Alex Esplin
Garret Allen
Justin Carpenter

Search Engine Report

Statistics on DC

There are a total of 1,662,756 documents provided in the JSON file. While trying to only tokenize, we ran into memory space issues. From there, we decided to do all pre-processing steps detailed below before generating a full index. The size of this final index is 2,103,198 tokens.

Details on implementation decisions pertaining to text processing

We used the nltk word tokenizer, and the nltk snowball stemmer, set to English words. We used a custom stopword list that was compiled of all the suggested stopword lists. We used the nltk list as a starting point and extended it to also use the lists found at:
https://gist.github.com/sebleier/554280
http://www.lemurproject.org/stopwords/stoplist.dft
https://github.com/stanfordnlp/CoreNLP/blob/master/data/edu/stanford/nlp/patterns/surface/stopwords.txt
https://www.ranks.nl/stopwords
https://drive.google.com/file/d/1GgXVQg11M2h0RMftEH_-o1HPcJgsdWSw/view?usp=sharing

Initially we combine the title and content of the document, then we tokenize it. We then make all the tokens lowercase and check if they are in the stop word list. If they aren't then we check if they are ASCII characters, adding those that are ASCII to our list of filtered tokens. We then take this list of filtered tokens, and remove those that are punctuation. Finally we stem this list to create our final list of tokens that is now ready to be put into our index. We then decided to check once more for stop words just in case anything was stemmed into one, to ensure our index consisted of relevant tokens.

We decided that we wanted to use a stemmer for our index because we felt that the benefits from grouping similar words that had been stemmed to the same one, would outweigh the negatives of combining two words with similar spelling and very different meanings. We decided to remove stopwords, because the size of the index with the stopwords left in would be much too large for it to be efficient. We chose to use the stopwords from several different lists to try and remove as many as we could. We felt like we were unable to add any collection specific stop words due to the size and variety of the collection, so we just removed the basic sentence structure type of words. Finally we ended up removing the ASCII terms after a series of testing to try and make the list of tokens a workable size to load into memory.

Details on implementation decisions pertaining to query suggestions

To generate query suggestions, I began by joining all the query logs into a single file that I could process. There were several factors that needed to be determined across the entire log before I could accurately score individual suggestions, such as the maximum occurrence of any query in the log and the length of the longest query. Once these values were attained, I hardcoded them into variables to eliminate having to calculate them again. From there, I proceeded to score 1,000 candidate queries, returning the top 10 for display in the suggestions box. The decision to only process 1,000 candidates was rooted in the efficiency vs effectiveness trade off. The process was already slow, so processing more than 1,000 would make the search engine nigh on unusable. As it stands, that value could probably be lowered or the calculation optimized because it's still very slow, albeit effective. Similarly, we decided to only display the top 10 suggestions because the likelihood of a user scanning through more than 10 slowly generated suggestions was miniscule.

Details on implementation decisions pertaining to relevance ranking

We started with a small sample of the data we were working with (only 1,000 documents) and worked with getting all the words in each document into the index. Using this we were able to implement the TF-IDF scores for each term. This idea became unusable when we expanded to the full document. Our index became too large and slow to load into memory. Upon finding a way to store the index as a hashed index, we made significant progress in a retrievable index, but still not enough for it to be stored in memory. We then split the documents by 8 and created 8 different indexes. This was workable but the IDF scores were off leading to the TF-IDF scores being off.

Once we were able to generate valid indexes we made the choice to try and store all the TF-IDF values in a dictionary format. We used the same methods as we used for the index to make a retrievable hashed object that contains all the tokens with each tokens TF-IDF value if it was greater than zero in each document. Using this we could then grab the already processed TF-IDF linked with the document for each token in the query. Using these values we were able to find only the documents that contained all the tokens in the query.

While we finally got the document ID's for documents that contained every token in the query, this wasn't enough to rank the documents. We had to grab all the tokens with the same document ID and add each value's TF-IDF together and divide by the total number of tokens. This gave us the total normalized TF-IDF for each document with all tokens in the query. This is the final value we used to determine relevance ranking. We sorted by the highest to the lowest and then for the snippets took the top 20 document results and presented them to the user in the required format.

Details on implementation decisions pertaining to snippet generation

To generate the snippets we created a function that would take as arguments a doc id, the list of pre-processed query terms used to retrieve that document, and the relevance score. We then open the file that matches the doc id. Next we create a string that will be returned out of the function. We append the doc id, title of the document, and relevance score to the string. After that we look to the content of the document, and use nltk sentence tokenizer on the document to break it into sentences. We do this so we can get the cosine similarity value for each sentence and compare them to find the top two sentences to be used for the snippet. We then find the number of sentences each term in the query appears in and the total number of sentences. These values will be used to determine the IDF later on.

To assist in determining the cosine similarity we created a number of helper functions. We will make special mention of two of them that had interesting design decisions, the rest were simple functions to apply the cosine similarity equation. We also took advantage of tuples to keep the term and whatever related value we calculated together. This allowed us to make sure that in the various summations and multiplications used to find the cosine similarity we were using the appropriate values.

The first function is used to determine the term frequency. It takes in a list of words, either a sentence or the query, and then finds the term frequency for each word in this list. To do this we used the Counter class. This is a subclass of dict, and will hash each item in a list and count how many times it appears. Using this we can easily find the most common word in the sentence and find out how many times it appears. From there it is simple to calculate the TF.

The next helper function we used was to determine the IDF, this function used the number of sentences we calculated above. We realized that the IDF would be zero for any term that wasn't also in the query, and thus would be essentially removed when performing the summations those values were involved in. So to save some time we only calculate the IDF for terms in the sentence that also appear in the query.

With the TF and IDF computed for the query as well as each term in each sentence we could easily get the TF-IDF values. We then used these values to find the numerator and denominator of the cosine similarity function. The difficult part here was to make sure to multiply the same terms different values together, we were able to solve this because each function we used return a list of tuples that contained the term and the value.

We go through every sentence in the document and find its cosine similarity score, each score is added to a list that contains the sentence and the score. Then this list is sorted on the scores and we take the top two from the list and append them to our result string. Due to being able to directly look up the document from the doc id this function runs fairly quickly, usually returning a result in about a second.

Discussion

When starting this project we had a couple ideas spread out between the three of us. We collaborated our pre-processing programs we all did for homework 3 into a usable file, this we thought initially was enough. Once we parsed the total file and processed it (removed all the stop words using nltk stopwords list, using nltk tokenizer and nltk stemmer) we attempted to run the full index and store the document ID and the position in the document which the token occurred at. This process, once we tested it using the 1,000 document test file, we tried on the full document collection. This process we let run for days and resulted in an 8GB file which we thought was great. Only to find that when we tried to load in the index to calculate the TF and IDF for the terms, it was too large for our computers to handle and was incredibly slow.

We knew this index wasn't going to work so we ventured onto finding a way to speed the creation of the index up. While doing this Garrett figured out how to implement multithreading in

Python. This was a new idea that we all joined into learning. We believed we were ahead of the point we were aiming for so we switch tasks and divided up to work on the UI of the search index, index generation, and snippet generation. This was working for a while and once the UI and the snippet was in need of a workable index, we hit a brick wall.

We tried to think of why we couldn't load the index and assumed it was due to the data structure we were using. We all did some research and found some better data structures. We all were working on making an index with our hardware available and even tried using the AWS servers to run the index generation. The implementations we tried went in order from a list which we could generate but couldn't retrieve any information from, to a dictionary which we didn't have the memory to store all the information needed to generate it, to a B-Plus Tree which was developed to read and write to our storage instead of memory but a 70 thousand document index was over 300Gb to store, to finally a hashed index which we don't know for sure if it is better than the rest but we ended up getting a usable index for a few reasons.

The reasons why we ended up after countless tries of lists, dictionaries, arrays, and B-Plus trees with the hashed index was because at this time, we tried to split the data collection into 8 parts. This allowed us to make and retrieve the full index of ⅛ of the data collection. We were in need of an index to work with to get us past this brick wall. This leads us to the major turning point. Once we were able to retrieve the index we were able to implement the TF-IDF for each term. The initial thought of calculating the TF-IDF for each term while processing the users query was seemingly inefficient, so we attempted to calculate all the terms TF-IDF for the entire 200k documents for each index. This was a decent approach but we noticed we were storing so many zeros for the TF-IDF scores where the term didn't exist in the document. So once we fixed this to only add the document ID and TF-IDF to the term in the dictionary we had a workable, retrievable data structure of each tokens TF-IDF and the document's ID it's in.

This broke past the brick wall we were all stuck at, we all were able to divide and conquer the remainder parts of this project to then join back in to fully develop our working search engine.

Once we had a working search engine we went with these five queries: "Adolf the swedish architect", "Palestine", "Baltimore Police", "Australia surfing competition", and "Apple computer". We chose these queries because they felt unrelated enough to test the flexibility and usefulness of our search engine. We expected these queries to return snippets that contained at least some of our query tokens, because this would validate that our processing and ranking system actually does return relevant information. The outcomes of searching our test sets are given in the tables below. We were pleasantly surprised that our search engine did indeed return mostly relevant results. However, it was far from perfect and could use some additional iterations to improve the returned results. Our results and the results obtained from running the same queries on Wikipedia's search module were not the same at all, none of our results matched at all with theirs. This may be because we only have 1.6 million Wikipedia documents, so we may not have the documents they return. It more likely is because their pre-processing and ranking algorithms are different than the ones we implemented in our search engine. Overall we are happy with what we were able to create during this project.

Alex Esplin
Garret Allen
Justin Carpenter

Tables

Query: Adolf the swedish architect			
Ranking	Document ID	Snippet	Ranking Score
1	607164	Gunnar_Gunnarsson In 1939, Gunnarsson moved back to Iceland and first settled on Skriðuklaustur, a farm in East Iceland, where he built a house designed by German architect Fritz Höger. In 1940 Gunnarsson, a long time Nazi sympathiser, travelled war-time Germany in an extensive lecture tour, also meeting with Adolf Hitler.	9.923591323883495e-08
2	926744	List_of_Dutch_inventions_and_discoveries Like Ivanovsky and Adolf Mayer, predecessor at Wageningen, Beijerinck could not culture the filterable infectious agent. Bluetooth, a low-energy, peer-to-peer wireless technology was originally developed by Dutch electrical engineer Jaap Haartsen and Swedish engineer Sven Mattisson in the 1990s, working at Ericsson in Lund, Sweden.	9.918964401713599e-09
3	695690	Rainbow_Room In the decade following its opening, the Rainbow Room had hosted former Spanish queen Victoria Eugenie of Battenberg; Norwegian Crown Prince Olav and Crown Princess Martha; and Swedish Crown Prince Gustaf Adolf and Crown Princess Louise. The Rainbow Room was originally designed by architect Wallace K. Harrison, of Rockefeller Center's Associated Architects, as well as interior designer Elena Bachman Schmidt.	9.555844537035481e-08
4	122283	Edward_Wood__1st_Earl_of_Halifax	9.5478031531333

Alex Esplin
Garret Allen
Justin Carpenter

		He is regarded as one of the architects of the policy of appeasement prior to World War II, although after the German occupation of Czechoslovakia in March 1939, he was also one of those who pushed for a new policy of attempting to deter further German aggression by promising to go to war to defend Poland. On being taken to meet Adolf Hitler at Berchtesgaden, Halifax almost created an incident by nearly handing his coat to him, believing him to be a footman: "As I looked out of the car window, on eye level, I saw in the middle of this swept path a pair of black trousered legs, finishing up in silk socks and pumps.	48e-09
5	202025	Adolphe_Théodore_Brongniart He was the son of the geologist Alexandre Brongniart and grandson of the architect, Alexandre-Théodore Brongniart. In 1851, he was elected a foreign member of the Royal Swedish Academy of Sciences.	9.4549772891445 52e-08
Query: Palestine			
Ranking	Document ID	Snippet	Ranking Score
1	1185214	Bayt_Dajan By the time of the Mandatory Palestine, the village housed two elementary schools, a library and an agronomic school. In the 11th century, Bayt Dajan served as a headquarters for the Fatimid army in Palestine.	9.9988667950965 57e-07
2	123700	Chocolate_coated_marshmallow_treats European chocolate-coated marshmallow treats were popular as homemade sweets in Mandate Palestine, when it was known as Kushi (Hebrew כושי, ""Nubian"" or Black African) and Rosh Kushi (Hebrew language: ראש כושי ""Nubian's head"") This name was borrowed from the names then used in	9.9968969631826 27e-08

		Europe.	
3	466686	United_Nations_Convention_against_Corruption As of October 2017, there are 183 parties, which includes 178 UN member states, the Cook Islands, the Holy See, the State of Palestine, and the European Union.	9.979203340238942e-08
4	672663	History_of_the_Jews_in_Portugal Thousands had flooded the city, trying to obtain the documents necessary to escape to the United States or Palestine.	9.979203340238942e-08
5	775947	Legitimacy_of_the_2003_invasion_of_Iraq He had a long history of supporting terrorists in Palestine by giving money to families of suicide bombers and gave refuge to other terrorist groups against neighboring states in the region.	9.961572238931099e-08
Query: Baltimore Police			
Ranking	Document ID	Snippet	Ranking Score
1	528983	Crisis_pregnancy_center Christian Action Council founded its first center in Baltimore, Maryland, in 1980. Crisis pregnancy centers, along with hospitals and fire and police stations, are designated by state law in Louisiana as emergency care facilities where parents may surrender custody of newborn infants.	9.994422212907198e-09
2	1108424	Ben_Bass__actor_ He is best known for his role as 16th century Spanish vampire Javier Vachon in the series "Forever Knight", and for his more recent role as officer/detective Sam Swarek on the Global police television series "Rookie Blue," which also aired on ABC. Bass was born in Baltimore, Maryland.	9.988543141017253e-08

3	531273	West_Africa_Squadron As the Royal Navy began interdicting slave ships, the slavers responded by adopting faster ships, particularly Baltimore clippers. In order to enforce this ruling in 1808 the Admiralty dispatched two vessels to police the African Coast.	9.968996089860663e-08
4	250764	GoDaddy GoDaddy was founded in 1997 by Baltimore, Maryland, entrepreneur Bob Parsons. Some reports said there had been complaints from police.	9.941758395626072e-09
5	1162446	Tommy_Carcetti Carcetti is idealistic and ambitious, and has the backing of local Democrats in Baltimore's 1st district as well as Major Stan Valchek of the Baltimore Police Department. Burrell, believing the mayor is making a ploy against the police, leaks the information to Carcetti.	9.941758395626072e-08
Query: Australia surfing competition			
Ranking	Document ID	Snippet	Ranking Score
1	225512	Nissan_Skyline The crease that appeared over the rear wheels beginning with the second generation, referred to as the "surf line", no longer appeared starting with this generation. Also lauded were the cars road manners, as evident by the six-cylinder Skyline's competition successes.	9.972620917218346e-09
2	127731	Unilever Unilever owns over 400 brands, with a	9.72226802040607e-09

Alex Esplin
Garret Allen
Justin Carpenter

		turnover in 2016 of over 50 billion euros, and thirteen brands with sales of over one billion euros: Axe/Lynx, Dove, Omo, Becel/Flora, Heartbrand ice creams, Hellmann's, Knorr, Lipton, Lux, Magnum, Rama, Rexona/Degree, Sunsilk and Surf. Lynx/Axe: Axe, known as Lynx in the United Kingdom, the Republic of Ireland, Australia and New Zealand, is a toiletries brand marketed towards young men between the age of 16 and 24.	
3	183096	Kelly_Slater Kelly Slater Robert Kelly Slater (born February 11, 1972) is an American professional surfer known for his competitive prowess and style. Some of his favorite surf spots include Mondos in Ventura, California, Pipeline in Hawaii, Kirra in Australia, Jeffreys Bay in South Africa, Minis in Ireland, Taghazout in Morocco, Veiny's in New Zealand, Soup Bowls in Barbados, and Sebastian Inlet near his home in Florida.	9.671295864669944e-09
4	1292624	The_Arts_Centre_Gold_Coast From that point, Meyer's ferry would take the travellers across the river to Meyer's Ferry Road at Elston (as Surfers Paradise was then known) along which the travellers would walk to the surf beach. Although the surf beaches of Elston were a popular destination, the lack of road access limited the extent of residential and commercial development.	9.65531025167049e-08
5	1323581	Serena_Brooke Brooke began to compete in amateur competitions in 1990 where she was crowned the Queensland amateur surfing title as well as the Australian National Title. Among her notable accomplishments was winning the Billabong Pro Australia title and	9.65531025167049e-08

Alex Esplin
Garret Allen
Justin Carpenter

		achieving a temporary #1 overall ranking in 2001.	
Query: Apple computer			
Ranking	Document ID	Snippet	Ranking Score
1	789207	Network_Computer_Reference_Profile Network Computer Reference Profile Network Computer Reference Profile (NC reference profle, NCRP) was a specification for a network computer put forward by Oracle Corporation, endorsed by Sun Microsystems, IBM, Apple Computer, and Netscape, and finalized in 1996. The minimum hardware requirements were:	9.997266303069793e-07
2	169957	GeForce_3_series Apple would later announce launch rights for its new line of computers. A separate professional version, with a feature-set tailored for computer aided design, was sold as the Quadro DCC.	9.985449203420776e-08
3	338532	William_Campbell__business_executive_ Campbell became CEO of GO Corporation, a startup pioneering a tablet computer operating system. He was VP of Marketing and board director for Apple Inc. and CEO for Claris, Intuit, and GO Corporation.	9.985449203420776e-08
4	564025	Tass_Times_in_Tonetown "Compute!" Game reviewers Hartley and Pattie Lesser similarly commented on the game in their "The Role of Computers" column in "Dragon" #116 (1986), stating "This one is truly bizarre."	9.985449203420776e-08
5	1036856	Zattoo	9.9854492034207

		Zattoo also offers TV applications for Internet-connected TVs (LG, Samsung, VideoWeb TV box), Xbox, Xbox One, Amazon Fire TV, Apple TV, Android TV, and supports streaming via Apple AirPlay and Chromecast. Live TV and on-demand content from Zattoo can be watched on computers, smartphones (iPhone, Android, WP7), and tablets (iPad, Android, Windows 8, Windows 10).	76e-08

Table 1

Query: Adolf the swedish architect		
Ranking	Candidate Suggestion	Score
1	adolf hitler coin	1.0000119507152503
2	adolf hitler pics	1.0000239014305006
3	adolf hitler pictures	1.0000836550067522
Query: Palestine		
Ranking	Candidate Suggestion	Score
1	palestine herald press	1.1950715250307731e-05
2	palestine transit	2.3901430500615462e-05
3	palestine tx	3.5852145750923193e-05
Query: Baltimore Police		
Ranking	Candidate Suggestion	Score
1	baltimore county public schools payroll department policy regarding garnishments	1.1950715250307731e-05
2	baltimore limo	2.3901430500615462e-05
3	baltimore sun obituaries	3.5852145750923193e-05

Query: Australia surfing competition		
Ranking	Candidate Suggestion	Score
1	australian famous people	1.1950715250307731e-05
2	australian outback	2.3901430500615462e-05
3	australian kelpie	3.5852145750923193e-05
Query: Apple computer		
Ranking	Candidate Suggestion	Score
1	apple tress	1.1950715250307731e-05
2	apple valley ohio	2.3901430500615462e-05
3	apple juice	3.5852145750923193e-05

Table 2

Query: Adolf the swedish architect	
Ranking	Document Title
1	Adolf
2	Asolf Fredrik’s Music School
3	List of Swedish architects
4	Adolf Hitler
5	Fick
Query: Palestine	
Ranking	Document Title
1	Palestine
2	State of Palestine
3	Mandatory Palestine
4	Palestine (regine)

5	Israeli-Palestineian conflict
Query: Baltimore Police	
Ranking	Document Title
1	Baltimore Police Department
2	2015 Baltimore protests
3	Crime in Baltimore
4	Police of The Wire
5	Baltimore
Query: Australia surfing competition	
Ranking	Document Title
1	Surfing in Australia
2	World Surf League
3	Stubbies (surfing)
4	Australian Open of Surfing
5	Noosa Festival of Surfing
Query: Apple computer	
Ranking	
1	Apple Inc.
2	History of Apple Inc.
3	Timeline of Apple Inc. products
4	Apple Corps v Apple Computer
5	Macintosh

Table 3