CS246 Mining Massive Datasets Winter 2013 - Final Answers

Charlie Zhang

Mar 2013

Name: Xiao Zhang

SUNetID: charliez@stanford.edu

I acknowledge and accept the Honor Code.

Signature: Xiao Zhang

1 1 MapReduce

1.1 (a)

Pseudocode in Python:

```
def map(key, val):
    output(val, '')
def reduce(key, vals):
    output(key, key)
```

The values in reducer output are all the distinct elements.

The reduce function could be used as Combiner.

1.2 (b)

Pseudocode in Python:

```
def map(key, val):
    output(rand(m), val)
def reduce(key, vals):
    for v in vals:
        output('', v)
```

Assuming that the value in map phase is the actual element from the dataset.

1.3 (c)

Total data flow between mapper and reducer (ignoring possible existence of combiner):

600M * 1KB = 1 GB

If we use Combiner and only calculate the network cost between combiner and reducer, the result depends on the distribution of IDs and number of mappers.

TODO

1.4 (d)

Simpler version, without any optimization on network cost and assuming not using combiner:

TODO

```
def map(key, val):
    output(value.supplier, (value.productId, value.price))
def reduce(key, vals):
    totalPrice = defaultdict(float)
    count = defaultdict(int)
    for (productId, price) in vals:
        totalPrice[productId] += price
        count[productId] += 1
    for k in totalPrice.keys():
        output(key, (k, totalPrice[k]))
```

1.5 (e)

False

1.6 (f)

False

1.7 (g)

True

1.8 (h)

False

2 2 Distance Measures

2.1 (a)

Jaccard distance: 1 - 1/5

2.2 (b)

1 - arccos(1/3)

2.3 (c)

For binary vectors,

$cossim(v1, v2) = \frac{v1*v2}{|v1|*|v2|} = \frac{|S(v1) \cap S(v2)|}{\sqrt{|S(v1)|*\sqrt{|S(v2)|}}}$,
Whereas $jacccardsim(v1, v2) = \frac{|S(v1) \cap S(v2)|}{|S(v1) \cup S(v2)|}$

Where S(v) represents the set of indexes where $v_i = 1$.

$\sqrt{|S(v1)|} * \sqrt{|S(v2)|} \leq max(|S(v1)|, |S(v2)|) \leq |S(v1) \cup S(v2)|$,

So $jacccardsim(v1, v2) \leq cossim(v1, v2)$

Therefore, Jaccard distance is always greater or equal to Cosine distance.

3 3 Shingling

3.1 (a)

Runtime complexity is $O(nk)$. using a double-ended queue to generate each shingle in O(1) time.

3.2 (b)

Not necessarily identical.

A: {a, b, c, a b}

B: {b, c, a, b, c}

3.3 (c)

The statement is false, counter example:

A: {1, 2, 3, ... n, 1, 2, 3, ... n}

B: {2, 3, ... n, 1, 2, 3, ... n, 1}

4 4 Minhashing

4.1 (a)

Assuming we used AND construction in each band, and OR constructions between bands.

$1-(1-\frac{1}{2}^r)^b$, i.e. $1-(1-\frac{1}{2}^2)^5$

4.2 (b)

$1-(1-x^r)^b=\frac{1}{2}$

$x=(1-\frac{1}{2}^{\frac{1}{b}})^{\frac{1}{r}}$

5 5 Random hyperplanes

5.1 (a)

Vector Sketch

a -1, 1, 1

b -1, 1, -1

c 1, -1, -1

5.2 (b)

Vector Angle

a,b arccos(1/3)

b,c arccos(-1/3)

a,c arccos(-1)

6 6 Market Baskets

6.1 (a)

Minimum: 1 (if items in one basket can be duplicated)

Minimum: 10 (if items in one basket are unique)

Maximum: 100000 items.

Reasoning:

If all the baskets are identical, this will give us minimum number of frequent items.

1 million baskets * 10 items per basket = 10^7 items. Dividing 10^5 different items we get 100 support for each item on average. If every item appears exactly 100 times, we'll have 100000 frequent items, which is the maximum.

6.2 (b)

Minimum: 0

Reasoning: There are $10^5 * (10^5 - 1)/2$ possible different pairs, but only $10 * 9/2 * 10^6 = 4.5 * 10^7$ pairs appeared from baskets. $\frac{4.5*10^7}{10^5*(10^5-1)/2} << 100$,In worst case, none of the pairs are frequent.

Maximum: $floor(\frac{4.5*10^7}{100}) = 4.5 * 10^5$

Reasoning: $10 * 9/2 * 10^6 = 4.5 * 10^7$ pairs from basket, assuming each distance pair appears exactly 100 times, this gives us $4.5 * 10^5$ frequent pairs.

7 7 Counting Pairs of Items

7.1 (a)

We need 5*4 = 20 bytes per node for the binary search tree and there are p nodes.

So total memory consumption is 20p bytes.

7.2 (b)

Triangluar matrix takes $n(n-1)/2 * 4 = 2n(n-1)$ bytes.

When $20p < 2n(n-1)$, using binary-search tree would be more efficient than matrix.

8 8 Clustering

Steps,	Old clusters,	New cluster,	new clustroid
Step 1,	{1}, {4}	{1, 4}	2.5
Step 2,	{1,4}, {9}	{1,4,9}	4.67
Step 3,	{16}, {25}	{16,25}	20.1
Step 4	{1,4,9}, {16,25}	{1,4,9,16,25}	11

9 9 Singular Value Decomposition

9.1 (a)

[5.22, 1.42]

9.2 (b)

Tony would like MMDS more than Mechanics, since his previous reviews suggests he has stronger interest in the Computer Science concept than Mechanical Engineering

9.3 (c)

[4.06, 6.39]

9.4 (d)

$sim = \frac{5.22*4.06+1.42+6.39}{\sqrt{5.22^2+1.42^2}\sqrt{4.06^2+6.39^2}} = 0.71$

10 10 Recommender Systems

10.1 (a)

R could be Content-based recommender system. Since the recommended movie B's genre meet U1's interest.

TODO

10.2 (b)

TODO

11 11 PageRank

11.1 (a)

$r(1) = 0.2 * \frac{1}{6} + 0.8 * (1/5 * r(2) + 1/2 * r(4) + r(6))$

$r(2) = 0.2 * \frac{1}{6} + 0.8 * (1/3 * r(3) + 1/2 * r(5))$

$r(3) = 0.2 * \frac{1}{6} + 0.8 * (1/5 * r(2) + 1/2 * r(5))$

$r(4) = 0.2 * \frac{1}{6} + 0.8 * (1/5 * r(2))$

$r(5) = 0.2 * \frac{1}{6} + 0.8 * (1/5 * r(2) + 1/3 * r(3))$

$r(6) = 0.2 * \frac{1}{6} + 0.8 * (1/5 * r(2) + 1/2 * r(4) + 1/3 * r(3))$

11.2 (b)

1 6 5 3 2 4

TODO

11.3 (c)

M =

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$h = \beta * L * a$

$a = \beta * L^T * h$

12 12 Machine Learning

12.1 (a)

When dataset is large, Model A tends to over fit to the data. (the C term becoming relatively large)

However, Model B will have larger training error but less over fitting.

12.2 (b)

C

12.3 (c)

C

12.4 (d)

C

12.5 (e)

Works well on linearly non-separable data

Easy to interpret the classification process by human

13 13 AMS 3rd moment calculation

$E[X] = 1/n * \sum_{k=1}^{m_a} (n(3k^2 - 3k + 1)) = m_a^3$

14 14 Streams: DGIM

(16, 148) (8, 162) (8, 177) (4, 183) (4, 200) (2, 204) (2, 208) (1, 210)

TODO

15 15 Streams: Finding The Majority Element

count = 0

result = None

for i in elements:

if i != result:

```
        if count == 0:
            result = i
            count = 1
        else: count -= 1
    else: count += 1
```

Result will always be the majority element.

Since the count -1 when current majority don't agree with the next element, and +1 when agrees, the majority element has n/2 occurrence, it will 'win over' all other elements summing together, so the final result will always be the majority element.