Appendix C

User Manual

In this appendix, we describe how to use the ME toolkit to train ME models and how to compute conditional probabilities given an ME model. It is worth mentioning here that this toolkit is not a panacea for all ME modeling problems. However, it can solve many problems; especially, those in natural language processing, such as language modeling, part-of-speech tagging and syntactic parsing.

We expect that the readers of this manual either know the maximum entropy principle, or have read this dissertation.

C.1 Overview

In this section, we provide the availability, system requirements and the installation steps of our ME toolkit. We also briefly describe the functionality provided by the ME toolkit.

C.1.1 Availability

ME toolkit is available from

http://www.clsp.jhu.edu/junwu/METK/Release-0.2.tar.

Readers may contact the author by email

junwu@clsp.jhu.edu.
C.1.2 Functionality

The ME toolkit supports the estimation and the prediction for maximum entropy models in discrete domains. The toolkit can be used to find the maximum entropy distribution for conditional probability \( p(y|x) \), and to compute this probability using ME models. The current version of the toolkit can handle MN-gram (or similar) models, in which the condition \( x \) has less than or equal to \( M \) types of constraints and none of them are more complicated than N-gram constraints.

C.1.3 System Requirements

The toolkit will be run on the Unix/Linux systems. They have been tested on Sun Solaris and Linux for Intel CPUs.

The memory and disk space requirements are dependent on the size of the training data (or the number of constraints in the ME model). Today, the disk space is not a problem in building most ME models. The following table shows the approximated memory required for training trigram models in different tasks. Training complicated models needs much more memory than training trigram models. For example, training the syntactic models described in Chapter 5 needs three- to ten-fold more memory than for training the trigram models on the same task.

<table>
<thead>
<tr>
<th>Task (training size)</th>
<th>Memory Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switchboard (&lt;3M words)</td>
<td>&lt; 128M</td>
</tr>
<tr>
<td>WSJ/BN Subset (~10M words)</td>
<td>&lt; 512M</td>
</tr>
<tr>
<td>BN Whole set (~100M words)</td>
<td>1.5G</td>
</tr>
</tbody>
</table>

Table C.1: Memory requirement for ME models.

C.1.4 Installation

After downloading the tar file to the local directory selected for installation, the user needs to expand the tar file and create training and evaluation programs by

\[
\text{tar} \ -xvf \ Release-0.2.tar
\]
The user must then change to `Release-0.2/src/3gram`, `(Release-0.2/src/flat`, or `Release-0.2/src/composite`) and build executable files by

```make
make -f trainX.make, and
make -f testX.make
```

where `X` can be `3gram`, `composite` or `flat` for trigram models, composite models and flat models (i.e., M2-gram models described in Section 3.4), respectively. Then, move executable files

```plaintext
train_3gram
test_3gram
train_flat
test_flat
train_composite
test_composite
```

to the `Release-0.2/bin` directory.

### C.1.5  Tree Structure of Directories

The tree structure of directories is shown in Figure C.1.

![Directory Tree](image)

**Figure C.1: Directory Tree.**

Table C.2 shows the contents in each directory. In some directories, there is a `ReadMe` file containing update instructions and the content of the corresponding directories.
bin/ all executables.
data/ data (in the ME toolkit formats) used in training and evaluating models, such as N-gram files, index files and model parameter files.
doc/ this manual and release notes, etc.
sample_data/ sample data files that users can use to train some small ME models, and the answer files against which users can check their results.
src/ source code files, including c++, header and make files.
work/ temperate files, such as initial models and files for debugging purposes.

Table C.2: Content of Directories.

C.1.6 Executables

We build three groups of training and evaluation programs for users:

1. train3gram and test3gram
2. train_composite and test_composite, and
3. train_flat and test_flat.

They are used to train and evaluate regular topic-(in)dependent trigram models, syntactic/composite models with two types of syntactic features (and topic-dependent features) and flat models with no nested features for small vocabularies. To train a topic-dependent model, the argument file must have the name of a valid topic-dependent unigram file. Further, the training data also need to be split according to topics, and training tuples (with their histories) are gathered from different topics.

The command line for executing a program is

eXecutable_file argument_file.

The argument file defines the model name, N-gram file names, training and test history-future \( \langle x, w_i \rangle \) tuple file names, etc.
C.2 Training an ME Model

There are three steps in training an ME model: data preparation, including feature selection; computation of feature expectations and update of model parameters. The first step is executed only once, whereas the last two steps need several iterations.

C.2.1 Data Preparation and Feature Selection

The sample data (e.g., sentences, articles, etc.) need to be converted to the format that the toolkit can accept. Since the sources of samples vary tremendously, we cannot build a universal data preprocessing tool for all applications. Therefore, in the ME toolkit we provide the data formats to which users can convert their own data. For applications in natural language processing, this transformation can be done using Perl scripts. File formats will be introduced in Section C.4.1.

All features look like N-grams, whose format will be described in following sections. Users can select any feature that is observed\(^1\) in the training data. Target expectations are usually empirical estimates of features after some smoothing, e.g., Good-Turing discounting.

C.2.2 Computing Feature Expectations

An ME model is first initialized as a uniform model, i.e., all model parameters \(\alpha = 1\). The initialization is done by running

```bash
trainX argument-file -init
```

The initial uniform model must be renamed `old_para`, which is a reserved word in the toolkit. After the initial model is created, users can run

```bash
trainX argument-file
```

to obtain the feature expectation file, whose name is defined in the argument file.

If the computation of feature expectations is distributed among many machines, each machine must have an individual argument file indicating which parts of the training data are used in this machine.

\(^1\)Otherwise, that feature cannot be trained.
C.2.3 Updating Parameters

Since the computation of feature expectations may be distributed among several machines, several files of partial expectations may be generated. The following command is used to merge these files and also update model parameters.

\texttt{UpdateAlpha old\_parameters Ei\_0 \ldots Ei\_k new\_parameters}

The executable \texttt{UpdateAlpha} is obtained by running

\texttt{g++ -o UpdateAlpha UpdateAlpha.cc}

in the directory \texttt{src/}.

It should be noted here that steps C.2.2 and C.2.3 need to be executed for many iterations to train an ME model. Training programs will display the divergence between model expectations and target expectations of features. When this divergence is very small, the model is converged.

C.3 Computing Probabilities Using ME Models (Evaluation)

The ME toolkit supports computation of the log-probability for a future token given the history using the ME model generated. The test data need to be converted to the format of the toolkit. This procedure is similar to the training. An argument file, whose format will be introduced later, needs to be generated to specify the setup of evaluation. Log-probabilities can be computed by running

\texttt{testX test-arguments}

The log-probability is created in an ASCII file, whose name is defined in the argument file.

C.4 File Formats

In this section, we describe the file formats to follow when using the ME toolkit. We also show some real examples of these files.
C.4.1 Model Parameter File

We use the parameter format described in Ristad (1997) to represent model parameter files. If the model has $V$ tokens, $M$ marginal features (including topic-dependent features) and $C$ conditional ones, then the model looks like

\begin{verbatim}
begin.constraints V C+M
begin.marginal M

1 alpha_1 a_1
... 
M alpha_M a_M
end.marginal
begin.conditional M

M+1 alpha_{M+1} a_{M+1}
... 
M+C alpha_{M+C} a_{M+C}
end.conditional
end.constraints
\end{verbatim}

where alpha’s are model parameters and a’s are the corresponding target expectations.

The model is initialized as a uniform one, with the initial values of alpha’s equal to one. The target expectations should be the counts (after some discounting) of features over the number of training samples. For example, the expectation of a bigram feature $g(w_1, w_2)$ can be $\frac{\#[w_1, w_2]}{\#\text{[training data]}}$.

C.4.2 N-gram File

All N-gram files have the common properties of

- one N-gram per line,
- sorted by N-grams in the increasing order, and
- no lines with zero counts.
Unigram File

A unigram file contains \( \langle w_i, c(w_i) \rangle \) pairs, where \( c(w_i) \) is the count of word \( w_i \). A part of a unigram file may look like

\[
\begin{array}{ll}
20 & 37 \\
21 & 40 \\
24 & 1 \\
\end{array}
\]

Bigram File

A bigram file contains a list of \( \langle w_i, w_j, c(w_i, w_j) \rangle \) triples. An example of bigram files may look like

\[
\begin{array}{llll}
13 & 41 & 20 \\
13 & 934 & 3 \\
14 & 2 & 10 \\
\end{array}
\]

Trigram File

A trigram file contains a list of \( \langle w_i, w_j, w_k, c(w_i, w_j, w_k) \rangle \) 4-tuples. It may look like

\[
\begin{array}{lllll}
10 & 25 & 342 & 7 \\
10 & 34 & 1 & 2 \\
11 & 7 & 43 & 4 \\
\end{array}
\]

Topic-dependent Unigram File

A topic-dependent unigram file contains a list of \( \langle index, t, w_i, c(t, w_i) \rangle \), where \( t \) is a topic and \( index \) is the index of the topic feature \( g(t, w_i) \). This file must be sorted by \( t \) and \( w_i \). It may look like

\[
\begin{array}{llll}
10 & 3 & 34 & 7 \\
\end{array}
\]
C.4.3 Discounting File

As we have already mentioned, target expectations of features should be smoothed from their empirical counts. To make the smoothing procedure easy, we assume that the target expectations of N-grams (from the same information source) with the same counts should be the same after discounting. The training programs will read the discounting file that indicates the value d for count c after discounting. Of course, users may set their own target expectations directly in their parameter files, as we will introduce in Section C.5.3. The discounting file has the format of

M T
0 d1,1 d1,2 ... d1,T
...
0 dM,1 dM,2 ... dM,T

where M is the number of feature types, T is the maximum threshold for discounting, and di,j is the discounted value of the count j for the ith kind of feature. An example of the discount file for trigram models is shown below.

3 5
0 0.7 1.7 2.7 3.7 4.7
0 0.5 1.5 2.6 3.6 4.7
0 0.3 1.4 2.5 3.6 4.7

C.4.4 History Equivalence Class File (By Information Source)

As we have described in preceding sections, the constraints of an ME model may come from M sources. We create history equivalence classes according to each information source. For example, in the syntactic model, we have three information sources: head-words, non-terminal labels and two preceding words. The corresponding history equivalence classes are pairs of hi−2, hi−1, nt−2, nt−1, wi−2, wi−1, etc. The history equivalence class file contains tokens in the history equivalence class and the
history equivalence class index. For example, the head word history equivalence class
file contains tuples of \( \langle h_{i-2}, h_{i-1}, \text{index} \rangle \). A real file looks like:

\[...
2 3 42
2 6 43
2 40 44
...
\]

### C.4.5 History File

Training data can be split into parts so that the model can be trained on different
machines in parallel. Each part of training data is represented by a history file that
will be introduced in this section, and a history-future tuple file that will be introduced
in the following section. It is worth noting here that all files introduced in preceding
sections are universal for the whole training data, whereas files introduced in this
section and following sections are related to the parts of training data.

The history file contains history equivalence class indices (of different information
sources) and its own index. This file is slightly different for trigram models, flat
models and syntactic models. For trigram models, the history file has only two
columns, which indicate the history equivalence class index for bigram \( w_{i-2}, w_{i-1} \) and
the history index, respectively. For syntactic models, the file has four columns, which
look like

\[...
102 422 13 59
103 423 12 60
103 425 14 61
...
\]

The first three columns are the indices for history equivalence classes \( w_{i-2}, w_{i-1},
h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1} \), respectively, and the last one is the index of the history itself.
Figure C.2 illustrates the relation between the history file and history equivalence
class files.

The history file of a flat model (3.28) contains history tuples \( u_1, u_2, \cdots, u_M \) and
their indices \( \text{idx}(u_1, u_2, \cdots, u_M) \), one for each line.
C.4.6 Training Tuple File

The tuple file looks like a bigram file. It contains tuples of \( \langle x, w_i, c(x, w_i) \rangle \). A tuple file may look like

\[
\ldots
59 \quad 93 \quad 7 \\
59 \quad 323 \quad 4 \\
60 \quad 26 \quad 1 \\
\ldots
\]

C.4.7 History Equivalence Class File (By Order)

To train ME models (e.g., syntactic model) hierarchically, users need to create history equivalence classes \( w_{i-1}, h_{i-1}, nt_{i-1} \) and words following them in the training
data. The history equivalence class file (by order) is similar to the history file; it contains four columns, which indicate \( w_{i-1}, h_{i-1}, nt_{i-1} \) and history equivalence class index, respectively. A real file for the syntactic/composite model may look like

\[
\begin{array}{cccc}
9 & 13 & 4 & 101 \\
9 & 20 & 7 & 102 \\
10 & 6 & 1 & 103 \\
\ldots
\end{array}
\]

C.4.8 Tuple File for History Equivalence Classes and Futures

This file contains tuples of history equivalence class (by order) index, future word and their count, whose format is similar to that of the training tuple file.

History and tuple files must be stored in one directory whose name is set in the argument file. The files must be named as follows:

- `history.part0 - history.partK`, and
- `tuple.part0 - tuple.partK`

if the training data has \( K + 1 \) parts. Similarly, history equivalence class files and corresponding tuple files must be named

- `historyclass.part0 - historyclass.partK`, and
- `tupleclass.part0 - tupleclass.partK`.

C.4.9 Test Tuple File

In the test tuple file, the first line indicates the number of test tuples. This number must be equal to or smaller than the real number of test tuples. If this number is less than the real size, then the evaluation program will read tuples up to this number. A test tuple file looks like

\[
\begin{array}{cccc}
\text{number_of_tuples} \\
0 & w_i & w_{i-2} & w_{i-1} & 1
\end{array}
\]
... for trigram models, and it looks like

\[
\text{number_of_tuples} = \sum_{k=1}^{k} \rho(nt_{i-2}^{k}, nt_{i-1}^{k}, h_{i-2}^{k}, h_{i-1}^{k}) \quad \text{for syntactic models, where } k \text{ is the number of candidate parses for } w_i, \text{ and } \rho \text{ is the likelihood of candidate parses.}
\]

C.4.10 Argument File

The argument files set environments for training and test programs, and for indicating the input and output.

Training Argument File

Here is a sample training argument file for trigram models.

```
10000 vocab_size
./data/train.gt  good_turing_discount_file
./data/train.his  N-gram_history_set
./data/train.1gram unigram_file
./data/train.2gram bigram_file
1  bigram_cutoff
./data/train.3gram trigram_file
2  trigram_cutoff
./data/topic_unigram  topic-files
./data  history-tuple_directory
0  starting_part_of_training_data
9  ending_part_of_training_data
./data/feature.expect  expectation_file
./data/model  new_model
10  number_of_iterations
```
Each line must have exactly two columns, even though the second one is simply for comments. If the second raw, which is for Good-Turing discount file, gives a non-existent file name, then the default discounting values will be used. If `topic_unigram_file` is valid, then the model is a topic-dependent model; otherwise, it is a topic-independent model.

In the example above, the initial model is saved in `./data/model`. As already explained, it should be renamed `old_para` before the first iteration of training.

Users can find more sample argument files (with comments for each line)

```plaintext
train Composite argu, and
train flat argu
```

in the `Release-0.2/sampledata/` directory.

**Test Argument File**

Here is a sample test argument file for trigram models.

```plaintext
60860  vocabulary size
13  id of the unknown token
  ./data/train.hist history name
  ./data/train.1gram unigram file
  ./data/train.2gram bigram file
  1 bi_cutoff
  ./data/train.3gram trigram file
  2 tri_cutoff
  ./data/topic_unigram topic unigram file
bn dev-test96.tuple testtuple
  ./data/model ME_model
bn dev-test96.ppl -logprob_for_test_tuples
5 topic id for test data
```

The last line, `topid id for test data`, is optional and can be omitted for regular (topic-independent) trigram models. Users can find more sample test argument files

```plaintext
test Composite argu, and
test flat argu
```

in the `Release-0.2/sampledata/` directory.
C.5 Advanced Topics

In this section, we describe for advanced users some hacks using the ME toolkit.

C.5.1 Reducing the Number of Iterations

N-gram models need only a few iterations to converge, and each iteration also takes a very short time. However, models with non-nested features, such as the syntactic model, need many iterations to converge, and each one takes a very long time. An efficient way of reducing the number of iterations for models with non-nested features is to train the N-gram model, the head-word N-gram model, etc., first, and then use the parameters of these models as the initial values of the target model with non-nested features. For example, the trigram models need about 20 iterations to converge, and the syntactic models need even more iterations if the process is started from the uniform model. However, the syntactic models can be converged in about 5 iterations if the process is started from the model whose parameters are set by the trained N-gram model, head-word N-gram model and non-terminal label N-gram model.

C.5.2 Add or Drop features

To add or drop features, users need to edit N-gram files, set correct target expectations and then run

```
train3gram train3gram.argu init
```

to create a new parameter file.

Users should NOT add or drop features directly from the parameter file.

C.5.3 Using Other Smoothing Methods

Users can change the last column of the initial model and set target expectations using their own smoothing methods. However, it should be mentioned here that the
feature expectations should be consistent. Otherwise, the convergence rate may be very slow.

C.5.4 High Order Features

The current ME toolkit does not directly support features of order 4 or above. However, the toolkit can be used to train and evaluate models with some kinds of order 4 features. For example, to train a model with features \(g(nt_{i-3}, nt_{i-2}, nt_{i-1}, nt_{i})\) and \(g(nt_{i-2}, nt_{i-1}, nt_{i})\), we can define a super non-terminal label set

\[ NT^2 = NT \times NT + NT \]

where \(NT\) is the original non-terminal label set. Then both \(g(nt_{i-3}, nt_{i-2}, nt_{i-1}, nt_{i})\) and \(g(nt_{i-2}, nt_{i-1}, nt_{i})\) are order 3 features defined on \(NT^2\). It should be noted that \(g(nt_{i-2}, nt_{i-1}, nt_{i})\) now applies to We reduce the order of features by this means. However, this trick does not apply if the size of the token set is large, e.g., a vocabulary.

C.6 Exercises

We provide some sample training and test data from *Jane Eyre* for users to learn how to use our toolkit. We recommend three exercises and provide answers for these exercises. All data are stored in directory `sampledata/`.

C.6.1 Building a Trigram Model

To build a trigram model, users can use the training data in `je.training`. Then, users can evaluate the ME model by computing the trigram probabilities of the test words in `je.test`.

The sample model (of 20 iterations) is `je.model.3gram`; the log probability file is `je.ppl.3gram`. 


C.6.2 Building a Flat Model

Here we explain how to train an ME model to predict consonant / vowel of the next character (without looking at the character) using 5 preceding characters independently. The model looks like

\[
p(y_i | c_{i-5}, \ldots, c_{i-1}) = \frac{\alpha_{g(y_i)} \cdot \alpha_{g(c_{i-5}, y_i)} \cdots \alpha_{g(c_{i-1}, y_i)}}{z(c_{i-5}, \ldots, c_{i-1})}
\]

where \( y_i \) is either \( c \) or \( v \). Users can use the same training and test data above. Both the sample model \texttt{je.model.cv} and the results \texttt{je.cv.results} are saved in the directory \texttt{sampledata/flat/}. There is a \texttt{ReadMe} file indicating how the model and the results are generated.

C.6.3 Building a Skipped N-gram Model

Users can build and evaluate the skipped N-gram model

\[
p(w_i | w_{i-3}, w_{i-2}, w_{i-1}) = \frac{1}{z} \alpha_{g(w_i)} \cdot \alpha_{g(w_{i-1}, w_{i})} \cdot \alpha_{g(w_{i-2}, w_{i-1}, w_{i})} \cdot \alpha_{g(w_{i-2}, w_{i})} \cdot \alpha_{g(w_{i-3}, w_{i-2}, w_{i})}
\]

using the same training and test data above. The sample model is \texttt{je.model.skipped} and the results are saved in \texttt{je.ppl.skipped}. More instructions can be found in the \texttt{ReadMe} file in the directory \texttt{sampledata/composite/}.

C.7 Troubleshooting

Most of the problems that users encounter in using the ME toolkit is related to invalid files. Here is the checklist for troubleshooting:

- Are files listed in the correct order in the argument file?
- Are files in correct formats?
- Are data in files sorted in the correct way?
- Are data files compatible? (Do indices agree with each other?)
• Contact junwu@clsp.jhu.edu

C.8 Update

Our toolkit is far from perfect right now. Users can check the update information for these toolkit from

http://www.clsp.jhu.edu/junwu/METK.html.