Name:	Matrikelnummer:

Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Patrick Brosi

Information Retrieval
WS 2019/2020

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

UNI FREIBURG

Exam

Thursday, February 27, 2020, 10:15 - 12:45 h, HS 082-00-006 ("Kinohörsaal") and SR 101-00-010/14

General instructions:

There are five tasks, of which you can select *four tasks of your choice*. Each task is worth 25 points. If you do all five tasks, we will only count the best four. That is, you can reach a maximum number of 100 points.

You need 50 points to pass the exam. You have 150 minutes of time overall. If you do four tasks, this is 30 minutes per task on average, plus 30 minutes buffer to double-check and think again.

You are allowed to use any amount of paper and books, except for master solutions of exercise sheets or previous exams. 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 or parallel universes.

You may write down your solutions in either English or German.

Please write your solutions on this hand-out, below the description of the tasks! You can also use the back side of the pages. Please write your name and Matrikelnummer on the top of this cover sheet in the framed box. If you need additional pages, please write your name and Matrikelnummer on each of them, too.

Important:

For the programming tasks: You can use Python, Java, or C++. None of your functions must be longer than FIFTEEN lines.

For all other tasks: Do not simply write down the final result. It should also be clear how you derived it.

Good luck!

This is the version of the exam with solution sketches. Do not distribute, it's for your personal use only. Don't use it when you do old exams for training, it's the worst way to learn. Use it only for checking your solutions, after you tried to solve the tasks yourself.

Task 1 (Zipf, Ranking, and Exponential Search, 25 points)

1.1 (5 points) Consider a document collection with the following eleven words: *bla, bli, bla, bla, bli, bli, bla, blu, bla, blu, bla*. Find C and α such that Zipf's law fits exactly for this collection.

1. The frequencies are: $F_1 = 6$, $F_2 = 3$, $F_3 = 2$
2. With $C = 6$ and $\alpha = 1$, this fits exactly $F_n = C \cdot n^{-\alpha} = 6/n$

1.2 (10 points) Write a function *ndcg(rels)* that for a given array of relevances, computes the nDCG@k, where k is the length of the array. The ith element of the array is the relevance of the ith document in the result list. The elements are: 0 for not relevant, 1 for relevant, 2 for very relevant. You can assume that $k \geq 1$ and you can use *math.log2* to compute the base-2 logarithm.

```
1.  def ndgc(rels):
2.    rels_sorted = reversed(sorted(rels))
3.    dcg = rels[0]
4.    idcg = rels_sorted[0]
5.    for i in range(1, len(rels)):
6.      dcg += rels[i] / math.log2(i + 2)
7.      idcg += rels_sorted[i] / math.log2(i + 2)
8.    return dcg / idcg
```

1.3 (10 points) Consider an array A with integer values in ascending order and an exponential search for value x starting from $A[0]$. Let i be the position of the first element in A that is strictly larger than x. The search stops when an element strictly larger than x is encountered. You can assume that A has at least $2i$ elements, so that the exponential search does not jump beyond the last element of A. Provide an exact formula for the number of jumps in terms of i and prove it.

1. Let k be the number of jumps in the exponential search
2. Then the kth position checked is $0+1+2+\ldots+2^{k-1} = 2^k-1$, hence $2^k-1 \geq i \Leftrightarrow k \geq \log_2(i+1)$
3. The position before that is $2^{k-1} - 1$, hence $2^{k-1} - 1 < i \Leftrightarrow k - 1 < \log_2(i+1)$
4. It follows that $k = \lceil \log_2(i+1) \rceil$

Task 2 (Encodings and UTF-8, 25 points)

2.1 (5 points) Consider the following sequence of symbols: A, B, A, A, B, C, A, D. Find an entropy-optimal encoding for the four symbols and write down the corresponding encoding of the entire sequence. Use the relative frequencies of the symbols in the sequence to determine their probabilities.

1. The probabilities are: $Pr(A) = 1/2$, $\Pr(B) = 1/4$, $\Pr(C) = 1/8$, $\Pr(D) = 1/8$
2. An entropy-optimal encoding would be: $A = 0$, $B = 10$, $C = 110$, $D = 111$
3. The corresponding encoding of the whole sequence would be: 01000101100111

2.2 (10 points) For a given probability distribution $p_1, \ldots, p_n$, consider all codes such that the code lengths satisfy exactly $\sum_{i=1}^{n} 2^{-L_i} = 1$. Prove that for each of these codes, the expected code length is at least the entropy. If you use Lagrangian multipliers, you can assume without proof that your function has no saddle points and that all local optima are minima.

1. Define $L = \sum_{i=1}^{n} p_i \cdot L_i + \lambda \cdot (\sum_{i=1}^{n} 2^{-L_i} - 1)$
2. Then $\partial L / \partial L_i = p_i - \lambda \cdot \ln 2 \cdot 2^{-L_i}$, which is 0 for $p_i = \lambda \cdot \ln 2 \cdot 2^{-L_i}$
3. Summing on both sides, we get $1 = \lambda \cdot \ln 2$
4. Hence $p_i = 2^{-L_i}$ or equivalently $L_i = \log_2(1/p_i)$
5. Hence we have for the expected code length that $\sum_{i=1}^{n} p_i \cdot L_i \geq \sum_{i=1}^{n} p_i \cdot \log_2(1/p_i) = H(X)$

2.3 (10 points) The Unicode code point for the smiling face with sunglasses, written in hexadecimal, is 1F60E. Use this information to write down the UTF-8 code in hexadecimal. Do not only write down the end result, but also your intermediate steps. Note that hexadecimal can be easily translated to binary and vice versa (each hex digit corresponds to four bits).

1. Written in binary (four bits per hex digit), 1F60E is 1 1111 0110 0000 1110
2. We have 17 significant bits and hence need a four-byte UTF-8 code
3. The template for the code is 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
4. Filling in the code point, we get 1111 0000 1001 1111 1001 1000 1000 1110
5. Converting back to hexadecimal, we get F0 9F 98 8E

Task 3 (Web Applications, 25 points)

3.1 (10 points) Write valid HTML for a web page with a heading, an input field, and an empty paragraph. The heading should be colored, using CSS. The input field and the paragraph should have an ID. The HTML should include a JavaScript file, to be written for task 3.2.

```
1.   <html>
2.     <head>
3.       <style>h1 { color: darkblue }</style>
4.       <script src="webapp.js"></script>
5.     </head>
6.     <body>
7.       <h1>Title</h1>
8.       <input id="query"></input>
9.       <p id="result"></p>
10.    </body>
11.  </html>
```

3.2 (10 points) Write JavaScript with the following functionality. After the page has loaded, write the message *Please enter your query* into the paragraph. Then, after each character typed by the user in the input field, send the content of the input field to a backend, using a GET request with a URL and port of your choice. You can assume that the backend returns a list of matching items in a JSON array. Display these matches in the paragraph, as a comma-separated list. Protect yourself against code injection (you can assume that for non-malicious responses, the elements of your array are strings composed of A-Z, a-z and spaces). You can use *jQuery* if you want.

```
1.  document.addEventListener("DOMContentLoaded", function() {
2.    $("result").html("Please enter your query")
3.    $("query").keyup(function() {
4.      var url = "http://backend.org:8888/?query=" + $("query").val();
5.      $.get(url, function(result) {
6.        $("result").html(result.join(", ").replace(/[^A-Za-z, ]/g, ""))
7.      })
8.    })
9.  })
```

3.3 (5 points) When the user typed *bla*, what is the first line (until the first newline) of the last GET request sent to the backend. If that GET request succeeded, what is the first line of the response and how does the line specifying the content type look like.

1. The first line of the GET request is: *GET /api?query=bla HTTP/1.1*
2. The first line of the response is: *HTTP/1.1 200 OK*
3. The line specifying the content type is: Content-Type: application/json

Task 4 (Vector Space Model and Naive Bayes, 25 points)

4.1 (5 points) Let A be a 5×7 term-document matrix, where the columns correspond to the documents. Let C be a 5×2 matrix, where the first column is the average of the first three documents and the second column is the average of the last four documents. Specify a matrix B, such that $A \cdot B = C$.

$$B = \begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix}^T$$

4.2 (10 points) Write a function *smooth(pwc, eps)* that takes as input the p_{wc} learned during training by Naive Bayes (as an array of k arrays, one for each of the k classes) and a positive real value *eps*. The function should modify the p_{wc} such that afterwards the following properties are fulfilled: (1) all probabilities are non-zero, (2) the relative order of the probabilities from the same class is maintained (that is, if for any two probabilities from the same class, one is smaller than the other, this is still the case after the modification), (3) the probabilities for each class are still a probability distribution. Argue how your function fulfills these properties.

```
1.  def smoothing(pwc, eps):
2.    for pw in pwc:
3.      for i in len(pw):
4.        pw[i] += eps
5.      sum_pw = sum(pw)
6.      for i in len(pw):
7.        pw[i] /= sum_pw
```

4.3 (10 points) Write a function *predict(doc, pwc, pc)* that takes as input a document (given as an array, where *doc[i]* contains the frequency of the ith word from the vocabulary), and the probabilities p_{wc} (like in 4.2, now all non-zero) and p_c (as a one-dimensional array) from the training of Naive Bayes. The function should predict the most likely class using a suitable matrix-vector multiplication. You can simply use * to multiply a matrix given as a two-dimensional array with a vector given as a one-dimensional array, provided that the dimensions match. You can use *argmax* to compute the index of the largest value in a one-dimensional array.

```
1.  def predict(doc, pwc, pc):
2.    doc.append(1)
3.    for i in len(pwc):
4.      pwc[i].append(pc)
5.      for j in len(pwc[i]):
6.        pwc[i] = math.log(pwc[i])
7.    predictions = pwc * doc
8.  return argmax(predictions)
```

Task 5 (SVD and Knowledge Bases and POS-Tagging, 25 points)

5.1 (5 points) Let $A = U \cdot S \cdot V$ be the singular value decomposition of A, where S is an $r \times r$ diagonal matrix and r is the rank of A. Which of the following two products always gives an identity matrix and why and why not the other: $U^T \cdot U$ or $U \cdot U^T$?

1. The columns of U are the normalized eigenvectors of $A \cdot A^T$
2. Eigenvectors are pairwise orthogonal, hence $U^T \cdot U = I_r$
3. If U has m rows with $m > r$, we cannot have $U \cdot U^T = I_m$ because U has only rank r

5.2 (10 points) In the course, we have seen an algorithm for translating SPARQL queries to SQL queries. Write a function *num_sql_conditions(sparql_query)* that computes the minimal number of conditions (after the *WHERE*) in the corresponding SQL query. The *sparql_query* is given as an array of triples, where each element of each triple is either a variable (a string starting with *?*) or an explicit predicate or literal (a string not starting with *?*).

```
1.  def num_sql_conditions(sparql_query):
2.    var_counts = {}
3.    num = 0
3.    for triple in sparql_query:
4.      for x in triple:
5.        if x.startswith("?"):
6.          var_counts[x] = 1 if x not in vars else var_counts[x] + 1
7.        else:
8.          num += 1
9.    for var in vars:
10.     num += var_counts[var] - 1
11.   return num
```

5.3 (5 points) Prove or provide a counterexample: Let D be a finite set and let $\mathbf{R}_0^+$ be the set of all non-negative real numbers. Then for each pair of functions $f, g : D \to \mathbf{R}_0^+$, it holds that $\max\{f(x) \cdot g(x) : x \in D\} = \max\{f(x) : x \in D\} \cdot \max\{g(x) : x \in D\}$.

1. Let $D = \{1, 2\}$ and $f(1) = 5$, $f(2) = 2$ and $g(1) = 2$, $g(2) = 5$
2. Then $\max\{f(x) \cdot g(x) : x \in D\} = 10$
3. But $\max\{f(x) : x \in D\} \cdot \max\{g(x) : x \in D\} = 5 \cdot 5 = 25$

5.4 (5 points) In our application of the Viterbi algorithm to POS-tagging, we introduced the *END* tag. Was this merely a convenience to make the implementation easier or does it influence the result? Give a concise explanation.

1. Along with the *END* tag, we also introduced transition probabilities from the last hidden state
2. These transition probabilities were also learned from data
3. They reflect that some hidden states are more likely as the final hidden state of a sentence
4. So yes, the *END* tag (and the associated transition probabilities) influence the result