Let’s build a search engine: Exploring the past with Natural Language Processing

Peter Nadel

Digital Humanities Natural Language Processing Specialist

Research Technology, TTS

What is a search engine

- We’re all used to tools like Google or JumboSearch but how do they actually work?

- Information retrieval: Understand a user query and check it against documents in a corpus quickly and efficiently

What is a search engine

- Key components:
  - An index – like a book's index but for all of the documents in a collection
  - A query processor – a tool that understands language and know what you're looking for
  - A ranking system – something that determines what documents are most relevant to your query

Indexing

- Indexing happens before any searching begins
- The index is what we will search through to get our documents
  - Breaks down each into individual words (tokenization)
  - Records where each word appears
  - Creates signposts for each word’s location
- Very similar to creating finding aids in an archive

Indexing

- The indexer prepares documents in a couple steps:
  - Normalization: Converting words with the same meaning to the same form (“Theatre” and “Theater”)
  - Stemming: Recognizing certain words are related to each other (“runs” and “ran” are both forms of “run”)
  - Stop words: Filtering out very common (“The” and “This”)
- What kinds of text does this work best for? What kinds of text does this not work for? How about other languages?

Querying

- What happens when you type into a search bar
- The search engine processes your search **in the same way** it processed the documents
  - Critical that it is the same way
  - Need to be comparing like to like
- Looks up processed terms in the index
- Finds all documents containing these terms

Ranking

- Once a search engine has found documents related to the query, how does it rank them?
  - Want a ranking system that gives us the most relevant result first
  - So far our index only tells us where words occur
  - No measure of relevancy yet
- We need a method that *automates* this relevancy quickly and efficiently

Ranking

- Term Frequency - How often does the word appear?
  - More mentions is a topic is more relevant than other documents
  - Is this assumption always true?
- Inverse document frequency – How unique is this word across all documents?
  - Common words appear everywhere, so they'll be less useful
  - Rare terms will likely be more meaningful
- Combined score: **TF-IDF**

Ranking

- Term frequency-Inverse document frequency (TF-IDF)
  - Goal: Balance how often a term appears with how distinctive it is
  - Helps find documents that are strongly focused on search terms
  - Prioritizes sources that both frequently discuss the topic and provide new context
- But how can we use it?

TF-IDF Example

- Here is our corpus:
  - First document: “Call me Ishmael.”
  - Second document: “I am an invisible man.”
  - Third document: “Mother died today. Or maybe, yesterday; I can't be sure”
  - Fourth document: “All this happened, more or less.”
- We are going to:
  - Normalize
  - Stem
  - Use TF-IDF to create a tool to rank texts

Example: Normalization

- Normalization, in this case, will just consist of lower-casing and removing punctuation:
  - First document: “call me ishmael”
  - Second document: “i am an invisible man”
  - Third document: “mother died today or maybe yesterday i cant be sure”
  - Fourth document: “all this happened more or less”

Example: Stemming

- When we stem, we will turn all words into their base forms:
  - First document: “call i ishmael”
  - Second document: “i be a invisible man”
  - Third document: “mother die today or maybe yesterday i cant be sure”
  - Fourth document: “all this happen more or less”

Example: TF-IDF

- Now we can make a document term matrix (DTM) out of our documents using TF-IDF
- First, we will just do term frequency
- Need to make a spreadsheet where each column is a unique word from all of our documents and each row is a document

Example: Term frequency

	all	be	call	cant	die	happen	invisible	ishmael	less	man	maybe	more	mother	or	sure	this	today	yesterday
Document 1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Document 2	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
Document 3	0	1	0	1	1	0	0	0	0	0	1	0	1	1	1	0	1	1
Document 4	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0

Example: Document Term Matrix

	all	be	call	cant	die	happen	invisible	ishmael	less	man	maybe	more	mother	or	sure	this	today	yesterday
Document 1	0.000000	0.000000	0.707107	0.000000	0.000000	0.000000	0.000000	0.707107	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Document 2	0.000000	0.486934	0.000000	0.000000	0.000000	0.000000	0.617614	0.000000	0.000000	0.617614	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Document 3	0.000000	0.274603	0.000000	0.348299	0.348299	0.000000	0.000000	0.000000	0.000000	0.000000	0.348299	0.000000	0.348299	0.274603	0.348299	0.000000	0.348299	0.348299
Document 4	0.421765	0.000000	0.000000	0.000000	0.000000	0.421765	0.000000	0.000000	0.421765	0.000000	0.000000	0.421765	0.000000	0.332524	0.000000	0.421765	0.000000	0.000000

Example: Querying

- Now that we have a document term matrix, we can create a query and rank the document accordingly
- First we need to normalize it and stem it like we did to the documents
- Then we need to create a single row like this query is a new document
- Last we will take the dot product of each document row and this query row to get a single number for each

Example: Query row

Our example query will be “invisible man”

	all	be	call	cant	die	happen	invisible	ishmael	less	man	maybe	more	mother	or	sure	this	today	yesterday
Query	0.0	0.0	0.0	0.0	0.0	0.0	0.707107	0.0	0.0	0.707107	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We can now compare this query row to all of the other rows with a simple matrix: the dot product.

- Take the product of corresponding elements
- Sum all of these together
- Rank documents by the result

Example: Ranking

	Document 1
all	0.000000
be	0.000000
call	0.707107
cant	0.000000
die	0.000000
happen	0.000000
invisible	0.000000
ishmael	0.707107
less	0.000000
man	0.000000
maybe	0.000000
more	0.000000
mother	0.000000
or	0.000000
sure	0.000000
this	0.000000
today	0.000000
yesterday	0.000000

	all	be	call	cant	die	happen	invisible	ishmael	less	man	maybe	more	mother	or	sure	this	today	yesterday
Query	0.0	0.0	0.0	0.0	0.0	0.0	0.707107	0.0	0.0	0.707107	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- For ‘all’: 0 * 0 = 0
- For ‘be’: 0 * 0 = 0
- For ‘call’: .7 * 0 = 0
- For ‘cant’: 0 * 0 = 0
- For ‘die’: 0 * 0 = 0
- For ‘happen’: 0 * 0 = 0
- For ‘invisible’: 0 * .7 = 0
- For ‘ishmael’: .7 * 0 = 0
- For ‘less’: 0 * 0 = 0
- For ‘man’: 0 * .7 = 0
- For ‘maybe’: 0 * 0 = 0
- For ‘more’: 0 * 0 = 0
- For ‘mother’: 0 * 0 = 0
- For ‘or’: 0 * 0 = 0
- For ‘sure’: 0 * 0 = 0
- For ‘this’: 0 * 0 = 0
- For ‘today’: 0 * 0 = 0
- For ‘yesterday’: 0 * 0 = 0

0 + 0 + 0 ... = 0, so we can say that this query has no relevancy to document 1.

Example: Ranking

	Document 2
all	0.000000
be	0.486934
call	0.000000
cant	0.000000
die	0.000000
happen	0.000000
invisible	0.617614
ishmael	0.000000
less	0.000000
man	0.617614
maybe	0.000000
more	0.000000
mother	0.000000
or	0.000000
sure	0.000000
this	0.000000
today	0.000000
yesterday	0.000000

	all	be	call	cant	die	happen	invisible	ishmael	less	man	maybe	more	mother	or	sure	this	today	yesterday
Query	0.0	0.0	0.0	0.0	0.0	0.0	0.707107	0.0	0.0	0.707107	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- For ‘all’: 0 * 0 = 0
- For ‘be’: .49 * 0 = 0
- For ‘call’: 0 * 0 = 0
- For ‘cant’: 0 * 0 = 0
- For ‘die’: 0 * 0 = 0
- For ‘happen’: 0 * 0 = 0
- For ‘invisible’: .62 * .7 = 0
- For ‘ishmael’: 0 * 0 = 0
- For ‘less’: 0 * 0 = 0
- For ‘man’: .62 * .7 = 0
- For ‘maybe’: 0 * 0 = 0
- For ‘more’: 0 * 0 = 0
- For ‘mother’: 0 * 0 = 0
- For ‘or’: 0 * 0 = 0
- For ‘sure’: 0 * 0 = 0
- For ‘this’: 0 * 0 = 0
- For ‘today’: 0 * 0 = 0
- For ‘yesterday’: 0 * 0 = 0

(.62 * .7) + (.62 * .7) = .87, so we can say that this query is very relevant to document 2.

Application

- I’ve created a simple application that implement these principles: https://search-engine-creator.streamlit.app/
- Upload a CSV where each row is a new document
- Index your data
- Search your data