Evaluating Word Embedding

Overview

This module is to evaluate and compare the performance of multiple word embeddings (word vectors generated by algorithms, such as Word2Vec, FastText, GloVe) on a given test data set. The module finds out: how many times there is a representation in a particular embedding for test questions from a test data (X%), and how many times the test questions are solved adequately if the representation for a question exists (Y%).

Built-in word vectors (embeddings) were pre-trained from 4 different algorithms, namely, Word2Vec (Skip-Gram), Word2Vec (CBOW), FastText, GloVe, and included in '`models/`'. Some example test data sets are provided in '`testdata/`'.

To execute the evaluation: `python eval_wordEmbeddings.py <testdata_filename>`

The code was written with the assumption that one and only one argument is given, which is the filename of the test data in form of text file. This file MUST be located in folder '`testdata`' (or path '`testdata/`').

Files in folders

`evalWordEmbeddings/`	
`eval_wordEmbeddings.py`	● Source code written in *Python 3.7*, with dependence on *Numpy*
`README.pdf`	● This README file
`models/`	● Folder for model and parameter related data
`model_lib.txt`	● Summary of models to be evaluated, a text file with lines each line in form of: <model_name> : <model_filename>
`wv_FastText.npz`	● Pretrained word embedding from FastText algorithm, 100 dims
`wv_GloVe.npz`	● Pretrained word embedding from GloVe algorithm, 100 dims
`wv_Word2Vec_CBOW.npz`	● Pretrained word embedding from CBOW algorithm, 100 dims
`wv_Word2Vec_SG.npz`	● Pretrained word embedding from Skip-Gram algorithm, 100 dims
`testdata/`	● Folder for test data sets (now only contains example data sets)
`Test_similarity.txt`	● Data set for word similarity test (from famous WS353)
`Test_analogy.txt`	● Data set for word analogy test (from Google Analogy test set)
`Test_outlier.txt`	● Data set for outlier detection test (self-prepared)
`Test_exist.txt`	● Data set for representation existence test (a piece of online news)

For '`models/`' folder:

All word embeddings were pre-trained using *Gensim* package, with famous 'text8' corpus, and contains ~70,000 word representations of 100 dims, except for the GloVe model, which is 'glove-wiki-gigaword-100' and obtained from *Gensim api.*, and contains 400,000 word representations. For fairness and computational workload consideration, only those appearing in the other three word embeddings are kept (resulting in ~68,000 word representations for GloVe). For all such `wv_xxx.npz` files, there is a 'w' key, containing all represented words, and a 'v' key, containing the vector representations of all words.

The `model_lib.txt` file invites users to indicate the name and `.npz` files of models to be evaluated. That is, users can prepare their own pre-trained word embeddings as `.npz` form (with a 'w' key for words and 'v' key for vectors). The format of text in `model_lib.txt` is:

```
model_name1:parameter_npz_filename1
model_name2:parameter_npz_filename2
... (following the same style above)
```

Note: use colon ':' to separate model_name and its parameter filename. The `model_name`, just for user's convenience to recognize, can be any string (excluding ':'), and the model `parameter_npz_filename` must belong to an existing `.npz` file with above mentioned parameter data. Your model data must be located in '`/models`' for the module to be able to locate.

For '`testdata/`' folder:

There are four example data sets for different testing. Users can follow the examples to prepare the text files for a particular sort of testing on their needs. The test type (benchmark) will be determined by the data set:

(1) Word similarity. Example file: `Test_similarity.txt`. The format of the text file:

```
Line 1: !similarity <a number>
Line 2: word1 word2 <a number>
Line 3: word1 word2 <a number>
... (following the same style above)
```

The beginning string '`!similarity`' in line 1 tells the module to conduct a word similarity test. The following number in line 1 specify the scale of the human-labeled similarity score. For example, `10` means scoring on a scale of 0-10 (i.e., 10 = two words are highly similar, 0 = they are not similar at all).

Each of the following lines provides 2 words and a human-labeled score, separated by a space or '`\t`'. If a model predicted similarity (using 'cosine similarity' method) is close to the human label (within a difference of 0.2 on a unity scale), the word representation is considered 'good'.

The example file is from the famous 'WS353' data set ('WS353' stands for 'word similarity 353 pairs', *Finkelstein, Gabrilovich, et al., "Placing search in context: The concept revisited," 2001*)

(2) Word analogy. Example file: `Test_analogy.txt`. The format of the text file:

```
Line 1: !analogy
Line 2: a a* b b*
Line 3: a a* b b*
... (following the same style above)
```

The beginning string '`!analogy`' in line 1 tells the module to conduct a word analogy test.

Each of the following lines provides 4 words, separated by a space or '`\t`', where the relationship between a and a* is quite similar to that between b and b* (from human's perspective). These data are human labeled. The module will predict b* based on (a, a* and b), by '3CosAdd' method. If the model predicted b* is the same as the human label, the word representation is considered 'good'.

The example data set is a commonly used 'Google Analogy' data set (19544 query questions, *Mikolov, Chen, et al., "Efficient estimation of word representations in vector space," 2013*)

(3) Outlier Detection. Example file: `Test_outlier.txt`. The format of the text file:

```
Line 1: !outlier
Line 2: word1 word2 word3 … wordn <a number>
Line 3: word1 word2 word3 … wordn <a number>
... (following the same style above)
```

The beginning string '`!outlier`' in line 1 tells the module to conduct an outlier detection test.

Each of the following lines provides a few words and a number, separated by a space or '`\t`', where the number indicates the human-labeled 1-base index of the word least 'compatible' with all other words in the group. If the predicted index (based on the 'compactness score', *Camacho-Collados, Navigli, "Find the word that does not belong: A Framework for an Intrinsic Evaluation of Word Vector Representations," 2016*) is the same as the human label, the word representation is considered 'good'.

The example data set is self-prepared by the author, containing 39 word groups covering semantic and syntactic questions.

(4) When you prepare your own test data set, make sure to specify the test type in the first line, and the only recognizable types include '`!similarity`', '`!analogy`', or '`!outlier`'. If a type is specified, the following lines should follow the respective style mentioned above. Note that all lines (except for the first line) starting with a non-English-character (e.g., start with any special character, such as comma, period, colon, slash) are ignored for testing, so users can feel free to give sub-headings, sector dividers, or comments in the data text file by starting a line with a special character.

If, at the beginning of the text file, there is no such string for a recognizable type, the module will treat the data set an English article, and conduct an existence check, i.e., counting how many words in the article have a representation in the models. The `Test_exist.txt` is an example data set for this type of test, and is obtained from a piece of online news.

Requirements

Python >= 3.5.0
Numpy >=1.10.0

Note:

For your reference, the module was developed and tested under Python 3.7.2 and Numpy v1.16.2

The tests were done in Windows 10 Command Window (cmd) and Spyder IPython console. It runs well in 'cmd' for all tests, but in Spyder IPython console, Analogy Test may not run, causing computer stuck, but OK for all other tests (so this may be due to big data workload for the analogy test data set).

Sample running line and results/outputs by using test data sets

(1) Running SIMILARITY test using data from '`testdata/Test_similarity.txt`'

```
In [21]: !python eval_wordEmbeddings.py Test_similarity.txt
------------------------------------------------------------
Loading model parameters...
------------------------------------------------------------
    Word2Vec_SG: word embedding parameters loaded OK!
  Word2Vec_CBOW: word embedding parameters loaded OK!
       FastText: word embedding parameters loaded OK!
          GloVe: word embedding parameters loaded OK!
------------------------------------------------------------
All model parameters have been loaded!
------------------------------------------------------------

------------------------------------------------------------
Performing SIMILARITY test (on "testdata/Test_similarity.txt") ...
------------------------------------------------------------
          model   n_test    n_avail (X%)       n_good (Y%)
------------------------------------------------------------
    Word2Vec_SG      353      333 ( 94.3%)      211 ( 63.4%)
  Word2Vec_CBOW      353      333 ( 94.3%)      145 ( 43.5%)
       FastText      353      333 ( 94.3%)      203 ( 61.0%)
          GloVe      353      333 ( 94.3%)      206 ( 61.9%)
------------------------------------------------------------
The Best model on the given data set is: Word2Vec_SG
As there are representations for X = 94.3% of the testing questions,
and Y = 63.4% good representations if there is a representation.
```

(2) Running ANALOGY test using data from '`testdata/Test_analogy.txt`'

```
In [22]: !python eval_wordEmbeddings.py Test_analogy.txt
------------------------------------------------------------
Loading model parameters...
------------------------------------------------------------
    Word2Vec_SG: word embedding parameters loaded OK!
  Word2Vec_CBOW: word embedding parameters loaded OK!
       FastText: word embedding parameters loaded OK!
          GloVe: word embedding parameters loaded OK!
------------------------------------------------------------
All model parameters have been loaded!
------------------------------------------------------------

------------------------------------------------------------
Performing ANALOGY test (on "testdata/Test_analogy.txt") ...
------------------------------------------------------------
Warning: Analogy Test is computationally expensive.
It may take 3-10 min per model, or a total of 15-30 min for all 4 models

          model   n_test    n_avail (X%)       n_good (Y%)
------------------------------------------------------------
    Word2Vec_SG    19544     9310 ( 47.6%)       336 (  3.6%)
  Word2Vec_CBOW    19544     9310 ( 47.6%)       511 (  5.5%)
       FastText    19544     9310 ( 47.6%)      1579 ( 17.0%)
          GloVe    19544     9310 ( 47.6%)      1712 ( 18.4%)
------------------------------------------------------------
The Best model on the given data set is: GloVe
As there are representations for X = 47.6% of the testing questions,
and Y = 18.4% good representations if there is a representation.
```

(3) Running OUTLIER test using data from '`testdata/Test_outlier.txt`'

```
In [23]: !python eval_wordEmbeddings.py Test_outlier.txt
------------------------------------------------------------
Loading model parameters...
------------------------------------------------------------
    Word2Vec_SG: word embedding parameters loaded OK!
  Word2Vec_CBOW: word embedding parameters loaded OK!
       FastText: word embedding parameters loaded OK!
          GloVe: word embedding parameters loaded OK!
------------------------------------------------------------
All model parameters have been loaded!
------------------------------------------------------------

------------------------------------------------------------
Performing OUTLIER test (on "testdata/Test_outlier.txt") ...
------------------------------------------------------------
          model   n_test    n_avail (X%)       n_good (Y%)
------------------------------------------------------------
    Word2Vec_SG       39       33 ( 84.6%)       23 ( 69.7%)
  Word2Vec_CBOW       39       33 ( 84.6%)       24 ( 72.7%)
       FastText       39       33 ( 84.6%)       23 ( 69.7%)
          GloVe       39       33 ( 84.6%)       23 ( 69.7%)
------------------------------------------------------------
The Best model on the given data set is: Word2Vec_CBOW
As there are representations for X = 84.6% of the testing questions,
and Y = 72.7% good representations if there is a representation.
```

(4) Running with a data set with no specified test type, then the module conducts an EXISTENCE test (data from '`testdata/Test_outlier.txt`')

```
In [24]: !python eval_wordEmbeddings.py Test_exist.txt
------------------------------------------------------------
Loading model parameters...
------------------------------------------------------------
    Word2Vec_SG: word embedding parameters loaded OK!
  Word2Vec_CBOW: word embedding parameters loaded OK!
       FastText: word embedding parameters loaded OK!
          GloVe: word embedding parameters loaded OK!
------------------------------------------------------------
All model parameters have been loaded!
------------------------------------------------------------

------------------------------------------------------------
Performing EXISTENCE test (on "testdata/Test_exist.txt") ...
------------------------------------------------------------
          model   n_test    n_avail (X%)
------------------------------------------------------------
    Word2Vec_SG      315      303 ( 96.2%)
  Word2Vec_CBOW      315      303 ( 96.2%)
       FastText      315      303 ( 96.2%)
          GloVe      315      303 ( 96.2%)
------------------------------------------------------------
The Best model on the given data set is: Word2Vec_SG
As there are representations for X = 96.2% of the testing questions,
and Y = 96.2% good representations if there is a representation.
```