Q1

An example of an ad-hoc information retrieval system that is not web search is the Purdue Libraries.

The corpus from which the documents are retrieved is the database of research papers, books, articles and more that Purdue has access to. An example of an object in the corpus is "The Thermalization, Condensation and Flickering of Photons." Journal of Physics B: Atomic, Molecular and Optical Physics by Klaers, Jan.

The structure of the query is the name of the book, paper, article or author that we are searching for. For example, to search for the article I’ve mentioned above I would write the query, “Thermalization, Condensation and Flickering of Photons.”

The retrieved information is returned as a list of all the articles and books related to the query with the title, name of the author and a brief description about whether it is available for free, peer reviewed and more.

By an anonymous student

Q2

An example that is not ad-hoc IR might come from the music streaming service Spotify. Spotify records data about a given user’s preferences and music tastes and uses these preferences that are at least somewhat consistent over time for music recommendation. On a weekly basis, Spotify automatically generates a "Release Radar" playlist for the user using these preferences as some form of a filter.

The documents in this retrieval are songs from some corpus of music released within the past week. They can be just about anything, ranging from the latest Taylor Swift track to a new single from BROCKHAMPTON. The query is not performed by the user and is based on Spotify’s user data, so the query question is not as relevant here. A small set of 30 tracks is retrieved and presented to the user in the form of a playlist.

This example still fits into the idea of information retrieval as Spotify’s recommendation system is filtering a collection of new tracks that are supposed to be relevant to the user and satisfy an information need for enjoyable new music. In particular, this might be closer to filtering instead of ad-hoc IR: the system is filtering tracks based on user preferences.

By Joel Edward Klasa
The “Search Contacts” functionality included in the contacts app on an iPhone is an example of an ad-hoc information retrieval system that is not web search. The corpus of the Search Contacts function is all the contacts, with their respective information (such as name, phone number(s), email(s), etc.), saved on the iPhone. An example of a retrieved contact would be the following:

Name: Jane Doe
Number: (765) 123-4567
Work Phone: (765) 456-1237
Email: jdoe@purdue.edu

The query used to search for such a contact would typically be the desired contact’s name (or partial name). However, a contact can be searched for by inputting any piece of information that the user knows about the contact, such as a partial phone number or email. An example of a query to look up Jane Doe’s contact info can be any of the following:

Search: Jane
Search: jdoe
Search: 4567

Upon performing a search, a list of all contact that contain the searched for information (or partial information) will be listed, permitting the user to choose the desired contact. The chosen contact’s information will then be listed.

By Zaid Al Haddadin

Controlled Vocabulary Indexing Better: Example – legal and law-related databases

Controlled Vocabulary Indexing would be better in this case because, first of all, lawyers know what they’re looking for. They know the terms for the forms and laws and cases that they need to find, and so using a controlled vocabulary to find a collection of documents pertaining to a particular form or law or case allows the lawyer to retrieve a more broad spectrum of very relevant documents to what they’re searching for.

It also reduces the noise in the results as long as the user knows how to effectively use the CVI. Example: In a situation where a law database uses full text indexing, what if a lawyer searched “children in need of services”. This is a common phrase in law, and has a specific meaning and often involves petitions and court orders. As you can imagine, a lawyer has a specific set of things they want returned when they search “children in need of services”. However, if you full text index both this search and the documents in question, you could get very noisy results due to the presence of stop words as well as common words in the query that you could also find.
strewn throughout case files. So a lot of case files containing the words “children”, “in”, and “of” would probably be high in the list of relevant documents. These cases may be better referred to by a particular word or phrase too!

Q5 (almost all of this was done with the Porter stemmer)

Stopword removal should come before stemming.

This is because complicated situations could arise from stemming before stopword removal, namely two specific complications:

1. Stopwords could be stemmed to a word that is no longer a stopword
2. Valuable non-stopwords could be stemmed to a stopword (or the same stem as a stemmed stopword)

Examples of complication 1:
Suppose you have the document “shoes are cool”.
The porter stemmer stems “shoes are cool” to “shoe ar cool”
Now, if the word “are” were present in your stopwords list, it would have found the stopword in the original pre-stemmed document if you’d run it first, but because you stemmed “shoes are cool” to “shoe ar cool”, it fails to identify that “ar” was originally a stopword.
This means stopword removal turns “shoe ar cool” into “shoe ar cool”

Alternatively, if you’d removed stopwords and then stemmed, the document transformation would look like so:
“shoes are cool” -> stopword removal -> “shoes cool”
“shoes cool” -> stemming -> “shoe cool”

As you can see, stemming then stopword removal resulted in “shoe ar cool”, whereas stopword removal then stemming resulted in “shoe cool”. The significance of this difference will be discussed in the concluding paragraph.

Note that proper use of a stemmer would be to stem both the stopword list and the document before applying stopwords, which would alleviate this complication. However, this is not a safe fix. There exist some situations where this would lead us to complication 2.

Example of complication 2:
Suppose you have the document “her herring”

If you stem first, you’ll get the document “her her”, which then if you remove stopwords you end up with an empty document “”

Alternatively, if you remove stopwords first, you transform “her herring” into just “herring”, which then stems to “her”.
So, if you stem then remove stopwords, you actually end up with an empty document “”, while if you remove stopwords first and then stem, you end up with “her”.

Another example to highlight another issue, look at the document “hers herrings”

This example highlights the problem with the solution posed earlier where I could just add the stemmed stopwords to the stopword list so I still end up removing the stopwords.

In this case, assume “hers” is a stopword but “her” is not.

If we stem then remove stopwords, we transform “hers” to “her” and “herrings” to “her”. This means we wouldn’t catch the resulting word “her” from stemming the stopword, essentially failing to remove a stopword on the stopword removal pass, so the resulting query would be “her her”. But we want to add the stopword stem to the stopwords list so we make sure to get rid of any stopwords. So we add “her” to the stopword list. But! If we look at the stemmed query, it’s “her her”. If we added “her” to the stopword list, we’d be removing a word that stemmed into the same stem as a stopword! This is why this solution method involving adding stopword stems to the stopword list doesn’t work.

Of course, you could just make your stemmer not stem stopwords at all by utilizing the stopword list in your stemmer. But by that point, on iterations the stemmer is already making checks between the document terms and the stopword list in order to avoid stemming the stopwords present in the document, so why not just remove them during these checks instead?

Why does this difference between resulting documents matter? Let us look at the resulting documents from complication example 1. If we stemmed then removed stopwords, we get the document “shoe ar cool”. If we remove stopwords and then stem, we get “shoe cool”. If we look at the original document, “shoes are cool”, “are” is a stopword. When comparing “shoe ar cool” vs. “shoe cool”, it’s evident that the former contains “ar”, which is a stem of a stopword and in turn has the same meaning value as the stopword itself; it is not valuable to the meaning of the document. Many unrelated documents are bound to have the word stem “ar” in it because many documents will have originally had the word “are” in them. This means removal of the word stem “ar” will improve our results for the same reason removing stopwords improves our results: because it means we won’t get matches between unrelated documents and queries just because they both contain the word stem “ar”.

Example:
Query = “dogs are fun” -> stem then stop -> “dog ar fun”
Doc1 = “dogs fun” -> stem then stop -> “dog fun”
Doc2 = “are fun” -> stem then stop -> “ar fun”
Doc3 = “dogs” -> stem then stop -> “dog”
You should notice that the number of word matches between the query and doc1, and the query and doc2 are actually the same. They both have 2 words that match the query. Though doc 2 logically seems like a much less relevant result than doc1. This is a problem. This occurred because the stemmed stopword “ar” was left over, resulting in matching with irrelevant docs, and generating noisier results that have less relevant documents ranked higher than they should have been.