Problem Set 3: Inverted Index and Query Processor

Renae Fisher

CS 4903: Text Mining – Professor Mackey

July 27, 2019

Methodology

Text Normalization

The UAInvertedIndex class transforms any text that it will read or write. But, it expects the tokens to be lowercase and devoid of most punctuation. Its text cleaning method replaces all non-ASCII characters with a single-byte character and returns a trimmed String. It's necessary to replace such characters to ensure that each record will have a fixed length. The UAQuery class performs the same steps as UAInvertedIndex, except it uses additional techniques from the pre-processing phase. Its method `convertText` removes some punctuation, and it uses other forms of punction as a delimiter. The method also reduces the case of the word, performs stemming, and trims the word to a specific length.

BuildInvertedIndex

Time complexity	$O(DT \log V_D) + O(D \log D) + O(T \log^2 D) + O(V)$
Space complexity	$O(V_D) + O(D) + O(logD) + O(V)$

The BuildInvertedIndex method first makes a call to algoOne. This method uses a loop to process a list of files in an input directory. At each iteration, algoOne uses a loop and a `TreeMap` to count the frequencies of tokens in a document. The two loops take $O(DT)$ time total. In this case, D and T refer to the documents and tokens in the input directory. The `TreeMap` is comprised of a red-black tree and it maintains order for all items in the tree. It takes $log(V_D)$ time, where V_D represents the distinct tokens in a document, to retrieve and add items to the map. Altogether, it takes ALGOONE exactly $O(DT \log V_D)$ time to count the frequencies of tokens and arrange them in sorted order. After the inner loop expires, ALGOONE calls WRITETEMPFILE. This method uses a loop to write the distinct tokens in `TreeMap` to separate files. This adds an insignificant cost of $O(V_D)$.

Next, BUILDINVERTEDINDEX calls the method MERGESORT, which uses an iterative sort to combine most of the temporary files. It first creates an array to hold all temporary files, then it uses two loops to merge the contents of those files. The outer loop divides the file array into segments. Its control variable increases by a multiple of two after each iteration, which approximates a logarithmic run time. The code in the inner loop, including the call to MERGE, processes a segment delineated by the outer loop in linear time. Altogether, the method MERGESORT takes $O(D \log D)$ time and $O(D)$ space to merge the files in the temporary directory.

Last, the method BUILDINVERTEDINDEX uses ALGOTWO to construct the `post.raf` and `dict.raf` files. First, ALGOTWO opens the directory of merged files, which have been reduced to $O(\log D)$ space by MERGESORT. Next, the method uses two loops to determine the appropriate order of the tokens as it writes them to `post.raf`. The outer loop runs until there are no empty `BufferedReaders`, and there are $\log D$ total readers. The inner loop examines each reader as it finds the token alphabetically first in the buffer. Together, the two loops take $O(T \log^2 D)$ time to create the postings file from the directory of merged files. Last, ALGOTWO uses a loop to write the global hash table to the hard drive as `dict.raf`. This process takes $O(V)$ time and space, where V represents the distinct terms between all documents.

RUNQUERY

Time complexity	$O(QD_TT) + O(V_TD_T)$
Space complexity	$O(V_T + V_D + V_Q) + O(V_TV_D) + O(\log V_D)$

The method RUNQUERY uses two methods to build a term-document matrix. First, it creates three hash tables that each hold the number of distinct terms, documents, or words in the query. The `termMap` hash table takes V_T space, the `docMap` hash table takes V_D space, and the `q` hash table takes V_Q space. Next, `runQuery` calls MAPROWSCOLS, which maps distinct terms and document IDs to the rows and columns of a matrix. The method MAPROWSCOLS first iterates over the words in the query. It locates each word in the dictionary file, which would take constant time at best. If the word exists, the method looks for more information in the postings file. It takes D_T time, the number of documents with term T, to examine each posting. For each posting, MAPROWSCOLS opens the document and adds each term in it to the `termMap`. Altogether, MAPROWSCOLS takes $O(QD_TT)$ time and $O(V_T + V_D + V_Q)$ space total.

Once the method `runQuery` is finished, RUNQUERY makes a call to `buildTDM`. This method creates a term-document matrix that is V_T rows by V_D columns, which corresponds to the number of items in $termMap$ and $docMap$. Then, `buildTDM` iterates over the items in $termMap$. Like MAPROWSCOLS, the method retrieves information from the dictionary file and postings. But, it stores the rtf-idf values of a term and document in the term-document matrix. Overall, it takes `buildTDM` $O(V_TD_T)$ time and $O(V_TV_D)$ space to construct the term-document matrix.

After the term-document matrix is complete, the method RUNQUERY uses `getDocs` to find documents that match the query vector. It iterates over the `docMap` and uses its information to calculate the cosine similarity between each document and the query vector. It takes V_D time to iterate over `docMap`, and V_T time to calculate cosine similarity between two vectors. The method stores each result in a priority queue, whose add operation takes logarithmic time. Altogether, the method `getDocs` takes $O(V_DV_T)$ time and uses $O(\log V_D)$ space to find the k documents that are most similar to a query.

Results

The following pages demonstrate the performance of the BuildInvertedIndex and RunQuery methods against a portion of tokenized data from the Common Crawl data set. Each section shows the run-time of the application as it performs an indexing or querying task. The RunQuery section shows a sample of the top ten documents that the search-engine returned for each query.

BuildInvertedIndex

real	4m37.982s
user	3m38.643s
sys	1m7.179s

RunQuery

`Computer`

#22963	022963uafs.html.out	0.5634
#13962	013962uafs.html.out	0.4854
#34826	034826uafs.html.out	0.4805
#33335	033335uafs.html.out	0.4293
#39357	039357uafs.html.out	0.3856
#26145	026145uafs.html.out	0.3856
#26146	026146uafs.html.out	0.3856
#21331	021331uafs.html.out	0.3827
#15852	015852uafs.html.out	0.3807
#36639	036639uafs.html.out	0.3260

real	0m37.702s
user	0m22.608s
sys	0m17.162s

Big Data

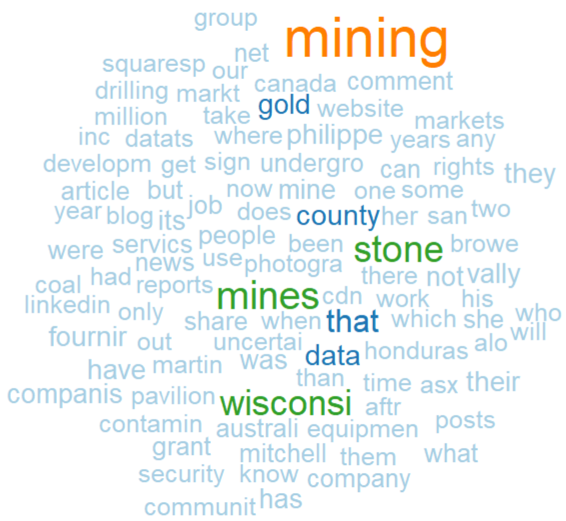
#23529	023529uafs.html.out	0.6231639
#42494	042494uafs.html.out	0.6147945
#42495	042495uafs.html.out	0.6147945
#438	000438uafs.html.out	0.5418439
#1794	001794uafs.html.out	0.5348657
#43510	043510uafs.html.out	0.4739626
#35849	035849uafs.html.out	0.43092135
#1795	001795uafs.html.out	0.42124513
#7360	007360uafs.html.out	0.4105744
#42770	042770uafs.html.out	0.39694285

real	0m48.793s
user	0m32.625s
sys	0m20.384s

Big Data and Data Analytics

#1794	001794uafs.html.out	0.7212369
#42494	042494uafs.html.out	0.66177726
#42495	042495uafs.html.out	0.66177726
#34892	034892uafs.html.out	0.5747477
#438	000438uafs.html.out	0.5336642
#1795	001795uafs.html.out	0.52219814
#43510	043510uafs.html.out	0.510183
#23529	023529uafs.html.out	0.48249996
#35849	035849uafs.html.out	0.46385247
#35976	035976uafs.html.out	0.43618143

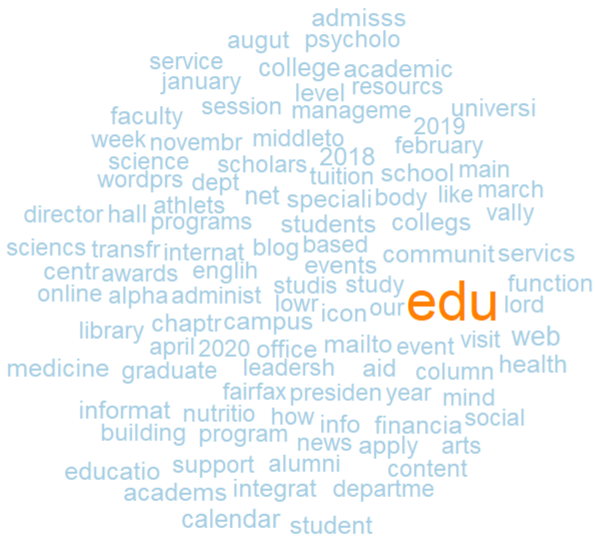
real	0m50.788s
user	0m34.912s
sys	0m20.473s



#17094	017094uafs.html.out	0.84758025
#17093	017093uafs.html.out	0.768427
#27844	027844uafs.html.out	0.7282029
#2745	002745uafs.html.out	0.52677673
#1005	001005uafs.html.out	0.46474215
#32854	032854uafs.html.out	0.43216804
#31332	03133uafs.html.out	0.40972403
#19839	019839uafs.html.out	0.33095247
#11745	011745uafs.html.out	0.28476787
#17117	017117uafs.html.out	0.24425438

real	0m31.389s
user	0m18.427s
sys	0m14.111s

uafs.edu



#22711	uafs.html.out	0.96920776
#25562	025562uafs.html.out	0.9411387
#27083	027083uafs.html.out	0.9291655
#33572	033572uafs.html.out	0.9002196
#33571	033571uafs.html.out	0.8953159
#41777	0.87936157uafs.html.out	041777
#30227	030227uafs.html.out	0.87801754
#22872	02287uafs.html.out	0.8082933
#35015	035015uafs.html.out	0.7995487
#12063	012063uafs.html.out	0.78565556

real	0m35.274s
user	0m20.472s
sys	0m16.331s

#29905	029905uafs.html.out	0.25840327
#12101	012101uafs.html.out	0.25634882
#5573	005573uafs.html.out	0.24116346
#7849	007849uafs.html.out	0.23287354
#2662	002662uafs.html.out	0.21350399
#28585	028585uafs.html.out	0.20389752
#28717	028717uafs.html.out	0.1999392
#28586	028586uafs.html.out	0.19864368
#27943	027943uafs.html.out	0.18948705
#22705	022705uafs.html.out	0.18580127

real	0m47.304s
user	0m31.457s
sys	0m19.926s