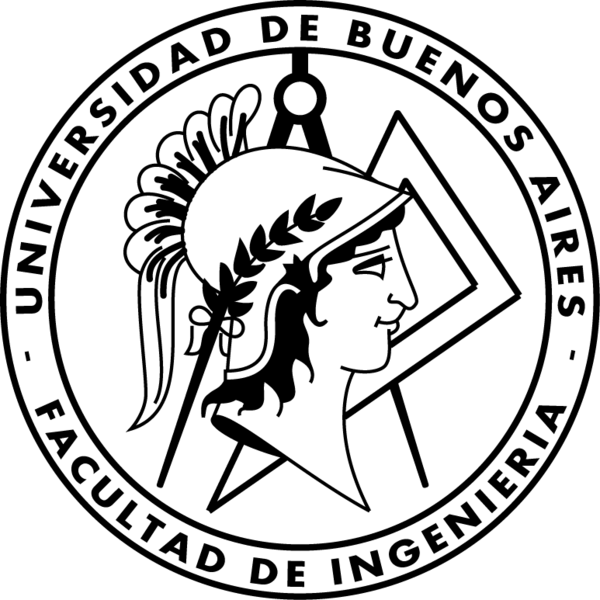
**Organización de Datos [75.06]**

**Clustering y Clasificación de Textos**



**Fecha de entrega:** 30/9/2013

|  |  |  |
| --- | --- | --- |
| **Integrantes** | **Padrón** | **E-Mail** |
| Mateo Bosco | 93488 | mateo.bosco@hotmail.com |
| Germán Fuchs | 94041 | german\_2410@hotmail.com |
| Juan Manuel Hidalgo | 93383 | juanmah\_@hotmail.com |

**Introducción**

Para la resolución del trabajo práctico utilizaremos el algoritmo de clustering llamado K-Means, el cual funciona como se describirá a continuación:

Primero se determina un “k” y un “j” (siendo “k” la cantidad de clusters y “j” la cantidad de documentos líderes), dependiendo de la cantidad de documentos que contenga la colección. El valor de “k” se lo puede asignar por parámetro al comienzo del algoritmo.

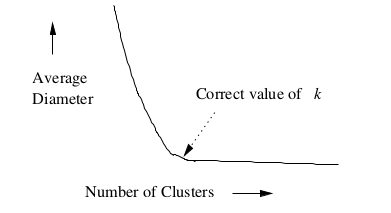
Posteriormente, se elegirán al azar “k” cantidad de documentos que serán los centros de los clusters y se asignará a los demás documentos un cluster dependiendo de qué centro tenga más cercano.

Luego, se reasignarán los centros de los clusters, y se repetirá el paso anterior.

El algoritmo termina cuando después de 2 iteraciones no cambia el centro, no cambian los documentos del cluster o cuando se alcanza un máximo de iteraciones.

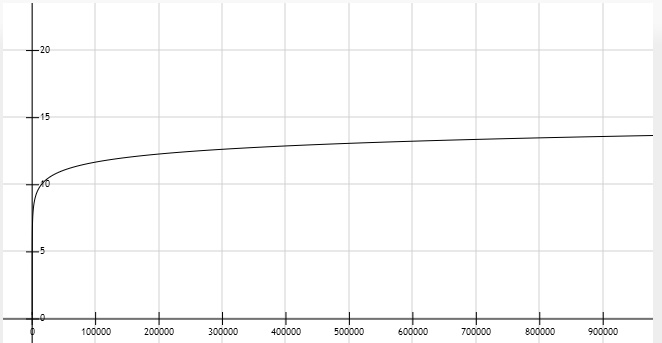
**Elección de k y j**

En el caso de que la k no sea ingresada por parámetro, el programa se encargara de encontrar el valor óptimo de ella. Comenzamos corriendo el algoritmo K-Means con k= 1, 2, 4, 8, 16... y observamos, para cada k, el valor promedio del diámetro de los clusters. Llegara un punto en el cual entre dos valores seguidos de k, que llamaremos 'u' y '2u', no se observara un cambio notorio en el valor promedio del diámetro de los clusters.



Una vez encontrado el valor de 'u', se realizara una búsqueda binaria para hallar el valor óptimo de k. Se analizara diámetro de los clusters con , si no hay un cambio notorio en el promedio de los diámetros de clusters con este valor de k y k=v, se procede a correr el algoritmos nuevamente entre y

La variable “j” será la cantidad de documentos lideres a ser analizados para armar un índice de shingles y de ahí obtener una matriz de incidencias para luego armar un vector de hashmin (se vera luego). Para determinar el valor de “j” utilizaremos la siguiente formula, siendo “n” la cantidad de documentos en la colección . Elegimos esta función de tipo logarítmica ya que a medida la cantidad de textos de la colección es más grande, el valor de “j” no crece tanto. En el caso de que tengamos 10 textos en la colección, seran 30 lideres, y en el caso de 1.000.000 de textos, quedarán 130 lideres. Observamos cómo queda graficada la función:



**Algoritmo K-Means**

Inicialmente se eligen los “j” documentos al azar y de esos se elegirán los “k” centros también al azar. Luego se recorrerán esos “j” textos seleccionados buscando “shingles” para de esta forma crear una matriz. Esta será una matriz de incidencia de “k” columnas, que indicará con 1 y 0 si aparece el shingle en determinado documento centro. Esta matriz estará ordenada alfabéticamente para que la búsqueda y adición de nuevos shingles sea más rápida. Al estar ordenado alfabéticamente, podemos utilizar una búsqueda binaria para encontrar si ya existe el shingle en la matriz. En caso contrario se agrega en forma ordenada.

Una vez terminada la matriz de incidencia se crea la matriz de hashmin con “m” filas, y “k” columnas (siendo “k” la cantidad de documentos centros y “m” la cantidad de funciones de hash). Por el momento elegimos que usaremos 50 funciones de hash, pero al momento de escribir el código y realizar pruebas se puede modificar buscando mejores resultados (se desarrollara este punto mas adelante).

Se inicializa la matriz con todos los valores puestos como “infinito”. A continuación se aplican todas las funciones de hash al número de fila, y se remplaza el valor correspondiente (donde aparece un 1 en la matriz de incidencias), si es menor al valor actual. Esto se repite para cada fila. De esta forma tenemos armada la matriz de hashmin, en la cual tenemos en las columnas los diferentes documentos centros de la colección y en las filas las “m” funciones de hash. Una vez terminada la matriz, tenemos para cada documento centro, un vector hashmin, por lo que procedemos a calcular los documentos que se encuentran cerca a los centros de los clusters. Esto se hará de la siguiente manera: se crea un vector de 'q' elementos, siendo 'q' la cantidad de shingles que se encuentran en la matriz de incidencias. Luego se recorren los documentos restantes para asignarles un cluster. Para eso, se busca si cada shingle del documento se encuentra en los shingles de la matriz de incidencia a través de una búsqueda binaria para cada shingle. En el caso de que se encuentre, se coloca un 1 en la posición del vector que corresponde, caso contrario se pasa a la próxima shingle. Luego se procede a calcular el vector de hashmin del documento de la misma forma que se utilizo para la creación de la matriz de hashmin, y este vector queda asociado al documento para su posterior utilización. A continuación se compara el vector de hashmin del nuevo documento, elemento por elemento, con cada columna de la matriz de hashmin, se calcula la función de Jaccard, a través de hashmin, y se busca cual es el que tiene mayor semejanza. Una vez encontrado el de mayor semejanza, se le asigna al documento el cluster correspondiente a ese centro.

Una vez recorridos todos los textos tenemos a cada documento asociado a un vector de hashmin, por lo tanto aproximar la función de Jaccard entre dos textos será una operación relativamente sencilla.

Luego se busca en cada cluster el nuevo centro, que será el documento que se posea la menor distancia a todos los elementos del cluster. Para buscar estos nuevos centros, se recorren todos los vectores de hashmin del cluster y se arma un nuevo centro “ideal” que tenga asociado un vector de hashmin que en cada posición tendrá el numero que mas aparece en esa posicion para todos los vectores de hasmin de todos los documentos del cluster. Luego se busca el documento mas cercano al centro “ideal” y se lo declarara como nuevo centro del cluster. Posteriormente se armaran nuevamente los clusters buscando los documentos que estén más cerca de los nuevos centros creando una nueva matriz de hashmin. Esta matriz estará conformada por todos los vectores de hashmin de estos nuevos documentos centros y nos permitirá mientras recorremos el resto de los vectores de hashmin de la colección cual es el documento centro que más se parece para agregar ese documento al cluster.

**Funciones de Hash**

Se utilizaran 50 funciones de hash generadas por el programa al momento de la ejecución. Las funciones serán de la forma de . Encontramos que m=O(1/ε2) es un estimador de la cota del error esperado. Si utilizamos 50 funciones de hash el error sera menor o igual que 0,141.

**Largo de los shingles**

El largo de los shingles es recomendable que se encuentre entre 5 y 9 caracteres. En nuestro caso, elegimos que el largo sea de 7 ya que es un valor intermedio de dicho intervalo y así podemos estimar cuanto ocupara almacenar todos los shingles de un documento. Teniendo en cuenta que una hoja tiene en promedio 2000 caracteres y suponiendo que un texto cuenta con 250 hojas, almacenar todos los shingles ocupara 3500 kBytes aproximadamente y este es un valor que se puede manejar en memoria. Hay que tener en mente que vamos a guardar los shingles de todos los documentos centros pero muchos de estos shingles se van a repetir.

**Preprocesamiento de los documentos:**

Para procesar los textos y obtener los shingles, se ira recorriendo cada texto hasta completar un vector de 7 char y así obtener un shingle. Para este trabajo se considerarán a todas las letras como minúsculas. Cuando se encuentre un punto, este no formara parte del shingle que se esta armando sino que el shingle se considerara hasta el punto y otro nuevo a partir del punto. Este caso ocurrirá también en signos como '?', '!' , ':', '(', ')', '”', '-'. En el caso de ',' y ';' estos signos se ignoraran por completo. Por ejemplo, en el texto “Hola, como andas? Mi nombre es Pedro.” los shingles obtenidos serán: 'hola co' 'ola com' 'la como' 'a como ' ' como an' 'omo and' 'mo anda' 'o andas' 'mi nomb' 'i nombr' ' nombre' 'nombre e' 'ombre es' 'mbre es ' 'bre es ' 're es p' 'e es pe' ' es pedr' 's pedro'. Como vemos los shingles que quedan de tamaño menor a 7 son descartados.

**Criterios de corte:**

El algoritmo de K Means se detiene cuando:

* Luego de dos iteraciones no cambia ninguno de los centros de los clusters
* Luego de dos iteraciones no cambia ninguno de los integrantes de cada cluster.
* Se pasa un máximo de iteraciones, que sera por defecto de 1000 y se podrá elegir por parámetro un valor entre 1 y 1000.

**Almacenamiento de las estructuras**

Se almacenaran todos los nombres de los documentos en un archivo relativo a medida que se van recorriendo para la lectura y extracción de los shingles. De esta forma se relaciona un numero de registro con el nombre del archivo y a partir de allí los archivos se conocerán por numero de registro y no por nombre de archivo. Usando este tipo de archivo sera muy rápido, teniendo en numero de registro, obtener el nombre del archivo

Teniendo en cuenta los cálculos realizados, vamos a suponer todos los shingles y que la matriz de hashimin de los centros de los clusters pueden estar en memoria sin necesidad de ser comprimidos. Teniéndolos en memoria se realizaran los cálculos con una mayor rapidez.

Para almacenar los vectores de hashmin de todos los documentos también utilizaremos un archivo relativo, haciendo coincidir el numero de registro del vector de hashmin con el numero de registro del relativo de los nombres de archivo. Esta organización nos permite recorrer secuencialmente para la primera asignación de clusters.

**Bibliografía:**

* Editorial Dunken: <http://www.dunken.com.ar/web2/calcular_paginas.php?sector=1>
* Universidad de Murcia: <http://www.um.es/lacell/aelinco/contenido/pdf/51.pdf>
* Wikipedia: <http://en.wikipedia.org/wiki/MinHash>
* Apuntes de la materia: Minhash-LSH-apunte-datos.pdf y D008\_Clasificacion y Clustering.pdf
* Mining Massive Datasets