FINAL EXAM 2018 – SOLUTION SKETCH

CLASSIFICATION

(a) See 14.5 in textbook.

(b) See http://www.uio.no/studier/emner/matnat/ifi/INF3800/v18/undervisningsmateriale/classification-2-pierre.pdf.

(c) All answers must be justified, don't reward guessing.
- (i) False. E.g., we can introduce slack variables. (Could also be true, as "will fail" is fuzzy. Look at the candidate's justification.)
- (ii) False. We only need the support vectors, for all other examples the Lagrange multipliers $\alpha_i = 0$. (Can/should of course precompute the decision plane w from these and use this during classification.)
- (iii) False. This is in fact the basis for the SVM kernel trick.
- (iv) True. E.g., by ordering the documents according to the (signed) distance to the decision boundary. See 15.4.2 in textbook.

RELEVANCY EVALUATION

(a) Consider a result list with two breakeven points at ranks k_1 and k_2. Let R denote the total number of relevant documents for q. Furthermore, let the number of relevant documents among the first k_i documents be denoted as r_i. By definition we have that $r_1 / k_1 = r_1 / R$ and that $r_2 / k_2 = r_2 / R$ must both hold. Solutions are $k_1 = k_2$, $r_1 = 0$, $r_2 = 0$. Hence, there is only one breakeven point, unless precision and recall are 0 in which case there can be additional breakeven points.

(b) We have that $F_1 = (2PR) / (P + R)$. In the breakeven point thus $F_1 = P = R$.

(c) MAP is a single-number metric we can use to compare two rankers, given that we have a set of relevancy judgments for a set of benchmark user queries over a benchmark corpus. The AP is the average of (uninterpolated) precision values at all ranks where relevant documents are found. The MAP is then the mean of the AP values, taken over all the user queries.

QUERY EXECUTION

(a) Reward good, concise explanations. We can assume without loss of generality that s is evaluated before t.
- (i) TAAT traversal: Simple sequential scan for each term, i.e., $[p_1, \ldots, p_8]$.
- (ii) DAAT traversal: We traverse the posting lists concurrently visiting the posting having the smallest document identifier, i.e., $[p_5, p_1, \{p_2, p_6\}, \{p_3, p_7\}, p_4, p_8]$.

(b) Reward good, concise explanations. E.g.:

(i) DAAT requires that all posting lists are sorted by the same key *g(d)*. If that's not the case (e.g., impact-ordered posting lists) then we must use TAAT.

(ii) DAAT avoids the need for accumulators per document, potentially making evaluation and bookkeeping much less memory-intensive for large collections. Using less memory per lookup/search can have beneficial implications on performance (e.g., more cache hits and more data in memory) and concurrency (e.g., more queries can be evaluated in parallel.)

(iii) Both TAAT and DAAT can be used in situations where, e.g., you have a fixed time budget, although DAAT is arguably simpler since evaluated documents are completely scored when the budget is spent.

PAGERANK

(a) A surfer starts at a random page. He can either (with probability α) teleport to a randomly selected page, or (with probability $1 - \alpha$) follow one of the outgoing links from the page he's currently on. (When teleporting all nodes, including the current node, are assumed equally probably teleportation targets. When following an outgoing link all outgoing links are assumed equally probable.) Let the surfer do this forever (or a sufficiently long time), and for each page, keep track of the proportion of time that the user spends on that page. That proportion is the PageRank of the page and can be used as a static quality score *g(d)*.

(b) The "follow-an-outgoing link" transition probability matrix, i.e., the scaled adjacency matrix:

A =

0,	0,	1/1,	1/2
1/3,	0,	0,	0
1/3,	1/2,	0,	1/2
1/3,	1/2,	0,	0

The "teleport-me-somewhere" transition probability matrix:

B =

1/4,	1/4,	1/4,	1/4
1/4,	1/4,	1/4,	1/4
1/4,	1/4,	1/4,	1/4
1/4,	1/4,	1/4,	1/4

Then we have P = (1 – 0.5) * A + 0.5 * B:

P =

1/8,	1/8,	5/8,	3/8
7/24,	1/8,	1/8,	1/8
7/24,	3/8,	1/8,	3/8
7/24,	3/8,	1/8,	1/8

(c) We want the eigenvector of *P*. We can approximate this by repeatedly applying *P* to an arbitrary starting point vector *x* (whose entries sum to 1) and stop after *k* iterations or before (if we detect convergence.)

MISCELLANEOUS

(a) Reward good, concise explanations.
   (i) E.g., see Figure 6.15, page 118. A family of scoring techniques that reward a term in a document if the term occurs often in that document (TF) and also rarely appears in other documents (IDF). We multiply the TF and IDF scoring components together, and sum over all query terms.
   (ii) Equation 6.10, page 111. Imagine being in the origin, looking at the cloud of dots (vectors). Similarity between vectors is then the cosine of angle between vectors. So the distance our from origin (vector length) doesn't matter, just the angle.

(b) Reward good, concise explanations.
   (i) The wildcard query has multiple * symbols. We can use a permuterm index to find all words that start with *fi* and end with *er*, by executing the prefix search *er$fi**. But we then have to postprocess all the results to ensure that they also contain *mo* somewhere. To execute the actual search that retrieves the documents, we then form the disjunction of all terms that survived the postprocessing.
   (ii) See https://github.com/aohrn/inf3800-2018/blob/master/papers/tries-for-approximate-string-matching.pdf.

(c) Reward good, concise explanations.
   (i) The algorithm might compress well, but that alone isn't a sufficient reason for using it: If the algorithm is too slow during compression, that will have an adverse effect on indexing performance and we might not be able to meet our requirements. Similarly, if the algorithm is too slow during decompression, that will have an adverse effect on search performance and we might not be able to meet our requirements. (This question is deliberately open-ended, there can be lots of good answers here. Reward students that have well thought-out answers.)
   (ii) Pack multiple integers into a 32-bit word. Then, 4 control bits tell you how to interpret the remaining 28 bits: As one 28-bit number, or as two 14-bit numbers, and so on. (In all, 9 different ways to interpret and break up those 28 bits, sometimes with wasted bits.)