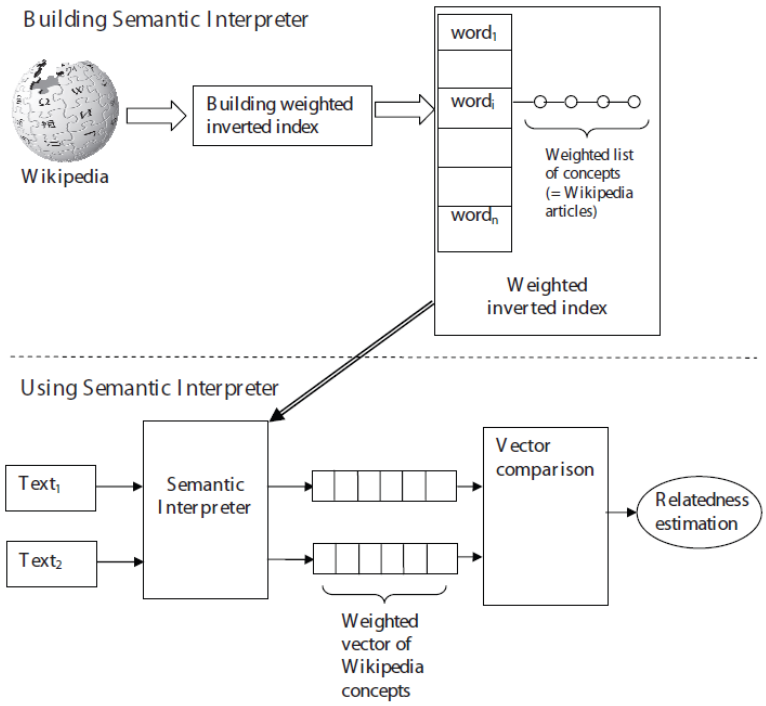
Wiki Knows

Python Implementation of Wikipedia-based Explicit Semantic Analysis

Department of Computer Science
Technion—Israel Institute of Technology



*Implemented by **Roei Bar Aviv** and **Ivan Nesmeyanov***
2013

Content

Overview

There are many methods for computing semantic relatedness of natural texts. We re-implemented Explicit Semantic Analysis (ESA) – a method, which represents the meaning of texts in a high-dimensional space of concepts derived from Wikipedia. This method is described in paper "Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis 2007" by Evgeniy Gabrilovich and Shaul Markovitch and implemented in perl programming language. More information at: http://www.cs.technion.ac.il/~gabr/resources/code/wikiprep/.

System requirements

First ESA implementation written in perl and were hard to understand, support and extend. Our goal was write clean implementation is python (which is much easier to read), which will be easy to use, extend and tune. More specific requirements:

- Simple code (no long all-in-one functions or god objects)
- Clear interchangeable working blocks.
- Same mode of operation as original implementation
- Mechanism of tuning (finding better implementations of working blocks)

Implementation Considerations

We wanted to use as many existing solutions as possible. And keep our code as glue between those components if we didn't find good and simple implementation for some block we used simple implemented stub with clear interface.

There are number of intermediate steps we need to do, each of them has number of implementations

1. Reading Wikipedia xml dump file
2. Cleaning wiki markdown (remove wiki formatting and leave plain text)
3. Extract links from wiki markdown
4. Saving parsed documents to xml file
5. Stemming of article text
6. Numeric calculations
7. Inverted Index RAM Representation
8. Inverted Index Disk Representation

Each step implementation should be easily switched to another one, which will be faster or better, than existing.

Used Libraries

We choose next libraries for intermediate steps implementation

Reading Wikipedia xml dump file	**xml.etree** python library for xml parsing http://docs.python.org/2/library/xml.etree.elementtree.html#
Cleaning wiki markdown	**WikiExtractor** The Wikipedia extractor tool generates plain text from a Wikipedia database dump, discarding any other information or annotation present in Wikipedia pages. http://medialab.di.unipi.it/wiki/Wikipedia_Extractor
Extract links from wiki markdown	**Mwparserfromhell** Python package that provides an easy-to-use and outrageously powerful parser for MediaWiki wikicode http://pythonhosted.org/mwparserfromhell/
Saving parsed documents to xml file	**xml.etree**
Stemming of article text	**The Porter Stemming Algorithm** The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems http://tartarus.org/martin/PorterStemmer/
Numeric calculations	**Scipy** SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering. It is also the name of a very popular conference on scientific programming with Python. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation http://www.scipy.org
Inverted Index RAM Representation	**Scipy** Scipy supports sparse matrices and vectors. It allow representing a vector (or matrix) as sparse while it supports all numerical calculations while keeping the sparse representation.
Inverted Index Disk Representation	**Pickle** algorithm for serializing and de-serializing a Python object structure. http://docs.python.org/2/library/pickle.html

Terminology

Wikipedia article or Wikipedia page

Xml element in Wikipedia dump contains Meta information and article text in wiki markdown.

Example:

```
  <page><title>Politics</title><ns>0</ns>
    <id>22986</id><revision><id>552015170</id>
      <parentid>551999101</parentid>
      <timestamp>2013-04-24T20:34:13Z</timestamp>
      <contributor><username>Fat&amp;amp;Happy</username>
        <id>9861893</id>
      </contributor>
      <minor/>
      <text xml:space="preserve" bytes="40425">{{About||the political
magazine|The Politic|other uses}} {{Politics}} …
```

Wikipedia document

Our xml representation of Wikipedia topic contains necessary information and regular text.

Example:

```
<?xml version="1.0" ?>
<wikirep>
<doc id="22986" rev_id="552015170" title="Politics">Politics…
</doc></wikirep>
```

Wikipedia Concept

Statistical representation of Wikipedia topic, maps every word (term) in article to number of its occurrences in article. Example:

> title: “Politics”,
> words: {“ government” : 36, “democracy”: 6}

System description

System is designed to be a research platform of SemanticIntepreter. It is separated layers to allow testing by using different concepts collections, parsing tools, stemmers and comparison metrics.

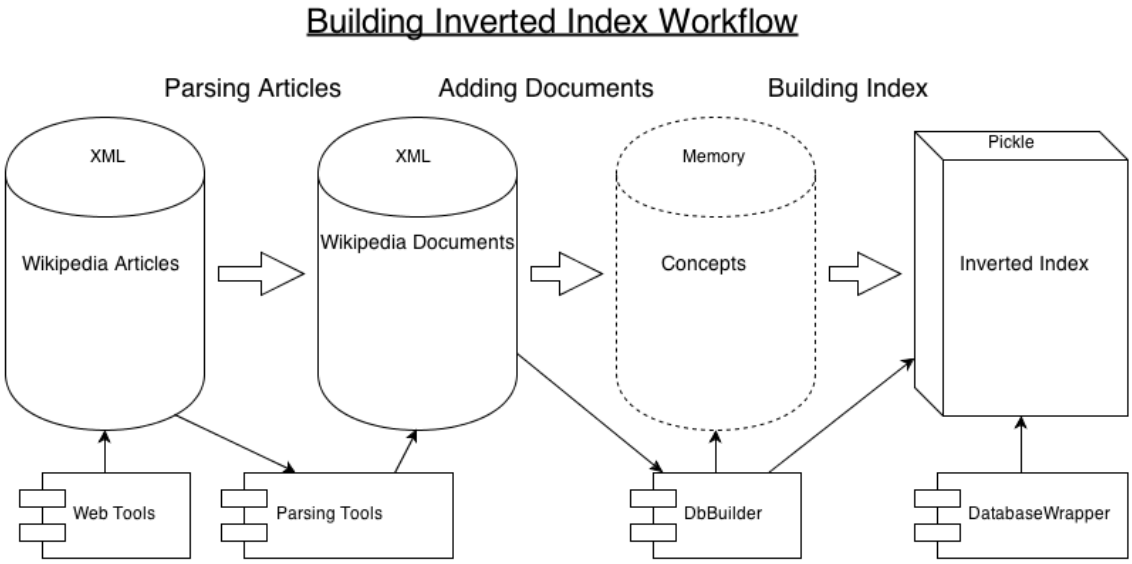
As such, it has a main executable which allows to:

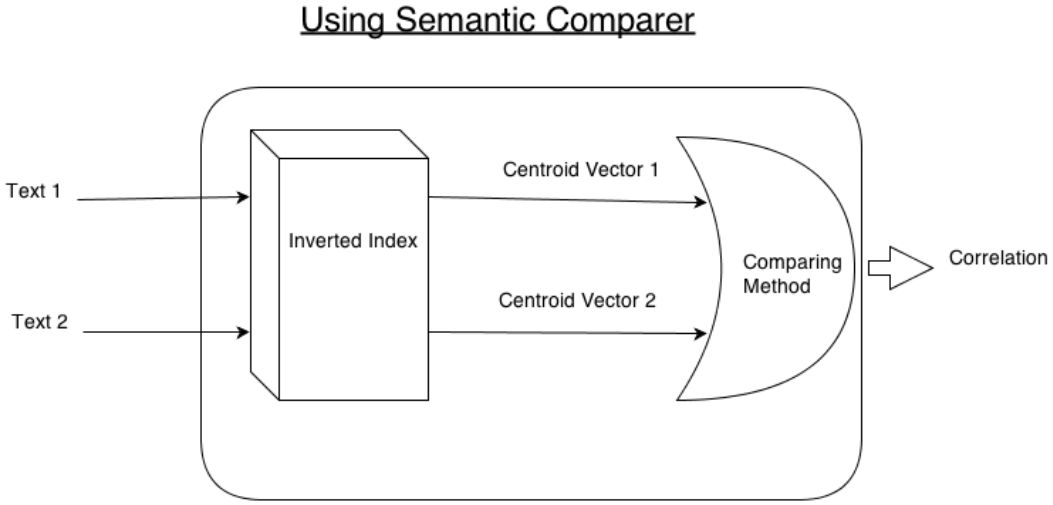
1. Create a collection of Wikipedia articles.
2. Parse a collection and extract its text and links to Wikipedia documents.
3. Build inverted words index using a configurable stemmer.
4. Compare texts based on word index using a configurable comparison metric.

First we describe how each stage of the execution is handled, and later we would describe the databases being used during the process.

Start with diagram, which describes two main workflows. The dashed cylinder exists only in memory.

Diagram





Mode of operation

1. Downloading and building Wikipedia dump from provided article names.
2. Parsing the Wikipedia dump to our format.
3. Building Inverted Index
   3.1. Stemming articles
   3.2. Getting links from articles
   3.3. Simple Pruning
   3.4. Creating DF index, which counts number of articles term appears for every term.
   3.5. Building TD-IDF index for all concepts
4. Compare between texts
   4.1. Stemming input text with same stemmer, which has been used in 3.1
   4.2. Calculate centroid for text according to vectors values of every actual term (not stemmed)
   4.3. Comparing centroids using cosine metrics

Create a collection of concepts file

This part could be completed by in two ways:

1. Use **make_dump** command which download specified articles from Wikipedia site, merges them into one file and compresses it as Wikipedia dump file. This mode is used mainly for testing purposes, and allows use of various small collections of Wikipedia concepts.
2. Download Wikipedia dump from http://en.wikipedia.org/wiki/Wikipedia:Database_download. According to our calculations (described under Future Implementation) this step requires using databases as execution is expected to be memory consuming

Parse a collection and extract its text and links

Implemented serially, by iterating through Wikipedia article in dump, (although could be implemented with parallel threads), and completed by using **`parse_tools.iterate_wikidocs_from_dump`** function. The results of the parsing are saved to a database (or file). This stage uses external tools such as WikiExtractor and mwparserfromhell.

Build inverted words index

Use the parsed collection in order to build inverted words index while using a configurable stemmer.

	Concept 1	Concept 2	Concept 3	….	Concept N
Word 1					
Word 2					
Word 3					
…					
Word M					

This is done by iterating all saved wiki concepts, represent each of them as “bag of words” and build words inverted index (columns are the wiki concepts while each word English is represents a row).

For now, we use only subset of words in English, which is the union of all words in the input concepts. If Words that are not in this subsets but given in a text for comparison are ignored.

Compare texts based on word index

Use the inverted words index in order to:

1. Compare texts using a configurable metric.
    a. Stem the input text
    b. For each word in text select the corresponding wiki concepts vector (As saved in database)
    c. Calculate centroid for all words in text
    d. Evaluate texts correlation by calculating the distance between the vectors using the configured metric method.
2. Analyze text and find most relevant wiki concepts.

a. Stem the input text
b. For each word in text select the corresponding wiki concepts vector (As saved in database)
c. Calculate centroid for all words in text
d. Select N concepts with the highest values

Databases

The system is designed to have two databases.

1. Building database - used for building the word inverted index, as the amount of concepts and words required to build word index from whole Wikipedia does not allow the whole process to run in memory. Currently we see no further use to this database, and unless decided otherwise it will be deleted once the word inverted index is calculated.
2. Words Invereted Index database – used to save to results of built database. This flexibility allows, for example, avoiding building the word inverted index in order to compare stemmers.

Please note that both databases are currently unimplemented, but the infrastructure is set in code such that we only have to replace saving to file with saving to database.

Parsing Layer

This layer is intended to allow us to translate given wiki dumps into our domain, its output is later used by the model to create word inverted index.

The parsing layer cleans Wikipedia dumps (which is a collection of Wikipedia pages in Wiki Markup Language) and creates parsed XML file, which is comprised of Wikipedia concept's ID, title, revision ID and text.

In future implementation, this layer will output to a database in order allow handling of big data.

The parsing layer is comprised of:

- WikiDocument - Wikipedia concept wrapper class based on our model.
- WikiTextProcessor - Handles wiki markup language input file and updates the WikiDocument.

Implementation Details

Folder structure

Folder was organized according to pypi module for creating and distributing a distribution. Using the python setup tool simplifies the process of project installation and makes it completely automatic.

This setup allow to automatically install dependencies when your package will be installed and include information about dependencies (so that package management tools like Pip can use the information). It takes a string or list of strings containing requirement specifiers.[1]

[1] http://guide.python-distribute.org/creation.html

Root

```
setup.py   - application setup file
dist       - zipped distributive
docs       - documents and papers
refs       - reference code and samples
wikirep    - main code place
     model      - core modules
     parsers    - parsing, pruning, cleaning, extraction and downloads
     test       - test collection
     wiki_knows - python interface module
     main.py    - command line interface
```

Model folder

build_utils.py	-	Helpers for inverted index build process
concept.py	-	Processed Wikipedia article representation
database_wrapper.py	-	Build Inverted Index Table API
db_builder.py	-	Can build Inverted Index Table
semantic_interpreter.py	-	Can compare text using given Wrapper and compare method
simple_splitter.py	-	Text splitter
stemmers.py	-	Text stemmer collection
wiki_doc.py	-	Parsed Wikipedia document representation

Parsers folder

WikiExtractor.py	-	tool for cleaning wiki markdown
parse_tools.py	-	parsers collection
web_tools.py	-	work with Wikipedia site
wikitext_processor.py	-	powerful tool for cleaning wiki markdown

Main Modules

- **WikiDocument** – represents parsed Wikipedia document as described in terminology
- Concept – represents Wikipedia concept
- **DbBuilder** – Converts **WikiDocuments** to Concepts on addition, and creates Inverted Index Table on build command.
- **DatabaseWrapper** – represents Inverted Index Table.
- **SemanticComparer** – uses provided **DatabaseWrapper** and compare method for texts comparison
- **WikiKowledge** - The highest class level in the system. Provides Python API to all system functionality.

WikiDocument

There are number of different texts representations for each Wikipedia topic:

Wiki Text	Original raw text including wiki markup language
Clean text	Text after parsing the Wiki text
Stemmed text	Text after stemming the clean text

DbBuilder

Provides API for adding WikiDocuments, each document is read and converted converts them to Concepts which is a bag of words representation of the input. Once all documents are added, we can execute the build command which will build a DatabaseWrapper.

At first, it was implemented using numpy, but since numpy doesn't support sparse vectors/matrices and in order to have a scalable system this implementation was replaced with scipy.

As mentioned, this class currently executes in RAM and should be modified such that it will database for creating DatabaseWrapper.

DataBaseWrapper

Wrapper class with nice API that wraps a sparse matrix representing the words inverted index, and can also be saved to file. This class handles the extraction of wiki vectors from the database.

Provides API for querying Inverted Index Table, can calculate word vector and text centroid.

Currently implemented w pickle, in future implementations we should use database for temporary results to reduce memory consumptions.

SemanticComparer

Uses provided DatabaseWrapper and compare method for texts comparison.
Independent of their implementation.

WikiKnowledge

The highest class level in the system. It is initiated with a wiki dump file, and creates a SemanticIntepreter from the dump, by doing:

- Create dump from articles list (if a dump already exist, this step may be skipped)
- Iterate all wiki documents in the dump file, for each do:
    - use parse_tools in order to generate a WikiDocument
    - use WikiTextProcessor in order to extract the links and text from the raw document (the raw is formatted in wiki markup)
    - serialize the created WikiDocument into an element in the current opened XML
- Build SemanticIntepreter from the XML file
    - for each wiki element in the input XML
        - Create WikiDocument
        - Add the WikiDocument to the DbBuilder
            - Apply text stemming on the wiki document
            - Use the stemmed text to create a wiki concept (A concept is just a bag of words, which represent the corresponding wikipedia page)
            - Add the concept to DB concept list

    - build the DB
        - Build word_index (iterate all concepts and create the overall words index)
        - Build index_by_word (convert list of words to dictionary {word => it's index})
        - Build weight table
- Use SemanticIntepreter in order to calculate a given text vector according to the created words database
    - Note the SemanticIntepreter is created with a stemmer, which will be used for stemming the input text
    - The stemmer used here is unrelated to stemmer used for building the database

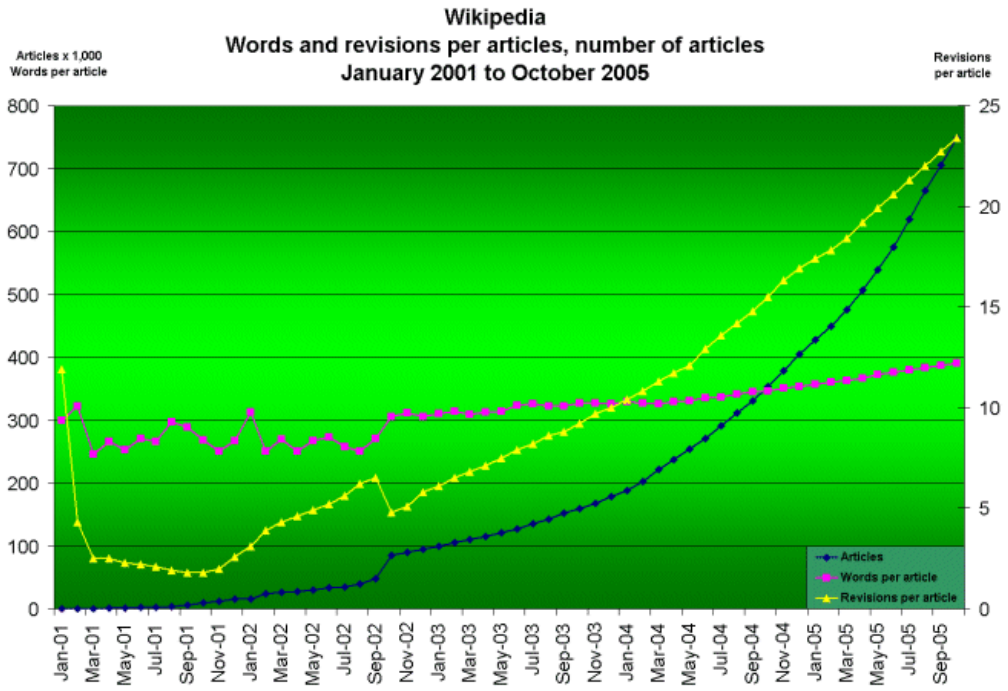
Future Roadmap

Use real database

So far all the development was made using small subsets of Wikipedia concepts, since wiki_knows runs in memory and wiki dump size is 42GB.

Wikipedia as of October 2006 had 1.4 million articles with an average length of 3,300 characters.

In the graph below, which deals with the English Wikipedia, the number of articles (blue, multiplied by 1,000) and revisions (yellow, right hand scale) and words per article (purple).



Although we use scipy[2] sparse vector, prune lots of articles we tend to handle 5,000 words x 1,000,000 articles, and each value of the sparse matrix is a float (at least 4Bytes) – that's 20GB and cannot be executed in memory, or at least not be designed as one.

[2] scipy – python library for science computations, see http://www.scipy.org

Normal Person (Graduate) = 5,000 to 6,000+ words
University Professor = 15,000+ words

In order to handle this amount of data and calculate the TF-IDF we must use database. The database is used for both saving the TF-IDF as well as creating it.

Better Implementations for intermediate tasks

Not all of implementation we chosen are good enough. For instance if we want to use real database and parse topics to database we need to implement DbBuilder using real database such as sqlite or mongodb. Another issue is Inverted Index Disc representation. Pickle is not good enough, we need to define database format, not language specific, so other NLP tools (in different programming languages) could benefit from the system results.

Research

There are number of high level parameters, which can be tuned to achieve maximal accuracy.

Pruning

Pruning is document filtration subsystem: when we add WikiDocument to DbBuilder it decides according to chosen pruning mechanism which Wikipedia topics should be included in Inverted Index which should be skipped. In original implementation next topics were skipped: Dates, Topics with small number of incoming links, topics with small number of words in them.

Stemmer

Which stemmer should be used, such that the results are optimized, compared to human judgment. This can be accomplished using stemmers pipeline, see class ComplexStemmer.

Vectors comparison

There are number of metrics, which can be used to calculate texts correlation there need to evaluate and find best of them.

Links

- Project Source Code: https://github.com/roeiba/WikiRep/
- Project Wiki: https://github.com/roeiba/WikiRep/wiki/
    - Installation Instructions
    - Usage Examples
    - More Documentation
- Sample usage: https://zvulon.pythonanywhere.com/
    - Web site with small index (20 concepts), which demonstrates analyzing process

References

- Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis[Evgeniy Gabrilovich and Shaul Markovitch 2007]

- Wikipedia-based Semantic Interpretation for Natural Language Processing [Evgeniy Gabrilovich and Shaul Markovitch 2009]