DCA 0.200 User Guide

NICTA Technical Report NICTA-SML-09-004
Statistical Machine Learning Group, Canberra
July 2009

Wray Buntine[*]
NICTA and Australian National University
Locked Bag 8001, Canberra, 2601, ACT Australia
`wray.buntine@nicta.com.au`

Abstract

This report is a user guide for the *Discrete Component Analysis* (DCA) software in development, Version 0.200. It shows how to use the software, and should be read in conjunction with the worked examples in the `examples/` directory.

∗. Also a fellow at Helsinki Institute of IT.

Contents

1. Introduction

This report is a user guide for the *Discrete Component Analysis* (DCA) software in development, Version 0.200. It shows how to use the software. For details on development and installation, please refer to the companion document "DCA: Discrete Component Analysis Software". For details on the theory underlying the software, please refer to the companion report "An Approach to Hierarchical Discrete Component Analysis". Both are included in the documentation directory `doc/` in the release.

1.1 What is it?

The *Discrete Component Analysis* (DCA) software is being developed as a stand-alone package, and as a plug-in to the Python-based *Elefant* system[1], a machine learning toolbox from NICTA. Currently the software is being run in stand-alone mode using the data streaming libraries from the older and now unsupported *MPCA* system[2]. The software itself is written in the C language and compiles on a Linux and a Mac OS X environment.

The models presented here are known under many names [BJ06], such as latent Dirichlet allocation (LDA), multi-aspect models, multinomial PCA, etc.

The main features of the current system are:

Gamma-Poisson models as well: Both Dirichlet-Multinomial and Gamma-Poisson models are supported.

Multi-threaded: POSIX threads can be included at compile time, and the multiple threads will run. This achieves a reasonable speed-up and can be used on 2-4 CPU systems.

Test likelihoods: State of the art unbiased methods for estimating likelihood on test sets are included (with the "-X" option). These also work with threads achieving near 100% speedup even with 8 threads on an 8 CPU system.

Hyper-parameter fitting: The $\vec{\alpha}$ parameters for the Dirichlet or Gamma component priors can be fit during the Gibbs sampling, as can the $\vec{\gamma}$ priors for the Word×Topic model. This is an optimisation, so it works well for the $\vec{\alpha}$ parameters, but the $\vec{\gamma}$ priors have a high dimension so optimisation is statistically suspect.

No hierarchical support, yet: A hierarchical extension was developed and tested, and corresponds most closely to the hierarchical Pachinko allocation models of [MLM07]. *This version, 0.200, thus has extensive support for hierarchical models in the code but it is currently disabled at the top level (in the Gibbs sampling routines).* The system was rebuilt to incorporate optimisations such as sparse vector handling and threads, and the hierarchical support disabled while this was done.

2. Available Information

All sorts of useful documents and demonstrations are retained in the release directory and not installed. Thus, after installation, keep the release there to access the following:

doc/ the documentation directory contains four PDF files with various documentation.

examples/ the examples directory is explained in a later section of this document. It contains several worked examples illustrating different ways to feed data into `DCA`.

tests/ the tests directory contains simple tests with known answers. Use this for a basic system test.

3. Input and Output Files

The system manages a number of data files as input and output to the system. There are two main sets of files, those for the bags of tokens that represent documents, managed by the `Data` class, whose interface is documented in "datalib/data.h", those for general variables, hyper-parameters and latent variables of the model, manipulated by the classes in the "hierlib/" directory, and by "utillib/drt.h" and "utillib/imtx.h" files.

1. `http://elefant.developer.nicta.com.au/`
2. `http://www.componentanalysis.org`

This documentation currently only documents the basic aspects of the system. Fairly extensive customisation of parameters and Gibbs sampling exists.

3.1 Document data

Document data is managed by the `mpdata` programme. Data is input in text files giving the sparse vectors that represent each document (or record). Data is assumed to be tokenised already, which means tokens are represented as integers from 0 to $J-1$ where J is the maximum number of tokens, and documents are indexed likewise. All data and its various supporting files are stored using file suffices such as ".bag", ".tokens", etc.

Data can be input in bagged format, or as a sequence of tokens, and is stored in the ".txtbag" file. Each document is written on a single line, with integers space separated. The file has the count of documents and the count of tokens/words prepared on separate lines. So the sequential format is

```
300
2000
3 78 263 1 409 78
2 2 1009 2 901 3 901
...
```

This means there are 300 documents to follow (and thus 302 lines in the file), with 2000 different tokens. This first document has tokens "3 78 263 1 409 78". The bagged format changes each dcument line.

```
300
2000
5 3 1 78 2 263 1 1 1 409 1
4 2 3 1009 1 901 2 3 1
...
```

The first integer in the document line gives the count of tokens to follow. Following are pairs giving token index and its count. Thus the first document has 5 different tokens, with indices "3 78 263 1 409", and only "78" has a count of 2.

One can include with the inputs a list of tokens, one per line, in the ".tokens" file, and a list of titles for documents, one per line, in the ".titles" file. The bags can be subsequently mapped (*e.g.*, to eliminate rare or frequent words, empty documents, etc.), and these token ad title files will be mapped as well.

The two different data formats are then "bagged" into an efficient binary encoding, using the `mpdata` programme. This produces a ".bag" file, which generally achieves compression rates better than `bzip2`, and an index file ".iinx" that by default only stores every 10th document location in the ".bag" file. The index allows efficient random access of document bags. Other files produced are a word count file, ".wc" and parameter file ".par".

The `mpdata` programme can also print bags, remap bags be converting indices and documents. Its options are given in the appendix.

3.2 Other data formats

A number of other data formats are supported by conversion scripts.

Wikipedia and Reuters data: the `examples/` directory contains worked examples and scripts on how to convert data from the Wikipedia and Reuters Corpus Volume 1 to a format for `DCA`.

The UCI data format: The UCI depository has a number of data sets available for topic modelling[3] including the "nips," "enron," and "nytimes" sets. These are in a sparse data format with an accompanying dictionary.

In the UCI format, documents are numbered from 1, and words are numbered from 1. The words/tokens are listed in a separate file, given the file stem `STEM`:

vocab.STEM.txt : lists the vocabulary, one entry per line, which is then accessed numbered from 1.

docwords.STEM.txt : gives the dimensions on three lines and the sparse matrix in the format of one document+word entry per line.

3. `http://archive.ics.uci.edu/ml/datasets/Bag+of+Words`

```
doc-count
word-count
total-words
(doc word count ) ...
```

To convert these to the sparse format for DCA use the conversion script `docword2bag` available in the DCA-Bag Perl scripts. Thus the previous small example becomes:

```
300
2000
1476298
1 3 1
1 78 2
1 263 1
1 1 1
1 409 1
2 2 3
2 1009 1
2 901 2
2 3 1
...
```

3.3 Model data

The `mphier` programme builds DCA models from data, and also allows restarting from previously saved models. The `mpupd` programme produces simple reports and extracts auxilliary data.

The input to the `mphier` programme is the parameter file with ".par" suffix. Parameters are documented in the `mphier` man page. Here is an example:

```
#
#     This file documents all the input parameters for .par files.
#     Currently its bare bones ... just minimum necessary to get started.
#

#   file stem for your bag file, path *must* be
#   relative to where the programme is called from
datastem="PATH/datastem"

#  dimensions read from the document bag file, see PATH/datastem.par
documents=50182
features=28251
mean_featurecount=150.97485

#  the last contigious records in the bag can be for testing,
#  give the count here if so
testdocs=5000

#  count of components to use
components=100

#  Set this for Gamma components instead of Dirichlet
# alpha.fullgamma
#  Set this for conditional Gamma components (still experimental)
#      WARNING:  requires special attention with alpha_update
# alpha.conditional

# if you want to initialise component alphas, do it here
#   this says to set all values to 0.1
alpha.alpha=0.1+
```

```
# But, if you have patience, list all hundred out, using multiple
# lines of needed
# alpha.alpha=0.1,0.1,0.1,....
#    likewise for betas, if using Gamma components
# alpha.beta=0.1+
#    likewise for rate (rate of zeros) if using conditional Gamma
# alpha.rate=0.1+

# if fitting hyper-parameters for alpha component priors, use this
# WARNING:   fitting of conditional Gamma's so far only works
#            if the model is fixed, so first run with
#            "alpha.fullgamma", then add "conditional" to STEM.alpha
#            and refit using "mphier -n -r"
#    flag to switch on fitting of hyper-parameters
# alpha_update
#    only change every so many cycles
gamma_cyclefactor=2
#    only start fitting after this burnin
gamma_burnin=10

#  the constant prior term for the Theta matrix (topic by word)
#  should be constant, cannot be fit as yet
thetaprior=0.001
#   this sets the prior proportionally to observed frequencies
#   with total given by "thetaprior"
# thetaempiricalprior
```

The programme produces a number of files as output. Some of these are in the `Pam` text parameter file format, some are float matrices, and topic assignments for words in documents are in a special `IMtx_t` format that allows efficient streaming.

These files are:

".comps": This is the major output and is the topic assignments for words in documents (denoted by $\vec{k}$ in the theory papers) are in a special `IMtx_t` format. This file is typically the same size as the ".bag" file storing the document data. The topic by word matrix (the major model output) is derived from this using "`mpupd -T`". These are accessed sequentially by the `hdoc_t` class.

".alpha": This is the Dirichlet or Gamma prior parameters, and may be a constant or may be estimated (if the `alpha_update` flag is in the ".par" file). They are stored internally as `M.alpha`.

".theta": This is the word by topic matrix (the Theta matrix stored internally as `M.theta`, and denoted $\vec{\Theta}$ in the theory papers). It is not output by `mphier` but derived by `mpupd`.

".gamma": This is the prior Dirichlet parameters for Theta above, by default a constant set by `thetaprior` in the ".par" file, and are stored internally as `M.gamma`). This may also be estimated, or set using empirical Bayes to be proportionally to the observed frequency in the data.

".meanM": when the "-A" command line option is used in `mphier`, the Gibbs sampler records the average of the topic probabilities per document (denoted by $\vec{m}$ or $\vec{l}$ in the theory papers).

".meantheta": when the "-A" command line option is used in `mphier`, the Gibbs sampler records the average of the and the topic matrix. A snapshot is given in ".theta".

".struct": This stores the structure of the Pachinko graph, and there are several ways of inputting this. They are stored internally in the `hstruct_t` class. This part is undocumented for now.

".psi": These are the Dirichlet priors for the Pachinko, and roughly correspond to ".alpha". Their dimensions match ".struct" and are stored internally as `H.psi`). This part is undocumented for now.

4. Worked Examples

Several worked examples are given in the `examples/` sub-directory. These demonstrate a variety of ways of creating inputs, and a variety of uses. In each directory, the `README` file explains where to get the necessary data and how to run the example.

4.1 Tests

A number of simple test sets are given in the top level `tests/` directory. This are generated from known models recorded in stems like `t2.src` using the modelling sampling option, `mpupd -S`.

In this way, a known "truth" is set down in the model, and then data generated. Thus, when `mphier` is run to estimate a model from the data, it can be compared with the truth. This allows careful testing of algorithms.

The test directory contains a number of known models, scripts for generating the data in a `Makefile`, and some test scripts. Details are in the `README` file.

4.2 Examples/UCI

The UCI data sets for topic modelling are available at:

`http://archive.ics.uci.edu/ml/datasets/Bag+of+Words`

They were made available by David Newman. These are a nice variety of content to play with as a starting point, and have been used on a good number of research papers. To start with, download some of the data sets and run the worked examples, as explained in the `README` file.

4.3 Examples/ijcv_unsup

The IJCV data sets were created from the Caltech256 [GHP07] data set[4] and the MSRC2 data set[5]. They are the subject of an extensive comparative study [TTB09] reported at the website:

`http://homes.esat.kuleuven.be/~tuytelaa/unsup/unsup.html`

This directory includes a number of scripts for processing the data and for doing testing.

ijcv2bag: This converts the featurised data available from the website to the binary bagged format used by`mphier`.

xent.pl: Given that images are know to correspond to given classes, this script scores the mismatch between the topics discovered by tt DCA and the "true" classes. In this case, the number of topics used in the `mphier` run must be the same as the true number of classes, and each topic is assigned to its best class.

MtoSVML.pl: This allows the `xent.pl` script to be applied to the case where more topics are built than there are "true" classes. Here, groups of topics area assigned to best classes.

Earlier versions of these evaluation scripts where used in the paper.

4.4 Examples/Reuters

These examples show how you can use DCA with the Reuters Corpus Volume 1. Reuters stopped distributing the corpus in 2004. Instead, the Reuters corpus is now available from NIST, the National Institute of Science and Technology.

The Reuters news items are in NewsML, an XML format, with one file per item. An XSLT script is given so that text data suitable for `linkTables` can be extracted from the XML. The `README` file shows how this is used.

4. `http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001`
5. `http://research.microsoft.com/en-us/projects/objectclassrecognition/`

4.5 Examples/Wikipedia

Two examples are given for the Wikipedia.

In the first example, we have had a small collection of documents packaged up as an XML suite. The intent is, you have document collections available to you in XML. You use XSLT or another XML formatting system to convert the documents to the simple tokenised input required by `linkBags`. This then manages the preparation of bags for the running of topic models. The problem with XSLT is that it requires a full parse of the XML, so you need to keep the XML files down to chunks of 50Mb or less. Otherwise its a good way to process files since everything is correctly parsed. In this example, an XSLT script and a matching XML file are given. This just illustrates the use of XML and XSL for topic modelling.

The second example allows working on the full Wikipedia collection. The Freebase Wikipedia Extraction (WEX)[6] is a processed dump of the English language Wikipedia, provided free of charge for any purpose with regular updates by Metaweb Technologies. A Perl script is provide in DCA-Bags called `wex2link` that converts the WEX article files into the format expected of `linkTables`.

5. Evaluating Models on Test Data

The system is set up so that some final set of documents in the bags can be used as test data. To use this feature, set the `testdocs` parameter in the ".par" file to a positive integer indicating how many of the last documents in the document collection are to be used for testing. When this is done, training of the model will only be done on the first `documents-testdocs` number of documents. A simple score will be reported for the training documents and the testing documents during the run, but at the end of the run, the "-X" option to `mphier` can be used to estimate likelihoods on the testing set. The variations available here are described in the companion report "Estimating Likelihoods for Topic Models" available in the documentation directory.

As an example, see the `tests/` directory in the release. From the `tests/` directory, do the following to estimate likelihoods in four different ways, and records all results in "t4Z.err" file.

```
rm t4Z.err
#   importance sampling using 100 samples
mphier -L200 -X100,I -e t4Z
#   left-to-right particle filter with 20 samples
mphier -r -L0 -X20,L -e t4Z
#   left-to-right sampler filter with 20 samples
mphier -r -L0 -X20,M -e t4Z
#   harmonic mean with 400 samples
mphier -r -L0 -X400,G -e t4Z
mpupd -T t4Z
```

Note the first option "-L200" says build the model with 200 full Gibbs cycles. The subsequent options "-r -L0" says restart from the last values.

ACKNOWLEDGEMENTS.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Wray Buntine acknowledges support from the EU project CLASS (IST project 027978). Wray Buntine also thanks Helsinki Institute of Information Technology for providing some of the libraries, available from `MPCA`, for relicensing in `DCA`.

References

[BJ06] W.L. Buntine and A. Jakulin. Discrete components analysis. In C. Saunders, M. Grobelnik, S. Gunn, and J. Shawe-Taylor, editors, *Subspace, Latent Structure and Feature Selection Techniques*. Springer-Verlag, 2006.

[GHP07] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

6. `http://download.freebase.com/wex`

[MLM07] D. Mimno, W. Li, and A. McCallum. Mixtures of hierarchical topics with Pachinko allocation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 633–640. ACM, 2007.

[TTB09] M. B. Blaschko T. Tuytelaars, C.H. Lampert and W. Buntine. Unsupervised object discovery: A comparison. *International Journal on Computer Vision*, (to appear), 2009.

A. DCA Command Line Options

The man pages for the three compiled executables, `mpdata`, `mphier` and `mpupd` follow.

NAME

mpdata − input from text and manipulate binary data for bags

SYNOPSIS

mpdata [**-p**|**-s Fstart,Fend**|**-P featfile**] [**−m** *mindoc*] [**−D** *maxdoc*] [**−F** *maxfeat*] [**−S** *seed*] *stem*

DESCRIPTION

This manual page documents briefly the **mpdata** command, Version 0.200 released July 2009.

mpdata does data input conversion, or printing if the **-p**, **-s** or **-P** options are used. Conversion is driven by a configuration file "*stem*.cnf" and has a variety of nodes including input from sparse integer vectors, or feature/document transformations from other bag files.

The input data for conversion is either another binary file set, as produced by **mpdata**, or a text file giving the sparse vectors for each document. The text file formats are described in the user guide.

OPTIONS

−m mindoc
Select the document to start from with any printing.

−s Fstart,Fend
Only print features within this range. Not used with **-p** or **-P**.

−D maxdoc
Sets the maximum document for printing or data conversion.

−F maxfeat
Sets the maximum feature for printing or data conversion.

−P featfile
Print only a subset of documents, those containing features in the file **featfile** (white space delimited). Not used with **-p** or **-s**.

−S seed
Set the seed for the *rand48* random number generator.

−p perform printing, not data conversion. Not used with **-s** or **-P**.

FILES

Data conversion requires the input configuration file "*stem*.cnf". This file has parameter values, one per line. Optionally, characters after any "#" are stripped on a line. One set of parameters control the style of conversion done.

inputfile="S"
name of a text file containing docs one per line with each line being space delimited feature indexes. The format is described in the user guide. "*stem*.txtbag" is used by convention.

subset="S"
name of data set stem, from which a subset will be taken. Document indices giving the subset to take are listed in the file "*stem*.indmap". Features are preserved.

featuresubset="S"
name of data set stem, from which a subset will be taken. Feature indices giving the subset to take are listed in the file "*stem*.feats". Documents are otherwise preserved. Features in the new data set are given the index corresponding to their line number.

The following options are also available.

baggedinput
a boolean indicating sparse vectors in **inputfile** have been bagged, so each feature index has a following count. By default, sparse vectors are just a list of features.

indexdensity=I
Record document indexes in the "*stem*.iinx" file for every N-th document.

maxdoc=I
a count indicating number of documents to include. Others ignored.

maxfeat=I
a count indicating number of features to include. Others ignored.

mincount=I
works with **subset** and **featuresubset** only, discards documents with less than this number of total words. Default is 0.

minfeatcount=I
works with **subset** only, discards features with less than this number of total occurrences. Default is 2.

trainlimit=I
Count giving maximum number of documents used in computing word counts (stored in the "*stem*.wc" file).

The following files are output after data conversion.

stem.bag
The binary bag file that digests the sparse vectors.

stem.iinx
Document indexes for random access to the "*stem*.bag". By default only every 10th document is indexed.

stem.par
Basic text parameter file giving useful parameters for subsequent runs of **mphier**(1).

stem.wc
Binary vector giving word-counts for each feature in the full collection. The format is an *int* followed by *float*s.

SEE ALSO

mphier(1), **mpupd**(1).
The programs are documented by *DCA 0.200: Discrete Component Analysis Software*, and *DCA 0.200 User Guide*, available in the software release.

AUTHOR

The program and manual page was written by Wray Buntine <wray.buntine@nicta.com.au>.

NAME

mphier − train or test a DCA model

SYNOPSIS

mphier [**−elmnrvw**] [**−b** *blocks*] [**−d** *bsize*] [**−o** *dots*] [**−s** *seed*] [**−t** *threads*] [**−z** *rate*] [**−ABCLMRT** *loops*] [**−H** *st,end*] [**−S** *dim*] [**−X** *samples,code*] *stem*

DESCRIPTION

This manual page documents briefly the **mphier** command, Version 0.200 released July 2009.

mphier runs in two major modes. In the first an initial run is where the file "*stem*.par" is read for parameters, a DCA estimation cycle run, and then result files are produced. In the second mode, initiated with the **-r** flag, a previous run is restarted so the result files and the parameter file are read initially, and then the run continues.

OPTIONS

−b blocks
: If the document vectors *H+K* and *M* are really big, split them into **blocks** blocks, reading each into memory sequentially.

−d bsize
: Break the data set up into **bsize**Mb blocks during reading, keeping each block in memory as it is used.

−o dots
: During each cycle, after each **dots** documents are processed, a dot is printed.

−s seed
: Seed for the *rand48* random number generator.

−t threads
: Number of threads to run (if threads are compiled).

−z rate
: Store the total topic counts for each word (statistics for the *Theta* matrix) internally as a sparse vector when the non-zero proportion goes below the **rate**.

−A loops
: Records averages during Giibs cycles of the *M* matrix, the topic proportions per document, and the *Theta* matrix, word probabilities for each topic. Start this after **loops** cycles.

−B loops
: Rebuild all internal statistics ever so many cycles.

−C loops
: Do a checkpoint of files (allowing a safe restart) every so many cycles.

−H Tstart,Tend
: With the **-T** option, only fix *Theta* between these ranges of words indices.

−L loops
: Run for so many Gibbs cycles.

−M loops
: Start averaging document log-probabilities after so many cycles.

−R loops
: Do a one line report every so many cycles. Default is 5.

−S dim
: Slice sampling dimension.

−T loops
: Read in the supplied *Theta* and use for Gibbs sampling (instead of estimates from statistics) in the first so many cycles.

−X cycles,code
: After Gibbs sampling is complete, estimate the log-likelihood on the test set (documents numbered *documents-testdocs* to *documents*-1). The sampling method uses **cycles** cycles and uses the methods in the report *Estimating Likelihoods for Topic Models* with **code**s as "I" for mean-field based importance sampling (MFI), "G" for the harmonic mean (HM), "L" for left-to-right particle filter (LR) and "M" for left-to-right sampler (LRS).

−e
: save the error output to "*stem*.err".

−l
: report scores in log probability, not in bits per token (the default).

−m
: do local maximising instead of Gibbs during the run.

−n
: do not update the *H+K* topic assignments during the Gibbs sampling, so the run is a no-op statistically, but still records and does the additional work.

−r
: do a restart from a previous run on the same file set with stem "*stem*".

−v
: increment the verbosity level.

−w
: adjust the scores for the (default) Dirichlet model to make them compatable with the Poisson model.

FILES

Requires the input parameter file "*stem*.par". The following files are output during checkpoints and at the end, and may also be read on a restart. All files are text files except "*stem*.comps" which is a directory with binary data.

stem.alpha
: This is the Dirichlet or Gamma prior parameters, and may be a constant or may be estimated (if the *alpha_update* flag is in the "*stem*.par" file).

stem.comps
: This is the major output and is the topic assignments for words in documents (denoted by *k* in the theory papers) and is in a special binary format in a directory.

stem.gamma
: This is the prior Dirichlet parameters for the *Theta* matrix, the topic by word matrix (by default a constant *thetaprior* from the "*stem*.par" file). This may also be estimated, or set using empirical Bayes to be proportionally to the observed frequency in the data.

stem.meanM
: when the **-A** command line option is used, the Gibbs sampler records the average of the topic probabilities per document (denoted by *m* or *l* in the theory papers).

stem.meantheta
: when the **-A** command line option is used, the Gibbs sampler records the average of the *Theta* matrix, the topic by word matrix.

stem.theta
: This is the word by topic matrix (the *Theta* matrix). It is not output by **mphier** but derived by **mpupd** from "*stem*.comps".

The input file "*stem*.par" has parameters, and comments following a "#" are stripped on input. It must have the following parameters.

datastem="S"
: file stem given in quotes for the data file, path must be relative to where the programme is called from.

documents=I
: count of documents, any after this are ignored.

features=I
count of words/tokens/features, any after this are ignored.

mean_featurecount=F
float giving average number of words per document, used as a normaliser in various report, and an initialiser for some parameters.

components=I
count of topics in model.

The following parameters are optional and mostly control the parameters to the topic model.

testdocs=I
the count of test documents, which make up the last part of the document data set.

alpha.fullgamma
a boolean (so no "=" is needed to set a value), makes the topics Gamma-Poisson rather than Dirichlet-Multinomial.

alpha.alpha=F,F,...F
a comma-delimited list of floats giving initial values for the *alpha* vector.

alpha.beta=F,F,...F
a comma-delimited list of floats giving initial values for the *beta* vector (if **alpha.fullgamma** is set).

alphamin=F
do not allow individual *alpha* values to go below this. Default is 0.01.

gammamin=F
do not allow individual *gamma* values to go below this. Gamma distributions with a rate of less than 0.001 start to behave badly due to underflow. Default is 0.001.

alpha_update
a boolean says to update the *alpha* vector and the *beta* vector during Gibbs to find a local maxima for them.

gamma_update
a boolean says to update the *gamma* vector during Gibbs to find a local maxima for them. This is a high dimension, so statistically not recommended.

thetaprior=F
initial constant that the *gamma* vector is set to, the prior for the *Theta* matrix. Default is 0.1.

thetaempiricalprior
a boolean sets the *gamma* vector proportionally to observed frequencies with total given by *thetaprior*.

gamma_burnin=I
updating of *alpha* and *beta* vector and the *gamma* vector only started after a burn-in of so many cycles. Default is 10.

gamma_cyclefactor=I
updating of *alpha* and *beta* vector and the *gamma* vector only done every so many cycles. Default is 5.

The following parameters control the lay-out of features during reports.

n_partition=I
split the features into so many parts, default is 1.

partition=I,I,...,I
a comma separated list of the boundaries of the parts, starting at 0, so has **n_partition**+1 entries.

tags=C,C,...,C
a comma separated list of tags for each part, usually a single letter.

tags_long=S,S....,S
a comma separated list of names for each part, usually a word for each.

SEE ALSO

mpupd(1), **mpdata**(1).
The programs are documented by *DCA 0.200: Discrete Component Analysis Software*, and *DCA 0.200 User Guide*, available in the software release.

AUTHOR

The program and manual page was written by Wray Buntine <wray.buntine@nicta.com.au>.

NAME

mpdata − input from text and manipulate binary data for bags

SYNOPSIS

mpdata [**-p**|**-s Fstart,Fend**|**-P featfile**] [**−m** *mindoc*] [**−D** *maxdoc*] [**−F** *maxfeat*] [**−S** *seed*] *stem*

DESCRIPTION

This manual page documents briefly the **mpdata** command, Version 0.200 released July 2009.

mpdata does data input conversion, or printing if the **-p**, **-s** or **-P** options are used. Conversion is driven by a configuration file "*stem*.cnf" and has a variety of nodes including input from sparse integer vectors, or feature/document transformations from other bag files.

The input data for conversion is either another binary file set, as produced by **mpdata**, or a text file giving the sparse vectors for each document. The text file formats are described in the user guide.

OPTIONS

−m mindoc
Select the document to start from with any printing.

−s Fstart,Fend
Only print features within this range. Not used with **-p** or **-P**.

−D maxdoc
Sets the maximum document for printing or data conversion.

−F maxfeat
Sets the maximum feature for printing or data conversion.

−P featfile
Print only a subset of documents, those containing features in the file **featfile** (white space delimited). Not used with **-p** or **-s**.

−S seed
Set the seed for the *rand48* random number generator.

−p perform printing, not data conversion. Not used with **-s** or **-P**.

FILES

Data conversion requires the input configuration file "*stem*.cnf". This file has parameter values, one per line. Optionally, characters after any "#" are stripped on a line. One set of parameters control the style of conversion done.

inputfile="S"
name of a text file containing docs one per line with each line being space delimited feature indexes. The format is described in the user guide. "*stem*.txtbag" is used by convention.

subset="S"
name of data set stem, from which a subset will be taken. Document indices giving the subset to take are listed in the file "*stem*.indmap". Features are preserved.

featuresubset="S"
name of data set stem, from which a subset will be taken. Feature indices giving the subset to take are listed in the file "*stem*.feats". Documents are otherwise preserved. Features in the new data set are given the index corresponding to their line number.

The following options are also available.

baggedinput
a boolean indicating sparse vectors in **inputfile** have been bagged, so each feature index has a following count. By default, sparse vectors are just a list of features.

indexdensity=I
Record document indexes in the "*stem*.iinx" file for every N-th document.

maxdoc=I
a count indicating number of documents to include. Others ignored.

maxfeat=I
a count indicating number of features to include. Others ignored.

mincount=I
works with **subset** and **featuresubset** only, discards documents with less than this number of total words. Default is 0.

minfeatcount=I
works with **subset** only, discards features with less than this number of total occurrences. Default is 2.

trainlimit=I
Count giving maximum number of documents used in computing word counts (stored in the "*stem*.wc" file).

The following files are output after data conversion.

stem.bag
The binary bag file that digests the sparse vectors.

stem.iinx
Document indexes for random access to the "*stem*.bag". By default only every 10th document is indexed.

stem.par
Basic text parameter file giving useful parameters for subsequent runs of **mphier**(1).

stem.wc
Binary vector giving word-counts for each feature in the full collection. The format is an *int* followed by *float*s.

SEE ALSO

mphier(1), **mpupd**(1).
The programs are documented by *DCA 0.200: Discrete Component Analysis Software*, and *DCA 0.200 User Guide*, available in the software release.

AUTHOR

The program and manual page was written by Wray Buntine <wray.buntine@nicta.com.au>.

B. `link` Command Line Options

The man pages for the four Perl scripts for processing of simple token/text files into a fomrat suitable for DCA.

NAME

linkRedir −− rocess out the redirects and normalise links in a link file.

SYNOPSIS

linkRedir [−−nocase|−−noclean|−−gzip|−−bzip2] LINK-FILE STEM

Options:

```
LINK−FILE          Filename for input link file usually created by XSL
STEM               stem for output file, several extensions read and made
−−bzip2            pipe input through bzip2(1)
−−gzip             pipe input through gzip(1)
−−init             empty the STEM.redirect file at startup
−−nocase           ignore case of URLs
−−noclean          don't use built−in URL cleaning
−h, −−help         display help message and exit.
−−man              print man page and exit.
```

DESCRIPTION

Input file of links, link text and redirects. Process out the redirects and normalise links, taking two passes. This is intended as a preprocessor for *linkTables*(1), and the input format is found there.

First pass reads in previously assembled redirects from *STEM.redirect* and then filter any more from LINK-FILE. The second pass reads the links from LINK-FILE and processes out the redirects. Output processed links to STDOUT and saves all redirects in *STEM.redirect*. Both output files are appended to so can be applied repeatedly to incrementally build up the result files from a series of batches, though redirects read in a later batch will not be applied to earlier batches. Normalises output URIs using URI−>canonical. Assumes input URIs have no embedded spaces, they must be encoded.

SEE ALSO

linkBags(1), *linkDca*(1), *linkTables*(1), *URI*(3).

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE

NAME

linkTables − input file of links and tokens for document set, and generated token and document tables.

SYNOPSIS

linkTables [−−docs|−−linktext|−−nocase|−−noclean|−−titletext]
[−−mincount N] [−−stopfile FILE] LINK-FILE STEM

Options:

```
LINK−FILE          Filename for input link file usually created by XSL
STEM               Stem for output file, several extensions read and made
−−docs             only update the .docs file, all else remains fixed
−−linktext         add link text, delimit by spaces, to text type
−−mincount M       only add tokens with this many
−−nocase           ignore case of URLs
−−noclean          don't use built−in URL cleaning
−−stopfile F       do not enter these words in text tables
−−stemming         runs the default Lingua::Stem (Perl) stemmer on text
−−titletext        add title text, delimit by spaces, to text type
−h, −−help         display help message and exit.
 −−man             print man page and exit.
```

DESCRIPTION

Input file of links, link text and redirects in the data format described next. Use file name '−' to input stdin. Builds the tables used in bag processing:

STEM.tokens	N−th line is the token for items with index (N−1).
STEM.words	A map for the token file includes token, its type and the hash code.
STEM.docs	N−th line is the details for the N−th document
STEM.docfeats	mapping of token index to document index

The token to document index in .docfeats is implied after standardising OUTGOING-URLs for a document and the document URls themselves

DATA FORMAT

Input lines can have the R form for redirects:

R <URL> <URL−REDIRECTED−TO>

These entries are ignored by this script, and should be first eliminated with *linkRedir*(1). The main input is the D form for documents and their links and link text

D <URL> <HASHID> <TITLE>
<OUTGOING−URL> <LINK−TEXT>
...
EOL
<TYPE> <TOKEN>
...
EOD

The text "EOD" acts as a document terminator and can be missing if no tokens exist. The text "EOL" is a link terminator. The <URL>s and <HASHID>s must not have spaces or the processing will get confused since R and D records are split on spaces. Note text at the end of the line is an exception. <HASHID> is any externally defined record identifier. The default is a 32 character hexadecimal from an MD5 hash of the text.

<TYPE> is intended to be a short bit of alphabetic text describing the type such as 'person', 'company', etc. Reserved <TYPE>s are 'doc', link to a document in the collection, 'link' which is a link out of the collection, and 'text' which is any text.

SEE ALSO

DCA::URLs(3), *linkBags*(1), *linkDca*(1), *linkRedir*(1), *mpdata*(1).

DCA website is inside *http://www.nicta.com.au*

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE

NAME

linkBags − input file of links and tokens for document set, plus tables generated with *linkTables*, to produce forward bags.

SYNOPSIS

linkBags [−−linktext|−−nocase|−−noclean|−−stemming|−−titletext|−−update] LINK-FILE STEM

Options:

```
LINK−FILE          Filename for input link file usually created by XSL.
STEM               Stem for output file, severl extensions read and made.
−−linktext         add link text to the bag
−−nocase           set nocase flag in DCA::URLs
−−noclean          set noclean flag in DCA::URLs
−−stemming         must match usage in linkTables
−−titletext        add title text to the bag
−−update           indicate to the output config file that this is an update
−h, −−help         display help message and exit.
 −−man             print man page and exit.
```

DESCRIPTION

This package works in conjunction with an XSL script which is used to generate a text file giving URL+title+link+tag information for the input XML files. Use name '−' to input stdin. The final output files are created when the DCA *mpdata*(1) utility is called.

Input file of links and tags is assumed to be in UTF−8 encoding in the format given in *linkTables*(1). Separate tables (*STEM.words*, *STEM.docs*, *STEM.docmap*) have previously been made by running *linkTables*(1). The linktext, nocase, noclean and titletext flags should be the same as those used.

If redirects exist in the links file, process them out first using the *linkRedir*(1) script. Code assumes that collection is small enough so that all required hash tables fit into memory.

Output files in form STEM.EXT, for EXT one of:
.txtbag[.gz] : constructed input text bag for mpdata
.bag, .cnf, .iinx, .par : usual output of mpdata

See *mpdata*(1) for detail of these formats.

SEE ALSO

DCA::URLs(3), *linkDca*(1), *linkRedir*(1), *linkTables*(1), *mpdata*(1).

DCA website is inside *http://www.nicta.com.au*

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE

NAME

linkDca −− default run of DCA on data from linkBags(1)

SYNOPSIS

linkDca K STEM

```
Options:

  K                   number of components to use
  STEM                file stem for linkBags(1) bag
  -h, --help          display help message and exit.
  --man               print man page and exit.
```

DESCRIPTION

Builds K components with default mphier. Requires a basic STEM to exist with .par and bags already created, presumably by *linkBags*(1). It builds the new model with name STEM plus K.

SEE ALSO

linkRedir(1), *linkBags*(1), *linkTables*(1).

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE

C. Utility Command Line Options

The following are printed when the programmes are run without command line options. The first routine `dca2html` produces a HTML report on a model. The `docword2bag` utility converts UCI sparse matrix format to `mpdata` sparse matrix format, and builds binary bags for DCA. The `wex2link` utility converts WEX Wikipedia dumps into the links format used by the `link` utilities.

NAME

dca2html − script to pretty print DCA results in HTML.

SYNOPSIS

```
dca2html OPTIONS+ STEM

Options:
  -c  --tokencount C   only include this many tokens
  -d, --docs D         number of documents to report on per component
  --dot                create a dot file for use with graph
  --error              add the error log to the report
  -p  --parent C       only include parents greater than the cutoff
  -s, --syms S         number of tokens to report on per component
  -t, --titles F       titles file used to generate prototypical documents
  -v  --verbose        print more details
      --no-verbose     print less details
```

DESCRIPTION

Calls *mpupd* to gather details of the components and (optionally) prototypical documents, and then pretty prints results in HTML.

Reads the STEM.par file to get hierarchy dimensions and feature dimensions. Reads the STEM.tokens file to get tokens.

Documents only reported on if a titles file given with ‘‘−t’’.

SEE ALSO

mpupd(1).

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE

NAME

docword2bag −− convert UCI docword format to DCA bag format

SYNOPSIS

docword2bag STEM

```
Options:
  -h, --help          display help message and exit.
  --man               print man page and exit.
```

DESCRIPTION

Expects to find vocab.STEM.txt and docword.STEM.txt or docword.STEM.txt.gz in the UCI data format. These are converted to the input required for DCA’s *mpdata*(1), the ‘‘.txtbag’’ format, and then converted by running *mpdata*.

This a slow Perl script, so conversion is not fast.

SEE ALSO

mpdata(1),

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE

NAME

wex2link − convert WEX format Wikipedia articles to *linkTables*, format.

SYNOPSIS

wex2link

Options:

```
    -h, --help          display help message and exit.
     --man              print man page and exit.
```

DESCRIPTION

The Freebase Wikipedia Extraction (WEX) is a processed dump of the English language Wikipedia, provided free of charge for any purpose with regular updates by Metaweb Technologies. See *http://download.freebase.com/wex* . *wex2link* converts records in the articles file of the WEX dump to the format expected by *linkTables*(1).

Works as a filter, using standard input and output. Has been written using unoptimised Perl, and with no proper XML handling, e.g., no SAX parser, just string processing to handle the XML, so it is fairly slow and may be prone to string/parse errors. Note also link text is ignored.

The processing drops pages that are (1) disambiguation pages, (2) timelines, (3) times, (4) categories, (5) files (e.g., images), and (6) lists. Recognition of these isn't foolproof, so some will be included, and some other pages will get wrongly excluded.

Tokens in the pages are typed into the following (1) links (usually to other Wikipedia pages), (2) parameters (of templates), (3) template types, (4) categories, and of course (5) the page text. Note link text is ignored. See *linkTables*(1) for format these are in.

SEE ALSO

linkBags(1), *linkTables*(1),

DCA website is inside *http://www.nicta.com.au*

AUTHOR

Wray Buntine

COPYRIGHT AND LICENSE