Information Retrieval

WS 2017 / **2018**

Lecture 12, Tuesday January 23rd, 2018

(Knowledge Bases, SPARQL, Translation to SQL)

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

UNI FREIBURG

Overview of this lecture

Official evaluation:

Tell us in the forum if you did not receive any mail so far

- Organizational
  - Your experiences with ES11 — Naive Bayes
  - Exam registration — **Deadline 31.01.2018**
- Content
  - Knowledge bases + SPARQL — explanation + examples
  - Databases + SQL — explanation + examples
  - SQLite — a lightweight database
  - SPARQL to SQL — algorithm + example
  - Performance — joins and join order
  - **ES12:** Implement the SPARQL → SQL translation and use it to process SPARQL queries with Python+SQLite

Experiences with ES11 1/3

- Summary / excerpts
  - Nice topic / exercise / datasets
  - For about half of you it was relatively easy

    Look at the master solution, it's very little code
  - For about half, it took more time due to the usual reasons

    Finding the right numpy operations took some time, too
  - The remaining half had no time to do the exercise

Experiences with ES11 2/3

- Results Genres **Ho**rror, **Dr**ama, **Do**cumentary, **Co**medy, **We**stern
  - Variant 1: Ho 76%, Dr 79%, Do 77%, Co 65%, We 91%
  - Variant 2: Ho 87%, Dr 57%, Do 86%, Co 64%, We 98%
  - In Variant 2, all classes are equally frequent in the training set, and hence the class probabilities p_c are all equal

    Note that p_c is a factor in the formula for predicting class c
  - Some classes have more specific words than others:

    Western: gang, town, sheriff, man, ranch, men, father, ...

    Comedy: man, life, wife, young, family, money, time, ...
  - Excluding stopwords as features does not improve the result quality: in prediction, they contribute equally to all classes

Experiences with ES11 3/3

- Results Ratings
  - Variant 1: R 76%, PG-13 30%, PG 36%
  - Variant 2: R 52%, PG-13 41%, PG 49%
  - Again, having all classes equally frequent in the training data helps the classes which are relatively rare in the original data
  - Top words are not really specific for any of the classes

    R: life, man, young, find, family, finds, father, wife, ...

    PG-13: life, family, man, young, world, find, father, old, ...

    PG: life, young, father, man, old, world, family, find, ...

    The more specific words come further down in the list (but again, the unspecific words do not really hurt)

Knowledge Bases and SPARQL 1/7

- What is a knowledge base
  - A knowledge base is a database of statements about entities and their relations
  - Critical: **unique** identifiers for each entity and predicate
  - A common format / schema is to express all statements as subject predicate object triples:

Nicole Kidman	acted in	Eyes Wide Shut
Brad Pitt	acted in	Burn After Reading
Tom Cruise	acted in	Eyes Wide Shut
Sidney Pollack	acted in	Eyes Wide Shut
Joel Cohen	directed	Burn After Reading
Ethan Cohen	directed	Burn After Reading
Nicole Kidman	married to	Tom Cruise

Knowledge Bases and SPARQL 2/7

- Freebase and WikiData
  - Freebase is the largest open general-purpose KB to date

    Started by Metaweb in 2007, acquired by Google in 2010

    Freebase has become read-only in March 2015 and WikiData has taken over to become **the** standard general-purpose KB

    Final size: **3 billion** triples on **60 million** entities
  - Wikidata is like Wikipedia for knowledge bases: anybody can contribute (with some amount of editorial control)

    Current size: **377 million** triples on **43 million** entities

    In Wikidata, entities are called "items" and triples are called "statements"

Knowledge Bases and SPARQL 3/7

- 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

It's a so-called **mediator** entity, the purpose of which is to serve as a link between the entities it connects

The full Wikidata dataset has a similar mechanism

For simplicity, the dataset for ES12 has no mediators

Knowledge Bases and SPARQL 4/7

- 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 (show example)
  - **Variant 2:** web pages containing only structured data

    Typical format: RDF (a particular kind of XML)
  - 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

Knowledge Bases and SPARQL 5/7

- What is SPARQL
  - The standard query language for knowledge bases

    SPARQL = **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
  - Example query in natural language: actors who are married and played together in at least one movie
  - The same query expressed in (simplified) SPARQL

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

Knowledge Bases and SPARQL 6/7

- SPARQL syntax
  - In the lecture today, we use a simplified syntax

    In "real" SPARQL, names of subjects / predicates / objects may contain whitespace and are surrounded by <...>
  - The actual SPARQL syntax is slightly more complicated and has many more features

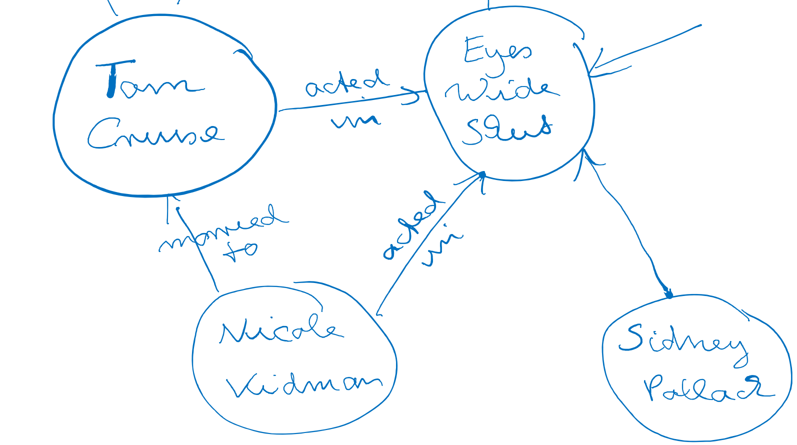
    In particular, it involves namespace prefixes, so that names can be made globally unambiguous

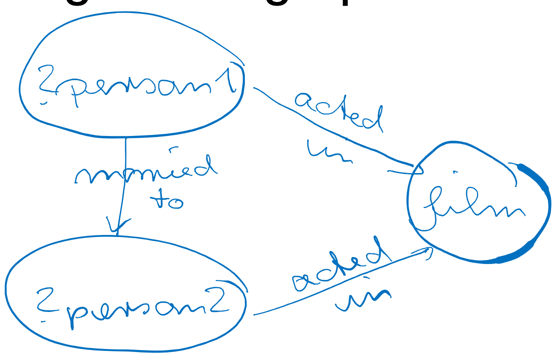
    See the Wikipedia page or the W3C specification if you are interested

Knowledge Bases and SPARQL 7/7

- SPARQL queries as subgraphs
  - One 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

KB





QUERY

Databases and SQL 1/4

- Introduction
  - Data from a knowledge base can also be stored in an ordinary database

    This is what we do in the lecture today and for ES12
  - The standard query language for databases is SQL

    SQL = **S**tructured **Q**uery **L**anguage
  - On the following slides, let us recap the basics from databases and SQL via a few examples

Databases and SQL 2/4

- What is a database
  - For this lecture, a database is a collection of tables, where each table has a fixed number of columns
  - For example, we could have one table for each predicate from our knowledge base, with two columns each

Table for "acted in" predicate

actor	**film**
Brad Pitt	Burn after Reading
Tom Cruise	Eyes Wide Shut
...	...

Table for "married to" predicate

person1	**person2**
Nicole Kidman	Tom Cruise
Ellen DeGeneres	Portia de Rossi
...	...

For ES12, you should work with **one** big table for the whole database, with three columns (subject, predicate, object)

Databases and SQL 3/4

- SQL example 1
  - Example query FROM **one** table "acted_in" with two columns "actor" and "film"

```
SELECT   actor
FROM     acted_in
WHERE    film = "Burn After Reading";
```

    In words: all actors from movie "Burn After Reading"

  - Principle: select those rows from the specified table which satisfy properties specified in WHERE clause

Databases and SQL 4/4

- SQL example 2
  - Example query FROM **multiple** tables, each with three columns "subject", "predicate", "object"

```
SELECT  ex1.subject, ex2.subject, ex1.object
FROM    example AS ex1, example AS ex2
WHERE   ex1.predicate = "acted in"
AND     ex2.predicate = "acted in"
AND     ex1.object = ex2.object
```

    In words: all pairs of actors who acted in the same movie

  - Principle: selects items from cross-product $T_1 \times \cdots \times T_k$ which satisfy properties specified in WHERE clause
  - Syntax: us AS for unique names of copies of same table; use table.column to refer to that column from that table

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

SQLite 2/4

- 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

SQLite 3/4

- Some useful SQL commands by example
  - Create a table with a given schema

    ```
    CREATE TABLE acted_in(actor TEXT, film TEXT);
    ```

  - Create an index for a column of a table

    ```
    CREATE INDEX acted_in_index ON acted_in(actor);
    ```

  - Extract / combine data from tables

    ```
    SELECT * FROM acted_in WHERE ... LIMIT 100;
    ```

  - Delete table / index (without error msg if it's not there)

    ```
    DROP TABLE IF EXISTS acted_in;
    DROP INDEX IF EXISTS acted_in_index;
    ```

SQLite 4/4

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

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

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

SPARQL to SQL Translation 1/4

- 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 acted_in(actor TEXT, film TEXT)
    CREATE TABLE married_to(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 ES12, use **one big table** for all the data, with three columns named **subject**, **predicate**, **object**

    This is deliberately different from how we did it in the lecture, so that you have to do some thinking yourself

SPARQL to SQL Translation 2/4

acted_in

actor	film
...	...

married_to

person1	person2
...	...

- Example
  - SPARQL query

```
SELECT ?p1 ?p2 ?f WHERE {
   ?p1  acted_in  ?f .
   ?p2  acted_in  ?f .
   ?p1  married_to  ?p2          }
```

  - SQL query:

```
SELECT  mar.person1, mar.person2, act1.film
FROM    acted_in AS act1, acted_in AS act2, married_to AS mar
WHERE   act1.film = act2.film
AND     act1.actor = mar.person1
AND     act2.actor = mar.person2
```

SPARQL to SQL Translation 3/4

- Algorithm
  - It is up to you in ES12, 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 how SPARQL and SQL work

    It's a perfect exercise to understand the basics !
  - On the next slide we give you some valuable advice

SPARQL to SQL Translation 4/4

- 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

    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.

    ?x: f1.subject, f2.object, f5.object

  - Then, when building the SQL query, add the corresponding equalities to the WHERE clause, e.g.

    WHERE f1.subject = f2.object AND f2.object = f5.object

    Note: if ?x occurs m times, m – 1 equalities are enough

Performance 1/4

- Cross product of tables
  - Recall that, conceptually, an SQL statement like

    SELECT ... FROM $T_1, T_2, \ldots, T_k$ WHERE ...

    selects elements from the **cross-product**

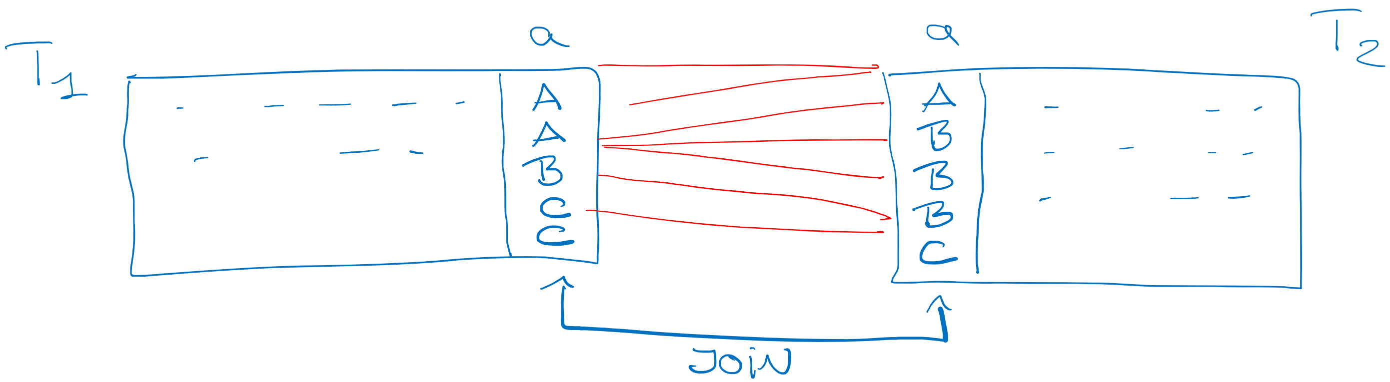
    $T_1 \times \cdots \times T_k$ (which has $|T_1| \cdot \cdots \cdot |T_k|$ elements)

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

Performance 2/4

- Joining of tables
  - Each ... = ... in the WHERE clause effectively ask for a JOIN operation between two tables
  - Algorithmically, a JOIN requires 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



Performance 3/4

- Join ordering
  - Typical SQL-from-SPARQL queries require multiple joins
  - Order of joins can make a **huge** performance difference
  - For our example query, the acted_in table (actors – films) is about ten times larger than the married_to table
  - **Join order 1:** 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 order 2:** 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)

Performance 4/4

with all indexes

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

16.4 seconds

```
SELECT  married_to.person1, married_to.person2
FROM    acted_in as acted1, acted_in as acted2, married_to
WHERE   married_to.person1 = film1.actor
AND     married_to.person2 = film2.actor
AND     acted1.film = acted2.film;
```

Alternatives: (note that there are 6 possible orderings)

2.0 seconds

```
FROM    married_to, acted_in as acted1, acted_in as acted2
```

```
FROM    married_to, acted_in as acted2, acted_in as acted1
```

2.5 seconds

References

- Textbook
  - Nothing in the text book by Manning, Raghavan, Schütze
- Wikipedia
  - http://en.wikipedia.org/wiki/Knowledge_base
  - http://en.wikipedia.org/wiki/SPARQL
  - http://en.wikipedia.org/wiki/SQL
  - http://en.wikipedia.org/wiki/SQLite
  - http://en.wikipedia.org/wiki/Freebase
  - https://www.wikidata.org
  - https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format