i=1:length(af),
        a(af(i, 1)) = af(i, 2);
    end
    for i=1:length(bf),
        b(bf(i, 1)) = bf(i, 2);
    end
    for i=1:length(cf),
        c(cf(i, 1)) = cf(i, 2);
    end

    sg = max(s, g);
    sa = max(s, a);
    sb = max(s, b);
    sc = max(s, c);

    sug = betam(m) * m^2 / sum(2.^(-sg));
    sua = betam(m) * m^2 / sum(2.^(-sa));
    sub = betam(m) * m^2 / sum(2.^(-sb));
    suc = betam(m) * m^2 / sum(2.^(-sc));

    sn = betam(m) * m^2 / sum(2.^(-s));
    gn = betam(m) * m^2 / sum(2.^(-g));
    an = betam(m) * m^2 / sum(2.^(-a));
    bn = betam(m) * m^2 / sum(2.^(-b));
    cn = betam(m) * m^2 / sum(2.^(-c));

    sig = sn + gn - sug;
    sia = sn + an - sua;
    sib = sn + bn - sub;
    sic = sn + cn - suc;
end

function [betam_output] = betam(m)
    fun = @(u) (log2((2+u)./(1+u)).^m);
    betam_output = (m*integral(fun, 0, Inf))^-1;
end
```

estr3dtrue.sh:

```
#!/bin/bash

hadoop dfs -rm -R /user/1155062041/single
rm part-r-00000
hadoop jar hadoop-mapreduce-examples.jar wordcount /user/1155062041/data
/user/1155062041/single
```

```
hadoop dfs -copyToLocal /user/1155062041/single/part-r-00000
mv part-r-00000 shakespeare_single

hadoop dfs -rm -R /user/1155062041/single
rm part-r-00000
hadoop jar hadoop-mapreduce-examples.jar wordcount
/user/1155062041/gutenberg /user/1155062041/single
hadoop dfs -copyToLocal /user/1155062041/single/part-r-00000
mv part-r-00000 gutenberg_single

hadoop dfs -rm -R /user/1155062041/single
rm part-r-00000
hadoop jar hadoop-mapreduce-examples.jar wordcount
/user/1155062041/gutenberg_A /user/1155062041/single
hadoop dfs -copyToLocal /user/1155062041/single/part-r-00000
mv part-r-00000 gutenberg_A_single

hadoop dfs -rm -R /user/1155062041/single
rm part-r-00000
hadoop jar hadoop-mapreduce-examples.jar wordcount
/user/1155062041/gutenberg_B /user/1155062041/single
hadoop dfs -copyToLocal /user/1155062041/single/part-r-00000
mv part-r-00000 gutenberg_B_single

hadoop dfs -rm -R /user/1155062041/single
rm part-r-00000
hadoop jar hadoop-mapreduce-examples.jar wordcount
/user/1155062041/gutenberg_C /user/1155062041/single
hadoop dfs -copyToLocal /user/1155062041/single/part-r-00000
mv part-r-00000 gutenberg_C_single

python estr3d.py
```

estr3d.py:

```python
sf = open('shakespeare_single', 'r').read().split('\n')[:-1]
gf = open('gutenberg_single', 'r').read().split('\n')[:-1]
af = open('gutenberg_A_single', 'r').read().split('\n')[:-1]
bf = open('gutenberg_B_single', 'r').read().split('\n')[:-1]
cf = open('gutenberg_C_single', 'r').read().split('\n')[:-1]

s = set([x.split()[0] for x in sf])
g = set([x.split()[0] for x in gf])
a = set([x.split()[0] for x in af])
```

```
b = set([x.split()[0] for x in bf])
c = set([x.split()[0] for x in cf])

sug = s.union(g)
sua = s.union(a)
sub = s.union(b)
suc = s.union(c)
sig = s.intersection(g)
sia = s.intersection(a)
sib = s.intersection(b)
sic = s.intersection(c)

print "sug: {0}".format(len(sug))
print "sua: {0}".format(len(sua))
print "sub: {0}".format(len(sub))
print "suc: {0}".format(len(suc))
print "sig: {0}".format(len(sig))
print "sia: {0}".format(len(sia))
print "sib: {0}".format(len(sib))
print "sic: {0}".format(len(sic))
```

The final statistics table:

	Estimated cardinality of Union set	True cardinality of union set	relative error	Estimated cardinality of intersection set	True cardinality of intersection set	relative error
Shakespeare and gutenberg_A	73048	72412	0.9%	746	627	19.0%
Shakespeare and guterberg_B	73048	72418	0.9%	880	799	10.1%
Shakespeare and gutenberg_C	73190	72478	1.0%	1573	1515	3.8%
Shakespeare and gutenberg	279560	284377	1.7%	61879	58578	5.6%