Aardvark: An Acronym Expansion tool

Benjamin D. Turtel bturtel@gmail.com

Courant Institute, New York University

Kshitiz Sethia kshitiz.sethia@nyu.edu

Courant Institute, New York University

Dennis Shasha (Advisor) shasha@courant.nyu.edu

Courant Institute, New York University

Abstract

We present here a web-based tool for expanding acronyms in academic texts. The tool takes in a PDF document, finds acronyms in it and then expands them based on the context of the acronym’s occurrence. An acronym can have multiple expansions, we predict the most likely one based on similarity measures between documents we have already seen and the current acronym’s context. This problem is very similar to the Word Sense Disambiguation problem in Natural Language Processing.

**Keywords:** Natural Language Processing, Word Sense Disambiguation

# Introduction

When reading a text from a field the reader is not familiar with, we often come across acronyms for which we do not know the expansion. This tool helps by choosing amongst the many possible expansions of the acronym, and presenting the user with it.

This paper is organized as a record of the experiments conducted, and as a user-manual for the back-end code.

# Architecture Description

The web interface is launched by running flask\_app.py using the command: “python flask\_app.py”. The requirements for running the script are present in requirements.txt. It is recommended to install them in a virtual environment. All executable scripts are run from the “server” path. Some data generation is required before you can run the script.

## AcronymDisambiguator

The main class needed to do the disambiguation work is AcronymDisambiguator. It has three main components:

1. TextExtractor: Extracts text to be processed from any kind of document.
	1. Example: TextExtractors/Extract\_PdfMiner.py
	2. If you want to implement a new text extractor, simply inherit the TextExtractors/TextExtractor class and implement the methods in it.



* 1. It is recommended that all text inside the system be in Unicode. Please return a Unicode string from any new TextExtractor. To convert any type of string to a Unicode string use TextTools.toUnicode method (extend as needed).
1. AcronymExtractor: Takes the Unicode text extracted from a document and extracts acronyms from it.
	1. Example: AcronymExtractors/AcronymExtractor\_v1.py
	2. To implement your own, inherit AcronymExtractors/AcronymExtractor class and implement its methods.



1. AcronymExpander: Takes a list of acronyms to be expanded from AcronymExtractor, text from TextExtractor and predicts expansions for the acronyms.
	1. Example: AcronymExpanders/Expander\_fromText.py
	2. To implement your own, inherit AcronymExpander or PredictiveExpander from AcronymExpanders/AcronymExpander.py and implement the methods.
		1. AcronymExpander is for non-predictive expanders, without a confidence measure for their predictions.
		2. PredictiveExpander inherits AcronymExpander, and also provides a confidence score for each prediction
		3. Put a new Enum in AcronymExpanderEnum, and a new conditional in AcronymDisambiguator.\_createExpander.
	3. NOTE: AcronymExpanders are implemented to be chained together. So you can have an “array” of different expanders. Each expander can choose whether to either expand an acronym based on a confidence measure or other factors. If an expander does not expand an acronym, the next one will try to.

### Using an AcronymDisambiguator

An object of type AcronymDisambiguator can be initialized as follows:



As you can see, the disambiguator takes one TextExtractor, and one Acronym Extractor, and an array of AcronymExpanders, denoted by their Enum. The enum is converted to an expander by AcronymExpander.\_createExpander.

## Data Sources

The project uses the following data sources:

1. articles: a csv with article\_id and article\_text
	1. example: server/storage/data\_all/scraped\_articles.csv
	2. This can be stored in multiple files to facilitate easy upload/addition of articles
	3. The files with the csv data can be populated in string\_constants. file\_scraped\_articles\_list
2. Definitions: Contains expansions of acronyms and associated article\_id
	1. a csv file with acronym, acronym\_expansion and article\_id
	2. example: server/storage/data\_all/scraped\_definitions.csv
	3. This can also be stored in multiple files, populated in string\_constants. file\_scraped\_definitions\_list
3. Article Info: Extra info about articles, which might be used in the future
	1. csv storing article\_id, article\_title, article\_source
4. MSHCorpus and ScienceWISE: benchmarking datasets for standard acronym disambiguation datasets. Read more at

## Data Creators

The project has just got the scripts and the data sources to begin with. To run the server, we need to have all data sources stored in binary format, and any models need to be pre-built. All the scripts doing this are present in DataCreators. These scripts need to be run with the “server” folder in PYTHONPATH.

Here is a list of scripts which will do the needful:

1. ArticleDB, AcronymDB and ArticleInfoDB: These scripts make binary versions of the data sources above and encode everything in binary.
2. LDAModel.py: Trains an LDA model using data created by ArticleDB. Needed for any model using Latent Dirichlet Allocation of the articles.
3. MSHCorpus.py and ScienceWISE.py: reads the respective datasets, and builds data sources similar to 1 above. Needed for benchmarking the performance of the acronym expansion framework.

## Initializing the project

Here are the steps to start the project from the bare essentials:

1. Install python2.7 and pip.
2. Navigate to “server” directory in bash.
3. Create a virtual environment with virtualenv, let’s name it “pyvenv”. See <http://docs.python-guide.org/en/latest/dev/virtualenvs/> for details.
4. Activate the virtual environment.
5. Install all required modules with the following command. You will need to be on a linux machine to install gensim correctly.

“pip install –r requirements.txt”

Gensim on linux is only needed to train the LDA model. Once trained, the models can be used on any platform.

1. navigate back to the “server” directory
2. run “python flask\_app.py”

## Benchmarking

We have evaluated our framework on two datasets: MSHCorpus and ScienceWISE corpus. Both are human annotated. Here’s a description of both of them:

1. MSH WSD Corpus:
	1. This is a word sense disambiguation dataset from the medical domain. Hosted by the US National Library of Medicine.
	2. See storage/data\_all/MSHCorpus/README.txt for details
2. ScienceWISE dataset:
	1. This is an entity disambiguation dataset hosted by eXascale Lab, University of Fribourg
	2. The dataset is downloadable from <https://github.com/XI-lab/entity-disambiguation-data-ecir2013>
	3. The dataset gives arxiv ID of a pdf with an acronym and expansion pair.
	4. The pdfs need to be downloaded separately, this is done by DataCreators/ScienceWISE.py

### TextExtractors used:

1. PDFMiner: This is a text extractor for PDF documents written in python. We use the TextConverter module in it, instead of HTMLConverter. HTMLConverter was being used earlier, and gave HTML tag artifacts which broke the pipeline of the rest of the framework.

### AcronymExtractors used:

Here’s a list of the AcronymExtractors tried:

1. V1: an acronym extractor using regex to extract acronyms of length 3-8, and discarding acronyms spelling same as English words.
2. V2: like v1, but keeps acronyms spelling same as English words. Example: PEP (phosphoenolpyruvate), MOO, etc.
3. V2\_small: like v2, but detects acronyms 2-8 in length
4. V3\_small: supports length 2-8, and treats words with more than half length capitalized as acronyms

### AcronymExpanders used:

Here’s a list of AcronymExpanders tried:

1. fromText: expands the acronym within the document’s text, with any phrase starting with the same initial words as the acronym
2. fromText\_v2:
	1. supports expansions which span multiple lines in text.
	2. support for acronyms like **I**ntegrated **Def**inition (IDEF) in which one word can contain multiple alphabets from the acronym.
3. Tfidf\_multiclass: This expander gets the documents with the same acronym and associated expansions as training data, and tries to find the document most similar to the current document. The similarity between documents is measured by a one-hot encoded representation of the document, weighted by tfidf, fed to a linear multiclass Support Vector Classifier.
4. LDA\_cossim: This expander gets the documents with the same acronym and associated expansions as training data, and tries to find the document most similar to the current document. The similarity is determined by the cosine similarity between the LDA vectors of the training documents and the current document.
5. LDA\_multiclass: uses LDA vectors for training and current documents, but classifies using a linear multiclass Support Vector Classifier.

### Performance on MSH dataset

The performance of all models was evaluated using ten-fold cross validation, therefore all tried models have a mean and standard deviation of the performance. This helps in judging the reliability of a model and whether it is significantly better than another method. Detailed results are present in Algos.xlsx

The AcronymExpanders in the table below are a list of expanders chained together for one model evaluation.

|  |  |  |  |
| --- | --- | --- | --- |
| Mean (0-1) | Standard Deviation (0-1) | AcronymExtractor | AcronymExpander(s) |
| 0.9092 | 0.007662 | v3\_small | tfidf\_multiclass |
| 0.809068 | 0.011044 | v2\_small | tfidf\_multiclass |
| 0.788255 | 0.011328 | v2\_small | fromText, tfidf\_multiclass |
| 0.769473 | 0.010144 | v2\_small | fromText, LDA\_multiclass |
| 0.767627 | 0.012021 | v2\_small | fromText, LDA\_cossim |
| 0.758359 | 0.014669 | v2\_small | LDA\_multiclass |
| 0.753487 | 0.008364 | v2\_small | LDA\_multiclass |
| 0.750831 | 0.010872 | v2\_small | LDA\_multiclass |
| 0.749395 | 0.008868 | v2\_small | LDA\_multiclass |
| 0.746967 | 0.009885 | v2\_small | LDA\_cossim |
| 0.744544 | 0.011891 | v2\_small | LDA\_cossim |
| 0.734595 | 0.009107 | v2\_small | LDA\_cossim |
| 0.733482 | 0.012735 | v2\_small | LDA\_cossim |

The original algorithm used for this project had a mean accuracy of 0.59 with a standard deviation of 0.021.

### Learning from MSH performance

1. Our scores are close to the state of the art 93.10% accuracy. The ten-fold accuracy on MSH Corpus is most affected by improving the AcronymExtractor. Future work could focus on a machine learning method of detecting acronyms.
2. Despite this high dataset accuracy, we are getting poor performance on real world data. We should include more data sources, so disambiguation models know more acronyms, and have more reference documents for each expansion.

The current infrastructure allows for addition of more data. Simply appending articles to scraped\_articles, scraped\_article\_info and scraped\_definitions would suffice. Running the DataCreators again would incorporate the new data into the framework.

## Other features

1. Unit testing
	1. Unit testing has been included for non-trivial parts of the codebase. These are standard python unit tests. These help in triaging errors and containing regressions.
	2. Tests are located in the “tests” package
	3. A nice example is testing of the v3\_small AcronymExtractor module.
2. Temp scripts
	1. There were a lot of experiments run as a one-time test to see whether useful or not.
	2. These have been kept safely in temp\_scripts.py with ample documentation for each snippet of code.
3. Small data set
	1. There’s a small version of all data being used by the framework.
	2. There are two reasons to keep them:
		1. Small files can be checked-in to github, where this codebase is being maintained.
		2. Whenever there are changes in the framework, verifying all individual pieces are working fine is possible very soon.
4. string\_constants.py
	1. All of the files and strings used by the framework are mentioned in one single file string\_constants.py
	2. This facilitates a single point of change to any configuration of the framework.
5. Using virtual environment in python
	1. Using a virtual environment has many advantages:
		1. Conflicting versions installed on a system do not affect our implementation
		2. Freezing the virtual environment (with “pip freeze”) lets you capture all dependencies of the framework
		3. When using a new system, “pip install –r requirements.txt” can handle all the dependencies of the framework.

# References

1. Li, Chao, Lei Ji, and Jun Yan. "Acronym Disambiguation Using Word Embedding." *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.
2. Ehrmann, Maud, et al. "Acronym recognition and processing in 22 languages." *arXiv preprint arXiv:1309.6185* (2013).
3. Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." *the Journal of machine Learning research* 3 (2003): 993-1022.
4. He, Ying, and Mehmet Kayaalp. "A Comparison of 13 Tokenizers on MEDLINE." *Bethesda, MD: The Lister Hill National Center for Biomedical Communications* (2006).
5. Jimeno-Yepes, Antonio J., Bridget T. McInnes, and Alan R. Aronson. "Exploiting MeSH indexing in MEDLINE to generate a data set for word sense disambiguation." *BMC bioinformatics* 12.1 (2011): 1.
6. Prokofyev, Roman, et al. "Ontology-based word sense disambiguation for scientific literature." *Advances in Information Retrieval*. Springer Berlin Heidelberg, 2013. 594-605.
7. Yeates, Stuart, David Bainbridge, and Ian H. Witten. "Using compression to identify acronyms in text." *arXiv preprint cs/0007003* (2000).
8. Benjamin D. Turtel’s Master’s Thesis on “Acronym Disambiguation” from Courant Institute, New York University.