MeTA: A Unified Toolkit for Text Retrieval and Analysis

Sean Massung, Chase Geigle, and ChengXiang Zhai
December 5th, 2017
Motivation
Document Classification
Step 1

SvmLight\textsuperscript{1}

\textsuperscript{1}svmlight.joachims.org/
Step 2

SvmPerf$^2$

$^2$www.cs.cornell.edu/people/tj/svm_light/svm_perf.html
Step 3

SvmMulticlass\textsuperscript{3}

\textsuperscript{3}http://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html
Data munging
Step 5

Feature generation
Step 6

Handling unicode
Out of memory
Incapacitated server, different algorithm?
Despair???
MeTA!
To the Rescue

MeTA\textsuperscript{4} has been created to solve these text miner’s headaches once and for all by

1. uniting machine learning and information retrieval algorithms under one roof (no more tool hunting, separate compiling and configuration, etc.),
   • indexing, ranking, searching, topic modeling, classification, parallelization, \textit{n}-gram language modeling, sequence labeling, parsing, word embeddings, graph mining, ...
2. integrating \textbf{all steps of text tokenization} into one framework,
3. respecting the \textbf{Unicode} standards,
4. supporting out-of-the-box out-of-core classification,
5. being competitive with existing, highly specific tools, and
6. \textbf{letting you write as little code as possible} (often none at all)

\textsuperscript{4}https://meta-toolkit.org
Components of MeTA
MeTA can be broken down into the following major components (bold I’ll talk about today):

- **Indexes** (data storage)
- **Analyzers and Filters** (text tokenization and processing)
- **Search** engine (on top of inverted_index)
- **Classification** (on top of forward_index)
- Regression (on top of forward_index)
- **Topic modeling** (on top of forward_index)
- **Sequence labeling**
- **Syntactic parsing**
- Word embeddings
- Graph mining
Feature Generation
Tokenizers convert documents to token streams;
Filters add, remove, or modify tokens;
and **Analyzers** count.
Example

icu-tokenizer → lowercase → length(2, 35) → unigrams

[[analyzers]]
method = "ngram-word"
ngram = 1
    [[analyzers.filter]]
type = "icu-tokenizer"

[[analyzers.filter]]
type = "lowercase"

[[analyzers.filter]]
type = "length"
min = 2
max = 35
Porter2 English Stemmer

[[analyzers.filter]]
type = "porter2-filter"

{waits, waited, waiting} → wait

{consist, consisted, consistency, consists} → consist

{run, runs, running} → run

https://github.com/smassung/porter2_stemmer
Feature Combining

$(\text{default-chain} \rightarrow \text{bigrams}) \cup \text{POS-tags}$

```python
[[analyzers]]
method = "ngram-word"
ngram = 2
filter = "default-chain"

[[analyzers]]
method = "ngram-pos"
ngram = 1
filter = [{type = "icu-tokenizer"}, {type = "ptb-normalizer"}]
crf-prefix = "path/to/model/files"
```
Features from Parse Trees

Example parse tree of a sentence

Rewrite rule features
Structural Features from Parse Trees

Diagram of parse trees with 'X' nodes.
Indexing Process

corpus → documents → (analyzer: tokenizer → filters) → indexer

Corpus variants:

• line_corpus: one document per line
• gz_corpus: one document per line, gzip compressed
• file_corpus: one document per file
Three criteria:

- Indexing speed
- Index size
- Query speed

Table 1: (IR) The two TREC datasets used. Uncleaned versions of blog06 and gov2 were 89 GB and 426 GB respectively.
## Indexing Speed and Size

<table>
<thead>
<tr>
<th></th>
<th>Indri</th>
<th>Lucene</th>
<th>MeTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blog06</td>
<td>55m 40s</td>
<td>20m 23s</td>
<td>11m 23s</td>
</tr>
<tr>
<td>Gov2</td>
<td>8h 13m 43s</td>
<td>1h 59m 42s</td>
<td>1h 12m 10s</td>
</tr>
</tbody>
</table>

Table 2: (IR) Indexing speed.

<table>
<thead>
<tr>
<th></th>
<th>Indri</th>
<th>Lucene</th>
<th>MeTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blog06</td>
<td>31.02 GB</td>
<td>2.06 GB</td>
<td>2.84 GB</td>
</tr>
<tr>
<td>Gov2</td>
<td>170.50 GB</td>
<td>11.02 GB</td>
<td>10.24 GB</td>
</tr>
</tbody>
</table>

Table 3: (IR) Index size.
Query Speed and Ranking Performance

<table>
<thead>
<tr>
<th></th>
<th>Indri</th>
<th>Lucene</th>
<th>MeTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blog06</td>
<td>55.0s</td>
<td>1.60s</td>
<td>3.67s</td>
</tr>
<tr>
<td>Gov2</td>
<td>24m 6.73s</td>
<td>57.53s</td>
<td>1m 3.98s</td>
</tr>
</tbody>
</table>

Table 4: (IR) Query speed.

<table>
<thead>
<tr>
<th></th>
<th>Indri</th>
<th>Lucene</th>
<th>MeTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blog06</td>
<td>MAP 29.13</td>
<td>P@10 63.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P@10 63.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gov2</td>
<td>MAP 25.96</td>
<td>P@10 53.69</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P@10 30.23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: (IR) Query performance via Mean Average Precision and Precision at 10 documents.
• Configurable, state-of-the-art existing rankers

    [ranker]
    method = "bm25"
    k1 = 1.2 # optional
    b = 0.75 # optional
    k3 = 500 # optional

    • Absolute discounting
    • Dirichlet prior
    • Jelinek-Mercer
    • Okapi BM25(L)
    • Pivoted length normalization
    • Rocchio pseudo-relevance feedback
    • KL-Divergence pseudo-relevance feedback

• Easy to add new rankers (see score_data)
Other Search Components

• **Caching**: configurable caches for improving retrieval speed (allows optimization of memory utilization)

• **Parallelization**: indexing will utilize all machine cores during tokenization, most during index merging

• **Memory-mapped I/O**: efficient access to all index files is performed via using OS-level memory mapping
DEMO
Classification
Classification Methods

Binary classifiers:

• SGD trained:
  • SVM (hinge, smooth hinge, squared hinge)
  • Perceptron
  • Logistic regression

Multiclass adapters:

• One-vs-all
• One-vs-one

Multiclass classifiers:

• Wrapper for LIBSVM/LIBLINEAR
• Naïve Bayes
• Winnow
• Dual perceptron (kernels)
• kNN (using the search engine)
Out-of-core Support

An experiment on rcv1 with only **200MB** of memory:

LIBLINEAR crashes; MeTA works just fine!\(^6\)

---

\(^6\)https://meta-toolkit.org/online-learning.html
...is it good?

<table>
<thead>
<tr>
<th></th>
<th>Docs</th>
<th>Size</th>
<th>Classes</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>20news</td>
<td>18,846</td>
<td>86 MB</td>
<td>20</td>
<td>112,377</td>
</tr>
<tr>
<td>Blog</td>
<td>19,320</td>
<td>778 MB</td>
<td>3</td>
<td>548,812</td>
</tr>
<tr>
<td>rcv1</td>
<td>804,414</td>
<td>1.1 GB</td>
<td>2</td>
<td>47,152</td>
</tr>
<tr>
<td>Webspam</td>
<td>350,000</td>
<td>24 GB</td>
<td>2</td>
<td>16,609,143</td>
</tr>
</tbody>
</table>

**Table 6:** (ML) Datasets used.
Table 7: (ML) Accuracy and speed classification results. Reported time is to both train and test the model. For all except Webspam, this excludes IO.
DEMO
Topic Modeling
Currently has many flavors of LDA inference:

- Collapsed Gibbs sampling
- Parallel collapsed Gibbs sampling (approximate)
- Collapsed Variational Bayes (0-th order approximation)
- Stochastic Collapsed Variational Bayes (0-th order approximation)
Sequence Labeling
Sequence Labeling

General task on sequence data in NLP, useful for

• Part of speech (POS) tagging
• Chunking/shallow parsing
• Named-entity recognition (NER)
• Information extraction

Supported in MeTA via an implementation of linear-chain conditional random fields (CRFs) or a greedy averaged perceptron.
Sequence Labeling Framework

sequences $\rightarrow$ sequence_analyzer $\rightarrow$ model

Sequence analyzer creates observation features for each element:

- Lexical features: $w_t = v_i$, $w_t$ is capitalized, $w_t$ is a number, ...
- Context features: $w_{t+1} = v_j$, $w_{t+2} = v_k$, $w_{t-1} = v_\ell$, $w_{t-2} = v_m$, ...

User-configurable: specify a function that takes an observation and generates features (represented as strings)

(Ships with a default analyzer for POS tagging)
CRF Results for POS Tagging

<table>
<thead>
<tr>
<th></th>
<th>Sentences</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>42,685</td>
<td>969,905</td>
</tr>
<tr>
<td>Development</td>
<td>6,526</td>
<td>148,158</td>
</tr>
<tr>
<td>Testing</td>
<td>7,505</td>
<td>171,138</td>
</tr>
</tbody>
</table>

Table 8: (NLP) The Penn Treebank dataset.

<table>
<thead>
<tr>
<th></th>
<th>Extra Data</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human annotators</td>
<td>✓</td>
<td>97.0%</td>
</tr>
<tr>
<td>CoreNLP</td>
<td>✓</td>
<td>97.3%</td>
</tr>
<tr>
<td>LTag-Spinal</td>
<td></td>
<td>97.3%</td>
</tr>
<tr>
<td>SCCN</td>
<td>✓</td>
<td>97.5%</td>
</tr>
<tr>
<td>MeTA (CRF)</td>
<td></td>
<td>97.0%</td>
</tr>
<tr>
<td>MeTA (AP)</td>
<td></td>
<td>96.9%</td>
</tr>
</tbody>
</table>

Table 9: (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).
Syntactic Parsing
Shift-reduce Constituency Parser

A **linear-time** algorithm for producing parse trees from input sequences (POS-tagged words)

Around **34 times** faster than similar alternatives\(^7\)!

\(^7\)nlp.stanford.edu/software/srparser.shtm
### Table 10: (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.

<table>
<thead>
<tr>
<th></th>
<th>CoreNLP</th>
<th></th>
<th>$F_1$</th>
<th>MeTA</th>
<th></th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
<td></td>
<td>Training</td>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td>Greedy</td>
<td>7m 27s</td>
<td>18.6s</td>
<td>86.7</td>
<td>17m 31s</td>
<td>12.9s</td>
<td>86.9</td>
</tr>
<tr>
<td></td>
<td>8.85 GB</td>
<td>1.53 GB</td>
<td></td>
<td>0.79 GB</td>
<td>0.29 GB</td>
<td></td>
</tr>
<tr>
<td>Beam (4)</td>
<td>6h 10m 43s</td>
<td>46.8s</td>
<td>89.9</td>
<td>2h 17m 25s</td>
<td>59.2s</td>
<td>88.1</td>
</tr>
<tr>
<td></td>
<td>10.84 GB</td>
<td>3.83 GB</td>
<td></td>
<td>2.29 GB</td>
<td>0.94 GB</td>
<td></td>
</tr>
</tbody>
</table>
DEMO
Conclusion
https://meta-toolkit.org

MIT & UIUC/NCSA Licensed
(Liberal; use it at work!)

Questions?