Comparación de eficiencia y efectividad de algoritmos de indexamiento *KD-tree* y *K-means tree* versus *Linear Scan* sobre búsquedas de vecino más cercano

CC5213-1: Recuperación de Información Multimedia

Profesor de Cátedra: Juan Manuel Barrios

Ayudante: Pablo Polanco Galleguillos

Juan Andrés Moreno
18.641.636-8
Ingeniería Civil en Computación, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile
amorenocb@gmail.com

ABSTRACT

El objetivo de esta tarea fue estudiar la eficiencia y efectividad que tienen los algoritmos de indexamiento *KD-tree*[1] y *K-means tree*[2] al momento de resolver un conjunto de preguntas de vecino más cercano.

Las preocupaciones principales de esta tarea fueron investigar el impacto que tienen la dimensión de los datos y los parámetros asociados a cada algoritmo en el desempeño de estos en comparación con el realizar las mismas búsquedas pero con un algoritmo de fuerza bruta: *Linear Scan*.

Para incrementar la validez de las conclusiones obtenidas se hicieron pruebas con tres descriptores distintos: SIFT[3], MEL[4] y AlexNet[5]. Además, para analizar el impacto de la dimensión en los resultados, se aplicó PCA[6] sobre estos descriptores y se repitieron los experimentos.

En esta misma línea se efectuaron experimentos con distintos valores para los parámetros asociados a cada algoritmo.

Conceptos de CCS

• **Estructuras de datos → Estructuras de datos particionadoras de espacio;** *KD-tree*; • **Procesamiento de señales → Análisis de Clusters;** *K-means* • **Estadística →** Análisis de componentes principales (PCA) **;**

Palabras clave

Estructuras de datos; Indexamiento; Vecino más cercano; *K-means tree*; *KD-tree* ; SIFT ; MEL ; AlexNet ; PCA ; KNN[7]

1. INTRODUCCIÓN

En clases se vio que existen distintas estructuras de datos las cuales pueden resultar útiles para resolver problemas de *k vecinos más cercanos* bastante más rápido que un algoritmo de fuerza bruta. Entre estas se encuentran:

- *KD-tree*
- *K-means tree*

En esta tarea se busca analizar el rendimiento que tienen estas estructuras en comparación a *Linear Scan*, algoritmo de fuerza bruta que resuelve consultas de vecinos más cercanos.

En rendimiento fue medido con dos métricas:

1. **Efectividad**: La fracción de las consultas cuyo vecino más cercano coincide con el encontrado por *Linear Scan.*
2. **Eficiencia**: La fracción del tiempo utilizado en resolver las búsquedas en comparación con el tiempo requerido por *Linear Scan*.

Para cada descriptor (con y sin PCA aplicado) se resolvieron las consultas de vecino más cercano utilizando *Linear Scan*. Luego de esto se utilizaron las estructuras de datos *KD-tree* y *K-means tree* para construir índices y luego resolver las mismas consultas.

Ambos procedimientos de construcción de índices requieren de un parámetro en particular:

1. *Branching* : En el caso del *K-means tree* este parámetro indica en cuantos clusters se divide cada región del espacio. (recursivamente dentro de cada región también)
2. *Trees* : Cantidad de *KD-trees* a construir. Estos serán utilizados paralelamente para encontrar el vecino más cercano.

Por otro lado el procedimiento de búsqueda de un vecino más cercano requiere de otro parámetro: *checks* el cual indica cuantas veces el árbol de indexamiento debe ser recursivamente consultado. De esto último se desprende que en realidad lo que se está haciendo con estas estructuras de datos es resolver búsquedas de vecinos más cercano **aproximadas**.

2. DESARROLLO

2.1 KD-tree

Los *KD-tree* corresponden a arboles binarios balanceados, los cuales en cada nivel separan los vectores (de descriptores en nuestro caso) en dos conjuntos, comparando el valor de una dimensión k con un cierto umbral.

Existen dos tipos de nodos en este árbol:

- Internos: almacenan un numero de dimensión y el umbral aplicado sobre esta para dividir los vectores.
- Externos: almacenan los vectores.

La idea es entonces generar divisiones grandes, es decir divisiones que logren efectivamente dividir en dos grupos grandes los vectores. Para esto el proceso de construcción es el siguiente:

i. Para cada uno de los vectores calcula la varianza para cada una de las dimensiones.
ii. Elige la dimensión de mayor varianza (k).
iii. Divide el conjunto en dos según el valor de la mediana.
iv. Continua dividiendo recursivamente ambos conjuntos.

Se puede ver del proceso de construcción que siempre se elige aquella dimensión con mayor varianza. Esto último no siempre necesariamente es lo óptimo y es lo que justifica la existencia del parámetro de construcción *trees* el cual indica la cantidad de árboles distintos a generar. Estos son distintos pues se eligen distintas dimensiones de manera aleatoria para particionar el espacio.

Luego se utiliza una única APL para buscar en todos los arboles al mismo tiempo.

2.2 K-means tree

Consiste básicamente en utilizar *K-means* para dividir la región en K sub-regiones y luego emplear el mismo proceso recursivamente. El proceso termina cuando quedan menos de K vectores. El producto final es un árbol balanceado k-ario.

Luego para efectuar una búsqueda, se recorre el árbol hasta encontrar el centroide más cercano al punto que está siendo consultado.

El parámetro *branching* del algoritmo de construcción corresponde al parámetro K del proceso *K-means* que se aplica recursivamente sobre las regiones.

Cabe destacar que a diferencia de los *KD-trees*, en esta estructura de datos no se han reportado mejoras significativas en los tiempos de búsqueda al construir múltiples *K-means trees*.

2.3 Experimentos

2.3.1 Archivos de Prueba

Los experimentos y pruebas de esta tarea fueron generados utilizando distintos descriptores: MEL, SIFT y descriptores obtenidos desde una red profunda AlexNet previamente entrenada. Además para cada uno de estos tres descriptores se cuenta con una versión con dimensionalidad reducida producto de aplicar PCA sobre cada uno. Cabe mencionar que si bien los descriptores eran distintos siempre se generaron la misma cantidad: 48000. La tabla a continuación resume las dimensiones de cada descriptor:

Tabla 1. Descriptores usados con cantidad de vectores y sus dimensiones.

	Cantidad de Vectores	**Dimensión**
AlexNet	48000	400
AlexNet - PCA	48000	50
MEL	48000	80
MEL – PCA	48000	50
SIFT	48000	128
SIFT-PCA	48000	50

3. RESULTADOS

Los resultados obtenidos fueron divididos en dos secciones: descriptores con dimensionalidad reducida y descriptores sin dimensionalidad reducida.

Estos se presentan como gráficos de efectividad versus eficiencia. Donde cada curva es una instancia de un índice especifico, es decir cada curva representa la construcción de un índice con un parámetro asociado especifico. A modo de ejemplo, en la figura 1 en el grafico izquierdo, la curva roja muestra seis puntos de efectividad y eficiencia, donde cada punto está definido por un valor para el parámetro *checks* distinto.

Por último, cabe mencionar que el punto óptimo corresponde a efectividad 1 y eficiencia 0.

3.1 Resultados de *Linear Scan*

En la tabla a continuación se muestran los tiempos que le tomo al algoritmo *Linear Scan* en completar todas las consultas en cada archivo de prueba:

Tabla 2: Tiempos de demora de consultas con *Linear Scan*.

Archivo de prueba	Tiempo de demora en completar consultas [s]
AlexNet	360.421
AlexNet-PCA	46.969
MEL	80.820
MEL-PCA	49.325
SIFT	122.317
SIFT-PCA	49.143

Como es de esperarse, a medida que aumenta la dimensión de los vectores los tiempos aumentan. Así mismo, dado que todos los archivos a los cuales se les aplico PCA tienen todos la misma dimensión es esperable que tengan tiempos similares.

3.2 Descriptores regulares

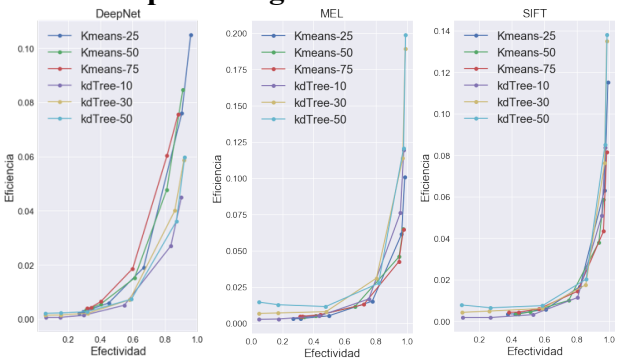

Figura 1: Gráficos de efectividad versus eficiencia para descriptores sin dimensionalidad reducida.

Primero que nada se puede ver en los tres gráficos que las curvas de *K-means tree* parten siempre desde una efectividad mayor a todas las curvas de *KD-trees*. Por otro lado se observa consistentemente que a medida que incrementa el valor del parámetro *checks* **aumenta la efectividad y la eficiencia.** Esto último en una relación exponencial. Salvo en el caso de la curva para *KD-tree* con parámetro *trees* cincuenta, aquí se observa una anomalía en la eficiencia. Esto se le puede atribuir a diferencias de carga del procesador al tomar el tiempo de los procesos.

En segundo lugar se puede ver que para los descriptores MEL y SIFT las curvas de *K-means tree* logran los mejores niveles de eficiencia (recordar que el óptimo es cercano a cero), no así para el caso de los descriptores de AlexNet, en el cual *K-means tree* se desempeña peor que *KD-tree* en todas sus instancias.

En tercer lugar se nota que en todos los descriptores, nuevamente salvo los obtenidos desde el AlexNet, al llegar a los mil *checks* se logran efectividades marginalmente distintas y cercanas a uno. En el caso de AlexNet se ve que para todas las instancias de *K-means tree* se obtienen efectividades distintas.

3.3 Descriptores con PCA aplicado

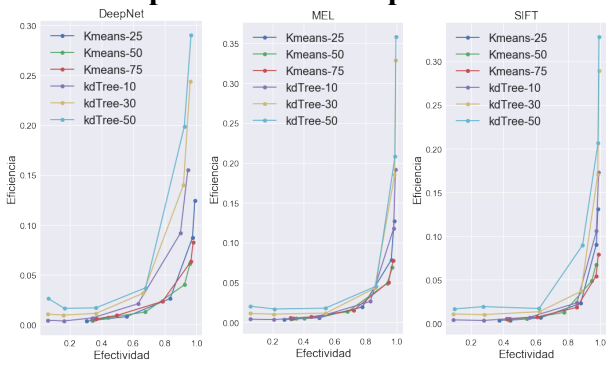

Figura 2: Gráficos de efectividad versus eficiencia para descriptores con PCA aplicado.

En primer lugar y a diferencia de los resultados obtenidos en el punto 3.1 se puede apreciar ahora que los índices *KD-tree* en todas sus instancias son los con peores indicadores de eficiencia. Se nota que a medida que aumenta el parámetro *trees* empeora la eficiencia.

La diferencia se encuentra en los resultados obtenidos para AlexNet. La figura 2 muestra que ahora las instancias de *K-means tree* obtienen niveles de eficiencia superiores (más cercanos a cero) a las de *KD-trees*.

En segundo lugar se nota que se mantiene la tendencia de que las instancias de *K-means tree* parten desde niveles de efectividad y eficiencia más altos en comparación a las instancias de *KD-trees*.

4. CONCLUSIONES

De los gráficos de las secciones 3.2 y 3.3 se puede comprobar que no existe un índice con una mejor performance en todos los casos. Aun así el que mejor rendimiento presento en promedio fue *K-means tree*. Este con parámetro de *branching* igual a setenta y cinco obtiene los mejores resultados, excepto para el caso de los descriptores de AlexNet.

Además se puede ver que a medida que se reduce el parámetro *branching* de *K-means tree* y se aumentan el número de checks aumenta significativamente los tiempos de búsqueda. Esto tiene sentido pues se el espacio se divide en más porciones, lo que se ocasiona que se asemeje el método más a *Linear Scan*. El caso extremo es tomar *branching* igual a 1. Además, dado que se aumentan el número de *checks* se está recorriendo recursivamente más veces una partición más grande del espacio (mayor cantidad de sub-regiones), lo que conlleva a mayores tiempos de ejecución.

Volviendo al primer punto esto lleva a concluir que efectivamente existe una relación entre el índice usado y el descriptor, pues como se vio *K-means tree* obtiene buenos resultados excepto en descriptores de alta dimensionalidad (cuatrocientos en el caso de AlexNet). En el caso de esta tarea, para los descriptores de alta dimensionalidad *KD-tree* obtuvo mejores resultados. Esto último puede deberse al hecho de que este índice apunta justamente a dividir los datos recursivamente por dimensión.

Por otro lado es claro que todas las curvas obtenidas siguen una tendencia exponencial, lo que lleva a concluir que cada vez que se trata de maximizar la efectividad de los índices se paga un alto precio en eficiencia. Aun así los índices presentan puntos de funcionamiento (valores particulares del parámetro *checks*) para los cuales se obtienen buenos rendimientos (relación efectividad, eficiencia). Es importante tener en consideración esto, pues dependiendo del uso que se le da estos índices puede o no ser deseable que las búsquedas sean totalmente efectivas. Puede suceder que se quiera priorizar el que las consultas sean más rápidas a que sean más certeras. Por ejemplo, un sistema de recomendación en tiempo real.

Luego del trabajo realizado las nociones obtenidas fueron, en primer lugar, para descriptores de alta dimensionalidad *KD-trees* tiene un mejor rendimiento y en segundo lugar, no es tan sencillo definir cuáles son los mejores parámetros para la construcción de un índice y cuantas veces se desea consultar este recursivamente (parámetro *checks* de consulta), esto pues existen distintos puntos de operación los cuales se adecuan a distintos escenarios de uso. En algunos será preferible maximizar la eficiencia y en otros la efectividad.

5. REFERENCIAS

[1] : Jon Louis Bentley. 1975. "*Multidimensional binary search trees used for associative searching*". Commun. ACM 18, 9 (Septimebre 1975), 509-517. DOI=http://dx.doi.org/10.1145/361002.361007.

[2] : Christian Böhm, Stefan Berchtold, and Daniel A. Keim. 2001. "*Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases"*. ACM Comput. Surv. 33, 3 (Septiembre 2001), 322-373. DOI=http://dx.doi.org/10.1145/502807.502809

[3] : D.G. Lowe. "*Object recognition from local scale-invariant features*". Computer Vision. The Proceedings of the Seventh IEEE International Conference (1999). DOI= https://doi.org/10.1109/ICCV.1999.790410

[4] : P. Mermelstein. "*Distance measures for speech recognition, psychological and instrumental*". Pattern Recognition and Artificial Intelligence (1976).

[5] : Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever. Disponible en línea en : [http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf] , consultado el 23 de Oct. de 2017.

[6] : Jolliffe I.T. "*Principal Component Analysis, Series: Springer Series in Statistics*", Segunda edición., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4

[7] : Peterson, L. E. "*K-nearest neighbor*". Scholarpedia (2009). Disponible en línea en : [http://www.scholarpedia.org/article/K-nearest_neighbor], consultado el 23 de Oct. de 2017.