MEAD Documentation
v3.08

Dragomir Radev, University of Michigan
John Blitzer, University of Pennsylvania
Adam Winkel, University of Michigan
Tim Allison, University of Michigan
Michael Topper, University of Michigan

With contributions by:

Arda Çelebi, USC/ISI
Wai Lam, Chinese University of Hong Kong
Hong Qi, University of Michigan
Horacio Saggion, University of Sheffield
Simone Teufel, University of Cambridge
Zhu Zhang, University of Michigan

`http://www.summarization.com/mead/`

October 14, 2003

Contents

1 Introduction **5**
1.1 What is automatic text summarization? . . . 5
1.2 Sentence Extraction . . . 5
1.3 The MEAD System . . . 5
1.4 MEAD Functionality . . . 5
1.5 Sample Use Cases . . . 6

2 Downloading and Installation **6**
2.1 Downloading the MEAD Distribution . . . 6
2.2 External Software . . . 6
2.3 Installing MEAD . . . 7

3 Simple MEAD tutorial **8**
3.1 An Example Cluster: GA3 . . . 8
3.2 Summarizing the GA3 cluster . . . 9
3.3 So what does mead.pl really do? . . . 10
3.4 How mead.pl locates clusters . . . 10
3.5 Command-line options to mead.pl . . . 11

4 Advanced MEAD **13**
4.1 Specifying summaries using mead-config files . . . 13
4.2 Using .meadrc files to specify defaults . . . 14

5 MEAD XML Objects **15**
5.1 Common Objects . . . 15
5.1.1 mead-config . . . 15
5.1.2 Cluster . . . 17
5.1.3 Docsent . . . 17
5.1.4 Extract . . . 18
5.1.5 Summary . . . 18
5.2 Less Common MEAD Objects . . . 18
5.2.1 SentFeature . . . 19
5.2.2 SentJudge . . . 19
5.2.3 Query . . . 19
5.3 XML Objects not used by core MEAD . . . 19
5.3.1 Document . . . 19
5.3.2 Docpos . . . 19
5.3.3 Docjudge . . . 19
5.3.4 Sentalign . . . 21
5.3.5 Sentrel . . . 21

6 MEAD Interfaces **21**
6.1 mead.pl . . . 21
6.2 driver.pl . . . 21
6.3 MEADlib . . . 24

7 Features and Feature Scripts **25**
7.1 Introduction to MEAD Features . . . 25
7.2 Sent-Feature Files . . . 25
7.3 Three-Pass Feature Calculation . . . 25
7.3.1 The Cluster Stage . . . 25
7.3.2 The Document Stage . . . 27
7.3.3 The Sentence Stage . . . 27
7.4 A Skeleton Feature Extraction Routine . . . 27

8 MEAD Classifiers **27**
8.1 The Default Classifier: default-classifier.pl . . . 27
8.1.1 Command Line Arguments . . . 27
8.1.2 Using new features with default-classifier.pl . . . 28
8.2 Other Classifiers . . . 28
8.2.1 random-classifier.pl . . . 28
8.2.2 leadbased-classifier.pl . . . 28
8.3 Creating a New Classifier . . . 29

9 MEAD Rerankers **29**
9.1 The Default Reranker: default-reranker.pl . . . 29
9.1.1 Command Line Arguments . . . 31
9.1.2 Changing the parameters of default-reranker.pl . . . 31
9.2 Other Rerankers . . . 31
9.2.1 identity-reranker.pl . . . 31
9.2.2 novelty-reranker.pl . . . 31
9.3 Creating a New Reranker . . . 32

10 MEAD Add-Ons **32**
10.1 Additional scripts . . . 32
10.2 Pre-processing MEAD input and Post-processing MEAD output . . . 34

11 Really Advanced MEAD **34**
11.1 Producing Query-Based Summaries . . . 34
11.2 Producing Alignment-Based Summaries . . . 35
11.3 Constructing IDF Databases . . . 35
11.4 Creating Docjudges . . . 35

12 Creating baselines for use in evaluation **35**
12.1 Lead-based and Random . . . 35
12.2 Extracts from Sentjudges . . . 36
12.3 Creating Sentjudges from manual summaries . . . 36

13 Evaluation **37**
13.1 Evaluation using MEAD Eval . . . 37
13.2 Evaluation using Relevance Correlation . . . 37
13.3 Evaluation using Lexical Similarity . . . 38

14 CST in summarization and evaluation **40**
14.1 Interjudge agreement and directional interjudge agreement . . . 40
14.2 Calculating connectivity . . . 41
14.3 Merging sentrel files . . . 41
14.4 Extracting only certain types of CST relations from an extract . . . 41
14.5 Calculating similarity with sentrels . . . 42

15 New Rerankers in mead_addons **42**
15.1 CST rerankers, generally speaking . . . 42
15.2 Source-based rerankers . . . 42
15.3 Time-based rerankers . . . 42
15.4 CST-based rerankers . . . 43
15.5 MMR rerankers . . . 43

16 Converting HTML and text files to docsent and cluster format **43**
16.1 Preliminaries . . . 43
16.2 From text files to docsents and/or a MEAD cluster . . . 44
16.3 From HTML files to a cluster . . . 44
16.4 Caveats . . . 44

17 CIDR: How to create document clusters **44**
17.1 How CIDR works . . . 45
17.2 The details of running CIDR . . . 45

18 Running MEAD in Other Languages **45**
18.1 Summarizing Chinese Documents with MEAD . . . 45
18.1.1 Preliminary Notes . . . 46
18.1.2 List Format . . . 46
18.1.3 GB18030 Compatability . . . 46
18.1.4 System Compatibility . . . 46
18.1.5 Running The Example . . . 46

19 Training MEAD using Support Vector Machines **47**
19.1 Data Format . . . 47
19.2 Porting . . . 48
19.3 Training . . . 48
19.4 Tuning (Development) . . . 48
19.5 Testing . . . 49

20 JHU 2001 summer workshop and SummBank **49**

21 The MEAD Web site **49**

22 Frequently Asked Questions **50**
22.1 Is additional MEAD-compatible data available? . . . 50
22.2 Is there a mailing list for MEAD? . . . 50
22.3 Can I contribute to MEAD? . . . 50
22.4 Do I need a license to use MEAD? . . . 50
22.5 How can I get help? . . . 50
22.6 How can I make sure I understand the details of how MEAD works? . . . 50

23 Demos **50**

24 Future Work **51**

25 Credits **52**

A XML DTDs **53**
A.1 cluster.dtd . . . 53
A.2 docjudge.dtd . . . 53
A.3 docpos.dtd . . . 54
A.4 docsent.dtd . . . 55
A.5 document.dtd . . . 55
A.6 extract.dtd . . . 56
A.7 mead-config.dtd . . . 56
A.8 query.dtd . . . 57
A.9 reranker-info.dtd . . . 58
A.10 sentalign.dtd . . . 58
A.11 sentfeature.dtd . . . 59
A.12 sentjudge.dtd . . . 59
A.13 sentrel.dtd . . . 59

1 Introduction

1.1 What is automatic text summarization?

According to [Man01], ”the goal of automatic summarization is to take an information source, extract content from it, and present the most important content to the user in a condensed form and in a manner sensitive to the user’s or application’s need”.

1.2 Sentence Extraction

Extractive summarization is a very robust method for text summarization. It involves assigning salience scores to some units–usually sentences or paragraphs–of a document or a set of documents and extracting these with the highest scores.

1.3 The MEAD System

MEAD is a publicly available toolkit for multi-lingual summarization and evaluation. The toolkit implements multiple summarization algorithms (at arbitrary compression rates) such as position-based, Centroid[RJB00], TF*IDF, and query-based methods. Methods for evaluating the quality of the summaries include co-selection (precision/recall, kappa, and relative utility) and content-based measures (cosine, word overlap, bigram overlap).

MEAD v1.0 and v2.0 were developed at the University of Michigan in 2000 and early 2001. MEAD v3.01 – v3.06 were written in the summer of 2001, an eight-week summer workshop on Text Summarization was held at Johns Hopkins University. See `http://www.clsp.jhu.edu/ws2001/groups/asmd` for workshop information. The workshop final report [RTS+02] is also available from the JHU site. As of Version 3.07, MEAD has been back to the University of Michigan, undergoing continuous development by the Computational Linguistics And Information Retrieval (CLAIR) group. The most recent version of the documentation is 3.08, but this refers to the software version 3.07 plus the new addons. The new addons are available at `http://www.summarization.com/mead/addons`, and they are described in sections 10 and 13-17 of this document.

MEAD is written in Perl and requires several XML-related Perl modules and an external software package to run. Also, a number of other modules and packages can be used in various extensions of MEAD. A full list of these software packages is included in the Downloading and Installation Section.

MEAD is intended to be used with multiple languages. The MEAD system can summarize English documents on all POSIX-conforming operating systems (Unix, Linux, etc.). MEAD can also summarize Mandarin Chinese documents on some POSIX operating systems.

Because of the inconsistencies in encodings, we have only tested MEAD in Mandarin Chinese on certain versions of the Solaris operating system and some versions of Linux. Please contact the mailing list (see below) if you are interested in porting MEAD to new OS’s. Adding new (human) languages should also be relatively easy, especially languages that use common character sets.

1.4 MEAD Functionality

MEAD can perform many different summarization tasks. It can summarize individual documents or clusters of related documents (multi-document summarization).

MEAD includes two baseline summarizers: lead-based and random. Lead-based summaries are produced by selecting the first sentence of each document, then the second sentence of each, etc. until the desired summary size is met. A random summary consists of enough randomly selected sentences (from the cluster) to produce a summary of the desired size.

MEAD has been primarily used for summarizing documents in English, but recently, Chinese capabilities have been added to the publicly available version of MEAD. We regret to inform the reader that Chinese summarization is in the alpha stage and may fail, depending on the encoding used and the platform. We envision it being relatively easy to coerce MEAD to produce summaries of other languages as well, but we have yet to back up this claim.

Query-based summarization is often used in natural language circles, and is (not coincidentally) included in MEAD as well. A later section of this document details query-based summarization in MEAD.

The MEAD evaluation toolkit (MEAD Eval), previously available as a separate piece of software, has been merged into MEAD as of version 3.07. This toolkit allows evaluation of human-human, human-computer, and computer-computer agreement. MEAD Eval currently supports two general classes of evaluation metrics: co-selection and content-based metrics. Co-selection metrics include precision, recall, Kappa, and Relative Utility, a more flexible cousin of Kappa. MEAD's content-based metrics are cosine (which uses TF*IDF), simple cosine (which doesn't), and unigram- and bigram-overlap. Relevance correlation has previously been used in conjunction with MEAD, but is not included with the current distribution. MEAD Eval has its very own section, near the end of this document.

1.5 Sample Use Cases

MEAD can be used in a variety of scenarios; here are a few in which MEAD can come in handy. MEAD can be used to evaluate an existing summarizer. Users can use the MEAD framework to build a summarizer entirely from scratch or they may reuseexisting pieces of MEAD. MEAD can be used to evaluate the impact of a new sentence feature on the summarization process or to test a new evaluation metric.

2 Downloading and Installation

2.1 Downloading the MEAD Distribution

Several versions of MEAD, including the current version, 3.07, can be downloaded from the MEAD website:

`http://tangra.si.umich.edu/clair/mead`

MEAD-3.07.tar.gz contains everything you need to get started using MEAD.

2.2 External Software

External software is either used directly by MEAD (e.g. Perl and Perl modules) or in various applications of MEAD (e.g. Support Vector Machines (SVM) to train MEAD). Required software includes a recent release of Perl itself, and several modules. Optional software includes several other Perl modules and some non-Perl related software packages. A full list is given below.

We have included the essential packages of external software with the MEAD distribution, but if you wish to download them and other useful packages yourself, the most recent versions can be found on CPAN (www.cpan.org). Note that the versions of the modules included with the MEAD distribution may be out of date. The user may download the latest versions from CPAN (www.cpan.org) and install them himself.

MEAD's installation script will install all the required modules on your system, if they are not present. However, MEAD doesn't install the modules as the superuser, so they aren't generally available to other Perl programs. If you want these modules installed in the regular Perl way, you should install them yourself. Again, see CPAN (www.cpan.org) for instructions on how to install Perl modules.

- **Perl 5.5 or above**
  - `http://www.perl.com`
  - MEAD will likely work for any version of Perl at or above 5.0, but this is NOT guaranteed.
- **XML::Parser - required**
  - `http://www.cpan.org/authors/id/C/CO/COOPERCL/XML-Parser.2.30.tar.gz`
- **Expat – required**
  - `http://sourceforge.net/projects/expat/`
  - Expat is a publicly avaible parsing tool used by the XML::Parser module.
- **XML::Writer - required**
  - `http://www.cpan.org/authors/id/DMEGG/XML-Writer-0.4.tar.gz`
- **XML::TreeBuilder - required**

  - `http://www.cpan.org/authors/id/S/SB/SBURKE/XML-TreeBuilder-3.08.tar.gz`

- **Text::Iconv - required**
  - `http://search.cpan.org/CPAN/authors/id/M/MP/MPOITR/Text-Iconv-1.2.tar.gz`

- **Support Vector Machines (SVM) - optional**
  - `http://ais.gmd.de/\~http://www.cs.cornell.edu/People/tj/svm_light/`
  - SVM can be used to train MEAD's classifier.

- **SMART - optional**
  - `ftp://ftp.cs.cornell.edu/pub/smart/`
  - for evaluation by Relevance Correlation

- **LT-XML - optional**
  - `http://www.ltg.ed.ac.uk/software/xml/index.html`
  - This software can be used to check an XML document against the DTD that specifies its form. It is highly recommended that you use this or similar software.

2.3 Installing MEAD

1. You must have Perl installed to install MEAD. The English examples referred to in this documentation have been tested with Perl 5.6.0 on Solaris 5.7 and Linux (kernel 2.2 and 2.4). The Chinese examples have been tested on Linux (kernel 2.2) with Perl 5.6.0.

2. Unpack MEAD-3.07.tar.gz. Several directories will be created. Of course, the absolute paths to each of these conceptual directories will be different from machine to machine (or from installation to installation). **Thus, when you see $MEAD_DIR or another conceptual subdirectory in this documentation, substitute the appropriate directory on your machine.**

   - $MEAD_DIR = the result of the unpacking process. This is the base directory where MEAD is to be installed.
   - $BIN_DIR = $MEAD_DIR/bin
     This directory contains MEAD executables, including mead.pl, driver.pl, and MEAD's core classifiers and rerankers.
   - $SCRIPTS_DIR = $BIN_DIR/feature-scripts
     This subdirectory of $BIN_DIR contains MEAD feature scripts.
   - $LIB_DIR = $MEAD_DIR/lib
     The $LIB_DIR directory contains Perl modules used by MEAD.
   - $DOCS_DIR = $MEAD_DIR/docs
     Documentation is kept here.
   - $DATA_DIR = $MEAD_DIR/data
     Example clusters live here.
   - $DTD_DIR = $MEAD_DIR/dtd
     DTD's for all the MEAD XML objects are kept here.
   - $ETC_DIR = $MEAD_DIR/etc
     contains IDF DBM files and possibly other peripheral resources.
   - $USER_DIR = $MEAD_DIR/user
     contains a sample .meadrc file (NOTE: it's really there, you may have to run "ls -a" to display it, though.) and a sample mead/data directory that a user might have in his home directory.

3. Before running the installation script, you may want to check if you have the four required modules installed and install them in the normal Perl manner. If you don't have a Perl module installed, MEAD's install script will automatically install a copy that only it can use, so if you plan on doing XML-related work in Perl, it may well be worth your while to install these modules yourself before installing MEAD.

   Note that if at some later time, you install any of the four required modules, pre-existing versions on MEAD will not use them. You must re-install MEAD if you really want it to use your newly installed modules.

4. From $MEAD_DIR, run "`perl Install.PL`". This installs any needed modules that were not previously on your sytstem, automatically changes the #! directives in all Perl scripts to the correct path for your system, updates a MEAD path variable in the MEAD::MEAD module, and builds English and Chinese IDF DBM files in your architecture's specific format.

   If you experience installation-related problems with MEAD, refer to `Install.PL`'s output when contacting the MEAD team.

3 Simple MEAD tutorial

In the following, we present the basic just about all one needs to use MEAD.

3.1 An Example Cluster: GA3

On August 23, 2000, a Gulf Air Airbus crashed off the coast of Bahrain in the Persian Gulf. This disaster triggered hundreds (probably thousands) of online news articles in the following days and weeks. Many of these articles were collected as a sample cluster on a large news story.

Three such documents (in docsent format) are included with the MEAD distribution as an example cluster. These documents were selected from a much larger set of documents on the Gulf Air crash, thus explaining their somewhat cryptic names: articles 41, 81, and 87. This cluster is located in the `$DATA_DIR/GA3` directory and has the following structure:

```
$DATA_DIR/GA3/GA3.cluster
             /GA3.config
             /GA3.10.extract
             /GA3.20.extract
             /GA3.sentjudge
             /GA3.query
             /docsent/*.docsent
             /feature/GA3.*.sentfeature
```

The `GA3` directory contains a cluster file, GA3.cluster; a config file, GA3.config; two extract files, one at 10%, the other at 20%; a sentjudge file; an example query file; and subdirectories containing docsent files and feature files. Each of these types of files will be explained in detail in later sections. For purposes of summarizing the cluster, only the cluster and docsent files are needed:

```
$DATA_DIR/GA3/GA3.cluster
             /docsent/*.docsent
```

The mead.pl script assumes (by default) that a cluster has this structure (minimally, a cluster file in the directory and a subdirectory named docsent containing all the docsent files named in the cluster file), unless the user tells the script otherwise (See the section on .meadrc files for how this is done, although this is not required.). For now, stick to this structure. Note that MEAD's $DATA_DIR is by default not writable except by the user who installed MEAD. The GA3/feature directory, which is a by-product of summarization, is already created. If it were not, MEAD would run into problems.

3.2 Summarizing the GA3 cluster

For the purpose of these examples (and to make them easier to read), we assume that we're in the `$BIN_DIR` directory. MEAD can be run from other directories by correctly specifying the path to mead.pl. To go to `$BIN_DIR`, use the following command, **replacing `$BIN_DIR` with the corresponding path on your machine.**

```
% cd $BIN_DIR
```

Alternately, one can append `$BIN_DIR` to the PATH environment variable, which will effectively do the same thing. On my machine, the following command does the job.

```
% export PATH=$PATH:$BIN_DIR
```

where $BIN_DIR is the conceptual directory described in the Installation Section. This above command may be a bit confusing, as $XXX is used in an overloaded manner in this document. First of all, Perl's scalar variables are named beginning with a '$'. We have tried to keep the referencing of Perl variables to a minimum in this document. Second, Unix, Linux, etc. environment variables begin with a '$'. However, the only environment variable mentioned in this document is PATH ($PATH). Also, this document uses $XXX to refer to one of MEAD's conceptual directories. These conceptual directories are as listed in a previous section; when in doubt, assume that a word beginning with a '$' is a conceptual directory. An example, given that your $BIN_DIR is located at /usr/mead/bin, is:

```
% export PATH=$PATH:/usr/mead/bin
```

In general, MEAD is run as follows:

```
% ./mead.pl [options] cluster_name
```

Here are some examples of summarizing the GA3 cluster.

```
% ./mead.pl GA3
```

summarizes the GA3 cluster and writes a summary to the standard output.

```
% ./mead.pl -extract GA3
```

writes an extract to standard output.

```
% ./mead.pl -words -absolute 100 GA3
% ./mead.pl -w -a 100 GA3
```

write a summary of about 100 words to standard output. By default, MEAD's compression basis is "sentences" and by default, MEAD will give a 20 percent summary.

```
% ./mead.pl -sentences -percent 10 GA3
% ./mead.pl -s -p 10 GA3
```

In this example, the "-sentences" (or "-s") is unnecessary, as that is the default. The user may also specify an absolute number of sentences to extract, or a percentage of words. In general (and unless stated otherwise), mead.pl's options mix and match well.

```
% ./mead.pl -extract -output GA3.extract GA3
% ./mead.pl -output ../data/GA3.summary GA3
```

These examples make use of the -o (-output) option. This writes the output (be it a extract, summary, or meadconfig object) to the specified file instead of the standard output.

```
% ./mead.pl -RANDOM GA3
% ./mead.pl -system RANDOM GA3

% ./mead.pl -LEADBASED GA3
% ./mead.pl -system LEADBASED GA3
```

The above examples produce random and lead-based summaries, respectively. In general, the “-system” option just gives a name to the configuration that you’re using. The “RANDOM” and “LEADBASED” systems are special in that when you specify either of these systems, special classifiers and rerankers are used, instead of the default ones.

The user can specify non-standard classifiers and rerankers using the `-classifier` and `-reranker` options, respectively. For more information on classifiers and rerankers are and how to write new ones, see the appropriate section.

The user can also specify alternate scripts to calculate the default features: Length, Position, and Centroid; or add new features using the `-feature` option. See the MEAD Architecture Section for how to do this (and why you might want to).

If you just want to give a name to the configuration, then you can say:

```
% ./mead.pl -system MySystem GA3
```

Note that the name of your system cannot start with a hyphen. This causes MEAD to think that no argument to the system option is given and that the name of your system is the next command line option. Obviously, MEAD may do something unpredictable. If you push the envelope when using MEAD, it may collapse around you.

3.3 So what does mead.pl really do?

mead.pl is really just a (hopefully) smart wrapper around the core MEAD driver script, driver.pl, which is in turn a wrapper around MEAD’s feature scripts, classifiers, rerankers. It also encapsulates the functionality of extract-to-summary.pl and the now-defunct write-config.pl.

First of all, driver.pl takes as input (to its standard input) a meadconfig object and writes an extract object to its standard output. (Both of these objects are described in detail in future sections.) In between, driver.pl is responsible for creating any specified features that don’t already exist and calling the classifier and reranker. For more information on these pieces of MEAD, see the appropriate sections.

MEAD does not check that feature files are up to date, or even that they specify features for the right cluster. If you change the cluster by adding, removing, or modifying the documents, you MUST also remove the old sentfeature files as well (these are usually stored in the “feature” subdirectory of each cluster) to avoid getting junk summaries or simply crashing MEAD. However, this aspect of MEAD can be useful, when the user already has such features computed. As an example, in the Document Understanding Conference (DUC) competition, participants were given the number of words each sentence contained. Since segmenters may segment a given sentence differently, resulting in different sentence lengths, we put the given length calculations in a feature file in the feature directory, thereby avoiding potential mixups.

The above is the limit of driver.pl’s functionality. It does not take command-line arguments and it cannot output summaries or meadconfig files. The mead.pl script was created to easily build a meadconfig XML object suitable for feeding to driver.pl and to receive driver.pl’s output, possibly creating a summary from the returned extract.

3.4 How mead.pl locates clusters

One of the major benefits of mead.pl is that it does its best to locate clusters that aren’t immediately available. This is an improvement over the operation of driver.pl and mead-config files, as with this approach, the user must explicitly specify the full path to the cluster. mead.pl locates clusters by checking for the cluster in some predefined locations (e.g. the /mead/data directory in the user's home directory, if present) as well as user-defined locations (via `-data_path` options to the command line and `data_path` in .meadrc files). mead.pl's procedure is as follows:

1. mead.pl checks if the cluster argument specified is an absolute path. If so, it will use that path. Example:

   ```
   % mead.pl /usr/home/mead/data/GA3
   ```

2. mead.pl checks if the cluster argument is a subdirectory of the current directory. If it is and an appropriate cluster file exists in the subdirectory, that directory is used.

3. Command-line data_path options to mead.pl are checked next. The user may specify one directory, or multiple directories separated by colons (:).

4. mead.pl checks the user's .meadrc file for a data_path option. Each colon-separated directory is checked for the cluster. If the -meadrc command-line option is specified, mead.pl checks that meadrc file instead.

5. The user's /mead/data directory is checked next. If the directory doesn't exist, this step is ignored.

6. Any data_path options in $MEAD_DIR/.meadrc are checked.

7. Finally, MEAD's data directory, $DATA_DIR, is checked. Note that this is a last-ditch effort to find the cluster. Also, this makes the examples more readable. :)

3.5 Command-line options to mead.pl

Most users will only use the first several of these options. These common options have already been detailed above.

What follows is a full listing of the command-line options for mead.pl. Note that the options for .meadrc files (described in a future section) make use of these same options.

- `-help`, `-?`

  print help information and exit. Help info consists of the list of available options and is meant to be a reminder of what options are available, **not** a comprehensive how-to. Refer to this section for how to use each option.

- `-summary`

  produce a summary (This is the default.)

- `-extract`

  produce an extract (instead of a summary)

- `-meadconfig`

  produce a meadconfig object (instead of a summary or extract).

- `-centroid`

  produces centroid information

- `-scores`

  print the feature values and composite score (as assigned by the classifier) for each sentence. The user can specify a non-standard feature set, which will be the features printed, and/or a non-standard classifier, which computes the composite scores.

- `-output_mode mode`

  mode can be either "summary", "extract", or "meadconfig". The -summary, -extract, and -meadconfig options are shorthand for "-output_mode summary", "-output_mode extract", and "-output_mode meadconfig", respectively.

- `-basis basis, -b basis, -compression_basis basis`

  The argument can be either “words” or “sentences”. See below.

- `-sentences, -s`

  produce a summary whose length is either an absolute number or a percentage of the number of *sentences* of the original cluster. (This is the default.)

- `-words, -w`

  produce a summary whose length is either an absolute number or a percentage of the number of *words* of the original cluster.

- `-percent num, -p num, -compression_percent num`

  produce a summary whose length is num% the length of the original cluster. (The default is -percent 20)

- `-absolute num, -a num, -compression_absolute num`

  produce a summary whose length is num (words/sentences) regardless of the size of the original cluster. NOTE: if both -percent and -absolute are specified, MEAD’s behavior may be erratic.

- `-system sys`

  give the configuration the name “sys”. Except for the special cases RANDOM and LEADBASED, this just adds this field to the extract.

- `-system RANDOM, -RANDOM`

  produce a random summary (and name the system “RANDOM”).

- `-system LEADBASED, -LEADBASED`

  produce a lead-based summary, selecting the first sentence from each document, then the second sentence, etc.. (and name the system “LEADBASED”). NOTE: RANDOM and LEADBASED systems override any classifier, reranker, and features that may be specified.

- `-feature name commandline`

  add a feature named “name” whose feature script is run by “commandline” to the feature set or replace the existing feature of the same name. Note however, that if the target cluster already have sentfeature files for the specified feature, those will be used and the specified script will NOT be run.

  NOTE: this option takes two arguments, as opposed to the rest, which take only one.

- `-classifier commandline`

  use the specified classifier instead of the default. NOTE: if the classifier command takes arguments, make sure to enclose the commandline with quotes to make the shell interpret it as a single argument. However, when specifying a classifier in a .meadrc file, quotes are not necessary.

- `-reranker commandline`

  use the specified ranker instead of the default. See the NOTE for -classifier.

- `-lang language`

  The default is “ENG”. This option doesn’t really do a whole lot currently. Since mead.pl doesn’t currently do Chinese summarization, you’ll probably never have to specify “CHIN”. To do summarizaton in Chinese, refer to the appropriate section (you’ll have to use the old-fashioned meadconfig file method).

- `-data_path path`

  look for the target cluster in each of the directories specified by path. The name of the cluster (the last thing on the command line) is appended to each element of path. Each of these constructed directories is checked to see if it contains the cluster (and the desired directory structure) as described in the beginning of this section.

The user can specify multiple directories by separating two directories with a colon, e.g., "/usr/mead/data:-/usr/local/clusters/".

Any data_path options given in the user's and system's .meadrc files are appended to the search path. The search begins in the current directory, regardless of whether "." was actually specified in any data_path option (command line or .meadrc file).

- `-cluster_dir dir`

  look for the target cluster in the specified directory ONLY. This blocks any data_path arguments provided.

  If this option is specified, the cluster file MUST be in the argument to this option (not in a subdirectory, nor a directory with the same name as the cluster).

  NOTE: use this option sparingly. Probably the only time when this option should be used is when there are multiple clusters named the same thing in different pieces of data_path (which can be specified in the .meadrc files).

- `-docsent_dir dir`

  look for docsent files in the specified directory, instead of some subdirectory of the cluster directory.

- `-feature_dir dir`

  look for and/or output feature files in/to the specified directory, instead of some subdirectory of the cluster directory.

- `-docsent_subdir subdir`

  look for docsent files in the specified **sub**directory of the cluster directory. By default, this is "docsent". This option is supported in order to conform to the cluster file structures used by older systems.

- `-feature_subdir subdir`

  look for feature files in the specified **sub**directory of the cluster directory. By default, this is "feature". This option is supported in order to conform to the cluster file structures used by older systems.

- `-meadrc rcfile`

  use the specified .meadrc file instead of the one (possibly) located in the user's home directory. You must supply the entire path (including the filename) of the new rc file.

4 Advanced MEAD

4.1 Specifying summaries using mead-config files

Prior to version 3.07, MEAD was used like

```
% cat ../data/GA3/GA3.config | ./driver.pl > ../data/GA3/GA3.xx.extract
```

which produces an extract file in the ../data/GA3 directory. The above command assumes that ../data/GA3/GA3.config is a mead-config file that specifies the desired summary, including compression rates, etc.

If one wanted a summary, then he would run

```
% ./extract-to-summary.pl ../data/GA3/GA3.cluster ../data/GA3/docsent \
      ../data/GA3/GA3.xx.extract
```

which writes a summary to the standard output. Of course, this can be redirected to a file, e.g. ../data/GA3/GA3.xx.summary.

Now, the user can instead run

```
% ./mead.pl -o GA3.xx.summary GA3
```

(or)

```
% ./mead.pl -p 17 -w -o GA3.17.pct.words.summary GA3
```

Of course, the old ways of doing things, catting files to driver.pl and using extract-to-summary.pl still work, but it is recommended that all but the most die-hard of MEAD purists (if there are such individuals) use mead.pl instead.

Regardless, there are probably instances where using a config file is preferable, especially with aberrant file structures, etc. In such a case, you can use mead.pl to produce a meadconfig file (using the "-meadconfig" option) and then modify the resulting config file. Alternately, you can write your own meadconfig file from scratch. Obviously, the latter is not recommended.

4.2 Using .meadrc files to specify defaults

Many Unix-style applications allow specification of options using .rc files. These "dot files" allow the user to specify options in a file without having to type them in at the command line each time MEAD is used.

The mead.pl script reads two rc files and uses the options they specify. First, MEAD reads the file: $MEAD_DIR/-.meadrc, which contains system-wide defaults. Next, MEAD reads the user's .meadrc file. By default, MEAD looks for a file named .meadrc in the user's home directory. (This is fairly standard in Unix-style apps.) Alternately, the user can specify a different rc file with

```
% ./mead.pl -rc new.meadrc
```

This allows the user to create many different configurations for MEAD without having to specify many options on the command line. If the user wishes to ignore the .meadrc file in his/her home directory (if there is one), then specify a non-existant file, as in

```
% ./mead.pl -rc none
```

given that no file named "none" exists in the current directory.

Specifying a meadrc file is as easy as creating a file named .meadrc in your home directory and adding any number of options, one per line. Figure 1 is a sample .meadrc file which specifies that mead.pl should run in extract mode instead of the default summary mode, and produce summaries that on average are about 200 words in length. Also, the RANDOM special system is specified. This means that random baseline summaries will be produced. Note that it's probably not a good idea to specify the RANDOM system in a .meadrc file, as this system is intended as a baseline for evaluation of high-quality summarizers; this example is for illustrative purposes only.

```
compression_basis       words
compression_absolute    200
output_mode             extract
system                  RANDOM
```

Figure 1: Sample .meadrc file

The options supported in .meadrc files are *almost* identical to those supported by mead.pl via the command line. Note that the an rc file does not have to be named .meadrc; such a file can be named anything allowable on the platform. However, names that include "meadrc" in some way (e.g. "meadrc2") allow other users a more intuitive understanding of which files do what. Additionally, secondary .meadrc files should probably **not** be dotfiles, as they may get lost or forgotten. Note also that mead-config files generally end in ".config" or ".meadconfig", which means that non-sadists should avoid using these extensions in naming .meadrc files.

A .meadrc file can currently have all the following options that a mead.pl can take as command-line options (without the hyphens). Those command-line options that have "short-hand" versions (e.g. "-summary", "-p 20", and "-RANDOM") should be expanded to their more/most verbose versions "-output_mode summary", "-compression_percent 20", "system RANDOM", respectively. A few other common options used in .meadrc files are "compression_basis" and "data_path".

The “feature”, “classifier”, and “reranker” options all involve specifying scripts (and any needed arguments) for MEAD to run via the command line. When specifying them as arguments to mead.pl, they should appear in quotes so the shell will group the script and its arguments as one item. However, when specifying these options in a .meadrc file, no quotes are needed. In the case of “feature” options, the second group of non-spaces is taken to be the feature name and the remainder of the line is taken to be the script (with argument, if necessary). Everything after “classifier” or “reranker” is taken to be the script and arguments.

Figure 2 shows a .meadrc file that specifies a “data_path” option, as well as a nonstandard feature and a new classifier command that uses that feature. See the appropriate sections for more information on features, feature scripts, and classifiers.

```
compression_basis       words
compression_absolute    200
output_mode             meadconfig

# Notice the ':' separating two directories.
data_path               /usr/home/winkela:/usr/home/winkela/data

# This is a comment.
# Note that the '\' on the next line indicates a continuation
# of that line.
feature SimWithFirst \
   /clair4/mead/bin/feature-scripts/SimWithFirst.pl

# Note that we don't need quotes here...
classifier /clair4/mead/bin/default-classifier.pl Length 9 \
   Centroid 1 Position 1 SimWithFirst 1
```

Figure 2: A more adventurous .meadrc file

Although the .meadrc files can have an option that can be specified on the command line to mead.pl, some options should not in general be specified in .meadrc files. One such option is “cluster_dir”, which specifies the only place where MEAD should look for the cluster file.

For a full listing of the options that .meadrc files can have, see the appropriate section in the Getting Started section on mead.pl command-line options or type “`./mead.pl -help`” from `$BIN_DIR`.

5 MEAD XML Objects

This subsection describes the XML objects used as input to and output from MEAD, as well as communication between the different components of MEAD. The DTDs describing these objects used are listed at the end of this document.

5.1 Common Objects

By “Common Objects”, we mean objects that are often used directly by the user. These include the mead-config, cluster, docsent, extract, and summary objects. Mead-config objects and extract objects are the input and output, respectively, of driver.pl. mead.pl and extract-to-summary.pl can output summary objects. And of course, cluster and docsent objects represent the actual cluster to be summarized.

5.1.1 mead-config

The MEAD driver program (`$BIN_DIR/driver.pl`) reads a mead-config file from its standard input. This file specifies several pieces of information that it needs to summarize a cluster, including the location of the cluster, the features to compute and where to store them on disk, the classifier to use (with any arguments), the reranker to use, and the desired summary size. Figure 3 shows an example mead-config file.

We now describe the functions of the various elements of a mead-config file and their attributes:

```
<MEAD-CONFIG TARGET='GA3' LANG='ENG' CLUSTER-PATH='/clair4/mead/data/GA3'
  DATA-DIRECTORY='/clair4/mead/data/GA3/docsent'>

<FEATURE-SET BASE-DIRECTORY='/clair4/mead/data/GA3/feature/'>
  <FEATURE NAME='Centroid'
SCRIPT='/clair4/mead/bin/feature-scripts/Centroid.pl HK-WORD-enidf ENG'/>
  <FEATURE NAME='Position'
SCRIPT='/clair4/mead/bin/feature-scripts/Position.pl'/>
  <FEATURE NAME='Length'
SCRIPT='/clair4/mead/bin/feature-scripts/Length.pl'/>
</FEATURE-SET>

<CLASSIFIER COMMAND-LINE='/clair4/mead/bin/default-classifier.pl \
  Centroid 1 Position 1 Length 9' SYSTEM='MEADORIG' RUN='10/09'/>

<RERANKER COMMAND-LINE='/clair4/mead/bin/default-reranker.pl MEAD-cosine 0.7'/>

<COMPRESSION BASIS='sentences' PERCENT='20'/>

</MEAD-CONFIG>
```

Figure 3: An example Mead Config object

- <MEAD-CONFIG>
  - LANG: the language of the cluster (currently “ENG” or “CHIN”)
  - CLUSTER-PATH: Path to the .cluster file you want to summarize
  - DOC-DIRECTORY: Path where the source documents in docsent format are located
  - TARGET: The name of the cluster file (without the .cluster or directory)
    Thus, if you want to summarize the cluster defined by /usr/home/data/GA3.cluster, TARGET = “GA3”
- <FEATURE-SET>
  - BASE-DIRECTORY: Path where MEAD will look for and/or produce features
- <FEATURE>
  - NAME: The name of the feature to use
  - SCRIPT: The full-path command (including options) to run the script used to generate this feature if it does not exist in BASE-DIRECTORY (above). Note: DOC-DIRECTORY (above) will be appended as an argument to this command when the feature script is run. See the section on Features for more information.
- <CLASSIFIER>
  - COMMAND-LINE: Command to call the classifier to be used, including all command-line arguments
- <RERANKER>
  - COMMAND-LINE: Command to call the reranker to be used, including all command-line arguments
- <COMPRESSION>
  - BASIS: sentences or words
  - PERCENT: length, in percent of the size of the full cluster (in the specified BASIS), that the summary should be
  - ABSOLUTE: length, in absolute number of sentences of words (as specified by basis), that the summary should be. Note that exactly one of PERCENT and ABSOLUTE should be specified.

5.1.2 Cluster

A cluster object lists the names of the documents that will be summarized. Figure 4 shows the cluster file for the GA3 cluster. Each DID corresponds to a file named $DID.docsent in the GA3 cluster's docsent subdirectory, e.g. $DATA_DIR/GA3/docsent/41.docsent.

```
<?xml version='1.0'?>
<!DOCTYPE CLUSTER SYSTEM '/clair4/mead/dtd/cluster.dtd'>

<CLUSTER LANG='ENG'>
        <D DID='41' />
        <D DID='81' />
        <D DID='87' />
</CLUSTER>
```

Figure 4: An example Cluster object

5.1.3 Docsent

A Docsent object is a document segmented into sentences. Firgure 5 shows the first several sentences from document 41 from the GA3 cluster.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE DOCSENT SYSTEM '/clair4/mead/dtd/docsent.dtd'>

<DOCSENT DID='41' LANG='ENG'>
<BODY>
<HEADLINE>
<S PAR="1" RSNT="1" SNO="1">Egyptians Suffer Second Air
Tragedy in a Year </S>
</HEADLINE>
<TEXT>
<S PAR='2' RSNT='1' SNO='2'>CAIRO, Egypt   -- The crash of a
Gulf Air flight that killed 143 people in Bahrain is a disturbing
deja vu for Egyptians: It is the second plane crash within a
year to devastate this Arab country.</S>
<S PAR='2' RSNT='2' SNO='3'>Sixty-three Egyptians were on
board the Airbus A320, which crashed into shallow Persian Gulf
waters Wednesday night after circling and trying to land in
Bahrain.</S>
<S PAR='2' RSNT='3' SNO='4'>On Oct. 31, 1999, a plane carrying
217 mostly Egyptian passengers crashed into the Atlantic Ocean
off Massachusetts.</S>
<S PAR='2' RSNT='4' SNO='5'>The cause has not been determined,
providing no closure to the families, whose grief was reopened
this month with the release of a factual report by the National
Transportation Safety Board.</S>
</TEXT>
</BODY>
</DOCSENT>
```

Figure 5: An example Docsent object

5.1.4 Extract

An Extract contains a list of sentences that will be used in the summary. Sentences are sorted in the order they appear. Figure 6 shows a seven-sentence extract of the GA3 cluster.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE EXTRACT SYSTEM '/clair/tools/mead/dtd/extract.dtd'>

<EXTRACT QID='GA3' LANG='ENG' COMPRESSION='7'
SYSTEM='MEADORIG' RUN='Sun Oct 13 11:01:19 2002'>
<S ORDER='1' DID='41' SNO='2' />
<S ORDER='2' DID='41' SNO='3' />
<S ORDER='3' DID='41' SNO='11' />
<S ORDER='4' DID='81' SNO='3' />
<S ORDER='5' DID='81' SNO='7' />
<S ORDER='6' DID='87' SNO='2' />
<S ORDER='7' DID='87' SNO='3' />
</EXTRACT>
```

Figure 6: An example Extract object

5.1.5 Summary

The Summary is the final output from the summarization process. Note that Summary objects are not XML-compliant. Instead, they are meant to be the human readable output of MEAD.

```
[1]The Disaster Relief Fund Advisory Committee has approved a
grant of $3 million to Hong Kong Red Cross for emergency relief
for flood victims in Jiangxi, Hunan and Hubei, the Mainland.
[2]Together with the earlier grant of $3 million to World Vision
Hong Kong, the Advisory Committee has so far approved $6 million from the
Disaster Relief Fund for relief projects to assist the victims
affected by the recent floods in the Mainland.
[3]The Disaster Relief Fund Advisory Committee has approved a
grant of $3 million to the Salvation Army for emergency relief
for flood victims in Hunan and Guangxi, the Mainland.
[4]The Disaster Relief Fund Advisory Committee has approved a
grant of $5.39 million to Medecins Sans Frontieres for emergency
relief for flood victims in Hunan, Sichuan and Yunnan, the Mainland.
[5]To ensure that the money will be used for the purpose
designated, the Government has required Medecins Sans Frontieres
to submit an evaluation report and audited accounts on the use of
the grant after the project has been completed.
```

Figure 7: An example Summary object

5.2 Less Common MEAD Objects

This category of MEAD objects are used by driver.pl and may be used by the user, for instance, when using the relative-utility.pl script to evaluate an extract.

5.2.1 SentFeature

A sentfeature object assigns (possibly several) feature scores to each sentence in a cluster. Figure 8 shows a few selected lines from the Centroid sentfeature file for the GA3 cluster.

```
<?xml version='1.0'?>

<SENT-FEATURE>
        <S DID="87" SNO="1" >
                <FEATURE N="Centroid" V="0.274894051973267" />
        </S>
        <S DID="87" SNO="2" >
                <FEATURE N="Centroid" V="0.828824590972425" />
        </S>
        <S DID="81" SNO="1" >
                <FEATURE N="Centroid" V="0.153774221389168" />
        </S>
        <S DID="81" SNO="2" >
                <FEATURE N="Centroid" V="1.000000000000000" />
        </S>
        <S DID="41" SNO="1" >
                <FEATURE N="Centroid" V="0.153896779384946" />
        </S>
        <S DID="41" SNO="2" >
                <FEATURE N="Centroid" V="0.981981515400504" />
        </S>
</SENT-FEATURE>
```

Figure 8: An example SentFeature object

5.2.2 SentJudge

A sentjudge object is used to describe sentence utility scores given by judges to individual sentences in a document or cluster. Figure 9 shows an example sentjudge file.

5.2.3 Query

A query object describes the text of a retrieval query. Query objects are used in query-based summarization. See the appropriate section in Really Advanced MEAD.

5.3 XML Objects not used by core MEAD

5.3.1 Document

The document object represents a document which is not explicitly segmented into sentences. The document object is not currently used by core MEAD. Instead, the documents in a cluster are stored as docsent files.

5.3.2 Docpos

A docpos object is a document with Part of Speech Tags.

5.3.3 Docjudge

A docjudge object describes the retrieval ranking obtained from the search engine (Smart) given a query.

```
<?xml version='1.0'?>
<SENT-JUDGE QID='551'>
<S DID='D-19980731_003.e' PAR='1' RSNT='1' SNO='1'>
                <JUDGE N='smith' UTIL='10'/>
                <JUDGE N='huang' UTIL='10'/>
                <JUDGE N='moorthy' UTIL='6'/>
</S>
<S DID='D-19980731_003.e' PAR='2' RSNT='1' SNO='2'>
                <JUDGE N='smith' UTIL='6'/>
                <JUDGE N='huang' UTIL='10'/>
                <JUDGE N='moorthy' UTIL='10'/>
</S>
<S DID='D-19980731_003.e' PAR='3' RSNT='1' SNO='3'>
                <JUDGE N='smith' UTIL='6'/>
                <JUDGE N='huang' UTIL='9'/>
                <JUDGE N='moorthy' UTIL='10'/>
</S>
<S DID='D-19981105_011.e' PAR='5' RSNT='2' SNO='7'>
                <JUDGE N='smith' UTIL='2'/>
                <JUDGE N='huang' UTIL='1'/>
                <JUDGE N='moorthy' UTIL='4'/>
</S>
</SENT-JUDGE>
```

Figure 9: An example Sentjudge object

```
<?xml version='1.0'?>
<!DOCTYPE QUERY SYSTEM "/clair4/mead/dtd/query.dtd" >

<QUERY QID="Q-551-E" QNO="551" TRANSLATED="NO">
<TITLE>
Natural disaster victims aided
</TITLE>
<DESCRIPTION>
The description is usually a few sentences describing the
cluster.
</DESCRIPTION>
<NARRATIVE>
The narrative often describes exactly what the user is looking
for in the summary.
</NARRATIVE>
</QUERY>
```

Figure 10: An example Query object

```
<?xml version='1.0'?>
<!DOCTYPE DOCUMENT SYSTEM '/clair4/mead/dtd/document.dtd'>

<DOCUMENT DID='D-19970701_001.e' DOCNO ='1' LANG='ENG' >
<EXTRACTION-INFO SYSTEM='./hkmead.pl Centroid 1 Position 1
Length 9' RUN='' COMP
RESSION='20' QID='D-19970701_001.e'/>
<BODY>
<TEXT>
The ceremony took place in the Grand Hall of the Hong Kong Convention
and Exhibition Centre (HKCEC) Extension and was attended by some 4,000
guests, including foreign ministers and dignitaries from more than 40
countries and international organisations, and about 400 of the
world's media. Representing China were Mr Jiang; HE Mr Li Peng,
Premier of the State Council of the PRC; HE Mr Qian Qichen, Vice
Premier of the State Council of the PRC; General Zhang Wannian, Vice
Chairman of the Central Military Commission of the PRC; and HE Mr Tung
Chee Hwa, the Chief Executive of the Hong Kong Special Administrative
Region (HKSAR) of the PRC.  This was followed at the stroke of
midnight by the playing of the Chinese National Anthem and the raising
of the Chinese national flag and the flag of the Hong Kong Special
Administrative Region (HKSAR) within the first minute of the new day
(Tuesday).  Entry of Guards of Honour Entry of Officiating Parties
Salute by Guards of Honour Speech by His Royal Highness The Prince of
Wales Entry of Flag Parties British National Anthem Lowering of Union
and Hong Kong Flags

Chinese National Anthem Raising of Chinese and Hong Kong Special
Administrative Region Flags Departure of Flag Parties Speech by
President of the People's Republic of China, Mr Jiang Zemin Departure
of Officiating Parties

Departure of Guards of Honour
</TEXT>
</BODY>
</DOCUMENT>
```


Figure 11: An example Document object

5.3.4 Sentalign

A sentalign object describes the sentence mappings between two translations of the same document.

5.3.5 Sentrel

A sentrel object describes relationships between pairs of sentences.

6 MEAD Interfaces

6.1 mead.pl

The mead.pl script is the primary interface to the MEAD summarization system. However, it is little more than a wrapper around MEAD's driver.pl script (which in turn calls other scripts) that combines options and defaults from various sources (command-line options and .meadrc files) into mead-config object that is passed to driver.pl's standard input. It reads driver.pl's output—an extract—and either writes the extract or the corresponding summary, to either a file (if one is specified) or by default, the standard output.

The mead.pl script has already been described extensively in the MEAD Usage section. See that section for said description.

6.2 driver.pl

The driver.pl script ties together all the inner workings of the MEAD summarizer. It takes as input a mead-config object and yields an extract object as output. It is used by mead.pl to produce extracts (which can then be converted to summary objects) but may also be used directly by the user by `cat`ing a mead-config file to its standard ouput.

driver.pl's execution has several stages:

1. driver.pl first reads the mead-config object from the standard input.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE DOCPOS SYSTEM '/clair4/mead/dtd/docpos.dtd' >

<DOCPOS DID='D-19970701_001.e' LANG='ENG'>
<BODY>
<HEADLINE>
<S PAR='1' RSNT='1' SNO='1'> <W C='JJ'>Solemn</W> <W
C='NN'>ceremony</W> <W C='VBZ'>marks</W> <W
C='NNP'>Handover</W> </S>
</HEADLINE>
<TEXT>
<S PAR='2' RSNT='1' SNO='2'><W C='DT'>A</W> <W
C='JJ'>solemn</W><W C=','>,</W> <W C='JJ'>historic</W> <W
C='NN'>ceremony</W> <W C='VBZ'>has</W> <W C='VBN'>marked</W> <W
C='DT'>the</W> <W C='NN'>resumption</W>
<W C='IN'>of</W> <W C='DT'>the</W> <W C='NN'>exercise</W> <W
C='IN'>of</W> <W C='NN'>sovereignty</W> <W
C='IN'>over</W> <W C='NNP'>Hong</W> <W C='NNP'>Kong</W> <W
C='IN'>by</W> <W C='DT'>the</W> <W
C='NNS'>People</W><W C='POS'>'s</W> <W C='NNP'>Republic</W> <W
C='IN'>of</W> <W C='NNP'>China</W><W
C='.'>.</W></S>
<S PAR='3' RSNT='1' SNO='3'><W C='PRP$'>His</W> <W
C='NNP'>Royal</W> <W C='NNP'>Highness</W> <W
C='NNP'>The</W> <W C='NNP'>Prince</W> <W C='IN'>of</W> <W
C='NNP'>Wales</W> <W C='CC'>and</W> <W
C='DT'>the</W> <W C='NNP'>President</W> <W C='IN'>of</W> <W
C='DT'>the</W> <W C='NNS'>People</W><W
C='POS'>'s</W> <W C='NNP'>Republic</W> <W C='IN'>of</W> <W
C='NNP'>China</W> <W C='('>(</W><W
C='NNP'>PRC</W><W C=')'>)</W> <W C='NNP'>HE</W> <W C='NNP'>Mr</W>
<W C='NNP'>Jiang</W> <W
C='NNP'>Zemin</W> <W C='DT'>both</W> <W C='NN'>spoke</W> <W
C='IN'>at</W> <W C='DT'>the</W> <W
C='NN'>ceremony</W><W C=','>,</W> <W C='WDT'>which</W> <W
C='VBD'>straddled</W> <W C='NN'>midnight</W> <W
C='IN'>of</W> <W C='NNP'>June</W> <W C='CD'>30</W> <W
C='CC'>and</W> <W C='NNP'>July</W> <W
C='CD'>1</W><W C='.'>.</W></S>
<S PAR='4' RSNT='1' SNO='4'><W C='DT'>The</W> <W
C='NN'>ceremony</W> <W C='VBD'>was</W> <W
C='VBN'>telecast</W> <W C='JJ'>live</W> <W C='IN'>around</W> <W
C='DT'>the</W> <W C='NN'>world</W><W
C='.'>.</W></S>
</TEXT>
</BODY>
</DOCPOS>
```

Figure 12: An example Docpos object

```
<?xml version='1.0'?>
<!DOCTYPE DOC-JUDGE SYSTEM '/clair4/mead/dtd/docjudge.dtd'>

<DOC-JUDGE QID='Q-2-E' SYSTEM='SMART' LANG='ENG'>
    <D DID='D-19981007_018.e' RANK='1' SCORE='9.0000' />
    <D DID='D-19980925_013.e' RANK='2' SCORE='8.0000' />
    <D DID='D-20000308_013.e' RANK='3' SCORE='7.0000' />
    <D DID='D-19990517_005.e' RANK='4' SCORE='6.0000' />
    <D DID='D-19981017_015.e' RANK='4' SCORE='6.0000' />
    <D DID='D-19990107_019.e' RANK='12' SCORE='5.0000' />
    <D DID='D-19990713_010.e' RANK='12' SCORE='5.0000' />
    <D DID='D-19991207_006.e' RANK='12' SCORE='5.0000' />
    <D DID='D-19990913_001.e' RANK='20' SCORE='4.0000' />
    <D DID='D-19980609_005.e' RANK='20' SCORE='4.0000' />
    <D DID='D-19990825_018.e' RANK='1962' SCORE='0.0000' />
    <D DID='D-19990924_047.e' RANK='1962' SCORE='0.0000' />
</DOC-JUDGE>
```

Figure 13: An example Docjudge object

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SENTALIGN SYSTEM "/clair4/mead/dtd/sentalign.dtd">
<SENTALIGN ENG="20000119_002.e" CHI="20000119_002.c" LANG="english-chinese">
<SENT ORDER="1" EDID="D-20000119_002.e" ESNO="1" CDID="D-20000119_002.c"
CSNO="1" />
<SENT ORDER="2" EDID="D-20000119_002.e" ESNO="2" CDID="D-20000119_002.c"
CSNO="2" />
<SENT ORDER="3" EDID="D-20000119_002.e" ESNO="3" CDID="D-20000119_002.c"
CSNO="3" />
<SENT ORDER="4" EDID="D-20000119_002.e" ESNO="4" CDID="D-20000119_002.c"
CSNO="4" />
<SENT ORDER="5" EDID="D-20000119_002.e" ESNO="5" CDID="D-20000119_002.c"
CSNO="5" />
<SENT ORDER="6" EDID="D-20000119_002.e" ESNO="6" CDID="D-20000119_002.c"
CSNO="5" />
</SENTALIGN>
```

Figure 14: An example Sentalign object

```
<?xml version='1.0'?>
<!DOCTYPE SENT-REL SYSTEM "/clair4/mead/dtd/sentrel.dtd" >
<SENT-REL>
<R SDID='47' SSENT='7' TDID='30' TSENT='29'>
<RELATION TYPE='19' JUDGE='1'/>
<RELATION TYPE='6' JUDGE='1'/>
</R>
</R>
<R SDID='81' SSENT='45' TDID='87' TSENT='10'>
<RELATION TYPE='19' JUDGE='1'/>
<RELATION TYPE='15' JUDGE='5'/>
</R>
<R SDID='63' SSENT='15' TDID='7' TSENT='11'>
<RELATION TYPE='12' JUDGE='4'/>
</R>
<R SDID='87' SSENT='24' TDID='81' TSENT='1'>
<RELATION TYPE='15' JUDGE='2'/>
</R>
<R SDID='47' SSENT='1' TDID='30' TSENT='24'>
<RELATION TYPE='19' JUDGE='1'/>
</R>
</SENT-REL>
```

Figure 15: An example Sentrel object

2. The mead-config object specifies the features that should be computed via the FEATURE-SET element. Each FEATURE subelement has NAME and SCRIPT attributes. The NAME attribute, not surprisingly, specifies the name of the feature. The SCRIPT attribute specifies the shell command that should be run to produce the feature.

   This second stage locates these features on disk by looking in the BASE-DIRECTORY attribute of the FEATURE-SET element in the mead-config object. A feature named "FeatureName" will be stored on disk in a file named "ClusterName.FeatureName.sentfeature" in the BASE-DIRECTORY directory. If this file exists, it is assumed to contain a feature named "FeatureName" for all and only the sentences in the cluster. If sentences in the cluster do not have entries in the sentfeature file, they effectively have value zero in subsequent calculations. On the other side of the coin, MEAD will not complain if a given sentfeature entry does not correspond to a sentence in a document specified by the cluster file. It may even be included in the extract of the cluster! However, errors may occur when producing a summary from the extract.

   If the file does not exist, driver.pl calls the script specified and redirects the output to the appropriate file, named as described above. Thus, if a certain cluster is summarized multiple times, the features are calculated the first time and are read from disk during subsequent summarization passes.

   If the user already is given sentence features, he may create sentfeature files manually. If the sentfeature files are named appropriately, they will be used just as if MEAD had created them during a previous summarization run.

   This "feature caching" is a double-edged sword. It speeds up the summarization process, sometimes considerably, but it may also result in erroneous summaries caused by inconsistencies in the contents of the cluster file and the sentfeature files. This commonly happens when documents are added to or removed from the cluster, or in query-based summarization, when using two different queries in summarizing the same cluster. More on the latter in the section on query-based summarization.

3. Next, driver.pl calls the classifier, which is specified via the CLASSIFIER element of the mead-config file. The COMMAND-LINE attribue specifies the script to run (with arguments) to produce sentence scores. These scores are for each individual sentence, and should not take into account inter-sentence relationships. The combined sentfeatures are written to the classifier's standard input, and the classifier must write a sentjudge object to its standard output. For more information on the workings of classifiers and the classifiers included with the MEAD distribution, see the section on classifiers.

4. The fourth task assigned to driver.pl is to call the reranker. As compared to the classifier, which assigns scores to individual sentences, the reranker modifies these scores depending on relationships between sentences. The input to the reranker is a reranker-info XML object. A reranker is an XML object made up of three smaller components: the compression information (words or sentences, plus how many), the cluster's DIDs, and the sentence scores output by the classifier in the form of a sentjudge. A reranker must output the modified sentence scores, again in sentjudge form. However, this sentjudge has the SENTS-FOR-SUM attribute in the top-level element. This is interpreted by driver.pl as meaning that it should output the SENTS-FOR-SUM top-scoring sentences as the extract.

5. The final job driver.pl does is to write out the exract, which is specified by the sentjudge written out by the reranker.

6.3 MEADlib

MEADlib is a collection of Perl modules used extensively by MEAD. They may be useful for related applications, especially those related to language processing. MEADlib can be used to evaluate extractive as well as abstractive summaries, segment text, read in files of many of MEAD's XML formats, convert between these formats, and write files of these formats.

MEADlib is a work in progress, so documentation is not included here. Any documentation will be linked off the MEAD web site.

7 Features and Feature Scripts

7.1 Introduction to MEAD Features

MEAD extractive summaries score sentences according to certain sentence features (hence the "sentfeature" object). The default classifier (default-classifier.pl) uses Position, Centroid, and Length, but MEAD features can potentially refer to any feature that a sentence has (how many named entities or anaphora it contains, for instance). The only stipulation that MEAD places on its features is that they be real-valued. However, the authors recommend that each feature be normalized to have a value between zero and one for each sentence. Although not technically necessary, it allows humans to more easily select appropriate weights for the linear combination used by the classifier to score sentences based on features.

In addition to the default features of Position, Centroid, and Length, the MEAD distribution contains several other potentially useful features including SimWithFirst, which computes the cosine similarity between a sentence and the first sentence in the document; IsLongestSentence, which assigns a score of one if the sentence is the longest in its document, and zero otherwise; and three query-based feature scripts: QueryCosine.pl, QueryCosineNoIDF.pl, and QueryWordOverlap.pl, which compute the various measures of similarity between the sentence and (potentially) three parts of the given query. See the Query-based Summarization section for more information on these last feature scripts.

In order to facilitate the creation and integration of new features, MEAD provides an interface to the feature calculation, and writing via the MEAD::SentFeature Perl module. This section describes the use of this library.

7.2 Sent-Feature Files

Sent-Feature files contain the values of features for each sentence. These are the output of all Feature Calculation scripts. An example sentfeature file is shown in an earlier section. A feature script that uses MEAD::SentFeature::extract_sentfeatures does not need to explicitly write Sent-Feature files; the library will do this for you. Essentially, the library function opens the cluster, and makes a series of callbacks to subroutines of the feature script. During these callbacks, the script can calculate the value of one or more features for each sentence and tell MEAD these values.

7.3 Three-Pass Feature Calculation

Feature Calculation is done in three stages: Cluster, Document, and Sentence. Each stage is implemented by a subroutine in the feature script corresponding to that stage. The cluster subroutine is called only once. Then the document subroutine is called once for each document in the cluster. Finally the sentence subroutine is called for each sentence in the cluster. All the information available at the sentence stage is available at each of the other stages. Feature calculation is done in this three-stage manner to save the user the time and aggravation of writing for-loops to iterate over the documents and sentences in the cluster.

The Cluster and Document stages are optional, and need not be implemented by the user. However, every feature script **must** use the Sentence stage, as this is the **only** time that a score can be assigned to a sentence.

Rather than confuse the reader by trying to explain how to use the MEAD::SentFeature::extract_sentfeatures library function to compute features, the author will refer the reader to an example feature script, Skeleton.pl, which lives in $SCRIPTS_DIR. An annotated version of this file is shown in Figure 16.

The key point to notice is the call to extract_sentfeatures several lines into the script. This routine will read the cluster filename (and the query filename, if given) from the standard input and open this (these) object(s).

The $datadir variable points to the directory containing the docsent files whose sentences you want to calculate the features for.

Note that the 'Sentence', 'Document', and 'Cluster' strings must appear verbatim (case-sensitive) as the key of the hash entry whose value is the reference to the corresponding subroutine.

7.3.1 The Cluster Stage

The Cluster routine is passed a cluster and do any needed processing using that cluster. This routine is called once per cluster.

- Clusters are references to hashes whose keys are DIDs (strings) and whose values are Documents.

```
#!/usr/bin/perl

use strict;

use FindBin;
# These are just the lib directories where MEAD's modules live.
use lib "$FindBin::Bin/../../lib", "$FindBin::Bin/../../lib/arch";

use MEAD::SentFeature;

my $datadir = shift;

extract_sentfeatures($datadir, {'Cluster' => \&cluster,
                                'Document' => \&document,
                                'Sentence' => \&sentence});

sub cluster {
    my $clusterref = shift;

    foreach my $did (keys %$clusterref) {
        # This cycling through the DIDs will produce each
        # document passed to the document subroutine.
        my $docref = $$clusterref{$did};
    }
}

sub document {
    my $docref = shift;

    for my $sentref (@$docref) {
        # This will produce each sentence in the document,
        # and because the document subroutine is called with
        # each document, it will produce every sentence in the
        # cluster.
        my $text = $$sentref{'TEXT'};
    }
}

sub sentence {
    my $feature_vector = shift;
    my $sentref = shift;

    my $did = $$sentref{"DID"};
    my $sno = $$sentref{"SNO"};
    my $text = $$sentref{"TEXT"};

    # You can compute more than one feature at a time,
    # but all but one may be "lost" as driver.pl looks for features
    # in files with names that include the feature name.
    $$feature_vector{"Skeleton"} = $sno/10;
    $$feature_vector{"Feature2"} = length($did);
}
```

Figure 16: An example Feature Script

- Documents are references to arrays of Sentences.
- Sentences are references to hashes whose keys are features names (strings) and whose values are the values of those features. The features passed to the feature script in the $sentref variable are:

  "TEXT" (string) The text of the sentence

  "DID" (string) The DID of the document to which the sentence belongs

  "SNO" (string) The number of the sentence in its document

7.3.2 The Document Stage

The Document routine is passed a Document and can do the desired processing (if any) using that Document. This routine is called once for each document in the cluster. Documents are references to arrays of Sentences (See above for a description of a Sentence).

7.3.3 The Sentence Stage

Sentence routines are passed two variables: A Sentence and a reference to a feature vector. Sentences are described in the "Cluster Stage" section above. Feature vectors are hashes whose keys are the names of features (strings) and whose values are the (real numbered) values of the features named by those strings. For example:

```
{'Centroid' => 0.2, 'Position' => 0.5}
```

After the Sentence routine has been called for every sentence in every document in the cluster, the extract_sentfeatures routine writes to standard output a sentfeature file containing the values for the features specified in the feature_vector for each sentence. (Subsequently, driver.pl redirects this output to the appropriate file.)

7.4 A Skeleton Feature Extraction Routine

As previously mentioned an example feature script, $SCRIPTS_DIR/Skeleton.pl, is including with the MEAD distribution. Note that the "use lib" statement near the top of Skeleton.pl must be modified to point to "$LIB_DIR" and "$LIB_DIR/arch". The example works as given only if the feature script is in $SCRIPTS_DIR.

The following command will write a sentfeature object to the standard output:

```
echo '$DATA_DIR/GA3/GA3.cluster' | $SCRIPTS_DIR/Skeleton.pl $DATA_DIR/GA3/docsent
```

8 MEAD Classifiers

8.1 The Default Classifier: default-classifier.pl

The classifier computes scores for each sentence.

- Input: A sentfeature file in XML format is written to the classifier's standard input. Every feature file specifies, for each sentence in a cluster, a set of features and a value for each feature.
- Output: A classifier must write a sentjudge file in XML format to its standard output. As previously described, a sentjudge for a cluster contains a a real number utility judgement for each sentence in that cluster. In the case of a classifier's output, this utility judgment is interpreted as the sentence's score.

8.1.1 Command Line Arguments

The COMMAND-LINE attribute of CLASSIFIER (in a mead-config file) should read

```
$BIN_DIR/default-classifier.pl feature1 weight1 feature2 weight2 ...
```

Each sentence receives a score that is a linear combination of the features listed (provided they are in the input feature file) EXCEPT for the "Length" feature. The weight of each feature in the linear combination is specified after each feature name. "Length", if it is given, is a cutoff feature. Any sentence with a length shorter than "Length" is automatically given a score of 0, regardless of its other features. "Length" is the only feature that has these semantics.

Thus, a COMMAND-LINE attribute of

```
$BIN_DIR/default-classifier.pl Centroid 2 Position 0.5 Length 12
```

has the following interpretation:

$$score(sentence) = \left\{ \begin{array}{rcl} 2 \cdot (Centroid) + 0.5 \cdot (Position) & : & Length(s) > 12 \\ 0 & : & Length(s) < 12 \end{array} \right\}$$

The default weights for Centroid and Position are both 1. The default Length cutoff is 9. The above example's weights were changed for illustrative purposes.

8.1.2 Using new features with default-classifier.pl

Using new features with default-classifier.pl is realtively easy. First, the desired feature must either be precomputed in sentfeature format in an appropriately-named and -located file **or** the user must have a feature script for computing the feature. Next, the user must add a FEATURE element to the FEATURE-SET in the mead-config file used as input to driver.pl. Alternately, mead.pl will do this for you: use the "-feature" command-line option or the "feature" .meadrc file option. See the appropriate sections for more information on these alternatives. Finally, the user must modify the COMMAND-LINE attribute of the CLASSIFIER element in the mead-config file. Again, mead.pl can also do this. Then run MEAD as usual using this modified mead-config file (or the arguments to mead.pl).

8.2 Other Classifiers

MEAD comes with a couple classifiers in addition to default-classifier.pl. These are random-classifier.pl and leadbased-classifier.pl, which are vital parts of the lead-based and random summarizers that are part of the MEAD distribution.

8.2.1 random-classifier.pl

The random-classifier.pl script is part of the random baseline summarizer. This classifier assigns a random score between 0 and 1 to each sentence. If a "Length" argument is provided, random-classifier will assign a score of 0 to all sentences that do not meet this length cutoff. Note that for the cutoff mechanism to work, the Length feature must be specified in the FEATURE-SET. Two valid COMMAND-LINE attributes of the CLASSIFIER element are:

```
$BIN_DIR/random-classifier.pl
```

and

```
$BIN_DIR/random-classifier.pl Length 10
```

8.2.2 leadbased-classifier.pl

The leadbased-classifier.pl script is part of the leadbased baseline summarizer. This classifier assigns a score of $1/n$ to each sentence, where n is the sentence's SNO in the corresponding docsent file. This means that the first sentence in each document will have the same scores, the second sentence in each document will have the same scores, etc. Again, if a "Length" feature argument is provided, the sentences with lengths less than the specified value are thrown out. Valid COMMAND-LINE attributes utilizing the leadbased-classifier.pl script are:

```
$BIN_DIR/leadbased-classifier.pl
```

and

```
$BIN_DIR/leadbased-classifier.pl Length 12
```

8.3 Creating a New Classifier

A classifier is a script that follows the covenant for MEAD classifiers (it doesn't even have to be written in Perl): it takes as input a sentfeature object, possibly with multiple features, does some processing, and outputs a sentjudge object with a score for each sentence in the cluster.

Figure 17 shows a bare-bones classifier that assigns a score of 0.5 to each sentence. Note that several portions of this example have been abbreviated in the interest of clarity and brevity. All of the classifiers that are packages with MEAD have this general form. The one thing that changes is the `compute_scores` function. For the actual bodies of the subroutines mentioned here, look in any of these scripts. A few things to notice about this script:

1. `%fnames` and `@all_sents`
   These are global variables that all functions can use. `%fnames` is a hash from feature names to feature weights to use in a linear combination. `@all_sents` is an array of sentrefs of the same form that are used in feature script callbacks.

2. The block of code marked by the "`# Execution Code`" comment need not change. The only thing that need change is the `compute_scores` subroutine. The following items detail the execution inside this block of code.

3. The `parse_options` subroutine builds the `%fnames` hash from the command-line options passed to the script as detailed in any of the classifier descriptions.

4. The next two lines, which call the `read_sentfeatures` and `flatten_cluster` from `MEAD::SentFeature` and `MEAD::Cluster`, respectively, read the sentfeature from the standard input and populate the `@all_sents` array.

5. The next call is key: `compute_scores`. Just about any classifier can be constructed by implementing this one sentence. See the MEAD classifiers to example `compute_scores` functions. The idea is to compute scores, probably but not necessarily based on the input features, and put each sentence's score into the sentref's "FinalScore" hash.

6. The final line of this block of code calls `write_sentjudge`, which writes out the required sentjudge object whose "UTIL" values are the values of each sentence's "FinalScore" hash value.

The framework given in Figure 17 has proven flexible enough for us to write any needed classification script. Although the user is not required to use this format when writing his own classifier, it may save much time and aggravation to do so.

9 MEAD Rerankers

9.1 The Default Reranker: default-reranker.pl

The reranker is used to modify sentence scores based on relationships between pairs of sentences. For example, it can be used to give lower scores to repeated instances of a sentence or higher scores to a sentence that has an anaphoric relationship with another sentence.

The input to a reranker is a reranker-info file A reranker-info file has three components: compression information, cluster information, and the sentence scores as computed by the reranker. The compression information has the same form as it does in the mead-config file: it specifies whether the BASIS should be "words" or "sentences", and how large the summary should be, either in comparison to the entire cluster (PERCENT) or as an absolute size (ABSOLUTE). The cluster information looks almost exactly like a cluster file, but without the XML headers. Rerankers may use this in order to open the cluster to examine and compare the text of each sentence. The sentence scores take the form of a sentjudge file.

The default reranker orders the sentences by score from highest to lowest, and iteratively decides whether to add each sentence to the summary or not. At each step, if the quota of words or sentences has not been filled, and the sentence is not too similar to any higher-scoring sentence already in the summary, the sentence in question is added to the summary. After the summary has been "filled," the default reranker increases the scores of the chosen sentences and decreases the scores of the disqualified (by similarity) or unchosen sentences.

```
#!/usr/bin/perl

use strict;

use FindBin;
use lib "$FindBin::Bin/../lib", "$FindBin::Bin/../lib/arch";

use XML::Writer;

use MEAD::Cluster;
use MEAD::SentFeature;

my %fnames = ();
my @all_sents = ();

# Execution Code.
{
    &parse_options();

    my %features = read_sentfeature();
    @all_sents = @{ flatten_cluster(\%features) };

    # Score the sentences based upon their weights
    &compute_scores();

    # Write out the new scores to a sentjudge file
    &write_sentjudge();
}

sub compute_scores {
    foreach my $sentref (@all_sents) {
        $$sentref('FinalScore') = 0.5;
    }
}

sub parse_options {
    # subroutine body omitted
}

sub write_sentjudge {
    # subroutine body omitted
}
```

Figure 17: An example classifier

Note that because the default reranker excludes sentences that are too similar to sentences already in the summary, that the user may not be able to get a 100% summary using the default parameters. A simple workaround is to set the similarity threshold to something greater than 1 or to simply use the identity reranker (see below).

The reranker, like the classifier, writes a sentjudge file to its standard output.

9.1.1 Command Line Arguments

The COMMAND-LINE attribute of RERANKER should resemble:

```
$BIN_DIR/default-reranker.pl SimFunction ThresholdValue IDFName
```

The SimFunction argument specifies the similarity function to compare sentences for similarity. Currently, the only supported similarity function is "MEAD-cosine". ThresholdValue specifies the maximum pairwise similarity (according to the named metric) that sentences in the summary can have. So if similarity between any two sentences is above the threshold, at most one of them will be included in the summary. The IDFName is the name of the IDF DBM to use in computing similarity. In the case of default-reranker.pl, it is used by the MEAD-cosine routine. An example COMMAND-LINE attribute follows.

```
$BIN_DIR/default-reranker.pl MEAD-cosine 0.7 enidf
```

This example says:

> When comparing sentences in the above fashion, use the MEAD-cosine similarity routine (and the enidf IDF database). If this routine returns a value greater than 0.7 for a pair of sentences, possibly include only the higher-scoring of the sentences in a summary.

9.1.2 Changing the parameters of default-reranker.pl

As previously stated, the only similarity function currently supported is MEAD-cosine. We have also empirically found that a cutoff of 0.7 disqualifies sentences that say roughly the same thing, but allows sentences that are on the same topic but say different things. The user can choose to use a different IDF database, but this will likely have little to no effect on the summary. See the section on Construction new IDF DBM's for instructions on how to do this.

9.2 Other Rerankers

The MEAD distribution comes with two rerankers in addition to default-reranker.pl.

9.2.1 identity-reranker.pl

The identity reranker does not modify the scores of sentences. It simply selects the appropriate number of sentences given the compression information passed to it. What is such a reranker useful? Since the reranker is in charge of selecting the number of sentences in a summary, identity-reranker.pl performs this task. Also, some types of summaries may not take inter-sentence relationships into account. This is true of the random and lead-based summarizers included with MEAD. In these cases, the identity reranker acts as a placeholder in the MEAD framework, which requires that summarizers be built from features, a classifier, and a reranker. The identity reranker takes no command-line arguments.

9.2.2 novelty-reranker.pl

The novelty reranker is the reranker we used as we participated in the Novelty Track in TREC 2002 (`http://trec.nist.gov`). For this competition, users were asked to identify sentences which contain new information, as sentences are passed sequentially through the system. We noticed that human judges often pick clusters of sentences, whereas MEAD normally does not care about the spatial relationships between sentences within a document. To exploit this hunch, we made a slight modification to the default-reranker.pl, boosting a sentence's score slightly if the previous sentence had a relatively high score. The difference between this script and the default reranker amounts to a single subroutine call during the script's execution. The novelty reranker takes exactly the same command-line arguments as the default reranker.

9.3 Creating a New Reranker

Creating a new reranker is a bit more complicated than creating a classifier. Our formula for rerankers may not work for all reranker functions. For this reason, we recommend thoroughly understanding two of the rerankers that come with MEAD: default-reranker.pl and identity-reranker.pl. The latter may serve as a jumping-off point for users wishing to write a reranker that doesn't exclude similar sentences, while the former may be extended by those who wish to do additional processing in addition to removing overly similar sentences.

However, we now briefly go over the workings of the default reranker in order to give the user a glimpse into how writing a new reranker might be done. Figure 18 shows annotated portions of default-reranker.pl minus some declarations at the top and boring subroutines at the bottom. We now go through each numbered part of the `# Execution Code` block of the script.

1. The first few lines parse the reranker-info input and fill in the global variables declare above the block shown.
2. This block decides whether to do percent- or absolute-based compression.
3. Along with block #5, this block lays the groundwork for ensuring that all chosen sentences have higher scores than non-chosen sentences. Here, we make sure that `$bonus_for_chosen_sentences` has a value higher than any sentence's score.
4. This block calls one of two methods, depending on whether we're using word-based or sentence-based summarization. The called method fills in an array called `@final_sents`, which after the call, contains all and only the sentences to be included in the summary.
5. Along with block #3, this block actually adds the bonus to the chosen sentences' scores, ensuring that chosen sentences have higher scores than unchosen sentences.
6. This writes the output sentjudge to the standard output. All sentences (and their scores) are written out, not just the chosen sentences.

We recommend that the reranker-writing user reuse at least the XML-reading and -writing code from the standard MEAD reranker scripts. Most changes can be made in the get_final_sents methods, which may make life easier for the user and for the MEAD team, when trying to interpret problematic output from the user.

10 MEAD Add-Ons

10.1 Additional scripts

MEAD has some additional scripts that do useful things. These scripts include:

- extract-to-summary.pl
  This script converts an extract object to a summary object. It opens the specifies cluster and attempts to fetch the text of each extracted sentence from the cluster. Use extract-to-summary.pl as follows:

  ```
  % ./extract-to-summary.pl cluster_file docsent_dir extract_file
  ```

  If you have $BIN_DIR in your path, and are in `$DATA_DIR/GA3`, you can say:

  ```
  % ./extract-to-summary.pl GA3.cluster ./docsent GA3.10.extract
  ```

- write-idf.pl
  This script builds an IDF database file in an architecture-specific format. See the Constructing IDF Databases subsection in Really Advanced MEAD for details.

```
#!/usr/bin/perl

# much code omitted...

# Execution Code
{
    # 1
    my $xml_parser = new XML::Parser(Handlers => {
        'Start' => \&read_rerankerinfo_handle_start,
        'End' => \&read_rerankerinfo_handle_end});
    $xml_parser->parse(\*STDIN);

    # 2
    if ($compression_absolute) {
        undef $compression_percent;
    } elsif ($compression_percent) {
        undef $compression_absolute;
    } else {
        $compression_percent = 20;
        Debug("Neither percent nor absolute specified; using 20%",
              3, "Main");
    }

    # 3
    my $bonus_for_chosen_sentences = 10000.0;
    foreach my $sentref (@all_sents) {
        if ($$sentref{'Score'} > $bonus_for_chosen_sentences) {
            $bonus_for_chosen_sentences += $$sentref{'Score'};
        }
    }

    # 4
    if ($compression_basis eq "sentences") {
        get_final_sents_by_sentences();
    } else { # $compression_basis eq "words"
        get_final_sents_by_words();
    }

    # 5
    foreach my $sentref (@final_sents) {
        $$sentref{'Score'} += $bonus_for_chosen_sentences;
    }

    # 6
    &write_sentjudge();
}

sub get_final_sents_by_sentences {
    # subroutine body omitted.
}
sub get_final_sents_by_words {
    # subroutine body omitted.
}

# Several subroutines to do XML parsing and writing
# have been omitted.
```

Figure 18: An example reranker

- make-CHIN-docsent.pl This script is used to build docsent files from Chinese text, a process that may cause problems if not done correctly. See the Summarizing Chinese Documents with MEAD section for details.

Many additional convenience-type scripts are planned. These scripts will include support for producing docsent and cluster files quickly and easily from a set of (possibly diverse) documents and document formats. Also, this functionality will be built into mead.pl, so the user can directly summarize any type of document or documents, e.g. text or HTML.

10.2 Pre-processing MEAD input and Post-processing MEAD output

MEAD can be used even when the input documents are not in docsent format and/or the desired output format is not extract format or MEAD's summary format. Most input formats will probably resemble docsent format, especially for extractive summarization tasks. Most formats have some idea of a document ID's (DID) and most number sentences (SNO), so the conversion between formats may be accomplished easily using an XSL engine, or can be done using parts of MEADlib. MEAD's output may be converted to an extract that contains the text either by using an XSL engine, or again by using MEAD::Extract and MEAD::Cluster, two pieces of MEADlib. MEADlib's documentation is only online at this point, so refer to the MEAD homepage for this information.

See below for the new add_on scripts which convert html and text files to docsent and cluster format.

11 Really Advanced MEAD

Some MEAD-related activities that should be in this section are so complicated that they each merit a section of their own. So the following two sections—on running MEAD in other languages (mainly Chinese), and using SVM to train MEAD—are effectively separate, but have strong ties to this section.

11.1 Producing Query-Based Summaries

Producing query-based summaries within the MEAD framework is as easy as specifying a modified set of features and modifying the classifier command, minus a caveat or two.

Three MEAD feature scripts: QueryCosine.pl, QueryCosineNoIDF.pl, and QueryWordOverap.pl, compute various sentence features in relation to the given query. Each of these feature scripts can compute up to three features, one for each of the three parts of a query object. For example, QueryCosine.pl can compute QueryTitleCosine, QueryDescriptionCosine, and QueryNarrativeCosine, which are the cosine similarity between each sentence and the title, description, and narrative of the query, respectively. QueryCosineNoIDF.pl and QueryWordOverlap.pl do much the same, but with different similarity metrics: `MEAD::Evaluation::simple_cosine` for the former, and `MEAD::Evaluation::unigram_overlap` for the latter.

The first argument to each of these scripts is a "-q" option, which takes **exactly** one of four options:

- t, title
  Compute only QueryTitleXXX.
- d, description
  Compute only QueryDescriptionXXX.
- n, narrative
  Compute only QueryNarrativeXXX.
- a, all
  Compute all three of the above features.

The second argument to each of these scripts is the name of the query file to use in the query-based summary.

Note that for use with MEAD, only the first three arguments to "-q" should be used. This is because MEAD looks for each feature in a separate file in the feature directory. A sample SCRIPT attribute of for one of these features is:

```
$SCRIPTS_DIR/QueryCosine.pl -q title $DATA_DIR/GA3/GA3.query
```

This will compute the QueryTitleCosine feature, so the NAME attribute should be "QueryTitleCosine" as well. Note that as with all MEAD feature scripts, the "datadir" argument will be appended to the command and the cluster filename will be echoed to the standard input. See the section on MEAD Features for more information.

Now to use the newly-computed feature, you must modify the CLASSIFIER COMMAND-LINE attribute in your mead-config file. The following example is a slight modification of MEAD's default feature weighting, adding "QueryTitleCosine" to the linear combination with weight 1.

```
$BIN_DIR/default-classifier.pl Length 9 Centroid 1 Position 1 \
    QueryTitleCosine 1
```

Then run MEAD as usual.

NOTE: The caveat mentioned is regarding MEAD's caching of features. If the user changes the query (or the contents of the cluster), he must delete the cluster's QueryXXX feature files as well.

11.2 Producing Alignment-Based Summaries

This was done in the JHU 2001 summer workshop. The idea behind this is that if you have documents which are translations of one another, if you have a summary in one language, you should just be able to create a summary in the other language by mapping each sentence from the summarized language to the other.

In the JHU 2001 summer workshop data, the objects which contained the mapping from Chinese sentences to English sentences were called sentaligns. These objects are described in the appendix. See section below on the JHU 2001 summer workshop.

11.3 Constructing IDF Databases

Half of the construction of new IDF databases is made easy for the user. The write-idf.pl script takes as input the name of the destination IDF DBM and the name of a plain text file, which has pairs of word/idf values. The script creates the DBM if it isn't there already, and puts each of the pairs into the hash.

The harder part of making IDF databases is actually computing the IDF value for each word in the corpus. Such a script is included with versions 3.06 and below, but this second script has been removed for version 3.07, as it only works in very specific situations. It may be rewritten and included in future versions of MEAD. In the meantime, writing a script that computes IDF for a large corpus is not very taxing. Besides, the default IDF databases included with the MEAD distribution work reasonably well.

11.4 Creating Docjudges

The docjudge object is explained in the subsection entitled "XML Objects not used by core MEAD." Creating this object should be straightforward with an XML module available from CPAN or even with a homemade procedural script.

12 Creating baselines for use in evaluation

12.1 Lead-based and Random

These are easily implemented. As mentioned, just add the -LEADBASED or -RANDOM flag to your MEAD commandline:

```
mead.pl -LEADBASED GA
mead.pl -RANDOM GA
```

12.2 Extracts from Sentjudges

The script to run this is sentjudge-to-extract.pl in mead/bin. As the default, this creates a 10 sentence extract based on all the judges' scores.

The commandline is:

```
./sentjudge-to-extract.pl anysentjudgefile.sentjudge
```

The user can adjust the length of the extract with -percent, -p or -absolute, -a.

If you wanted a 12 sentence extract, type,

```
./sentjudge-to-extract.pl -a 12 anysentjudgefile.sentjudge
```

Further, the user has the ability to create an extract based on only one judge's judgments. Let's say you want a 15 sentence extract based on the judgments of "bill," ust type,

```
./sentjudge-to-extract.pl -a 15 -j bill anysentjudgefile.sentjudge
```

12.3 Creating Sentjudges from manual summaries

For DUC 2002 and 2003, we developed a way of creating sentjudges from manual summaries. In other words, given the manual summaries, we create relevance scores (in sentjudge format) for each sentence in the cluster. These sentjudges can then be used to create a "manual" extract, which could be used in evaluation. We have two scripts to accomplish this task, one designed for DUC 2002, MUsentjudge.pl, and one for DUC 2003, MUsentjudge-2003.pl.

As the scripts are currently configured, the directories that it needs are hard-coded into the script. You will have to change these. The input directories are: the cluster directory and the model unit directory. The output directory is the sentjudge directory. You will also have to set the path to the idf database.

The commandline is:

`./MUsentjudge.pl clustername`

The cluster directory should have the format of the tradtional MEAD cluster directory. That is, there should be a `.cluster` file and a `docsent` directory in the cluster directory.

A model unit file looks like this:

```
<html>
<head>
<title>D061.M.050.J.B</title>
</head>
<body bgcolor="white">
<a name="1">[1]</a> <a href="#1" id=1>Hurricane \
Gilbert, a category 5 storm, caused death, massive \
flooding and damage</a>
<a name="2">[2]</a> <a href="#2" id=2>as it moved through \
the Caribbean Islands and on to the Yucatan Peninsula.</a>
<a name="3">[3]</a> <a href="#3" id=3>After skirting \
several island nations,</a>
<a name="4">[4]</a> <a href="#4" id=4>it caused major \
death and destruction in Jamaica.</a>
<a name="5">[5]</a> <a href="#5" id=5>It then pummeled \
the Yucatan Peninsula before moving out to sea.</a>
</body>
</html>
```

Figure 19: An example DUC Model Unit document

13 Evaluation

There are many ways of evaluating summaries. We have included in the MEAD distribution a toolkit called MEAD eval. We have also included a number of other evaluation metrics in the MEAD add-ons distribution, Relevance Correlation, and some content-based evaluations.

13.1 Evaluation using MEAD Eval

An old version of the MEAD Eval toolkit (which implements precision, recall, kappa, cosine, unigram and bigram overlap, and relative utility) is available at `http://tangra.si.umich.edu/clair/meadeval`.

As of MEAD 3.07, MEAD Eval is integrated with the core MEAD distribution. However, some functionality in the standalone version of MEAD Eval (namely, content-based similarity metrics) has not been included in the MEAD Eval scripts: meadeval.pl and relative-utility.pl. However, this code still lives in MEADlib in MEAD::Evaluation module. User code that uses this module in the standalone version of MEAD Eval should be able to use the same module in MEAD v3.07. See the online documentation for the MEAD Eval API.

The meadeval.pl script, given two extract filenames as input, calculates three co-selection metrics and prints them to the standard output.

Thus, a sample call to MEAD Eval is:

```
% ./meadeval.pl $DATA_DIR/GA3/GA3.10.extract $DATA_DIR/GA3/GA3.20.extract
```

Although the output of this example is rather uninteresting (as the 20% extract contains all the sentences in the 10% extract), it shows how meadeval.pl is used. Take a look at the code of this script to see how one might call the content-based metrics in MEAD Eval.

Relative Utility (RU) has its own script: relative-utility.pl. In order to use RU, you must have a sentjudge file for the cluster you're evaluating. Sentjudge information is expensive, hence rare, so the use of RU may be limited to the few clusters that have sentjudge information already computed. Regardless, the script is used as follows:

```
% ./relative-utility.pl $DATA_DIR/GA3/GA3.10.extract \
    $DATA_DIR/GA3/GA3.sentjudge
```

The above command prints sentjudge and relative-utility information to the standard output. Most of the guts of RU is actually implemented in the MEAD::SentJudge module of MEADlib. See this section of the online documentation for more info.

Finally, in the new MEAD addons package, we have included two scripts written for the JHU 2001 summer workshop. These allow the user to calculate kappa for 3 or 4 judges. One must first convert sentjudge files to a tab file with read_multi_into_kappatab.pl, and then one runs kappa.pl on this new file.

Before using these, it is important to edit read_multi_into_kappatab.pl. You must properly set $mandirand$autodir to the directories which contain the automatic and manual extracts.

There are three arguments for read_multi_into_kappatab.pl: mode ("H" for human or "C" human+computer), cluster (one of the ldc clusters) and length (length of the extract, in the terms of the workshop, these can be 50, 50W, 50S, etc.; see the LDC/JHU documenation for an explanation of these).

So, the commandline for read_multi_into_kappatab.pl is this:

```
./read_multi_into_kappatab.pl H 1014 50
```

This creates a file called `kappatab.H.199.050`. Then you can pipe this into kappa.pl with:

```
cat kappatab.H.199.050 | ./kappa.pl
```

13.2 Evaluation using Relevance Correlation

One of the evaluation metrics used in the JHU 2001 summer workshop on document summarization is based on principles from Information Retrieval. For their project, they used the SMART information retrieval system. See section 3.2 of their report for an explanation of how that system works. Their report is available at `http://www.clsp.jhu.edu/ws2001/groups/asmd/`.

The idea behind using IR as an evaluation metric for summaries is the following. We assume that good summaries of a set of documents should be ranked in the same order as the original documents. So, we used

SMART to rank a set of full documents for a query. We created summaries of each of those documents, and we had SMART rank the summaries. We then used two statistical methods for comparing the rankings, Pearson product moment and Spearman's rank correlation coefficient. These scripts are in evaluation/ir_based.

We have included examples from the summer workshop in the directory data/docjudges.

The object we used to record the document rankings is called a docjudge. For the JHU summer workshop, we also included information about the corresponding document in the other language

```
<?xml version='1.0'?>
<!DOCTYPE DOC-JUDGE SYSTEM "/export/ws01summ/dtd/docjudge.dtd" >
<DOC-JUDGE QID="Q-2-E" SYSTEM="SMART" LANG="ENG">
<D DID="D-19981007\_018.e" RANK="1" SCORE="34.0000" CORR-DOC="D-19981007\_023.c"/>
<D DID="D-19980925\_013.e" RANK="2" SCORE="33.0000" CORR-DOC="D-19980925\_015.c"/>
etc.
</DOC-JUDGE>
```

Figure 20: An example Docjudge object

We have included 3 docjudges for one query (125). If we want to see whether MEAD is doing a better job than Random, we can measure how well MEAD's summaries' rankings correlate with the rankings of the full documents, and then we can measure how well random summaries' rankings correlate with the rankings of the full documents.

So, for example, to compute the Pearson product moment of the rankings of the full documents with the 20% (sentence-based) MEAD summaries, we would do the following.

The correlation scripts are easy to run.

```
./pearson.pl docjudgeA docjudgeB

./spearman.pl docjudgeA docjudgeB
```

If the two docjudge files were in the same directory as the evaluation scripts, the command would be:

```
./pearson.pl mead-Q00125.docjudge fulldoc.docjudge
```

Typing in the paths is a bit ugly, but not too bad:

```
./pearson.pl ../../data/docjudges/mead-Q00125.docjudge \
   ../../data/docjudges/fulldoc.docjudge}
```

And, we find that the Pearson product moment is 0.91, when we compare the rankings of the full documents with the rankings of MEAD's 20% summaries. However, when we create random 20% summaries, rank them with query 125, and compare those rankings with the rankings of the full documents, we find a Pearson product moment of 0.78.

13.3 Evaluation using Lexical Similarity

We have included this functionality in the new addons (3.08) distribution.

Before beginning, you must have installed MEAD and have mead/lib set as a perl library location if it isn't already. If you don't know how do to this, adapt the following line appropriately:

```
export PERL5LIB=\$PATH:/clair4/projects/mead307/stable/mead/lib
```

The other thing you must do is to open summ_sim.pl and change the directory for the location of "bleu-1.pl" in the variable $bleu_DIR. The line currently reads: `my $bleu_DIR = "/clair4/tools/bleu";`

The commandline is:

```
./summ_sim.pl summary1 summary2
```

```
a) Simple Cosine: 0.310086836473021
b) Cosine       : 0.15299879834494
c) Token Overlap: 0.182926829268293
d) Big.  Overlap: 0.108910891089109
e) Norm. LCS    : 0.275
f) Bleu         : 0.1800
```

Figure 21: Sample output from summ_sim.pl

The output looks like this:

Simple cosine calculates the cosine similarity with a simple binary count (1 if a word exists (no matter how many times) in a sentence, 0 if it doesn't). Cosine uses idf weights and includes the actual count of tokens for each type. In "The quick brown fox jumped over the lazy dog," simple cosine would only count "the" as 1, while cosine would count it twice and multiply it by its idf weight.

Big. Overlap measures the bigram overlap, and norm. LCS measures the normalized longest common substring.

The input for this script can be a summary of this format:

```
[1]  Solemn ceremony marks Handover
[2]  A solemn, historic ceremony has marked the resumption of the exercise
of sovereignty over Hong Kong by the People's Republic of China.
[3]  His Royal Highness The Prince of Wales and the President of the
People's Republic of China (PRC) HE Mr Jiang Zemin both spoke at the
ceremony, which straddled midnight of June 30 and July 1.
[4]  The ceremony was telecast live around the world.
```

Figure 22: Sample summary

or a "document" file of this format:

```
<?xml version='1.0'?>
<!DOCTYPE DOCUMENT SYSTEM "documentation/dtd/document.dtd">
<DOCUMENT DID='19970701_001.e' DOCNO ='1' LANG='ENG' >
<EXTRACTION-INFO SYSTEM="LEAD-BASE" RUN="D-19970701_001.e.996941177" \
COMPRESSION="20S" QID="none"/><BODY>
<TEXT>
 Solemn ceremony marks Handover
A solemn, historic ceremony has marked the resumption of the exercise
of sovereignty over Hong Kong by the People's Republic of China.
His Royal Highness The Prince of Wales and the President of the People's
Republic of China (PRC) HE Mr Jiang Zemin both spoke at the ceremony,
which straddled midnight of June 30 and July 1.
The ceremony was telecast live around the world.
</TEXT>
</BODY>
</DOCUMENT>
```

Figure 23: An example Document object

The content-based method of evaluation is particularly useful when comparing automatically created summaries with human summaries.

14 CST in summarization and evaluation

We have included in the MEAD add-ons package several scripts for working with Cross-Document Structure Theory (CST) relations. On CST, see [Rad00] and [ZBGR02].

We encode sentence relations in a sentrel object, and we offer several CST utility scripts in sentrelutils. This is a brief example of the information stored in a sentrel object:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE TABLE SYSTEM "/clair4/projects/cst/gulfair/judgement/sentrel.dtd">
<TABLE>
<R SDID="16" SSENT="3" TDID="43" TSENT="8">
<RELATION TYPE="18" JUDGE="Z"/>
</R>
<R SDID="2" SSENT="4" TDID="23" TSENT="5">
<RELATION TYPE="15" JUDGE="Z"/>
</R>
<R SDID="16" SSENT="6" TDID="19" TSENT="10">
<RELATION TYPE="4" JUDGE="Z"/>
</R>
<R SDID="23" SSENT="13" TDID="2" TSENT="7">
<RELATION TYPE="18" JUDGE="Z"/>
</R>
</TABLE>
```

Figure 24: An example Sentrel object

The way to read this table is the following. In the first example, Judge Z believes that there is a CST relationship of type 18 (fulfillment) between source document 16, sentence number 3 and target document 43, sentence number 8. Note that this relationship is directional. The judge is declaring that 16.3 asserts the occurrence of an event predicted in 43.8.

See the next section for our use of CST relations in a new set of rerankers.

14.1 Interjudge agreement and directional interjudge agreement

There are two scripts for measuring this. The first measures interjudge agreement regardless of the directionality (ijagree.pl) and the second measures interjudge agreement and considers directionality.

The commandlines are straightforward:

```
./ijagree.pl sentrel1 sentrel2
```

and

```
./ijagree-dir.pl sentrel1 sentrel2
```

Given the example data we include in the mead_addons directory, try

```
./ijagree.pl examples/MI9-Zhu.sentrel examples/MI9-jahna.sentrel
```

Part of the output from this is:

43-12-25-20	S
43-36-23-11	F
43-3-23-16	F
1-3-19-8	P

Figure 25: Sample output from ijagree.pl

This means that document 43, sentence 12 has an S relationship with document 25, sentence 20. ijagree.pl reports three different relationships:

- F— fully agree
- P— Overlapping judgement
- S— One says something, the other nothing.

As mentioned ijaagree-dir.pl reports specifically on directionality of judgments. An example output from this script looks like:

```
16-4-2-2        R
16-4-2-3        R
16-2-25-20      R
21-22-43-45     R
```

Figure 26: Sample output from ijagree-dir.pl

This means that two judges disagreed in the directionality of the relationships between document 16, sentence 4 and document 2, sentence 2. This script ignores the specific CST relationship type.

14.2 Calculating connectivity

We have designed cstconnectivity.pl to calculate the CST connectivity of an extract

The syntax is `./cstconnectivity.pl extract sentrel`.

So for example, by typing:

```
./cstconnectivity.pl examples/gulfair3.extract.10 \
  examples/judgements.sentrel}
```

one can learn that the connectivity of the extract and the sentrel is 2.

14.3 Merging sentrel files

This is as straightforward as one would imagine: `reljoin.pl sentrelA sentrelB`.

14.4 Extracting only certain types of CST relations from an extract

The syntax for this is `subrelation.pl extract sentrel -rellist LabelList`, where LabelList is a list of CST labels, e.g. ”1,3,7.”

So for example, if you type:

```
./subrelation.pl examples/gulfair3.extract.10 \
   examples/judgements.sentrel -rellist 1,3,7}
```

a part of the output will read:

```
<R SDID="81" SSENT="42" TDID="87" TSENT="22">
<RELATION JUDGE="1" TYPE="1"></RELATION>
<RELATION JUDGE="5" TYPE="1"></RELATION>
<RELATION JUDGE="9" TYPE="1"></RELATION>
</R>
```

Figure 27: An example output from subrelation.pl

14.5 Calculating similarity with sentrels

We include in the sentrelutils directory of MEAD add_ons a script called sim.pl. This has two functions. First, it can take a cluster as input, and it will output the lexical similarity between all sentence pairs above a user-defined threshold. It has four ways of measuring sentence similarity, bleu, cosine, word overlap and longest common substring.

If the user provides a sentrel file, sim.pl will output precision and recall based on the lexical similarity and the information in the sentrel file.

15 New Rerankers in mead_addons

In the following, we present several new rerankers that we have developed for MEAD. Zhu Zhang wrote a number of rerankers for his work on Cross-document Structure Theory (CST), and although several of his rerankers involve CST relations, there are some which don't involve CST per se. For example, cst-rerankers 3 and 4 rerank articles based on the epoch time of the articles.

15.1 CST rerankers, generally speaking

The general syntax for these rerankers is:

```
mead.pl -p 2 -reranker "./cst-reranker1.pl \
  10000000000001/10000000000001.metadata \
  10000000000001/10000000000001.sentrel" 10000000000001
```

Inside the double quotes is the syntax of the reranker itself, which of course can be different, depending upon how the reranker is written.

15.2 Source-based rerankers

News articles come from different sources, which may not be equally reliable or interesting. Therefore the first pair of rerankers reflects source preference. Specifically, we arbitrarily arrange the news sources in the order of "CNN, ABC, MSNBC, USATODAY, FOX" and give each source an integer priority value $Delta_s$ according to its place in the list.

- In reranker 1, the score of each sentence s is *increased* by Δ_s(source(s)).
- In reranker 2, the score of each sentence s is *decreased* by Δ_s(source(s)).

Rerankers 7 and 8 are very similar to rerankers 1 and 2 respectively, except that the Δ_s values are normalized into the range between 0 and 1.

15.3 Time-based rerankers

The second set of rerankers reflects chronological ordering. Specifically, we define Δ_t as the epoch seconds corresponding to publication time of each article and use it to adjust sentence scores.

- In reranker 3, the score of each sentence s is *increased* by Δ_t(source(s)). Therefore more recent information is favored.
- In reranker 4, the score of each sentence s is *decreased* by Δ_t(source(s)).

Rerankers 9 and 10 are very similar to reranker 3 and 4 respectively, except that the Δ_t values are normalized into the range between 0 and 1.

15.4 CST-based rerankers

With the aim of better understanding the role CST can potentially play in the summarization process, we also built a set of CST-based rerankers that utilize human-labeled CST relationships in a relatively simple way. We view a document cluster as a graph and each sentence as a node in it. Two sentences are linked by arcs (for now only undirected) if they are CST-related. The number of arcs are determined by the instances of CST relationships between the sentence pair. For example, if judge 1 finds one, judge 2 finds three, and other judges find no relationship(s) between two sentences, there will be four arcs linking the corresponding nodes. With this graph structure, each node (sentence) has a degree as defined in graph theory, and denoted by Δ_c in our scenario.

- In reranker 5, the score of each sentence s *increased* by Δ_c(s).
- In reranker 6, the score of each sentence s *decreased* by Δ_c(s).

Rerankers 11 and 12 are very similar to reranker 5 and 6 respectively, except that the Δ_c values are normalized into the range between 0 and 1.

15.5 MMR rerankers

We also include an MMR reranker in our mead_addons package. For a discussion of MMR, see Carbonell and Goldstein [CG98]. The basic idea behind the use of MMR in summarization is that the user should be able to select how much of the information space should be included in the summary. On the one hand, a user may want a summary which focuses only on the central point or points but which has some redundancy. On the other hand, a user may want a summary which captures a broader view of the information (and is very low on redundancy). The main parameter in any MMR system is called lambda, which has a range of 0-1. If set at 1, the summary will capture the most central sentences, and it will be redundant. As the user decreases this number, the summary will include sentences which offer a more diverse collection of information.

The syntax for mmr-reranker.pl looks like this:
mmr-reranker.pl $lambda $sim-function $idf

- $lambda is the relevance-weight that is used in computing new scores
- $sim-function is the name of a MEAD function for computing sentence similarity
- $idf is the name of the IDF DBM to use when computing similarity

In general, the sim-function is MEAD-cosine and the idf is enidf.
So, the command line for a 40% summary of cluster 10000000000001 would be:

```
mead.pl -p 40 -reranker "./mmr-reranker.pl 1.0 MEAD-cosine enidf" \
  10000000000001
```

We have also included an mmr reranker that allows one to choose a summary rate based on the number of words. This reranker is called mmr-reranker-word.pl.

16 Converting HTML and text files to docsent and cluster format

We have included 3 scripts in the add_ons to MEAD which will convert HTML and text files to clusters. These scripts are located in the `formatting` directory. We are always working to improve these, so check out the MEAD webpage for newer versions. Depending on your needs, you should only need to use one of the scripts, so fret not.

16.1 Preliminaries

Before beginning, UTIL.pm must be at the same level as the script you want to use (or at least in your Perl library path), and the directory “etc” must be at the same level as the script you want to use. Further, you will need to set the absolute path of the dtd directory variable $DTD_DIR on line 18 of UTIL.pm

```
my $DTD_DIR ="/clair6/projects/ldc/documentation/dtd";
```

16.2 From text files to docsents and/or a MEAD cluster

The script for this is text2cluster.pl, and depending on your needs, it can be called in a number of different ways.

1. if you have a file of unsegmented text which you want to convert to a docsent file:

   ```
   ./text2cluster.pl FILE
   ```

2. if you have a directory of unsegmented text files which you want to convert to a full cluster:

   ```
   ./text2cluster.pl DIRECTORY
   ```

3. if you have a file of segmented text you want to convert to a docsent file, and your sentences are segmented with '\n\n':

   ```
   ./text2cluster.pl FILE '\n\n'
   ```

4. if you have a directory of segmented text files, which you want to convert to a full cluster, and your sentences are segmented by "the" (admittedly, not best choice ;))

   ```
   ./text2cluster.pl DIRECTORY '\bthe\b'
   ```

5. finally, if you are preprocessing files for CIDR and you just want to convert a number of text files to docsent files without creating a cluster, you can do two things.

   (a) create a cluster and just use the docsent folder

   (b) create a simple driver which will run text2cluster.pl on each of the individual files in your directory.

16.3 From HTML files to a cluster

As with text files, begin by putting the HTML files you want to convert in a directory, and then call html2cluster.pl on that directory. This does the same thing that text2cluster.pl does, except that it has a preprocessing step which extracts text from HTML. The commandline is the same and the results (ideally) should be the same.

We have also included html2text.pl, which converts HTML files to text files.

16.4 Caveats

See the README file in mead_addons/formatting for a description of how these scripts parse HTML and segment sentences. In general, these scripts have been optimized to handle news articles, and so the default values of the control variables are set to ignore a number of types of text, including short sentences and spans of text which do not end in a period, exclamation point or question mark.

Note especially the following variables in UTIL.pm: $min_words, $sentends, $final_straw, $split_on, and $html_or_body.

17 CIDR: How to create document clusters

CIDR is our document clustering system. CIDR takes as input a list of files and their paths and outputs those files into directories based on the files' similarities with each other.

17.1 How CIDR works

CIDR works with five parameters.

- cluster number. default: 0, range: any positive whole number. This represents the first cluster number. If you begin at 14, CIDR expects that clusters 1-13 already exist.
- threshold. default: .1, range: between 0 and 1. This value determines how similar documents must be to be placed in the same cluster.
- word decay. default: .01, range: between 0 and 1. This value determines how quickly cidr discounts words which appear often.
- keep threshold. default: 3, range: any positive whole number. This determines the minimum nidf value of words to be added to the centroid
- to keep. default: 10, range: any positive whole number. This determines the number of keywords to be kept in the centroid.

17.2 The details of running CIDR

To use CIDR, the following things are necessary:

1. nidf.dir and nidf.pag must be present in the target directory.
2. cidr.pl expects that the input files are in docsent format. Plain text files should also work, but we make no guarantees
3. cidr.pl should be run in the directory where you wish the clusters to be created.

The command-line for cidr.pl has some nuances.

Let's begin with the basics. If your file of document paths is named ALL and cidr.pl is at the same level, a typical call to CIDR looks like this: `cat ALL | ./cidr.pl`

This creates all of the clusters in that directory, which can get a bit ugly, so, practically speaking, it is better to create a directory below ALL and cidr.pl, and put the nidf.pag and nidf.dir files in that directory. Then just call cidr in this way: `cat ../ALL | ../cidr.pl`

The order of the parameters is as given above. If you want to change any of the default settings, you must include the default values for those parameters up to the parameter you want to change (going from left to right).

So, for example, if you wanted to have cidr run with a threshold of .4, a keep threshold of 2, and a decay of .5, you would have to 1) order them as they are above and 2) include the default "number of clusters," which is here 0. This would then be the commandline: `cat ../ALL | ../cidr.pl 0 .4 .5 2`

18 Running MEAD in Other Languages

MEAD can summarize Chinese clusters as well as English. We anticipate that it would not be hard to modify MEAD to summarize French or Spanish clusters, for example. One would only have to create an IDF database and change the idf environment variable in MEAD.

18.1 Summarizing Chinese Documents with MEAD

NOTE: the Chinese summarization functionality is effectively in an alpha stage of development. It may break at any time without warning. This example may not be exactly correct, but is reasonably close.

18.1.1 Preliminary Notes

We have provided routines for converting clusters of plain text Chinese documents into MEAD compatible data. The only stipulations we place on document formatting are as follows:

1. You should know the encoding of the documents you are working with. If you're not sure, a good rule of thumb is as follows:

   Simplified Chinese: GB2312

   Traditional Chinese: BIG5

2. All of the documents in the cluster should be encoded using the same standard (i.e. don't mix BIG5 and GB2312 documents).

3. The documents should be word-segmented. Note: We used the segmenter at `http://www.mandarintools.com` to segment the example. This segmenter is quite old, and we advise finding another one for best results.

18.1.2 List Format

If you wish MEAD to summarize your documents as a multidocument cluster, you should provide to us a file in the following format:

```
<pointer-to-file1>
<pointer-to-file2>
..
<pointer-to-filen>
```

18.1.3 GB18030 Compatability

As of the writing of this document, the glibc implementation of iconv, a library which converts among different encodings is NOT fully compatible with the latest encoding standard of the People's Republic of China (GB18030). This means that many documents (I find about 1/4 on xin hua wang) from up-to-date Chinese websites will crash the conversion routines.
NOTE: GB18030 is backwards compatible (all GB2312 encodings map to the same characters), so many documents that are actually encoded in GB18030 are labeled as GB2312 documents. If these documents contain a character which is undefined in GB2312, they will crash the conversion scripts.

18.1.4 System Compatibility

1. Linux: Fully compatible with GB2312. Compatible with SOME parts of GB18030.

2. Solaris 7 and below: I have NOT gotten these to work on GB2312. This will be addressed ASAP. To test your system try " `iconv -f gb2312 -t BIG5`".

3. Solaris 8 and above: I haven't had a chance to test these. Sun claims that Solaris 8 (02/02 patch) and above are fully compatible with GB18030.

18.1.5 Running The Example

This example is a two-article cluster from xin hua wang (The website of China's largest news agency). It discusses Taiwan's decision to use "Common pinyin". It is encoded in GB2312.

1. Go to $BIN_DIR.

   ```
   % cd $BIN_DIR
   ```

2. Run the conversion script.

```
<LINUX USERS>
% ./make-CHIN-docsent.pl CHIN-example/commonpy.list GB2312

<SOLARIS USERS>
% ./make-CHIN-docsent.pl CHIN-example/commonpy.list gb2312
```

3. Edit the mead-config file.

   You'll need to have the mead-config file in a directory that is writable by you (chances are that $BIN_DIR is not). The rest of this example assumes that the mead-config file is named `mead.config` and is located in the current directory.

```
<Change the cluster>
Replace TARGET="GA3" with TARGET="commonpy.list" in the
top-level MEAD-CONFIG element.

<Change the Language>
Replace LANG="ENG" with LANG="CHIN"
Replace "ENG" with CHIN" in the COMMAND-LINE attribute of
the Centroid FEATURE element.

<Change the IDF database>
Replace "enidf" with "cnidf" in the SCRIPT attribute of
the Centroid FEATURE element.
Replace "enidf" with "cnidf" in the COMMAND-LINE attribute of
the RERANKER element.
```

4. Run MEAD on the example.

   Once the segmented documents are converted to docsent format and a cluster file is produced (by the above commands), you are ready to run MEAD using driver.pl:

```
% cat mead.config | ./driver.pl > ../data/commonpy/commonpy.extract
% ./extract-to-summary.pl \
      ../data/commonpy/commonpy.list.cluster ../data/commonpy/docsent \
      ../data/commonpy/commonpy.extract > ./data/commonpy/commonpy.summary
```

19 Training MEAD using Support Vector Machines

This section describes the data format and instructions for training and evaluation for sentence extraction in MEAD using Support Vector Machines(SVM). Note that training MEAD amounts to little more than selecting feature weights for MEAD's classifier, thus any other toolkit that can learn a linear combination given training data can be used in place of SVM.

19.1 Data Format

The format of training, tuning (development), and testing data are similar. The format is also similar to the data format expected by the SVM package. Each data file contains cases or samples. Each sample corresponds to a sentence and its feature values. Each sample is described by one line of record with syntax as follows:

<class> <feature-id1>:<feature-value1> <feature-id2>:<feature-value2> ...

<class> can be 1 or -1 representing the corresponding sentence is included or not included in the sample summary.

<feature-idx> is an integer representing a feature id.

<feature-valuex> is a real number representing a feature value.

Therefore, each record contains those features and their corresponding values for a particular sentence. It also contains whether or not the sentence is included in the sample summary.

Note that the feature values should be normalized so that the values fall between 0 and 1.

19.2 Porting

- Make a directory, e.g. trainable_mead which will contain all the data files and SVM package.
- Download SVM package
- Copy svm_classify.c to replace the original svm_classify.c (save a backup of the original svm_classify.c as advised)
- Compile the SVM package
- Prepare the training, tuning (development), and testing data. Follow the data format described above. (Note that the feature values should be normalized.)

19.3 Training

%SVM/svm_learn -j <cost-parameter> <training.data> <learned-model>
where:
<cost-parameter>is a parameter by which training errors on positive examples outweight errors on negative examples (default 1)
<training.data> is the training data set
<learned-model> is the output learned model
e.g.
%SVM/svm_learn -j 5 training.data learned-model-j5

The above command invoke the training process using the training data (training.data) with cost parameter 5.

The output of the learned model is stored in the file learned-model-j5. This learned-model will be used in the tuning and evaluation stages

19.4 Tuning (Development)

% **SVM/svm_classify <train.dev.data> <learned-model>**
where:
<train.dev.data> is the tuning (development) data set
<learned-model> is the learned model obtained from the training stage
e.g.

% **SVM/svm_classify train.dev.data learned-model-j5**

This command invokes the classification process on the tuning data - train.dev.data using the learned model - learned-model-j5. The linear weights of each feature are displayed. The accuracy, precision, and recall metrics are also shown.

Typically, one will conduct training using different parameters such as different cost factors. Then invoke the classification process for each learned model. One can choose the desired model based on a particular metric such as recall.

19.5 Testing

% **SVM/svm_classify <testing.data> <learned-model>**
where:
<testing.data> is the testing data <learned-model> is the selected learned model after tuning
e.g.
% **SVM/svm_classify testing.data learned-model-j5**
This command invokes the classification process on the testing data - testing.data

20 JHU 2001 summer workshop and SummBank

As of the writing of this section, we are in the process of publishing data which was collected during the JHU 2001 summer workshop entitled, "Evaluation of Text Summarization in a Cross-lingual Information Retrieval Framework." For updates on this information, please see `http://www.summarization.com/summbank` and for the final report from that workshop, `http://www.clsp.jhu.edu/ws2001/groups/asmd`.

Eventually, we envision SummBank offering a collection of summaries which can be used as gold-standards in evaluation. For now, SummBank 1.0 offers a portion of the total amount of data collected during the JHU summer workshop.

- Forty bilingual (Chinese and English) news clusters with 10 documents each, all formatted to be used by MEAD. Many of the clusters include, sentjudges, sentrels and docjudges.
- Sentence alignment information from English to Chinese documents and vice versa.
- Human written summaries: for each cluster, 3 judges at 3 compression ratios.
- Single and multi-document extracts created from human sentjudge scores.
- Single and multi-document extracts created at numerous compression ratios by the following automatic methods:
  - Alignment-based
  - Greg Silber's Lexical Chain-based
  - Chin-Yew Lin's summarizer
  - MEAD
  - Websumm
  - Lead-based
  - Random
- Docjudges, as ranked by SMART, for full documents and summaries created by the above methods.
- Features: document features as calculated by the JHU2001 version of MEAD.

In total, there are 40 clusters (comprising 400 documents each in English and Chinese), 360 human-written multidocument summaries, 5520 retrievals and nearly 2 million extracts.

To get the most out of SummBank 1.0, it will be helpful to have the Hong Kong News Corpus (LDC2000T46) available as well.

21 The MEAD Web site

The MEAD project's new Web page is at the University of Michigan. All versions of MEAD can be downloaded there.

`http://www.summarization.com/mead`

The MEAD project also has an older Web page at Johns Hopkins University. MEAD v3.06 and below can be obtained there. Additionally, this web page contains information about the JHU Summer 2001 Workshop, the Automatic Summarization of Multiple Documents (ASMD) group, its participants, and their contact information.

`http://www.clsp.jhu.edu/ws2001/groups/asmd`

22 Frequently Asked Questions

22.1 Is additional MEAD-compatible data available?

All the data used at the JHU workshop will be released soon in conjunction with the Linguistic Data Consortium (LDC). Check the MEAD web page often, or better yet, subscribe to the MEAD mailing list. See below.

22.2 Is there a mailing list for MEAD?

You betcha. Visit the MEAD homepage for subscription information:

`http://www.summarization.com/mead`

22.3 Can I contribute to MEAD?

Sure. Please send mail to the MEAD mailing list:

mead@majordomo.si.umich.edu

22.4 Do I need a license to use MEAD?

Not for the moment. Once we are beyond the beta stage, we will look into this issue.

22.5 How can I get help?

Please refer to the MEAD homepage for help.

`http://www.summarization.com/mead`

If all else fails, send mail to the MEAD mailing list:

mead@majordomo.si.umich.edu

22.6 How can I make sure I understand the details of how MEAD works?

As an exercise, try implementing your own features (e.g., CueWord) or rerankers (e.g., MMR).

23 Demos

- NewsInEssence (`http://www.newsinessence.com`) NewsInEssence is a freely accessible news search and summarization system maintain by the CLAIR group at the University of Michigan's School of Information. The CLAIR group also has strong ties to the Department of Electrical Engineering and Computer Science.
- Online MEAD Demo (`http://tangra.si.umich.edu/clair/meaddemo`) The Online MEAD Demo is a web interface to an older version of MEAD. The user can summarize documents on his local machine, generic text, or the contents of a url (or a combination of all three).

24 Future Work

Much work has been put into MEAD, but much still remains to be done:

1. Finish moving repeated code to library modules, e.g. SimWithFirst feature script.
2. Sentjudge as the output of MEAD.
3. Build MEAD API.
4. Improve, e.g., MEAD::Extract::read_extract by returning not only the sentrefs, but also the SYSTEM, RUN, QID, etc. in a hash of things.
5. combine driver.pl and mead.pl, moving most functionality to the MEAD API.
6. pass datadir via stdin to feature scripts (and on to MEAD::SentFeature::extract_sentfeatures) along with cluster filename.
7. Add choice of policies on feature caching: recompute, don't save on disk, do plausibility check, etc.
8. Allow the user to specify that default features (Length, Centroid, Position) NOT be computed, perhaps using a '-nofeature' option in mead.pl
9. Differentiate between weighted features and cutoff features, perhaps with a 'Length-cut'-ish suffix.
10. Enhance support for Chinese (and other language) summaries in mead.pl. This should be relatively easy, as all we really have to do is segment in a language-dependent way, and create or use IDF databases for other languages.
11. Support summarization of plain text documents as input to mead.pl. Integrate preprocessing made available in recent addons release
12. Allow mead.pl to accept urls as input (MEAD could fetch the documents and summarize them). Again, integrate preprocessing available as standalone.
13. Make DTD's better, more consistent (with each other), and make things make sense.
14. Improve MEAD's robustness and error checking (check for errors in returned XML data from classifier, reranker, etc.)
15. Build suite of regression and unit tests for MEAD.
16. Add relevance correlation to MEAD Eval; incorporate scripts available in recent addons package.
17. Include evaluation of summaries (as opposed to extracts) with content-based measures on meadeval.pl; again, incorporate scripts available in recent addons package.
18. Improve Install.PL.
19. Finish this document, including making sure that all examples work (build automated tests).
20. Fill in cross-references in this document, instead of saying, e.g., "See the MEAD Architecture Section".
21. ... and for those that find recursion funny: Organize and prioritize this TODO list.

25 Credits

Research on MEAD was partially supported by NSF-IIS-ITR Grant 0082884 as well as NSF-IIS Grant 0097467, which included support from the Defense Advanced Research Projects Agency.
The following people have been involved in the development in one form or another.

- Dragomir Radev: MEAD 1.0 (2000)
- Sasha Blair-Goldensohn: MEAD 2.0 (Spring 2001)
- John Blitzer, Elliott Drabek, Arda Çelebi, Hong Qi, Dragomir Radev, Simone Teufel, Horacio Saggion, Wai Lam, Danyu Liu: MEAD 3.01-3.06 (Summer and Fall 2001)
- Arda Çelebi: Web site at Johns Hopkins and distribution.
- Michael Topper: Online MEAD Demo, documentation, and porting.
- Adam Winkel: MEAD 3.07, NewsInEssence, and documentation.
- Zhu Zhang: CST scripts (including CST-Rerankers) and documentation.
- Hong Qi: DUC scripts and documentation.
- Anna Osepayshvili: MMR-reranker and documentation.
- Ali Hakim: made CIDR more user-friendly and wrote documentation.
- Tim Allison: Preprocessing text and html files and documentation.
- Inderjeet Mani, Chin-Yew Lin: project affiliates.
- Fred Jelinek, Bill Byrne, Sanjeev Khudanpur, Laura Graham, Jacob Laderman: hosts of the Johns Hopkins 2001 Summer Workshop where MEAD 3.0 was developed.
- Stephanie Strassel, Chris Cieri, David Graff (all from LDC) - corpus creation and annotation.
- Ralph Weischedel, Regina Barzilay, David Day, Greg Silber, Dan Melamed, Sean Boisen: miscellaneous advice and resources.
- The MEAD beta testers, especially John Murdie, Hans van Halteren, Wessel Kraaij, and Tristan Miller.

References

[CG98] Jaime G. Carbonell and Jade Goldstein. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In Alistair Moffat and Justin Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336, Melbourne, Australia, 1998.

[Man01] Inderjeet Mani. *Automatic Summarization*. John Benjamins Publishing Company, Amsterdam/Philadephia, 2001.

[Rad00] Dragomir R. Radev. A Common Theory of Information Fusion from Multiple Text Sources, Step One: Cross-document Structure. In *Proceedings of the 1st Workshop on Discourse and Dialogue of the Association for Computational Linguistics*, Hong Kong, October 2000.

[RJB00] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. Centroid-Based Summarization of Multiple Documents: Sentence Extraction, Utility-Based Evaluation, and User Studies. In Udo Hahn, Chin-Yew Lin, Inderjeet Mani, and Dragomir R. Radev, editors, *Proceedings of the Workshop on Automatic Summarization at the 6th Applied Natural Language Processing Conference and the 1st Conference of the North American Chapter of the Association for Computational Linguistics*, Seattle, WA, April 2000.

[RTS+02] Dragomir R. Radev, Simone Teufel, Horacio Saggion, Wai Lam, John Blitzer, Arda Çelebi, Hong Qi, Elliott Drabek, and Danyu Liu. Evaluation of Text Summarization in a Cross-lingual Information Retrieval Framework. Technical report, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD, June 2002.

[ZBGR02] Zhu Zhang, Sasha Blair-Goldensohn, and Dragomir R. Radev. Towards CST-Enhanced Summarization. In *Proceedings of the 18th National Conference on Artificial Intelligence*, Edmonton, Alberta, August 2002.

A XML DTDs

A.1 cluster.dtd

```
<!ELEMENT CLUSTER (D)*>
<!ATTLIST CLUSTER
   LANG (CHIN|ENG) "ENG">

<!ELEMENT D EMPTY>
<!ATTLIST D
   DID   ID    #REQUIRED
   ORDER CDATA #IMPLIED>
```

A.2 docjudge.dtd

```
<!ELEMENT DOC-JUDGE (D)*>
<!ATTLIST DOC-JUDGE
     QID    CDATA #REQUIRED
     SYSTEM CDATA #REQUIRED
     LANG   (CHIN|ENG) "ENG">

<!-- LANG refers to the language of the retrieval process.
     Thus, it is the language of the documents.
     However, the original language of the query might be
     different.
     Look this up in QID. -->

<!ELEMENT D EMPTY>
<!ATTLIST D
     DID   ID     #REQUIRED
     RANK  CDATA  #IMPLIED
     CORR-DOC CDATA #IMPLIED
     SCORE CDATA  #REQUIRED>
```

A.3 docpos.dtd

```
<!--  DTD for POS tagged text  -->
<!ELEMENT DOCPOS (EXTRACTION-INFO?, BODY)>
<!ATTLIST DOCPOS
   DID   CDATA      #REQUIRED
   DOCNO CDATA      #IMPLIED
   LANG  (CHIN|ENG) "ENG"
   CORR-DOC CDATA   #IMPLIED>
   <!-- DID : documentid
       LANG: language    -->

<!ELEMENT EXTRACTION-INFO EMPTY>
<!ATTLIST EXTRACTION-INFO
    SYSTEM  CDATA  #REQUIRED
    RUN     CDATA  #IMPLIED
    COMPRESSION CDATA #REQUIRED
    QID     CDATA  #REQUIRED>

<!ELEMENT BODY (HEADLINE?,TEXT)>

<!ELEMENT HEADLINE (S)*>
<!ELEMENT TEXT (S)*>

<!ELEMENT S (W)*>
<!ATTLIST S
    PAR  CDATA #REQUIRED
    RSNT CDATA #REQUIRED
    SNO  CDATA #REQUIRED>
    <!-- PAR: paragraph no
         RSNT: relative sentence no (within paragraph)
         SNO: absolute sentence no -->

<!ELEMENT W (#PCDATA)>
<!ATTLIST W
    C   CDATA #REQUIRED
    L   CDATA #IMPLIED>

<!-- C is the POS category. L is the lemma -->
```

A.4 docsent.dtd

```
<!--  DTD for sentence-segmented text  -->
<!ELEMENT DOCSENT (EXTRACTION-INFO?, BODY)>
<!ATTLIST DOCSENT
   DID   CDATA      #REQUIRED
   DOCNO CDATA      #IMPLIED
   LANG  (CHIN|ENG) "ENG"
   CORR-DOC CDATA   #IMPLIED>
   <!-- DID : documentid
       LANG: language    -->

<!ELEMENT EXTRACTION-INFO EMPTY>
<!ATTLIST EXTRACTION-INFO
    SYSTEM  CDATA  #REQUIRED
    RUN     CDATA  #IMPLIED
    COMPRESSION CDATA #REQUIRED
    QID     CDATA  #REQUIRED>

<!ELEMENT BODY (HEADLINE?,TEXT)>

<!ELEMENT HEADLINE (S)*>
<!ELEMENT TEXT (S)*>

<!ELEMENT S (#PCDATA)>
<!ATTLIST S
    PAR  CDATA #REQUIRED
    RSNT CDATA #REQUIRED
    SNO  CDATA #REQUIRED>
    <!-- PAR: paragraph no
         RSNT: relative sentence no (within paragraph)
         SNO: absolute sentence no -->
```

A.5 document.dtd

```
<!--  DTD for original, non-segmented text  -->
<!ELEMENT DOCUMENT (EXTRACTION-INFO?, BODY)>
<!ATTLIST DOCUMENT
   DID   CDATA      #REQUIRED
   DOCNO   CDATA      #IMPLIED
   LANG  (CHIN|ENG) "ENG"
   CORR-DOC CDATA   #IMPLIED>
   <!-- DID : documentid
       LANG: language    -->

<!ELEMENT EXTRACTION-INFO EMPTY>
<!ATTLIST EXTRACTION-INFO
    SYSTEM  CDATA  #REQUIRED
    RUN     CDATA  #IMPLIED
    COMPRESSION CDATA #REQUIRED
    QID     CDATA  #REQUIRED>

<!ELEMENT BODY (HEADLINE?,TEXT)>

<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT TEXT (#PCDATA)>
```

A.6 extract.dtd

```
<!ELEMENT EXTRACT (S)*>
<!ATTLIST EXTRACT
   QID         CDATA #REQUIRED
   COMPRESSION CDATA #REQUIRED
   SYSTEM      CDATA #REQUIRED
   JUDGE       CDATA #IMPLIED
   JUDGENO     CDATA #IMPLIED
   RUN         CDATA #IMPLIED
   SENTS_TOTAL CDATA #IMPLIED
   WORDS_TOTAL CDATA #IMPLIED
   LANG        CDATA #REQUIRED>

<!ELEMENT S EMPTY>
<!ATTLIST S
   ORDER       CDATA #REQUIRED
   DID         CDATA #REQUIRED
   SNO         CDATA #IMPLIED
   PAR         CDATA #IMPLIED
   RSNT        CDATA #IMPLIED
   UTIL        CDATA #IMPLIED>
```

A.7 mead-config.dtd

```
<!ELEMENT MEAD-CONFIG (FEATURE-SET, CLASSIFIER, RERANKER,
COMPRESSION) >
<!ATTLIST MEAD-CONFIG
   LANG CDATA #REQUIRED
   CLUSTER-PATH CDATA #IMPLIED
   DATA-DIRECTORY CDATA #IMPLIED
   TARGET CDATA #IMPLIED >

<!ELEMENT FEATURE-SET (FEATURE*) >
   BASE-PATH CDATA #IMPLIED >

<!ELEMENT FEATURE EMPTY >
<!ATTLIST FEATURE
   FEATURE CDATA #REQUIRED >

<!ELEMENT CLASSIFIER EMPTY >
<!ATTLIST CLASSIFIER
   COMMAND-LINE CDATA #REQUIRED
   SYSTEM CDATA #IMPLIED
   RUN CDATA #IMPLIED >

<!ELEMENT RERANKER EMPTY >
<!ATTLIST RERANKER
   COMMAND-LINE CDATA #REQUIRED>

<!ELEMENT COMPRESSION EMPTY >
<!ATTLIST COMPRESSION
   BASIS (sentences|words) #REQUIRED
   PERCENT CDATA #IMPLIED
   ABSOLUTE CDATA #IMPLIED >
```

A.8 query.dtd

```
<!ELEMENT QUERY (TITLE,DESCRIPTION?,NARRATIVE?)>
<!ATTLIST QUERY
   QID  CDATA #REQUIRED
   QNO  CDATA #REQUIRED
   LANG (CHIN|ENG) "ENG"
   TRANSLATED (YES|NO) "NO"
   ORIGLANG (CHIN|ENG) "CHIN"
   TRANS-METHOD (AUTO|MAN) "AUTO">

<!-- QID: unique query no, eg. 125-CA or 125-E
     QNO: LDC query no for content, eg. 125
     LANG: of query
     TRANSLATED: is it an original query or not?
     ORIGLANG: If translated, from which language (from the other
     one, of course!)
     TRANS-METHOD: Automatically translated or manually? -->

<!ELEMENT TITLE       (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT NARRATIVE   (#PCDATA)>
```

A.9 reranker-info.dtd

```
<!-- DTD for input to rerankers -->
<!ELEMENT RERANKER-INFO (COMPRESSION, CLUSTER, SENT-JUDGE)

<!ELEMENT COMPRESSION EMPTY>
<!ATTLIST COMPRESSION
    PERCENT CDATA #REQUIRED
    BASIS   CDATA #REQUIRED>

<!ELEMENT CLUSTER (D)*>
<!ATTLIST CLUSTER
    LANG (CHIN|ENG) "ENG">

<!ELEMENT D EMPTY>
<!ATTLIST D
    DID   ID    #REQUIRED
    ORDER CDATA #IMPLIED>

<!ELEMENT SENT-JUDGE (S)*>
<!ATTLIST SENT-JUDGE
    QID  CDATA  #REQUIRED>

<!ELEMENT S (JUDGE)*>
<!ATTLIST S
    DID  CDATA #REQUIRED
    PAR  CDATA #REQUIRED
    RSNT CDATA #REQUIRED
    SNO  CDATA #REQUIRED>

<!ELEMENT JUDGE EMPTY>
<!ATTLIST JUDGE
    N    CDATA #REQUIRED
    UTIL CDATA #REQUIRED>
```

A.10 sentalign.dtd

```
<!ELEMENT SENTALIGN (SENT+)>
<!ATTLIST SENTALIGN
   ENG          CDATA #REQUIRED
   CHI          CDATA #REQUIRED
   LANG         CDATA #REQUIRED>

<!ELEMENT SENT EMPTY>
<!ATTLIST SENT
   ORDER        CDATA #REQUIRED
   EDID         CDATA #REQUIRED
   ESNO         CDATA #REQUIRED
   CDID         CDATA #REQUIRED
   CSNO         CDATA #REQUIRED>

<!-- ORDER: the pairwise number
     EDID:  english document name
     ESNO:  english sentence number
     CDID:  chinese document name
     CSNO:  chinese sentence number -->
```

A.11 sentfeature.dtd

```
<!ELEMENT SENT-FEATURE (S)*>

<!ELEMENT S (FEATURE)*>
<!ATTLIST S
    DID  CDATA #REQUIRED
    SNO  CDATA #REQUIRED>

<!ELEMENT FEATURE EMPTY>
<!ATTLIST FEATURE
    N CDATA #REQUIRED
    V CDATA #REQUIRED>
```

A.12 sentjudge.dtd

```
<!ELEMENT SENT-JUDGE (S)*>
<!ATTLIST SENT-JUDGE
    QID  CDATA  #REQUIRED>

<!ELEMENT S (JUDGE)*>
<!ATTLIST S
    DID  CDATA #REQUIRED
    PAR  CDATA #REQUIRED
    RSNT CDATA #REQUIRED
    SNO  CDATA #REQUIRED>

<!ELEMENT JUDGE EMPTY>
<!ATTLIST JUDGE
    N    CDATA #REQUIRED
    UTIL CDATA #REQUIRED>
```

A.13 sentrel.dtd

```
<!ELEMENT SENT-REL (R)*>

<!ELEMENT R (RELATION)*>
<!ATTLIST R
   SDID         CDATA   #REQUIRED
   SSENT        CDATA   #REQUIRED
   TDID         CDATA   #REQUIRED
   TSENT        CDATA   #REQUIRED>

<!ELEMENT RELATION EMPTY>
<!ATTLIST RELATION
   TYPE         CDATA   #REQUIRED
   JUDGE        CDATA   #REQUIRED>
```