models.ldamodel – Latent Dirichlet Allocation

For a faster implementation of LDA (parallelized for multicore machines), see `gensim.models.ldamulticore`.

Latent Dirichlet Allocation (LDA) in Python.

This module allows both LDA model estimation from a training corpus and inference of topic distribution on new, unseen documents. The model can also be updated with new documents for online training.

The core estimation code is based on the *onlineldavb.py* script by M. Hoffman **[1]**, see **Hoffman, Blei, Bach: Online Learning for Latent Dirichlet Allocation, NIPS 2010.**

The algorithm:

- is **streamed**: training documents may come in sequentially, no random access required,
- runs in **constant memory** w.r.t. the number of documents: size of the training corpus does not affect memory footprint, can process corpora larger than RAM, and
- is **distributed**: makes use of a cluster of machines, if available, to speed up model estimation.

[1] http://www.cs.princeton.edu/~mdhoffma

---

class `gensim.models.ldamodel.LdaModel`(*corpus=None*, *num_topics=100*, *id2word=None*, *distributed=False*, *chunksize=2000*, *passes=1*,*update_every=1*, *alph*

Bases: `gensim.interfaces.TransformationABC`

The constructor estimates Latent Dirichlet Allocation model parameters based on a training corpus:

```
>>> lda = LdaModel(corpus, num_topics=10)
```

You can then infer topic distributions on new, unseen documents, with

```
>>> doc_lda = lda[doc_bow]
```

The model can be updated (trained) with new documents via

```
>>> lda.update(other_corpus)
```

Model persistency is achieved through its *load*/*save* methods.

If given, start training from the iterable *corpus* straight away. If not given, the model is left untrained (presumably because you want to call*update()* manually).

num_topics is the number of requested latent topics to be extracted from the training corpus.

id2word is a mapping from word ids (integers) to words (strings). It is used to determine the vocabulary size, as well as for debugging and topic printing.

alpha and *eta* are hyperparameters that affect sparsity of the document-topic (theta) and topic-word (lambda) distributions. Both default to a symmetric 1.0/num_topics prior.

alpha can be set to an explicit array = prior of your choice. It also support special values of ‘asymmetric’ and ‘auto’: the former uses a fixed normalized asymmetric 1.0/topicno prior, the latter learns an asymmetric prior directly from your data.

eta can be a scalar for a symmetric prior over topic/word distributions, or a matrix of shape num_topics x num_words, which can be used to impose asymmetric priors over the word distribution on a per-topic basis. This may be useful if you want to seed certain topics with particular words by boosting the priors for those words. It also supports the special value ‘auto’, which learns an asymmetric prior directly from your data.

Turn on *distributed* to force distributed computing (see the **web tutorial** on how to set up a cluster of machines for gensim).

Calculate and log perplexity estimate from the latest mini-batch every *eval_every* model updates (setting this to 1 slows down training ~2x; default is 10 for better performance). Set to None to disable perplexity estimation.

decay and *offset* parameters are the same as Kappa and Tau_0 in Hoffman et al, respectively.

minimum_probability controls filtering the topics returned for a document (bow).

random_state can be a numpy.random.RandomState object or the seed for one

Example:

```
>>> lda = LdaModel(corpus, num_topics=100)  # train model
```

```
>>> print(lda[doc_bow]) # get topic probability distribution for a document
>>> lda.update(corpus2) # update the LDA model with additional documents
>>> print(lda[doc_bow])
```

```
>>> lda = LdaModel(corpus, num_topics=50, alpha='auto', eval_every=5)  # train asymmetric alpha from data
```

---

bound(*corpus*, *gamma=None*, *subsample_ratio=1.0*)

Estimate the variational bound of documents from *corpus*: E_q[log p(corpus)] - E_q[log q(corpus)]

gamma are the variational parameters on topic weights for each *corpus* document (=2d matrix=what comes out of *inference()*). If not supplied, will be inferred from the model.

---

clear()

Clear model state (free up some memory). Used in the distributed algo.

---

do_estep(*chunk*, *state=None*)

Perform inference on a chunk of documents, and accumulate the collected sufficient statistics in *state* (or *self.state* if None).

---

do_mstep(*rho*, *other*, *extra_pass=False*)

M step: use linear interpolation between the existing topics and collected sufficient statistics in *other* to update the topics.

---

get_document_topics(*bow*, *minimum_probability=None*, *minimum_phi_value=None*, *per_word_topics=False*)

Return topic distribution for the given document *bow*, as a list of (topic_id, topic_probability) 2-tuples.

Ignore topics with very low probability (below *minimum_probability*).

If per_word_topics is True, it also returns a list of topics, sorted in descending order of most likely topics for that word. It also returns a list of word_ids and each words corresponding topics' phi_values, multiplied by feature length (i.e, word count)

---

get_term_topics(*word_id*, *minimum_probability=None*)

Returns most likely topics for a particular word in vocab.

---

get_topic_terms(*topicid*, *topn=10*)

Return a list of *(word_id, probability)* 2-tuples for the most probable words in topic *topicid*.

Only return 2-tuples for the topn most probable words (ignore the rest).

---

inference(*chunk*, *collect_sstats=False*)

Given a chunk of sparse document vectors, estimate gamma (parameters controlling the topic weights) for each document in the chunk.

This function does not modify the model (=is read-only aka const). The whole input chunk of document is assumed to fit in RAM; chunking of a large corpus must be done earlier in the pipeline.

If *collect_sstats* is True, also collect sufficient statistics needed to update the model's topic-word distributions, and return a 2-tuple*(gamma, sstats)*. Otherwise, return *(gamma, None)*. *gamma* is of shape *len(chunk) x self.num_topics*.

Avoids computing the *phi* variational parameter directly using the optimization presented in **Lee, Seung: Algorithms for non-negative matrix factorization, NIPS 2001**.

---

init_dir_prior(*prior*, *name*)

classmethod load(*fname*, **args*, ***kwargs*)

Load a previously saved object from file (also see *save*).

Large arrays can be memmap'ed back as read-only (shared memory) by setting *mmap='r'*:

```
>>> LdaModel.load(fname, mmap='r')
```

---

log_perplexity(*chunk*, *total_docs=None*)

Calculate and return per-word likelihood bound, using the *chunk* of documents as evaluation corpus. Also output the calculated statistics. incl. perplexity=2^(-bound), to log at INFO level.

---

print_topic(*topicid*, *topn=10*)

Return the result of *show_topic*, but formatted as a single string.

---

print_topics(*num_topics=10*, *num_words=10*)

`save`(*fname, ignore=['state', 'dispatcher'], *args, **kwargs*)

Save the model to file.

Large internal arrays may be stored into separate files, with *fname* as prefix.

separately can be used to define which arrays should be stored in separate files.

ignore parameter can be used to define which variables should be ignored, i.e. left out from the pickled lda model. By default the internal *state* is ignored as it uses its own serialisation not the one provided by *LdaModel*. The *state* and *dispatcher* will be added to any ignore parameter defined.

Note: do not save as a compressed file if you intend to load the file back with *mmap*.

Note: If you intend to use models across Python 2/3 versions there are a few things to keep in mind:

1. The pickled Python dictionaries will not work across Python versions
2. The *save* method does not automatically save all NumPy arrays using NumPy, only those ones that exceed*sep_limit* set in *gensim.utils.SaveLoad.save*. The main concern here is the *alpha* array if for instance using*alpha='auto'*.

Please refer to the wiki recipes section (**https://github.com/piskvorky/gensim/wiki/Recipes-&-FAQ#q9-how-do-i-load-a-model-in-python-3-that-was-trained-and-saved-using-python-2**) for an example on how to work around these issues.

`show_topic`(*topicid*, *topn=10*)

Return a list of *(word, probability)* 2-tuples for the most probable words in topic *topicid*.

Only return 2-tuples for the topn most probable words (ignore the rest).

`show_topics`(*num_topics=10*, *num_words=10*, *log=False*, *formatted=True*)

For *num_topics* number of topics, return *num_words* most significant words (10 words per topic, by default).

The topics are returned as a list – a list of strings if *formatted* is True, or a list of *(word, probability)* 2-tuples if False.

If *log* is True, also output this result to log.

Unlike LSA, there is no natural ordering between the topics in LDA. The returned *num_topics* <= *self.num_topics* subset of all topics is therefore arbitrary and may change between two LDA training runs.

`sync_state`()

`top_topics`(*corpus*, *num_words=20*)

Calculate the Umass topic coherence for each topic. Algorithm from **Mimno, Wallach, Talley, Leenders, McCallum: Optimizing Semantic Coherence in Topic Models, CEMNLP 2011.**

`update`(*corpus*, *chunksize=None*, *decay=None*, *offset=None*, *passes=None*, *update_every=None*, *eval_every=None*, *iterations=None*,*gamma_threshold=N*

Train the model with new documents, by EM-iterating over *corpus* until the topics converge (or until the maximum number of allowed iterations is reached). *corpus* must be an iterable (repeatable stream of documents),

In distributed mode, the E step is distributed over a cluster of machines.

This update also supports updating an already trained model (*self*) with new documents from *corpus*; the two models are then merged in proportion to the number of old vs. new documents. This feature is still experimental for non-stationary input streams.

For stationary input (no topic drift in new documents), on the other hand, this equals the online update of Hoffman et al. and is guaranteed to converge for any *decay* in (0.5, 1.0>. Additionally, for smaller *corpus* sizes, an increasing *offset* may be beneficial (see Table 1 in Hoffman et al.)

Args:

corpus (gensim corpus): The corpus with which the LDA model should be updated.

chunks_as_numpy (bool): Whether each chunk passed to *.inference* should be a numpy array of not. Numpy can in some settings turn the term IDs into floats, these will be converted back into integers in inference, which incurs a performance hit. For distributed computing it may be desirable to keep the chunks as numpy arrays.

For other parameter settings, see `LdaModel` constructor.

`update_alpha`(*gammat*, *rho*)

Update parameters for the Dirichlet prior on the per-document topic weights *alpha* given the last *gammat*.

`update_eta`(*lambdat*, *rho*)

Update parameters for the Dirichlet prior on the per-topic word weights *eta* given the last *lambdat*.

class gensim.models.ldamodel.LdaState(*eta*, *shape*)

Bases: **gensim.utils.SaveLoad**

Encapsulate information for distributed computation of LdaModel objects.

Objects of this class are sent over the network, so try to keep them lean to reduce traffic.

---

blend(*rhot*, *other*, *targetsize=None*)

Given LdaState *other*, merge it with the current state. Stretch both to *targetsize* documents before merging, so that they are of comparable magnitude.

Merging is done by average weighting: in the extremes, *rhot=0.0* means *other* is completely ignored; *rhot=1.0* means *self* is completely ignored.

This procedure corresponds to the stochastic gradient update from Hoffman et al., algorithm 2 (eq. 14).

---

blend2(*rhot*, *other*, *targetsize=None*)

Alternative, more simple blend.

---

get_Elogbeta()

---

get_lambda()

---

load(*fname*, *mmap=None*)

Load a previously saved object from file (also see *save*).

If the object was saved with large arrays stored separately, you can load these arrays via mmap (shared memory) using *mmap='r'*. Default: don't use mmap, load large arrays as normal objects.

If the file being loaded is compressed (either '.gz' or '.bz2'), then *mmap=None* must be set. Load will raise an *IOError* if this condition is encountered.

---

merge(*other*)

Merge the result of an E step from one node with that of another node (summing up sufficient statistics).

The merging is trivial and after merging all cluster nodes, we have the exact same result as if the computation was run on a single node (no approximation).

---

reset()

Prepare the state for a new EM iteration (reset sufficient stats).

---

save(*fname_or_handle*, *separately=None*, *sep_limit=10485760*, *ignore=frozenset([])*, *pickle_protocol=2*)

Save the object to file (also see *load*).

fname_or_handle is either a string specifying the file name to save to, or an open file-like object which can be written to. If the object is a file handle, no special array handling will be performed; all attributes will be saved to the same file.

If *separately* is None, automatically detect large numpy/scipy.sparse arrays in the object being stored, and store them into separate files. This avoids pickle memory errors and allows mmap'ing large arrays back on load efficiently.

You can also set *separately* manually, in which case it must be a list of attribute names to be stored in separate files. The automatic check is not performed in this case.

ignore is a set of attribute names to *not* serialize (file handles, caches etc). On subsequent load() these attributes will be set to None.

pickle_protocol defaults to 2 so the pickled object can be imported in both Python 2 and 3.

---

gensim.models.ldamodel.dirichlet_expectation(*alpha*)

For a vector *theta~Dir(alpha)*, compute *E[log(theta)]*.

---

gensim.models.ldamodel.get_random_state(*seed*)

Turn seed into a np.random.RandomState instance.

Method originally from maciejkula/glove-python, and written by @joshloyal

---

gensim.models.ldamodel.update_dir_prior(*prior*, *N*, *logphat*, *rho*)

Updates a given prior using Newton's method, described in **Huang: Maximum Likelihood Estimation of Dirichlet Distribution Parameters.** **http://jonathan-huang.org/research/dirichlet/dirichlet.pdf**