Lab Session 8: Locality Sensitive Hashing

Alex Carrillo, Sonia Rabanaque
Cerca i Anàlisi de la Informació, Ciència i Enginyeria de Dades (UPC)

November 4, 2019

4.1 Task 1: Understanding the code and basic working of lsh

When increasing `k`, does the time it takes to execute this code vary? How?

Si executem el codi diversos cops fixant un valor de `m` (en aquest cas, l'hem fixat a $m = 10$) i variant el de `k` (el número de bits que comparem) podem veure que el temps d'execució canvia. En concret observem que, quan el valor de `k` augmenta, el temps d'execució també ho fa.

When increasing `m`, does the time it takes to execute this code vary? How?

Si fem el contrari, és a dir, fixar la `k` i modificar el valor de `m` (que és el número de repeticions, és a dir, les vegades que escollim `k` funcions de hash diferents) també veiem que el temps d'execució no és el mateix sempre. Com en el cas anterior, podem dir que són proporcionals, ja que augmentant el valor de `m` també ho fa el temps d'execució.

What happens to the size of the candidate set when increasing `k`? And when you increase `m`? Can you explain why?

Però, al modificar els valors d'aquestes variables no només varia el temps d'execució, sinó que també ho fa el número de possibles candidats. Al fixar la `m` i augmentar la `k`, cada cop és més difícil que coincideixin en els `k` bits i, per tant, reduim els possibles candidats. En canvi, al fixar la `k` i augmentar la `m`, el número de candidats augmenta. Això és degut a que, en cada repetició agafem `k` posicions de bits per comparar. Aquestes no seran les mateixes sempre i, per tant, fem comparacions diferents que poden tenir resultats (dir si són "iguals" o no, si van a la mateixa casella de la taula de hash o no) diferents. Així doncs, el conjunt dels possibles candidats d'una de les repeticions, en general, no serà el mateix que el d'una altra repetició. Per tant, el número de candidats augmenta amb `m` perquè, normalment, anem afegint nous en cadascuna de les repeticions.

4.2 Task 2: Does lsh work?

De tot el conjunt de Test, que són 297 imatges, hem mirat les 20 primeres per observar la variació entre els resultats.

És important resaltar que, en els casos en els que `lsh_search` i `br_search` no troben el mateix *nearest neighbour*, sempre el que té distància més petita és el de força bruta. Això és el que esperàvem, ja que és una cerca exhaustiva i, per tant, prova totes les imatges de *training*. En canvi, lsh només prova els que tenen la mateixa funció de hash. Per tant, l'algorisme de força bruta sempre troba resultats iguals o amb menys distància.

Si comparem els temps que triga en mitjana cada mètode en trobar el *nearest neighbour* també podem veure que, quan fem servir les funcions de hash el temps és d'uns 0.01 segons (si la `k` és prou gran) mentre que, amb força bruta triga, aproximadament, uns 0.13 segons.

A continuació, mostrem alguns valors per corroborar aquest fets i explicar-los amb més deteniment.

Primer, fixem el valor de `m` ($m = 5$) i canviem el de la `k` $(5, 50, 100)$.

k	Image	Method	Times	NN	Distances
5	**1502**	bf	0.12 sec	1429	60.0
		lsh	0.12 sec	1429	60.0
5	**1518**	bf	0.12 sec	1421	60.0
		lsh	0.15 sec	1421	60.0
50	**1503**	bf	0.12 sec	1045	64.0
		lsh	0.00 sec	1045	64.0
50	**1512**	bf	0.12 sec	1439	40.0
		lsh	0.00 sec	None	inf
50	**1518**	bf	0.13 sec	1498	51.0
		lsh	0.00 sec	865	75.0
100	**1514**	bf	0.13 sec	1462	38.0
		lsh	0.00 sec	1462	38.0
100	**1517**	bf	0.12 sec	1461	55.0
		lsh	0.00 sec	None	inf

Com mostrem, en el primer cas ($k = 5$), el *nearest neighbour* tant amb cerca exhaustiva com amb les funcions de hash és el mateix en totes les imatges que hem provat.

En canvi, quan $k = 10$ ens surten les 3 possibilitats: que siguin iguals, que la funció de hash no trobi cap candidat, i que trobin diferents imatges com les més similars. Destaquem que hi ha molt pocs casos en els quals NN de `lsh` sigui `None`.

Per útlim, quan la `k` és molt gran, només trobem 2 casos dels 20 que hem provat en els que la funció `lsh_search` ha trobat algun *nearest neighbour*.

El temps de `lsh_search` és, en els dos últims casos, de 0.00 segons aproximadament. Això és degut a que, com la `k` ha augmentat, el número de candidats ha disminuit.

A continuació farem el contrari, és a dir, fixem el valor de `k` a 50 i canviem el de la variable `m` $(5, 10, 50)$.

m	Image	Method	Times	NN	Distances
5	**1507**	bf	0.12 sec	1452	45.0
		lsh	0.00 sec	1282	80.0
5	**1512**	bf	0.12 sec	1439	40.0
		lsh	0.00 sec	None	inf
10	**1503**	bf	0.12 sec	1045	64.0
		lsh	0.00 sec	1045	64.0
10	**1512**	bf	0.12 sec	1439	40.0
		lsh	0.00 sec	None	inf
10	**1518**	bf	0.13 sec	1498	51.0
		lsh	0.00 sec	865	75.0
50	**1513**	bf	0.13 sec	1475	79.0
		lsh	0.01 sec	1460	85.0
50	**1514**	bf	0.13 sec	1462	38.0
		lsh	0.01 sec	1462	38.0

Quan la `m` és massa petita, la majoria dels *nearest neighbours* són diferents, però també n'hi ha que no troben cap i d'altres que són iguals (aquests últims en la mateixa proporció, més o menys).

En el segon cas, passa el mateix que en l'anterior, tot i que n'hi ha una mica més de diferents i no tants amb valor `None` i iguals.

Quan la `m` és més gran, pràcticament tots (excepte un) coincideixen tant usant l'algorisme de força bruta com lsh. En els casos que hem vist, cap ha donat `None`.

Això és degut a que, quan augmentem el nombre de repeticions, el nombre de possibles candidats augmenta. Per aquest motiu també veiem que, en l'últim cas, augmenta el temps d'execució ja que s'ha de comparar amb més candidats.