IGLU
AI/IR for Java

Manual by Travis Bauer
2002

Email: trbauer@indiana.edu.

Contents

List of Figures

Part I

Introduction and Setup

Chapter 1

Introduction

Welcome to the IGLU source code library. IGLU is an open source project started an Indiana University's Computer Science Department. It is an attempt to build a library of general purpose Java classes for researchers in Information Retrieval. There are some algorithms and utility classes which might be useful to many people doing research in these areas which are time consuming to write when trying to build computer simulations and models. The IGLU library is a set of packages to provide classes to reduce this overhead.

This library does not provide many complete programs, other than a few example classes. It contains the building blocks you need to build your own applications. The goal is to provide useful code and examples while maintaining maximum flexibility. So, for example, our search engine API does not create its own indices. We don't assume what algorithms will be used to index documents.

The library was first assembled by and code first contributed by Travis Bauer. Ryan Scherle followed up with a set of utility classes and much of the iglu.ir library.

This library is distributed under the IGLU license, included with the source distribution. Although we use these classes in our own research, we cannot verify their correctness or usefulness for any particular purpose. Use them at your own risk.

This manual is not intended to be a replacement for the javadoc. Much of the specific functionality is described in the javadoc files which can be view from IGLU's main homepage or generated from the source. This manual provides a broad overview of IGLU's functionality and to give the reader a model of its capabilities.

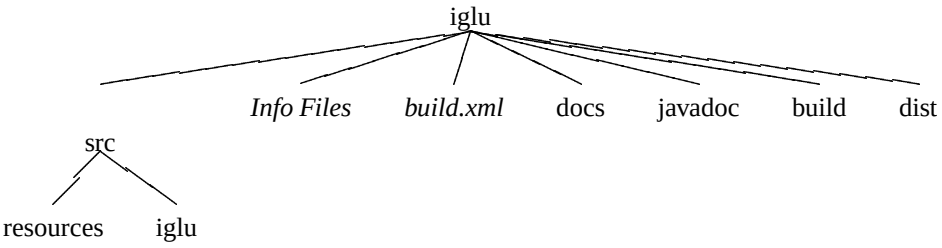
Please tell us if you find them useful (or if you find a bug). The team contributors can be reached at iglu@cs.indiana.edu.

Travis Bauer: trbauer@indiana.edu

Chapter 2

Organization

The source code distribution of IGLU is as follows:



iglu/src/iglu Under this directory are placed the actual source files

iglu/src/resource Non-java resource files (gifs, stop lists, etc).

Info files License files, README, etc.

build.xml Ant's build file

docs The source code for this document.

javadoc Generated by `ant javadoc`, containing the javadoc for iglu.

build Contains the compiled jar files. Generated by `ant build`

dist Contains the distribution jar files. Generated by `ant dist`

The packages related to IGLU follow this same methodology for source code packaging.

Chapter 3

Installation

3.0.1 Requirements

1. Java 1.3 or later (http://java.sun.com)
2. Ant 1.4 or later (http://jakarta.apache.org/ant)

3.0.2 Installation

IGLU is available from http://www.cs.indiana.edu/ trbauer

IGLU is distributed as two packages: one containing the source code and one containing the compiled class files. If you only want to use IGLU and are not interested in modifying it, all you need is the jar with the class files.

To install the class-only file, simply include it in your classpath.

To install the source code, download and unjar the source code package. Then include the iglu/build directory in your classpath. Compile the source code by executing `ant build` from within the iglu directory.

Chapter 4

Conventions and Contributing

We welcome any additional relevant class, bug fixes, and other enhancements.

All source files in the IGLU library should follow these conventions:

1. The first think in any file is a header which identifies the class name, the initial author's name and email address, and carries the IGLU copyright notice following the format in the other source files.
2. The java documentation for the class should contain the initial author listed first with a "@author" tag, and any contributing authors (people who modified the class) following with "@author" tags.

Debugging statements can be permanently put in the code using the "Debug" call `Debug.debugOut(CLASSNAME, ''debug statement'')`. This allows programmers to turn on and off the debugging statements for a given class with the `Debug.setLevel` method. See the section on the `Debug` class later in this document. Keep in mind that these statement require a function call even when their statement does not get printed out. This uses up processor time, so don't leave these statement in long loops, etc.

Although none of the classes currently contain it, authors should include a static method with the following signature: `public static boolean test()`. This method should run some tests on the class, and return true if the tests pass. Eventually, we will add a utility which runs these tests on all the classes.

Part II

Core IGLU Library

Chapter 5

iglu.examples
Short Sample Programs

The iglu.examples package offers some short example programs to illustrate how iglu operates. The examples here are small. Some of the classes in IGLU also have main methods which illustrate how they are used. See the javadoc pages for more information.

5.0.3 examples.ObjCat

A simple utility which uses iglu.util.IOTools.objectFromFile to load a previously serialized object from the disk. It then prints the object to the console using its toString() method. If the object loaded is an array, it loops through the array, printing each object.

This is basically a “cat” utility for serialized objects which have been saved to a file.

5.0.4 examples.ProxyServerExample

A simple WWW proxy server. Illustrates how to use the iglu.net package.

Chapter 6

iglu.ir
Information Retrieval

6.0.5 Introduction

This package has three main purposes:

1. To specify an API for Vector Space Information Retrieval research
2. To implement a set of classes to represent and implement documents, document sources, and related algorithms.
3. To implement a testing harness for evaluating performance of Information Retrieval algorithms using standard IR evaluation techniques

We assume that the reader is familiar with the vector space model for information retrieval. If not, there are numerous resources available.

6.0.6 API/Implementation for Information Retrieval

The `Document` interface serves as the basic interface for all document types in the library. A document is an object which has content, indexable content, and can answer basic questions about itself. It does not contain a location, a title, or any other "metadata." A document is a document's contents. The `AbstractDocument` abstract class implements some basic functionality making it easier to create actual document

types. Subclasses should implement actual document types, such as "HTMLDocument" which stores information about HTML documents.

The `VectorCreator` interface is the basic interface for all objects which can create vectors from documents. Individual VectorCreators might have to be trained before they can be used. The TFIDFVectorCreator is able to generate *Term Frequency/Inverse Document Frequency* vectors from documents, but has to be loaded with a corpus first.

The `SearchEngine` interface is a java interface for search engines. By Search Engine, we mean an object which can store documents and related vectors, and conduct search based on queries. Search Engines do not create vectors. The basic idea is that VectorCreators are used to generate vectors for documents, and these vectors are given, along with the documents to the search engine for storage. We have two search engines implemented. One search engine keeps all the information in RAM. This is useful for small tests or simple search engines that are not permenantly stored. The FileSearchEngine is more powerful, indexing documents using an inverted index stored in a B-Tree.

6.0.7 IR Evaluation

A basic type of evaluation for IR techniques it Precision/Recall. You start with a document set, a set of queries, and a mapping from queries to documents telling you which documents should be retrieved for each query. Then, using some indexing technique, you generated indices for the documents and put them into a search engine. The queries are sent to the search engine, and the results returned are analyzed to see how well the indexing technique and search engine were able to retrieve the correct documents.

The kind of Precision/Recall results this package provides is consistent with the methods specified in the book Modern Information Retrieval by Ricardo Baeza-Yates and Berthier Ribeiro-Neto.

IGLU provides a set of classes for performing this evaluation and analyzing the results. This evaluation would be useful for the following:

1. Comparing two indexing techniques.
2. Comparing the effect of varying free parameters on a single indexing technique.
3. The performance of an indexing technique on different corpora.

For example, let's say you have two different methods for generating indices for documents and want to know which method works better. To test the two methods, you would do the following steps (Each step is described in more detail below):

Develop Vector Creators Write two vector creators, one that implements each of the two methods you want to test.

Problem definition Get a corpus as a document set, a set of sample queries, and a list of which documents in the document set should be retrieved for each query. These are put together as an `IRPacket` object.

Precision/Recall Tests Create two PrecisionRecall objects, one for each of the vector creators. Both should be constructed with the same `IRPacket`. Get the results of running the test from each PrecisionRecall object.

Examine the results Use the IGLU classes, or your own classes to analyze the results of the tests.

Develop Vector Creators

The first step in using an indexing technique with IGLU is to implement this technique as a VectorCreator. A VectorCreator takes a document and produces a TermVector for the document using the technique it implements. Some vector creators need to be trained before being used. This functionality needs to be provided by the specific implementation. Other vector creators (such as LSI) take an entire document set and produce vectors for only those documents. An api for this kind of vector generation will be included in a future version of IGLU.

Problem definition

To test an indexing technique, you need to gather a corpora, a set of queries, and a mapping from the queries to documents, telling you which documents should be retrieved for each query. IGLU provides an `IRPacket` class which combines this information into a single object. For more information on how these classes work, see IGLU's javadoc.

Precision/Recall

The PrecisionRecall class takes a vector creator and an IRPacket, and performs an IR test. The test is as follows:

1. Index all the documents into a search engine using the given vector creator
2. Submit each query to the search engine and analyze the results, producing a PRResult object.

3. Return an array of PRResult objects, one for each query.

Examine the results

A single PRResult's toString() method produces a summary of the query which looks as follows:

```
Queryid:        1
    Relevant:   299

Interpolated Recall - Precision Averages:
    at 0.0      1.0
    at 0.1      1.0
    at 0.2      0.9855
    at 0.3      0.9375
    at 0.4      0.8523
    at 0.5      0.8523
    at 0.6      0.8295
    at 0.7      0.0
    at 0.8      0.0
    at 0.9      0.0
    at 1.0      0.0

Average precision (non-interpolated)
                0.9403924816533139

Precision (interpolated)
    at 5.0 docs:        1.0
    at 10.0 docs:       1.0
    at 15.0 docs:       1.0
    at 20.0 docs:       1.0
    at 30.0 docs:       1.0
    at 100.0 docs:      0.8523
    at 200.0 docs:      0.8309
    at 500.0 docs:      0.0
    at 1000.0 docs:     0.0

Actual Results
5.0 (1.0,0.017)
10.0 (1.0,0.033)
15.0 (1.0,0.05)
20.0 (1.0,0.067)
```

```
25.0 (1.0,0.084)
30.0 (1.0,0.1)
35.0 (1.0,0.117)
40.0 (1.0,0.134)
45.0 (1.0,0.151)
50.0 (1.0,0.167)
61.0 (0.984,0.201)
69.0 (0.986,0.227)
96.0 (0.938,0.301)
138.0 (0.833,0.385)
143.0 (0.839,0.401)
176.0 (0.852,0.502)
207.0 (0.831,0.575)
217.0 (0.829,0.602)
263.0 (0.776,0.682)
```

The top two sections are the interpolated precision recall numbers for the query results. The last section are the actual results from the test.

This gives you the results for a single query. However, it is also desirable to get the averages over the results for all the queries. The `PRResultUtils` class will perform such calculations. It takes the entire array of PRResult objects returned by Precision-Recall and summaries them. The summary method of PRResultUtils produces a string that looks like the following:

```
#Recall -- Precision Averages
0.0     0.875
0.1     0.7431
0.2     0.3964
0.3     0.2344
0.4     0.2131
0.5     0.2131
0.6     0.2074
0.7     0.0
0.8     0.0
0.9     0.0
1.0     0.0

#Number of documents -- Precision Averages
5.0     0.75
10.0    0.7431
15.0    0.7431
20.0    0.7431
30.0    0.7431
```

```
100.0   0.2131
200.0   0.2077
250.0   0.1939
300.0   0.0
350.0   0.0
500.0   0.0
1000.0  0.0
```

HINT: PRResult is serializable, as are arrays. That means you can get the results from a PrecisionRecall test, and save them to disk using `iglu.util.IOTools.ObjectToFile(...)` and get them back later using `iglu.util.IOTools.ObjectFromFile(...)`.

Chapter 7

iglu.jdbc
Database Connectivity Utilities

This package contains utility classes for accessing databases. It has been extensively tested with Postgres and contains a few Postgres specific methods, but should work with any JDBC connection. Probably the most useful class is `iglu.jdbc.JDBCUtils`, which hides the Exception handling and converts it into optional debug statements.

The `SQLCascadingColumn` class provides a way for the user to create a multi-colum view of information in the database, where clicking on an item in one column will trigger an sql query which fills in the next column. Figure 7.1 shows this class at work.

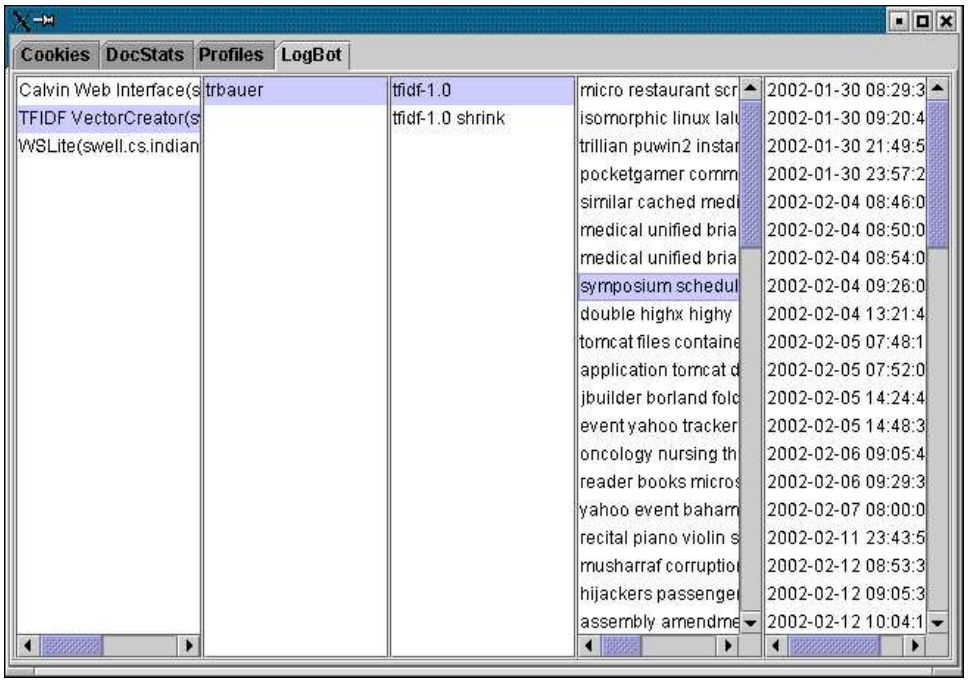


Figure 7.1: Illustration of Cascading Columns

Chapter 8

iglu.net
Internet Utilities

This package contains classes which aid with TCP/IP networking. The main class currently in this package is a WWW proxy server. This proxy server can be run standalone using `iglu.examples.ProxyServerExample` or incorporated into another program. This proxy server is vastly improved over the previous version. It's faster and handles a wider range of web sites.

This aids in Information Retrieval research by providing a tool by which you can observer the documents passing through a user's web browser and record, index, analyze those documents to develop user profiles or build a document database.

See the javadoc for more information.

Chapter 9

iglu.polka
Swing Extensions

This package contains a number of extensions to the Swing classes. Probably the most interesting set of classes in this package are the graphing classes. You can use the classes to create customizable two dimensional line graphs. The basic process of creating a graph is as follows:

1. Create a unique GraphData2D object for each line you want on the graph.
2. Load these object with the points on the graph and set any customizable parameters you want.
3. Create the PLineGraph object and set customizable parameters.
4. Add the GraphData2D objects to the graph.

PLineGraph has the additional feature that it can highlight which point the mouse pointer is closest to, and provide that point's data for display or use elsewhere in the program. A sample line graph is shown in Figure 9.1. This graph was generated from the main function of PLineGraph, which thus provides a good base example for how to use these classes.

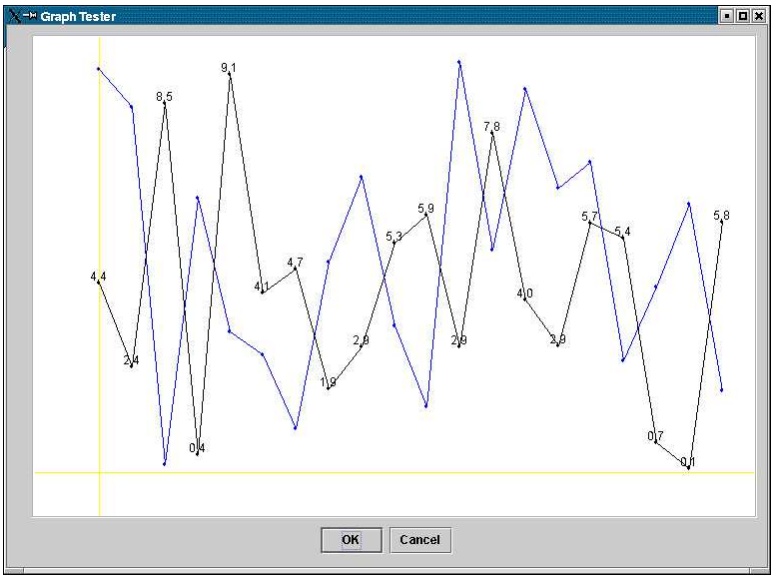


Figure 9.1: A Sample Line Graph

Chapter 10

iglu.util
Utility Classes

This package contains a host of utility classes. All of them are described in their javadoc. A few of them will get further description here.

10.0.8 Debug

This class is contains a number of methods useful for getting debugging information.

tic/toc

This is used for timing different parts of the code. Debug.toc() return the number of seconds since the last time Debug.tic() was called.

setLevel(string[]), debugOut(string, string)

These methods are used to put in debugging statements which are conditionally printed out. setLevel will specify which statements should be printed and debugOut actually prints them out. So a call like `Debug.setLevel(new String[] ‘‘JDBCUtils’’, ‘‘FileBTree’’)` will cause the program to print out any debugOut statements which have the format: `Debug.debugOut(‘‘JDBCUtils’’, . . .)` or `Debug.debugOut(‘‘JDBCUtils’’, . . .)`. This is useful because

you can leave debugging statements in the code but "turn them off" in the sense that they will not show up.

10.0.9 ObjectPager and descendents

In some situation, you may want thousands of objects in memory at one time, more than the memory can hold. ObjectPagers gives you a repository to put such objects, and retrieve them when needed. The FileObjectPager is the most useful of these classes. It swaps object out to disk and brings them back in when requested. You give an object to an ObjectPager and you get and ID back which you can later use to retrieve the object.

The greatest use of this class in IGLU is in the TermVector class. TermVectors are transparently backed by an ObjectPager. By default, it is a RAMObjectPager, which does not swapping, and only allows you to have around 10,000 or less large vectors. But if you tell the TermVector class to use a FileObjectPager, *making no other changes to you code*, you can create a virtually unlimited number of TermVectors without worrying about swapping them in an out of memory.

10.0.10 TagTokenizer

Just like `StringTokenizer`, but can use string, not only characters as delimiters.