MeTA: A Unified Toolkit for Text Retrieval and Analysis

Sean Massung, Chase Geigle and **ChengXiang Zhai**
Computer Science Department, College of Engineering
University of Illinois at Urbana-Champaign
{massung1,geigle1,czhai}@illinois.edu

Abstract

MeTA is developed to unite machine learning, information retrieval, and natural language processing in one easy-to-use toolkit. Its focus on indexing allows it to perform well on large datasets, supporting online classification and other out-of-core algorithms. MeTA's liberal open source license encourages contributions, and its extensive online documentation, forum, and tutorials make this process straightforward. We run experiments and show MeTA's performance is competitive with or better than existing software.

1 A Unified Framework

As NLP techniques become more and more mature, we have great opportunities to use them to develop and support many applications, such as search engines, classifiers, and integrative applications that involve multiple components. It's possible to develop each application from scratch, but it's much more efficient to have a general toolkit that supports multiple application types.

Existing tools tend to specialize on one particular area, and as such there is a wide variety of tools one must sample when performing different data science tasks. For text-mining tasks, this is even more apparent; it is extremely difficult (if not impossible) to find tools that support both traditional information retrieval tasks (like tokenization, indexing, and search) alongside traditional machine learning tasks (like document classification, regression, and topic modeling).

Table 1 compares MeTA's many features across various dimensions. Note that only MeTA satisfies all the areas while other toolkits focus on a particular area. In the case where the desired functionality is scattered, data science students, researchers, and practitioners must find the appropriate software packages for their needs and compile and configure each appropriate tool. Then, there is the problem of data formatting—it is unlikely that the tools all have standardized upon a single input format, so a certain amount of "data munging" is required. All of this detracts from the actual task at hand, which has a marked impact on productivity.

The goal of the MeTA project is to address these issues. In particular, we provide a unifying framework for existing machine learning and natural language processing algorithms, allowing researchers to quickly run controlled experiments. We have modularized the feature generation, instance representation, data storage formats, and algorithm implementations; this allows users to make seamless transitions along any of these dimensions with minimal effort. Finally, MeTA is dual-licensed under the University of Illinois/NCSA Open Source Licence and the MIT License to reach the broadest audience possible.

Due to space constraints, in this paper, we only delve into MeTA's natural language processing (NLP), information retrieval (IR), and machine learning (ML) components in section 3. However, we briefly outline all of its components here:

Feature generation. MeTA has a collection of tokenizers, filters, and analyzers that convert raw text into a feature representation. Basic features are n-gram words, but other analyzers make use of different parts of the toolkit, such as POS tag n-grams and parse tree features. An arbitrary number of feature representations may be combined; for example, a document could be represented as unigram words, bigram POS tags, and parse tree rewrite rules. Users can easily add their own feature types as well, such as sentence length distribution in a document.

Search. The MeTA search engine can store

	Indri *IR*	**Lucene** *IR*	**MALLET** *ML/NLP*	**LIBLINEAR** *ML*	**SVM^{MULT}** *ML*	**scikit** *ML/NLP*	**CoreNLP** *ML/NLP*	**MeTA** *all*
Feature generation	✓	✓	✓			✓	✓	✓
Search	✓	✓						✓
Classification			✓	✓	✓	✓	✓	✓
Regression			✓	✓	✓	✓	✓	✓
POS tagging			✓				✓	✓
Parsing							✓	✓
Topic models			✓			✓		✓
n-gram LM								✓
Word embeddings			✓				✓	✓
Graph algorithms								✓
Multithreading		✓	✓			✓	✓	✓

Table 1: Toolkit feature comparison. Citations for all toolkits may be found in their respective comparison sections.

document feature vectors in an inverted index and score them with respect to a query. Rankers include vector space models such as Okapi BM25 (Robertson et al., 1994) and probabilistic models like Dirichlet prior smoothing (Zhai and Lafferty, 2004). A search demo is online[1].

Classification. MeTA includes a normalized adaptive stochastic gradient descent (SGD) implementation (Ross et al., 2013) with pluggable loss functions, allowing creation of an SVM classifier (among others). Both ℓ_1 (Tsuruoka et al., 2009) and ℓ_2 regularization are supported. Ensemble methods for binary classifiers allow multiclass classification. Other classifiers like naïve Bayes and k-nearest neighbors also exist. A confusion matrix class and significance testing framework allow evaluation and comparison of different methods and feature representations.

Regression. Regression via SGD predicts real-valued responses from featurized documents. Evaluation metrics such as mean squared error and R^2 score allow model comparison.

POS tagging. MeTA contains a linear-chain conditional random field for POS tagging and chunking applications, learned using ℓ_2 regularized SGD (Sutton and McCallum, 2012). It also contains an efficient greedy averaged perceptron tagger (Collins, 2002).

Parsing. A fast shift-reduce constituency parser using generalized averaged perceptron (Zhu et al., 2013) is MeTA's grammatical parser. Parse tree featurizers implement different types of structural tree representations (Massung et al., 2013). An NLP demo online presents tokenization, POS-tagging, and parsing[2].

Topic models. MeTA can learn topic models over any feature representation using collapsed variational Bayes (Asuncion et al., 2009), collapsed Gibbs sampling (Griffiths and Steyvers, 2004), stochastic collapsed variational Bayes (Foulds et al., 2013), or approximate distributed LDA (Newman et al., 2009).

n-gram language models (LMs). MeTA takes an ARPA-formatted input[3] and creates a language model that can be queried for token sequence probabilities or used in downstream applications like SyntacticDiff (Massung and Zhai, 2015).

Word embeddings. The GloVe algorithm (Pennington et al., 2014) is implemented in a streaming framework and also features an interactive semantic relationship demo. Word vectors can be used in other applications as part of the MeTA API.

Graph algorithms. Directed and undirected graph implementations exist and various algorithms such as betweenness centrality, PageRank, and myopic search are available. Random graph generation models like Watts-Strogatz and preferential attachment exist. For these algorithms see Easley and Kleinberg (2010).

Multithreading. When possible, MeTA algorithms and applications are parallelized using C++ threads to make full use of available resources.

2 Usability

Consistency across components is a key feature that allows MeTA to work well with large datasets. This is accomplished via a three-layer architecture. On the first layer, we have tokenizers, analyzers, and all the text processing that accompanies them. Once a document representation is determined, this tool chain is run on a corpus. The indexes are the second layer; they pro-

[1]`https://meta-toolkit.org/search-demo.html`
[2]`https://meta-toolkit.org/nlp-demo.html`

[3]`http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html`

vide an efficient format for storing processed data. The third layer—the application layer—interfaces solely with indexes. This means that we may use the same index for running an SVM as we do to evaluate a ranking function, *without processing the data again*.

Since all applications use these indexes, MeTA supports out-of-core classification with some classifiers. We ran our large classification dataset that doesn't fit in memory—Webspam (Webb et al., 2006)—using the `sgd` classifier. Where LIBLINEAR failed to run, MeTA was able to finish the classification in a few minutes.

Besides using MeTA's rich built-in feature generation, it is possible to directly use LIBSVM-formatted data. This allows preprocessed datasets to be run under MeTA's algorithms. Additionally, MeTA's `forward_index` (used for classification), is easily convertible to LIBSVM format. The reverse is also true: you may do feature generation with MeTA, and use it to generate input for any other program that supports LIBSVM format.

MeTA is hosted publicly on GitHub[4], which provides the project with community involvement through its bug/issue tracker and fork/pull request model. Its API is heavily documented[5], allowing the creation of Web-based applications (listed in section 1). The project website contains several tutorials that cover the major aspects of the toolkit[6] to enable users to get started as fast as possible with little friction. Additionally, a public forum[7] is accessible for all users to view and participate in user support topics, community-written documentation, and developer discussions.

A major design point in MeTA is to allow for most of the functionality to be configured via a configuration file. This enables minimal effort exploratory data analysis without having to write (or recompile) any code. Designing the code in this way also encourages the components of the system to be pluggable: the entire indexing process, for example, consists of several modular layers which can be controlled by the configuration file.

An example snippet of a config file is given below; this creates a bigram part-of-speech analyzer. Multiple `[[analyzers]]` sections may be added, which MeTA automatically combines while processing input.

```
[[analyzers]]
method = "ngram-pos"
ngram = 2
filter = [{type = "icu-tokenizer"},
          {type = "ptb-normalizer"}]
crf-prefix = "crf/model/folder"
```

A simple class hierarchy allows users to add filters, analyzers, ranking functions, and classifiers with full integration to the toolkit (*e.g.* one may specify user-defined classes in the config file). The process for adding these is detailed in the MeTA online tutorials.

This low barrier of entry experiment setup ease led to MeTA's use in text mining and analysis MOOCs reaching over 40,000 students[8,9].

Multi-language support is hard to do correctly. Many toolkits sidestep this issue by only supporting ASCII text or the OS language; MeTA supports multiple (non-romance) languages by default, using the industry standard ICU library[10]. This allows MeTA to tokenize arbitrarily-encoded text in many languages.

Unit tests ensure that contributors are confident that their modifications do not break the toolkit. Unit tests are automatically run after each commit and pull request, so developers immediately know if there is an issue (of course, unit tests may be run manually before committing). The unit tests are run in a continuous integration setup where MeTA is compiled and run on Linux, Mac OS X[11], and Windows[12] under a variety of compilers and software development configurations.

3 Experiments

We evaluate MeTA's performance in NLP, IR, and ML tasks. All experiments were performed on a workstation with an Intel(R) Core(TM) i7-5820K CPU, 16 GB of RAM, and a 4 TB 5900 RPM disk.

3.1 Natural Language Processing

MeTA's part-of-speech taggers for English provide quite reasonable performance. It provides a linear-chain CRF tagger (CRF) as well as an averaged perceptron based greedy tagger (AP). We report the token level accuracy on sections 22–24 of the Penn Treebank, with a few prior model results trained on sections 0–18 in Table 3. "Human annotators" is an estimate based on a 3% error rate reported in the Penn Treebank README

[4]https://github.com/meta-toolkit/meta/
[5]https://meta-toolkit.org/doxygen/namespaces.html
[6]https://meta-toolkit.org/
[7]https://forum.meta-toolkit.org/
[8]https://www.coursera.org/course/textretrieval
[9]https://www.coursera.org/course/textanalytics
[10]http://site.icu-project.org/
[11]https://travis-ci.org/meta-toolkit/meta
[12]https://ci.appveyor.com/project/skystrife/meta

	CoreNLP			MeTA		
	Training	Testing	F_1	Training	Testing	F_1
Greedy	7m 27s 8.85 GB	18.6s 1.53 GB	86.7	17m 31s 0.79 GB	12.9s 0.29 GB	86.9
Beam (4)	6h 10m 43s 10.84 GB	46.8s 3.83 GB	89.9	2h 17m 25s 2.29 GB	59.2s 0.94 GB	88.1

Table 2: (NLP) Training/testing performance for the shift-reduce constituency parsers. All models were trained for 40 iterations on the standard training split of the Penn Treebank. Accuracy is reported as labeled F_1 from `evalb` on section 23.

	Extra Data	Accuracy
Human annotators		97.0%
CoreNLP	✓	97.3%
LTag-Spinal		97.3%
SCCN	✓	97.5%
MeTA (CRF)		97.0%
MeTA (AP)		96.9%

Table 3: (NLP) Part-of-speech tagging token-level accuracies. "Extra data" implies the use of large amounts of extra unlabeled data (*e.g.* for distributional similarity features).

	Indri	Lucene	MeTA
Blog06	55m 40s	20m 23s	11m 23s
Gov2	8h 13m 43s	1h 59m 42s	1h 12m 10s

Table 5: (IR) Indexing speed.

	Indri	Lucene	MeTA
Blog06	31.02 GB	2.06 GB	2.84 GB
Gov2	170.50 GB	11.02 GB	10.24 GB

Table 6: (IR) Index size.

	Docs	Size	$\|D\|_{avg}$	$\|V\|$
Blog06	3,215,171	26 GB	782.3	10,971,746
Gov2	25,205,179	147 GB	515.5	21,203,125

Table 4: (IR) The two TREC datasets used. Uncleaned versions of blog06 and gov2 were 89 GB and 426 GB respectively.

and is likely overly optimistic (Manning, 2011). CoreNLP's model is the result of Manning (2011), LTag-Spinal is from Shen et al. (2007), and SCCN is from Søgaard (2011). Both of MeTA's taggers are within 0.6% of the existing literature.

MeTA and CoreNLP both provide implementations of shift-reduce constituency parsers, following the framework of Zhu et al. (2013). These can be trained greedily or via beam search. We compared the parser implementations in MeTA and CoreNLP along two dimensions—speed, measured in wall time, and memory consumption, measured as maximum resident set size—for both training and testing a greedy and beam search parser (with a beam size of 4). Training was performed on the standard training split of sections 2–21 of the Penn Treebank, with section 22 used as a development set (only used by CoreNLP). Section 23 was held out for evaluation. The results are summarized in Table 2.

MeTA consistently uses less RAM than CoreNLP, both at training time and testing time. Its training time is slower than CoreNLP for the greedy parser, but less than half of CoreNLP's training time for the beam parser. MeTA's beam parser has worse labeled F_1 score, likely the result of its simpler model averaging strategy[13]. Overall, however, MeTA's shift-reduce parser is competitive and particularly lightweight.

3.2 Information Retrieval

MeTA's IR performance is compared with two well-known search engine toolkits: Lucene's latest version 5.5.0[14] and Indri's version 5.9 (Strohman et al., 2005)[15].

We use the TREC blog06 (Ounis et al., 2006) permalink documents and TREC gov2 corpus (Clarke et al., 2004). To ensure a more uniform indexing environment, all HTML is cleaned before indexing. In addition, each corpus is converted into a single file with one document per line to reduce the effects of many file operations.

During indexing, terms are lower-cased, stop words are removed from a common list of 431 stop words, Porter2 (MeTA) or Porter (Indri, Lucene) stemming is performed, a maximum word length of 32 characters is set, original documents are not stored in the index, and term position information is not stored[16].

We compare the following: indexing speed (Table 5), index size (Table 6), query speed (Table 7), and query accuracy (Table 8) with BM25 using $k_1 = 0.9$ and $b = 0.4$. We use the standard TREC queries associated with each dataset and

[13]At training time, both CoreNLP and MeTA perform model averaging, but MeTA computes the average over all updates and CoreNLP performs cross-validation over a default of the best 8 models on the development set.

[14]`http://lucene.apache.org/`

[15]Indri 5.10 does not provide source code packages and thus could not be used. It is also known as Lemur.

[16]For Indri, we are unable to disable positions information storage.

	Indri	Lucene	MeTA
Blog06	55.0s	1.60s	3.67s
Gov2	24m 6.73s	57.53s	1m 3.98s

Table 7: (IR) Query speed.

	Indri		Lucene		MeTA	
	MAP	P@10	MAP	P@10	MAP	P@10
Blog06	29.13	63.20	29.10	63.60	32.34	64.70
Gov2	25.96	53.69	30.23	59.26	29.97	57.43

Table 8: (IR) Query performance via Mean Average Precision and Precision at 10 documents.

score each system's search results with the usual `trec_eval` program[17].

MeTA leads in indexing speed, though we note that MeTA's default indexer is multithreaded and Lucene does not provide a parallel one[18]. MeTA creates the smallest index for gov2 while Lucene creates the smallest index for blog06; Indri greatly lags behind both. MeTA follows Lucene closely in retrieval speed, with Indri again lagging. As expected, query performance between the three systems is relatively even, and we attribute any small difference in MAP or precision to idiosyncrasies during tokenization.

3.3 Machine Learning

MeTA's ML performance is compared with liblinear (Fan et al., 2008), scikit-learn (Pedregosa et al., 2011), and SVMMulticlass[19]. We focus on linear classification with SVM across these tools (MALLET (McCallum, 2002) does not provide an SVM, so it is excluded from the comparisons). Statistics for the four ML datasets can be found in Table 9.

The 20news dataset (Lang, 1995)[20] is split into its standard 60% training and 40% testing sets by post date. The Blog dataset (Schler et al., 2006) is split into 80% training and 20% testing randomly. Both of these two textual datasets were preprocessed using MeTA using the same settings from the IR experiments.

The rcv1 dataset (Lewis et al., 2004) was processed into a training and testing set using the `prep_rcv1` tool provided with Leon Bottou's SGD tool[21]. The resulting training set has 781,265 documents and the testing set has 23,149. The Webspam corpus (Webb et al., 2006) consists of the subset of the Webb Spam Corpus used in the Pascal Large Scale Learning Challenge[22]. The corpus was processed using the provided `convert.py` into byte trigrams. The first 80% of the resulting file is used for training and the last 20% for testing.

	Docs	Size	k	Features
20news	18,846	86 MB	20	112,377
Blog	19,320	778 MB	3	548,812
rcv1	804,414	1.1 GB	2	47,152
Webspam	350,000	24 GB	2	16,609,143

Table 9: (ML) Datasets used for k-class categorization.

	liblinear	scikit	SVM$^{\text{mult}}$	MeTA
20news	79.4% 2.58s	74.3% 0.326s	67.1% 2.54s	80.1% 0.648s
Blog	75.8% 61.3s	76.2% 0.801s	72.2% 17.5s	72.2% 1.11s
rcv1	94.7% 17.6s	94.0% 1.66s	83.6% 2.01s	94.8% 3.44s
Webspam	✗	97.4% 11m 52s	✗	99.4% 1m 16s

Table 10: (ML) Accuracy and speed classification results. Reported time is to both train and test the model. For all except Webspam, this excludes IO.

In Table 10, we can see that MeTA performs well both in terms of speed and accuracy. Both liblinear and SVMMulticlass were unable to produce models on the Webspam dataset due to memory limitations and lack of a minibatch framework. For scikit-learn and MeTA, we broke the training data into 4 equal sized batches and ran one iteration of SGD per batch. The timing result includes the time to load each chunk into memory; for MeTA this is from its forward-index format[23] and for scikit-learn this is from libsvm-formatted text files.

4 Conclusion

MeTA is a valuable resource for text mining applications; it is a viable and competitive alternative to existing toolkits that unifies algorithms from NLP, IR, and ML. MeTA is an extensible, consistent framework that enables quick development of complex application systems.

Acknowledgements

This material is based upon work supported by the NSF GRFP under Grant Number DGE-1144245.

[17]`http://trec.nist.gov/trec_eval/`

[18]Additionally, we did not feel that writing a correct and threadsafe indexer as a user is something to be reasonably expected.

[19]`http://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html`

[20]`http://qwone.com/~jason/20Newsgroups/`

[21]`http://leon.bottou.org/projects/sgd`

[22]`ftp://largescale.ml.tu-berlin.de/largescale/`

[23]It took 12m 24s to generate the index.

References

Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. 2009. On Smoothing and Inference for Topic Models. In *UAI*.

Charles L. A. Clarke, Nick Craswell, and Ian Soboroff. 2004. Overview of the TREC 2004 Terabyte Track. In *TREC*.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *EMNLP*.

David Easley and Jon Kleinberg. 2010. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA.

R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *JMLR* pages 1871–1874.

J. Foulds, L. Boyles, C. DuBois, P. Smyth, and M. Welling. 2013. Stochastic Collapsed Variational Bayesian Inference for Latent Dirichlet Allocation. In *KDD*.

T. L. Griffiths and M. Steyvers. 2004. Finding Scientific Topics. *PNAS* 101.

Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *ICML*.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *JMLR* 5.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proc. CICLing*.

Sean Massung and ChengXiang Zhai. 2015. SyntacticDiff: Operator-based Transformation for Comparative Text Mining. In *IEEE International Conference on Big Data*.

Sean Massung, ChengXiang Zhai, and Julia Hockenmaier. 2013. Structural Parse Tree Features for Text Representation. In *Proc. IEEE ICSC*.

Andrew Kachites McCallum. 2002. MALLET: A Machine Learning for Language Toolkit. `http://mallet.cs.umass.edu/`.

David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. 2009. Distributed Algorithms for Topic Models. *JMLR* 10.

I. Ounis, C. Macdonald, M. de Rijke, G. Mishne, and I. Soboroff. 2006. Overview of the TREC 2006 Blog Track. In *TREC*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *JMLR* 12.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP*.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *TREC*.

Stéphane Ross, Paul Mineiro, and John Langford. 2013. Normalized online learning. In *UAI*.

Jonathan Schler, Moshe Koppel, Shlomo Argamon, and James W. Pennebaker. 2006. Effects of Age and Gender on Blogging. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *ACL*.

Anders Søgaard. 2011. Semi-supervised condensed nearest neighbor for part-of-speech tagging. In *ACL-HLT*.

Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. 2005. Indri: A language-model based search engine for complex queries (extended version). IR 407, University of Massachusetts.

Charles Sutton and Andrew McCallum. 2012. An Introduction to Conditional Random Fields. In *Foundations and Trends in Machine Learning*.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *ACLIJCNLP*.

Steve Webb, James Caverlee, and Carlton Pu. 2006. Introducing the webb spam corpus: Using email spam to identify web spam automatically. In *CEAS*.

ChengXiang Zhai and John Lafferty. 2004. A Study of Smoothing Methods for Language Models Applied to Information Retrieval. *ACM Trans. Inf. Syst.* 22(2).

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *ACL*.