Subject: ***Information Retrieval and Analysis***
Student: ***Stanciu Iulia-Cristina***
Group: *12*
Practicals: ***Lab9 – LSH – 09.01.2023***

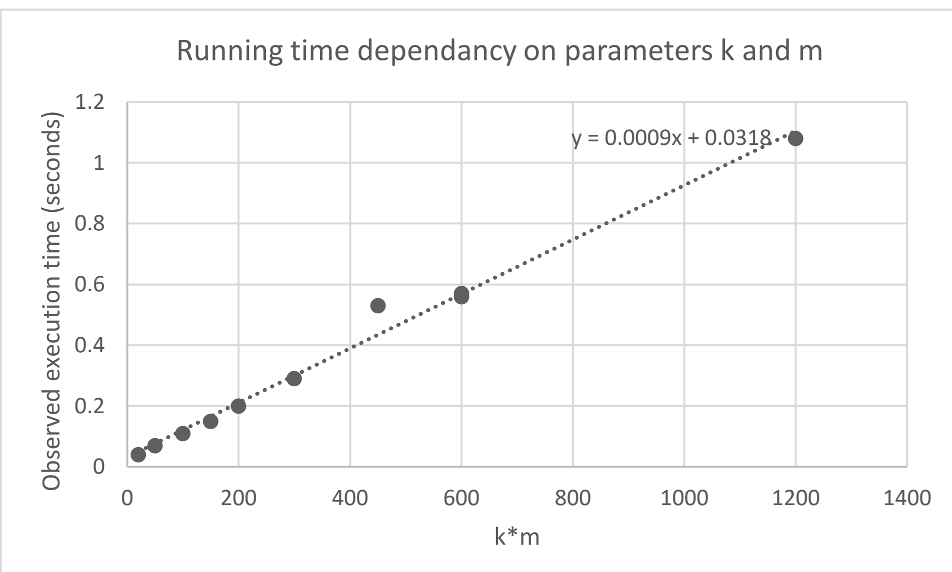
Locality Sensitive Hashing (LSH)

Task 1: Understanding the code and basic working of LSH

- The running time of the hashing process is $O(km)$ in theory, by looking at the code. Does this agree with your experiments?

Experiments:

k	m	k*m	Running time (s)
10	2	20	0.04
10	5	50	0.07
20	5	100	0.11
20	10	200	0.20
30	5	150	0.15
30	10	300	0.29
30	15	450	0.53
30	20	600	0.56
40	15	600	0.57
40	30	1200	1.08



As it can be observed in the graphic above, the running time best follows a linear dependancy on the parameters k and m. This agrees with the observations made by analysing the code (that the running time is O(km)).

- What happens to the size of the candidate set when increasing k?

  1. **k is increasing.**

The candidates set is decreasing. Each image of a number is read as pixels. Each pixel is treated as a characteristic of the image. In the first step of lsh, we convert each image into a set of characteristics of length k. Afterwards, images with the same set of characteristics are being found. A small value for k will result in a large number of documents peresented as “similar” because the characteristics will be present in most images (higher false positives). A large number of shingkles may result in a small number of candidates, if there will be any (higher false negatives).

Subject: ***Information Retrieval and Analysis***
Student: ***Stanciu Iulia-Cristina***
Group: *12*
Practicals: ***Lab9 – LSH – 09.01.2023***

2. m is increasing.

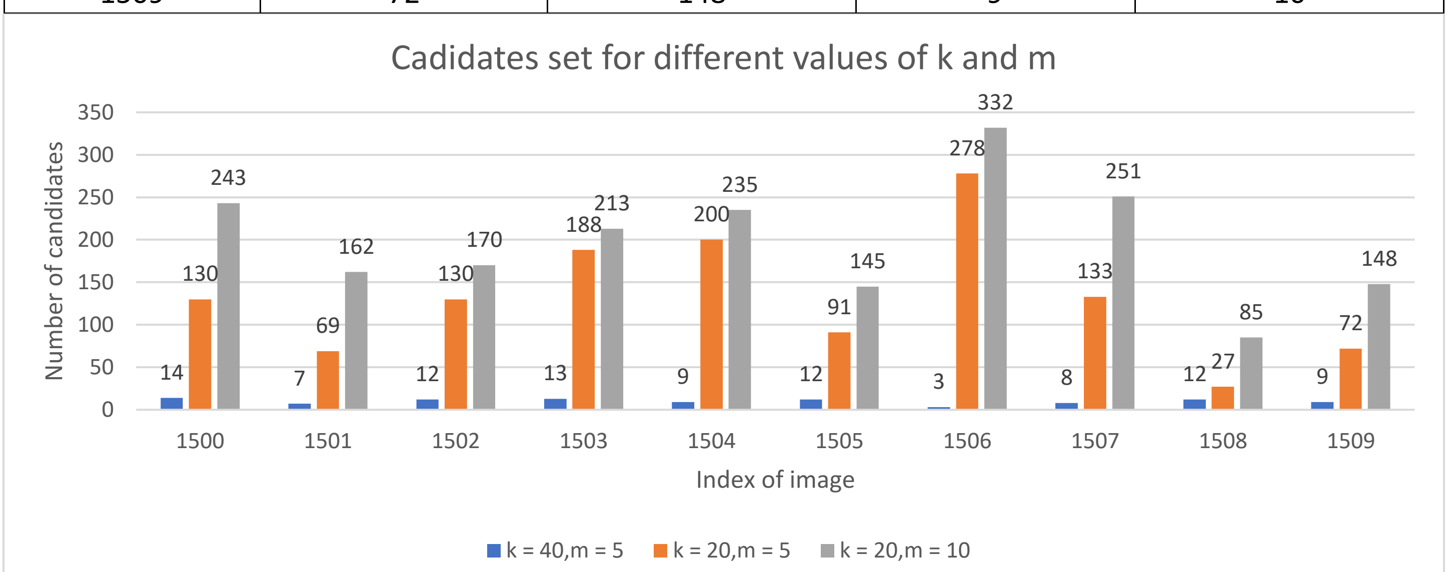
```
Running lsh.py with parameters k = 20 and m = 5
there are  130 candidates for image 1500
there are   69 candidates for image 1501
there are  130 candidates for image 1502
there are  188 candidates for image 1503
there are  200 candidates for image 1504
there are   91 candidates for image 1505
there are  278 candidates for image 1506
there are  133 candidates for image 1507
there are   27 candidates for image 1508
there are   72 candidates for image 1509
```

```
Running lsh.py with parameters k = 20 and m = 10
there are  243 candidates for image 1500
there are  162 candidates for image 1501
there are  170 candidates for image 1502
there are  213 candidates for image 1503
there are  235 candidates for image 1504
there are  145 candidates for image 1505
there are  332 candidates for image 1506
there are  251 candidates for image 1507
there are   85 candidates for image 1508
there are  148 candidates for image 1509
```

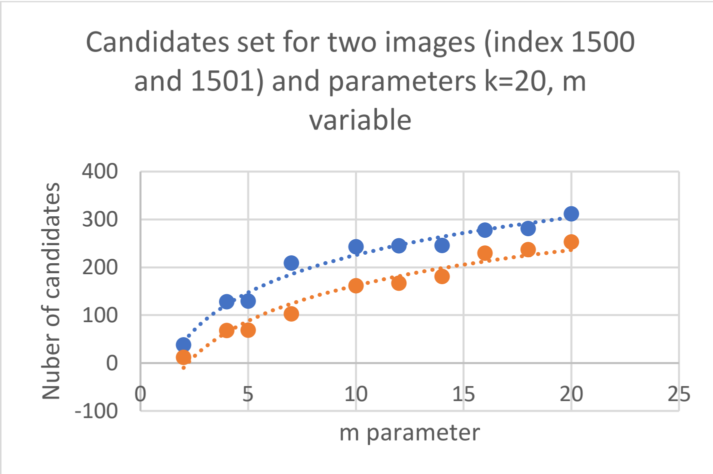
The candidates set is increasing. m is the number of dictionaries in witch the hashed images are stored. When m is increasing, the chance of each hash to be part of more dictionaries rises, therefore, the number of candidates per image rises.

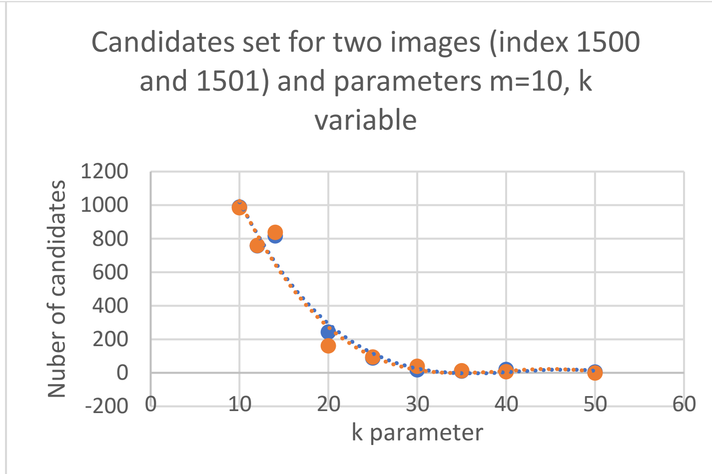
- Can one give a function $f(k,m)$ expressing the size of the candidate set? (or not?)

Parameters	**k = 20**	**m = 5**	**k = 20**	**m = 10**	**k = 40**	**m = 5**	**k = 40**	**m = 10**
Image Index	**Candidates**							
1500	130		243		14		20	
1501	69		162		7		7	
1502	130		170		12		19	
1503	188		213		13		15	
1504	200		235		9		12	
1505	91		145		12		16	
1506	278		332		3		14	
1507	133		251		8		20	
1508	27		85		12		14	
1509	72		148		9		10	



Subject: ***Information Retrieval and Analysis***
Student: ***Stanciu Iulia-Cristina***
Group: *12*
Practicals: ***Lab9 – LSH – 09.01.2023***





Analyzing the modifications in the number of candidates per image in the case of two images, I have hound that the best trending line for a increase of the m factor is an increasing logarithmic, while a rise of the k factor can be best represented as a decreasing polynomial of order 3. I believe that a function f(k, m) expressing the size of the candidate set should exist and could be discovered after a longer analysis.

Task 2: Does lsh work?

Algorithm	Brute-force search	LSH k=5, m=2	LSH k=10, m=5	LSH k=20, m=5	LSH k=20, m=10	LSH k=30, m=5	LSH k=30, m=10	LSH k=30, m=15	LSH k=30, m=20	LSH k=50, m=20
Time (s)	3.57	1.54	2.15	0.46	0.69	0.21	0.41	0.60	0.79	1.02
Accuracy	1	0.885	0.990	0.774	0.926	0.441	0.630	0.734	0.815	0.427
Balanced Accuracy	1	0.900	0.992	0.754	0.923	0.421	0.578	0.675	0.730	0.394

Conclusions:

- For high numbers of k or low values for m, lsh cannot find candidates for some images., therefore the accuracy score is lower.
- Brute-force search checks all the possible combinations and gives the exact nearest neighbor, but its complexity makes it not scalable at all. Depending on the parameters given, LSH manages to give the exact same answer or a near neighbor at a comparable l1 distance in a decently large to good amount of cases.
- Similar distances are given even in the worst-case scenario. Analyzing the table above, we can see that for medium values of k and m, results are very good, but runtime and the small test set must also be taken into consideration. For k=10 and m=5, the runtime is almost half compared to brute-force search and the accuracy is almost 99%. For an even faster approach, k=20, m=10 parameters were chosen; In a fifth of the time used by bfs, lsh manages to give a very good accuracy of approximately 92%, while the different neighbors have similar distances (for example, for k=10, m=5, for image 1789, the nearest neigbour found by lsh is at distance 30, while the actual one is at 26.

Subject: ***Information Retrieval and Analysis***
Student: ***Stanciu Iulia-Cristina***
Group: *12*
Practicals: ***Lab9 – LSH – 09.01.2023***

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya