CAIM LAB8:
Local-Sensitive Hashing

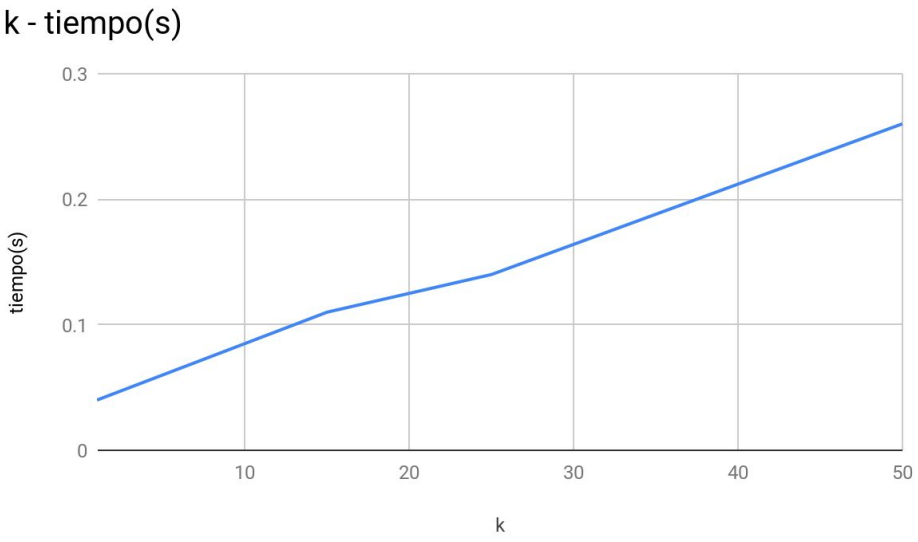
Yimin Pan

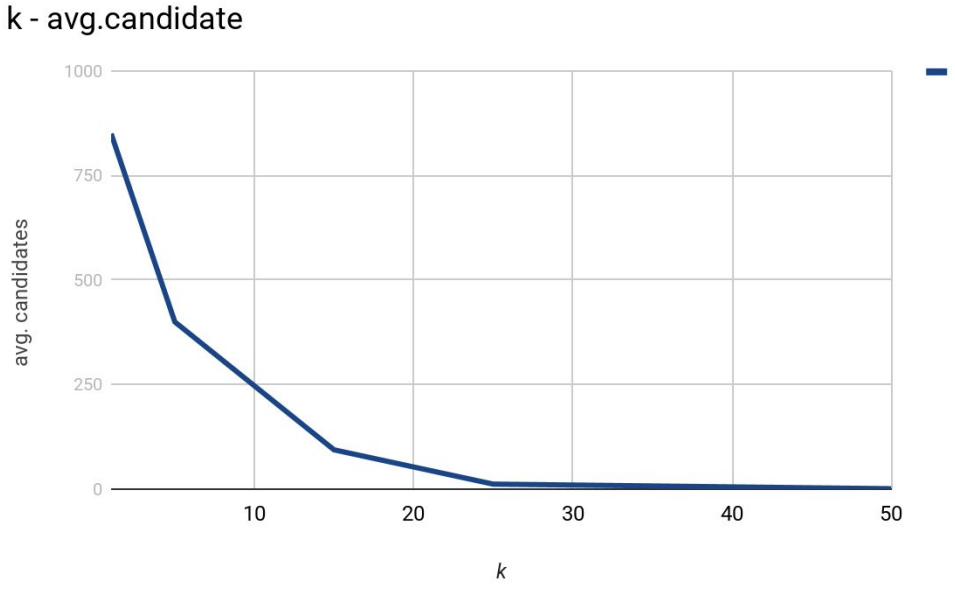
Introducción

En esta práctica aprenderemos a usar local-sensitive hashing para encontrar los vecinos más cercanos de unas imagenes. Así como, ver cómo afecta al tiempo y resultando cuando modificamos los parámetros **k** y **m**.

Cómo afecta ‘m’ y ‘k’ a los resultados?

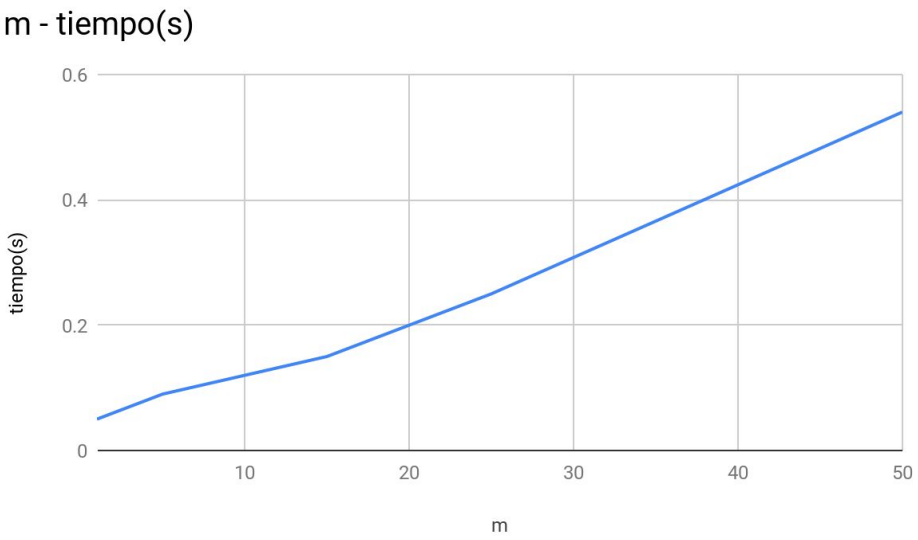
Para poder estudiar mejor estos parámetros, fijaremos el valor del otro. De este modo, fijaremos **m = 1** y vemos cómo afecta **k** a los resultados.

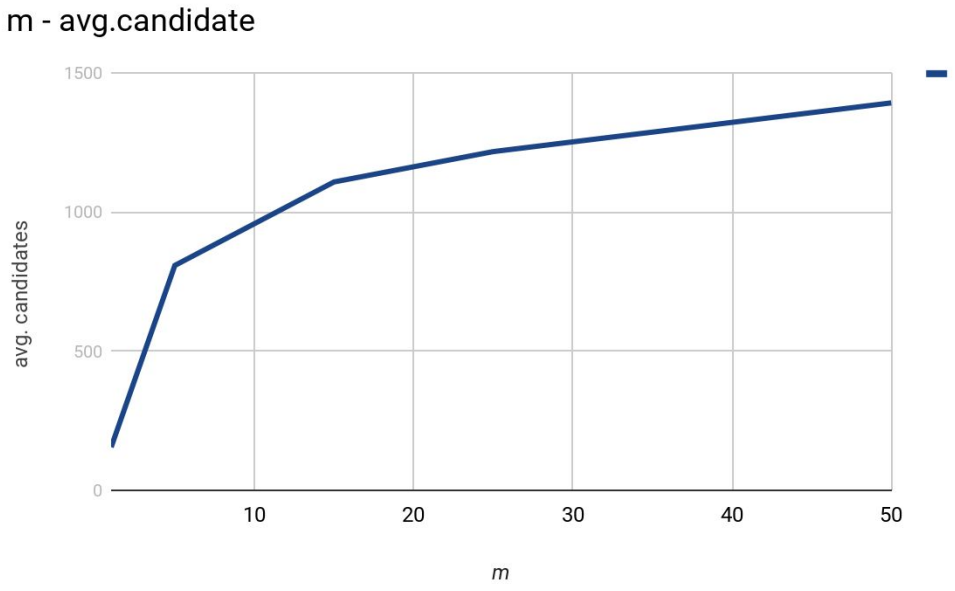




Ahora, de forma similar, fijaremos **k = 1** para el análisis del tiempo de ejecución, y **k = 10** para la media de candidatos, dado que con un **k = 1** y una m no necesariamente grande ya

es suficiente para tener todas las 1500 imágenes como candidatos, por lo que aumentamos **k** a 10 para compensar.





En cuanto al tiempo de ejecución podemos ver que tanto **k** como m afecta linealmente al tiempo de cómputo, cuando mayor sean estos parámetros más tarda el programa en terminar. Desde el código no es difícil verlo, el tiempo de la función hashcode() es proporcional al valor de **k**, y la influencia de m es incluso mayor, marca las repeticiones de varios procesos, como por ejemplo en la búsqueda de candidatos. También por este motivo, que la función de tiempo, aunque en ambos casos son lineales, el incremento es más rápido para **m**.

Por otra parte, en cuanto al número de candidatos, vemos comportamientos diferentes. Para **k**, cuando más funciones de hash usemos, menos probabilidad de colisión tenemos, de este modo obteniendo menos candidatos. Sin embargo, con **m** pasa lo contrario, cuando

más repeticiones le damos al proceso, tenemos más oportunidades de encontrar imágenes con el mismo hash, la probabilidad de colisión incrementa. Todo esto se ve también reflejado en los plots de media de número de candidatos.

Comparación Brute-force y LSH

Velocidad

Ahora procederemos a comparar el rendimiento de local-sensitive hashing y búsqueda a fuerza bruta. Cabe decir que la implementación de las funciones que se pide en el enunciado para esta tarea son relativamente fáciles, por lo que una vez entendido el resto de código, no se debería de encontrar dificultad.

El tiempo de ejecución de brute-force, aunque no lo parece, también se vé afectado por los parámetros **k** y **m**. Esto principalmente se debe a que, siempre inicializamos la instancia lsh, y este depende de **k** y **m** para crear las tablas de hash. Dicho esto, para minimizar el tiempo debido a inicialización, escogeremos **k = 1** y **m = 1**, de modo que brute-force tarda 0.4s en media para encontrar los vecinos más cercanos de 10 imágenes.

Como hemos visto antes, para lsh, **k** y **m** son los que afectan al tiempo de ejecución. Haremos experimentos con diferentes combinaciones de estos.

	tiempo de ejecución(s)
k = 1, m = 1	0.32
k = 5, m = 1	0.13
k = 10, m = 1	0.17
k = 15, m = 1	0.15
k = 5, m = 5	0.6
k = 10, m = 5	0.71
k = 15, m = 5	0.51

Podemos ver que brute-force es más rápido en casos de que m sea mayor que 1, esto es, como hemos visto antes **m** tiene un efecto mayor en el tiempo de ejecución que **k**.
Por otra parte, vemos que fijando **m = 1**, ahora **k = 1** tarda más tiempo que k = 5, recordando que en el apartado anterior vimos que el tiempo es proporcional al valor de, es bastante curioso. Pensamos que este comportamiento se debe a que ahora hacemos una búsqueda en la lista de candidatos para encontrar el más cercano, y cuando la **k** es mayor tenemos menos candidatos, por lo que lógicamente tardaríamos menos en la búsqueda.

Dado que el tamaño de nuestro test set es relativamente pequeño (10), intentaremos experimentar incrementándolo 100. De forma que ahora brute-force tarda 4.7s.

	tiempo de ejecución(s)
k = 1, m = 1	3.69
k = 5, m = 1	2.06
k = 10, m = 1	1.46
k = 15, m = 1	1.25
k = 5, m = 5	4.8
k = 10, m = 5	3.7
k = 15, m = 5	2.39

Cuando el test set es lo suficientemente grande, vemos que lsh es más rápido que brute-force en la mayoría de casos. La diferencia es más notable cuando el set es mayor, ya que uno hace una búsqueda exhaustiva entre todos los 1500 imágenes mientras que el otro solo en la lista de candidatos.

Resultado y Conclusión

Ahora compararemos los resultados de lsh y fuerza bruta.

	k = 1, m = 1	k = 5, m = 1	k = 5, m = 5	k = 5, m = 10
Imagen (índice)	1500	1502		
Porcentage de coincidencia entre lsh y fuerza bruta	90%	70%	100%	100%
Imagen más cercano	1416, 1416	67, 60		
Distancia	52, 52	840, 1429		

Desde estos experimentos podemos ver que los parámetros **k** y **m** no sólo afecta al tiempo de ejecución, sino también a los resultados de la búsqueda.
Sabemos que por fuerza bruta siempre podemos encontrar las soluciones óptimas (vecinos más cercanos), entonces podemos compararlos con lsh y ver cómo nos acercamos al óptimo.
En generar los resultados son muy parecidos, de las 10 imágenes del test set, 9 encuentran los mismos vecinos con ambos métodos. Sin embargo, observamos una tendencia muy

notable, cuando aumentamos el valor de **k**, estamos restringiendo la lista de candidatos, y de este modo tenemos más posibilidad de excluir el óptimo, por lo que cuando **k** es grande solemos obtener resultados diferentes a fuerza bruta.
Por otra parte, **m** al contrario que **k**, como habíamos comentado anteriormente, cuando es grande estamos incluyendo más posibilidades en la lista de candidatos. Y como hacemos un recorrido en toda la lista, tenemos más probabilidad de coincidir con el resultado de la fuerza bruta.
Al final en realidad es un trade-off entre tiempo de ejecución y calidad de solución. Un **k** mayor implica menos tiempo de cómputo pero peor calidad de solución (más diferencia con brute-force), mientras que cuando **m** sea mayor, más tarda en ejecutarse pero con una solución más cercana al óptimo al mismo tiempo.