INTRODUCTION

The Aggregate Distributed Dictionary Representation (ADDR) method allows users to measure the presence of specific semantic and conceptual domains in natural language. This method uses distributed representations of words to generate high-dimensional latent representations of specific domains. These representations can then be compared to representations of documents by calculating the similarity between them, yielding what is functionally a loading of documents on the domains of interest (a document-domain loading). These loadings or similarities can then be used as variables in subsequent models. Accordingly, the ADDR method combines the flexibility and power of distributed representations with the precision of theoretically constrained construct measurement.

The addr package provides an easy to implement set of functions that can be used for calculating document-domain loadings. The package is designed to be usable for people with minimal programming experience and, to this end, it contains functions that convert texts and term-dictionaries from a range of formats (LIWC, CSV, and text files) into the requisite format.

FEATURES

- Load documents and term-dictionaries from a range of formats
- Save aggregate distributed representations of documents and term-dictionaries to CSV formatted file for later use
- Calculate document-domain loadings

INSTALL

The addr package is currently under development and is not yet publicly deployed. However, addr can be easily installed using the following steps:

1. Ensure that Python 2.7 and pip is installed on your current system
2. Download and decompress addr.zip
3. Open a terminal window and navigate into the unzipped addr directory
4. Use pip to install the packages required by addr, which are listed in 'requirements.txt', using the following command:

pip install -r requirements.txt

5. In the same terminal session (still in the unzipped addr directory) install addr using the following command:

python setup.py install

Quick-Start

To get started, after downloading the addr package, you can download a pre-trained Word2Vec model that was trained on the Google News Corpus here.

```
>>> import addr
```

Define file paths. These will be used later in function calls.

```
>>> documents_path = "/../../documents.csv"
>>> dictionary_directory = "/../../MFD-Seed-text-format"
>>> model_path = "/../../GoogleNews-vectors-negative300.bin"
>>> dictionary_vector_path = "/../../agg_dic_vectors.tsv"
>>> document_vector_path = "/Users/joe/offline_research/addr-testing/agg_doc_vecs.tsv"
>>> document_loadings_out_path = "/../../document_dictionary_loadings.tsv"
```

Load Word2Vec Model and specify model dimensionality and word index.

```
>>> model, num_features, model_word_set = addr.load_model(model_path)
```

Build Python dict of dictionary terms

```
>>> dic_terms=  addr.terms_from_txt(input_path=dictionary_directory)
```

Make aggregate vector representations of dictionary dimensions

```
>>> agg_dic_vecs = addr.dic_vecs(dic_terms=dic_terms, model=model, num_features=num_feat
```

Write aggregate dictionary vectors to file

```
>>> addr.write_dic_vecs(dic_vecs=agg_dic_vecs, output_path=dictionary_vector_path)
```

Make aggregate vector representations of documents and write these to file

```
>>> addr.doc_vecs_from_csv(input_path=documents_path, output_path=document_vector_path,
                       model=model, num_features=num_features, model_word_set=model_word
                       text_col="text")
```

Calculate similarities between document and dictionary vectors

```
>>> addr.get_loadings(agg_doc_vecs_path=document_vector_path,
                  agg_dic_vecs_path=dictionary_vector_path,
                  out_path="document_loadings_out_path",
                  num_features=num_features)
```