UNIVERSITY OF OSLO

Faculty of Mathematics and Natural Sciences

Exam in: INF3800/INF4800 Search Technology
Day of exam: Tuesday June 12th, 2012
Exam hours: 14:30-18:30 (4 hours)
This examination paper consists of 4 pages
Appendices: None
Permitted materials: None

Make sure that your copy of this examination paper is complete before answering.

QUESTION 1: INVERTED INDEX

(a) Describe at least three different choices for how to keep the postings in a posting list sorted, and the purpose of each of these. How do these choices impact the ways we can traverse the posting lists when we evaluate queries?
(b) A document might have a number of different fields, sometimes also referred to as "zones". We often want to issue queries like *title:hello* if we want to retrieve only those documents that have the value *hello* in the document field named *title*. Describe at least two different choices for how you would structure the inverted index to allow for the evaluation of such queries, and outline their relative merits.
(c) The MapReduce paradigm can be used as a distributed index construction method. For such an application, what would the instantiation of the schema of the map and reduce functions look like? Provide a simple example.

QUESTION 2: HEAPS OF FUN

(a) What is Heaps' law?
(b) What is Zipf's law?
(c) Describe what γ encoding is, and how it works. When might it make sense to use this instead of variable byte encoding?
(d) Provide commented pseudo-code for the dynamic programming algorithm for computing the edit distance between two strings s_1 and s_2, not counting transposition of adjacent characters as a primitive edit operation. How would you modify the algorithm to include transposition of adjacent characters as a primitive edit operation?

QUESTION 3: LEARNING TO RANK

(a) You know how to compute hundreds of features for (*document*, *query*) pairs, but don't know how to best combine all these to yield a good ranking function. To this end you decide to have the computer learn a suitable ranking function from a set of training examples, using a support vector machine (SVM). For each (*document*, *query*) pair in your training set you also have a binary relevancy judgment. Provide a short description of how the learning algorithm informally operates, and how you would use the model to score the documents. (We are not asking for highly mathematical details of the SVM here.)
(b) What is a ranking SVM?
(c) A common evaluation measure you want your model to maximize is NDCG. Describe what this metric is, and also describe a situation where NDCG is maximized but that doesn't provide the user with a very useful result page.

QUESTION 4: NAÏVE BAYES

(a) Outline which assumptions the naïve Bayes model makes when used for text classification. For each assumption, provide at least one example where it can be violated. How might such violations impact the ability to classify correctly?
(b) Explain what smoothing is and its purpose.
(c) Consider the documents $\{d_1, d_2, d_3, d_4\}$ below. Using these as the training set for a multinomial naïve Bayes classifier, show what the trained model looks like. Use simple add-one or Laplace smoothing.
(d) Consider the naïve Bayes model developed above and the document d_5 below. Show the computations involved in classifying d_5.

d_1: chinese beijing chinese	*c = china*
d_2: chinese chinese shanghai	*c = china*
d_3: chinese macao	*c = china*
d_4: tokyo japan chinese	*c ≠ china*
d_5: chinese chinese chinese tokyo japan	*c = ?*

QUESTION 5: APPLICATIONS

Henry works as a freelance programmer, specializing in search technology. He has now been approached by *stuff.com*, a company with a site where they sell stuff online. The client wants to know if Henry can build them a system that implements a simple people-who-bought-X-also-bought-Y recommendation feature on their site, similar to what they've seen on *amazon.com*. The site has a non-trivial amount of traffic, so the implementation has to run quite efficiently. The company can supply Henry with the detailed purchase history of their customers. He decides to take the job and to use his favorite search engine to do the implementation, a search engine that among other things supports the following three features:

- Fielded search: Henry can issue a query like *foo:bar* if he wants to retrieve only those documents that have the value *bar* in the document field named *foo*.
- Multi-valued fields: A document is allowed to have fields holding lists of values, so a document with a field *foo:[bar, baz]* could be retrieved by either queries *foo:bar* or *foo:baz*.
- Faceted navigation. The result set for a query can contain distribution statistics for specified fields, computed across the full set of matching documents. E.g., along with the top 10 results for the query *foo:bar* (a query that matches more than 1000 documents), Henry can receive back a data structure like, e.g., *color:{red:897, blue:124, green:12}* that tells him that 897 of the matching documents have the value *red* in the field *color*.

Henry now ponders how he should actually implement the system, and turns to you for guidance:

(a) Suggest how Henry can achieve the implementation task using this search engine. Outline both how the index schema would look like (i.e., which fields the document in the index have and the properties these fields should have) and show how the index would be consulted to realize the desired functionality (i.e., which backend queries and operations that would be performed to populate the recommendation widget to be displayed on *stuff.com*.)

(b) Suggest how Henry can extend this recommendation feature with a filter to only considering products within the same product category as the reference item. We assume that *stuff.com* can provide Henry with category information for each product.