LECTURE 2

What is **indexing**? Indexing is the process of extracting features (such as word counts) from the documents (in other words: preprocessing the documents). The process ends with putting the information in a special data structure on disk in order to allow:

- fast query processing (given a query can quickly find the documents which contain all or any of the query words)
- compressed storage of the original information. After indexing we do not need the original collection any more.

Why perform indexing? Compare with the scenario where every time a query comes one needs to open all the documents and search for word occurrences in them (a bad idea).

How do we do indexing? Need to process each document. The basic steps are:

- convert to text; the file may be in a .html, .pdf or .doc format. We need only the text. Later in the course we will consider what to do the html tags and what information they provide that is useful for retrieval purposes.
- Tokenize: break the text into a stream for words
- **Stop word** removal: this step is optional. It is mainly for efficiency (= how fast the search engine will return results). Web search engines do not use it. Give examples where stop word removal will help and where it will hurt the quality of search results.
  Example where it hurts: “to be or not to be” from Shakespeare. All the words are removed because they are stop words.
  Example where it helps:

- **Stemming**: remove suffixes based on heuristic rules. Mainly for efficiency . The result is reduction in the total number of unique words considered. Stemming may help or may hurt the quality of search. Give examples where it helps and where it hurts.
  - Where it helps: most of the time when plurals are used, e.g. Schools, courses
  - Where it hurts: consider university vs. universe

  Stemming is putting words into equivalence classes. Two kinds of errors can happen:
  - two words with different stems are reduces to the same stem (university vs. universe; useful vs. us)
  - two words with the same stem are reduced to different stem

What remains from the original document after preprocessing is put in the **inverted index**. Basically, the index is list of lists. Each word points to a list which contains the documents where the word occurs. The index is stored in a file on disk. The file must allow random access. The information in the file is compressed. Note that there are other types of indexes, but the inverted index is the most common one.

The **term-document matrix**: the index is the physical representation while the term-document matrix is the logical representation of the indexed data. If there are M unique words and N documents in the corpus the term-document matrix is an M by N matrix where entry (i,j) contains how many times word i occurs in document j. This matrix is usually very sparse: 99% of the entries are zero. If you are to store that matrix on file you will store the matrix in a sparse format. Not surprisingly that format looks exactly like the inverted index. Using this matrix we will do operations on its

- columns when we want to compare or manipulate documents
- rows when we want to compare and manipulate words.

What is a **model**? An approximation of reality i.e. by definition not exact. Sometimes even clearly wrong.
What is a **retrieval model**? We want to model relevance; therefore the model needs to answer or explain the question "is a given document relevant to a particular query?".
How do we know if the model is good? The model is used to obtain predictions; in Information Retrieval the model predicts if a document is relevant to a query. To find out if a model is good, run the model (i.e. compute predictions) on a test set of queries and look at the results.

What retrieval models are there:

- Boolean (the simplest)
- Vector Space (the next simplest)
- Language model (requires basic probability to understand it)
- Okapi (also based on probability, more complicated than the language model; it was the first model that worked well and is still difficult to beat).
- KL-distance model: related to the language model, but derived from the perspective of Information Theory.
- Page rank: looks at the link structure of the web. The assumption is that a page is of high quality if many pages of high quality point to it. Not related to the concept of relevance. However, when used in combination with a text based model (Page rank is linked-based) can improve the quality of the search results.
- Many more

What is **set retrieval** and what is **ranked retrieva**l?

- Set retrieval: the model returns a set of documents, unranked; only the boolean model is an example of this category.
- Ranked Retrieval: the model ranks the documents, such that documents that match the query better are on top of the list.

Boolean model: accepts the queries that are in propositional logic, that is queries like

"INFORMATION" AND "RETRIEVAL",
"INFORMATION" OR "RETRIEVAL"
(NOT "INFORMATION") AND RETRIEVAL
NOT ((NOT "INFORMATION") OR RETRIEVAL)

Why is the boolean model good: it's simple to understand how it works, there is no uncertainty in the results, some people will intuitively distrust something that is not deterministic.
Why is the boolean model bad: requires a lot of experience and effort for the user to use the system. The boolean model is an example of a set retrieval model. However, ranked lists are easier to process by humans. Human time is precious, one should concentrate more attention on the documents that are likely to be relevant. Those are put on top of the list by the search engine.

Extended boolean model: queries also contain positional requirements about words, for example
"INFORMATION" NEAR(5) "RETRIEVAL"
Read this query as "the word information has to be at most 5 words away from retrieval" in the document.

Vector space model: the idea is that each document is represented as a vector in "word space". This means that we treat the words as the axes of the M-dimensional coordinate system, where M is the number of unique words in the collection. The documents are points (vectors) in this coordinate system. The assumption is that the angle between two vectors corresponds to the "semantic" similarity between two documents. The query is treated as a document in the sense that it also is represented as a vector in the same coordinate system. Then as a measure of the relevance of a document to a query we take the cosine between the document vector and the query vector.

Why not use the euclidean distance (i.e .the distance between two points) instead of the cosine of the angle between the two vectors? Because we need to account for document length. Let d be a document. Let d(n) be a document that consists of n copies of d pasted together. Notice that the euclidean distance between d and d(n) will grow with n. However the cosine will stay the same.

What is the idf (inverse document frequency) of a term (word)? Let df(term) be the number of documents in which a term occurs; let N be the number of documents in the collection (or corpus). Then idf(term) = log(N / df(term)).
What is the idf of a term that appears

- in all the documents: it is idf(term) = log(N / df(term)) = log(N/N) = log(1) = 0
- in most of the documents: it is low. This means that the term has low discriminating power, cannot use the term to discriminate well between documents
- in only a few documents: it is high. The term has high discriminating power.

What is the reason for tf-idf normalization? Several reasons:

- For idf: not all words are equal. Compare for example the word "the" with the word "sport". When you hear a word you can associated it with a context, for example you can associate "sport" with "baseball", "basketball" , etc. You cannot meaningfully associate "the" with anything. Idf measures the discriminating power of the word. Discriminating means one can discriminate between different documents or context on the basis of the word.
- For tf (term frequency): not all documents are of the same length. Let raw-tf(w, d) be the number of times the word "w" occurs in document "d". Then the simplest form of tf(w, d) can be defined as
  tf(w, d) = raw-tf(w,d) / (the length of document d). The length of the documents is the sum of the counts of all words that occur in the document, or to put it simply the number of total words in the document. If tf(w, d) is high the word "w" is important for the document "d" (for more explanation see also why normalize by document length below).
- Notice that tf and idf work in opposite ways. The words that will have high tf(w,d) are the stop words, because they appear frequently in any document (and in most documents). But the stop words are the ones that will have low idf. On the contrary, most of the words that appear in a few documents, generally tend to appear with low counts (in the documents where they do appear). Therefore their idf is high and there tf is low. What about words that appear a lot but in only a few documents? (see next bullet point)
- A word "w" will be important for a document d, if two things are true

    - the word w has high discriminating power; this means it has **high idf**, i.e. does not appear in many documents.
    - the word w appears many times in document d, this means it has **high tf**.
      To find the words that achieve those two goals we multiply the tf and the idf.
- Note that
    - **idf is a corpus statistic**, because we need to know the number of documents where the word appears.
    - **tf is document statistic**, because we need to have only local information about the document in which the word appears.

Why are there so many variants of the formulas for tf-idf (for example
tf(w, d) = raw-tf(w,d)/(doc_length) vs. tf(w,d) = raw-tf(w, d) / $\max_v$(raw-tf(v, d)), etc.)?
For the following reasons:

- there are many models of how information retrieval should work, each will derive its own tf-idf version (you can also read this statement as: no one knows how to do it "correctly", so every one comes with her own scheme)
- *sometimes* one version of tf-idf works better than another.

What is the reason for document normalization? Because not all documents are equal, some are longer some are shorter. What happens if a document is long? The following two things

- longer documents will contain a certain term more than a shorter document on the same topic. This results in artificially higher score of the longer document.
- A longer document will contain more distinct words than a short documents, therefore it will match more queries. If it matches more queries, then it is not very much on the topic for each of these queries, but still it may be ranked higher than a shorter document.

We need to "make documents equal", therefore we normalize for their length. See how the document normalization is implicit in the cosine formula. The cosine is the dot-product between two normalized vectors. Document normalization is also present in the tf (term-frequency) formula.

Why is a document long? For the following two reasons:

- all the time the document talks about the same thing, that is the document is verbose
- the document covers many topics, each term of the query matches a small segment. Now you can see what happens if the scoring function does not take into account how many words apart the words from the query occur. In that case, we can suffer loss of retrieval quality.

What is the relationship between the boolean and the vector space model?

- If the term-document matrix contains only zeros and ones and if the boolean model query is an OR query then a document matches a query in the boolean model if and only if the cosine between the query and the document is not 0.
- To achieve efficient implementation of the vector space model, we can run a boolean OR query, and then score only the documents in the Vector space model which satisfy the boolean OR-query.

What is a scoring function? Each IR model has a scoring function. The scoring function, denoted by Score(document, query), returns a number that means "how certain is the model that the document is relevant to the query". Typically the scoring function looks like this (please read the comments in the pseudo-code as well)

```
Function Score(document, query)
    accumulator = 0;
    For each term that appears in the query do
        w = weight(term, document) //how important is the term for the document
                                   //the weight is usually 0 if word does not
                                   //occur in the document

        accumulator += w           //evidence accumulation
    EndFor
    accumlator /= length(document) //usually the score needs to be normalized
                                   //to account for documents of different length
                                   //however some models may do the
                                   //normalization differently, normalization
                                   //may already be present in the weight(term, doc)
    return accumulator
EndFunction
```

Note: the above function is not for a particular model, it is a "template" function that has been extracted by observing how different models work. Note that is has the following components

- how should each word be weighted. "Important" words for the document should be weighted higher.
- Evidence accumulation: each word adds some evidence that the document is relevant
- document normalization: if you don't normalize, long documents will have misleadingly higher scores and short documents will have misleadingly lower scores. Note that document length may be accounted for in the weight(term, document) already.