

**Student Name:** \_\_\_\_\_

**Midterm Exam**  
Information Retrieval (11-741)  
March 6, 2008

Answer all of the following questions. Each answer should be thorough, complete, and relevant. Points will be deducted for irrelevant details. Use the back of the pages if you need more room for your answer.

The exam should take you about 70 minutes to complete. The points are a clue about how much time we think each question should take to answer. We assume about 1.5 points per minute, so a 10-minute question is worth 15 points. Plan your time accordingly.

Good luck.

1. The following are short-answer questions intended to test your familiarity with very basic IR concepts. Write the **formula** that defines the concept, and **explain** in a sentence or two why it is important.

a) inverse document frequency (idf) [5 points]

**Answer:**

$idf = \log(N / df)$ , where  $N$ =number of documents in the corpus  
 $df$ =number of documents containing this term.

idf indicates how well a term discriminates among documents. Or, you could say that idf approximates  $P(\text{term}|\text{NR})$ . Or, you could say that it is related to the entropy of the term.

I gave only partial credit for answers that said idf measures term importance, because that is too vague.  $tf$  also measures term importance. Your answer would not distinguish between them.

b) Zipf's Law [5 points]

**Answer:**

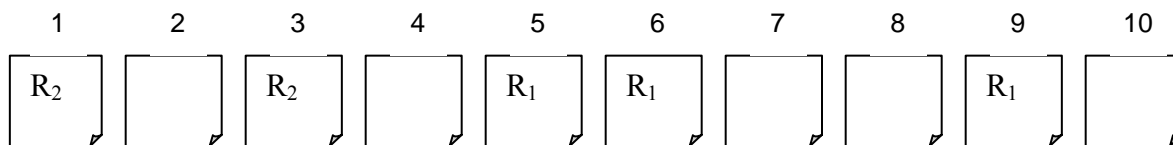
Zipf's Law:  $\text{Rank}(\text{term}_i) * \text{Freq}(\text{term}_i) = \text{Constant} \approx N / 10$  for English.

I accepted a wide variety of formulas equivalent to the above. If there was no formula, I usually deducted 1-3 points, depending upon the clarity of the text description.

Zipf's Law gives insight into the expected frequency patterns of term occurrences in a corpus. It explains that a few terms dominate the frequency distribution, and that there is a long tail of very rare terms. It enables prediction of data structure sizes. It might guide caching strategies. It might motivate term weighting in retrieval models.

We didn't cover this in lectures, so I tried to be lenient in grading this question. However, it is a basic concept. You needed to demonstrate some idea of what Zipf's Law is about.

2. Normalized Discounted Cumulative Gain (NDCG) is sometimes the preferred metric for evaluating web search engine performance because it focuses attention on the top of the rankings (e.g., first page of results), and allows multi-valued relevance assessments. Calculate NDCG for the ranking below, where  $R_i$  indicates a relevant document with value  $i$ . (Show the calculation; you do not need to produce a single number.) Assume that non-relevant documents have value 0, and ignore the normalizing constant. [5 points]



**Answer:**

If the constant is ignored, you are computing Discounted Cumulative Gain at rank 10.

$$DCG@10 = \frac{3}{\log 2} + \frac{3}{\log 4} + \frac{1}{\log 6} + \frac{1}{\log 7} + \frac{1}{\log 10}$$

There was an error in the lecture notes posted after class. Some students used that version, and got a slightly different answer. I gave credit for that version, too. If your answer differs from the above, check the new copy of the slides.

3. Web search engines usually divide their indexes into two tiers (two-tier indexing).

a) Does two-tier indexing affect search accuracy? Why, or why not? [7 points]

**Answer:**

Two-tier indexing probably improves accuracy for most queries, because the first tier contains the pages that are thought to be “high value”, i.e., pages that have high PageRank, are from sites that have high PageRank, have a low probability of being spam, have clickthrough activity, etc. Few queries are compared to the “low value” part of the index, thus they are protected from “accidentally” matching a low-value page. Queries that make it to Tier 2 probably don’t match many documents, and thus may be “exact match” queries for which ranking is less of an issue.

Some people argued that two-tier indexing has no effect on accuracy, because the second tier contains only pages that would be ranked lowly by the retrieval algorithm for most queries. My guess is that this is true in principle, but not in practice. In principle, the decision criteria that determine which tier a document goes into could also be incorporated into the ranking function. In practice, this would probably make the ranking function much more complex.

Two-tier indexing could hurt the accuracy of some queries, for example, queries where the best match (among many matches) is a page that happened to be in the second tier. However, this is probably rare.

I gave no credit for answers such as “It depends upon how well the tiers were defined” if I felt that the intent was to avoid expressing an opinion.

b) Briefly describe two ways two-tier indexing affects search efficiency. [8 points]

**Answer:**

First, the vast majority of queries are evaluated against a small index (10% of the data?). Because this index is small, several efficiencies are possible:

- fewer machines are required to store the index (lower cost per query)
- the same number of machines can evaluate queries more quickly (faster responses)
- more of the index can be cached in memory (faster responses)

Second, queries that make it to the second tier are known to have few good matches against Tier 1, thus i) they are likely to have few matches against Tier 2, and ii) they may be mistakes. These considerations allow the use of a simpler retrieval algorithm against the much larger index (90% of the data?).

Some people argued that two-tier indexing divides the queries into “fast” and “slow” groups, but I didn’t consider this compelling. Probably **all** queries are faster than if a single good retrieval algorithm is run against the entire index.

4. **Describe** how the MapReduce framework is used to create a set of inverted lists for a set of documents. Be specific about the **inputs** and **outputs** to each step. [10 points]

**Answer:**

The *input reader* takes a long list of <key, value> pairs, for example, a list of <doc\_id, doc\_contents> tuples, and divides it into *splits* of roughly equal size (e.g., 512 MB). Each split is queued up for the next available Map function.

A *Map* function receives a list of <key, value> pairs, for example, a list of <doc\_id, doc\_contents> tuples. The Map function runs a document parser that converts each document into a bag of words. The output of the Map step for one document is a list of <key, value> pairs, for example, a list of <word, (doc\_id, tf)> tuples, or a list of <word, (doc\_id, tf, locations+)> tuples. The output of the Map step for a list of documents is a list of such lists – essentially a list of partial inverted lists.

The input to the *Partition* (or shuffle) step is the output of the various Map steps. It groups all of the tuples that have the same key (word), and queues them up for the next available Reduce function.

A *Reduce* function receives a list of <key, (value+)> pairs that have been grouped by the partition step, for example, a list of <word, ((doc\_id, tf)+)> or <word, ((doc\_id, tf, locations+)+)> tuples. The Reduce function sorts the tuples for a given key (word) into a canonical order (e.g., by doc\_id) and uses them to create a complete inverted list. The output of the Reduce step is a list of complete inverted lists.

5. Document length normalization is a critical component of all retrieval models. **Describe** and **compare** the document length normalization in the **vector space**, **Okapi BM25**, and **query likelihood** retrieval models. Discuss the **characteristics** (advantages or disadvantages) of each approach to length normalization. [15 points]

**Answer:**

- The standard vector space normalizes by the length of the vector  $\sqrt{\sum x_i^2}$ . In principle, it is unbiased, but in practice it favors short documents. The standard method requires no tuning. A more sophisticated normalization is possible, e.g., pivoted length normalization, but this method has been ad-hoc, with no strong link between the guiding principle (pivoting) and the implementation.

- Okapi BM25 normalizes by  $\frac{tf}{tf + k_1 \cdot \left(1 - b + b \cdot \frac{\text{doclen}}{\text{avg\_doclen}}\right)}$ . It favors average-length documents.

This length normalization is very effective. Tuning is required to set **b** and  $k_1$ . This form of length normalization is motivated by the two Poisson model, but the only link between that retrieval model and this implementation is that they produce similar results in simple examples.

- Query likelihood uses a maximum likelihood estimate, so the length normalization is  $\frac{tf}{\text{doclen}}$ . Its length bias hasn't been studied. No tuning is required. This length normalization has a sound MLE justification. It was ineffective in earlier probabilistic models. The reasons for its effectiveness in this model are unclear; one might guess that it is due to an interaction between length normalization and smoothing, in which case tuning (of the smoothing method) *is* required. Document length can also be used to set  $p(d)$ , e.g., to favor documents of a certain length, although this is less common.

6. People often treat the query likelihood and KL-divergence retrieval models as if they were equivalent. However, they are only equivalent if a particular assumption is made. **What assumption?** Describe a retrieval task where this assumption might be valid. Describe a retrieval task where it probably is not. [10 points]

**Answer:**

It is common to treat both the query likelihood and KL-divergence models as ranking by:

$$\sum_{q_i \in Q} \log p(q_i | d)$$

The two retrieval models are equivalent under this interpretation. However, this interpretation is based on the assumption that the prior document probabilities  $p(d)$  are uniform. If one chooses to have non-uniform priors, e.g., using PageRank in a web search engine, then the two models differ:

Query likelihood:  $p(q | d) = \log p(d) + \sum_{q_i \in Q} \log p(q_i | d)$

KL divergence:  $p(q | d) = \log p(d) + \frac{1}{|Q|} \sum_{q_i \in Q} \log p(q_i | d)$

Prior probabilities have a much larger impact on the KL divergence model than the query likelihood model (for queries of length 2 or greater).

This assumption is probably valid in desktop and enterprise retrieval environments, where prior probabilities can be treated as uniform, and hence can be dropped. It probably is not valid in a web retrieval environment, where prior probabilities, e.g., estimated based on PageRank and page depth, can be very important.

Some people said that the assumption is that  $p(q_i/q)$  is uniform.  $p(q_i/q)$  is usually uniform because that is the MLE estimate for  $q$ , which is very sparse, *not* because it is a convenient assumption.

7. Some forms of search log analysis require that a person's queries be automatically grouped into "sessions" that correspond to information needs. Suppose that you are given a stream of queries  $Q_i, Q_j, Q_k, \dots$  from a single individual, and the top 30 documents retrieved for each query. Describe a solution that uses this information to identify queries that (probably) represent the same information need. **Be specific about how similarity is measured.** What types of **mistakes** will your solution make? **[15 points]**

**Answer:**

[10 points for the "how similarity is measured" and 5 points for "mistakes"]

Represent each document by a term vector, using tf.idf weights. Represent each query by the centroid (average) of the vectors for the top 30 documents it retrieved. Measure query similarity using cosine correlation. If  $\text{CosineCorrelation}(Q_i, Q_j) > \text{threshold}_s$  and if  $\text{TimeSubmitted}(Q_i, Q_j) < \text{threshold}_t$ , assume that the two queries are in the same session. You could use KL-divergence instead of cosine correlation.

This method may fail to group queries that represent different aspects of a topic, for example, "php sessions" and "php login", if the documents they retrieve have sufficiently different vocabularies. Search engine heuristics that favor diversity in search results may produce unfocused query vectors, e.g., for queries such as "jaguar", "Michael Jordan", and "Jamie Callan", which may produce weak similarity values for queries that are really similar. This method may also produce "stringy" query groups, in which  $Q_i$  is similar to  $Q_j$ , and  $Q_j$  is similar to  $Q_k$ , but  $Q_i$  and  $Q_k$  are very different. (Clustering might solve this, but we haven't covered it yet.)

Some people examined cosine similarity between query vectors. That is probably less robust than similarity of document vectors, e.g., it won't get "used cars" and "autotrader". Some people examined the number of documents retrieved in common by a pair of queries. This is probably low in most cases, i.e., high Precision and very low Recall, so it would not suffice as the main source of evidence. Some people examined the rank or score difference of documents retrieved in common by a pair of queries, which might improve Precision slightly but still be very low Recall. I deducted points for less robust solutions.

Some people worried that retrieved documents would have overlapping vocabulary, and that their query centroid vectors would be judged similar, even when the topics were fairly different. I accepted this as an answer, but I consider it unlikely, and a weak answer. It is probably only true under a very narrow interpretation of information need.

8. The PageRank algorithm models a random web surfer who can take the following actions at each time step:
- With probability  $1-\alpha$ , the user follows a link from the current page, and
  - With probability  $\alpha$ , the user jumps (teleports) to a uniformly-chosen random page on the Web.
- a) The standard PageRank algorithm computes a score for each Web page independent of the query or the user interests. Suppose that we wish to compute a PageRank score that reflects a user's interests in certain topics, e.g., Politics, Sports, and Science. How will you customize the PageRank algorithm for a user who has a specific distribution of interests, say, 50% Politics, 30% Sports, and 20% Science? Assume that you have access to a classifier that can label any Web page as one of these three categories. **[10 points]**

*Hint: Modify the "teleportation" step to reflect the fact that such a user would not jump to a uniformly random page.*

**Answer:**

In the modified Pagerank algorithm, the web surfer would jump to a Politics Web page with probability  $0.5\alpha$ , a Sports Web page with  $0.3\alpha$ , and Science Web page with  $0.2\alpha$ .

- b) What problems do you see with the application of topic-sensitive PageRank on the Web? Suggest possible solutions. **[10 points]**

**Answer:**

There are several possible solutions.

- Each Web user would have a different distribution of interest in various categories. It is computationally infeasible to apply topic-sensitive Pagerank individually for each such user.  
Solution – Apply topic-sensitive Pagerank for a few interest vectors and use them as a basis for computing personalized scores for each user.  
Note: It can be shown that the topic-sensitive Pagerank score for any user is equal to a linear combination of the topic-sensitive scores corresponding to each individual category. Hence, only  $k$  scores need to be pre-computed, where  $k$ =number of pre-determined categories.
- It might be non-trivial to determine a user's interests in various topics. Interests might even change with time.  
Solution – Allow the user to choose a set of topics he/she is interested in (this might be too coarse). Use the categories of past queries issued/web pages seen by the user to derive a distribution of interest. Limit the time window to capture the latest interests of the user.
- It is not clear what set of categories would be most helpful for representing an arbitrary Web user's interests. New categories might even emerge over time, e.g. "social networking sites" was not a category few years ago.  
Solution – Use clustering instead of a pre-determined set of categories.
- The stability and convergence properties of Pagerank might be affected by replacing uniform teleportation with an arbitrary teleportation step.  
Solution – Smoothen the user interest vector to preserve full connectivity of the graph.

c) EXTRA CREDIT: [5 points]

The update rule for standard PageRank is:

$$\vec{x}^{(k)} := (\alpha U + (1 - \alpha)M)^{k-1} \vec{z}$$

where  $U_{ij} = 1/n$  for all  $i, j$ , and  $M$  is the adjacency matrix of the Web, containing, say,  $n$  documents. Write the update rule for the topic-sensitive PageRank algorithm, given a user-interest vector  $\{p_1, p_2, \dots, p_k\}$  for  $k$  topics.

*Hint: Which matrix or vector will you have to modify in this equation to reflect the new teleportation step that you worked out in (a)?*

**Answer:**

We need to change the matrix  $U$  to reflect the new teleportation step, as follows:

$$U_{ij} = p_x, \quad \text{if document } j \text{ has category } x \quad (x \text{ is in } 1 \text{ to } k)$$

Next, normalize each row of  $U_{ij}$  so that it sums to 1.