CS410: Notes on Topics to be Covered in Exam 1

General advice

The best way to prepare for the exam is to (1) **make sure that you have watched and digested all the lecture videos**; (2) go over the quizzes you've worked on and make sure that you know how to solve all the questions in those quizzes. (3) pay special attention to the topics listed below, especially the underscored topics; (4) ask questions on Piazza if you have trouble with the materials, and email us or visit our office hours if your questions cannot be resolved on Piazza.

You are expected to know the following for Exam 1 (**underscored topics are especially important**):

1. **know the difference between text access and text mining**: can you give an example application of each?
2. **know the two modes of information access, i.e., pull mode vs. push mode**: what is the difference? can you give an example of application of each?
3. **know the two complementary ways of pull-mode information access, i.e., querying and browsing**: when is browsing more useful than querying?

4. **know the general picture of what we can do and what we can't do with today's NLP techniques**: The general picture is that it's possible to do shallow NLP (e.g., POS tagging and partial parsing) at large-scale and with reasonable accuracy, but deeper NLP (e.g., semantic analysis or even complete parsing) is generally hard and only works in limited domain or when restricted to a very special task such as entity recognition. The reason why deep NLP is difficult is because of the need for knowledge representation and inferences to precisely understand the meaning of a natural language text, and knowledge representation and inferences remain fundamental challenges in artificial intelligence. The state of the art NLP techniques tend to bypass such difficulties and rely on the application of supervised machine learning techniques to learn to do NLP tasks based on large amounts of labeled data (called training data).

5. **know what is POS tagging, what is parsing, and what is syntactic/structural ambiguity.**

   POS tagging is to assign a syntactic category (e.g., noun or verb) to each word in text. Parsing is to determine the structure of a sentence (e.g., figuring which words go together to form a noun phrase and which word modifies which other word etc). Syntactic/structural ambiguity refers to multiple possible structures of the same sentence or phrase. (Can you think of some sample sentences or phrases that are structurally ambiguous?)

6. **know what is a statistical language model, what is a unigram/bigram language model**

   A statistical language model (SLM) is a distribution over word sequences. Intuitively, it gives us a probability for any sequence of words, thus allows us to compare two sequences of words to see which has a higher probability. In general, SLMs help capture the uncertainties associated with the use of natural language. For example, in general, non-grammatical sentences would have much smaller probabilities than grammatical sentences. Specialized language models can be used to answer many interesting questions that are directly related to many information management tasks.

   While there are many different kinds of SLMs, we are particularly interested in the simplest one, i.e., the unigram language models. This model corresponds to a multinomial distribution over words. According to this model, a piece of text is "generated" by generating each word *independently*. As a result, the joint probability of generating all the words in a document D=w1 w2 ... wn is simply the product of generating each individual word, i.e., p(D)=p(w1)p(w2)...p(wn). Note that in general, the generation of one word may depend on another. For example, having seen "web search" being generated would make the probability of further generating a word like "engine" much higher. This means that p(w3="engine" |w1="web", w2="search") is much higher than p(w3="engine"). Thus the independence assumption made by the unigram language model doesn't really hold in reality. Indeed, with a bigram LM, we'd have p(D)=p(w1)p(w2|w1)p(w3|w2)...p(wn|wn-1), which would capture local dependency between two adjacent words.

7. **know the idea of maximum likelihood (ML) estimator, and how to estimate a unigram language model with an ML estimate**:

   You should know that the Maximum Likelihood (ML) estimator is to find an optimal setting of the language model p(w|theta) (i.e., optimal setting of the probability of each word in our vocabulary p(w|theta)) so that p(d|theta), or equivalently log p(d|theta), would achieve the maximum value. In other words, if we set these word probabilities to different values than their ML estimate, p(d|theta) would be smaller. The ML estimate is optimal in the sense that it maximizes the likelihood of the observed data, i.e., it finds the parameter setting that best explains the data. However, when the observed data sample is too small (e.g., the title of a document), it may be a biased representation of the entire population (e.g., the whole article), so if we overfit the observed data as the ML estimator would do, our estimated parameter values may not be optimal. For example, we would assign zero probability to all the unseen words (since the ML estimator would try to give as much probability mass to the observed words as possible in order to maximize the likelihood of data).

   You should know the fact that the ML estimate of a multinomial distribution (i.e., a unigram language model) would give each word w a probability equal to the relative frequency of the word. That is, if the word distribution is theta, and the observed data is a document d, according to the ML estimator, we would have p(w|theta)=c(w,d)/|d| where

c(w,d) is the count of word w in d, and |d| is the length of document d (i.e., total counts of words in d).

8. **know how text retrieval is different from database retrieval**

   One can make an interesting analogy between text retrieval and database search. Indeed, there are both similarities and differences between them. What's in common is the task of finding relevant information from a collection of information items. Thus both would involve a query (describing information need), a collection of data, and some criteria for selecting the results to answer the query. However, the two tasks differ in all these aspects. The data in a database has a clearly defined structure (i.e., schema), which makes it possible to specify a Boolean query that uniquely defines a set of records to retrieve. Given an SQL query, the answers are thus uniquely defined; the main challenge is to extract the specified answers as quickly as possible. The data involved in text retrieval is generally regarded as unstructured, and a query is often a fuzzy keyword query. Thus the query does not uniquely define the results. Indeed, a major challenge in text retrieval is to define which document should be in the answer set. Naturally, finding answers quickly is also necessary. But we first need to ensure accuracy before talking about efficiency. Because of the difference, the two fields have been traditionally studied in different communities with a different application basis. Databases have had widespread applications in virtually every domain with a well-established strong industry. The information retrieval community that studies text retrieval has been an interdisciplinary community involving library and information science and computer science, but had not had a strong industry base until the Web was born in early 90's. Since then, the search engine industry has emerged as a dominating industry in recent years, and as more and more online information is available, the search engine technologies (which include text retrieval and other technical components such as machine learning and natural language processing) will continue to grow. Soon we will find search technologies to be used widespreadly just like databases have been.

   Because the inherent similarity between database search and text retrieval, because both efficiency and effectiveness (i.e., accuracy) are important, and because most online data has text fields as well as some kind of structures, the two fields are now moving closer and closer to each other, leading to some common fundamental questions such as "what should be the right query language?", "how can we rank items accurately?", "how do we find answers quickly?", "how do we support interactive search?".

9. **know why ranking (without an explicit cutoff) is often preferred to selecting a subset of documents for the user.**

   The problem of TR can be formally formulated as to identify a subset of relevant documents to a query from a collection of documents. There are two strategies to implement this goal: (1) direct selection; and (2) indirect selection through ranking. In general, ranking is preferred and more fundamental because (1) ranking allows a user to control the cutoff, which is a natural way to obtain some "supervision" from the user; and (2) relevance is a matter of degree and even if we can select the right documents, it's still

desirable to rank them. As a result, most existing research in information retrieval has assumed that the goal is to develop a good ranking function. A ranking function is often designed based on some formal way of modeling relevance, which is called a retrieval model.

10. **know what is stemming, what is a stop word.**

    Stemming means to normalize lexical variations of words that have very similar or same meanings. For example, computer, computing, and computation can all be normalized into "compute". This technique generally improves recall, but sometimes, it may decrease precision. A stop word is a word that usually doesn't reflect the content of a document where it occurs; for example, functional words of English are generally stop words. Since a word that occurs in almost every document in a collection is generally not useful for search, words with very high document frequencies can often also be treated as stop words. Removing stop words is a useful retrieval heuristic to reduce the index size without affecting retrieval accuracy (that much).

11. **know what is relevance feedback, what is pseudo/blind feedback, and what is implicit feedback**

    Relevance feedback refers to the process of interactively obtaining relevance judgments from a user on some initial retrieval results and then learning from the relevance judgments to improve the query representation which can be used to retrieve more relevant documents than the original query representation. User studies have shown, however, a user is often unwilling to make such judgments, raising concerns about the practical value of relevance feedback. Pseudo feedback (also called blind/automatic feedback) simply assumes some top-ranked documents (e.g., top 10 documents) are relevant and applies relevance feedback techniques to improve retrieval performance. Thus it doesn't require a user to label documents. Pseudo feedback has also been shown to be effective on average, though it may hurt performance for some queries. Intuitively, pseudo feedback approach relies on term co-occurrences in the top-ranked documents to mine for related terms to the query terms. These new terms can be used to expand a query and increase recall. Pseudo feedback may also improve precision through supplementing the original query terms with new related terms and assigning more accurate weights to query terms.

    The difference between the two lies in how the feedback examples are obtained. There is another variant of feedback called implicit feedback where the documents viewed by a user would be assumed to be relevant while those that were skipped by a user would be assumed to be non-relevant. Documents ranked above a document clicked by a user can be assumed to be seen, but skipped by the user. The reliability of implicit feedback lies in between relevance feedback (which is most reliable) and pseudo feedback (which is least reliable).

---

12. **know the basic idea of the vector space model.**

    The basic idea of the Vector Space (VS) model is to represent both a document and a query as a vector in a high-dimensional space where each dimension corresponds to a term. The main assumption is that if document vector V1 is closer to the query vector than another document vector V2, then the document represented by V1 is more relevant than the one represented by V2. That is, relevance is modeled through similarity between a document and a query. There are three fundamental problems in this approach: (1) how to define the dimensions (i.e., term space)? (2) how to map a document/query into a vector? (3) how to measure the similarity? Different answers lead to different variants of the vector space model.

    Over decades, people have (heuristically) explored many variations of answers to all these questions. Empirical results seem to show that (1) single words are often good enough for defining dimensions, which is why single word-based indexing remains the dominant trend in retrieval and is what the current search engines are using; (2) TF-IDF weighting and document length normalization are among the most effective heuristics for computing term vectors; (3) the optimality of similarity measure highly depends on term weighting, which is not surprising as they are interacting components in the same retrieval function. Traditionally (before TREC), the most effective formula was believed to be TF-IDF plus cosine distance measure where TF doesn't involve document length normalization. In 1990's, largely due to the use of much larger data sets for evaluation in TREC, document length normalization was found to be quite important, which leads to the current version of TF-IDF weighting represented by the pivoted normalization formula and the BM25 (also called Okapi) formula. In both cases, dot-product has been found to be more appropriate than cosine measure.

13. **know the major term weighting heuristics (i.e., TF, IDF, and document length normalization).**

    TF-IDF weighting and document length normalization are the three most important term weighting heuristics. Please make sure that you fully understand them and know why these heuristics make sense. How to implement these heuristics exactly in a formula is still quite challenging and is mostly an open question. Empirically, people have found that some kind of sublinear transformation of the term frequency in a document is needed and incorporating document length normalization through the form "(1-s + s*doclength/avgdoclen)" (i.e., pivoted length normalization) is effective. While BM25/Okapi and pivoted normalization are among the most effective ways to implement TF-IDF, it remains the single most challenging research question in information retrieval what is the optimal way of implementing TF-IDF.

14. **know the formula of BM25/Okapi.**

    You should know the basic idea of the TF part of the BM25/Okapi retrieval function (you don’t need to remember the exact formula), which does a bounded non-linear transformation of the count of a word involving document length normalization. Know

the meanings of the two parameters k1 and b. k1 is the upper bound (i.e., the maximum value) of the TF formula, while b controls the influence of document length normalization. (What would happen if we set b=0?)

15. **know Zipf's law.**

    You should know the simple formula of Zipf's law, i.e., approximately Rank*Freq =Constant, i.e., the product of the rank of a word and its frequency in a collection is roughly constant. Thus, this implies that a few words (low values of rank) occur very frequently (high frequency), while many words occur rarely (low frequency, but high rank value).

16. **know what is an inverted index and how to build a large inverted index with only a limited amount of memory. know how to score documents quickly using an inverted index (i.e., how to use scoring accumulators for scoring).**

    Inverted index is the main data structure used in a search engine. It allows for quick lookup of documents that contain any given term. The relevant data structures include (1) term lexicon (mapping between a string form of a term and an integer ID); (2) document id lexicon (mapping between a string form of a document ID (e.g., URL) and an integer ID); (3) inverted index (mapping from any term ingeger ID to a list of document IDs and frequency/position information of the term in those documents).

    Indexing is the process of creating these data structures based on a set of documents. A popular approach for indexing is the following sorting-based approach:

    1. Scan the document stream sequentially. Store each document ID in the doc id lexicon. Parse each document to obtain terms and store any new term in the term lexicon.
    2. While scanning documents, we can collect term counts for each term-document pair and build an inverted index for a subset of documents in memory. When we reach the limit of memory, we write the incomplete inverted index into the disk.
    3. Continue this process to generate many incomplete inverted indices (called "runs") all written on disk.
    4. Marge all these runs in a pair-wise manner to produce a single sorted file.

    Once an inverted index is built, scoring a query can be done efficiently using the following procedure:

    5. Iterate over all terms in the query.
    6. For each term, fetch the corresponding entries in the inverted index.
    7. Create document score accumulators as needed.
    8. Scan the inverted index entries for the current term and for each entry (corresponding to a document containing the term), update its score accumulator based on some term weighting method.

9. As we finish processing all the query terms, the score accumulators should have the final scores for all the documents that contain at least one query term. (Note that we don't need to create a score accumulator if the document doesn't match any query term.)

The basic information stored in an inverted index is the term frequency. (IDF can be stored together with the term lexicon.) In order to support "proximity heuristics" (rewarding matching terms that are together), it is also common to store the position of each term occurrence. Such position information can be used to check whether all the query terms are matched within a certain window of text. In an extreme case, it can be used to check whether a phrase is matched.

17. **know the basic compression methods for integers (i.e., unary and gamma coding).**

    Another technical component in a retrieval system is integer compression, which is generally applied to compress the inverted index, which is often very large. A compressed index is not only smaller, but also faster when it's loaded into the memory. The general idea of compressing integers (and compression in general) is to exploit the non-uniform distribution of values. Intuitively, we will assign a short code to values that are frequent at the price of using longer codes for rare values. Some commonly used variable-length coding methods include unary coding, gamma-coding, and delta-coding. You should understand how unary coding and gamma coding work and be able to decode a gamma code. Since inverted index entries are often accessed sequentially, we may exploit this property to compress document ids based on their gaps. The document IDs would otherwise be distributed relatively uniformly, but the distribution of their gaps would be skewed since when a term is frequent, its inverted list would have many document IDs, leading to many small gaps. *There may be other variations of some of the coding methods, but for the exam, you should stick with the lecture slides*.

18. **know how to compute the basic retrieval evaluation measures (e.g., precision, recall, and mean average precision, NDCG, MRR, F1). You need to remember the formulas for precision, recall, average precision, MRR, and F1. You don't need to remember the NDCG formula precisely, but you should know the basic idea of this formula. You need to know the relative advantages and disadvantages of these measures (e.g., why isn't precision@10documents as good as average precision for comparing different ranking results? what's the advantage of NDCG over Mean Average Precision?)**

    Given a set of search results, the two basic measures of accuracy are precision and recall. Ideally we want to have perfect precision and prefect recall. In reality, they often compromise each other in the sense that in order to achieve higher precision we may have to sacrifice recall, and vice versa. A single point precision or recall isn't enough to measure the overall quality of *ranking*. In order to evaluate a ranked list of results, we need to plot a precision-recall curve to examine the performance at different cutoffs of ranking. To compare two systems, it is also desirable to summary the precision-recall curve with one single number. When we do this, we always have to assume some kind of

balance of precision and recall. For example, if we don't care so much about recall, we can focus more on the front-end precision or precision at low-level of recall, whereas if we emphasize more on recall, we should consider precision at high-level of recall as well. The optimal tradeoff would be user-dependent. So far, in the research literature, the "standard" measure is the mean average precision (MAP), which is the average of precisions at all points where we bring up a new relevant document. *It's very important that you know how to compute this measure*. This measure favors high recall because all missed relevant documents are assumed to have 0 precision at the corresponding (imaginary) cutoff points. Thus it may not reflect well the actual utility of a system from a user's perspective given that a user is usually looking at the first a few results in Web search. A measure such as precision at k documents can reflect the utility of retrieval results more directly, but it is not sensitive to the ranking of every relevant document (e.g., as long as we have five relevant documents in the top 10 documents, the precision at 10 documents would be the same, i.e., 0.5, no matter how these top 10 documents are ranked).

A limitation of MAP is that it assumes binary judgments of relevance (i.e., a document is assumed to be either relevant or non-relevant); in reality, it may be desirable to distinguish multi-level relevance. To address this limitation, a new measure called Normalized Discounted Cumulative Gain (NDCG) has been proposed, which can accommodate multi-level of relevance. "Cumulative" means to measure the overall utility of n documents by the sum of the gain of each relevant document. "Discounted" means to discount the gain of a document ranked low so that a highly ranked document will be "counted" more toward the gain. "Normalized" means to use the ideal ranking to compute a theoretic upper bound of the measure and then normalize the actual gain value with this upper bound. *It's very important for you to know why we need to do such normalization*. (Why didn't we need to do such explicit normalization for MAP?) NDCG is very popular for measuring the utility of Web search. However, there is no easy way to set the discounting coefficients in NDCG. When relevance is judged as a binary value, NDCG is similar to MAP. Note that in MAP, there is also a built-in discounting strategy in the sense that if you improve the ranking on the top, you'll improve MAP more than if you improve the ranking at the bottom of the ranked list (do you see why?). When there is precisely one relevant document in the entire collection, MAP becomes 1/R, where R is the rank of the single relevant document in the result list, and this measure is called Mean Reciprocol Rank (MRR). Another single-point measure of ranked results is the break-even point precision. This is the precision at the cutoff in the ranked list where precision and recall are equal. Also, F measure is also commonly used to compute a single value based on a precision value and a recall value. One variant called "F1", which is very popular, is simply the harmonic mean of precision and recall, i.e., F1=2/(1/P+1/R) where P is the precision and R is the recall.

19. **know why it is necessary to do statistical significance test when comparing retrieval results of two retrieval systems:** a system's performance varies a lot across queries, thus we need to be sure that the observed difference between two systems are not simply due to random fluctuations of performance.

20. **know the general retrieval formula of the query-likelihood retrieval method when the document language model is smoothed with a collection language model.**

    The basic idea of the query likelihood scoring method is to first estimate a language model for each document p(w|Di), and then rank documents based on the likelihood of the query given each document language model. Intuitively, a document matching more query terms would be scored higher because for such a document, the estimated language model would give a higher probability to a query term, thus also a higher likelihood for the whole query. Once a retrieval problem is defined in this way, all we need to do is to estimate p(w|Di), and different methods mainly differ in how they estimate this document language model. *Make sure that you know how to write down the **basic** query likelihood retrieval function in terms of p(w|D),* i.e., p(Q|D) = p(q1|D) * p(q2|D) * ... * p(qm|D), where Q=q1...qm is the query and p(qi|D) is the probability of a query term qi according to the estimated document language model p(w|D). A key issue here is smoothing, which is needed to prevent us from giving the query a zero probability when the document doesn't match all query terms. In general, smoothing can improve the accuracy of the estimated language model p(w|Di). There are many smoothing methods that can be applied to retrieval. So far, empirical results show that Dirichlet prior smoothing appears to work the best. However, it is still an open question what is the best way of smoothing a document language model for retrieval.

    It is interesting that if we use some general smoothing scheme that involves making the probability of an unseen word proportional to the probability of the word given by a reference language model estimated using the entire collection of documents, the query likelihood retrieval formula can be rewritten into a form very similar to BM25 or pivoted normalization retrieval formula with TF-IDF weighting and length normalization.

21. **know why smoothing is necessary when estimating a language model and know the formulas for Dirichlet prior smoothing, linear interpolation (JM) smoothing and their similarities and differences.**

    Smoothing is necessary when estimating a language model because the observed data is usually a small sample of the entire population and the maximum likelihood estimator may be biased as it would assign zero probabilities to all the unseen words. For example, if we estimate a LM based on a document, smoothing would try to assign non-zero probabilities to unseen words by taking away some probability mass from the seen words. You should remember the formulas of Dirichlet prior smoothing, and the fixed coefficient linear interpolation smoothing method (i.e., JM smoothing).

22. **know the idea and formula for Rocchio feedback.**

    In the VS model, feedback is often achieved using the Rocchio feedback method. In general, all feedback approaches try to modify the query representation based on feedback examples to obtain a presumably improved version of the query. The basic idea of Rocchio is simply to construct the new query vector (which is how you represent a query in the VS model) by moving the original query vector closer to the centroid vector

of the positive/relevant document vectors and farther away from the negative centroid. The actual formula has three parameters alpha, beta and gamma (see the slides), corresponding to the weight on the original query vector, the positive centroid and the negative centroid. These parameters need to be set empirically. You should fully understand the formula of Rocchio. (If you understand how to compute the centroid vector of a set of documents, it should be very easy to remember the Rocchio formula.) In practice, negative examples are often not very useful, so in some versions, Rocchio may involve just moving the query vector closer to the positive centroid. For the sake of efficiency, the new query vector is often truncated to contain only k terms with the highest weights. In order to avoid overfitting to the relevant examples (often a small sample), we generally need to put a relatively high weight on the original query. The relative weight of the original query vs. information extracted from feedback examples often affects feedback performance significantly. In the case of pseudo feedback, setting an optimal weight is even harder as there are no training examples to tune the weights. How to do robust and effective pseudo feedback (related to how to optimize weighting of original query terms and new terms) is another open research question in information retrieval research.

BM25/Okapi plus Rocchio (for pseudo feedback) is generally regarded as representing the state of the art performance of retrieval when we represent the documents and queries with a bag of words and have no relevance/implicit feedback data to learn from.

23. **know what is a Web crawler. Know how Web search engines handle the size of the Web.**

Web search engines are by far the most influential applications of text retrieval techniques. As an application domain of information retrieval, Web has many special characteristics that have to be considered when designing a search engine. The most important characteristic is the size and scale of the Web, raising challenges for both indexing and ranking. In early days of Web search engines, handling the size of the Web was the main challenge. A related challenge is to keep the index fresh and try to be as complete as possible. Later, spamming became a major issue to deal with, which never occurred in traditional retrieval applications such as library systems.

In order to solve the scaling challenge, Google invented several novel technologies including Google file system, which is a distributed file system that can support big files by storing them in multiple machines, Big Table, which is their column-based database system, and MapReduce, which is a software framework for doing parallel computation. Hadoop is an open source implementation of MapReduce mainly pushed by Yahoo!. Generally speaking, both indexing and ranking can be easily done through parallel processing, thus the solution generally involves using many machines to store inverted index and many machines to answer queries in parallel. The solution to the spamming problem varies according to the spamming strategy; the general strategy is to use many features to compute ranking, which makes ranking more robust against dramatic changes in only a small number of features, thus making it more difficult to spam. Ensuring the index to be as complete as possible and as fresh as possible is the job of the crawler.

Roughly speaking, a crawler is a program that goes to the Web to fetch all the web pages. We can distinguish exploratory crawling, where the crawler would follow hyperlinks and try to collect as many pages as possible, from updating crawling, where the crawler would revisit crawled pages periodically to obtain up-to-date content. How to optimize the operation of a crawler is itself an important research area. Historical observations about the Web pages can be exploited to learn how to optimize the crawling strategy.

A very important heuristic in Web search is to use anchor text, which is the text describing a hyperlink. Intuitively, the anchor text reflects how a user describes a page, thus it is likely similar to the query that the user would use to retrieve the page. Thus a query term matching the anchor text associated with a link pointing to a page provides good evidence that the page is relevant even if the page itself may not match the query term. Experiments in TREC show that rewarding matching titles and anchor text can improve search accuracy significantly.

24. **know how PageRank works and why it is helpful for Web search.**

    Intuitively, the PageRank value of a web page is a count of inlinks to the page with consideration of *indirect* inlinks pointing to the pages that point to the current page. You should know how to compute PageRank.

25. **know the basic idea of MapReduce and how it works.**

    MapReduce is a general parallel programming framework invented by Google. It is based on the Google File System (which enables easy management of large files on a cluster with multiple machines), and allows a programmer to conveniently write a program that can be run on a cluster in parallel. Specifically, a programmer would write a MAP function and a REDUCE function to direct the MapReduce system how to process the data. A MAP function takes as input a (key, value) pair, and process it to generate a set of output (key,value) pairs. In general, the key in the output is different from the key in the input. All the output pairs generated from all the MAP functions would be collected and sorted so that all the values corresponding to the same key would be grouped together. They along with the key would be passed to a REDUCE function which further processes a key together with all the collected values for that key and generates another set of (key, value) pairs as the final output.

26. **know the distinction of long term vs. short-term information need and how they can be satisfied in different ways (short-term = ad hoc retrieval; long-term= filtering).**

    There are broadly two kinds of information need: short term need and long term need. A short-term information need is temporary and usually satisfied through search/retrieval and navigation in the information space, whereas a long-term information need can be

better satisfied through filtering or recommendation where the system would take the initiative to push the relevant information to a user. Ad hoc retrieval is extremely important because ad hoc information needs show up far more frequently than long-term information needs and the techniques effective for ad hoc retrieval can usually be "re-used" for filtering and recommendation as well. Also, in the case of long-term information needs, it is also possible collect user feedback, which can be exploited. In this sense, ad hoc retrieval is much harder, as we do not have much feedback information from a user.

27. **know how basic retrieval techniques can be used to perform text filtering.**

    text filtering is a technique to filter out irrelevant information in a text document stream and select relevant information for recommendation to users. Such a technique is most useful for satisfying a user's long-term information need which can be assumed to be relatively stable (e.g., hobbies, research interests). A user can specify his/her interests by keyword descriptions or providing example documents that are known to be interesting to the user. The system maintains a set of user profiles (for multiple users). When a new document arrives, the system would generally match the document with all the user profiles and generate a score of the document for each profile. If a score w.r.t. a profile is above a threshold, the document would be delivered to the corresponding user. Scoring can usually be done by leveraging similarity/retrieval functions in a retrieval system. Once a user receives delivered documents, the user may choose to view some documents and skip others just as in the case of search. The system can thus collect feedback information from each user, which can then be leveraged to improve the scoring function for each user by using feedback techniques. The threshold is usually set to optimize the utility of the filtering decisions based on the collected feedback information. The exact utility function generally depends on specific applications and users' information need. For example, the utility function may emphasize high precision results at the price of sacrificing recall (generally delivering fewer documents) or do the opposite -- delivering more results to ensure high recall but at the price of decreasing precision.

28. **know the basic idea of collaborative filtering and how memory-based collaborative filtering algorithm works**.

    The basic idea of collaborative filtering is to use the preferences on an itme of similar users to a "current user" (also called "active user") to predict whether the current user would like the item. In the memory-based collaborative filtering method, we first normalize the ratings of items by each user by the average rating of that user. (Why do we want to do such normalization?). The predicted rating of an time by an active user is then assumed to be a weighted average of the ratings of the item by all other users, where the weight is the similarity between the active user a corresponding "other user". This similarity (between two users) is generally computed based on the correlation between the ratings of the two users, thus if two users tend to give similar ratings to the same item, their similarity would be higher.