LibShortText: A Library for Short-text Classification and Analysis

Hsiang-Fu Yu ROFUYU@CS.UTEXAS.EDU
Department of Computer Science, University of Texas at Austin, Austin, TX 78712 USA

Chia-Hua Ho B95082@CSIE.NTU.EDU.TW
Yu-Chin Juan R01922136@CSIE.NTU.EDU.TW
Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science, National Taiwan University, Taipei 106, Taiwan

Editor: Editor name

Abstract

LibShortText is an open source library for short-text classification and analysis. Properties of short texts are considered in its design and implementation. The package supports effective text pre-processing and fast training/prediction procedures. We provide an interactive tool to perform error analysis. Because of the short length, details of each short text can be investigated easily. In addition, the package is designed so that users can conveniently make extensions. The ease of use, efficiency, and extensibility of LibShortText make it very useful for practitioners working on short-text classification and analysis. The package is available at `http://www.csie.ntu.edu.tw/~cjlin/libshorttext`

Keywords: short-text classification, interactive error analysis, open source, linear classification, machine learning

1. Introduction

Recently, there has been a growing interest in short-text classification and analysis. Application domains include titles (e.g., Shen et al., 2012), questions (e.g., Zhang and Lee, 2003; Qu et al., 2012), sentences (e.g., Khoo et al., 2006), and short messages. Existing text-classification packages may not be suitable for analyzing short texts because of not considering their special properties. For example, it is difficult to analyze a text with many words, but we can easily investigate details in training/predicting short texts. Unlike general texts such as web pages and emails, short texts in a corpus may have similar lengths and words in a short text are often distinct. In addition, practitioners wonder if the standard procedures for text classification must be applied when experimenting with short texts.

We develop LibShortText as an open source tool (under the BSD license[1]) for short-text classification and analysis. It has the following features.

1. For large-scale short-text classification, it is more efficient than general text-mining tools.
2. Based on our study (Yu et al., 2012), we carefully select the default options for LibShortText. Users need not try many options to get the best performance for their applications.
3. We provide an interactive tool to perform error analysis. In particular, because of short lengths, details of each text can be investigated.

1. The New BSD licences approved by the Open Source Initiative.

Workflow	Libraries	Command-line tools
short texts ↓ pre-processing feature generation	`libshorttext.converter`	`text-train.py` `text2svm.py`
↓ training testing	`libshorttext.classifier`	`text-train.py` `text-predict.py`
↓ analysis	`libshorttext.analyzer`	

Figure 1: Workflow of LibShortText and the corresponding libraries/command-line tools.

4. The package is mainly written in Python for simplicity and extensibility, while time-consuming operations are implemented in C/C++ for efficiency.
5. We provide full documentation including class references and examples.

2. The Software Package

The workflow of LibShortText is shown in Figure 1. Each step corresponds to a library in the core Python module `libshorttext`.

1. `libshorttext.converter`: For given short texts, LibShortText follows the bag-of-word model to generate features. Users apply procedures in this library to pre-process short texts by tokenization, stemming (optional), and stop-word removal (optional). The library also allows users to choose between unigram and bigram features.
2. `libshorttext.classifier`: Following users' choice on feature representation (binary, word count, TF, or TF-IDF), this library generates sparse feature vectors, conducts instance-wise normalization (optional), and calls the popular linear-classification package LIBLINEAR (Fan et al., 2008) for training/testing. All these operations can be easily built upon LIBLINEAR's Python interface, but for computational and memory efficiency, part of the implementation is in C/C++. We carefully design this library so that no modification of LIBLINEAR is required. For multi-class classification, we follow LIBLINEAR to support the one-versus-rest approach (e.g., Bottou et al., 1994) and the method by Crammer and Singer (2001). In addition, a parameter selection tool for selecting the penalty parameter of SVM/logistic regression formulations of LIBLINEAR is provided.
3. `libshorttext.analyser`: The library provides a simple interactive tool to conduct error analysis in both macro and micro levels. By macro level we mean the overall performance (e.g., accuracy), while the micro level involves the analysis of each feature of a short text.

Based on the study (Yu et al., 2012) for product title classification, we set the following default options.

	Default options
pre-processing feature-generation	no stemming, no stop word removal, bigram features
training	instance-wise normalization, binary feature representation, Crammer & Singer multi-class SVM

For general texts, one option suitable for some texts may not suitable for others. In contrast, it is easier to select options based on properties of short texts. For example, because in a short text words are generally distinct, Yu et al. (2012) show that the SVM models obtained by binary and TF-IDF feature representations give similar performance.

2.1 Command-line and Interactive Use

We provide two main command-line tools: `text-train.py` pre-processes a text file and trains a model, while `text-predict.py` predicts another text file to obtain classification accuracy and generates an output file of predicted results.

```
$ python text-train.py train_file
[output skipped]
$ python text-predict.py test_file train_file.model predict_result
Accuracy = 90.7873% (9662757/10643283)
```

Because users may train with the same data several times after the pre-processing phase, we provide `text2svm.py` to convert short texts to sparse feature vectors of term frequency. Then `text-train.py` can directly train a model on these sparse vectors.

```
$ python text2svm.py train_file
[train_file.text_converter and train_file.svm are generated.]
$ python text-train.py -P train_file.text_converter train_file.svm
```

LibShortText supports interactive error analysis. First, we enter Python, import the module, load the prediction results, and create an object of 'Analyzer' by reading a model.

```
$ python
>>> from libshorttext.analyzer import *
>>> predict_result = InstanceSet('predict_result')
>>> analyzer = Analyzer('train_file.model')
```

We randomly select 1000 instances with two labels 'Books' and 'Music' and generate a confusion table.

```
>>> insts = predict_result.select(with_labels(['Books', 'Music']), subset(1000,
... 'random'))
>>> analyzer.gen_confusion_table(insts)
```

	Books	Music
Books	425	3
Music	9	563

Next, to see how a short text "MICKEY MOUSE POT STAKE" is predicted by our model, the following operation gives the feature weights of the top three classes. For multi-class SVM and logistic regression used in LibShortText, in the trained model, each class has a corresponding weight vector. The decision value is the inner product between the weight vector and the test instance. For this example, the class "Home & Garden" is correctly predicted because of the highest decision value. From the weights, we identify "stake" as an important word for the "Home & Garden" class.

```
>>> analyzer.analyze_single('MICKEY MOUSE POT STAKE', 3)
```

	Home & Garden	Specialty Services	Toys & Hobbies
pot	9.786e-01	4.070e-01	2.023e-01
mouse	2.683e-01	3.613e-01	1.649e-01
stake	1.392e+00	2.801e-01	3.924e-01
mickey mouse	3.564e-02	9.631e-02	6.446e-01
mouse pot	5.168e-01	-7.780e-02	-1.771e-01
pot stake	5.489e-01	-9.084e-03	-3.282e-01
mickey	-8.684e-02	-4.901e-02	1.075e-01
decval	1.381e+00	3.813e-01	3.804e-01

We provide more operations for error analysis. Details are in the documents.

2.2 Making Extensions

Because of the modularized framework, extending LibShortText is easy. We show an example to replace the tokenizer. The default tokenizer assumes that tokens are separated by space characters. If other characters are used, users may write their own tokenizer.

```
1 from libshorttext.converter import *
2
3 def comma_tokenizer(text):
4     return text.lower().split(',')
5
6 text_converter = text2svm_converter()
7 text_converter.text_prep.tokenizer = comma_tokenizer
```

We define `comma_tokenizer` to separate tokens by commas. We then create an instance of converter, `text_converter`, and assign the new tokenizer to it in lines 6–7.

3. Experiments

To demonstrate the efficiency and effectiveness of LibShortText, we crawled 20 million eBay product titles in 34 classes. The first 10 million titles are for training, while the next 10 million are for testing.[2] On a machine with Intel Xeon 2.4GHz CPU (E5620), 12MB Cache, and 64GB RAM, running `text-train.py` with default options takes about 37 minutes, where 21 minutes are for generating features from texts and 16 minutes are for training. For prediction, `text-predict.py` takes 28 minutes and achieves 90.7873% accuracy. In contrast, a general text-mining tool such as RapidMiner (Mierswa et al., 2006) fails to even pre-process the data because of the memory issue. On a subset of prod-title with 100K instances, LibShortText takes only 17 seconds and 240 MB memory, while RapidMiner requires 10 minutes and 3 GB memory.

4. Conclusions

LibShortText is an easy-to-use, efficient, and extensible tool for short-text classification and analysis. Its design and implementation incorporate properties of short texts. The package is still being improved by new research results and users' needs.

2. We cannot redistribute the data; see `http://developer.ebay.com/join/licenses/individual`.

References

L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwriting digit recognition. In *International Conference on Pattern Recognition*, pages 77–87. IEEE Computer Society Press, 1994.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9: 1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

Anthony Khoo, Yuval Marom, and David Albrecht. Experiments with sentence classification. In *Australasian Language Technology Workshop*, 2006.

Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 935–940, 2006.

Bo Qu, Gao Cong, Cuiping Li, Aixin Sun, and Hong Chen. An evaluation of classification models for question topic categorization. *Journal of the American Society for Information Science and Technology*, 63:889–903, 2012.

Dan Shen, Jean-David Ruvini, and Badrul Sarwar. Large-scale item categorization for e-commerce. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM)*, pages 595–604, 2012.

Hsiang-Fu Yu, Chia-Hua Ho, Prakash Arunachalam, Manas Somaiya, and Chih-Jen Lin. Product title classification versus text classification. Technical report, 2012. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/title.pdf`.

Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 26–32. 2003.