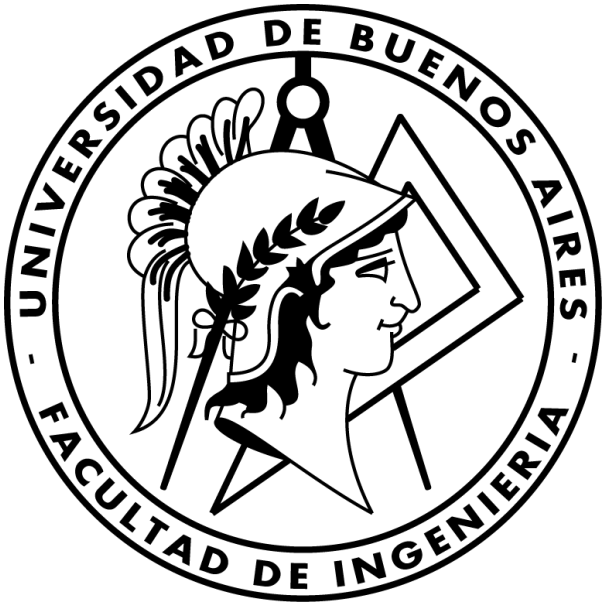

UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2013 - 2do Cuatrimestre

ORGANIZACIÓN DE DATOS (75.06)

TEMA: Clustering y Clasificación de Textos

FECHA: 30/09/2013

INTEGRANTES:

Martin Debora – 90934 <debbie1new.world@gmail.com>
Martinez Gaston – 91383 <gaston.martinez.90@gmail.com>
Vázquez Matías – 91523 <mfvazquezfiuba@gmail.com>

1. Introducción

Se requiere implementar un programa de consola que realizará automáticamente la división de un set de documentos de texto en grupos (clusters). Funcionará con una colección grande de documentos de texto en idioma inglés los cuales deberán ser repartidos automáticamente en "n" categorías o grupos. Una vez realizada esta división el programa deberá poder clasificar un documento nuevo dentro de las categorías creadas.

2. Especificaciones

El programa contará con las siguientes funcionalidades funcionalidades:

```
./Tp -d <path> -c <cantidad de categorías> -o <Y/N>
```

-d indica el path a donde están almacenados los documentos.
-c indica la cantidad de categorías a crear, si no se indica el programa debe decidir automáticamente cual es el número de clusters a generar de acuerdo a su entendimiento de los datos.
-o indica si un documento puede estar en más de un grupo.

```
./Tp -l
```

Lista todos los documentos del repositorio y la categoría a la cual pertenece cada uno.

```
./Tp -g
```

Lista los grupos o categorías existentes y los documentos dentro de cada grupo o categoría

```
./Tp -a <path del texto>
```

Agrega y clasifica el texto pasado como parámetro e indica a qué grupo lo ha agregado (a partir de aquí debería aparecer al listar con -l o -g)

3. Diseño

Como solución se propuso la utilización del algoritmo de clustering k-means utilizando distancia coseno y pesos TF-IDF.
A continuación se tratará en detalle la secuencia del programa:

3.1. Estructuras de datos

A continuación se detallan las estructuras que se requieren para la resolución del trabajo, indicando su uso y el porqué de su elección.

--Hash: Se utiliza un hash para contar las apariciones de las diferentes secuencias dentro de cada archivo. Si se pensara utilizar una Lista, las operaciones serían N^2, con N el número de

secuencias diferentes dentro de cada archivo (si un archivo posee solo 1 aparición de cada secuencia las operaciones serían muy costosas)

3.2. Algoritmo de vectorización

3.2.1 Parseo de Archivo

El primer problema que surge a la hora de construir los vectores es cuáles serán las ***construcciones*** que serán utilizadas para las coordenadas del mismo:

3.2.1.1 Construcciones válidas

Se identifican como secuencias válidas, a toda sucesión de caracteres (alfabeticos en mayuscula o minuscula por igual) de largo mayor a dos. Las secuencias se procesan de forma *CASE-SENSITIVE* de manera de agrupar términos iguales. Por ejemplo:

"*In figure 7.2, you can see that only the point (2, 1) was at an angle greater than 45˚ from the x-axis*"

Se extrae la lista de secuencias:

[*in , figure , you , can , see , that , only , the , point , was , angle , greater , than , from , the , axis*]

3.2.1.2 Consideraciones para casos excluidos:

*Palabras compuestas: el caso de "o'reilly" o "x-axis" se procesan como: "reilly" y "axis" . Esta eliminación de términos igual mantiene cierta información de la construcción original, y no agrega construcciones que pueden generar ruido.
*Palabras menores de 3 caracteres: Las palabras de este estilo son muy comunes (en todos los textos en general), y si bien se utiliza un algoritmo que evita que estas tengan un peso importante(tf-idf), se pueden omitir para el cómputo.
*Palabras con símbolos no pertenecientes al idioma inglés: Utilizar este criterio sobre textos en otro idioma (español o portugués por ejemplo) generaría que las palabras se separen de manera inesperada por tildes o diéresis. Se Construyó todo considerando aplicaciones únicamente sobre el idioma inglés.

3.2.2 Optimizaciones posibles

Utilizando algoritmos de *stemming* se puede reducir el vector de datos agrupando palabras relacionadas bajo un mismo término. Si se utiliza el algoritmo de Poter, (y aplicando una limpieza de los términos que aparecen en un unico documento) se logró reducir el numero de terminos totales de un conjunto de datos al 20%, lo cual es una considerable reducción, pero aun asi no suficiente.

3.2.3 Algoritmo para obtener los datos TF-IDF

Nota: Nos referiremos a los términos TF como un hash con la frecuencia de los términos en un archivo, y DF como un hash con la frecuencia de aparición en archivos de cada término.

En principio para poder aplicar TF-IDF (Term frequency–inverse document frequency) se requeriría recorrer 2 veces cada archivo, una para obtener TF y otra para DF. Ahora, si bien TF se puede obtener en una pasada, es información insuficiente para armar el vector (no se tienen todas la palabras). Por otro lado, si primero se obtiene DF, en el siguiente paso se puede armar el vector, pero implica recorrer 2 veces cada archivo.

Si se aplica la consideración, de que cada archivo no tiene una de cada palabra (es decir, que el 90% de las palabras válidas [más de 2 caracteres] tienen al menos una repetición) se puede construir DF en simultáneo con la creación de los TF para cada archivo. Mientras se obtiene TF por cada archivo, se actualiza automáticamente la cantidad de archivos en las que aparece cada palabra(1). Cada vez que se completa un TF, este se guarda en una base de datos en disco (la información a almacenar se espera sea considerablemente menor que la del archivo original) y se pasa al siguiente archivo. Para luego vectorizar, se lleva a memoria cada base de datos y combinada con la información del DF, se obtiene el valor de cada coordenada i del vector utilizando:

$$W_i = TF_i \ log\left(\frac{M}{DF_i}\right)$$

Siendo M el número de términos totales en todos los archivos.

Construir DF por cada archivo, sin optimizaciones, requiere recorrer cada archivo en búsqueda de palabras válidas, con lo que el parseo requiere esfuerzo mayor que N (si N es el número de chars dentro del archivo). Si las operaciones sobre la estructura auxiliar son O(1), construir DF requiere KN operaciones, con K una constante de chequeos por cada carácter.

Construir TF requiere un número similar de operaciones que para DF, con diferencia de otra constante,T. Combinando la cantidad de operaciones totales, siendo A el número de archivos y N el número de caracteres promedio, nos queda:

AKN + ATN = AN (K+T)

Obteniendo DF y TF en la misma iteración:

Construir DF y TF, requiere igual que antes KN operaciones, más un algoritmo de orden P (palabras diferentes) para almacenar el TF en cada cambio de archivo. Construir el vector, requiere un algoritmo de orden P para leer cada base de datos y construir con ella el vector. Queda entonces (si P es el número de palabras promedio):

AKN + AP + AP
AKN + 2AP

Comparando AKN + 2AP con ANK + ANT queda que la diferencia radica en que 2P es mucho menor que NT, porque P es mucho menor que N.

Algoritmo (pseudocódigo):

```
Por cada documento:
     Por cada palabra en el documento:
          Se suma 1 al contador de palabras del documento
          Se suma 1 a la palabra en el TF
          Si la palabra no esta marcada en DF y la ultima
```

```
                vez que se agregó no fué desde el documento
                actual, se le suma 1
          Se guarda una base de datos ordenada en disco con TF
          e información relevante necesaria como el contador de
          palabras del documento

>>APLICAR REDUCCIONES PARA NO TENER UN ESPACIO INFINITO<<

Por cada Base de datos en disco:
     Por cada palabra
          Se calcula el peso con la información del DF
     Se genera el vector para ese archivo
```

Base de datos en disco:

Tiene que ser un archivo que pueda mantener una organización del tipo: <dato>|<valor>. Como es necesario leer todos, en principio es más fácil leer un archivo secuencial y no agregar dificultades leyendo bases de datos que son más "óptimas" en disco pero más difíciles de implementar. Hacer una medición de tiempos influye mucho en esta decisión, pero en principio es más importante optimizar la generación de clusters y las búsquedas/indexación.

MEDICION 1: medidas, las bases que se hacen en disco ocupan en promedio el 10% del tamaño del conjunto de datos original. Si se usa un archivo secuencial de la forma <clave>'='<cantidad>'\n' con '=' como separador de datos y '\n' como separador de campos, se llega a ese porcentaje de uso de disco.

MEDICION 2: nuevas mediciones utilizando un algoritmo de stemming, el uso del disco se redujo al 6% del tamaño del conjunto de datos original.

3.3. Medición de distancia

Una vez obtenidos todos los vectores, es necesario definir una distancia entre los vectores para ser utilizada por el algoritmo de clustering. A continuación explicaremos las distancias a tratar:

No utilizaremos un espacio euclidiano ya que para textos se obtienen mejores resultados utilizando la distancia coseno.

3.3.1. Distancia coseno

La distancia coseno es una medida de distancia entre dos vectores de n dimensiones calculando el coseno del ángulo entre ellos, utilizado para la comparación de documentos de texto.

Dados dos vectores de dimensión n, A y B, la distancia coseno es obtenida con la siguiente fórmula:

$$d = 1 - \frac{A\,B}{||A||\,||B||}$$

Notar que esta distancia no tiene en cuenta el largo de los dos vectores, solo importa que los puntos están en la misma dirección desde el origen.

La distancia coseno es mejor para textos, porque los agrupa por las palabras claves que tengan en

común. Los pesos TF-IDF de los vectores tienen pesos altos para las palabras claves; así los documentos similares agrupados, usando la distancia coseno, tienen en común palabras claves entre ellos. Esto causa que el vector del centroide del cluster tenga un alto promedio de peso para palabras clave que para palabras "vacías".

3.3.2. Distancia Tanimoto

Otra medición de distancia que funciona bien para textos es la distancia Tanimoto. La distancia Tanimoto funciona de manera parecida a la distancia coseno, pero este tiene en cuenta la longitud de los vectores. La fórmula en este caso sería:

$$d = 1 - \frac{A\,B}{||A|| + ||B|| - A\,B}$$

La razón de tener en cuenta la distancia Tanimoto, es que la distancia coseno puede perder información relevante de los vectores. Por ejemplo, consideremos los vectores en 2D: A (1,1) B (7,7) y C (8,8). Los 3 vectores apuntan en la misma dirección, por lo que su distancia coseno es 0. El problema con esto es que de cierta manera, se pierde la información de que B y C son más cercanos entre sí que con A.

A priori, no hay indicios de que implementar esta distancia produzca resultados significativos respecto de la utilización de la distancia coseno. Es necesario entonces, probarlo una vez que el sistema esté implementado, por lo que se trabajará con la distancia coseno y luego se harán mediciones de calidad de clusters con el cambio de distancia.

3.4. Algoritmo de clustering

3.4.1 Posibles Algoritmos

*** K-means:** Fácil de implementar. Requiere de antemano un número definido de clusters a generar (saber cuantos es una tarea difícil). Los clusters son de tamaño similar lo cual no es deseable. Cada elemento pertenece simplemente a un solo cluster.

Algoritmo:

1) Establecer K - Elegir el número de clusters deseados, K.
2) Inicialización - Elegir los k puntos iniciales que serán utilizados como estimación inicial de los centroides de los clusters. Son tomados como valores iniciales de partida.
3) Clasificación - Examinar cada punto en el conjunto de datos y asignarlo al cluster cuyo centroide es más cercano.
4) Cálculo del centroide - Cuando cada punto en el conjunto de datos es asignado a un cluster, es necesario recalcular
los nuevos k centroides.
5) Criterio de convergencia - Los pasos 3) y 4) requieren ser repetidos hasta que ningún punto cambie de cluster asignado o hasta que el centroide no cambie de posición.

Orden de complejidad:

El orden es de $O(n\,k\,i\,d)$, siendo n el número de vectores, k el número de clusters, i el número de

iteraciones y *d* el número de atributos.

*** Fuzzy K-means**: Similar al K-means pero permite solapamiento de clusters.

Algoritmo:

1) Calcular el vector del centroide $\{V_{ij}\}$ para cada paso.

$$v_{ij} = \frac{\sum_{k=1}^{n} (u_{ik})^m x_{kj}}{\sum_{k=1}^{n} (u_{ij})^m}$$

2) Calcular la matriz distancia $D_{[c,n]}$

$$D_{ij} = \left(\sum_{j=1}^{m} (x_{kj} - v_{ij})^2 \right)^{1/2}$$

3) Actualizar la matriz partición para el paso r, U^R como

$$\mu_{ij}^{r-1} = \left(1 / \sum_{j=1}^{c} \left(\frac{d_{ik}^{\,r}}{d_{jk}^{\,r}} \right)^{\frac{2}{m-1}} \right)$$

Orden de complejidad:

El orden es de $O(n\,k^2 i\,d)$, siendo *n* el número de vectores, *k* el número de clusters, *i* el número de iteraciones y *d* el número de atributos.

*** DBSCAN**: No requiere un número fijo de clusters para iniciar. Agrupa según densidad y puede ser modificado para permitir solapamiento de clusters (elementos pueden corresponder a múltiples clusters de manera simultánea). No obstante, es un algoritmo de orden cuadrático en ejecución y en memoria, por lo que no es considerado como una opción viable en un primer análisis.

Usando un modelo probabilístico, se podría tomar un espacio muestral reducido, considerar que este representa al espacio completo y aplicar a este el algoritmo. Una vez formados los clusters se categorizan el resto de los documentos (de manera similar que al agregar un nuevo documento). El problema con esto es que ya los clusters tienen una probabilidad de ser buenos y no necesariamente reflejen la realidad.

3.4.2 Almacenamiento de la información del programa

*** Vectores (Temporales durante la ejecución):**

Salvo reducciones muy groseras de los vectores de datos, es claro que es imposible mantener todos los vectores en memoria. Luego de generar cada vector o grupo de vectores, se los tiene que volcar a disco para luego poder procesarlos con el algoritmo de clustering. Los archivos nuevamente, se pueden formar de manera secuencial. La forma de los mismos sería una concatenación de X cifras de 4 bytes

(dado el uso de TF-DF, serán flotantes de 4 bytes <flotantes IEEE 775 de 32 bits>), sin separadores porque cada campo tiene un ancho fijo. X es la cantidad de dimensiones de cada vector.

*** Índices de clusters:**

Es necesario para optimizar las consultas, se requiere de cierta información que debe ser almacenada luego de la generación de clusters. Esta es:

***índice documento-cluster**:

Archivo indexado que utilice como como claves los documentos y como valores, el número del cluster al cual pertenece.

<Documento>	#Cluster

***índice de clusters** : cluster, centroide, cantidad de elementos.

Archivo directo que tome el número de cluster como clave y cuyos datos sean el centroide de dicho cluster, y la cantidad de elementos contenida en el mismo. El número de clusters es fijo, entonces no hay problemas de inserción y al ser consecutivos no quedan registros vacíos. El valor de cada centroide tiene un tamaño fijo (definido durante la etapa de vectorización). Cada coordenada tiene un tamaño fijo. El número de elementos tiene longitud fija: Inicialmente lo suficiente para almacenar el número de documentos totales, y en caso de no poder representar un número más grande se regenera el indice. Se supone que esta última operación es poco común. Los registros quedan entonces:

#Cluster	V_0	...	V_n	#elementos

*í**ndice de cada cluster**: listado de la información del cluster, en particular los documentos pertenecientes a este.

Archivo secuencial. Guarda los nombres de los documentos del cluster. Como los nombres tienen longitud variable, se utilizará como separador de campos el carácter de fin de linea. Los campos serán entonces de la forma:

<documento>	‘\n’

***información**: Cantidad de documentos totales, vector con el que se hizo el algoritmo, restricciones de pertenencia a clusters..

Archivo secuencial que contiene los datos, guardando la cantidad de documentos al final, de modo que pueda modificarse su longitud sin necesidad de alterar el estado de los demás datos.

El esquema del archivo es el siguiente:

V_0	...	V_n

Multi cluster	¿otras opciones?

Cantidad Clusters	EOF

3.4.3 Conclusión

Utilizaremos el algoritmo de clustering fuzzy k-means con modificaciones de no ser requerido el solapamiento entre clusters. La principal ventaja respecto a k-means es la posibilidad del solapamiento de clusters. Además de que no requiere tener en memoria todos los vectores, ni de compararlos con todos los que están en disco.

3.5 Calidad de clusters

Para obtener clusters de buena calidad, se deberán tener en cuenta dos aspectos fundamentales:

- La medida usada para la distancia. En particular, la distancia coseno agrupa los documentos de acuerdo a la similitud en las palabras con mayor peso y permite normalizar la longitud del documento para que no influya en la similitud.

- Los elementos tomados en cuenta para la medición. En este caso, el peso otorgado a cada palabra debe ser consecuente con lo que se desea lograr, para lo cual debe tenerse en cuenta la importancia relativa de las mismas. TF-IDF cumple con esto.

Se puede evaluar la calidad de los clusters con respecto a conocimiento obtenido previamente o no. En este caso, no se tiene tal información, por lo que se comparará la calidad de acuerdo a las similitudes generales entre los documentos. Par estimar, se puede considerar:

- **Distancia entre clusters distintos (Inter-cluster distance).** La distancia entre sus centros es un parámetro de la similitud entre ambos. Si la distancia es chica, el proceso está creando clusters distintos con elementos cuyas características son similares. Si la máxima distancia entre dos clusters no es muy diferente en magnitud a la mínima distancia, se están creando clusters uniformemente distribuidos en el espacio. Si la mínima es muy pequeña, hay por lo menos dos clusters que se están superponiendo casi completamente.

- **Distancia entre los miembros de un mismo cluster (Intra-cluster distance).** La magnitud de esta distancia debe ser muy pequeña comparada con la distancia entre clusters. El resultado será bueno si tiene clusters cuyos elementos estén muy juntos entre sí, pero alejados del resto de los clusters. Siendo

c el centroide del cluster S, esta distancia se podría calcular como:

$$\frac{1}{|S|^2} \sum_{d \varepsilon S,\, d' \varepsilon S} cos(d',\ d) = ||c||^2$$

- Similitud entre los miembros de un cluster (Intra-cluster similarity technique). Este método es similar al anterior. Calcula la cercanía entre todos los documentos de un cluster con su centroide. La elección de qué par de clusters se deberán unir, se realiza de acuerdo a cual provocará la menor disminución en la cercanía. Se define como:

$$Sim(X) = \sum_{d \varepsilon X} cos(d,\ c)$$

Donde d es un documento del cluster X y c es el centroide del mismo. Por lo tanto, si Z es el cluster formado al unir los clusters X e Y, se desea maximizar:

$$Sim(Z) - (Sim(X) + Sim(Y))$$

Para grandes cantidades de datos a analizar, la forma más práctica de medir la calidad de los clusters es con la distancia entre clusters distintos, y entre los miembros de cada uno. Estos son criterios de evaluación internos, y puede que no devuelvan siempre buenos resultados. Por ejemplo, cuando hay solapamiento de clusters la distancia entre clusters no es un buen parámetro ya que esta es pequeña. En un caso común esto indicaría mala calidad de clusters, pero aquí puede deberse a solo un par de clusters que se solapan. Sin embargo, la distancia entre miembros de un cluster, y su similitud siguen siendo métodos factibles de ser utilizados.

Otra forma de evaluar la calidad es de forma externa, para un caso particular. El problema con estos métodos es que son muy costosos y se realizan sobre los clusters obtenidos como resultado de la operación. Además, necesitan conocimientos previos sobre el funcionamiento de los métodos con determinadas clases.

3.6. Cantidad de clusters

En el caso de no que no se sepa de antemano la cantidad de clusters en que se deben clasificar los datos, se puede usar un algoritmo de clustering precio a k-means.

Canopy cumple con las condiciones que se necesitan ya que puede crear clusters de manera extremadamente rápida, con un solo recorrido sobre los datos. Este algoritmo no devuelve buenos resultados, pero sí puede obtener la cantidad óptima de clusters sin saberlo de antemano, como k-means.

Para comenzar requiere una lista de puntos a analizar y dos distancias, $T1 > T2$, para realizar el procesamiento.

Algoritmo:

1) Selecciona un punto de la lista de manera aleatoria para que sea un centroide del subconjunto, o canopy.
2) Calcula todas las distancias desde el centroide hasta cada punto de la lista.
3) Pone todos los puntos cuya distancia es menor que T1 en el mismo subconjunto.
4) Quita de la lista original todos los puntos cuya distancia sea menor que T2. Tales puntos quedan excluidos de ser los centroides o de formar parte de nuevos subconjuntos.
5) Repite desde el paso 1 al 4 hasta que la lista original esté vacía.

3.7. Agregar un nuevo documento

Cuando se desee clasificar un nuevo documento con posterioridad al armado de los clusters, se procederá con el mismo de manera similar a lo mencionado para el caso general. Se procesa el documento y se crea el vector correspondiente. Se tomará en cuenta para ello el modelo original de vector, que se utilizó para formar los clusters. De este modo, las nuevas palabras que hubiere en este documento no influirá en su categorización.

Una vez que se tiene el vector, deberá ser calculada su distancia hasta los centroides de cada cluster, y se lo asociará con el cluster al que corresponda la distancia menor. Puede ser más de uno si existe superposición. Por último, se deberán actualizar los índices y la información en el disco.

- Bibliografía consultada

- Wikipedia: Determining the number of clusters in a data set (http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set)
- Owen, Anil, Dunning, Friedman: Mahout in Action (2011)
- Curso de Natural Languaje Processing Standford (http://www.youtube.com/watch?v=ZEkO8QSlynY)
- Manning, Raghavan, Schütze: Introduction to information retrieval (http://nlp.stanford.edu/IR-book/html/htmledition/vector-space-classification-1.html)
- Porter: The Porter stemming algorith (http://tartarus.org/martin/PorterStemmer/)
- Satya sree , Murthy: Clustering based on cosine similarity measure (http://ijesat.org/Volumes/2012_Vol_02_Iss_03/IJESAT_2012_02_03_21.pdf)
- Rajaraman, Leskovec, Ullman: Minning of massive datasets.
- Steinbach, Karypis, Kumar: A comparison of common document cluster techniques (http://glaros.dtc.umn.edu/gkhome/fetch/papers/docclusterKDDTMW00.pdf)
- Ghosh, Dubey: Comparative Analysis of K-Means and Fuzzy C-Means Algorithms (http://thesai.org/Downloads/Volume4No4/Paper_6-Comparative_Analysis_of_K-Means_and_Fuzzy_C_Means_Algorithms.pdf)