Non Negative Matrix Decomposition

Non Negative Matrix Decomposition is another way of reducing the number of dimensions. Similar to PCA, it is also a matrix decomposition method in the form V=WxH.

The main difference is that it can only be applied to matrices that have positive values as inputs, for example:

- pixels in a matrix
- positive attributes that can be zero or higher

In the case of word and vocabulary recognition, each row in the matrix can be considered a document, while each column can be considered a topic.

NMF has proven to be powerful for:

- word and vocabulary recognition
- image processing,
- text mining
- transcribing
- encoding and decoding
- decomposition of video, music, or images

There are advantages and disadvantages of only dealing with non negative values.

An advantage, is that NMF leads to features that tend to be more interpretable. For example, in facial recognition, the decomposed components match to something more interpretable like, for example, the nose, the eyebrows, or the mouth.

A disadvantage is that NMF truncates negative values by default to impose the added constraint of only positive values. This truncation tends to lose more information than other decomposition methods.

Unlike PCA, it does not have to use orthogonal latent vectors, and can end up using vectors that point in the same direction.

NMF for NLP

In the case of Natural Language Processing, NMF works as below given these inputs, parameters to tune, and outputs:

Inputs

Given vectorized inputs, which are usually pre-processed using count vectorizer or vectorizers in the form of Term Frequency - Inverse Document Frequency (TF-IDF).

Parameters to tune

The main two parameters are:

- Number of Topics
- Text Preprocessing (stop words, min/max document frequency, parts of speech, etc)

Output

The output of NMF will be two matrices:

1. W Matrix telling us how the terms relate to the different topics.
2. H Matrix telling us how to use those topics to reconstruct our original documents.

Syntax

The syntax consists of importing the class containing the clustering method:

```
from sklearn.decomposition import NMF
```

creating the instance of the class:

```
nmf=NMF(n_components=3, init='random')
```

and fit the instance and create a transformed version of the data:

```
x_nmf=NMF.fit(X)
```