Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Marjan Celikik

Search Engines WS 09/10

http://ad.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Final Exam

Friday, March 12, 2010, 2.00 pm - 4.30 pm

General instructions:

There are six tasks, of which you can select five tasks of your choice. If you do all six tasks, we will only count the best five. Each task gives 10 points. You have two and a half hours of time overall, that is, 30 minutes per task on average. You need half of the total number of 50 points to pass the exam. The exact correspondence between the number of points and marks will be decided when we correct your exams.

You are allowed to use any amount of paper, books, etc. You are not allowed to use any computing devices or mobile phones, in particular nothing with which you can communicate with others or connect to the Internet.

In some of the tasks you are asked to write some code. You can use Java, C++, C#, Python, PHP, or pseudo-code.

Please clearly write your name (in printed letters) and your Matrikelnummer on every page of your solutions, and number the pages consecutively!

Good luck!

Task 1 (Ranking, 10 points)

Here is the first verse of an old song by the Police:

```
1.  Every breath you take
2.  Every move you make
3.  Every bond you break
4.  Every step you take
5.  Ill be watching you
```

For this task, consider each of the 5 lines of this verse as one document.

1.1 Write down the term-document matrix for this collection with simple tf scores, that is, the entry for term t and a document d is just the number of times t occurs in d. (2 points)

1.2 Normalize each column in the L_2-norm (that is, the sum of the squares should be equal to 1). Write the query `every bond you make` as a vector, and also normalize it with respect to the L_2-norm. Then rank the documents with respect to their cosine similarity to the query. If two documents have the same score, the one with the smaller document id (line number) comes first. (3 points)

1.3 Assume that documents 1, 2, 3, 4 (all lines except the last) are relevant for the query from 1.2, and assume that all five documents are presented to the user in the order you computed in 1.2. What are the precision, recall, and F-measure of this result list? What is the precision at recall 50%? (2 points)

1.4 Now consider an arbitrary document collection and the corresponding term-document matrix with tf scores like in the example above, and an arbitrary query q. Prove that if we replace each word occurrence by two copies of the same word (for example, replace line 1 by `Every Every breath breath you you take take`), then the ranking of the documents by cosine similarity to q remains the same. (3 points)

Task 2 (Compression / Entropy, 10 points)

Consider the following string with letters from the alphabet $\{a, b, c, d\}$:

```
c a c b d b c c a c c d c c b b
```

2.1 Count the number of occurrences n_σ for each letter $\sigma \in \{a, b, c, d\}$ and compute the relative frequencies n_σ/n, where $n = 16$ is the length of the string. (2 points)

2.2 Assume that the string above was generated at random with each position being filled independently of the others, and with letter σ being chosen with probability $p_\sigma = n_\sigma/n$. Compute the entropy of this distribution. (2 points)

2.3 Provide a prefix-free code for $\{a, b, c, d\}$ such that the expected code length for a single letter is at most this entropy. Write down the string from above in your encoding. How much shorter is it than writing down the string in the straightforward encoding with 2 bits per letter? (3 points)

2.4 Compute the probability that the above string is generated, for arbitrary letter generation probabilities p_a, p_b, p_c, p_d with sum 1 (assuming independence as in 2.2). Prove that this probability is maximized for the probabilities defined in 2.2, that is, for $p_\sigma = n_\sigma/n$. (3 points)

Task 3 (List intersection, 10 points)

3.1 Write down the code for a method for intersecting two lists of sorted integers, given as an array / vector each. Your method should run in time linear in the total number of elements in the two lists (that is, you can use the straighforward algorithm). (2 points)

For the following tasks, consider the following trick to speed up the intersection. Assume that the integers are from the range $1..N$, and assume that along with each input list you also have an array of m so-called skip pointers, where skip pointer i tells you the position of the smallest element in the list that is $\geq i \cdot N/m$, for $i = 0, \ldots, m-1$. If there is no such element in the list, the value of the skip pointer is the size of the list (that is, it points to one after the last element).

Here is an example, how these skip pointers can help to compute the intersection faster. Assume $N = 1000$ and $m = 10$, and you are currently at a list element with value 537 in the first list. Then skip pointer 5 from the second list tells you the position of the smallest element in the second list that is ≥ 500 and you can skip all elements in the second list until that one.

3.2 Give an example of two input lists, where the skip pointers help a lot compared to the straighforward intersection algorithm from 3.1. Explain your example. (2 points)

3.3 Give an example of two input lists, where the skip pointers do not help at all compared to the straighforward intersection algorithm from 3.1. Explain your example. (2 points)

3.4 Write down the code for intersecting two lists of sorted integers, each with a given array of skip pointers (actually, you only need the skip pointers for the second list), and for given N (note that m is just the size of the skip pointer array and so it need not be passed as a parameter, assuming that you use arrays which know their size). (4 points)

Task 4 (Edit distance, 10 points)

4.1 Compute the edit distance between the two words *orange* and *grape*, using the dynamic programming table. You don't have to prove that the scheme is correct, you can just use it. (3 points)

4.2 Using your table from 4.1, list *all* sequences of transformations of minimal length to get from *orange* to *grape*. Recall that a transformation is one of insert / delete / replace + the position in the current string to which that transformation is applied. For replace, the transformation also contains the information about the replacing letter. (2 points)

4.3 Prove that for all strings x_1, y_1, x_2, y_2, it holds that $ED(x_1x_2, y_1y_2) \leq ED(x_1, y_1)+ED(x_2, y_2)$, where $ED(x, y)$ is the edit distance between two strings x and y, and xy is the concatenation of two strings x and y. (3 points)

4.4 Give an example of strings x_1, x_2, y_1, y_2, where $ED(x_1x_2, y_1y_2) < ED(x_1, y_1) + ED(x_2, y_2)$. (2 points)

Task 5 (Classification, 10 points)

Consider the following training set of records (strings with letters a or b) each assigned to one of the classes A or B:

```
A   aba
A   baabaaa
B   bbaabbab
A   abbaa
B   abbb
B   bbbaab
```

5.1 Compute the prior probabilities $\Pr(W = a|C = A)$, $\Pr(W = b|C = A)$, $\Pr(W = a|C = B)$, $\Pr(W = b|C = B)$, $\Pr(C = A)$, and $\Pr(C = B)$ of the Naive Bayes algorithm for this training set. (3 points)

5.2 Use Naive Bayes with your priors from 5.1 to predict the class for the string *aaa* and for the string *bbb*. (2 points)

5.3 Prove that with your priors from 5.1 every string with more as than bs is predicted to be from class A while every string with more bs than as is predicted to be from class B. (2 points)

5.4 According to 5.3, each string from the training set would be predicted to be of the class stated in the training set, for example, *aba* would indeed be predicted to be of class A. Prove that this is not true in general, by modifying the training set above so that there now is a string which in the training set is assigned to, say, class A, but with the priors computed from that training set would be predicted to be of class B. (3 points)

Task 6 (Clustering, 10 points)

For this task our points will be real numbers, and the distance between two points x and y is simply their absolute difference $|x - y|$.

6.1 Execute two rounds of 2-means clustering (that is, k-means with $k = 2$) on the following seven points:

```
1  4  7  8  12  15  21
```

Each round consists of assignment of each point to its nearest cluster center, and recomputation of the cluster centers. As initial cluster centers, simply pick the smallest and the largest point. (2 points)

For the reamining subtasks consider the following alternative algorithm for dividing a given set of n points (still real numbers) into k clusters. It only makes sense in one dimension and so we call it ONE-DEE.

Consider the points in sorted order, and compute the difference from one point to the next ($n-1$ differences in total). Think of the points on a line and cut the line at the places of the $k-1$ largest differences. This divides the points into k clusters. Note that ONE-DEE is not an iterative algorithm like k-means.

Here is an example application of ONE-DEE. Consider the sequence from 6.1 and let $k = 3$. The 2 largest differences are 6 (difference between 15 and 21) and 4 (difference between 8 and 12). ONE-DEE would then produce the 3 clusters $\{1, 4, 7, 8\}$, $\{12, 15\}$, and $\{21\}$.

6.2 Prove that algorithm ONE-DEE is *scale-invariant*, that is, prove that the resulting clustering is invariant under multiplication of all points by a constant (non-zero) factor. (2 points)

6.3 Show by a counterexample that algorithm ONE-DEE does not fulfill *richness*, that is, for a fixed k not all partinioning of the input into k sets can occur as a clustering result. (2 points)

6.4 Write a method that computes the clusters according to algorithm ONE-DEE. The method should take as input the set of n points and the number of clusters k and should output the set of points for each cluster. Your program should run in $O(n \cdot \log n)$ time. It's fine to represent the sets as arrays / vectors. (4 points)