Mining Big Data assignment 2

Made by

Yu Zhang a1795409, Qiming Liu a1798587

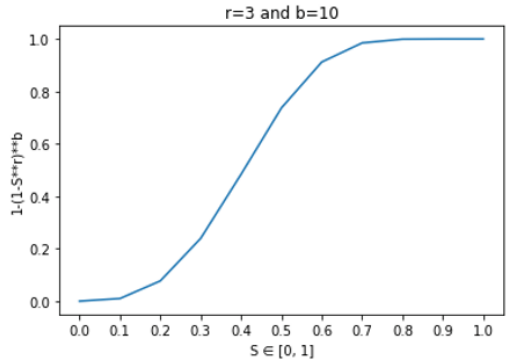
Exercise 1

See files: Exercise 1 S-curve.zip (code and image result)

	S	1-(1-S**r)**b
0	0.0	0.000000
1	0.1	0.009955
2	0.2	0.077181
3	0.3	0.239449
4	0.4	0.483871
5	0.5	0.736924
6	0.6	0.912267
7	0.7	0.985015
8	0.8	0.999234
9	0.9	0.999998
10	1.0	1.000000

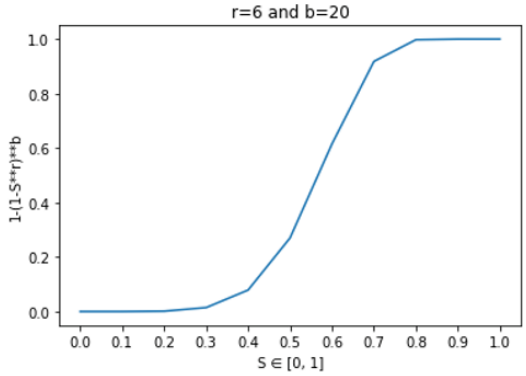
Plot S-curve

```
plt.title('r=3 and b=10')
plt.plot(x, y)
plt.xlabel('S ∈ [0, 1]')
plt.ylabel('1-(1-S**r)**b')
plt.xticks(x)
plt.show()
```



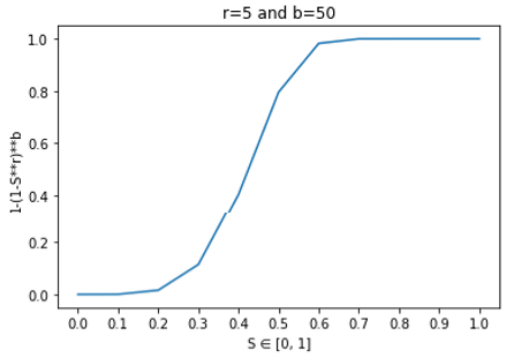
	S	1-(1-S**r)**b
0	0.0	0.000000
1	0.1	0.000500
2	0.2	0.015875
3	0.3	0.114540
4	0.4	0.402284
5	0.5	0.795551
6	0.6	0.982534
7	0.7	0.999899
8	0.8	1.000000
9	0.9	1.000000
10	1.0	1.000000

```
plt.title('r=6 and b=20')
plt.plot(x, y)
plt.xlabel('S ∈ [0, 1]')
plt.ylabel('1-(1-S**r)**b')
plt.xticks(x)
plt.show()
```



	S	1-(1-S**r)**b
0	0.0	0.000000
1	0.1	0.000500
2	0.2	0.015875
3	0.3	0.114540
4	0.4	0.402284
5	0.5	0.795551
6	0.6	0.982534
7	0.7	0.999899
8	0.8	1.000000
9	0.9	1.000000
10	1.0	1.000000

```
plt.title('r=5 and b=50')
plt.plot(x, y)
plt.xlabel('S ∈ [0, 1]')
plt.ylabel('1-(1-S**r)**b')
plt.xticks(x)
plt.show()
```



Exercise 2

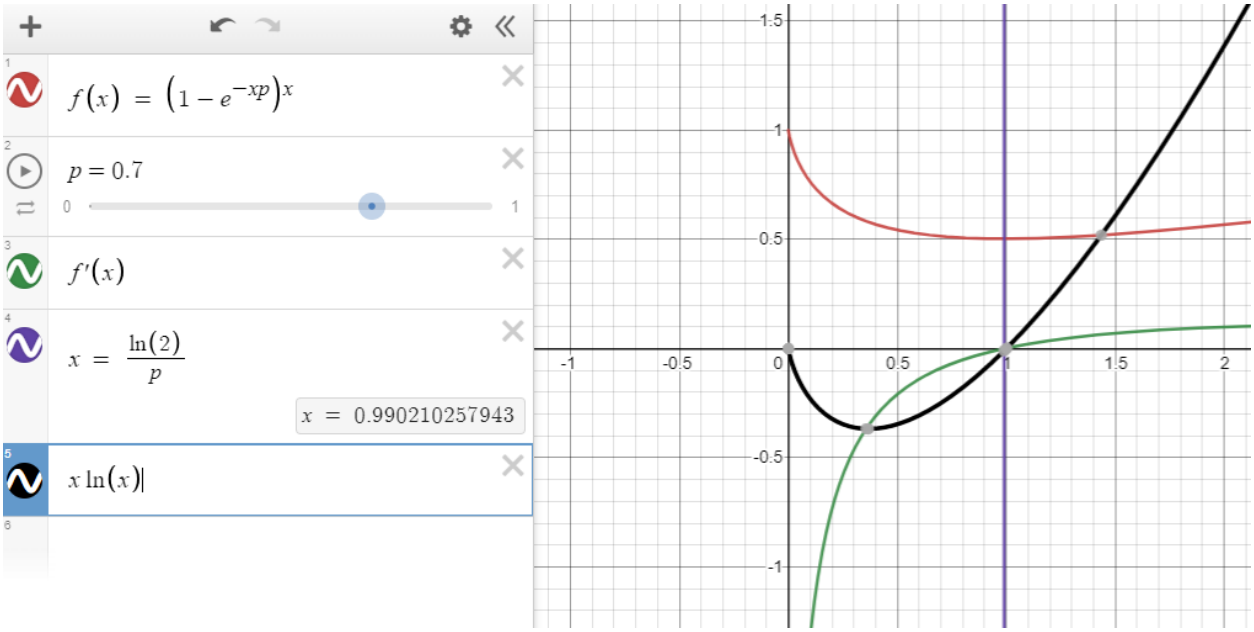
Q1: With changed conditions 10 billion bits, 2 billion members of Set S, calculate the false- positive rate.

False positive rate = (1-e^(-km/n)) ^ k which n is bit array length, m is number of S set, k is number of Hash function

1. K =3: False positive rate = (1-e^(-3*2 b/10 b)^3 = 0.09185
2. K =4: False positive rate = (1-e^(-4*2 b/10 b)^4 = 0.09195

Q2: AS n number of bits and m the number of members in the set S, what number of hash function minimize the false positive rate?

To Find the minimize false positive rate, the turning point needs to be found out.



dy/dk = f'(k) = ((1-e^(-km/n)) ^ k)'

Assume x = e^(-km/n)

Figure out the derivative.

xln(x) = (1-x)ln(1-x)

x = 1/2

e^(-km/n) = 1/2

So the minimize number of hash functions: k = n/m * ln(2)

Exercise 3

Result:

924: 439, 2409, 6995, 11860, 15416, 43748, 45881

8941: 8943, 8944, 8940

8942: 8939, 8940, 8943, 8944

9019: 9022, 317, 9023

9020: 9021, 9016, 9017, 9022, 317, 9023

9021: 9020, 9016, 9017, 9022, 317, 9023

9022: 9019, 9020, 9021, 317, 9016, 9017, 9023

9990: 13134, 13478, 13877, 34299, 34485, 34642,37941

9992: 9987, 9989, 35677, 9991

9993: 9991, 13134, 13478, 13877, 34299, 34485, 34642, 37941

Description:

In the mapping step.

1. Build another writable structure instead of single variable value called friendWritable, include a user and a mutual friend variables which user is the current friends, and the mutual friend is the status that two people are friends or not.
2. Split the friend list with ",". Map all the data, build the main user to its current friends with status 0 in new structure (user ,(friend, status code)), means they are already friends. Build user's friends each other's map with status 1, means they are potential friends(have mutual friend main user)
3. In the reduce step, build a Hashmap. To check every friendwritable value, if status is 0, means already friends, hash the target User to -1, if not, hash it start from one.
4. Use a queue to sort the friends id by its mutual friends number decreasing and Id number increasing. And out put the queue with ",".

Exercise 4

Question 1: Hash function: h(x) = (2x + 1) mod 32

Number	Hash	Binary	Tail length
3	7	00111	0
1	3	00011	0
4	9	01001	0
6	13	01101	0
5	11	01011	0
9	19	10011	0

Max Tail Length R = 0

Estimate number of distinct elements 2^R= 2^0 = 1

Question 2: Hash function: h(x) = (3x + 7) mod 32

Number	Hash	Binary	Tail length
3	16	00111	4
1	10	11010	1
4	19	10011	0
6	25	11001	0
5	22	10110	1
9	2	00010	1

Max Tail Length R = 4

Estimate number of distinct elements 2^R = 2^4 =16

Question 3: Hash function: h(x) = 4x mod 32

Number	Hash	Binary	Tail length
3	12	10000	4
1	4	00100	2
4	9	10000	4
6	16	11000	3
5	24	10100	2
9	4	00100	2

Max Tail Length R = 4

Estimate number of distinct elements 2^R = 2^4 =16

Question 4: Hash function: h(x) = (6x + 2) mod 32

Number	Hash	Binary	Tail length
4	26	11010	1
5	0	00000	0
6	6	00110	1
7	12	01100	2
10	30	11110	1
15	28	11100	2

Max Tail Length R = 2

Estimate number of distinct elements 2^R= 2^2 = 4

Question 5: Hash function: h(x) = (2x + 5) mod 32

Number	Hash	Binary	Tail length
4	13	01101	0
5	15	01111	0
6	17	10001	0
7	19	10011	0
10	25	11001	0
15	3	00011	0

Max Tail Length R = 0

Estimate number of distinct elements 2^R= 2^0 = 1

Question 6: Hash function: h(x) = 2x mod 32

Number	Hash	Binary	Tail length
4	8	01000	3
5	10	01010	1
6	12	01100	2
7	14	01110	1
10	20	10100	2
15	30	11110	1

Max Tail Length R = 3

Estimate number of distinct elements 2^R= 2^3 = 8

Exercise 5

Part 1 Section 3.6

The steepness of S-curve can show the effectively of avoiding false positive and false negatives in candidate pairs. There are also other functions can find candidate pairs effectively except minhash. There are three requirements for functions that can apply to space of set and Jaccard distance to another space or distance effectively.

1. The functions should make close pairs have higher priority than distant pairs.
2. The functions need to be statistically independent. Which makes possible to estimate that two or more functions would give certain response
3. The functions must be efficient. (a) Functions must identify candidate pairs much faster than scanning all pairs. (b) Functions can be combined to build a new set of functions to avoid false positive and false negative. Also, the functions need to take less time than scanning all pairs.

The method that the functions judging two items are candidate pair or not is hash the items. Assume there is a function f, and item x and y. The hash of x and y are f(x) and f(y). If f(x) = f(y), means the pair x and y can be candidate. If f(x) != f(y), that means this two item can not be candidate pairs in the function f. (They may still have possibility under other functions)

A collection of these kind of functions f is called a family of functions. Such as the function set based on the permutations of row in minhash, is a family.

Assume the d1 and d2 for distance and d1 < d2. A family of functions F is (d1, d2, p1, p2) which is sensitive for each function in F. They follow:

1. If distance for x and y, d(x, y) <= d1. The p1 is the least possibility for f(x) = f(y) (for every function in the family
2. For d(x, y) >= d2, the possibility of f(x) = f (y) is most p2.

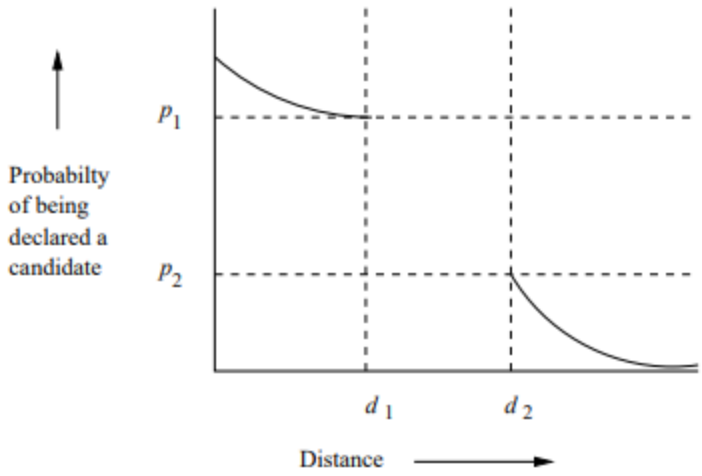


Figure 3.10: Behavior of a (d_1, d_2, p_1, p_2)-sensitive function

The picture shows that when the d1 and d2 get close, the p1 an p2 will also get close as well. Because the family function is (d1, d2, 1 - d1, 1 - d2). Which 0 <= d1 < d2 <= 1. For example, if d1 = 0.2 and d2 = 0.8, the family of functions is (0.2, 0.7, 0.8, 0.3). It means with most 0.2 Jaccard distance, there is least 0.8 chance that hash function send x and y with same value, which f(x) = f(y). Meanwhile, there most 0.3 chance the x and y with same hash value when the distance is at least 0.7.

Amplify

With a given functions family F. A new family function F' can be built with an AND-construction on F. The function family F' consists of a static number r on member of F. "r" is the rows in single band. Because the r row band made x and y a candidate pair if every one of r rows reflects f(x) = f(y). So the new function family F' is (d1, d2, (p1)^r, (p2)^r), since every p is independent. There is also OR-construction convert the original family F to F' (d1, d2, 1-(1-p1)^b, 1-(1-p2)^b) ((1-p)^b is the chance that none of family function will declare x and y are candidate pairs , which 1 – this possibility means at least there is one function show the f(x) = f(y)).

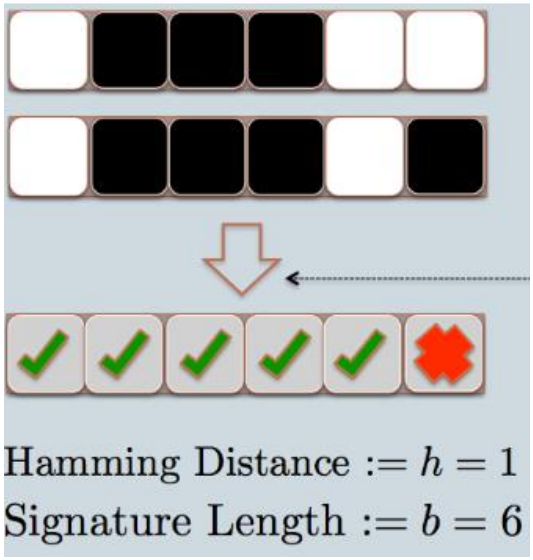
In conclusion, the AND-construction will lower the possibility, and OR-construction can increase the possibility that found the candidate pairs. But because of the change rate of exponent. AND-construction made the smaller possibility tend to ZERO, meanwhile the OR-construction made the larger possibility tend to 1. But when more construction be used, means the r and b is getting larger. Which will increase the time complexity of the family.

For example, apply a 4 way Or construction and 4 way AND construction on family F(0.2, 0.6, 0.8, 0.4). P= 0.8, 0.4. (1 - (1 - p) ^ 4) ^ 4 = 0.9936, 0.5740. Then constructed family is (0.2, 0.6, 0.9936, 0.574) sensitive.

Part 2 Section 3.7

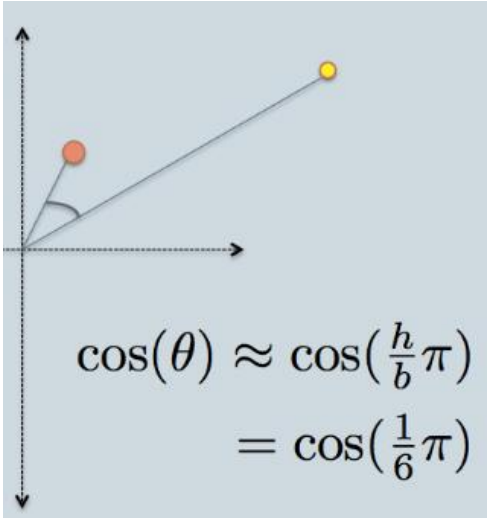
From the previous chapter, we learned that locality-sensitive hashing (LHS) is an approximate nearest neighbor fast search technology for massive high-dimensional data. This chapter covers other distance measures in locality-sensitive hashing families: hamming distance, cosine distance, and normal euclidean distance.

Hamming distance:



Hamming distance is the simplest one. The bit sampling of the Hamming distance depends on the number of differences between the two vectors. The figure shows an example of Hamming distance in a signature length of 6. In the book, this kind of S-curve becoming very steep is unlikely. In a d-dimensional space, if h(x, y) is used to represent the Hamming distance of x, y vectors, then randomly choose a position i, the probability that x, y intersect is 1 − h(x, y)/d. Thus, {f1, f2 … fd} is a (d1, d2, 1 − d1/d, 1 − d2/d)-sensitive family. If d1 is smaller than d2, the biggest difference between this f family and the previous minhash is that the size of hamming distance {f1, f2 … fd} is d. So, this makes it impossible for the S-curve to become very steep when d is very small. On the other hand, the Hamming distance is in the range of 0-d in the d-dimensional space, and all can be divided by d to get the probability at the appropriate time, which is not very important.

Cosine distance:

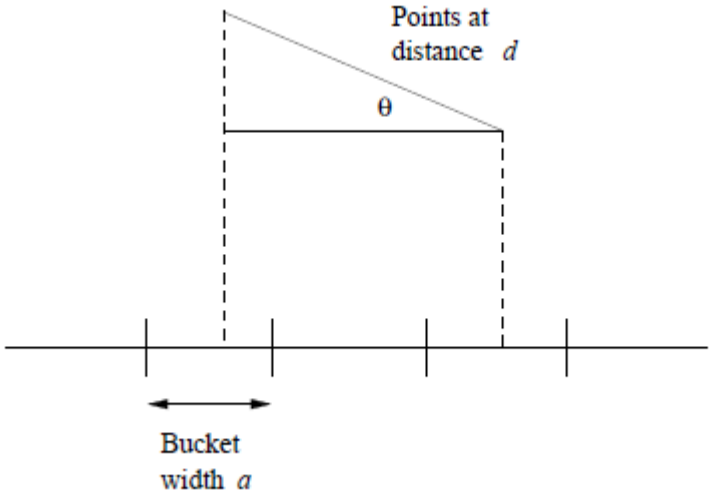


The hyperplane is one less than the current space dimension, it can split this space into two spaces. Cosine distance can be understood as the smaller angle between two hyperplanes in space. Then the scale of cosine distance is 0-180. An example in the figure is that the angle between the orange and yellow vectors is the cosine distance, and for the example in the Hamming distance, their cosine distance is cos(1/6π). It's a (d1, d2, (180 − d1)/180, (180 − d2)/180)-sensitive family. Similar to the minhash base family, it can amplify.

Sketches:

The sketch is more straightforward, we use +1 and -1 to denote normal vectors. For a vector v applied to a vector x, values greater than 0 are treated as +1, and values less than 0 are treated as -1. Then this sketch of x will only contain +1 and -1. Finally, we can calculate the ratio of the same exponent by comparing the two sketches and multiplying this percentage by 180.

Normal Euclidean distance:



If R is a random vector, a is the bucket width, b∈[0, a], and v · R is the projection of v on R, then this is a (a/2, 2a, 1/2, 1/3)-sensitive family and h(v) = | v · R + b| / a. It can amplify.

More Euclidean Spaces:

It is mentioned in the book that the disadvantage in the normal Euclidean distance above is that it is only responsible for the two-dimensional Euclidean space. The condition needs to be changed to d1<4d2 to develop the locality-sensitive family. For any d1<d2, we can split lines using random lines and buckets of width a. It is a (d1, d2, p1, p2)-sensitive family. Using amplification techniques, we can adjust the probabilities to increase the distance between them, and hopefully use a large number of underlying hash functions.