Some notes about Text Processing & Data Science

Introduction:

Several students ask about text-processing in DataScience. This note is a quick overview of the subject – but definitely from my experiences starting in the world of information retrieval.

Text Processing writ large:

Imagine a collection of any size consisting of "text" documents. These documents may be literature, technical & medical reports, newspaper articles, web pages … anything. Next look at the nature of these documents: some have no inherent structure, others do, in the form of title headings (e.g., "abstract:"), others in biomedical work often have a strict structure so we can determine the general domain from a paragraph's location in the document. This is the "document collection representation, *D*."

Next is the processing of the data. One approach is to create the "big bag of words." This over-characterization is where the entire *D* is *parsed*. Parsing a collection means breaking down the texts into individual word units, often stripping away *word endings*, excluding *stop words*, and then creating a *term frequency/document matrix*. [In English, the Porter Stemming Algorithm, 2nd ed, PSA2, is used to remove endings, transforming "goes" into "go" and so on. Other languages have their own stemmers. Stop words include the most commonly appearing words in a language. Some research says to keep them in the collection because they contribute to "noisy data"; others say to remove stop words because they do not contribute to meaningfulness. These may include ("the", "that", "is", "a", "an") and so on.]

The terms in the matrix are counted; a normalized numeric value is calculated (usually to the length of the document *d* and/or the entire *D*), and then some *term weight* is added to the numeric value. Adding term weights is intended to emulate how language is actually used in daily life; although other techniques apply weights based on some other criterion, such as "the number of web pages that refer back to the page being parsed." This latter technique is at the heart of Google's PageRank algorithm. Say given a collection of *n* documents, there is the possibility of *t* terms. Depending on the approach used, a boolean 1 means term exists; 0 means doesn't exist; most approaches use a value between 0:1. The cardinality isn't important except for determining normalization. The number of fields can be exceptionally large: a single column for every term that appears in any document in the collection. In this example, I've collapsed the various iterations from actual string of terms through weight through a final value. This final value is used to compare the numeric representation of the query to the candidate documents. For instance, say a query term is assigned [arbitrarily] a value of 0.5. All the terms that are within *x* of 0.5 are considered potentially relevant; then the potentially relevant set are ranked according to their degree of similarity. A popular teaching approach is Salton's *general vector model* where the Collection is represented as a 3d space of *n* vectors. The query vector is introduce arbitrarily. Then all the document representation queries within a certain value (the cosign value) within a range constitutes the retrieval set. [Of course there are reduction techniques, such as linked lists, doubly-linked lists, paragraph-length compressions, etc.]

Next there's the query - what are you looking for? The query can be a string of letters and words from a search box (as in a web page) or some other input. The inputs can be automated (such as a bot) or ever trigged by some state change (such as in evidence adaptive agents). Whatever the query's form, it is represented as *Q*.

Between the *D* and the *Q* there is a framework for modeling document representations, queries, and their relationships. The framework is the researcher's world … and why there are so many search engines.

Finally, there is a *relevancy ranking function, $R(q_i, d_j)$* that associates a real number with a query $q_i \in Q$ and a document $d_j \in D$. This ranking defines the ordering (the relevancy ranking) of the documents in response to the query q_i.

Collectively this is the world of "information retrieval." Python and SQL usually include libraries for a popular test algorithm, called "*inverse document frequency • term frequency*" [idf/tf] and "*inverse document frequency • term weighting*" [idf/tw]. Of course there are no end of approaches in each of the *Q, C, F(i,j),* and *R*! For a good intro, see Baeza-Yates & Ribierto-Neto, *Modern information retrieval*, 2nd ed. New York: ACM Press. For a good intro the the many algorithms, particularly data compression and reduction techniques, see Frakes and Baeza-Yates, *Algorithms for information retrieval*.

Types of algorithms:

It's not possible to list all the approaches but we can understand a few main ones. The most basic is binary - a query term exists in a document or it doesn't. Then there's extended boolean - the query term exists or it doesn't + other factors to organize the retrieval results (such as by year). Some others are algebraic, others are based on statistics.

Before continuing, it's important, indeed vital, to mention data mining and text mining. In the former case, relational databases are designed to respond quickly (and usually boolean) for questions that were anticipated in the design of the database. For example, "how many widgets did we sell in April in Paris?" SQL cannot answer "why did we sell those widgets?" The field of data mining takes data from the SQL tables and creates its version of the big bag of words and then queries the collection. Closely related is the text mining. Same idea but with an original corpus of text documents. Text- and data mining activities usually require (a) lots of statistics and (b) visualization of results with the help of a domain expert to interpret the results. [There are two approaches: *hypothesis generation* and *hypothesis confirmation*. And there's lots of other techniques, such as C4.5 and CART.]

Another approach is to cast algorithms into either *machine translation* and *artificial intelligence*. MT uses preprogrammed rules to help create retrieval sets. AI approaches range wildly, often using Bayesian techniques, predictive models, genetic algorithms, artificial neural nets, and *more*.

Finally, there's *natural language processing, NLP*. This is all the rage today … but it is rather muddied. Depending on the researcher's point of view, NLP is close to computational linguistics, breaking down texts into sentence units and then from there into parsed elements. This is not without controversy. Chomsky's famous test phrases are evidence. 1: "Sleeping green ideas dream furiously", 2: "Flying planes is dangerous", 3: "The boy saw the girl with the telescope." The problem with [1] is that the sentence is well-formed but not possible in our world, so can-

not be evaluated as true/false. [2] is ambiguous. [3] is syntactically ambiguous because it could mean "the boy, by means of a telescope, saw the girl" or "the boy saw a girl carrying a telescope."

In Data Science:

In DataScience, all of the above seem to flow together to answer today's "why" questions. The tools for creating the bag of words are greatly facilitated with "web scrapping" libraries. Next there are more efficient ways of pre-processing the data. Here DS diverges from IR in that some NLP techniques, such as parts of "speech tagging and shallow parsing" are included. Unlike PSA2, contractions are expanded: from "don't" to "do not"; like PSAs there's still lemmatization, removing stop words, and normalization.

Now we blend in some shallow parsing or chunking: the "the boy sees the girl with the telescope" becomes a sequence of Noun Phrase "the boy", Verb Phrase "sees + direct object"; NP (direct object) "the girl", and a Prepositional Phrase ("with the telescope."). There's also adjective and adverb phrases in other sentences to be collected. Some work aims to disambiguate the deixic (or relative time) phrases.

Determining whether "with the telescope" belongs to the boy or to the girl is helped through constituency and dependency parsing. [There's a Stanford Parser tool you can download; see also the probabilistic context-free grammar (PCFG) parsers.] (Read more about this in Sipser.)

As the data sets are so vast and so rich, a common technique is to use "NER" or named entity recognition. This helps to disambiguate terms. For instance "china" could refer to the dinner plates or to the country. NER might distinguish "China" the country and so automatically exclude the plates. [Reduction in dimensional space and NER really improve retrieval speed and relevancy.]

Finally, especially in light of the Internet [and so-called "social computing" and work by people like Jeff Herr's content analyses driven by information visualizations], there's sentimental analysis. Such tools help identify what TV shows are liked, what candidate is leading in an election … and why. [For an interesting first-step, see Teun Van Dijk's seminal work in this area, analyzing newspaper reports about the Vietnam War.]

But note!

Don't let the statistical analyses or various counts lead you to error. There are many common errors - such as "lift" (accidental co-occurrence of events) and failure to test for appropriate statistical tests, the requirements of the test, etc. Without knowledge of the domain, the data, and very careful statistical preparation, you may work against yourself.

The choice of Collection, types of queries you'll support, how you parse the data, and what kinds of algorithms you use (the domain), and especially any data reduction techniques you choose can lead to greatly helpful or greatly horrible results! If your text collection includes multiple [human] languages, then we're talking about CLIR and MLIR (cross-language IR and multilingual IR). In this case if the languages include Chinese-Japanese-Korean (CJK) or Thai, Laotian, Cambodian, Arabic and Hebrew often we focus on byte-streams (because of changes based on the linguistic environment; usually must convert all documents to UTF-16 and note that whitespace becomes a real issue when parsing!); else often the system architecture and/or the programming introduces an intermediary step (a real *n*-tier architecture).

Python has lots of great libraries to help you with text processing. Enjoy the journey and have fun!

Some references:

- Benoit, G. (2002). Data mining. In B. Cronin (Ed.), *Annual Review of Information Science and Technology* (ARIST), vol. 36, pp. 265-310. Medford, NJ: Information Today for ASIST.
- Bæza-Yates & Ribierto-Neto, Modern information retrieval, 2nd ed. New York: ACM Press
- Canfora, G., & Cerulo, L. (2004). Taxonomy of information retrieval models and tools. *J Computing & Info Tech, 12*(3), 175-194. https://pdfs.semanticscholar.org/a55c/10775dabe306c0543a46accee8e6c297ad02.pdf
- https://github.com/dipanjanS/practical-machine-learning-with-python/tree/master/notebooks/Ch07_Analyzing_-Movie_Reviews_Sentiment
- Libraries: StanfordNERTagger, nltk interface [needs Java; provides corpora]; implementation of linear chain CRF; spacy library. You may want to play with BeautifulSoup and scrappy.
- Sipser, M. (2018). *Introduction to the theory of computation*. 3rd ed.
- Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python*. O'Reilly.
- Thanaki, J. (2017). *Python natural language processing*. O'Reilly.

filename: TextProcessingForDataScieNotes.rtf