Documentation for the DSpace Recommender System Prototype

Desmond Elliott
Web Services and Systems Laboratory
HP Laboratories, Bristol
7th August 2008

Introduction

The purpose of this document is to provide future researchers and developers with a resource to help them use the recommender system prototype, to present our efficiency evaluation of the prototype, and to highlight possibilities for future work.

This prototype is a metadata-based recommender system which produces item-to-item and personalised recommendations. Item-to-item recommendations are produced by analysing specified item metadata stored in a repository; personalized recommendations are produced by analysing metadata extracted from items which users have expressed an explicit interest in.

We evaluated the efficiency of the system using 1,927 HP Labs Technical Reports from 1999 – 2007. The evaluation process informed a re-engineering of the system to improve performance. We have not evaluated recommendation effectiveness of the prototype, but we hope others will using the source code we have released with their own datasets and techniques.

This report should be read in conjunction with the technical report to gain a clearer understanding of the architecture of the prototype.

Installation Instructions

To install the recommender system prototype

1. Install the required supporting packages;
2. Download the prototype source from the DSpace Sandbox Subversion repository;
3. Build DSpace;
4. Run the initialisation program.

These instructions are based on installing the recommender system prototype on Ubuntu 8.04. Installation on other systems should be possible, but has not been tested. The following instructions assume a fresh system installation with superuser access.

These instructions currently mirror the DSpace installation instructions, but they are provided in full to prevent them being lost as development on the next version of DSpace continues.

1. At a terminal, run the following command to install the required supporting applications:
   a. `sudo apt-get install postgresql maven2 subversion sun-java5-jdk ant-optional`
2. Update java-alternatives to use the Sun Java 5 JDK
   a. `sudo update-java-alternatives --set /usr/lib/jvm/java-1.5.0-sun`
3. Export JAVA_HOME for Tomcat
   a. `export JAVA_HOME=/usr/lib/jvm/java-1.5.0-sun`
4. Download and extract the Tomcat 5.5.x Core **tar.gz** binary from http://tomcat.apache.org/download-55.cgi.

   a. The extracted binary is moved and the location will be referred to as `$tomcat`.
      a. `sudo mv apache-tomcat.5.5.* /opt`
   b. Change the ownership of `$tomcat`.
      a. `sudo chown <your username> -R $tomcat`
5. Enable database access on your local machine
   a. `sudo` *`<editor>`* `/etc/postgresql/<version>/main/postgresql.conf`
      a. Uncomment the `listen_address` line to allow connections from the local machine
   b. Add database access to the database you will create for DSpace
      a. `sudo` *`<editor>`* `/etc/postgresql/<version>/main/pg_hba.conf`
      b. Add `local <db> <your username> 127.0.0.1/32 md5` to the bottom of the file, where `<db>` is the name database you will create and use for DSpace
   c. Restart Postgres:
      a. `sudo /etc/init.d/postgresql-<version> restart`
4. Create a new database user as the `postgres` user and return to your login:
   a. `sudo su postgres -`
   b. `createuser -dRAP <username>`
   c. `exit`
5. Create a new database
   a. `createdb -E UNICODE <db>`
6. Create the directory you will install DSpace. This location will be referred to as `$recsys`
   a. `sudo mkdir /opt/dspace`
   b. `sudo chown <username> $recsys`
7. Pull the recsys-prototype source code from the DSpace Sandbox using subversion to your home directory. This location will be referred to as `$src`
   a. `svn checkout http://dspace-sandbox.googlecode.com/svn/prototypes/recsys-prototype recsys-prototype`
8. Edit `dspace.cfg` to reflect local settings
   a. *`<editor>`* `$src/dspace/config/dspace.cfg`
   b. `dspace.dir = $recsys`, where DSpace will be installed, specified in Step 6
   c. `db.url = jdbc:postgresql://locahost:5432/<db>`, the database you created in Step 5
   d. `db.username = <username>`, the user you added to pg_hba.conf in Step 4
   e. `db.password = <password>`, the password you set in Step 4
   f. Configure SMTP access as per DSpace Installation Instructions
9. Change to the `$src/dspace` directory and build DSpace
   a. `mvn package`
10. Change to the `$src/dspace/target/dspace-SNAPSHOT-build.dir/` directory and install DSpace
    a. `ant fresh_install`
11. Deploy the JSP User Interface webapp:
    a. `cp -R [recsys]/webapps/jspui [tomcat]/webapps`
12. Start Tomcat:
    a. `$tomcat/bin/catalina.sh start`
13. Create an administrator account:
    a. `$recsys/bin/create-administrator`
14. Initialise the recommender system:
    a. `$recsys/bin/dsrun org.dspace.app.recsys.util.InitialiseQuambo`
15. Open your web browser and point to the URL specified in `dspace.cfg`
    a. `http://<dspace.url>`

Configuration Options

The recommender system prototype adds 5 new configuration options to *dspace.cfg*.

Option	**Values**	**Description**
quambo.similarity.terms	anything from the metadatafieldregistry table, e.g. *dc.subject*, *dc.contributor.author*	The metadata from items stored in the database to use when calculating the similarity between items, also used to determine which metadata to extract from an item when bookmarking.
quambo.similarity.algorithm	*cosine* or *jaccard*	The similarity algorithm to use when determining the similarity between items or between and item and the metadata extracted from bookmarked items. We have not evaluated which algorithm provides more effective recommendations.
quambo.processing-style	*batch* or *real-time*	The recommender system can be run in real-time, however, running the recommender system as a batch process is strongly recommended.
quambo.recalculation-delta	Any whole number > 0	The recalculation delta is the number of hours which need to have passed between the previous calculation of recommendations for an item and the current attempt to determine recommendations for an item. We used a value of 6 in our experiments.
quambo.similarity-threshold	Any number between 0.0 and 1.0, inclusive	The similarity threshold is the minimum similarity required between items or between an item and the metadata extracted from bookmarked items for a recommendation to be produced. We used a value of 0.5 in our experiments.

Evaluation

We evaluated the prototype using 1,927 HP Labs Technical Reports from 1999 – 2007.

The test equipment was a 2.8 GHz single-core VMWare Ubuntu 8.04 guest operating system with 1GB RAM and 15GB hard disk space.

The recommender system specific configuration options in `dspace.cfg` were

- `quambo.similarity.terms = dc.contributor, dc.author`

- `quambo.similarity.algorithm = cosine`
- `quambo.processing-style = batch`
- `quambo.recalculation-delta = 6`
- `quambo.similarity-threshold = 0.5`

Our test dataset was the 1,979 technical reports published by HP Labs from 1999 – 2007. We split the dataset into an aggregation of consecutive years.

Dataset ID	No. of Items
1	142
2	297
3	589
4	921
5	1177
6	1396
7	1611
8	1789
9	1979
10	4214

Dataset 10 includes an entire duplicate of dataset 9. It was included in our experiments because we wanted to test the performance on a system representative of the number of items stored in real repositories. The mean number of items stored in a DSpace repository based on the data collected from OpenDOAR on 7th July 2008 is 4,180.

Our methodology for each experiment was:

1. Run the BatchProcessRecommendations tool;
   ```
   a. ./[recsys]/bin/dsrun
      org.dspace.app.recsys.util.BatchProcessRecommendations -i -a
   ```
2. Copy the logs and database dump to a remote machine;
3. Clear the database and assetstore;
4. Import the required items into DSpace using ItemImport tool for the next test;
5. Restart machine.

Results

Our experiments highlighted engineering issues in our original design. Test 1 provides results from our original design. Test 2 provides results for after implementing a hashtable which brings all required item metadata into memory, rather than reading from the database when required. Test 3 provides results after implementing the hashtable and a write-at-once approach to writing recommendations to the database.

No. of Items	Test 1	Test 2	Test 3
142	257	6.35	5.41
297	1073	26.47	15.36
589		142.8	46.07
921		724.23	103.9
1177		2058.3	162.29
1396		4548.67	227
1611		5023.25	296.89
1789		7511.81	358.88
1979		9662.48	437.44
4214			1886.81

Table 1: Time in seconds to perform complete batch process

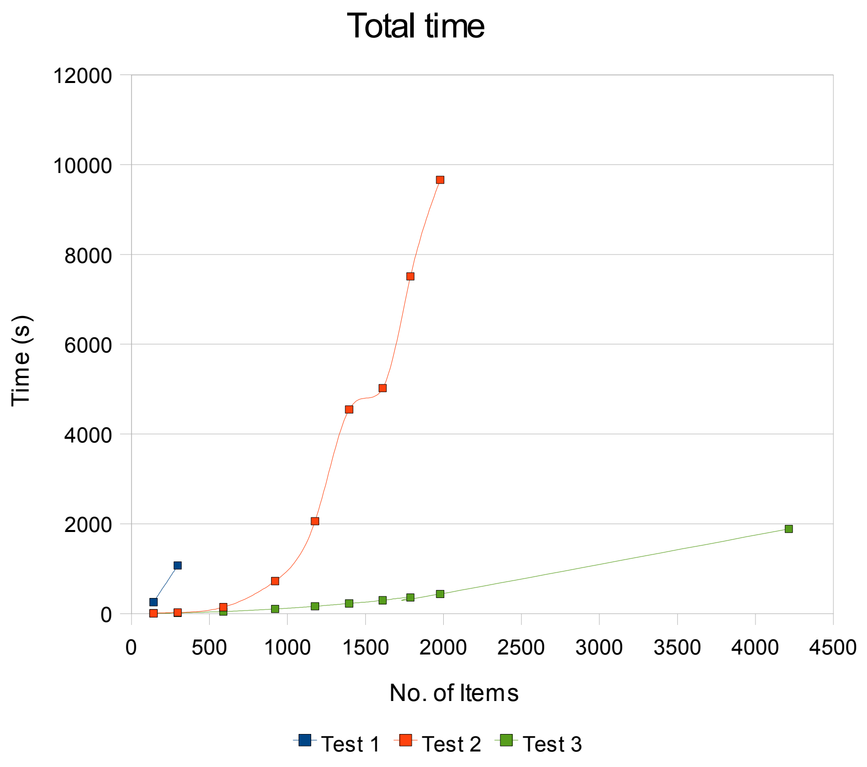

Figure 1: Time in seconds to perform complete batch process

Table 1 shows the total runtime of calculating item-to-item recommendations for every item in the repository. The total runtime for dataset 10 in Test 3 is misleading because it includes duplicate items from dataset 9, however, it suggests the system could scale to larger repositories.

Figure 1 shows the efficiency improvements of implementing an in-memory cache of item metadata, Test 2, and writing records to the database at the end of the operation, Test 3.

No. of Items	Test 1	Test 2	Test 3
142	1.81	0.05	0.02
297	3.61	0.09	0.03
589		0.24	0.06
921		0.79	0.09
1177		1.75	0.12
1396		3.25	0.14
1611		3.12	0.17
1789		4.2	0.18
1979		4.88	0.2
4214			0.43

Table 2: Time in seconds to calculate recommendations per item

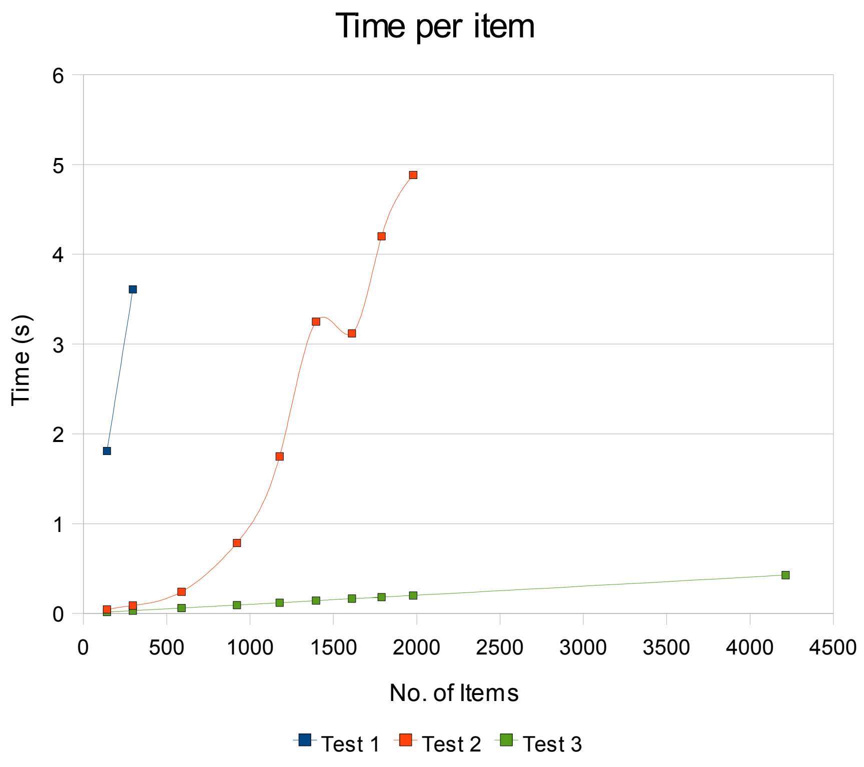

Figure 2: Time in seconds to calculate recommendations per item

Table 2 shows the time to calculate the recommendations for an individual item. This is done by comparing the item to all other items in the repository. The time for dataset 10 is not misleading in this instance because no unnecessary database access occurred when performing the calculations.

Like Figure 1, Figure 2 also shows the efficiency improvements of implementing an in-memory metadata cache, Test 2, and writing records to the database at the end of the operation, Test 3.

No. of Items	Test 2	Test 3
142	2.9	2.85
297	5.1	4.87
589	8.5	8.47
921	13.9	12.8
1177	16.5	16.54
1396	19	18.96
1611	21.7	22.54
1789	24.6	25.46
1979	26.5	25.92
4214	52.9	

Table 3: Time in seconds to create in-memory metadata cache

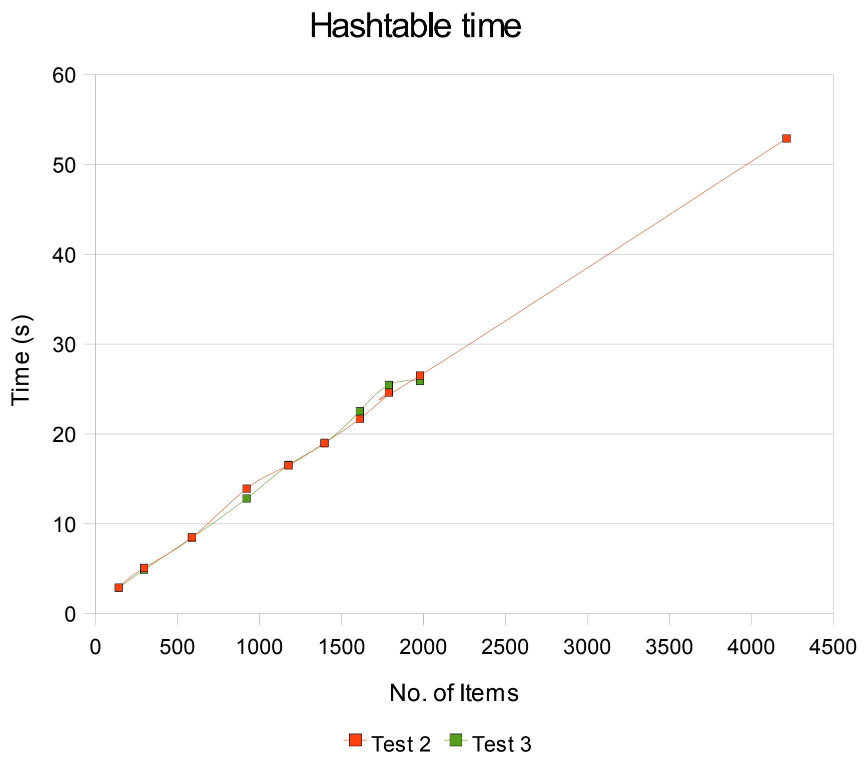

Figure 3: Time in seconds to create in-memory metadata cache

Table 3 shows linear performance when creating the in-memory metadata cache.

Future Work

There is a significant amount of engineering work to complete before this prototype can be integrated into an official release of DSpace.

- Investigate the benefits of threading the calculation process;
- Implement a Manakin user interface;
- Implement an Admin user interface;
- Store and retrieve similarities as floating point numbers in the database;
- Confirm the JSP user interface renders correctly on major web browsers;

- Investigate effectiveness of recommendations;
- Write a unit testing suite for the recommender system;
- Refactor the prototype as a DSpace add-on;
- Investigate a priority system for batch processing recommendations;
- Investigate which algorithms are most effective;
- Ensure appropriate Authorization checks are implemented;
- Re-consider the database design to improve efficiency;
- Use Lucene to extract most frequently occurring terms from bitstreams;
- Integrate the Atom code into the existing DSpace feed handling code;
- Investigate in-cache metadata memory use for multiple datasets.

Acknowledgements

Thanks to John Isbell and Tom Wilcox for feedback on the installation instructions. Thanks to Michael Rhodes for feedback on a draft of this document.