Exposing Wiktionary Translations

With Performance in Mind

Chris Cargile, Gayathri Santhanakrishnan, Aspen Olmsted
Dept. Computer Science, College of Charleston, Charleston, SC
cargilecm@g.cofc.edu, santhanakrishnang@g.cofc.edu, olmsteda@cofc.edu

***Abstract*— In this paper, we explore some of the challenges to extracting Wiktionary data so it can be used for machine translation. We provide a use case of considerations made in the development of a web service that transforms raw data in a wiktionary xml dump file into a web-service that provides language translation. Our case study focuses on the performance of this service in comparison to a service using remote calls to the Wiktionary API.**

Keywords-machine translation, web service, WordNet, Wiktionary, lexical knowledge base, machine readable dictionary

I. INTRODUCTION

Identifying API's and unifying them is a key to the efforts expended so far in pursuit of an optimal tool for creating a Machine-Readable-Dictionary (MRD)-format, multilingual, lexical, semantic resource (translation tool). While there are API's that use parsing of Wiktionary dumps as a one-time effort and expose the parsed dump through a normalized contract (API), only two are known to do so through a web-API: the Wiktionary API and the API by Yves Bourque [1]. The effort of this study is aimed at making the parsed data of a wiktionary dump available as it relates to the translation of word senses and to make that data available adequately fast for small-scale translations.

As there have been previous studies aimed at developing a schema for improved lexical analysis based on Wiktionary by i.e. Zesch, et al [2] or for aligning multiple resources for lexical analysis by Casses [3], none have defined Wiktionary to be a Lexical Knowledge Base (LKB) as opposed to a Collaborative Knowledge Base (CKB). While a potential cross-lingual LKB may be found by using the Global WordNet, this resource is not free. Rather, the purpose of this study is to utilize free, collaborative resources such as wikitionary to assist in standing up a living and evolving web services resource for quick word translations since none exists, yet.

We have provided the motivations for wanting the tooling this study aims to disseminate. In the next section, we describe tooling that could approximate the result we achieved and limitations associated with those methods. In section III we will provide a motivating example for our project. In section IV, the hypothesis, algorithm, and results will be described.

II. RELATED WORK

In Zesch, et al. [2], existing mechanisms for accessing Wiktionary data are introduced. The mechanisms include the Perl module WWW:Wikipedia, the wikipedia bot framework, and the Wiki Gateway toolbox but these methods are geared towards wikipedia and not towards extracting wiktionary data, specifically. None of these tools are specifically aimed at machine translation or extraction. To avoid parsing the wiktionary dump on-the-fly which results in enormous overhead for each query, [2] recommends using a database that has been populated according to various elements of the wiktionary database dump. Populating a database ensures near constant-time retrieval for each article as a result of applying indexes on certain fields of the database, which occurs in a one-time preprocessing effort.

There are developed API's associated with exploration in various aspects of NLP such as semantic relatedness, information extraction, and semantic information retrieval using wiktionary. The JWKTL API may draw on the capabilities of machine-aided parsing of each individual language's wiktionary to identify local and foreign aspects of a given article or page and is capable of fully merging wiktionaries from two languages. JWKTL [4] aggregates the "N" language iterations into a collection for an experience that is seamless to the end-user and presents a front-end for querying across languages. JWKTL as an API suffers from the drawback that plural forms of a lemma (word) are not associated with singular senses of the same lemma, so translation efforts are hindered in this way. Using the Wiktionary web-api, plurals are not associated with singular senses, as they are when querying Wikipedia through a browser (i.e.Redirecting). The limitation regarding associating the plural- to the singular- sense applies across most API's, with the Wiktionary browser interface being the exception, in that it indicates if a given word form derives from the singular sense.

Further, there are hindrances such as speed that could occur by using a web-API and the http communication overhead for multiple requests to a web server, such as re-communicating the user-agent for each request. Thus, it may be desirable to have a free, self-hosted resource rather than one which is subject to throttling by system administrators of WikiMedia.

III. A MOTIVATING EXAMPLE

Upon creating a machine-readable dictionary translation tool, a web-interface is needed to compel users to engage with the dictionary; else the application may exist only on the command line. It is unclear the volume of calls to the Wikimedia servers that would result in having an application such as this machine translator in being black-listed by Wikipedia system administrators, but in utilizing a locally queryable database, the application could easily overcome the *Request Limit* stated on API:Etiquette [4].

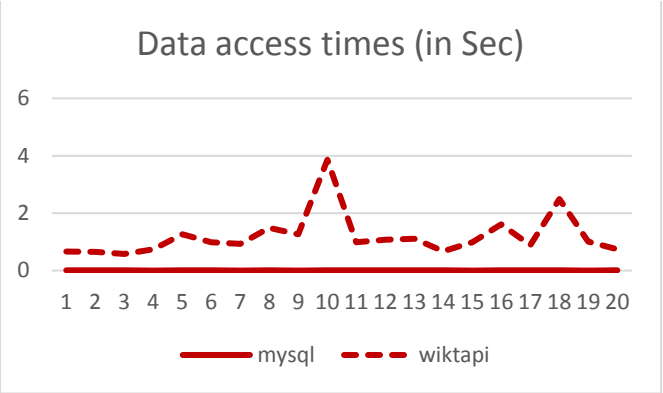

Figure 1: time to fetch data from local mysql database-backed web service vs. same that fetches from Wiktionary.org API

IV. Implementation And Results

A web service request using a locally queryable, indexed database as a backend will be more performant than a request which invokes a libcurl routine and faces a similar lookup routine server-side before returning the result to the user in their browser. A test word was passed to the remote server in an HTTP GET request, to test the performance of the libcurl and Wiktionary API [5] retrieval calls. Creating a locally queryable database of the Wiktionary data was a process requiring more steps but which this study indicated was more performant than a web service that involves a libcurl routine instead. The steps to create the local database were:

1) Download the wiktionary data: To download French, choose the frwiktionary-..-"pages-articles" xml.bz2 file.

2) Set up the database structure in mysql: create a new database and run the create_tables.sql script provided at igrec.ca (see [1]).

3) Create php code that applies a regular expresssion to filter the Wikitext to get a French to English translation: a regular expression that extracts the word sense between "{{trad+|en|" and "}}" will indicate at least one valid translation for a given query provided the word is not pluralized or otherwise inflected, the entry exists in the wiktionary dump, and the translation exists for that entry on its wiktionary page (returned as wikitext) to return as a list in the browser.

The word "malheur" is an example word that may be translated using the data from the 12262015 frwiktionary dump and illustrates additional challenges that are beyond the scope of this study such as what to do for words with multiple translation mappings, found in more than one translation-capable wiktionary dataset. "Malheur" is found in the french or english wiktionary dump datasets like many words. However, it is not expected that parsing will be a satisfactory means for categorizing the nativeness of a word in a given language, anytime in the near future. The word "bonjour" is known to be non-English specifically French, according to wikipedia, but Wiktionary accepts there is an English version of the word.

To compare the local, French to English translator application to one using the Wiktionary API, the above 3-step process was followed and timed using the microtime() function of PHP and the same timing was done against another script identical to the local version except for calls requiring an external http lookup from the Wikitionary API servers. The average run times for 20 successive calls were around .01s vs. 1.20s, with the local application outperforming the remote-calling application consistently (see Fig. 1).

A setback to the functionality of the translation tool is that character encodings have not been accounted for on Windows- works fine on Linux- which represents a significant loss in capability, as any French words with characters outside the ASCII character set will likely not be translateable for Windows installations in the release before this conference.

V. Conclusion

In this work, the successes and setbacks associated with creating a robust, efficient translation web service backed by a collaborative database are explored. We also explore the engineering aspects that had previously been unpublicized. We provide a means for creating a RESTful endpoint for retrieving Wiktionary-sourced French-to-English word translations, when accompanied by the scripts posted to github.com, and performance of this endpoint has been tested using two implementations – one utilizes a local mysql database created from a wiktionary database import for its data and the other relies on parsing a result obtained from a query to the Wiktionary API servers. Notable efforts which provided inspiration to this effort are identified as [5],[6] and [7].

VI. References

[1] Y. Bourque, "Wikparser .3," [Online]. Available: http://www.igrec.ca/programming/wikparser-0-3/. (Accessed 09 04 2016).

[2] Zesch, Torsten, C. Müller and I. Gurevych, "Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary," in *LREC, vol. 8*, 2008.

[3] B. Casses, "Aligning Wiktionary With Natural Language Processing Resources".

[4] "API:Etiquette," [Online]. Available: https://www.mediawiki.org/wiki/API:Etiquette#Request_limit. (Accessed 09 04 2016).

[5] [Online]. Available: https://en.wiktionary.org/w/api.php?action=parse&prop=wikitext&page=test&format=xml. (Accessed 2016).

[6] A. A. Krizhanovsky, "Transformation of Wiktionary entry structure into tables and relations in a relational database schema," arXiv preprint, (2010).

[7] M. Matuschek and I. Gurevych, "Where the journey is headed: Collaboratively constructed multilingual Wiki-based resources," 2011).

[8] "JWKTL Use Cases," [Online]. Available: https://dkpro.github.io/dkpro-jwktl/documentation/use-cases/#multilingual-processing. (Accessed 09 04 2016).