Joshua Sloan
University of Wyoming
jsloan3@uwyo.edu

Homework 5
COSC 5010 - Data Mining

1 Problems from the Book

1.1 2.3.1

a) The largest integer

```
Map: For each integer i, emit(1,i)
Reduce: For key = 1, emit(max(values))
```

(where max() is a function that returns the maximum value from a list)

b) The average of all integers

```
Map: For each integer i, emit(1,(1,i))
Reduce: For key = 1,

weightSum = 0
averageSum = 0
For each value in values:
    weightSum += value[0]
    averageSum += (value[0] * value[1])
emit(1,(weightSum, (averageSum/weightSum)))
```

c) The same set of integers, but with each integer appearing only once

```
Map: For each integer i, emit(i,1)
Reduce:

distincts = []
For each key in keys:
    distincts.append(key)
emit(result, distincts)
```

NOTE: For this to work, you need to ensure there is only 1 reducer!

d) The count of the number of distinct integers in the input

```
Map: For each integer i, emit(i,1)
Reduce:

numDistincts = 0
For each key in keys:
    numDistincts = numDistincts + 1
emit(result, numDistincts)
```

NOTE: For this to work, you need to ensure there is only 1 reducer!

Also, you could simply use the same Map/Reduce functions from part c) except emit(result, length(distincts)), however, this should be slightly faster (especially as the number of distinct elements grows to be very large).

1.2 3.2.1

"The most effective way to represent documents as sets, for the purpose of identifying lexically similar documents is to construct from the document the set of short strings that appear within it."

First 10 3-shingles (words, removing punctuation):
The most effective
most effective way
effective way to
way to represent
to represent documents
represent documents as
documents as sets
as sets for
sets for the
for the purpose

1.3 3.3.3

a)

h1 = 2x+1 mod 6
h2 = 3x+2 mod 6
h3 = 5x+2 mod 6

Element	c1	c2	c3	c4	h1	h2	h3
0	0	1	0	1	1	2	2
1	0	1	0	0	3	5	1
2	1	0	0	1	5	2	0
3	0	0	1	0	1	5	5
4	0	0	1	1	3	2	4
5	1	0	0	0	5	5	3

Now following the method detailed in 3.3.5 of the book we build the following min-hashing signatures:

	c1	c2	c3	c4
h1(0)	∞	1	∞	1
h2(0)	∞	2	∞	2
h3(0)	∞	2	∞	2
h1(1)	∞	1	∞	1
h2(1)	∞	2	∞	2
h3(1)	∞	1	∞	2
h1(2)	5	1	∞	1
h2(2)	2	2	∞	2
h3(2)	0	1	∞	0
h1(3)	5	1	1	1
h2(3)	2	2	5	2
h3(3)	0	1	5	0
h1(4)	5	1	1	1
h2(4)	2	2	2	2
h3(4)	0	1	4	0
h1(5)	5	1	1	1
h2(5)	2	2	2	2
h3(5)	0	1	4	0

From the above, we obtain the following final min-hash signature matrix:

c1	c2	c3	c4
5	1	1	1
2	2	2	2
0	1	4	0

b)

Functions h1 and h2 produce far too many identical signature values (despite having very different column values), therefore, only h3 is a true permutation.

c)

Similarity matrix:

	1-2	1-3	1-4	2-3	2-4	3-4
col/col	0	0	0.25	0	0.25	0.25
sig/sig	0.33	0.33	0.67	0.67	0.67	0.67

It appears as though the signature matrix is NOT a good estimate for the true Jaccard similarity between columns.

1.4 3.3.6

Let M be the following matrix:

c1	c2
0	1
1	1
0	0

such that all 3 possible permutations of M (following the given criteria from the problem) are:

c1	c2
0	1
1	1
0	0

= M1

c1	c2
1	1
0	0
0	1

= M2

and

c1	c2
0	0
0	1
1	1

= M3

The Jaccard similarity of of matrix M can be defined as the intersection over the union of both columns, such that sim(M) = $\frac{1}{2}$.

If we apply a min hashing function, h, to each permutation of M, we have:

h(M1) = 2,1
h(M2) = 1,1
and
h(M3) = 3,2 such that:

c1	c2
2	1
1	1
3	2

= h(M).

Now comparing column similarity, we can see that only the second row, h(M2), have the same column values, such that our estimate of the Jaccard similarity between columns is $\frac{1}{3} \neq \frac{1}{2}$. □

1.5 3.4.4

a)
Map:

```
Divide value into b bands //predetermined number
For i in bands
    emit(hash(bands[i],(i,bands[i]))
```

Reduce:

```
buckets = []
For each value in values:
    If (key,value[0]) not in buckets:
        buckets.append((key,value[0]))
For each bucket in buckets:
    list = []
    For each value in values:
        If (key,value[0]) == bucket
            list.append(value[1])
    emit(concatenate(bucket[0],bucket[1]),list)
```

NOTE: This reduce function will return each unique bucket with a key id specific to the hash output and the band it is associated with, as well as a value which is a list of lists containing all of the signature values hashing to the bucket within the band.

b)
Map:

```
For i=0; i < length(value-1); i++:
    For j=i+1; j < length(value); j++:
        emit(key,(value[i],value[j]))
```

NOTE: This function will get all list pairs from the list of lists (without duplicates).

Reduce:

```
pairs = []
For value in values:
    pairs.append(value)
emit(key,pairs)
```

1.6 4.3.2

Let:
k = number of hash functions,
m = number of of elements to add,
n = size of the bit array.

If we divide our n bits of memory into k smaller arrays (such that each array has 1 single corresponding hash function), then our false positive probability = $(1 - e^{-m/(n/k)})^k$.

We know that the false positive probability of the standard Bloom Filter algorithm (i.e. k different hash functions onto a single array of size n) is equal to $(1 - e^{-km/n})^k$.

Given that $\frac{-m}{\frac{n}{k}} = \frac{-mk}{n} = \frac{-km}{n}$, we can see that both of these functions are identical and therefore, will produce the exact same result. $\square$

1.7 4.3.3

Let:
k = number of hash functions,
m = number of of elements to add,
n = size of the bit array.

To minimize the false positive rate, we have already seen in the lecture notes that k = (n/m) ln(2) hash functions should be utilized.

1.8 4.4.1

a) h(x) = 2x + 1 mod 32
r = trailing zeros
estimate of distinct elements = $2^{max(r)}$

2(3) + 1 mod 32 = 7 = 00111, where r = 0 S.T. 2^r = 1
2(1) + 1 mod 32 = 3 = 00011, where r = 0 S.T. 2^r = 1
2(4) + 1 mod 32 = 9 = 01001, where r = 0 S.T. 2^r = 1
2(1) + 1 mod 32 = 3 = 00011, where r = 0 S.T. 2^r = 1
2(5) + 1 mod 32 = 11 = 01011, where r = 0 S.T. 2^r = 1
2(9) + 1 mod 32 = 19 = 10011, where r = 0 S.T. 2^r = 1
2(2) + 1 mod 32 = 5 = 00101, where r = 0 S.T. 2^r = 1
2(6) + 1 mod 32 = 13 = 01101, where r = 0 S.T. 2^r = 1
2(5) + 1 mod 32 = 11 = 01011, where r = 0 S.T. 2^r = 1

The maximum value for r = 0, $\therefore$ the estimated number of distinct elements for this hash function is 2^r = 1. □

b) h(x) = 3x + 7 mod 32
r = trailing zeros
estimate of distinct elements = $2^{max(r)}$

3(3) + 7 mod 32 = 16 = 10000, where r = 4 S.T. 2^r = 16
3(1) + 7 mod 32 = 10 = 01010, where r = 1 S.T. 2^r = 2
3(4) + 7 mod 32 = 19 = 10011, where r = 0 S.T. 2^r = 1
3(1) + 7 mod 32 = 10 = 01010, where r = 1 S.T. 2^r = 2
3(5) + 7 mod 32 = 22 = 10110, where r = 1 S.T. 2^r = 2
3(9) + 7 mod 32 = 2 = 00010, where r = 1 S.T. 2^r = 2
3(2) + 7 mod 32 = 13 = 01101, where r = 0 S.T. 2^r = 1
3(6) + 7 mod 32 = 25 = 11001, where r = 0 S.T. 2^r = 1
3(5) + 7 mod 32 = 22 = 10110, where r = 1 S.T. 2^r = 2

The maximum value for r = 4, $\therefore$ the estimated number of distinct elements for this hash function is 2^r = 16.
□

c) h(x) = 4x mod 32
r = trailing zeros
estimate of distinct elements = $2^{max(r)}$

4(3) mod 32 = 12 = 01100, where r = 2 S.T. 2^r = 4
4(1) mod 32 = 4 = 00100, where r = 2 S.T. 2^r = 4
4(4) mod 32 = 16 = 10000, where r = 4 S.T. 2^r = 16
4(1) mod 32 = 4 = 00100, where r = 2 S.T. 2^r = 4
4(5) mod 32 = 20 = 10100, where r = 2 S.T. 2^r = 4
4(9) mod 32 = 4 = 00100, where r = 2 S.T. 2^r = 4
4(2) mod 32 = 8 = 01000, where r = 3 S.T. 2^r = 8
4(6) mod 32 = 24 = 11000, where r = 3 S.T. 2^r = 8
4(5) mod 32 = 20 = 10100, where r = 2 S.T. 2^r = 4

The maximum value for r = 4, $\therefore$ the estimated number of distinct elements for this hash function is 2^r = 16.
□

1.9 4.5.3

Given the stream: 3,1,4,1,3,4,2,1,2

If we calculate $X_{element}$ and X_{value} at each potential starting point we have:

i = 0, S.T. $X_{element}$ = 3 and X_{value} = 2
i = 1, S.T. $X_{element}$ = 1 and X_{value} = 3
i = 2, S.T. $X_{element}$ = 4 and X_{value} = 2
i = 3, S.T. $X_{element}$ = 1 and X_{value} = 2
i = 4, S.T. $X_{element}$ = 3 and X_{value} = 1
i = 5, S.T. $X_{element}$ = 4 and X_{value} = 1
i = 6, S.T. $X_{element}$ = 2 and X_{value} = 2
i = 7, S.T. $X_{element}$ = 1 and X_{value} = 1
i = 8, S.T. $X_{element}$ = 2 and X_{value} = 1

Given that there are 9 elements in the stream, our estimation from the Alon-Matias-Szegedy Algorithm is:

$\frac{\sum_{i=0}^{8} 9*(2*X_{value}(i)-1)}{9}$

$= \frac{9*((2(2)-1)+(2(3)-1)+(2(2)-1)+(2(2)-1)+(2(1)-1)+(2(1)-1)+(2(2)-1)+(2(1)-1)+(2(2)-1))}{9}$

$= (2(2)-1)+(2(3)-1)+(2(2)-1)+(2(2)-1)+(2(1)-1)+(2(1)-1)+(2(2)-1)+(2(1)-1)+(2(2)-1)$

= 3 + 5 + 3 + 3 + 1 + 1 + 3 + 1 + 1

= 21 □

NOTE: I was unsure if we were also asked to calculate the value (given the wording of the first sentence). I did regardless just to be safe.

2 Bloom Filter

For my bloom filter implementation, I utilized a bit array of size 8*|S| (similarly to the example in the lecture notes). This can however be changed by modifying the BloomFilter's get_size() class method within bloomfilter.py.

Please see corresponding folder containing my python code, the output from running it on the datasets and the README file (instructions to replicate on your own machine).

For the output file, I calculate the expected FPR from our known equations and then verify this FPR on 10 trials of subsets from the data stream (utilize multiple subset trials to decrease overall run-time).

3 Flajolet-Martin Algorithm

For my implementation of the Flajolet-Martin Algorithm, I used various murmur hash functions (for different seeds). Results were obtained using 4 hash functions as well as 6 (both outputs are included in problem 3 folder). However, the algorithm is structured such that this number can be changed to any even integer by adjusting the numHashes variable within the code. That being said, memory requirements and run-time increase alongside the value of numHashes.

Please see corresponding folder containing my python code, the output from running it on the dataset files and the README file (instructions to replicate on your own machine).

NOTE: The algorithm takes a while to work through all the data, so please allow time if replicating the results locally!