Isa-program Documentation

Release 0.3.0

Oscar David Arbeláez Echeverri
German Augusto Osorio Zuluaga

April 16, 2016

Contents

lsa-program is a suite of tools to work with latent semantic search engines and other search engines as well, now it offers index database management computation of lsa ranking matrices and query tools.

Contents:

CHAPTER 1

Quickstart for users

1.1 Setup

In order to use the **lsa-program** you need to setup and configure a **mongodb** database and have it run in the default port as the application database connection is not yet configurable.

You can also install the **lsa-program** from PyPI using the pip tool

```
$ pip install lsa-program
```

Furhtermore, for language support you need to install some dictionaries and a spell checking engine:

```
# Arch
sudo pacman -S enchant aspell-es aspell-en aspell-fr aspell-it aspell-pt
```

```
# Ubuntu
sudo apt-get install enchant \
                     aspell-es aspell-en aspell-fr aspell-it aspell-pt
```

Once you have the mongodb daemon running in your system you can start building your models using any of the three supported formats:

1. Froac
2. Isi plain text
3. Bibtex

1.2 Repository preparation

Once you're done with the setup, you should start preparing your documents to feed the index database, you can order your sources anyway you want in your filesystem because the **lsa-program** cli uses a glob matching interface so you can find the documents you want, an example can be the example repository provided along with this package:

```
$ tree ../data
../data
+-- bib
|   +-- oaa.bib
+-- froac
|   +-- froac1
|   |   +-- 30Algebra relacional. OperaciÃ³n ComposiciÃ³n.xml
|   |   +-- 71Estandares.xml
```

```
|   |   +-- 7Programacion lineal.xml
|   |   +-- 83Modelo - Vista - Controlador.xml
|   |   +-- 85Video objeto de aprendizaje CrazyTalk.xml
|   |   +-- 86Aprendiendo con Cuadernia.xml
|   |   +-- 87Introduccion a eXe Learning.xml
|   |   +-- 89OA 1 Clase UTP.xml
|   |   +-- 92Prueba parcial 1.xml
|   |   +-- 97Respuesta libre en circuitos de primer orden.xml
|   |   +-- ...
|   +-- roapManizales1.xml
|   +-- roapManizales2.xml
|   +-- roapManizales3.xml
|   +-- roapManizales4.xml
|   +-- roapManizales5.xml
+-- isi
    +-- isi.txt
```

1.3 Populate the database

Once you have your dataset organized you can populate your database using the *lsapopupate* program.

```
lsapopupate --xml 'data/*/*.xml'
```

You can also stipupate a database name and specify if you want to wipe the database, if you want to combine records from different kinds of databases, you can do so by reruning the *lsapopupate* tool with the *–no-wipedb* flag,

```
lsapopupate --isi --no-wipedb 'data/*/*.isi'
lsapopupate --bib --no-wipedb 'data/*/*.bib'
```

However, whenever using this approximation, beaware of the record duplication as the hashing algorithms used to detect duplicates are different for the different kinds of record files.

1.4 Build a model for the database

Once your database is populated you can build a model or ranking matrix for your database using the command:

This program will create versioned models so that you can build different versions, or query with one model when another one is still being built.

Beaware that this is the most time consuming operation in the suite as it involves inverting a several thousand rank matrix.

1.5 Query the model

Once you have built the model you can start performing queries, you can do so by:

```
lsaquery search terms
```

This will perform a query to the latest available model in the model database.

1.6 Next steps

1. Learn more about latent semantic analysis
2. Learn more about index databases
3. Learn more about the different supported fileformats

CHAPTER 2

API Reference

lsa-program main part are the parsers that convert the different kinds of database documents into manageable dictionaries that only contain the interest metadata fields. Those are implemented in the `record` module.

2.1 The record API

The main class of the record API is the `RecordParser` class, which outlines an api that parses data out of a raw string, or raw data structure into a dictionary with the desired interest fields, details about how to extract that information will go into the `RecordParser` child classes.

Furthermore, the `RecordParser` is complemented by the `RecordIterator` class, that outlines an interface to iterate over a file containing several records and returning (yielding) all the records in a memory efficient fashion.

2.1.1 Implementations of the record API

Parser implementations

Iterator implementations

2.2 The utility module

2.3 The scripts package

2.3.1 Database manipulation

Utils to work with a mongo database, it contains a global connection to the database so that a new one is not created with every request which is a huge overhead. Furthermore it has a tool to use a pymongo collection as a context manager.

`lsa.scripts.dbutil.`**`collection`**(*name*, *dbname='program'*, *delete=True*)

Yields a mongo collection with name the given name in the specified database it has the advantage of not having to create the collection everywhere in the program.

Parameters

- **`name`** (*`str like`*) – name of the collection
- **`dbname`** (*`str like`*) – name of the database to get the collection from

- **delete** (*bool*) – either delete the content of the collection or not

Returns collection as a context manager

Return type ContextManager

lsa.scripts.dbutil.**collection_name**(*name*)
Prepends 'lsa-' to the given name, so that all collections for the lsa program have consistent names.

2.3.2 Script entry points

The entry points are organized in modules, this leads to some code duplication but it can be reduced in the future. The populate script, which yields the *lsapopulate* command is located in the `populate` module, and contains the information descripted bellow.

The model script, which yields the *lsamodel* command is located in the `model` module, and contains the information descripted bellow.

The query script, which yields the *lsaquery* command is located in the `query` module, and contains the information descripted bellow.

CHAPTER 3

Developer quickstart

lsa-program is a free and open source software, contributions are very wellcome and you can start contributing through the common *github* pattern.

3.1 Setting up the development environment

First you need to clone the repository:

```
git clone https://github.com/odarbelaeze/lsa-program.git
# Alternatively you can fork and clone your own fork
git clone https://github.com/<username>/lsa-program.git
```

Then the most recomendable way for you to do development is in a virtual environment, we will assume that you are familiar with *virtualenvwrapper*

```
cd lsa-program
mkvirtualenv -a $PWD -p $(which python3) -r piprequirements.txt lsaenv
pip install -e .  # This command will also install dependencies
```

3.2 Running tests

When contributing to an open source project, it's crucial that you are able to run its test suite, this project suggests pytest as testing framework and you can use its test runner to run our tests, in order to run our test you need to install testing dependencies, you can do so automatically using the *setup.py* script

```
python setup.py test
```

Otherwise, you could manually install the testing dependencies and run tests manually,

```
pip install pytest
python -m pytest
```

Furthermore, you can install pytest plugins such as **coverage** via the **pytest-cov** plugin and run the test suite using their custom flags.

3.3 Contributing to the code

3.3.1 What you can contribute

You can contribute to the code, not only by adding new features and smashing yourself to the keyboard at the core of the program, some examples of contributions you can make are the following:

- Write documentation
- Write tests, maybe while you are trying to understand what the code does
- Refactor code, here and there there are optimizations that can be made

Otherwise, you can expand the funcionality of th erecord parsers by adding support for more document kinds.

3.3.2 How to contribute

This project supports the feature branch and pull request (PR) way of doing contributions through git, you start by adding a new branch to your repo,

```
# While in the repo folder
git checkout -b <a-name-that-outlines-your-contribution>
# Do some ground work
git push -u <origin> <a-name-that-outlines-your-contribution>
```

Then you proced to create the pull request in github, right away, so you can start communicating with the core developers and other members of the community.

3.3.3 Recomendations and code guidelines

Honoring code guidelines is a key part of contributing to a project, that's because fixing indentation and correcting line lengths is assumed by git as a rewrite, and the authorship of the contributed code might be diluted. This project includes a *.editorconfig* file that can be used by modern text editors to automatically keep coding guidelines for different filetypes.

Although the code coverage of this project might be low at some point, contributors are encouraged to write tests for their features, and also, regression tests for the code that is already there. Test are writen using **pytest** which is a very easy to use testing framework, to start you just need to drop a function whos name starts with *test_* into one module within the *tests* folder, and the suite will automatically pick up your testing code.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

l

C

L