DDR Introduction

DDR contains interfaces and functionality to compute pair-wise document-dictionary similarities for a set of term-based dictionaries. The functions in ddr can also be used to convert pre-processed text and term-dictionary files from a range of formats into aggregate vector representations.

In general, ddr involves several steps:

1. Load a Word2Vec model and determine its dimensionality and vocabulary.
2. Generate distributed dictionary representations.
    a. Load dictionary terms from file
    b. For each term associated with a given dictionary, query its representation from the loaded Word2Vec file and average these together, thereby creating *distributed dictionary representations*.
    c. Write the distributed dictionary representations to file.
3. Generate distributed document representations.
    a. For each document in a corpus, query the representations of the words in the document from the loaded Word2Vec model and average these together, thereby creating a distributed document representation.
4. Calculate document loadings for each construct.
    a. For each distributed document representation generated in 3., calculate its cosine similarity with each distributed dictionary representation generated in 2 and write these to file.

After completion of these steps, the user will have a tab delineated file in where rows correspond to documents and columns correspond to constructs. Each [document_{i}, construct_{j}] cell thus contains the loading of document_{i} on construct construct_{j}.

Note, however, that these all steps do not necessarily to be completed for every application of ddr. Because the workflow involves writing distributed dictionary and distributed document representations to file, these can be used in later analyses. For example, the distributed document representations for a given corpus can be used later in conjunction between a new set of distributed dictionary representations.

Currently, the Google News Word2Vec model is hosted on git here: https://github.com/mmihaltz/word2vec-GoogleNews-vectors This model is excellent

for becoming familiar with Word2Vec and DDR. However, it is also sufficiently reliable for direct research applications.

To use DDR, make sure the following dependencies are installed:

- numpy
- pandas
- cython
- gensim

Example

Below is an example of how to use DDR.

To use the ddr module, it must be imported into the current python environment.

```
>>> import ddr
```

Define file paths

The next step is to define file paths, which will be used later in function calls:

- CSV file where documents are stored in a single column. Note documents can be loaded in other formats (See ddr API for other options)

```
>>> documents_path='/../../documents.csv'
```

- A directory containing one text file for each dictionary dimension of interest. Each text file should contain relevant words delineated by a space with no line breaks.

```
>>> dictionary_directory='/../../MFD-Seed-text-format'
```

- Path to a Word2Vec model

```
>>> model_path='/../../GoogleNews-vectors-negative300.bin'
```

- This path will be used both to write dictionary representations to file and to load them from file when calculating document loadings.

```
>>> dictionary_vector_path='/../../agg_dic_vectors.tsv'
```

- This path is will be used both to write document representations and to read them when calculating document loadings.

```
>>> document_vector_path='/Users/joe/offline_research/ddr-testing/agg_doc_vecs.tsv'
```

- This path will be used to write the document loadings to file.

```
>>> document_loadings_out_path='/../../document_dictionary_loadings.tsv'
```

Load a Word2Vec model

Load Word2Vec Model and specify model dimensionality and word index.

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

Distributed dictionary representations

The next step is to generate distributed dictionary representations.

First, you need to load the terms that represent each dictionary construct. Terms can be loaded from a number of formats, including .txt, LIWC, and csv files. For more information, see the load_terms module documentation in the DDR API.

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

Next, you need to aggregate the representations of each word associated with a construct to create a distributed representation of that construct:

```
>>> agg_dic_vecs=ddr.dic_vecs(dic_terms=dic_terms, model=model, num_features=num_features, m
```

The final step is to write these distributed dictionary representations to file for later use.

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

Distributed document representations

Now, you need to generate distributed document representations. Documents can be read from a delineated file in which documents are contained in the rows of a single column, from a text file in each each row corresponds to a document, or from a director that contains a single text file for each document. Below, we show how you can read documents from CSV format. For information on reading documents from other formats, see get_vecs module documentation in the DDR API.

```
>>> ddr.doc_vecs_from_csv(input_path=documents_path, output_path=document_vector_path,
                  model=model, num_features=num_features, model_word_set=model_word_set,
                  text_col='text')
```

Calculate document loadings

Finally, you need to calculate the loading of each document on each dictionary dimension. This can be accomplished with the following function call:

```
>>> ddr.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)
```