shorttext Documentation

Release 1.6.1

Kwan-Yuet Ho

Dec 21, 2023

CONTENTS

This repository is a collection of algorithms for multi-class classification to short texts using Python. Modules are backward compatible unless otherwise specified. Feel free to give suggestions or report issues through the Issue tab of the Github page. This is a PyPI project. This is an open-source project under the MIT License .

Contents:

CHAPTER

ONE

INTRODUCTION

This package *shorttext* is a Python package that facilitates supervised and unsupervised learning for short text categorization. Due to the sparseness of words and the lack of information carried in the short texts themselves, an intermediate representation of the texts and documents are needed before they are put into any classification algorithm. In this package, it facilitates various types of these representations, including topic modeling and word-embedding algorithms.

The package *shorttext* runs on Python 3.8, 3.9, 3.10, and 3.11.

Characteristics:

- example data provided (including subject keywords and NIH RePORT); (see *Data Preparation*)
- text preprocessing; (see *Text Preprocessing*)
- pre-trained word-embedding support; (see *Word Embedding Models*)
- *gensim* topic models (LDA, LSI, Random Projections) and autoencoder; (see *Supervised Classification with Topics as Features*)
- topic model representation supported for supervised learning using *scikit-learn*; (see *Supervised Classification with Topics as Features*)
- cosine distance classification; (see *Supervised Classification with Topics as Features*, *Word-Embedding Cosine Similarity Classifier*)
- neural network classification (including ConvNet, and C-LSTM); (see *Deep Neural Networks with Word-Embedding*)
- maximum entropy classification; (see *Maximum Entropy (MaxEnt) Classifier*)
- metrics of phrases differences, including soft Jaccard score (using Damerau-Levenshtein distance), and Word Mover's distance (WMD); (see *Metrics*)
- character-level sequence-to-sequence (seq2seq) learning; (see *Character-Based Sequence-to-Sequence (seq2seq) Models*)
- spell correction; (see *Spell Correctors*)
- API for word-embedding algorithm for one-time loading; (see tutorial_wordembedAPI) and
- Sentence encodings and similarities based on BERT (see *Word Embedding Models* and *Metrics*).

Before release 0.7.2, part of the package was implemented using C, and it is interfaced to Python using SWIG (Simplified Wrapper and Interface Generator). Since 1.0.0, these implementations were replaced with Cython.

Author: Kwan Yuet Stephen Ho (LinkedIn, ResearchGate, Twitter)

Home: *Homepage of shorttext*

CHAPTER

TWO

INSTALLATION

2.1 PIP

Package *shorttext* runs in Python 3.6, 3.7, and 3.8. However, for Python>=3.7, the backend of keras cannot be Tensorflow.

To install the package in Linux or OS X, enter the following in the console:

```
pip install -U shorttext
```

It is very possible that you have to do it as root, that you have to add `sudo` in front of the command.

On the other hand, to get the development version on Github, you can install from Github:

```
pip install -U git+https://github.com/stephenhky/PyShortTextCategorization@master
```

By adding `-U` in the command, it automatically installs the required packages. If not, you have to install these packages on your own.

2.2 Backend for Keras

The package keras (version >= 2.0.0) uses either Tensorflow, Theano, or CNTK as the backend, while Theano is usually the default. However, it is highly recommended to use Tensorflow as the backend. Users are advised to install the backend Tensorflow (preferred for Python 2.7, 3.5, and 3.6) or Theano (preferred for Python 3.7) in advance. Refer to *Frequently Asked Questions (FAQ)* for how to switch the backend. It is also desirable if the package Cython has been previously installed.

2.3 Possible Solutions for Installation Failures

Most developers can install *shorttext* with the instructions above. If the installation fails, you may try one (or more) of the following:

1. Installing Python-dev by typing:

```
pip install -U python3-dev
```

2. Installing *gcc* by entering

```
apt-get install libc6
```

2.4 Required Packages

- Numpy (Numerical Python, version >= 1.16.0)
- SciPy (Scientific Python, version >= 1.2.0)
- Scikit-Learn (Machine Learning in Python, version >= 0.23.0)
- keras (Deep Learning Library for Theano and Tensorflow, version >= 2.3.0)
- gensim (Topic Modeling for Humans, version >= 3.8.0)
- Pandas (Python Data Analysis Library, version >= 1.0.0)
- snowballstemmer (Snowball Stemmer, version >= 2.0.0)
- TensorFlow (TensorFlow, version >= 2.0.0)
- Joblib (Joblib: lightweight Python pipelining, version >= 0.14)

Home: *Homepage of shorttext*

CHAPTER THREE

TUTORIAL

After installation, you are ready to start testing the convenience and power of the package.

Before using, type

```
>>> import shorttext
```

You will get the message that *Theano*, *Tensorflow* or *CNTK* backend is used for *keras*. Refer to Keras Backend for information about switching backends.

3.1 Data Preparation

This package deals with short text. While the text data for predictions or classifications are simply *str* or list of *str*, the training data does take a specific format, in terms of *dict*, the Python dictionary (or hash map). The package provides two sets of data as an example.

3.1.1 Example Training Data 1: Subject Keywords

The first example dataset is about the subject keywords, which can be loaded by:

```
>>> trainclassdict = shorttext.data.subjectkeywords()
```

This returns a dictionary, with keys being the label and the values being lists of the subject keywords, as below:

```
{'mathematics': ['linear algebra', 'topology', 'algebra', 'calculus',
  'variational calculus', 'functional field', 'real analysis', 'complex analysis',
  'differential equation', 'statistics', 'statistical optimization', 'probability',
  'stochastic calculus', 'numerical analysis', 'differential geometry'],
 'physics': ['renormalization', 'classical mechanics', 'quantum mechanics',
  'statistical mechanics', 'functional field', 'path integral',
  'quantum field theory', 'electrodynamics', 'condensed matter',
  'particle physics', 'topological solitons', 'astrophysics',
  'spontaneous symmetry breaking', 'atomic molecular and optical physics',
  'quantum chaos'],
 'theology': ['divine providence', 'soteriology', 'anthropology', 'pneumatology',
↪'Christology',
  'Holy Trinity', 'eschatology', 'scripture', 'ecclesiology', 'predestination',
  'divine degree', 'creedal confessionalism', 'scholasticism', 'prayer', 'eucharist']}
```

3.1.2 Example Training Data 2: NIH RePORT

The second example dataset is from NIH RePORT (Research Portfolio Online Reporting Tools). The data can be downloaded from its ExPORTER page. The current data in this package was directly adapted from Thomas Jones' textMineR R package.

Enter:

```
>>> trainclassdict = shorttext.data.nihreports()
```

Upon the installation of the package, the NIH RePORT data are still not installed. But the first time it was ran, it will be downloaded from the Internet.

This will output a similar dictionary with FUNDING_IC (Institutes and Centers in NIH)
as the class labels, and PROJECT_TITLE (title of the funded projects)

as the short texts under the corresponding labels. This dictionary has 512 projects in total, randomly drawn from the original data.

However, there are other configurations:

3.1.3 Example Training Data 3: Inaugural Addresses

This contains all the Inaugural Addresses of all the Presidents of the United States, from George Washington to Barack Obama. Upon the installation of the package, the Inaugural Addresses data are still not installed. But the first time it was ran, it will be downloaded from the Internet.

The addresses are available publicly, and I extracted them from nltk package.

Enter:

```
>>> trainclassdict = shorttext.data.inaugural()
```

3.1.4 User-Provided Training Data

Users can provide their own training data. If it is already in JSON format, it can be loaded easily with standard Python's *json* package, or by calling:

```
>>> trainclassdict = shorttext.data.retrieve_jsondata_as_dict('/path/to/file.json')
```

However, if it is in CSV format, it has to obey the rules:

- there is a heading; and
- there are at least two columns: first the labels, and second the short text under the labels (everything being the second column will be neglected).

An excerpt of this type of data is as follow:

```
subject,content
mathematics,linear algebra
mathematics,topology
mathematics,algebra
...
physics,spontaneous symmetry breaking
physics,atomic molecular and optical physics
```

(continues on next page)

(continued from previous page)

```
physics,quantum chaos
...
theology,divine providence
theology,soteriology
theology,anthropology
```

To load this data file, just enter:

```
>>> trainclassdict = shorttext.data.retrieve_csvdata_as_dict('/path/to/file.csv')
```

Home: *Homepage of shorttext*

3.2 Text Preprocessing

3.2.1 Standard Preprocessor

When the bag-of-words (BOW) model is used to represent the content, it is essential to specify how the text is preprocessed before it is passed to the trainers or the classifiers.

This package provides a standard way of text preprocessing, which goes through the following steps:

- removing special characters,
- removing numerals,
- converting all alphabets to lower cases,
- removing stop words, and
- stemming the words (using Snowball Porter stemmer).

To do this, load the preprocesser generator:

```
>>> from shorttext.utils import standard_text_preprocessor_1
```

Then define the preprocessor, a function, by just calling:

```
>>> preprocessor1 = standard_text_preprocessor_1()
```

It is a function that perform the preprocessing in the steps above:

```
>>> preprocessor1('Maryland Blue Crab')  # output:  'maryland blue crab'
>>> preprocessor1('filing electronic documents and goes home. eat!!!')   # output: 'file␣
↪electron document goe home eat'
```

3.2.2 Customized Text Preprocessor

The standard preprocessor is good for many general natural language processing tasks, but some users may want to define their own preprocessors for their own purposes. This preprocessor is used in topic modeling, and is desired to be *a function that takes a string, and returns a string*.

If the user wants to develop a preprocessor that contains a few steps, he can make it by providing the pipeline, which is a list of functions that input a string and return a string. For example, let's develop a preprocessor that 1) convert it to base form if it is a verb, or keep it original; 2) convert it to upper case; and 3) tag the number of characters after each token.

Load the function that generates the preprocessor function:

```
>>> from shorttext.utils import text_preprocessor
```

Initialize a WordNet lemmatizer using *nltk*:

```
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
```

Define the pipeline. Functions for each of the steps are:

```
>>> step1fcn = lambda s: ' '.join([lemmatizer.lemmatize(s1) for s1 in s.split(' ')])
>>> step2fcn = lambda s: s.upper()
>>> step3fcn = lambda s: ' '.join([s1+'-'+str(len(s1)) for s1 in s.split(' ')])
```

Then the pipeline is:

```
>>> pipeline = [step1fcn, step2fcn, step3fcn]
```

The preprocessor function can be generated with the defined pipeline:

```
>>> preprocessor2 = text_preprocessor(pipeline)
```

The function *preprocessor2* is a function that input a string and returns a string. Some examples are:

```
>>> preprocessor2('Maryland blue crab in Annapolis')  # output: 'MARYLAND-8 BLUE-4 CRAB-4␣
↪IN-2 ANNAPOLIS-9'
>>> preprocessor2('generative adversarial networks')  # output: 'GENERATIVE-10␣
↪ADVERSARIAL-11 NETWORK-7'
```

3.2.3 Tokenization

Users are free to choose any tokenizer they wish. In *shorttext*, the tokenizer is simply the space delimiter, and can be called:

```
>>> shorttext.utils.tokenize('Maryland blue crab')   # output: ['Maryland', 'blue', 'crab']
```

3.2.4 Reference

Christopher Manning, Hinrich Schuetze, *Foundations of Statistical Natural Language Processing* (Cambridge, MA: MIT Press, 1999). [MIT Press]

"R or Python on Text Mining," *Everything About Data Analytics*, WordPress (2015). [WordPress]

Home: *Homepage of shorttext*

3.3 Document-Term Matrix

3.3.1 Preparing for the Corpus

We can create and handle document-term matrix (DTM) with *shorttext*. Use the dataset of Presidents' Inaugural Addresses as an example.

```
>>> import shorttext
>>> usprez = shorttext.data.inaugural()
```

We have to make each presidents' address to be one document to achieve our purpose. Enter this:

```
>>> docids = sorted(usprez.keys())
>>> usprez = [' '.join(usprez[docid]) for docid in docids]
```

Now the variable *usprez* is a list of 56 Inaugural Addresses from George Washington (1789) to Barack Obama (2009), with the IDs stored in *docids*. We apply the standard text preprocessor and produce a list of lists (of tokens) (or a corpus in *gensim*):

```
>>> preprocess = shorttext.utils.standard_text_preprocessor_1()
>>> corpus = [preprocess(address).split(' ') for address in usprez]
```

Then now the variable *corpus* is a list of lists of tokens. For example,

```
>>> corpus[0]     # shows all the preprocessed tokens of the first Presidential␣
↪Inaugural Addresses
```

3.3.2 Using Class *DocumentTermMatrix*

With the corpus ready in this form, we can create a *DocumentTermMatrix* class for DTM by:

```
>>> usprez_dtm = shorttext.utils.DocumentTermMatrix(corpus, docids=docids)
```

One can get the document frequency of any token (the number of documents that the given token is in) by:

```
>>> usprez_dtm.get_doc_frequency('peopl')  # gives 54, the document frequency of the␣
↪token "peopl"
```

or the total term frequencies (the total number of occurrences of the given tokens in all documents) by:

```
>>> usprez_dtm.get_total_termfreq('justic')   # gives 134.0, the total term frequency of␣
↪the token "justic"
```

or the term frequency for a token in a given document by:

```
>>> usprez_dtm.get_termfreq('2009-Obama', 'chang')    # gives 2.0
```

We can also query the number of occurrences of a particular word of all documents, stored in a dictionary, by:

```
>>> usprez_dtm.get_token_occurences('god')
```

Of course, we can always reweigh the counts above (except document frequency) by imposing tf-idf while creating the instance of the class by enforcing *tfidf* to be *True*:

```
>>> usprez_dtm = shorttext.utils.DocumentTermMatrix(corpus, docids=docids, tfidf=True)
```

To save the class, enter:

```
>>> usprez_dtm.save_compact_model('/path/to/whatever.bin')
```

To load this class later, enter:

```
>>> usprez_dtm2 = shorttext.utils.load_DocumentTermMatrix('/path/to/whatever.bin')
```

3.3.3 Reference

Christopher Manning, Hinrich Schuetze, *Foundations of Statistical Natural Language Processing* (Cambridge, MA: MIT Press, 1999). [MIT Press]

"Document-Term Matrix: Text Mining in R and Python," *Everything About Data Analytics*, WordPress (2018). [WordPress]

Home: *Homepage of shorttext*

3.4 Character to One-Hot Vector

Since version 0.6.1, the package *shorttext* deals with character-based model. A first important component of character-based model is to convert every character to a one-hot vector. We provide a class `shorttext.generators.SentenceToCharVecEncoder` to deal with this. Thi class incorporates the *OneHotEncoder* in *scikit-learn* and *Dictionary* in *gensim*.

To use this, import the packages first:

```
>>> import numpy as np
>>> import shorttext
```

Then we incorporate a text file as the source of all characters to be coded. In this case, we choose the file *big.txt* in Peter Norvig's websites:

```
>>> from urllib.request import urlopen
>>> textfile = urlopen('http://norvig.com/big.txt', 'r')
```

Then instantiate the class using the function `shorttext.generators.initSentenceToCharVecEncoder()`:

```
>>> chartovec_encoder = shorttext.generators.initSentenceToCharVecEncoder(textfile)
```

Now, the object *chartovec_encoder* is an instance of `shorttext.generators.SentenceToCharVecEncoder` . The default signal character is *n*, which is also encoded, and can be checked by looking at the field:

```
>>> chartovec_encoder.signalchar
```

We can convert a sentence into a bunch of one-hot vectors in terms of a matrix. For example,

```
>>> chartovec_encoder.encode_sentence('Maryland blue crab!', 100)
<1x93 sparse matrix of type '<type 'numpy.float64'>'
        with 1 stored elements in Compressed Sparse Column format>
```

This outputs a sparse matrix. Depending on your needs, you can add signal character to the beginning or the end of the sentence in the output matrix by:

```
>>> chartovec_encoder.encode_sentence('Maryland blue crab!', 100, startsig=True,␣
↪endsig=False)
>>> chartovec_encoder.encode_sentence('Maryland blue crab!', 100, startsig=False,␣
↪endsig=True)
```

We can also convert a list of sentences by

```
>>> chartovec_encoder.encode_sentences(sentences, 100, startsig=False, endsig=True,␣
↪sparse=False)
```

You can decide whether or not to output a sparse matrix by specifiying the parameter *sparse*.

3.4.1 Reference

Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (Sebastopol, CA: O'Reilly Media, 2017). [O'Reilly]

Home: *Homepage of shorttext*

3.5 Supervised Classification with Topics as Features

3.5.1 Topic Vectors as Intermediate Feature Vectors

To perform classification using bag-of-words (BOW) model as features, *nltk* and *gensim* offered good framework. But the feature vectors of short text represented by BOW can be very sparse. And the relationships between words with similar meanings are ignored as well. One of the way to tackle this is to use topic modeling, i.e. representing the words in a topic vector. This package provides the following ways to model the topics:

- LDA (Latent Dirichlet Allocation)
- LSI (Latent Semantic Indexing)
- RP (Random Projections)
- Autoencoder

With the topic representations, users can use any supervised learning algorithm provided by *scikit-learn* to perform the classification.

3.5.2 Topic Models in *gensim*: LDA, LSI, and Random Projections

This package supports three algorithms provided by *gensim*, namely, LDA, LSI, and Random Projections, to do the topic modeling.

```
>>> import shorttext
```

First, load a set of training data (all NIH data in this example):

```
>>> trainclassdict = shorttext.data.nihreports(sample_size=None)
```

Initialize an instance of topic modeler, and use LDA as an example:

```
>>> topicmodeler = shorttext.generators.LDAModeler()
```

For other algorithms, user can use `LSIModeler` for LSI or `RPModeler` for RP. Everything else is the same. To train with 128 topics, enter:

```
>>> topicmodeler.train(trainclassdict, 128)
```

After the training is done, the user can retrieve the topic vector representation with the trained model. For example,

```
>>> topicmodeler.retrieve_topicvec('stem cell research')
```

```
>>> topicmodeler.retrieve_topicvec('bioinformatics')
```

By default, the vectors are normalized. Another way to retrieve the topic vector representation is as follow:

```
>>> topicmodeler['stem cell research']
```

```
>>> topicmodeler['bioinformatics']
```

In the training and the retrieval above, the same preprocessing process is applied. Users can provide their own preprocessor while initiating the topic modeler.

Users can save the trained model by calling:

```
>>> topicmodeler.save_compact_model('/path/to/nihlda128.bin')
```

And the topic model can be retrieved by calling:

```
>>> topicmodeler2 = shorttext.generators.load_gensimtopicmodel('/path/to/nihlda128.bin')
```

While initialize the instance of the topic modeler, the user can also specify whether to weigh the terms using tf-idf (term frequency - inverse document frequency). The default is to weigh. To not weigh, initialize it as

```
>>> topicmodeler3 = shorttext.generators.GensimTopicModeler(toweigh=False)
```

Appendix: Model I/O in Previous Versions

For previous versions of *shorttext*, the trained models are saved by calling:

```
>>> topicmodeler.savemodel('/path/to/nihlda128')
```

However, we discourage users using this anymore, because the model I/O for various models in gensim have been different. It produces errors.

All of them have to be present in order to be loaded. Note that the preprocessor is not saved. To load the model, enter:

```
>>> topicmodeler2 = shorttext.classifiers.load_gensimtopicmodel('/path/to/nihlda128',␣
↪compact=False)
```

3.5.3 AutoEncoder

Note: Previous version (<=0.2.1) of this autoencoder has a serious bug. Current version is incompatible with the autoencoder of version <=0.2.1 .

Another way to find a new topic vector representation is to use the autoencoder, a neural network model which compresses a vector representation into another one of a shorter (or longer, rarely though) representation, by minimizing the difference between the input layer and the decoding layer. For faster demonstration, use the subject keywords as the example dataset:

```
>>> subdict = shorttext.data.subjectkeywords()
```

To train such a model, we perform in a similar way with the LDA model (or LSI and random projections above):

```
>>> autoencoder = shorttext.generators.AutoencodingTopicModeler()
>>> autoencoder.train(subdict, 8)
```

After the training is done, the user can retrieve the encoded vector representation with the trained autoencoder model. For example,

```
>>> autoencoder.retrieve_topicvec('linear algebra')
```

```
>>> autoencoder.retrieve_topicvec('path integral')
```

By default, the vectors are normalized. Another way to retrieve the topic vector representation is as follow:

```
>>> autoencoder['linear algebra']
```

```
>>> autoencoder['path integral']
```

In the training and the retrieval above, the same preprocessing process is applied. Users can provide their own preprocessor while initiating the topic modeler.

Users can save the trained models, by calling:

```
>>> autoencoder.save_compact_model('/path/to/sub_autoencoder8.bin')
```

And the model can be retrieved by calling:

```
>>> autoencoder2 = shorttext.generators.load_autoencoder_topicmodel('/path/to/sub_
↪autoencoder8.bin')
```

Like other topic models, while initialize the instance of the topic modeler, the user can also specify whether to weigh the terms using tf-idf (term frequency - inverse document frequency). The default is to weigh. To not weigh, initialize it as:

```
>>> autoencoder3 = shorttext.generators.AutoencodingTopicModeler(toweigh=False)
```

Appendix: Unzipping Model I/O

For previous versions of *shorttext*, the trained models are saved by calling:

```
>>> autoencoder.savemodel('/path/to/sub_autoencoder8')
```

The following files are produced for the autoencoder:

```
/path/to/sub_autoencoder.json
/path/to/sub_autoencoder.gensimdict
/path/to/sub_autoencoder_encoder.json
/path/to/sub_autoencoder_encoder.h5
/path/to/sub_autoencoder_classtopicvecs.pkl
```

If specifying *save_complete_autoencoder=True*, then four more files are found:

```
/path/to/sub_autoencoder_decoder.json
/path/to/sub_autoencoder_decoder.h5
/path/to/sub_autoencoder_autoencoder.json
/path/to/sub_autoencoder_autoencoder.h5
```

Users can load the same model later by entering:

```
>>> autoencoder2 = shorttext.classifiers.load_autoencoder_topic('/path/to/sub_
↪autoencoder8', compact=False)
```

3.5.4 Abstract Latent Topic Modeling Class

Both `shorttext.generators.GensimTopicModeler` and `shorttext.generators.AutoencodingTopicModeler` extends `shorttext.generators.bow.LatentTopicModeling.LatentTopicModeler`, an abstract class virtually. If user wants to develop its own topic model that extends this, he has to define the methods *train*, *retrieve_topic_vec*, *loadmodel*, and *savemodel*.

Appendix: Namespaces for Topic Modeler in Previous Versions

All generative topic modeling algorithms were placed under the package *shorttext.classifiers* for version <=0.3.4. In current version (>= 0.3.5), however, all generative models will be moved to *shorttext.generators*, while any classifiers making use of these topic models are still kept under *shorttext.classifiers*. A list include:

```
shorttext.classifiers.GensimTopicModeler  ->  shorttext.generators.GensimTopicModeler
shorttext.classifiers.LDAModeler  ->  shorttext.generators.LDAModeler
shorttext.classifiers.LSIModeler  ->  shorttext.generators.LSIModeler
shorttext.classifiers.RPModeler  ->  shorttext.generators.RPModeler
shorttext.classifiers.AutoencodingTopicModeler  ->  shorttext.generators.
↪AutoencodingTopicModeler
```

(continues on next page)

(continued from previous page)

```
shorttext.classifiers.load_gensimtopicmodel  ->  shorttext.generators.load_
↪gensimtopicmodel
shorttext.classifiers.load_autoencoder_topic  ->  shorttext.generators.load_autoencoder_
↪topicmodel
```

Before release 0.5.6, for backward compatibility, developers can still call the topic models as if there were no such changes, although they are advised to make this change. However, *effective release 0.5.7, this backward compatibility is no longer available.*

3.5.5 Classification Using Cosine Similarity

The topic modelers are trained to represent the short text in terms of a topic vector, effectively the feature vector. However, to perform supervised classification, there needs a classification algorithm. The first one is to calculate the cosine similarities between topic vectors of the given short text with those of the texts in all class labels.

If there is already a trained topic modeler, whether it is `shorttext.generators.GensimTopicModeler` or `shorttext.generators.AutoencodingTopicModeler`, a classifier based on cosine similarities can be initiated immediately without training. Taking the LDA example above, such classifier can be initiated as follow:

```
>>> cos_classifier = shorttext.classifiers.
↪TopicVectorCosineDistanceClassifier(topicmodeler)
```

Or if the user already saved the topic modeler, one can initiate the same classifier by loading the topic modeler:

```
>>> cos_classifier = shorttext.classifiers.load_gensimtopicvec_cosineClassifier('/path/
↪to/nihlda128.bin')
```

To perform prediction, enter:

```
>>> cos_classifier.score('stem cell research')
```

which outputs a dictionary with labels and the corresponding scores.

The same thing for autoencoder, but the classifier based on autoencoder can be loaded by another function:

```
>>> cos_classifier = shorttext.classifiers.load_autoencoder_cosineClassifier('/path/to/
↪sub_autoencoder8.bin')
```

3.5.6 Classification Using Scikit-Learn Classifiers

The topic modeler can be used to generate features used for other machine learning algorithms. We can take any supervised learning algorithms in *scikit-learn* here. We use Gaussian naive Bayes as an example. For faster demonstration, use the subject keywords as the example dataset.

```
>>> subtopicmodeler = shorttext.generators.LDAModeler()
>>> subtopicmodeler.train(subdict, 8)
```

We first import the class:

```
>>> from sklearn.naive_bayes import GaussianNB
```

And we train the classifier:

```
>>> classifier = shorttext.classifiers.TopicVectorSkLearnClassifier(subtopicmodeler,␣
↪GaussianNB())
>>> classifier.train(subdict)
```

Predictions can be performed like the following example:

```
>>> classifier.score('functional integral')
```

which outputs a dictionary with labels and the corresponding scores.

You can save the model by:

```
>>> classifier.save_compact_model('/path/to/sublda8nb.bin')
```

where the argument specifies the prefix of the path of the model files, including the topic models, and the scikit-learn model files. The classifier can be loaded by calling:

```
>>> classifier2 = shorttext.classifiers.load_gensim_topicvec_sklearnclassifier('/path/to/
↪sublda8nb.bin')
```

The topic modeler here can also be an autoencoder, by putting *subtopicmodeler* as the autoencoder will still do the work. However, to load the saved classifier with an autoencoder model, do

```
>>> classifier2 = shorttext.classifiers.load_autoencoder_topic_sklearnclassifier('/path/
↪to/filename.bin')
```

Compact model files saved by *TopicVectorSkLearnClassifier* in *shorttext* >= 1.0.0 cannot be read by earlier version of *shorttext*; vice versa is not true though: old compact model files can be read in.

3.5.7 Notes about Text Preprocessing

The topic models are based on bag-of-words model, and text preprocessing is very important. However, the text preprocessing step cannot be serialized. The users should keep track of the text preprocessing step on their own. Unless it is necessary, use the standard preprocessing.

See more: *Text Preprocessing* .

3.5.8 Reference

David M. Blei, "Probabilistic Topic Models," *Communications of the ACM* 55(4): 77-84 (2012).

Francois Chollet, "Building Autoencoders in Keras," *The Keras Blog*. [Keras]

Xuan Hieu Phan, Cam-Tu Nguyen, Dieu-Thu Le, Minh Le Nguyen, Susumu Horiguchi, Quang-Thuy Ha, "A Hidden Topic-Based Framework toward Building Applications with Short Web Documents," *IEEE Trans. Knowl. Data Eng.* 23(7): 961-976 (2011).

Xuan Hieu Phan, Le-Minh Nguyen, Susumu Horiguchi, "Learning to Classify Short and Sparse Text & Web withHidden Topics from Large-scale Data Collections," WWW '08 Proceedings of the 17th international conference on World Wide Web. (2008) [ACL]

Home: *Homepage of shorttext*

3.6 Word Embedding Models

3.6.1 Word2Vec

The most commonly used word-embedding model is Word2Vec. Its model can be downloaded from their page. To load the model, call:

```
>>> import shorttext
>>> wvmodel = shorttext.utils.load_word2vec_model('/path/to/GoogleNews-vectors-
↪negative300.bin.gz')
```

It is a binary file, and the default is set to be *binary=True*.

It is equivalent to calling,

```
>>> import gensim
>>> wvmodel = gensim.models.KeyedVectors.load_word2vec_format('/path/to/GoogleNews-
↪vectors-negative300.bin.gz', binary=True)
```

Word2Vec is a neural network model that embeds words into semantic vectors that carry semantic meaning. It is easy to extract the vector of a word, like for the word 'coffee':

```
>>> wvmodel['coffee']   # an ndarray for the word will be output
```

One can find the most similar words to 'coffee' according to this model:

```
>>> wvmodel.most_similar('coffee')
```

which outputs:

```
[(u'coffees', 0.721267819404602),
 (u'gourmet_coffee', 0.7057087421417236),
 (u'Coffee', 0.6900454759597778),
 (u'o_joe', 0.6891065835952759),
 (u'Starbucks_coffee', 0.6874972581863403),
 (u'coffee_beans', 0.6749703884124756),
 (u'latt\xe9', 0.664122462272644),
 (u'cappuccino', 0.662549614906311),
 (u'brewed_coffee', 0.6621608138084412),
 (u'espresso', 0.6616827249526978)]
```

Or if you want to find the cosine similarity between 'coffee' and 'tea', enter:

```
>>> wvmodel.similarity('coffee', 'tea')   # outputs: 0.56352921707810621
```

Semantic meaning can be reflected by their differences. For example, we can vaguely say *Francis* - *Paris* = *Taiwan* - *Taipei*, or *man* - *actor* = *woman* - *actress*. Define first the cosine similarity for readability:

```
>>> from scipy.spatial.distance import cosine
>>> similarity = lambda u, v: 1-cosine(u, v)
```

Then

```
>>> similarity(wvmodel['France'] + wvmodel['Taipei'] - wvmodel['Taiwan'], wvmodel['Paris
↪'])  # outputs: 0.70574580801216202
>>> similarity(wvmodel['woman'] + wvmodel['actor'] - wvmodel['man'], wvmodel['actress'])␣
↪ # outputs: 0.876354245612604
```

3.6.2 GloVe

Stanford NLP Group developed a similar word-embedding algorithm, with a good theory explaining how it works. It is extremely similar to Word2Vec.

One can convert a text-format GloVe model into a text-format Word2Vec model. More information can be found in the documentation of *gensim*: Converting GloVe to Word2Vec

3.6.3 FastText

FastText is a similar word-embedding model from Facebook. You can download pre-trained models here:

Pre-trained word vectors

To load a pre-trained FastText model, run:

```
>>> import shorttext
>>> ftmodel = shorttext.utils.load_fasttext_model('/path/to/model.bin')
```

And it is used exactly the same way as Word2Vec.

3.6.4 Poincaré Embeddings

Poincaré embeddings is a new embedding that learns both semantic similarity and hierarchical structures. To load a pre-trained model, run:

```
>>> import shorttext
>>> pemodel = shorttext.utils.load_poincare_model('/path/to/model.txt')
```

For preloaded word-embedding models, please refer to *Word Embedding Models*.

3.6.5 BERT

BERT (Bidirectional Transformers for Language Understanding) is a transformer-based language model. This package supports tokens and sentence embeddings using pre-trained language models, supported by the package written by HuggingFace. In *shorttext*, to run:

```
>>> from shorttext.utils import WrappedBERTEncoder
>>> encoder = WrappedBERTEncoder()   # the default model and tokenizer are loaded
>>> sentences_embedding, tokens_embedding, tokens = encoder.encode_sentences(['The car␣
↪should turn right.', 'The answer is right.'])
```

The third line returns the embeddings of all sentences, embeddings of all tokens in each sentence, and the tokens (with *CLS* and *SEP*) included. Unlike previous embeddings, token embeddings depend on the context; in the above example, the embeddings of the two “right“‘s are different as they have different meanings.

The default BERT models and tokenizers are *bert-base_uncase*. If you want to use others, refer to HuggingFace's model list .

3.6.6 Other Functions

3.6.7 Links

- Word2Vec
- GloVe
- FastText
- BERT
- HuggingFace

3.6.8 Reference

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv:1810.04805 (2018). [arXiv]

Jayant Jain, "Implementing Poincaré Embeddings," RaRe Technologies (2017). [RaRe]

Jeffrey Pennington, Richard Socher, Christopher D. Manning, "GloVe: Global Vectors for Word Representation," *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532-1543 (2014). [PDF]

Maximilian Nickel, Douwe Kiela, "Poincaré Embeddings for Learning Hierarchical Representations," arXiv:1705.08039 (2017). [arXiv]

Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, "Enriching Word Vectors with Subword Information," arXiv:1607.04606 (2016). [arXiv]

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, "Efficient Estimation of Word Representations in Vector Space," *ICLR* 2013 (2013). [arXiv]

Radim Řehůřek, "Making sense of word2vec," RaRe Technologies (2014). [RaRe]

"Probabilistic Theory of Word Embeddings: GloVe," *Everything About Data Analytics*, WordPress (2016). [WordPress]

"Toying with Word2Vec," *Everything About Data Analytics*, WordPress (2015). [WordPress]

"Word-Embedding Algorithms," *Everything About Data Analytics*, WordPress (2016). [WordPress]

Home: *Homepage of shorttext*

3.7 Word-Embedding Cosine Similarity Classifier

3.7.1 Sum of Embedded Vectors

Given a pre-trained word-embedding models like Word2Vec, a classifier based on cosine similarities can be built, which is `shorttext.classifiers.SumEmbeddedVecClassifier`. In training the data, the embedded vectors in every word in that class are averaged. The score for a given text to each class is the cosine similarity between the averaged vector of the given text and the precalculated vector of that class.

A pre-trained Google Word2Vec model can be downloaded here.

See: *Word Embedding Models* .

Import the package:

```
>>> import shorttext
```

To load the Word2Vec model,

```
>>> from shorttext.utils import load_word2vec_model
>>> wvmodel = load_word2vec_model('/path/to/GoogleNews-vectors-negative300.bin.gz')
```

Then we load a set of data:

```
>>> nihtraindata = shorttext.data.nihreports(sample_size=None)
```

Then initialize the classifier:

```
>>> classifier = shorttext.classifiers.SumEmbeddedVecClassifier(wvmodel)   # for Google␣
↪model, the vector size is 300 (default: 100)
>>> classifier.train(nihtraindata)
```

This classifier takes relatively little time to train compared with others in this package. Then we can perform classification:

```
>>> classifier.score('bioinformatics')
```

Or the result can be sorted and only the five top-scored results are displayed:

```
>>> sorted(classifier.score('stem cell research').items(), key=lambda item: item[1],␣
↪reverse=True)[:5]
[('NIGMS', 0.44962596182682935),
 ('NIAID', 0.4494126990050461),
 ('NINDS', 0.43435236806719524),
 ('NIDCR', 0.43042338197002483),
 ('NHGRI', 0.42878346869968731)]
>>> sorted(classifier.score('bioinformatics').items(), key=lambda item: item[1],␣
↪reverse=True)[:5]
[('NHGRI', 0.54200061864847038),
 ('NCATS', 0.49097267547279988),
 ('NIGMS', 0.47818129591411118),
 ('CIT', 0.46874987052158501),
 ('NLM', 0.46869259072562974)]
>>> sorted(classifier.score('cancer immunotherapy').items(), key=lambda item: item[1],␣
↪reverse=True)[:5]
[('NCI', 0.53734097785976076),
 ('NIAID', 0.50616582142027433),
 ('NIDCR', 0.48596330887674788),
 ('NIDDK', 0.46875755765903215),
 ('NCCAM', 0.4642233792198418)]
```

The trained model can be saved:

```
>>> classifier.save_compact_model('/path/to/sumvec_nihdata_model.bin')
```

And with the same pre-trained Word2Vec model, this classifier can be loaded:

```
>>> classifier2 = shorttext.classifiers.load_sumword2vec_classifier(wvmodel, '/path/to/
↪sumvec_nihdata_model.bin')
```

Appendix: Model I/O in Previous Versions

In previous versions of *shorttext*, `shorttext.classifiers.SumEmbeddedVecClassifier` has a *savemodel* method, which runs as follow:

```
>>> classifier.savemodel('/path/to/nihdata')
```

This produces the following file for this model:

```
/path/to/nihdata_embedvecdict.pkl
```

It can be loaded by:

```
>>> classifier2 = shorttext.classifiers.load_sumword2vec_classifier(wvmodel, '/path/to/
↪nihdata', compact=False)
```

3.7.2 Reference

Michael Czerny, "Modern Methods for Sentiment Analysis," *District Data Labs (2015). [DistrictDataLabs]

Home: *Homepage of shorttext*

3.8 Deep Neural Networks with Word-Embedding

3.8.1 Wrapper for Neural Networks for Word-Embedding Vectors

In this package, there is a class that serves a wrapper for various neural network algorithms for supervised short text categorization: `shorttext.classifiers.VarNNEmbeddedVecClassifier`. Each class label has a few short sentences, where each token is converted to an embedded vector, given by a pre-trained word-embedding model (e.g., Google Word2Vec model). The sentences are represented by a matrix, or rank-2 array. The type of neural network has to be passed when training, and it has to be of type `keras.models.Sequential`. The number of outputs of the models has to match the number of class labels in the training data. To perform prediction, the input short sentences is converted to a unit vector in the same way. The score is calculated according to the trained neural network model.

Some of the neural networks can be found within the module **:module:`shorttext.classifiers.embed.nnlib.frameworks`** and they are good for short text or document classification. Of course, users can supply their own neural networks, written in *keras*.

A pre-trained Google Word2Vec model can be downloaded here, and a pre-trained Facebook FastText model can be downloaded here.

See: *Word Embedding Models* .

Import the package:

```
>>> import shorttext
```

To load the Word2Vec model,

```
>>> wvmodel = shorttext.utils.load_word2vec_model('/path/to/GoogleNews-vectors-
↪negative300.bin.gz')
```

Then load the training data

```
>>> trainclassdict = shorttext.data.subjectkeywords()
```

Then we choose a neural network. We choose ConvNet:

```
>>> kmodel = shorttext.classifiers.frameworks.CNNWordEmbed(len(trainclassdict.keys()),␣
↪vecsize=300)
```

Initialize the classifier:

```
>>> classifier = shorttext.classifiers.VarNNEmbeddedVecClassifier(wvmodel)
```

Then train the classifier:

```
>>> classifier.train(trainclassdict, kmodel)
Epoch 1/10
45/45 [==============================] - 0s - loss: 1.0578
Epoch 2/10
45/45 [==============================] - 0s - loss: 0.5536
Epoch 3/10
45/45 [==============================] - 0s - loss: 0.3437
Epoch 4/10
45/45 [==============================] - 0s - loss: 0.2282
Epoch 5/10
45/45 [==============================] - 0s - loss: 0.1658
Epoch 6/10
45/45 [==============================] - 0s - loss: 0.1273
Epoch 7/10
45/45 [==============================] - 0s - loss: 0.1052
Epoch 8/10
45/45 [==============================] - 0s - loss: 0.0961
Epoch 9/10
45/45 [==============================] - 0s - loss: 0.0839
Epoch 10/10
45/45 [==============================] - 0s - loss: 0.0743
```

Then the model is ready for classification, like:

```
>>> classifier.score('artificial intelligence')
{'mathematics': 0.57749695, 'physics': 0.33749574, 'theology': 0.085007325}
```

The trained model can be saved:

```
>>> classifier.save_compact_model('/path/to/nnlibvec_convnet_subdata.bin')
```

To load it, enter:

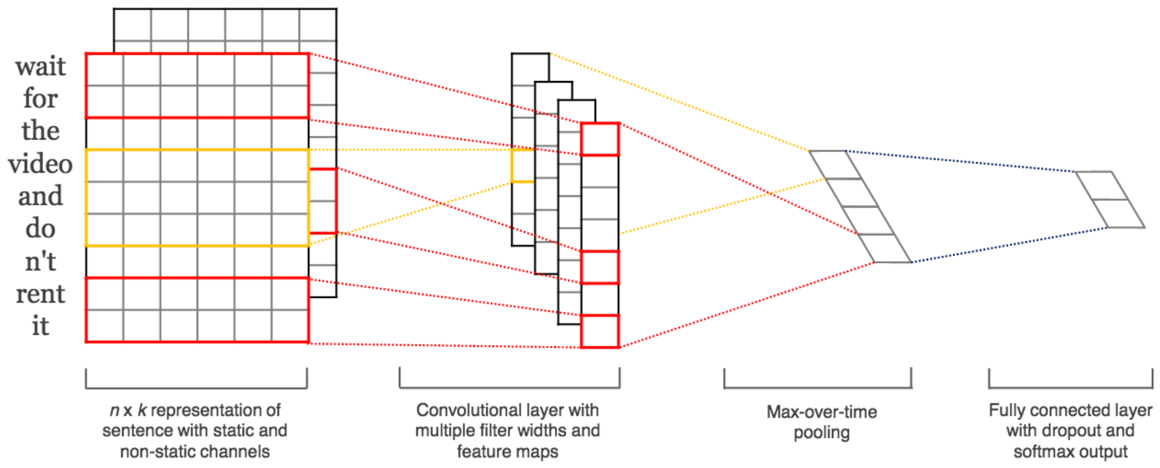
```
>>> classifier2 = shorttext.classifiers.load_varnnlibvec_classifier(wvmodel, '/path/to/
↪nnlibvec_convnet_subdata.bin')
```

3.8.2 Provided Neural Networks

There are three neural networks available in this package for the use in `shorttext.classifiers.VarNNEmbeddedVecClassifier`, and they are available in the module *shorttext.classifiers.frameworks*.

ConvNet (Convolutional Neural Network)

This neural network for supervised learning is using convolutional neural network (ConvNet), as demonstrated in Kim's paper.


wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

The function in the frameworks returns a `keras.models.Sequential` or `keras.models.Model`. Its input parameters are:

The parameter *maxlen* defines the maximum length of the sentences. If the sentence has less than *maxlen* words, then the empty words will be filled with zero vectors.

```
>>> kmodel = fr.CNNWordEmbed(len(trainclassdict.keys()), vecsize=wvmodel.vector_size)
```

Double ConvNet

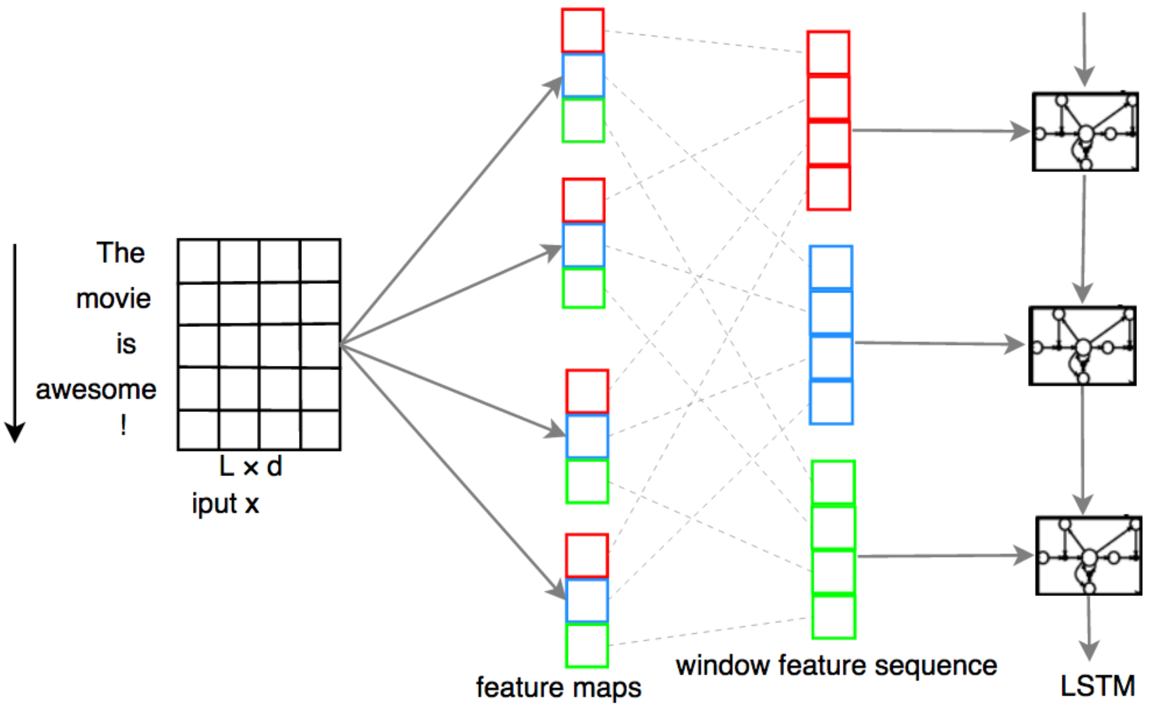
This neural network is nothing more than two ConvNet layers. The function in the frameworks returns a `keras.models.Sequential` or `keras.models.Model`. Its input parameters are:

The parameter *maxlen* defines the maximum length of the sentences. If the sentence has less than *maxlen* words, then the empty words will be filled with zero vectors.

```
>>> kmodel = fr.DoubleCNNWordEmbed(len(trainclassdict.keys()), vecsize=wvmodel.vector_
↪size)
```

C-LSTM (Convolutional Long Short-Term Memory)

This neural network for supervised learning is using C-LSTM, according to the paper written by Zhou *et. al.* It is a neural network with ConvNet as the first layer, and then followed by LSTM (long short-term memory), a type of recurrent neural network (RNN).



The function in the frameworks returns a `keras.models.Sequential` or `keras.models.Model`.

The parameter *maxlen* defines the maximum length of the sentences. If the sentence has less than *maxlen* words, then the empty words will be filled with zero vectors.

```
>>> kmodel = fr.CLSTMWordEmbed(len(trainclassdict.keys()), vecsize=wvmodel.vector_size)
```

User-Defined Neural Network

Users can define their own neural network for use in the classifier wrapped by `shorttext.classifiers.VarNNEmbeddedVecClassifier` as long as the following criteria are met:

- the input matrix is `numpy.ndarray`, and of shape *(maxlen, vecsize)*, where

maxlen is the maximum length of the sentence, and *vecsize* is the number of dimensions of the embedded vectors. The output is a one-dimensional array, of size equal to the number of classes provided by the training data. The order of the class labels is assumed to be the same as the order of the given training data (stored as a Python dictionary).

Putting Word2Vec Model As an Input Keras Layer (Deprecated)

This functionality is removed since release 0.5.11, due to the following reasons:

- *keras* changed its code that produces this bug;
- the layer is consuming memory;
- only Word2Vec is supported; and
- the results are incorrect.

3.8.3 Reference

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, Francis Lau, "A C-LSTM Neural Network for Text Classification," (arXiv:1511.08630). [arXiv]

"CS231n Convolutional Neural Networks for Visual Recognition," Stanford Online Course. [link]

Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom, "A Convolutional Neural Network for Modelling Sentences," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pp. 655-665 (2014). [arXiv]

Tal Perry, "Convolutional Methods for Text," *Medium* (2017). [Medium]

Yoon Kim, "Convolutional Neural Networks for Sentence Classification," *EMNLP* 2014, 1746-1751 (arXiv:1408.5882). [arXiv]

Zackary C. Lipton, John Berkowitz, "A Critical Review of Recurrent Neural Networks for Sequence Learning," arXiv:1506.00019 (2015). [arXiv]

Home: *Homepage of shorttext*

3.9 Maximum Entropy (MaxEnt) Classifier

3.9.1 Maxent

Maximum entropy (maxent) classifier has been a popular text classifier, by parameterizing the model to achieve maximum categorical entropy, with the constraint that the resulting probability on the training data with the model being equal to the real distribution.

The maxent classifier in *shorttext* is impleneted by *keras*. The optimization algorithm is defaulted to be the Adam optimizer, although other gradient-based or momentum-based optimizers can be used. The traditional methods such as generative iterative scaling (GIS) or L-BFGS cannot be used here.

To use the maxent classifier, import the package:

```
>>> import shorttext
>>> from shorttext.classifiers import MaxEntClassifier
```

Loading NIH reports as an example:

```
>>> classdict = shorttext.data.nihreports()
```

The classifier can be instantiated by:

```
>>> classifier = MaxEntClassifier()
```

Train the classifier:

```
>>> classifier.train(classdict, nb_epochs=1000)
```

After training, it can be used for classification, such as

```
>>> classifier.score('cancer immunology')   # NCI tops the score
>>> classifier.score('children health')     # NIAID tops the score
>>> classifier.score('Alzheimer disease and aging')    # NIAID tops the score
```

To save the model,

```
>>> classifier.save_compact_model('/path/to/filename.bin')
```

To load the model to be a classifier, enter:

```
>>> classifier2 = shorttext.classifiers.load_maxent_classifier('/path/to/filename.bin')
```

3.9.2 Reference

Adam L. Berger, Stephen A. Della Pietra, Vincent J. Della Pietra, "A Maximum Entropy Approach to Natural Language Processing," *Computational Linguistics* 22(1): 39-72 (1996). [ACM]

Daniel E. Russ, Kwan-Yuet Ho, Joanne S. Colt, Karla R. Armenti, Dalsu Baris, Wong-Ho Chow, Faith Davis, Alison Johnson, Mark P. Purdue, Margaret R. Karagas, Kendra Schwartz, Molly Schwenn, Debra T. Silverman, Patricia A. Stewart, Calvin A. Johnson, Melissa C. Friesen, "Computer-based coding of free-text job descriptions to efficiently and reliably incorporate occupational risk factors into large-scale epidemiological studies", *Occup. Environ. Med.* 73, 417-424 (2016). [BMJ]

Daniel Russ, Kwan-yuet Ho, Melissa Friesen, "It Takes a Village To Solve A Problem in Data Science," Data Science Maryland, presentation at Applied Physics Laboratory (APL), Johns Hopkins University, on June 19, 2017. (2017) [Slideshare]

Home: *Homepage of shorttext*

3.10 Character-Based Sequence-to-Sequence (seq2seq) Models

Since release 0.6.0, *shorttext* supports sequence-to-sequence (seq2seq) learning. While there is a general seq2seq class behind, it provides a character-based seq2seq implementation.

3.10.1 Creating One-hot Vectors

To use it, create an instance of the class `shorttext.generators.SentenceToCharVecEncoder`:

```
>>> import numpy as np
>>> import shorttext
>>> from urllib.request import urlopen
>>> chartovec_encoder = shorttext.generators.initSentenceToCharVecEncoder(urlopen('http:/
↪/norvig.com/big.txt', 'r'))
```

The above code is the same as *Character to One-Hot Vector* .

3.10.2 Training

Then we can train the model by creating an instance of `shorttext.generators.CharBasedSeq2SeqGenerator`:

```
>>> latent_dim = 100
>>> seq2seqer = shorttext.generators.CharBasedSeq2SeqGenerator(chartovec_encoder, latent_
↪dim, 120)
```

And then train this neural network model:

```
>>> seq2seqer.train(text, epochs=100)
```

This model takes several hours to train on a laptop.

3.10.3 Decoding

After training, we can use this class as a generative model of answering questions as a chatbot:

```
>>> seq2seqer.decode('Happy Holiday!')
```

It does not give definite answers because there is a stochasticity in the prediction.

3.10.4 Model I/O

This model can be saved by entering:

```
>>> seq2seqer.save_compact_model('/path/to/norvigtxt_iter5model.bin')
```

And can be loaded by:

```
>>> seq2seqer2 = shorttext.generators.seq2seq.charbaseS2S.loadCharBasedSeq2SeqGenerator(
↪'/path/to/norvigtxt_iter5model.bin')
```

3.10.5 Reference

Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (Sebastopol, CA: O'Reilly Media, 2017). [O'Reilly]

Ilya Sutskever, James Martens, Geoffrey Hinton, "Generating Text with Recurrent Neural Networks," *ICML* (2011). [UToronto]

Ilya Sutskever, Oriol Vinyals, Quoc V. Le, "Sequence to Sequence Learning with Neural Networks," arXiv:1409.3215 (2014). [arXiv]

Oriol Vinyals, Quoc Le, "A Neural Conversational Model," arXiv:1506.05869 (2015). [arXiv]

Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria, "Recent Trends in Deep Learning Based Natural Language Processing," arXiv:1708.02709 (2017). [arXiv]

Zackary C. Lipton, John Berkowitz, "A Critical Review of Recurrent Neural Networks for Sequence Learning," arXiv:1506.00019 (2015). [arXiv]

3.11 Stacked Generalization

"Stacking generates the members of the stacking ensemble using several learning algorithms and subsequently uses another algorithm to learn how to combine their outputs." It combines the classification results of several classifiers, and combines them.

Stacking is most commonly implemented using logistic regression. Suppose there are K classifiers, and l output labels. Then the stacking generalization is this logistic model:

$P(y=c|x)=\frac{1}{\exp\left(-\sum_{k=1}^{K} w_{kc}x_{kc}+b_c\right)+1}$

Here we demonstrate the use of stacking of two classifiers.

Import the package, and employ the subject dataset as the training dataset.

```
>>> import shorttext
>>> subdict = shorttext.data.subjectkeywords()
```

Train a C-LSTM model.

```
>>> wvmodel = shorttext.utils.load_word2vec_model('/path/to/GoogleNews-vectors-
↪negative300.bin.gz')
>>> clstm_nnet = shorttext.classifiers.frameworks.CLSTMWordEmbed(len(subdict))
>>> clstm_classifier = shorttext.classifiers.VarNNEmbeddedVecClassifier(wvmodel)
>>> clstm_classifier.train(subdict, clstm_nnet)
```

A test of its classification:

```
>>> clstm_classifier.score('linear algebra')
{'mathematics': 1.0, 'physics': 3.3643366e-10, 'theology': 1.0713742e-13}
>>> clstm_classifier.score('topological soliton')
{'mathematics': 2.0036438e-11, 'physics': 1.0, 'theology': 4.4903334e-14}
```

And we train an SVM, with topic vectors as the input vectors. The topic model is LDA with 128 topics.

```
>>> # train the LDA topic model
>>> lda128 = shorttext.classifiers.LDAModeler()
>>> lda128.train(subdict, 128)
>>> # train the SVM classifier
>>> from sklearn.svm import SVC
>>> lda128_svm_classifier = shorttext.classifiers.TopicVectorSkLearnClassifier(lda128,␣
↪SVC())
>>> lda128_svm_classifier.train(subdict)
```

A test of its classification:

```
>>>  lda128_svm_classifier.score('linear algebra')
{'mathematics': 1.0, 'physics': 0.0, 'theology': 0.0}
>>> lda128_svm_classifier.score('topological soliton')
{'mathematics': 0.0, 'physics': 1.0, 'theology': 0.0}
```

Then we can implement the stacked generalization using logistic regression by calling:

```
>>> stacker = shorttext.stack.LogisticStackedGeneralization(intermediate_classifiers={
↪'clstm': clstm_classifier, 'lda128': lda128_svm_classifier})
>>> stacker.train(subdict)
```

Now the model is ready. As a result, we can do the stacked classification:

```
>>> stacker.score('linear algebra')
{'mathematics': 0.55439126, 'physics': 0.036988281, 'theology': 0.039665185}
>>> stacker.score('quantum mechanics')
{'mathematics': 0.059210967, 'physics': 0.55031472, 'theology': 0.04532773}
>>> stacker.score('topological dynamics')
{'mathematics': 0.17244603, 'physics': 0.19720334, 'theology': 0.035309207}
>>> stacker.score('christology')
 {'mathematics': 0.094574735, 'physics': 0.053406414, 'theology': 0.3797417}
```

The stacked generalization can be saved by calling:

```
>>> stacker.save_compact_model('/path/to/logitmodel.bin')
```

This only saves the stacked generalization model, but not the intermediate classifiers. The reason for this is for allowing flexibility for users to supply their own algorithms, as long as they have the *score* functions which output the same way as the classifiers offered in this package. To load them, initialize it in the same way:

```
>>> stacker2 = shorttext.stack.LogisticStackedGeneralization(intermediate_classifiers={
↪'clstm': clstm_classifier, 'lda128': lda128_svm_classifier})
>>> stacker2.load_compact_model('/path/to/logitmodel.bin')
```

3.11.1 Reference

"Combining the Best of All Worlds," *Everything About Data Analytics*, WordPress (2016). [WordPress]

David H. Wolpert, "Stacked Generalization," *Neural Netw* 5: 241-259 (1992).

M. Paz Sesmero, Agapito I. Ledezma, Araceli Sanchis, "Generating ensembles of heterogeneous classifiers using Stacked Generalization," *WIREs Data Mining and Knowledge Discovery* 5: 21-34 (2015).

Home: *Homepage of shorttext*

3.12 Metrics

The package *shorttext* provides a few metrics that measure the distances of some kind. They are all under **:module:`shorttext.metrics`**. The soft Jaccard score is based on spellings, and the Word Mover's distance (WMD) embedded word vectors.

3.12.1 Edit Distance and Soft Jaccard Score

Edit distance, or Damerau-Levenshtein distance, measures the differences between two words due to insertion, deletion, transposition, substitution etc. Each of this change causes a distance of 1. The algorithm was written in C.

First import the package:

```
>>> from shorttext.metrics.dynprog import damerau_levenshtein, longest_common_prefix,␣
↪similarity, soft_jaccard_score
```

The distance can be calculated by:

```
>>> damerau_levenshtein('diver', 'driver')        # insertion, gives 1
>>> damerau_levenshtein('driver', 'diver')        # deletion, gives 1
>>> damerau_levenshtein('topology', 'tooplogy')   # transposition, gives 1
>>> damerau_levenshtein('book', 'blok')           # subsitution, gives 1
```

The longest common prefix finds the length of common prefix:

```
>>> longest_common_prefix('topology', 'topological')    # gives 7
>>> longest_common_prefix('police', 'policewoman')      # gives 6
```

The similarity between words is defined as the larger of the following:

$s = 1 - \frac{\text{DL distance}}{\max((len(word1)),(len(word2)))}$ and $s = \frac{\text{longest common prefix}}{\max((len(word1)),(len(word2)))}$

```
>>> similarity('topology', 'topological')    # gives 0.6363636363636364
>>> similarity('book', 'blok')               # gives 0.75
```

Given the similarity, we say that the intersection, for example, between 'book' and 'blok', has 0.75 elements, or the union has 1.25 elements. Then the similarity between two sets of tokens can be measured using Jaccard index, with this "soft" numbers of intersection. Therefore,

```
>>> soft_jaccard_score(['book', 'seller'], ['blok', 'sellers'])     # gives 0.
↪6716417910447762
>>> soft_jaccard_score(['police', 'station'], ['policeman'])        # gives 0.
↪2857142857142858
```

The functions *damerau_levenshtein* and *longest_common_prefix* are implemented using Cython . (Before release 0.7.2, they were interfaced to Python using SWIG (Simplified Wrapper and Interface Generator)).

3.12.2 Word Mover's Distance

Unlike soft Jaccard score that bases similarity on the words' spellings, Word Mover's distance (WMD) the embedded word vectors. WMD is a special case for Earth Mover's distance (EMD), or Wasserstein distance. The calculation of WMD in this package is based on linear programming, and the distance between words are the Euclidean distance by default (not cosine distance), but user can set it accordingly.

Import the modules, and load the word-embedding models:

```
>>> from shorttext.metrics.wasserstein import word_mover_distance
>>> from shorttext.utils import load_word2vec_model
>>> wvmodel = load_word2vec_model('/path/to/model_file.bin')
```

Examples:

```
>>> word_mover_distance(['police', 'station'], ['policeman'], wvmodel)                  ␣
↪   # gives 3.060708999633789
>>> word_mover_distance(['physician', 'assistant'], ['doctor', 'assistants'], wvmodel)  ␣
↪   # gives 2.276337146759033
```

More examples can be found in this IPython Notebook .

In *gensim*, the Word2Vec model allows the calculation of WMD if user installed the package PyEMD. It is based on the scale invariant feature transform (SIFT), an algorithm for EMD based on L1-distance (Manhattan distance). For more details, please refer to their tutorial , and cite the two papers by Ofir Pele and Michael Werman if it is used.

3.12.3 Jaccard Index Due to Cosine Distances

In the above section of edit distance, the Jaccard score was calculated by considering soft membership using spelling. However, we can also compute the soft membership by cosine similarity with

```
>>> from shorttext.utils import load_word2vec_model
>>> wvmodel = load_word2vec_model('/path/to/model_file.bin')
>>> from shorttext.metrics.embedfuzzy import jaccardscore_sents
```

For example, the number of words between the set containing 'doctor' and that containing 'physician' is 0.78060223420956831 (according to Google model), and therefore the Jaccard score is

$$0.78060223420956831/(2 - 0.78060223420956831) = 0.6401538990056869$$

And it can be seen by running it:

```
>>> jaccardscore_sents('doctor', 'physician', wvmodel)   # gives 0.6401538990056869
>>> jaccardscore_sents('chief executive', 'computer cluster', wvmodel)   # gives 0.
↪0022515450768836143
>>> jaccardscore_sents('topological data', 'data of topology', wvmodel)   # gives 0.
↪67588977344632573
```

3.12.4 BERTScore

BERTScore includes a category of metrics that is based on BERT model. This metrics measures the similarity between sentences. To use it,

```
>>> from shorttext.metrics.transformers import BERTScorer
>>> scorer = BERTScorer()    # using default BERT model and tokenizer
>>> scorer.recall_bertscore('The weather is cold.', 'It is freezing.')   # 0.
↪7223385572433472
>>> scorer.precision_bertscore('The weather is cold.', 'It is freezing.')   # 0.
↪7700849175453186
>>> scorer.f1score_bertscore('The weather is cold.', 'It is freezing.')   # 0.
↪7454479746418043
```

For BERT models, please refer to *Word Embedding Models* for more details.

3.12.5 Reference

"Damerau-Levenshtein Distance." [Wikipedia]

"Jaccard index." [Wikipedia]

Daniel E. Russ, Kwan-Yuet Ho, Calvin A. Johnson, Melissa C. Friesen, "Computer-Based Coding of Occupation Codes for Epidemiological Analyses," *2014 IEEE 27th International Symposium on Computer-Based Medical Systems* (CBMS), pp. 347-350. (2014) [IEEE]

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, Kilian Q. Weinberger, "From Word Embeddings to Document Distances," *ICML* (2015).

Ofir Pele, Michael Werman, "A linear time histogram metric for improved SIFT matching," *Computer Vision - ECCV 2008*, 495-508 (2008). [ACM]

Ofir Pele, Michael Werman, "Fast and robust earth mover's distances," *Proc. 2009 IEEE 12th Int. Conf. on Computer Vision*, 460-467 (2009). [IEEE]

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, Yoav Artzi, "BERTScore: Evaluating Text Generation with BERT," arXiv:1904.09675 (2019). [arXiv]

"Word Mover's Distance as a Linear Programming Problem," *Everything About Data Analytics*, WordPress (2017). [WordPress]

Home: *Homepage of shorttext*

3.13 Spell Correctors

This package supports the use of spell correctors, because typos are very common in relatively short text data.

There are two types of spell correctors provided: the one described by Peter Norvig (using n-grams Bayesian method), and another by Keisuke Sakaguchi and his colleagues (using semi-character level recurrent neural network).

```
>>> import shorttext
```

We use the Norvig's training corpus as an example. To load it,

```
>>> from urllib.request import urlopen
>>> text = urlopen('https://norvig.com/big.txt').read()
```

The developer just has to instantiate the spell corrector, and then train it with a corpus to get a correction model. Then one can use it for correction.

3.13.1 Norvig

Peter Norvig described a spell corrector based on Bayesian approach and edit distance. You can refer to his blog for more information.

```
>>> norvig_corrector = shorttext.spell.NorvigSpellCorrector()
>>> norvig_corrector.train(text)
>>> norvig_corrector.correct('oranhe')   # gives "orange"
```

3.13.2 Sakaguchi (SCRNN - semi-character recurrent neural network)

Keisuke Sakaguchi and his colleagues developed this spell corrector with the insight that most of the typos happen in between the spellings. They developed a recurrent neural network that trains possible change within the spellings. There are six modes:

- JUMBLE-WHOLE
- JUMBLE-BEG
- JUMBLE-END
- JUMBLE-INT
- NOISE-INSERT
- NOISE-DELETE
- NOISE-REPLACE

The original intent of their work was not to invent a new spell corrector but to study the "Cmabrigde Uinervtisy" effect, but it is nice to see how it can be implemented as a spell corrector.

```
>>> scrnn_corrector = shorttext.spell.SCRNNSpellCorrector('JUMBLE-WHOLE')
>>> scrnn_corrector.train(text)
>>> scrnn_corrector.correct('oranhe')    # gives "orange"
```

We can persist the SCRNN corrector for future use:

```
>>> scrnn_corrector.save_compact_model('/path/to/spellscrnn.bin')
```

To load,

```
>>> corrector = shorttext.spell.loadSCRNNSpellCorrector('/path/to/spellscrnn.bin')
```

3.13.3 Reference

Keisuke Sakaguchi, Kevin Duh, Matt Post, Benjamin Van Durme, "Robsut Wrod Reocginiton via semi-Character Recurrent Neural Networ," arXiv:1608.02214 (2016). [arXiv]

Peter Norvig, "How to write a spell corrector." (2016) [Norvig]

Home: *Homepage of shorttext*

CHAPTER

FOUR

CONSOLE SCRIPTS

This package provides two scripts.

The development of the scripts is *not stable* yet, and more scripts will be added.

4.1 ShortTextCategorizerConsole

```
usage: ShortTextCategorizerConsole [-h] [--wv WV] [--vecsize VECSIZE]
                                   [--topn TOPN] [--inputtext INPUTTEXT]
                                   model_filepath

Perform prediction on short text with a given trained model.

positional arguments:
  model_filepath        Path of the trained (compact) model.

optional arguments:
  -h, --help            show this help message and exit
  --wv WV               Path of the pre-trained Word2Vec model. (None if not
                        needed)
  --vecsize VECSIZE     Vector dimensions. (Default: 300)
  --topn TOPN           Number of top-scored results displayed. (Default: 10)
  --inputtext INPUTTEXT
                        single input text for classification. Run console if
                        set to None. (Default: None)
```

4.2 ShortTextWordEmbedSimilarity

```
usage: ShortTextWordEmbedSimilarity [-h] [--type TYPE] modelpath

Find the similarities between two short sentences using Word2Vec.

positional arguments:
  modelpath     Path of the Word2Vec model

optional arguments:
  -h, --help    show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --type TYPE  Type of word-embedding model (default: "word2vec"; other
               options: "fasttext", "poincare")
```

4.3 WordEmbedAPI

```
usage: WordEmbedAPI [-h] [--port PORT] [--embedtype EMBEDTYPE] [--debug]
                    filepath

Load word-embedding models into memory.

positional arguments:
  filepath              file path of the word-embedding model

optional arguments:
  -h, --help            show this help message and exit
  --port PORT           port number
  --embedtype EMBEDTYPE
                        type of word-embedding algorithm (default: "word2vec),
                        allowing "word2vec", "fasttext", and "poincare"
  --debug               Debug mode (Default: False)
```

Home: *Homepage of shorttext*

CHAPTER

FIVE

API

API unlisted in tutorials are listed here.

5.1 Shorttext Models Smart Loading

5.2 Supervised Classification using Word Embedding

5.2.1 Module *shorttext.generators.seq2seq.s2skeras*

5.2.2 Module *shorttext.classifiers.embed.sumvec.VarNNSumEmbedVecClassification*

5.3 Neural Networks

5.3.1 Module *shorttext.classifiers.embed.sumvec.frameworks*

5.4 Utilities

5.4.1 Module *shorttext.utils.kerasmodel_io*

5.4.2 Module *shorttext.utils.gensim_corpora*

5.4.3 Module *shorttext.utils.compactmodel_io*

5.5 Metrics

5.5.1 Module *shorttext.metrics.dynprog*

5.5.2 Module *shorttext.metrics.wassersterin*

5.6 Spell Correction

5.6.1 Module *shorttext.spell*

Home: *Homepage of shorttext*

CHAPTER

SIX

FREQUENTLY ASKED QUESTIONS (FAQ)

Q1. Can we use TensorFlow backend?

Ans: Yes, users can use TensorFlow and CNTK backend instead of Theano backend. Refer to Keras Backend for information about switching backends.

Q2. Can we use word-embedding algorithms other than Word2Vec?

Ans: Yes. Besides Word2Vec, you can use FastText and Poincaré embedding. See: *Word Embedding Models* .

Q3. Can this package work on Python 3?

Ans: Since release 1.0.0, this package can be run in Python 2.7, 3.5, and 3.6. (Before that, it operates only under Python 2.7.)
Since release 1.0.7, this package can also be run in Python 3.7 as well.

Q4. Warning or messages pop up when running models involving neural networks. What is the problem?

Ans: Make sure your *keras* have version >= 2.

Q5. How should I cite `shorttext` if I use it in my research?

Ans: For the time being, You do not have to cite a particular paper for using this package. However, if you use any particular functions or class, check out the docstring. If there is a paper (or papers) mentioned, cite those papers. For example, if you use *CNNWordEmbed* in frameworks, according to the docstring, cite Yoon Kim's paper. Refer to this documentation for the reference too.

Q6. Is there any reasons why word-embedding keras layers no longer used since release 0.5.11?

Ans: This functionality is removed since release 0.5.11, due to the following reasons:

- *keras* changed its code that produces this bug;
- the layer is consuming memory;
- only Word2Vec is supported; and
- the results are incorrect.

Q7. I am having trouble in install `shorttext` on Google Cloud Platform. What should I do?

Ans: There is no "Python.h". Run: *sudo apt-get install python3-dev* in SSH shell of the VM instance.

Home: *Homepage of shorttext*

CHAPTER SEVEN

REFERENCES

Adam L. Berger, Stephen A. Della Pietra, Vincent J. Della Pietra, “A Maximum Entropy Approach to Natural Language Processing,” *Computational Linguistics* 22(1): 39-72 (1996). [ACM]

Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (Sebastopol, CA: O’Reilly Media, 2017). [O'Reilly]

Chinmaya Pancholi, “Gensim integration with scikit-learn and Keras,” *Google Summer of Codes* (GSoC) proposal (2017). [Github]

Chinmaya Pancholi, “Chinmaya’s GSoC 2017 Summary: Integration with sklearn & Keras and implementing fastText,” *RaRe Incubator* (September 2, 2017). [RaRe]

Christopher Manning, Hinrich Schütze, *Foundations of Statistical Natural Language Processing* (Cambridge, MA: MIT Press, 1999). [MIT Press]

Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, *Introduction to Information Retrieval* (Cambridge, MA: Cambridge University Press, 2008). [StanfordNLP]

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, Francis Lau, “A C-LSTM Neural Network for Text Classification,” (arXiv:1511.08630). [arXiv]

Daniel E. Russ, Kwan-Yuet Ho, Calvin A. Johnson, Melissa C. Friesen, “Computer-Based Coding of Occupation Codes for Epidemiological Analyses,” *2014 IEEE 27th International Symposium on Computer-Based Medical Systems* (CBMS), pp. 347-350. (2014) [IEEE]

Daniel E. Russ, Kwan-Yuet Ho, Joanne S. Colt, Karla R. Armenti, Dalsu Baris, Wong-Ho Chow, Faith Davis, Alison Johnson, Mark P. Purdue, Margaret R. Karagas, Kendra Schwartz, Molly Schwenn, Debra T. Silverman, Patricia A. Stewart, Calvin A. Johnson, Melissa C. Friesen, “Computer-based coding of free-text job descriptions to efficiently and reliably incorporate occupational risk factors into large-scale epidemiological studies”, *Occup. Environ. Med.* 73, 417-424 (2016). [BMJ]

Daniel Russ, Kwan-yuet Ho, Melissa Friesen, “It Takes a Village To Solve A Problem in Data Science,” Data Science Maryland, presentation at Applied Physics Laboratory (APL), Johns Hopkins University, on June 19, 2017. (2017) [Slideshare]

David H. Wolpert, “Stacked Generalization,” *Neural Netw* 5: 241-259 (1992).

David M. Blei, “Probabilistic Topic Models,” *Communications of the ACM* 55(4): 77-84 (2012). [ACM]

Francois Chollet, “A ten-minute introduction to sequence-to-sequence learning in Keras,” *The Keras Blog*. [Keras]

Francois Chollet, “Building Autoencoders in Keras,” *The Keras Blog*. [Keras]

Hsiang-Fu Yu, Chia-Hua Ho, Yu-Chin Juan, Chih-Jen Lin, “LibShortText: A Library for Short-text Classification.” [NTU]

Ilya Sutskever, James Martens, Geoffrey Hinton, “Generating Text with Recurrent Neural Networks,” *ICML* (2011). [UToronto]

Ilya Sutskever, Oriol Vinyals, Quoc V. Le, “Sequence to Sequence Learning with Neural Networks,” arXiv:1409.3215 (2014). [arXiv]

Jayant Jain, “Implementing Poincaré Embeddings,” RaRe Technologies (2017). [RaRe]

Jeffrey Pennington, Richard Socher, Christopher D. Manning, “GloVe: Global Vectors for Word Representation,” *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532-1543 (2014). [PDF]

Keisuke Sakaguchi, Kevin Duh, Matt Post, Benjamin Van Durme, “Robsut Wrod Reocginiton via semi-Character Recurrent Neural Networ,” arXiv:1608.02214 (2016). [arXiv]

“Keras 2.0 Release Notes.” (2017) [Github]

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, Kilian Q. Weinberger, “From Word Embeddings to Document Distances,” *ICML* (2015).

Maximilian Nickel, Douwe Kiela, “Poincaré Embeddings for Learning Hierarchical Representations,” arXiv:1705.08039 (2017). [arXiv]

Michael Czerny, “Modern Methods for Sentiment Analysis,” *District Data Labs (2015). [DistrictDataLabs]

M. Paz Sesmero, Agapito I. Ledezma, Araceli Sanchis, “Generating ensembles of heterogeneous classifiers using Stacked Generalization,” *WIREs Data Mining and Knowledge Discovery* 5: 21-34 (2015).

Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom, “A Convolutional Neural Network for Modelling Sentences,” *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pp. 655-665 (2014). [arXiv]

Oriol Vinyals, Quoc Le, “A Neural Conversational Model,” arXiv:1506.05869 (2015). [arXiv]

Peter Norvig, “How to write a spell corrector.” (2016) [Norvig]

Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, “Enriching Word Vectors with Subword Information,” arXiv:1607.04606 (2016). [arXiv]

Radim Rehurek, Petr Sojka, “Software Framework for Topic Modelling with Large Corpora,” In Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks (2010). [ResearchGate]

Sebastian Ruder, “An overview of gradient descent optimization algorithms,” blog of Sebastian Ruder, arXiv:1609.04747 (2016). [Ruder or arXiv]

Tal Perry, “Convolutional Methods for Text,” *Medium* (2017). [Medium]

Thomas W. Jones, “textmineR: Functions for Text Mining and Topic Modeling,” CRAN Project. [CRAN or Github]

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, “Efficient Estimation of Word Representations in Vector Space,” *ICLR* 2013 (2013). [arXiv]

Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria, “Recent Trends in Deep Learning Based Natural Language Processing,” arXiv:1708.02709 (2017). [arXiv]

Xuan Hieu Phan, Cam-Tu Nguyen, Dieu-Thu Le, Minh Le Nguyen, Susumu Horiguchi, Quang-Thuy Ha, “A Hidden Topic-Based Framework toward Building Applications with Short Web Documents,” *IEEE Trans. Knowl. Data Eng.* 23(7): 961-976 (2011).

Xuan Hieu Phan, Le-Minh Nguyen, Susumu Horiguchi, “Learning to Classify Short and Sparse Text & Web withHidden Topics from Large-scale Data Collections,” WWW ‘08 Proceedings of the 17th international conference on World Wide Web. (2008) [ACL]

Yoon Kim, “Convolutional Neural Networks for Sentence Classification,” *EMNLP* 2014, 1746-1751 (arXiv:1408.5882). [arXiv]

Zackary C. Lipton, John Berkowitz, “A Critical Review of Recurrent Neural Networks for Sequence Learning,” arXiv:1506.00019 (2015). [arXiv]

Home: *Homepage of shorttext*

CHAPTER

EIGHT

LINKS

8.1 Project Codes and Package

- Github
- PyPI

8.2 Issues

To report bugs and issues, please go to Issues.

8.3 Gensim Incubator

Chinmaya Pancholi, a student in Indian Institute of Technology, Kharagpur, is supported by Google Summer of Code (GSoC) project to support the open-source project for *gensim*. Part of his project is to employ the wrapping ideas in *shorttext* to integrate *keras*, *scikit-learn* and *gensim*.

Chinmaya's blog posts: https://rare-technologies.com/author/chinmaya/

Chinmaya's proposal for GSoC: https://github.com/numfocus/gsoc/blob/master/2017/proposals/Chinmaya_Pancholi.md

8.4 Blog Entries

"R or Python on Text Mining," *Everything About Data Analytics*, WordPress (2015). [WordPress]

"Short Text Categorization using Deep Neural Networks and Word-Embedding Models," *Everything About Data Analytics*, WordPress (2015). [WordPress] (A code demonstration can be found in an early version of the Github repository for this package: here)

"Toying with Word2Vec," *Everything About Data Analytics*, WordPress (2015). [WordPress]

"Probabilistic Theory of Word Embeddings: GloVe," *Everything About Data Analytics*, WordPress (2016). [WordPress]

"Word-Embedding Algorithms," *Everything About Data Analytics*, WordPress (2016). [WordPress]

"Python Package for Short Text Mining," *Everything About Data Analytics*, WordPress (2016). [WordPress]

"Short Text Mining using Advanced Keras Layers and Maxent: shorttext 0.4.1," *Everything About Data Analytics*, WordPress (2017). [WordPress]

“Word Mover’s Distance as a Linear Programming Problem,” *Everything About Data Analytics*, WordPress (2017). [WordPress]

“Release of shorttext 0.5.4,” *Everything About Data Analytics*, WordPress (2017). [WordPress]

“Document-Term Matrix: Text Mining in R and Python,” *Everything About Data Analytics*, WordPress (2018). [WordPress]

“Package shorttext 1.0.0 Released,” Medium (2018). [Medium]

Home: *Homepage of shorttext*

CHAPTER NINE

NEWS

- 12/21/2023: *shorttext* 1.6.1 released.
- 08/26/2023: *shorttext* 1.6.0 released.
- 06/19/2023: *shorttext* 1.5.9 released.
- 09/23/2022: *shorttext* 1.5.8 released.
- 09/22/2022: *shorttext* 1.5.7 released.
- 08/29/2022: *shorttext* 1.5.6 released.
- 05/28/2022: *shorttext* 1.5.5 released.
- 12/15/2021: *shorttext* 1.5.4 released.
- 07/11/2021: *shorttext* 1.5.3 released.
- 07/06/2021: *shorttext* 1.5.2 released.
- 04/10/2021: *shorttext* 1.5.1 released.
- 04/09/2021: *shorttext* 1.5.0 released.
- 02/11/2021: *shorttext* 1.4.8 released.
- 01/11/2021: *shorttext* 1.4.7 released.
- 01/03/2021: *shorttext* 1.4.6 released.
- 12/28/2020: *shorttext* 1.4.5 released.
- 12/24/2020: *shorttext* 1.4.4 released.
- 11/10/2020: *shorttext* 1.4.3 released.
- 10/18/2020: *shorttext* 1.4.2 released.
- 09/23/2020: *shorttext* 1.4.1 released.
- 09/02/2020: *shorttext* 1.4.0 released.
- 07/23/2020: *shorttext* 1.3.0 released.
- 06/05/2020: *shorttext* 1.2.6 released.
- 05/20/2020: *shorttext* 1.2.5 released.
- 05/13/2020: *shorttext* 1.2.4 released.
- 04/28/2020: *shorttext* 1.2.3 released.
- 04/07/2020: *shorttext* 1.2.2 released.

- 03/23/2020: *shorttext* 1.2.1 released.
- 03/21/2020: *shorttext* 1.2.0 released.
- 12/01/2019: *shorttext* 1.1.6 released.
- 09/24/2019: *shorttext* 1.1.5 released.
- 07/20/2019: *shorttext* 1.1.4 released.
- 07/07/2019: *shorttext* 1.1.3 released.
- 06/05/2019: *shorttext* 1.1.2 released.
- 04/23/2019: *shorttext* 1.1.1 released.
- 03/03/2019: *shorttext* 1.1.0 released.
- 02/14/2019: *shorttext* 1.0.8 released.
- 01/30/2019: *shorttext* 1.0.7 released.
- 01/29/2019: *shorttext* 1.0.6 released.
- 01/13/2019: *shorttext* 1.0.5 released.
- 10/03/2018: *shorttext* 1.0.4 released.
- 08/06/2018: *shorttext* 1.0.3 released.
- 07/24/2018: *shorttext* 1.0.2 released.
- 07/17/2018: *shorttext* 1.0.1 released.
- 07/14/2018: *shorttext* 1.0.0 released.
- 06/18/2018: *shorttext* 0.7.2 released.
- 05/30/2018: *shorttext* 0.7.1 released.
- 05/17/2018: *shorttext* 0.7.0 released.
- 02/27/2018: *shorttext* 0.6.0 released.
- 01/19/2018: *shorttext* 0.5.11 released.
- 01/15/2018: *shorttext* 0.5.10 released.
- 12/14/2017: *shorttext* 0.5.9 released.
- 11/08/2017: *shorttext* 0.5.8 released.
- 10/27/2017: *shorttext* 0.5.7 released.
- 10/17/2017: *shorttext* 0.5.6 released.
- 09/28/2017: *shorttext* 0.5.5 released.
- 09/08/2017: *shorttext* 0.5.4 released.
- 09/02/2017: end of GSoC project.
- 08/22/2017: *shorttext* 0.5.1 released.
- 07/28/2017: *shorttext* 0.4.1 released.
- 07/26/2017: *shorttext* 0.4.0 released.
- 06/16/2017: *shorttext* 0.3.8 released.
- 06/12/2017: *shorttext* 0.3.7 released.

- 06/02/2017: *shorttext* 0.3.6 released.
- 05/30/2017: GSoC project (Chinmaya Pancholi).
- 05/16/2017: *shorttext* 0.3.5 released.
- 04/27/2017: *shorttext* 0.3.4 released.
- 04/19/2017: *shorttext* 0.3.3 released.
- 03/28/2017: *shorttext* 0.3.2 released.
- 03/14/2017: *shorttext* 0.3.1 released.
- 02/23/2017: *shorttext* 0.2.1 released.
- 12/21/2016: *shorttext* 0.2.0 released.
- 11/25/2016: *shorttext* 0.1.2 released.
- 11/21/2016: *shorttext* 0.1.1 released.

9.1 What's New

9.2 Released 1.6.1 (December 21, 2023)

- Updated package requirements.

9.3 Released 1.6.0 (August 26, 2023)

- Pinned requirements for ReadTheDocs documentation;
- Fixed bugs in word-embedding model mean pooling classifiers;
- Updated package requirements.

9.4 Release 1.5.9 (June 19, 2023)

- Support for Python 3.11;
- Removing flask.

9.5 Release 1.5.8 (September 23, 2022)

- Package administration.

9.6 Release 1.5.7 (September 22, 2022)

- Removal of requirement of pre-installation of *numpy* and *Cython*.

9.7 Release 1.5.6 (August 29, 2022)

- Speeding up inference of *VarNNEmbeddedVecClassifier*. (Acknowledgement: Ritesh Agrawal)

9.8 Release 1.5.5 (May 28, 2022)

- Support for Python 3.10.

9.9 Release 1.5.4 (December 15, 2021)

- Non-negative stop words.

9.10 Release 1.5.3 (July 11, 2021)

- Documentation updated.

9.11 Release 1.5.2 (July 6, 2021)

- Resolved bugs regarding *keras* import.
- Support for Python 3.9.

9.12 Release 1.5.1 (April 10, 2021)

- Replaced TravisCI with CircleCI in the continuous integration pipeline.

9.13 Release 1.5.0 (April 09, 2021)

- Removed support for Python 3.6.
- Removed buggy BERT representations unit test.

9.14 Release 1.4.8 (February 11, 2021)

- Updated requirements for *scipy* for Python 3.7 or above.

9.15 Release 1.4.7 (January 11, 2021)

- Updated version of *transformers* in *requirement.txt*;
- Updated BERT encoder for the change of implementation;
- Fixed unit tests.

9.16 Release 1.4.6 (January 3, 2021)

- Bug regarding Python 3.6 requirement for *scipy*.

9.17 Release 1.4.5 (December 28, 2020)

- Bugs fixed about Python 2 to 3 updates, *filter* in *shorttext.metrics.embedfuzzy*.

9.18 Release 1.4.4 (December 24, 2020)

- Bugs regarding *SumEmbedVeccClassification.py*;
- Fixing bugs due to Python 3.6 restriction on *scipy*.

9.19 Release 1.4.3 (November 10, 2020)

- Bugs about transformer-based model on different devices resolved.

9.20 Release 1.4.2 (October 18, 2020)

- Documentation requirements and PyUp configs cleaned up.

9.21 Release 1.4.1 (September 23, 2020)

- Documentation and codes cleaned up.

9.22 Release 1.4.0 (September 2, 2020)

- Provided support BERT-based sentence and tokens embeddings;
- Implemented support for BERTScores.

9.23 Release 1.3.0 (July 23, 2020)

- Removed all dependencies on *PuLP*; all computations of word mover's distance (WMD) is performed using *SciPy*.

9.24 Release 1.2.6 (June 20, 2020)

- Removed Python-2 codes (*urllib2*).

9.25 Release 1.2.5 (May 20, 2020)

- Update on *gensim* package usage and requirements;
- Removed some deprecated functions.

9.26 Release 1.2.4 (May 13, 2020)

- Update on *scikit-learn* requirements to *>=0.23.0*.
- Directly dependence on *joblib*;
- Support for Python 3.8 added.

9.27 Release 1.2.3 (April 28, 2020)

- PyUP scan implemented;
- Support for Python 3.5 decommissioned.

9.28 Release 1.2.2 (April 7, 2020)

- Removed dependence on *PyStemmer*, which is replaced by *snowballstemmer*.

9.29 Release 1.2.1 (March 23, 2020)

- Added port number adjustability for word-embedding API;
- Removal of Spacy dependency.

9.30 Release 1.2.0 (March 21, 2020)

- API for word-embedding algorithm for one-time loading.

9.31 Release 1.1.6 (December 1, 2019)

- Compatibility with TensorFlow 2.0.0.

9.32 Release 1.1.5 (September 24, 2019)

- Decommissioned GCP buckets; using data files stored in AWS S3 buckets.

9.33 Release 1.1.4 (July 20, 2019)

- Minor bugs fixed.

9.34 Release 1.1.3 (July 7, 2019)

- Updated codes for Console code loading;
- Updated Travis CI script.

9.35 Release 1.1.2 (June 5, 2019)

- Updated codes for Fasttext moddel loading as the previous function was deprecated.

9.36 Release 1.1.1 (April 23, 2019)

- Bug fixed. (Acknowledgement: Hamish Dickson)

9.37 Release 1.1.0 (March 3, 2019)

- Size of embedded vectors set to 300 again when necessary; (possibly break compatibility)
- Moving corpus data from Github to Google Cloud Storage.

9.38 Release 1.0.8 (February 14, 2019)

- Minor bugs fixed.

9.39 Release 1.0.7 (January 30, 2019)

- Compatibility with Python 3.7 with TensorFlow as the backend.

9.40 Release 1.0.7 (January 30, 2019)

- Compatibility with Python 3.7 with Theano as the backend;
- Minor documentation changes.

9.41 Release 1.0.6 (January 29, 2019)

- Documentation change;
- Word-embedding model used in unit test stored in Amazon S3 bucket.

9.42 Release 1.0.5 (January 13, 2019)

- Minor versioning bug fixed.

9.43 Release 1.0.4 (October 3, 2018)

- Package *keras* requirement updated;
- Less dependence on *pandas*.

9.44 Release 1.0.3 (August 6, 2018)

- Bugs regarding I/O of *SumEmbeddedVecClassifier*.

9.45 Release 1.0.2 (July 24, 2018)

- Minor bugs regarding installation fixed.

9.46 Release 1.0.1 (July 14, 2018)

- Minor bugs fixed.

9.47 Release 1.0.0 (July 14, 2018)

- Python-3 compatibility;
- Replacing the original stemmer to use Snowball;
- Certain functions cythonized;
- Various bugs fixed.

9.48 Release 0.7.2 (June 18, 2018)

- Damerau-Levenshtein distance and longest common prefix implemented using Cython.

9.49 Release 0.7.1 (May 30, 2018)

- Decorator replaced by base class *CompactIOMachine*;
- API included in documentation.

9.50 Release 0.7.0 (May 17, 2018)

- Spelling corrections and fuzzy logic;
- More unit tests.

9.51 Release 0.6.0 (February 27, 2018)

- Support of character-based sequence-to-sequence (seq2seq) models.

9.52 Release 0.5.11 (January 19, 2018)

- Removal of word-embedding *keras*-type layers.

9.53 Release 0.5.10 (January 15, 2018)

- Support of encoder module for character-based models;
- Implementation of document-term matrix (DTM).

9.54 Release 0.5.9 (December 14, 2017)

- Support of Poincare embedding;
- Code optimization;
- Script *ShortTextWord2VecSimilarity* updated to *ShortTextWordEmbedSimilarity*.

9.55 Release 0.5.8 (November 8, 2017)

- Removed most explicit user-specification of *vecsize* for given word-embedding models;
- Removed old namespace for topic models (no more backward compatibility).
- Integration of [FastText](https://github.com/facebookresearch/fastText).

9.56 Release 0.5.7 (October 27, 2017)

- Removed most explicit user-specification of *vecsize* for given word-embedding models;
- Removed old namespace for topic models (hence no more backward compatibility).

9.57 Release 0.5.6 (October 17, 2017)

- Updated the neural network framework due to the change in *gensim* API.

9.58 Release 0.5.5 (September 28, 2017)

- Script *ShortTextCategorizerConsole* updated.

9.59 Release 0.5.4 (September 8, 2017)

- Bug fixed;
- New scripts for finding distances between sentences;
- Finding similarity between two sentences using Jaccard index.

9.60 End of GSoC Program (September 2, 2017)

Chinmaya summarized his GSoC program in his blog post posted in RaRe Incubator.

9.61 Release 0.5.1 (August 22, 2017)

- Implementation of Damerau-Levenshtein distance and soft Jaccard score;
- Implementation of Word Mover's distance.

9.62 Release 0.4.1 (July 28, 2017)

- Further Travis.CI update tests;
- Model file I/O updated (for huge models);
- Migrating documentation to readthedocs.org; previous documentation at *Pythonhosted.org* destroyed.

9.63 Release 0.4.0 (July 26, 2017)

- Maximum entropy models;
- Use of *gensim* Word2Vec *keras* layers;
- Incorporating new features from *gensim*;
- Use of Travis.CI for pull request testing.

9.64 Release 0.3.8 (June 16, 2017)

- Bug fixed on *sumvecframeworks*.

9.65 Release 0.3.7 (June 12, 2017)

- Bug fixed on *VarNNSumEmbedVecClassifier*.

9.66 Release 0.3.6 (June 2, 2017)

- Added deprecation decorator;
- Fixed path configurations;
- Added "update" corpus capability to *gensim* models.

9.67 Google Summer of Code (May 30, 2017)

Chinamaya Pancholi, a Google Summer of Code (GSoC) student, is involved in the open-source development of *gensim*, that his project will be very related to the *shorttext* package. More information can be found in his first blog entry .

9.68 Release 0.3.5 (May 16, 2017)

- Refactoring topic modeling to generators subpackage, but keeping package backward compatible.
- Added Inaugural Addresses as an example training data;
- Fixed bugs about package paths.

9.69 Release 0.3.4 (Apr 27, 2017)

- Fixed relative path loading problems.

9.70 Release 0.3.3 (Apr 19, 2017)

- Deleted *CNNEmbedVecClassifier*;
- Added script *ShortTextWord2VecSimilarity*.

More Info

9.71 Release 0.3.2 (Mar 28, 2017)

- Bug fixed for *gensim* model I/O;
- Console scripts update;
- Neural networks up to Keras 2 standard (refer to this).

9.72 Release 0.3.1 (Mar 14, 2017)

- Compact model I/O: all models are in single files;
- Implementation of stacked generalization using logistic regression.

9.73 Release 0.2.1 (Feb 23, 2017)

- Removal attempts of loading GloVe model, as it can be run using *gensim* script;
- Confirmed compatibility of the package with *tensorflow*;
- Use of *spacy* for tokenization, instead of *nltk*;
- Use of *stemming* for Porter stemmer, instead of *nltk*;
- Removal of *nltk* dependencies;
- Simplifying the directory and module structures;
- Module packages updated.

More Info

9.74 Release 0.2.0 (Dec 21, 2016)

Home: *Homepage of shorttext*

CHAPTER TEN

INDICES AND TABLES

- genindex
- modindex
- search