Information Retrieval

WS 2015 / 2016

Lecture 13, Tuesday February 2\textsuperscript{nd}, 2016

(Knowledge Bases, SPARQL, Translation to SQL)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg
Overview of this lecture

- **Organizational**
  - Your experiences with **ES12** (Statistical Significance)

- **Content**
  - Statistical tests again some clarifications
  - Knowledge bases motivation + examples
  - SPARQL standard query language for KBs
  - SQLite lightweight database software
  - SPARQL to SQL algorithm + example
  - Performance making it fast

- **Exercise Sheet 13**: Implement SPARQL → SQL translation and use to process SPARQL queries with Python+SQLite
Experiences with ES12 1/3

Summary / excerpts

- Topic + exercise was interesting and useful
- Not too much work
- Technical details behind the T-Test were not clear
  I skipped this part for time reasons and the explanation on
  the slides were sup-optimal ... see slides 6 – 11
- Why two-sided test and not one-sided test
  Good question ... it was discussed on the forum
- p-value very small for the large test set ... see next slide
Experiences with ES12  2/3

Results

- Most improvements are by a few percent, for example:
  
  74% → 80%, 66% → 68%, 69.8% → 73.2%, ...

- Sometimes, the changes even make the results worse
  
  A common research experience when trying to improve stuff

- The p-values for 50 or 200 examples are very large
  
  Understand that this does not mean that the variant is not really better ... it means that the evaluation does not prove it (the difference might as well be due to random fluctuations)

- The p-values for 3140 ex. is small, even for small differences
  
  For example, p = 0.2% for a 3.4% difference
Experiences with ES12  3/3

Bottom line

- With few measurements (for ES12: tens or hundreds), even medium improvements are hardly significant

- With many measurements (for ES12: thousands), even small improvements can be significant

- Understand: statistical tests never show that the hypothesis (our actual interest) is likely

For ES12: we have proven nothing about the likeliness that the sophisticated Perceptron is better than the baseline

They only ever estimate how unlikely the null hypothesis is

For ES12: that the difference in precision is due to chance
Mathematics behind the Z-Test and T-Test

- I decided to skip that in the last lecture

  It's mathematically more demanding than what we did so far, and there was not too much time available
  
  When preparing the lecture, I tried to take away as much as possible from the complexity ... but not very successfully so, it's still relatively hard

- Bottom line: you do not need to know the mathematical details behind the Z-Test and T-Test for the exam

  The rest of what we did in Lecture 12 is of course relevant for the exam though
Biased vs. unbiased estimators

- Let \( X_1, ..., X_n \) be independent identically distributed random variables with mean \( \mu \) and variance \( \sigma^2 \)

- Since we don't know the underlying \( \mu \) and \( \sigma^2 \), we estimate them as follows

\[
M = \frac{\sum X_i}{n} \quad S^2 = \frac{\sum (X_i - M)^2}{n}
\]

- Mathematically, it seems reasonable to ask that these estimates do the right thing "on average", namely:

\[
\mathbb{E} M = \mu \quad \mathbb{E} S^2 = \sigma^2
\]

- With the definitions above, this is indeed true for \( M \), but for \( S^2 \) as defined above: \( \mathbb{E} S^2 = (1 - 1/n) \cdot \sigma^2 \)
Biased vs. unbiased estimators, continued

- Alternatively, we can define $S^2 = \frac{\sum (X_i - M)}{n - 1}$
- This estimate is called **unbiased** because $E S^2 = \sigma^2$
- In practice, the unbiased estimator is used more often

  It's not wrong to use the biased estimator though ... in the last lecture, I chose it because of simplicity

  Also note that for large values of $n$, the difference between the two (a factor $1 - 1/n$) is negligible

- For similar reasons, one often subtracts 1 from the number of measurements (per sequence) for the T-Test

  For our example from Lecture 12: $n - 2$ instead of $n$
Mistake on slide 31 of last lecture

- We correctly computed the estimates $\sigma_1^2$ and $\sigma_2^2$ of the variances of the two series of measurements

  We computed unbiased estimates, but that was ok

- Then we computed the total variance as $\sigma^2 = \sigma_1^2 + \sigma_2^2$

  That was a mistake, we should have computed the total variance as the average $\sigma^2 = (\sigma_1^2 + \sigma_2^2) / 2$

Then the value for $x$ becomes larger (by a factor of $\sqrt{2}$) and the (two-sided) p-values becomes smaller:

**Z-Test:** $\sigma^2 = 1.5 \rightarrow x = 2.3094 \rightarrow p = 2.1\%$

**T-Test:** $\sigma^2 = 1.5 \rightarrow x = 2.3094 \rightarrow p = 5.0\%$
Intuition of the x-value

- Recall the values from the previous slide:
  
  **Z-Test:** \( \sigma^2 = 1.5 \rightarrow x = 2.3094 \rightarrow p = 2.1\% 
  
  **T-Test:** \( \sigma^2 = 1.5 \rightarrow x = 2.3094 \rightarrow p = 5.0\% 
  
  - Understand that x and p have the following "meaning":
    
    **p:** the probability that what we see happened by chance
    
    **x:** what we see is as (un)likely as a random variable from
    the assumed distribution deviates by x times or more
    the standard deviation from its mean
    
    one-sided test: deviation in one direction
    
    two-sided test: deviation in either direction
R-Test vs. Z-Test and T-Test

- For the example in the last lecture, the R-Test had a much larger p-value (18%) than the Z-Test or T-Test.

- Reason: the Z-Test and T-Test make assumptions on the underlying distribution. These assumptions become more and more reasonable for large n but can be quite unrealistic for small n.

- However, the R-Test requires (extensive) computation. In the old days, that was simply not feasible. Nowadays, with ubiquitous access to computers, the R-Test is the method of choice.
Knowledge Bases 1/4

Definition

- A knowledge base is a database of statements about entities and their relations

  Critical: **unique** identifiers for each entity and relation

- A common format / schema is to express all statements as **subject predicate object** triples:

  Brad Pitt  acted in  Mr. and Mrs. Smith
  Brad Pitt  acted in  Burn After Reading
  Angelina Jolie acted in  Mr. and Mrs. Smith
  Joel Cohen  directed  Burn After Reading
  Ethan Cohen  directed  Burn After Reading
  Brad Pitt  married to  Angelina Jolie
Freebase and WikiData

- Freebase is the largest open general-purpose KB to date
  
  Started by Metaweb in 2007, acquired by Google in 2010
  
  Current size: \( \approx 3 \text{ billion} \) triples on \( \approx 50 \text{ million} \) entities
  
  Freebase has become read-only in March 2015 and most of its data will eventually be merged into WikiData

- WikiData is the soon-to-become largest open general-purpose knowledge base to data
  
  WikiData is the "Wikipedia" among the knowledge bases
  
  Current size: \( \approx 80 \text{ million} \) triples on \( \approx 20 \text{ million} \) entities
Knowledge Bases 3/4

- Reification

  - Restriction to triples is no real restriction: n-ary relationships can also be represented as triples:

    m/0jy6xg  film  Finding Nemo
    m/0jy6xg  actor  Ellen DeGeneres
    m/0jy6xg  character  Dory
    m/0jy6xg  type  Voice

    m/0jy6xg is an entity name from Freebase

    In the example above, it's a so-called mediator, which serves as a link between the entities it connects
Relation to the "Semantic Web"

- The Semantic Web initiative is concerned with making knowledge base data **explicitly** available on the web

  **Variant 1:** semantic mark-up in normal web pages
  Typical format: Microdata or JSON-LD

  **Variant 2:** web pages containing only structured data
  Typical format: RDF

- No rules that enforce consistent entity or relation names

  The hope is that people adhere to standards nevertheless, and that machines can resolve the remaining heterogeneity

Anyway: this is **not** the topic of this lecture / course
Definition

- The standard query language for knowledge bases

**SPARQL** = **SPARQL Protocol And RDF Query Language**

- Example query in natural language: actors who are married and starred together in at least one movie

- The same query expressed in SPARQL

```sparql
SELECT ?person1 ?person2 ?film WHERE {
  ?person1 acted_in ?film .
  ?person1 married_to ?person2
}
```
Syntax

- In the lecture today, we use a simplified syntax

  See the example from the last slide

- The actual SPARQL syntax is slightly more complicated and has many more features

  In particular, it involves namespaces, so that names can be made globally unambiguous

  See the Wikipedia page or the W3C specification if you are interested

  Not relevant for our lecture today
SPARQL queries as subgraphs

- On can view a knowledge base as a graph, where the nodes are the entities, and the edges are the relations.
- A SPARQL query is then a sub-graph with variables at some or all of the nodes.
- Solving the query then amounts to finding all matches of the subgraph in the (large) knowledge base graph.
Relation to SQL

- Data from a KB can be stored in an ordinary database
  This is also what we do in the lecture and for ES13
- The standard query language for databases is SQL
  SQL = Structured Query Language
SQL example

- Assume we have two tables film (with columns actor and movie) and spouse (with columns person1 and person2)

- Our example query can be expressed in SQL as follows:

```sql
SELECT spouse.person1, spouse.person2
FROM spouse, film as film1, film as film2
WHERE spouse.person1 = film1.actor
AND spouse.person2 = film2.actor
AND film1.film = film2.film;
```
SQLite 1/4

- A full-fledged database, easy to install and use
  - On Debian/Ubuntu install with: `sudo apt-get install sqlite3`
  - Two types of commands ... examples on next slides
    - SQL commands: must end with a semicolon
    - SQLite commands: start with a dot, no semicolon at end
  - Two modes to start SQLite:
    - `sqlite3` will work on an in-memory database
    - `sqlite3 <name>.db` create database in that file, and if file exists, use database from that file

Let's read our example tables in SQLite using the commands from the next two slides ... it's easy
Some useful SQLite commands by example

- Specifies the column separator used for input and output
  `.separator " "` use Ctrl+V TAB for TAB !

- Read table from TSV (tab-separated values) file
  `.import film.tsv film`

- Execute commands from script file (typical suffix is .sql)
  `.read <file with commands>`

- Show execution time of every command
  `.timer on`
Some useful SQL commands by example

- Create a table with a given schema
  
  CREATE TABLE film(actor TEXT, movie TEXT);

- Create an index for a column of a table
  
  CREATE INDEX file_index ON film(actor);

- Extract / combine data from tables
  
  SELECT * FROM film WHERE ... LIMIT 100;

- Delete table / index (without error msg if it's not there)
  
  DROP TABLE IF EXISTS film;
  DROP INDEX IF EXISTS film_index;
Python interface to SQLite

Executing SQL commands to a SQLite database from within Python is very easy:

```python
import sqlite3
db = sqlite3.connect("example.db")
cursor = db.cursor()
cursor.execute("SELECT * FROM table")
for row in cursor.fetchall():
    entries = [str(entry) for entry in row]
    print("\t".join(entries))
```

Beware: the SQLite command (starting with a dot) cannot be executed from within Python, you need SQLite for those
Motivation

- We want to translate a given SPARQL query to a SQL query that gives the desired results on a given database
- In the following example, we use one table per relation

```
CREATE TABLE film(actor TEXT, film TEXT)
CREATE TABLE spouse(person1 TEXT, person2 TEXT)
```

Note: all elements from one table are from one relation, so we don't need to store the relation name in the table

For ES13, use **one big table** for all the data, with three columns named **subject, predicate, object**
Example

- SPARQL query

```
SELECT ?p1 ?p2 ?f
WHERE {
  ?p1 spouse ?p2
}
```

SQL query:
```
SELECT spouse.partner1, spouse.partner2,
FROM spouse, film AS $1, film AS $2
WHERE spouse.partner1 = $1.actor
AND spouse.partner2 = $2.actor
AND $1.film = $2.film
```
Algorithm

- It is up to you in ES13, to design a generic algorithm that works for arbitrary basic SPARQL queries

Of the form SELECT <vars> { <triples> }

- The algorithm is not difficult, but requires understanding of how the data is stored and SPARQL and SQL work

So perfect exercise to understand the basics!

- On the next slide we give you valuable advice
Algorithm, advice for ES13

- If there are $k$ query triples in the SPARQL query, have $k$ entries in the `FROM` clause of the SQL query
  
  ```sql
  FROM freebase as f1, freebase as f2, ... , freebase as fk
  ```

- In your code, for each variable from the SPARQL query, build an `array` of all its occurrences in the query, e.g.
  
  ```sql
  ?x:  f1.subject, f2.object, f5.object
  ```

- Then, when building the SQL query, add the corresponding equalities to the `WHERE` clause, e.g.
  
  ```sql
  WHERE  f1.subject = f2.object AND f2.object = f5.object
  ```
  
  Note: if $?x$ occurs $m$ times, $m - 1$ equalities are enough
Cross product of tables

- Understand that, conceptually, an SQL statement like

\[
\text{SELECT } \ldots \text{ FROM } T_1, T_2, \ldots, T_k \text{ WHERE } \ldots
\]

selects elements from the **cross-product**

\[
T_1 \times \cdots \times T_k \quad (\text{which has } |T_1| \cdots |T_k| \text{ elements})
\]

(where some or all of the \(T_i\) can be the same table)
Joining of tables

- The `WHERE ... = ...` effectively ask for a `JOIN`
- This `JOIN` effectively asks for a **list intersection**
- If we `CREATE` an index for the respective tables on the respective join attributes, this list intersection gets fast

E.g., by sorting (a copy of) the table by that attribute
Join ordering

- Typical SQL-from-SPARQL queries require multiple joins
- Order of joins can make a **huge** performance difference
- For our example query, the film table (actors – films) is more than ten times larger than the spouse table

- **Join order 1:** look at all married couples and for each get their films and check whether they overlap
  materializes list of films of all married people (small)

- **Join order 2:** look at all pairs of actors who played in the same film, and for each check whether they are married
  materialized all pairs of actors from same film (large)
Join ordering, continued

- Without further ado, SQLite seems to take the order of the tables in the FROM clause as its join order

```sql
SELECT spouse.person1, spouse.person2
FROM film as film1, film as film2, spouse
WHERE spouse.person1 = film1.actor
AND spouse.person2 = film2.actor
AND film1.movie = film2.movie;
```

Alternatives: (note that there are 6 possible orderings)

```sql
FROM spouse, film as film1, film as film2
```

≈ 8 sec

```sql
FROM spouse, film as film2, film as film1
```

≈ 2.3 sec
References

- **Textbook**
  - Nothing about this topic in the text book by Manning, Raghavan, and Schütze

- **Wikipedia**