CS276B

Text Information Retrieval, Mining, and Exploitation

Practical 2

Jan 30, 2003

Topics

- Part 1B Organization
- Data structures review
- API review
- Parameters and properties
- JDBC conventions
- Neat things from Cora
- Name discussion
- Interface discussion

Part 1B Organization

- We have reviewed and made small changes to your code (checked into CVS)
- In part 1B stay with existing tasks and groups
- Meet with Chris and Teg to get detailed feedback about design and implementation
- To do in part 1B (by Feb 11):
  - Fix basic problems
  - Make improvements in design and organization
  - Add some new algorithms and functionality

Part 1B Organization (2)

- A:
  - Add PageInstance table initialization/restart
  - Add page scoring (poss. learning) for focused web crawling
  - Improve concurrency control and robustness
- B:
  - Add academic paper classification (Naïve Bayes?)
  - Add sanity checking on hub context extraction
  - Add platform independence
- C:
  - Add HMM for title/author/citation extraction from papers
  - Improve segmenting of individual citations
- D:
  - Improve performance of rule-based extraction with some sanity-checking
  - Add HMM for citation extractions

Part 1B Organization (3)

- E:
  - Improve scalability of n^2 clustering algorithm
  - Specialized publication models
  - Better term weighting (medium frequency is impt.)
- F:
  - Improve robustness by testing on large real datasets
  - Continue to augment interface
  - Add document similarity algorithm and interface
- Interact between groups that share components:
  - A&B: Classification, inc. Naïve Bayes
  - C&D: HMM Extraction, inc. author and title models
  - E&F: Similarity metrics, production data tables

Data Structures: Raw Schema

- `PageInstance(id, url, filename, status, score, isHub)`
- `PaperInstance(id, url, rawFilename, textFilename, status, author, title, abstract, citations, selfCitation, citationInstanceID, authorBegin, authorEnd, titleBegin, titleEnd, abstractBegin, abstractEnd, citationsBegin, citationsEnd, paperID)`
- `CitationInstance(id, fromPaperInstanceID, fromHubInstanceID, toPaperInstanceID, citationText, citationTag, author, title, date, publication, volume, pages, editor, publisher, citationID, publicationID, paperID, status)`
- `CitationContextInstance(citationInstanceID, paperInstanceID, contextBegin, contextEnd, context)`
- `AuthorInstance(id, authorText, first, middle, last, suffix, citationInstanceID, paperInstanceID, authorID)`
- Is someone using each of these?

Data Structures: Prod. Schema

- `Paper(id, citationInstanceID, paperInstanceID)`
- `Author(id, first, middle, last, suffix, email, affiliation, canonicalName)`
- `Authorship(paperID, authorID)`
- `Name(id, altName, isCanonical)`
- `Publication(id, canonicalName)`
- `PublicationName(publicationID, altName)`
- `Citation(fromPaperID, toPaperID, citationInstanceID)`
- Is someone using each of these?
- The description of the schema is at:
  http://www.stanford.edu/class/cs276b/project.html

Data Structures: File System

- The directory /afs/ir/class/cs276b/data/ will contain:
  - Special directories, in which a file for a record with an id of ABCDEFGH (in decimal form) is saved in a file with path AB/CD/EF/ABCDEFGH
    - webPages
    - rawPapers
    - textPapers
  - luceneIndex
  - Other specialized data files
  - The Properties file
  - Other specialized config files (e.g., luceneconfig)
- Do people agree that the filename can always be the id of the record it is associated with (an auto incremented key)

Code Organization

- All of your group's files in one package
- Main class implements the Runnable interface
- Package operation should be parameterized (parameters stored in Properties class)
- A Main.main(String[]) method which allows the program to be run at the command line, with no arguments and also with optional arguments
- When Main.main or Main.run is invoked, your program should process all available data from the database and then exit
- How should we report the status of a process back to the scheduler?
- I'll be writing a scheduler to start processes periodically (unless someone else wants to)

Code Organization (2)

- Each individual class should also have a main(String[]) function which tests the functionality of the class
  - Where appropriate, should be able to test w/o database, by giving data or filenames on the command line
- All main methods should also parse and check arguments, and print out usage with -h or when incorrect
- Output: don't print to stdout, print limited status information to stderr during normal operation,
- Can use a "verbose" property to indicate whether or not to print error information
- When you submit, make sure that it is tested and wired to work with the real tables in the real database

Code Organization (3)

- Package.html
  - Plain HTML (MS Word HTML doesn't work)
  - Special tags:
    - @author (classes and interfaces only, required)
    - @version (classes and interfaces only, required)
    - @param (methods and constructors only)
    - @return (methods only)
    - @exception (@throws is a synonym added in Javadoc 1.2)
  - Please give usage information – how we call your program from the command line, with some use cases – samples of what we can do and expect to see
  - Also please tell us what behavior to expect of your program – how many records it updates, whether it terminates, how it handles errors, when it checks the Properties file, etc.

Parameters and Properties

- You can specify properties (parameters) for your program in a file called properties.dat located in the project directory, in the format
  package.propertyname propertyvalue
  one on each line
- We will provide a static method getProperties which creates a new java.util.Properties object from the file
- There is also a setProperties for write-back
- If you have a short-running process you may only want to load the Properties when it starts
- If you have a long-running process you may want to load the Properties file periodically (every 10 seconds?)

Parameters and Properties (2)

- Ideas for what to keep in the Properties class:
  - Info used to access shared resources:
    - Database table names (so you can switch easily for testing!)
    - Location of base directories for data
    - Name of specific datafile on disk
    - Google key
  - Parameters:
    - Number of simultaneous connections
    - Include/exclude list of servers
    - Maximum bandwith
    - Number of records to process at a time
    - Length of time to run continuous process
    - Which of two algorithm implementations to use
    - Verbose flag for amount to print out
    - Location of error file to print to

JDBC Conventions

- A Connection object represents a database transaction (or series of transactions)
- Current MySQL setup doesn't have transactions, but they could be added with a performance hit. Do we need them?
- The default Connection is to auto-commit. conn.setAutoCommit(false) changes this.
- More efficient not to use auto-commit, but you must remember to commit periodically with conn.commit()
- Each Thread should have its own Connection object, to avoid "commitment confusion"
- Connections should be explicitly closed when you are done with them: conn.close()
- Also close ResultSets and Statements in finally {} block

JDBC Conventions (2)

- Where you will be doing the same type of query or update over and over (but with different data you should use a preparedStatement for efficiency
- Allows MySQL to compute the query/update strategy once, but run it multiple times

```
PreparedStatement ps1 = conn.prepareStatement("SELECT filename FROM
    PageInstance WHERE status=?");
ps1.setInt(1, 4);
ps1.executeQuery();
PreparedStatement ps2 = conn.prepareStatement("INSERT INTO
    PageInstance (id, url) VALUES (?, ?)");
ps2.setInt(1, 9293874);
ps2.setString(2, "http://www.stanford.edu");
ps2.executeUpdate();
```

- Can do UPDATE in similar fashion

JDBC Conventions (3)

- You should always name fields in an INSERT statement, in case we change the ordering of the fields later.
  - INSERT INTO PageInstance (status, score) VALUES (4, 0.9)
- If you are not using a PreparedStatement, you need to escape special characters in your String values:

\0	An ASCII 0 (NUL) character.
\'	A single quote (`'`) character.
\"	A double quote (`"`) character.
\b	A backspace character.
\n	A newline character.
\r	A carriage return character.
\t	A tab character.
\\	A backslash (`\`) character.

JDBC Conventions (4)

- We should all be using the same username and password to access the database. You should retrieve this from the Properties object.
- We should also put database table names in Properties

Discussion of Interface

- Demo
- Design comments
- Additional features?

Neat things from Cora

- McCallum, A.; Nigam, K.; Rennie, J.; and Seymore, K. 1999. A machine learning approach to building domainspecific search engines. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence.
  - http://citeseer.nj.nec.com/article/mccallum99machine.html
- Topics:
  - Reinforcement learning for focussed crawling
  - Text categorization for topic hierarchy
  - Information extraction of headers and citations with HMMs