Data & Information – Test 3: Solutions

29 May 2019, 13:45–14:45

Question 1: Information Retrieval / Full-text Search (35 points)

Suppose we have the tables below for a simplified system for storing reviews of mobile apps. The table 'app' contains and identifier (appid), name and description of all apps and the table 'review' contains the reviews for these apps storing an identifier (revid), the review itself (review), the user who wrote the review (user) and a reference to which app the review belongs (forapp).

```
CREATE TABLE app (
        appid INTEGER,
        name TEXT,
        description TEXT,
        PRIMARY KEY (appid)
)
```

```
CREATE TABLE review (
        revid INTEGER,
        user TEXT,
        review TEXT,
        forapp INTEGER,
        PRIMARY KEY (revid),
        FOREIGN KEY (forapp) REFERENCES app(appid)
)
```

a) [FT-search] Write an SQL query that uses the full-text search capabilities for finding all apps that have a description that matches "todos | todolist".

-
```
SELECT *
FROM app
WHERE to_tsquery('english','todos | todolist')
        @@ to_tsvector('english',name)
```

b) [FT-search] Write an SQL query that uses the full-text search capabilities for finding all apps that mention the word "todo" in their description or in one or more of the reviews of that app. Use the matching operator only once in the query, i.e., match against the combined text of an app's description and all its reviews.

-
```
SELECT appid, name
FROM app, review
WHERE review.forapp = app.appid
AND to_tsquery('english','todo') @@ to_tsvector('english',
        string_agg(review.review) || app.description)
GROUP BY appid, name
```

c) Which of the following statements is true; explain your answers

I. A "posting list" is a list of terms.

II. A search engine is based on an index-ask-match-rank pattern whereby the indexing and matching tasks happen off-line, and the asking and ranking tasks on-line (i.e., at the moment the user poses the query).

III. To convert a string to a tsvector, it is better to use to_tsvector instead of casting with '::', because to_tsvector also does normalization and stemming and such while casting does not.

- i: FALSE; it is a list of documents or document identifiers.
  ii: FALSE; only indexing happens off-line; the other three on-line.
  iii: TRUE

d) [IR; tf.idf] Given a table with quotes as illustrated below (filled with tweets from a search for #coffee on Twitter), calculate the tf.idf scores for the query 'drink coffee happy'. Explain your answer by giving the full calculation. The formulas for ranking and idf are given below the table. If you don't have a calculator for computing log, just invent a number between 0 and 1 (explicitly mention that you did this!). The terms of the query are given in bold for your convenience (we assume 'stemming', etc. so the words "drink" and "drinking" are both considered the same term "drink", and the words "#coffee" and "coffee" are both considered the same term "coffee")

Qid	User	Quote
1	@metalmamalady	Good Morning, tweets! #**Coffee**
2	@mohbasmah	The best barista in Jeddah #brista #**coffee**
3	@cazij	**Happy** Friday ! #**coffee** helps
4	@julianna_glass	Your living is determined not so much what life brings to you as by the attitude you bring to life. #TuesdayThoughts #**coffee** helps Wonderful day to all!!
5	@jonlalas	More #**coffee** please
6	@nationalcoffee	"First, **drink** more #**coffee**" - great advice from @Inc (and backed by science)
7	@lisalisnyr9	I'm **drinking** my **coffee** right now! Mmmmmm **coffee**!! #GoodMorning #HappyFriday #**Coffee** #ILoveCoffee
8	@ms_coffeelover	When the first sip of #**coffee** finally kicks in.

$$Idf(t) = \log (N/df)$$

$$Rank(d,q) = \sum_{t \in q} tf(d,t)\ idf(t)$$

- "drink"
  idf = log(8/2) = log(4) = 0.6
  tf is 1 for documents 6 and 7; otherwise 0

  "coffee"
  idf = log(8/8) = 0
  tf is 3 for document 7; otherwise 1

  "happy"
  idf = log(8/1) = 0.9
  tf is 1 for document 3; otherwise 0

  Ranking:
  0*0.6 + 1*0 + 1*0.9 = 0.9 for document 3
  1*0.6 + 3*0 + 0*0.9 = 0.6 for document 7
  1*0.6 + 1*0 + 0*0.9 = 0.6 for document 6
  0*0.6 + 1*0 + 0*0.9 = 0 for documents 1, 2, 4, 5, 8
  Hence the most relevant document is document 3

e) [IR; language models] Calculate P(drink coffee happy | document 3) and P(drink coffee happy | document 7). Give the full calculation to explain your answer.

- P(drink coffee happy | D_3) =
  = P(drink| D_3) * P(coffee| D_3) * P(happy| D_3)
  = 0/4 * 1/4 * 1/4 = 0
  P(drink coffee happy | D_7) =
  = P(drink| D_7) * P(coffee| D_7) * P(happy| D_7)
  = 1/12 * 3/12 * 0/12 = 0

f) [IR; language models] With language models, the actual ranking is done on $P(D|T_1…T_n)$ and not on $P(T_1…T_n|D)$ as we have calculated above. We can calculate the former based on Bayes rule which includes two other factors:
Bayes rule is $P(D|T_1…T_n)= P(T_1…T_n|D)P(D) / P(T_1…T_n)$
There are two other factors in this equation P(D) and $P(T_1…T_n)$. Which of the two factors P(D) or $P(T_1…T_n)$ makes it that the term "coffee" is weighed less than "drink"? Explain your answer.

- $P(T_1…T_n)$.
  "coffee" is a very often occurring term in this collection, perhaps also in the queries, definitely a more likely term than "drink". Therefore, P("coffee") > P("drink"). Since, probabilities are between 0 and 1, dividing by a larger number (still less than 1 but closer to 1) will produce a smaller result than dividing by a smaller number.
  P(D) is a prior probability for the relevance of a document; this has nothing to do with terms.

Question 2: REST (30 points)

During lecture WP3 we presented the following method that returns an XML document containing one `<hello>` element:

```
@GET
@Produces(MediaType.TEXT_XML)
public String sayXMLHello() {
  return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
}
```

Now suppose that we have a class `Todo` with the following definition:

```
public class Todo {
  private String summary;
  private String description;

  public Todo(){
  }
  public String getSummary() {
    return summary;
  }
  public void setSummary(String summary) {
    this.summary = summary;
  }
```

```
  public String getDescription() {
    return description;
  }
  public void setDescription(String description) {
    this.description = description;
  }
```

a. (15 points) How can we return an object of this class represented in XML as a response to an HTTP request in a method called `getXML()`, without defining the whole XML representation as `String` as we did in method `sayXMLHello()`? Explain what you need to do with the `Todo` class and with the `getXML()` method, and why this works. Don't forget to explain what should be done inside method `getXML()`!

Suppose now that you have to implement a RESTful service that manages a collection of user profiles, where each profile has a unique id (a string representation of an integer value), a nick name and a list of postings. Each posting has an id (a string representation of an integer value), a title and a body.

b. (15 points) Which elements should be given in an HTTP request message (HTTP method, URL and HTTP message body) to create a profile with id "1099" for "crazyRabbit" and one posting with id "123", title "My first posting" and body "Hello world"? Explain your answer, also giving an XML representation of this profile that could be sent in the body of the HTTP request message.

Answers:

a. In order to return an XML representation of the object we first need to annotate the Todo class with `@XmlRootElement` so that JAXB knows that this class will be serialized in XML. This means that whenever the code refers to an instance (object) of the Todo Java, XML serialisation and parsing is done automatically. Method `getXML()` should be annotated with `@GET` and `@Produces(MediaType.APPLICATION_XML)` to indicate that this method should be used to respond to an HTTP `GET` message that asks for XML representation. Finally, the object has to be created in the `getXML()` method and its attributes need to set and returned by this method.

b. The HTTP request to refer to the URL `http://somedomain/profiles/1099`, HTTP method should be PUT (or POST) and the HTTP message body should contain a representation of the profile according to the resource representation supported by the web service.

This representation could be the following:

```
<profile>
  <id>1099</id>
  <name>crazyRabbit</name>
  <postings>
    <posting>
    <id>123</id>
    <title>My first posting</title>
    <body>Hello World</body>
  </postings>
</profile>
```