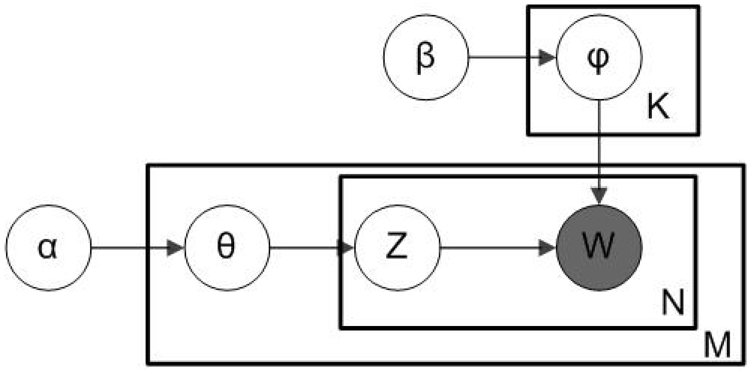
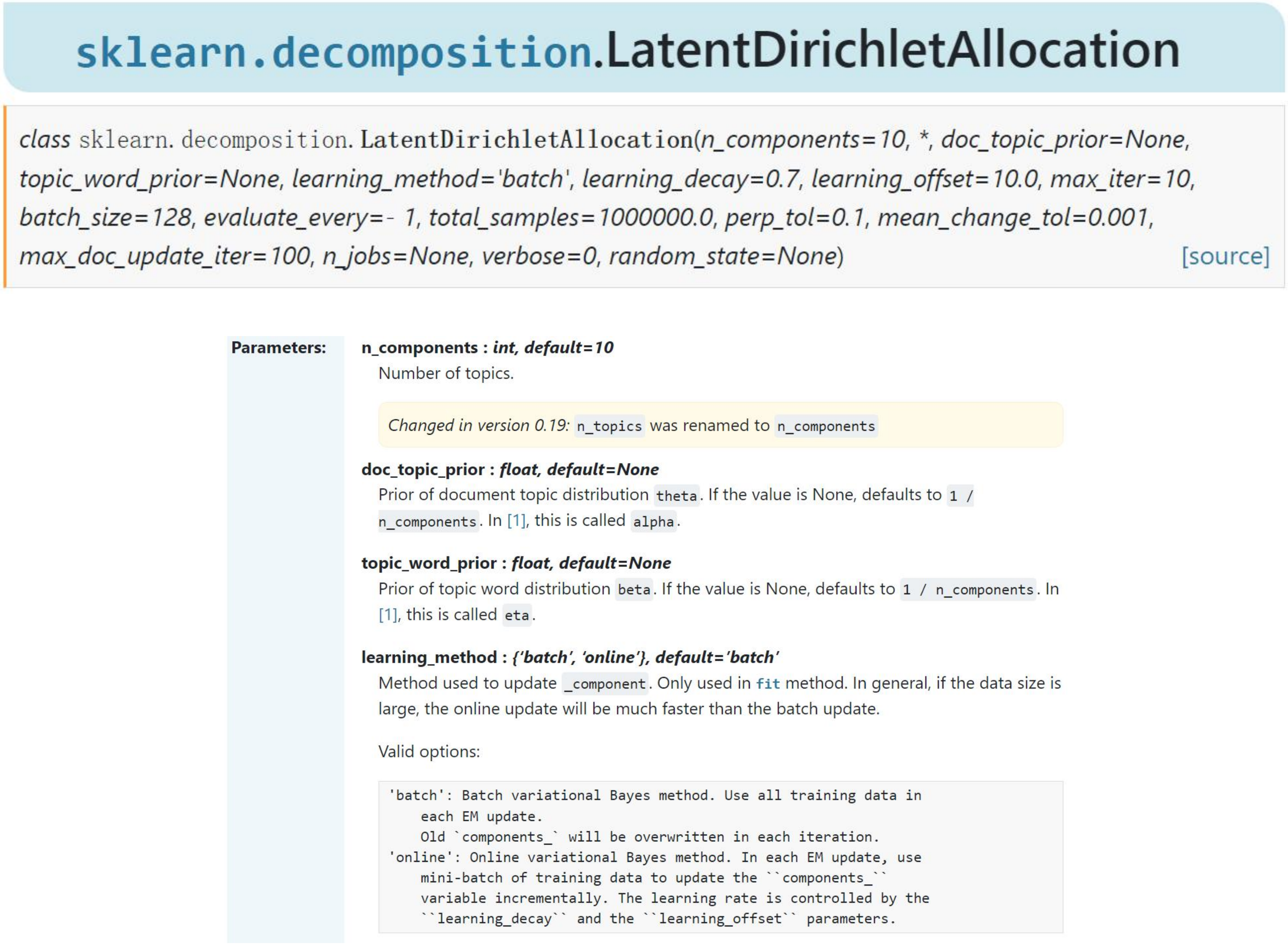
Social Media Analytics for Healthcare

Latent Dirichlet Allocation（LDA）

In natural language processing, **Latent Dirichlet Allocation**（**LDA**） is a generative statistical model that explains a set of observations through unobserved groups, and each group explains why some parts of the data are similar. LDA is an example of a topic model. In this, observations （e.g., words） are collected into documents, and each word's presence is attributable to one of the document's topics. Each document will contain a small number of topics.





sklearn.decomposition.LatentDirichletAllocation

class sklearn.decomposition.**LatentDirichletAllocation**(*n_components=10*, ***, *doc_topic_prior=None*, *topic_word_prior=None*, *learning_method='batch'*, *learning_decay=0.7*, *learning_offset=10.0*, *max_iter=10*, *batch_size=128*, *evaluate_every=- 1*, *total_samples=1000000.0*, *perp_tol=0.1*, *mean_change_tol=0.001*, *max_doc_update_iter=100*, *n_jobs=None*, *verbose=0*, *random_state=None*) [source]

Parameters:

n_components : *int, default=10*

Number of topics.

Changed in version 0.19: `n_topics` was renamed to `n_components`

doc_topic_prior : *float, default=None*

Prior of document topic distribution `theta`. If the value is None, defaults to `1 / n_components`. In [1], this is called `alpha`.

topic_word_prior : *float, default=None*

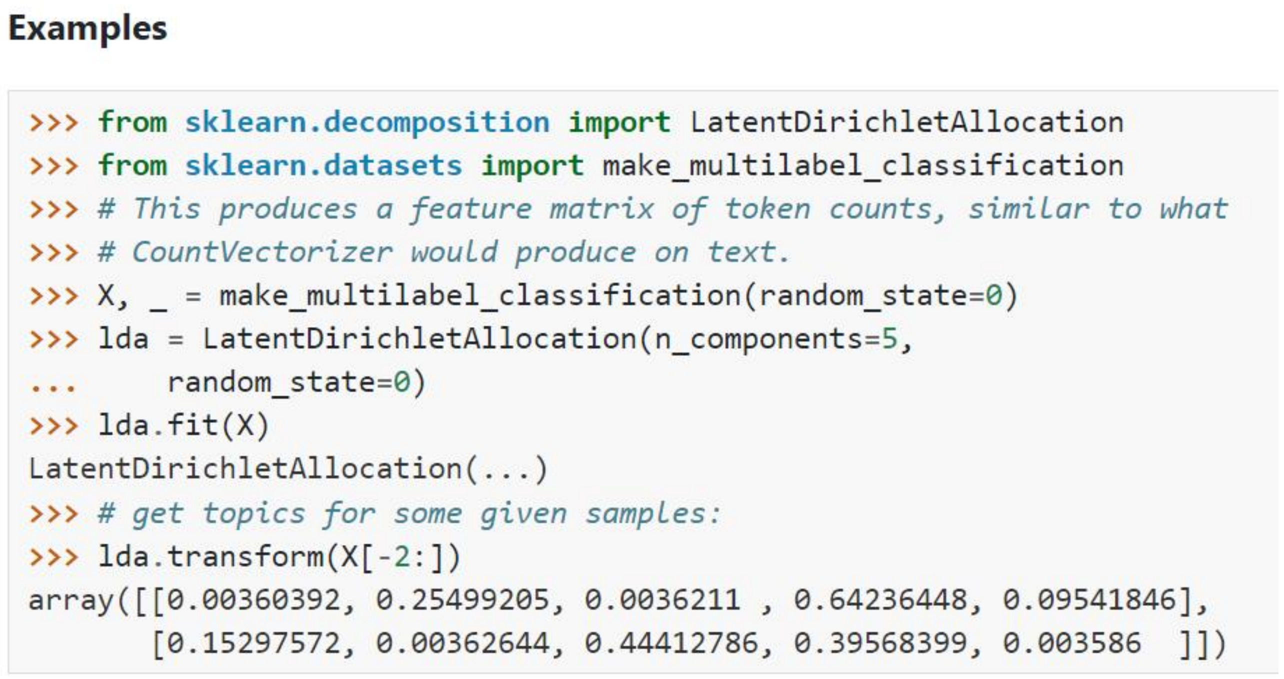
Prior of topic word distribution `beta`. If the value is None, defaults to `1 / n_components`. In [1], this is called `eta`.

learning_method : *{'batch', 'online'}, default='batch'*

Method used to update `_component`. Only used in `fit` method. In general, if the data size is large, the online update will be much faster than the batch update.

Valid options:

```
'batch': Batch variational Bayes method. Use all training data in
    each EM update.
    Old `components_` will be overwritten in each iteration.
'online': Online variational Bayes method. In each EM update, use
    mini-batch of training data to update the ``components_``
    variable incrementally. The learning rate is controlled by the
    ``learning_decay`` and the ``learning_offset`` parameters.
```



Examples

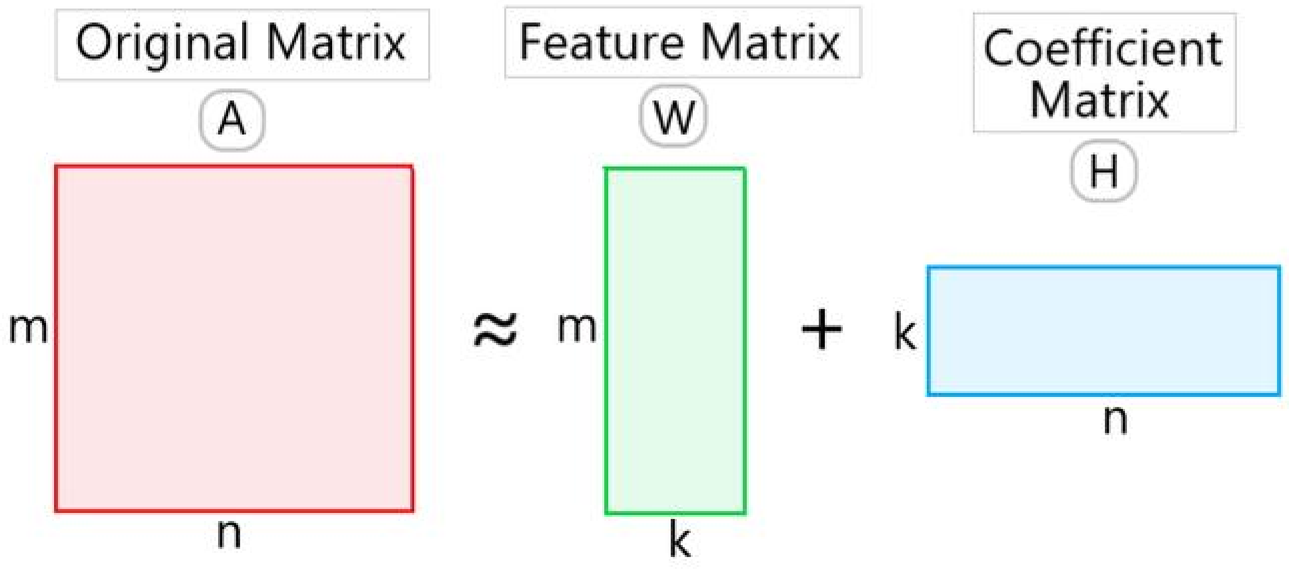
```
>>> from sklearn.decomposition import LatentDirichletAllocation
>>> from sklearn.datasets import make_multilabel_classification
>>> # This produces a feature matrix of token counts, similar to what
>>> # CountVectorizer would produce on text.
>>> X, _ = make_multilabel_classification(random_state=0)
>>> lda = LatentDirichletAllocation(n_components=5,
...     random_state=0)
>>> lda.fit(X)
LatentDirichletAllocation(...)
>>> # get topics for some given samples:
>>> lda.transform(X[-2:])
array([[0.00360392, 0.25499205, 0.0036211 , 0.64236448, 0.09541846],
       [0.15297572, 0.00362644, 0.44412786, 0.39568399, 0.003586  ]])
```

sklearn.datasets.make_multilabel_classification(n_samples=100, n_features=20, n_classes=5, n_labels=2, length=50, allow_unlabeled=True, sparse=False, return_indicator='dense', return_distributions=False, random_state=None)

- ***make_multilabel_classification***——Generate a random multilabel classification problem.

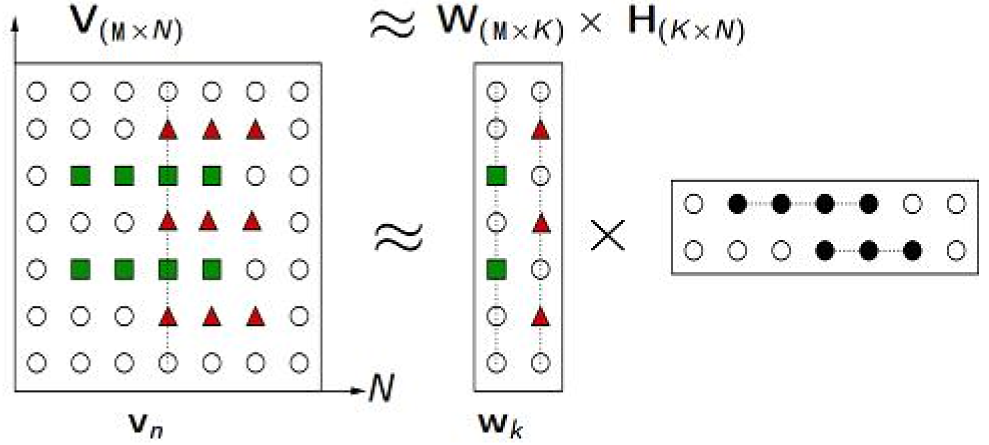
Non−negative matrix factorization

NMF approximates a matrix ***V*** with a low-rank matrix approximation such that ***V≈W·H***



Non−negative matrix factorization

- For a given nonnegative matrix ***V***, it is possible to find two nonnegative matrices ***W*** and ***H*** such that ***V≈W·H*** holds (ideally).
- In ***V***, each column represents an example and each row represents a feature or an attribute.
- ***W*** is called the base matrix and ***H*** is called the weight matrix or coefficient matrix.



Non−negative matrix factorization

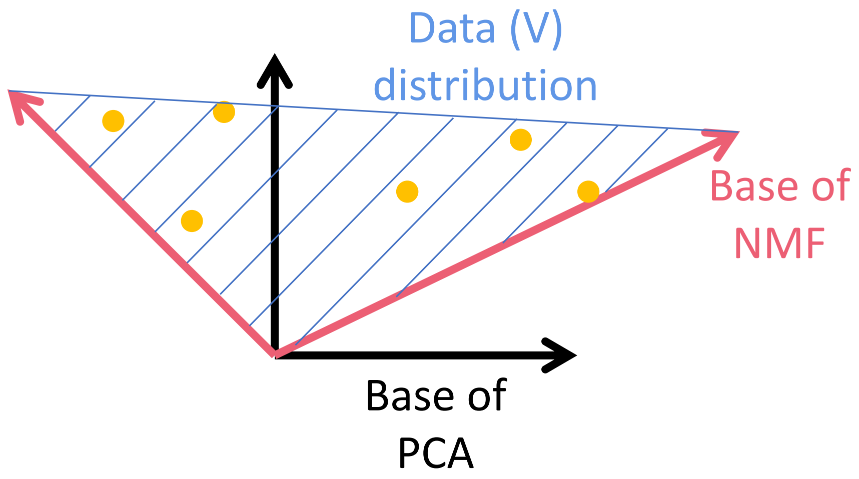
Comparison with Principal Component Analysis (PCA):

- PCA:
    - The base vectors are **orthogonal**. They may be distributed inside or outside the data distribution.
    - The reconstruction process simply sums the bases after multiplied by weights, and they do not affect each other.

- NMF:
    - The base vectors are the **edges of** the data **distribution**. They are not necessarily orthogonal.
    - The reconstruction process is realized by computing the matrix inner product.

Non−negative matrix factorization

Comparison with Principal Component Analysis (PCA):



Non−negative matrix factorization

Pros and Cons:

Pros:

1. Highly interpretable
2. The optimization is convex

   Therefore it can be optimized by gradient descent.

Cons:

1. Factorisation is not unique:

   Different optimization paths and optimization stopping times will lead to different results.

2. Feature redundancy among the bases

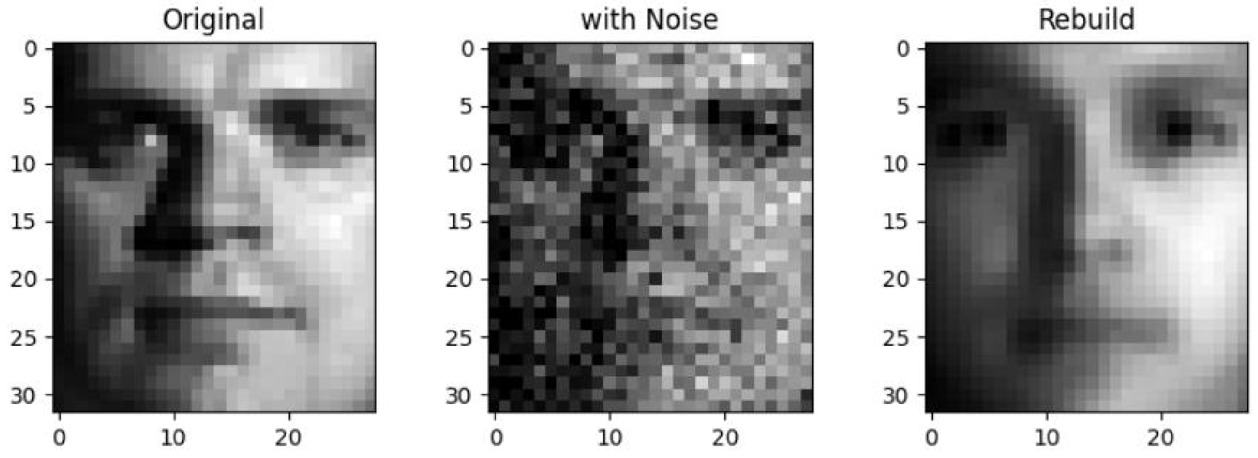
Non−negative matrix factorization

Applications of NMF:

- Compressed Storage
- Information Recovery
- Feature Extraction

Non-negative signals:

- Image
- Audio
- Natural Language
- Muscle Activity



Using NMF for Image Denoising

Non−negative matrix factorization

- The main implementation methods of NMF:

  Bounded optimization

- The objective of optimization:

$$\operatorname{argmin}_{W,H} \|V - W \cdot H\|_F^2$$

- Major Iterative Methods
  - Multiplicative Update Methods

$$W_{ia}^{k+1} = W_{ia}^{k} \frac{(V(H^k)^T)_{ia}}{(W^k H^k (H^k)^T)_{ia}}, \quad \forall i, a$$

$$H_{bj}^{k+1} = H_{bj}^{k} \frac{((W^{k+1})^T V)_{bj}}{((W^{k+1})^T W^{k+1} H^k)_{bj}}, \quad \forall b, j.$$

  - GradientApproaches

$$W^{k+1} = \max(0, W^k - \alpha_k \nabla_W f(W^k, H^k)),$$

$$H^{k+1} = \max(0, H^k - \alpha_k \nabla_H f(W^k, H^k)),$$

Lee, Daniel D., and H. Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization." Nature 401.6755 (1999): 788.

sklearn.decomposition.NMF ¶

class sklearn.decomposition.NMF(*n_components=None*, ***, *init=None*, *solver='cd'*, *beta_loss='frobenius'*, *tol=0.0001*, *max_iter=200*, *random_state=None*, *alpha='deprecated'*, *alpha_W=0.0*, *alpha_H='same'*, *l1_ratio=0.0*, *verbose=0*, *shuffle=False*, *regularization='deprecated'*) [source]

Non-Negative Matrix Factorization (NMF).

Find two non-negative matrices, i.e. matrices with all non-negative elements, (W, H) whose product approximates the non-negative matrix X. This factorization can be used for example for dimensionality reduction, source separation or topic extraction.

The objective function is:

$$\begin{aligned} L(W, H) &= 0.5 * ||X - WH||_{loss}^2 \\ &+ alpha\_W * l1\_ratio * n\_features * ||vec(W)||_1 \\ &+ alpha\_H * l1\_ratio * n\_samples * ||vec(H)||_1 \\ &+ 0.5 * alpha\_W * (1 - l1\_ratio) * n\_features * ||W||_{Fro}^2 \\ &+ 0.5 * alpha\_H * (1 - l1\_ratio) * n\_samples * ||H||_{Fro}^2 \end{aligned}$$

Where:

$||A||_{Fro}^2 = \sum_{i,j} A_{ij}^2$ (Frobenius norm)

$||vec(A)||_1 = \sum_{i,j} abs(A_{ij})$ (Elementwise L1 norm)

The generic norm $||X - WH||_{loss}$ may represent the Frobenius norm or another supported beta-divergence loss. The choice between options is controlled by the `beta_loss` parameter.

The regularization terms are scaled by `n_features` for `W` and by `n_samples` for `H` to keep their impact balanced with respect to one another and to the data fit term as independent as possible of the size `n_samples` of the training set.

The objective function is minimized with an alternating minimization of W and H.

Parameters:

n_components : *int, default=None*

Number of components, if n_components is not set all features are kept.

init : *{'random', 'nndsvd', 'nndsvda', 'nndsvdar', 'custom'}, default=None*

Method used to initialize the procedure. Default: None. Valid options:

- `None`: 'nndsvda' if n_components <= min(n_samples, n_features), otherwise random.
- `'random'`: non-negative random matrices, scaled with: sqrt(X.mean() / n_components)
- `'nndsvd'`: Nonnegative Double Singular Value Decomposition (NNDSVD) initialization (better for sparseness)
- `'nndsvda'`: NNDSVD with zeros filled with the average of X (better when sparsity is not desired)
- `'nndsvdar'` NNDSVD with zeros filled with small random values (generally faster, less accurate alternative to NNDSVDa for when sparsity is not desired)
- `'custom'`: use custom matrices W and H

Changed in version 1.1: When `init=None` and n_components is less than n_samples and n_features defaults to `nndsvda` instead of `nndsvd`.

solver : *{'cd', 'mu'}, default='cd'*

Numerical solver to use: 'cd' is a Coordinate Descent solver. 'mu' is a Multiplicative Update solver.

New in version 0.17: Coordinate Descent solver.

New in version 0.19: Multiplicative Update solver.

beta_loss : *float or {'frobenius', 'kullback-leibler', 'itakura-saito'}, default='frobenius'*

Beta divergence to be minimized, measuring the distance between X and the dot product WH. Note that values different from 'frobenius' (or 2) and 'kullback-leibler' (or 1) lead to significantly slower fits. Note that for beta_loss <= 0 (or 'itakura-saito'), the input matrix X cannot contain zeros. Used only in 'mu' solver.

Examples

```
>>> import numpy as np
>>> X = np.array([[1, 1], [2, 1], [3, 1.2], [4, 1], [5, 0.8], [6, 1]])
>>> from sklearn.decomposition import NMF
>>> model = NMF(n_components=2, init='random', random_state=0)
>>> W = model.fit_transform(X)
>>> H = model.components_
```