Topic Modeling

CS5154/6054

Yizong Cheng

11/15/2022

18 *Matrix decompositions and latent semantic indexing*

Theorem 18.1. (Matrix diagonalization theorem) *Let S be a square real-valued $M \times M$ matrix with M linearly independent eigenvectors. Then there exists an* eigen decomposition

EIGEN DECOMPOSITION

$$S = U\Lambda U^{-1}, \tag{18.5}$$

where the columns of U are the eigenvectors of S and Λ is a diagonal matrix whose diagonal entries are the eigenvalues of S in decreasing order

$$\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \cdots & \\ & & & \lambda_M \end{pmatrix}, \lambda_i \geq \lambda_{i+1}. \tag{18.6}$$

If the eigenvalues are distinct, then this decomposition is unique.

Theorem 18.2. (Symmetric diagonalization theorem) *Let S be a square, symmetric real-valued $M \times M$ matrix with M linearly independent eigenvectors. Then there exists a* symmetric diagonal decomposition

SYMMETRIC DIAGONAL DECOMPOSITION

$$S = Q\Lambda Q^T, \tag{18.8}$$

where the columns of Q are the orthogonal and normalized (unit length, real) eigenvectors of S, and Λ is the diagonal matrix whose entries are the eigenvalues of S. Further, all entries of Q are real and we have $Q^{-1} = Q^T$.

18.2 Term-document matrices and singular value decompositions

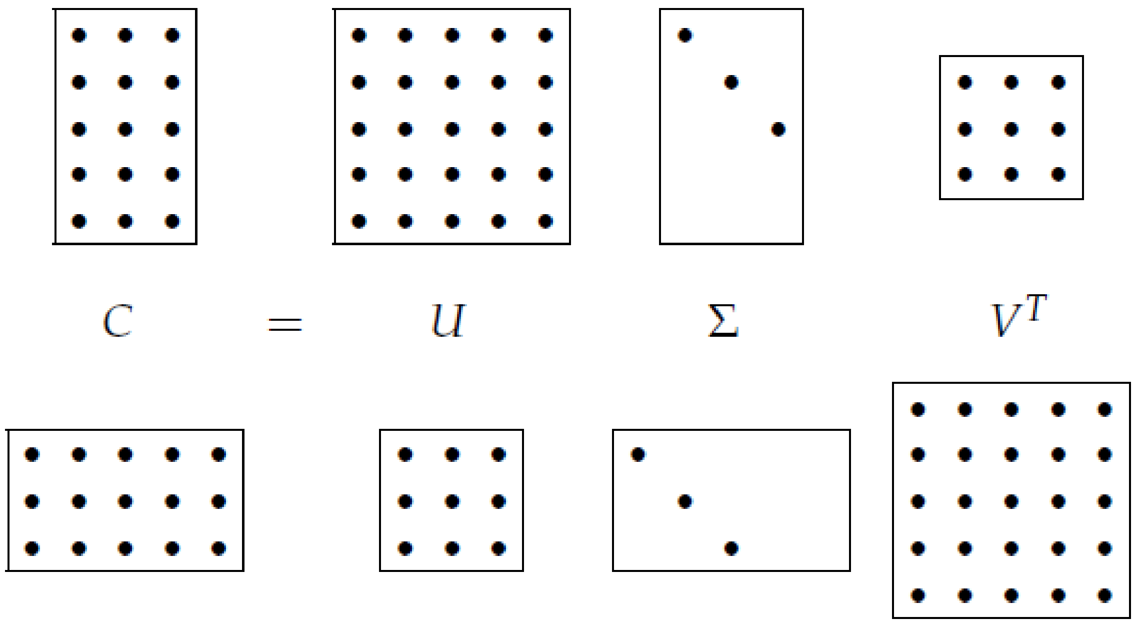
Theorem 18.3. *Let r be the rank of the $M \times N$ matrix C. Then, there is a* singular-
SVD value decomposition *(SVD for short) of C of the form*

$$C = U\Sigma V^T, \tag{18.9}$$

where

1. *The eigenvalues $\lambda_1, \ldots, \lambda_r$ of CC^T are the same as the eigenvalues of $C^T C$;*
2. *For $1 \leq i \leq r$, let $\sigma_i = \sqrt{\lambda_i}$, with $\lambda_i \geq \lambda_{i+1}$. Then the $M \times N$ matrix Σ is composed by setting $\Sigma_{ii} = \sigma_i$ for $1 \leq i \leq r$, and zero otherwise.*

The values σ_i are referred to as the *singular values* of C. It is instructive to



▶ **Figure 18.1** Illustration of the singular-value decomposition. In this schematic illustration of (18.9), we see two cases illustrated. In the top half of the figure, we have a matrix C for which $M > N$. The lower half illustrates the case $M < N$.

Example 18.3: We now illustrate the singular-value decomposition of a 4×2 matrix of rank 2; the singular values are $\Sigma_{11} = 2.236$ and $\Sigma_{22} = 1$.

$$C = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -0.632 & 0.000 \\ 0.316 & -0.707 \\ -0.316 & -0.707 \\ 0.632 & 0.000 \end{pmatrix} \begin{pmatrix} 2.236 & 0.000 \\ 0.000 & 1.000 \end{pmatrix} \begin{pmatrix} -0.707 & 0.707 \\ -0.707 & -0.707 \end{pmatrix}. \tag{18.11}$$

As with the matrix decompositions defined in Section 18.1.1, the singular value decomposition of a matrix can be computed by a variety of algorithms, many of which have been publicly available software implementations; pointers to these are given in Section 18.5.

?

Exercise 18.4

Let

$$C = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{18.12}$$

be the term-document incidence matrix for a collection. Compute the co-occurrence matrix CC^T. What is the interpretation of the diagonal entries of CC^T when C is a term-document incidence matrix?

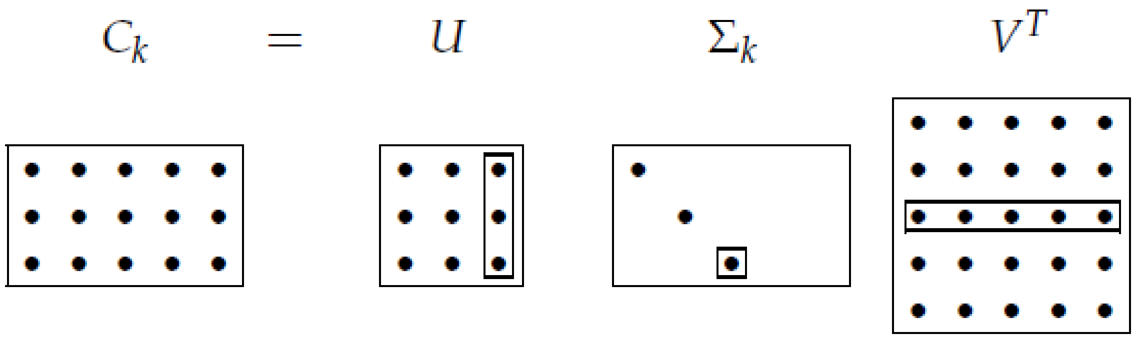
Exercise 18.5

Verify that the SVD of the matrix in Equation (18.12) is

$$U = \begin{pmatrix} -0.816 & 0.000 \\ -0.408 & -0.707 \\ -0.408 & 0.707 \end{pmatrix}, \Sigma = \begin{pmatrix} 1.732 & 0.000 \\ 0.000 & 1.000 \end{pmatrix} \text{ and } V^T = \begin{pmatrix} -0.707 & -0.707 \\ 0.707 & -0.707 \end{pmatrix}, \tag{18.13}$$

by verifying all of the properties in the statement of Theorem 18.3.

18.3 Low-rank approximations



▶ **Figure 18.2** Illustration of low rank approximation using the singular-value decomposition. The dashed boxes indicate the matrix entries affected by “zeroing out” the smallest singular values.

Theorem 18.4.

$$\min_{Z|\ rank(Z)=k} \|C - Z\|_F = \|C - C_k\|_F = \sigma_{k+1}. \tag{18.16}$$

$$
\begin{aligned}
C_k &= U\Sigma_k V^T && (18.17)\\
&= U \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \sigma_k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots \end{pmatrix} V^T && (18.18)\\
&= \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T, && (18.19)
\end{aligned}
$$

Example184.txt

Example 18.4: Consider the term-document matrix $C =$

	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
voyage	1	0	0	1	1	0
trip	0	0	0	1	0	1

```
1 0 1 0 0 0
0 1 0 0 0 0
1 1 0 0 0 0
1 0 0 1 1 0
0 0 0 1 0 1
```

Select IPython: C:classes/6054

```
C:\classes\6054>ipython
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
 (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.3.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import numpy as np

In [2]: import numpy.linalg as la

In [3]: C = np.genfromtxt('example184.txt', delimiter=' ')

In [4]: C
Out[4]:
array([[1., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [1., 1., 0., 0., 0., 0.],
       [1., 0., 0., 1., 1., 0.],
       [0., 0., 0., 1., 0., 1.]])
```

Its singular value decomposition is the product of three matrices as below. First we have U which in this example is:

	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
voyage	−0.70	0.35	0.15	−0.58	0.16
trip	−0.26	0.65	−0.41	0.58	−0.09

When applying the SVD to a term-document matrix, U is known as the *SVD term matrix*. The singular values are $\Sigma =$

$$\begin{matrix} 2.16 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.59 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.28 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.39 \end{matrix}$$

IPython: C:classes/6054

```
In [5]: u, s, vt = la.svd(C)

In [6]: u
Out[6]:
array([[ 4.40347480e-01, -2.96174360e-01, -5.69497581e-01,
         5.77350269e-01, -2.46402144e-01],
       [ 1.29346349e-01, -3.31450692e-01,  5.87021697e-01,
         7.77156117e-16, -7.27197008e-01],
       [ 4.75530263e-01, -5.11115242e-01,  3.67689978e-01,
         4.44089210e-16,  6.14358412e-01],
       [ 7.03020318e-01,  3.50572409e-01, -1.54905878e-01,
        -5.77350269e-01, -1.59788154e-01],
       [ 2.62672838e-01,  6.46746769e-01,  4.14591704e-01,
         5.77350269e-01,  8.66139898e-02]])

In [7]: s
Out[7]: array([2.16250096, 1.59438237, 1.27529025, 1.        , 0.39391525])

In [8]:
```

Finally we have V^T, which in the context of a term-document matrix is known as the *SVD document matrix*:

	d_1	d_2	d_3	d_4	d_5	d_6
1	-0.75	-0.28	-0.20	-0.45	-0.33	-0.12
2	-0.29	-0.53	-0.19	0.63	0.22	0.41
3	0.28	-0.75	0.45	-0.20	0.12	-0.33
4	0.00	0.00	0.58	0.00	-0.58	0.58
5	-0.53	0.29	0.63	0.19	0.41	-0.22

IPython: C:classes/6054

```
In [7]: s
Out[7]: array([2.16250096, 1.59438237, 1.27529025, 1.        , 0.39391525])

In [8]: vt
Out[8]:
array([[ 7.48623048e-01,  2.79711603e-01,  2.03628802e-01,
         4.46563110e-01,  3.25095956e-01,  1.21467154e-01],
       [-2.86453991e-01, -5.28459139e-01, -1.85761186e-01,
         6.25520701e-01,  2.19879758e-01,  4.05640944e-01],
       [-2.79711603e-01,  7.48623048e-01, -4.46563110e-01,
         2.03628802e-01, -1.21467154e-01,  3.25095956e-01],
       [-5.55111512e-16,  1.11022302e-15,  5.77350269e-01,
         3.60822483e-16, -5.77350269e-01,  5.77350269e-01],
       [ 5.28459139e-01, -2.86453991e-01, -6.25520701e-01,
        -1.85761186e-01, -4.05640944e-01,  2.19879758e-01],
       [ 1.77575232e-16, -1.60672986e-16, -1.77575232e-16,
        -5.77350269e-01,  5.77350269e-01,  5.77350269e-01]])

In [9]:
```

IPython: C:classes/6054

```
       [ 1.77575232e-16, -1.60672986e-16, -1.77575232e-16,
        -5.77350269e-01,  5.77350269e-01,  5.77350269e-01]])

In [9]: s2 = np.append(np.diag(s), np.zeros((len(s), 1)), axis=1)

In [10]: s2
Out[10]:
array([[2.16250096, 0.        , 0.        , 0.        , 0.        ,
        0.        ],
       [0.        , 1.59438237, 0.        , 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 1.27529025, 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , 1.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.39391525,
        0.        ]])

In [11]:
```

IPython: C:classes/6054

```
       [0.        , 0.        , 0.        , 0.        , 0.39391525,
        0.        ]])

In [11]: Creconstructed = np.matmul(np.matmul(u, s2), vt)

In [12]: Creconstructed
Out[12]:
array([[ 1.00000000e+00,  1.80411242e-16,  1.00000000e+00,
        -3.95516953e-16,  9.02056208e-17, -5.34294831e-16],
       [ 5.55111512e-17,  1.00000000e+00, -5.55111512e-16,
         1.17961196e-16,  4.44089210e-16, -3.05311332e-16],
       [ 1.00000000e+00,  1.00000000e+00, -6.66133815e-16,
         2.28983499e-16,  2.63677968e-16, -3.19189120e-16],
       [ 1.00000000e+00,  7.07767178e-16, -4.09394740e-16,
         1.00000000e+00,  1.00000000e+00,  6.19296281e-16],
       [-4.82253126e-16, -4.54497551e-16, -8.53483950e-16,
         1.00000000e+00,  5.95010152e-16,  1.00000000e+00]])

In [13]:
```

Exercise 18.10

Exercise 18.9 can be generalized to rank k approximations: we let U'_k and V'_k denote the "reduced" matrices formed by retaining only the first k columns of U and V, respectively. Thus U'_k is an $M \times k$ matrix while ${V'}_k^T$ is a $k \times N$ matrix. Then, we have

$$C_k = U'_k \Sigma'_k {V'}_k^T, \tag{18.20}$$

where Σ'_k is the square $k \times k$ submatrix of Σ_k with the singular values $\sigma_1, \ldots, \sigma_k$ on the diagonal. The primary advantage of using (18.20) is to eliminate a lot of redundant columns of zeros in U and V, thereby explicitly eliminating multiplication by columns that do not affect the low-rank approximation; this version of the SVD is sometimes known as the reduced SVD or truncated SVD and is a computationally simpler representation from which to compute the low rank approximation.

Select IPython: C:classes/6054

```
In [13]: for i in range(2, 5):
    ...:     s[i] = 0
    ...:

In [14]: s
Out[14]: array([2.16250096, 1.59438237, 0.        , 0.        , 0.        ])

In [15]: s2 = np.append(np.diag(s), np.zeros((len(s), 1)), axis=1)

In [16]: C2 = np.matmul(np.matmul(u, s2), vt)

In [17]: C2
Out[17]:
array([[ 0.8481456 ,  0.51590232,  0.28162515,  0.12986018,  0.20574267,
        -0.07588249],
       [ 0.36077778,  0.35750764,  0.15512454, -0.20565325, -0.02526436,
        -0.18038889],
       [ 1.00327014,  0.71828543,  0.36077778, -0.05052871,  0.15512454,
        -0.20565325],
       [ 0.97800578,  0.12986018,  0.20574267,  1.0285345 ,  0.61713858,
         0.41139591],
       [ 0.12986018, -0.38604214, -0.07588249,  0.89867432,  0.41139591,
         0.4872784 ]])
```

By "zeroing out" all but the two largest singular values of Σ, we obtain $\Sigma_2 =$

$$\begin{matrix} 2.16 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.59 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{matrix}$$

From this, we compute $C_2 =$

	d_1	d_2	d_3	d_4	d_5	d_6
1	−1.62	−0.60	−0.44	−0.97	−0.70	−0.26
2	−0.46	−0.84	−0.30	1.00	0.35	0.65
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

Notice that the low-rank approximation, unlike the original matrix C, can have negative entries.

IPython: C:classes/6054

```
In [18]: import matplotlib.pyplot as plt

In [19]: T = ['ship', 'boat', 'ocean', 'voyage', 'trip']

In [20]: u2 = u[:, :2]

In [21]: plt.scatter(u[:, 0], u[:, 1])
Out[21]: <matplotlib.collections.PathCollection at 0x1b6231b9860>

In [22]: for i in range(5):
    ...:     plt.annotate(T[i], u2[i])
    ...:

In [23]: plt.show()

In [24]:
```

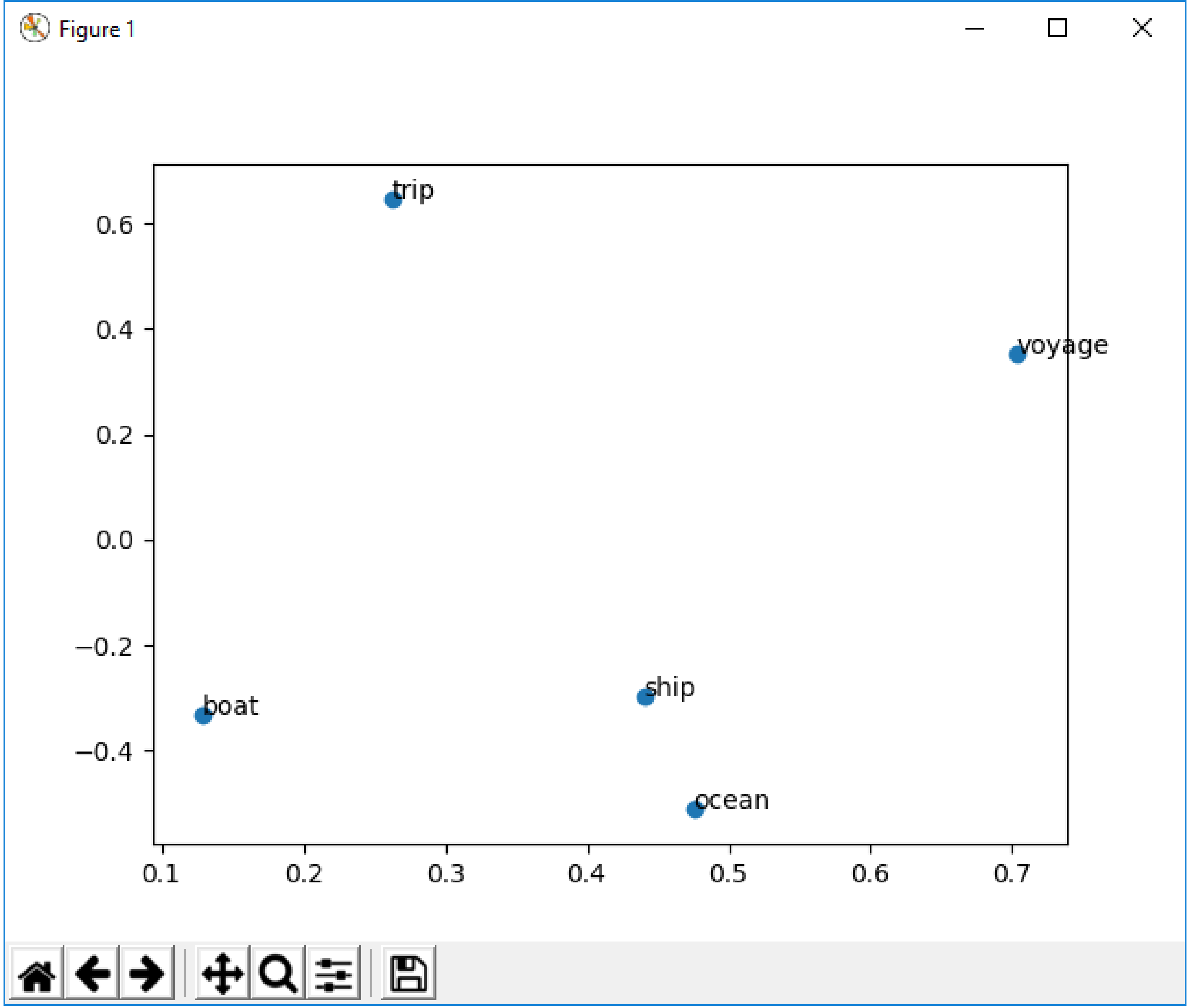


Figure 1
trip
voyage
boat
ship
ocean
0.6
0.4
0.2
0.0
−0.2
−0.4
0.1
0.2
0.3
0.4
0.5
0.6
0.7

IPython: C:classes/6054

```
In [22]: for i in range(5):
    ...:     plt.annotate(T[i], u2[i])
    ...:

In [23]: plt.show()

In [24]: v2 = np.transpose(vt)[:, :2]

In [25]: plt.scatter(v2[:, 0], v2[:, 1])
Out[25]: <matplotlib.collections.PathCollection at 0x1b62461e7f0>

In [26]: for i in range(6):
    ...:     plt.annotate((i + 1), v2[i])
    ...:

In [27]: plt.show()
```

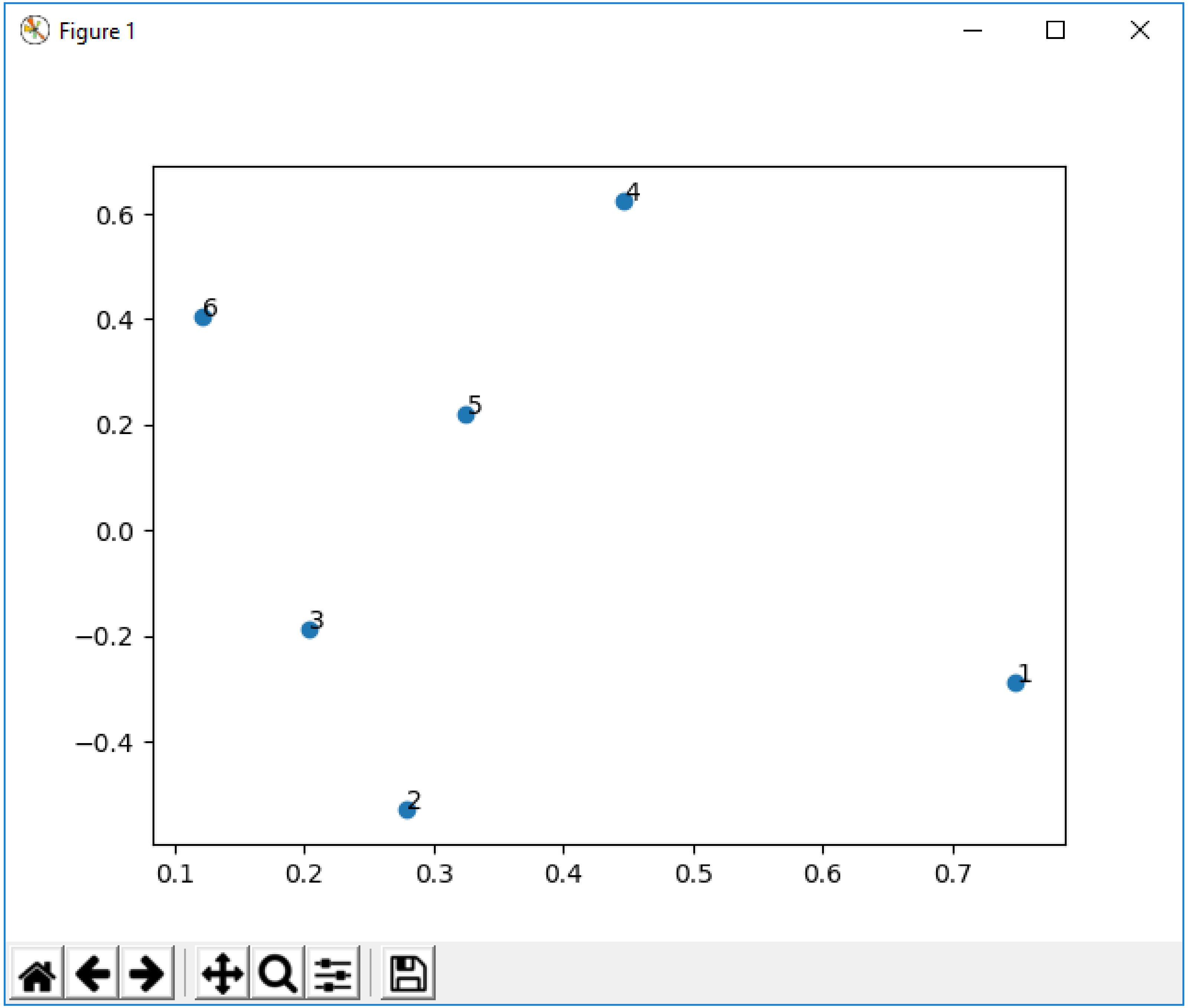
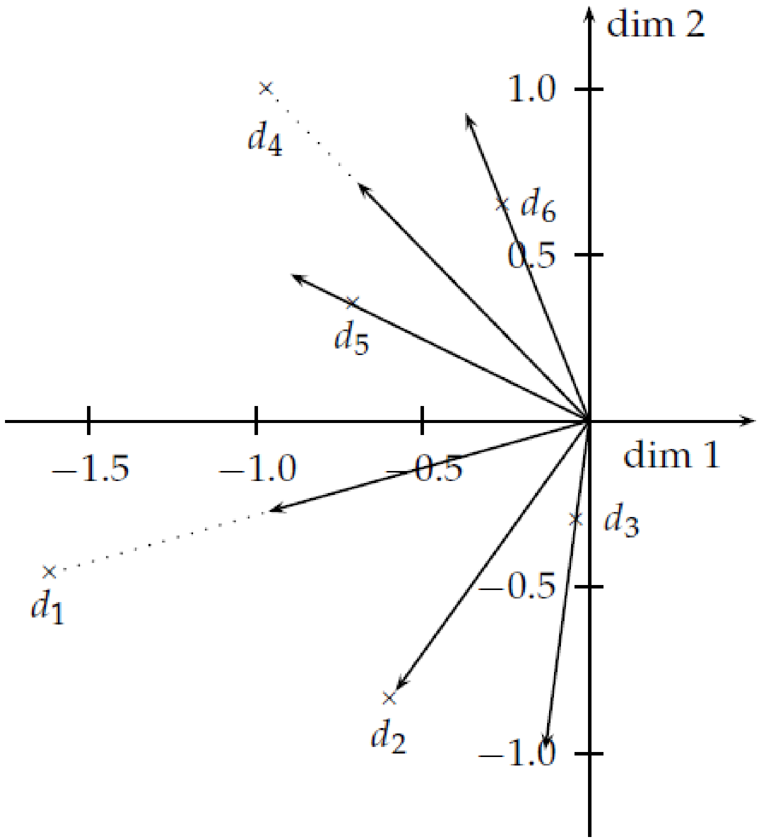


Figure 1
0.6
0.4
0.2
0.0
−0.2
−0.4
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1
2
3
4
5
6

Plotting Docs in 2D with $\Sigma'_2 V'^T$



igure 18.3 The documents of Example 18.4 reduced to two dimensions in $(V')^T$.

O'REILLY

Blueprints for Text Analytics Using Python

Machine Learning-Based Solutions for Common Real World (NLP) Applications

Jens Albrecht,
Sidharth Ramachandran
& Christian Winkler

Chapter 8. Unsupervised Methods: Topic Modeling and Clustering

Latent Semantic Analysis/Indexing

Blueprint: Creating a Topic Model for Paragraphs with SVD

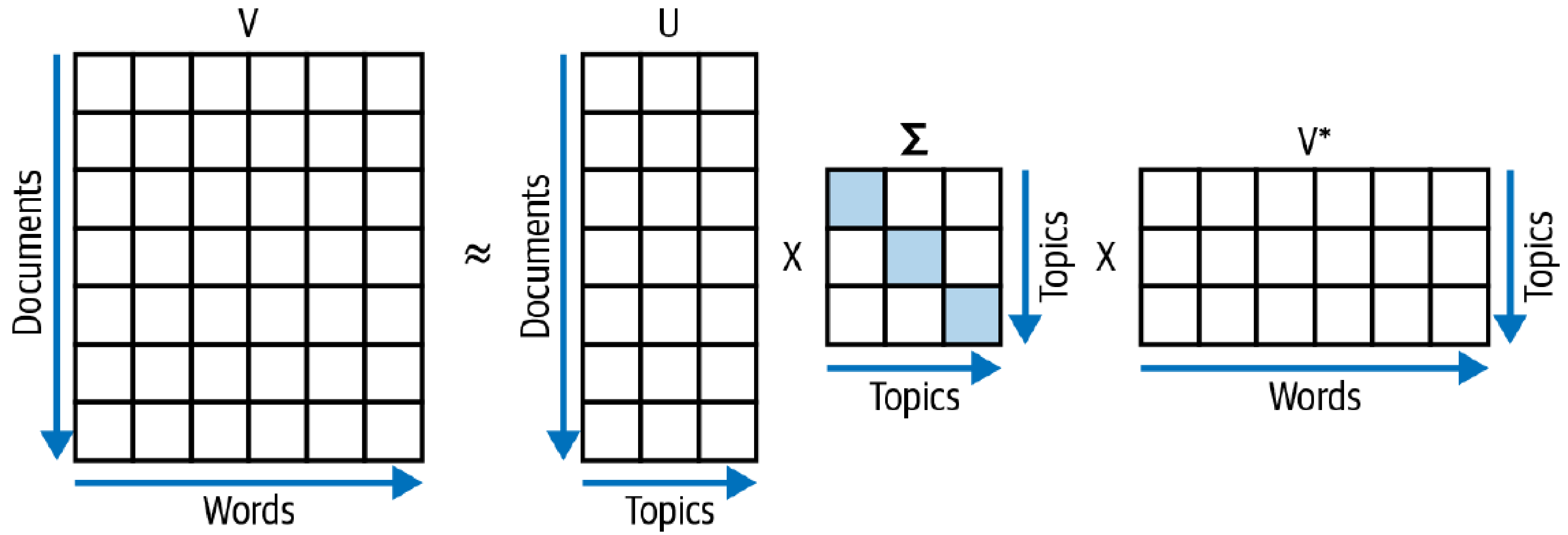


Figure 8-3. Schematic singular value decomposition.

sklearn.decomposition.TruncatedSVD

class `sklearn.decomposition.TruncatedSVD`(*n_components=2*, ***, *algorithm='randomized'*, *n_iter=5*, *n_oversamples=10*, *power_iteration_normalizer='auto'*, *random_state=None*, *tol=0.0*) [source]

Dimensionality reduction using truncated SVD (aka LSA).

This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse ntly.

Toggle Menu

File Edit View History Bookmarks Tools Help

sklearn.decomposition.Truncate

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition

150%

Attributes::

components_ : ***ndarray of shape (n_components, n_features)***

The right singular vectors of the input data.

explained_variance_ : ***ndarray of shape (n_components,)***

The variance of the training samples transformed by a projection to each component.

explained_variance_ratio_ : ***ndarray of shape (n_components,)***

Percentage of variance explained by each of the selected components.

singular_values_ : ***ndarray of shape (n_components,)***

The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-

Toggle Menu al space.

Nonnegative Matrix Factorization (NMF)

The conceptually easiest way to find a latent structure in the document corpus is the factorization of the document-term matrix. Fortunately, the document-term matrix has only positive-value elements; therefore, we can use methods from linear algebra that allow us to represent the matrix as the product of two other nonnegative matrices. Conventionally, the original matrix is called V, and the factors are W and H:

$$V \approx W \cdot H$$

Or we can represent it graphically (visualizing the dimensions necessary for matrix multiplication), as in Figure 8-1. Depending on the dimensions, the factorization can be performed exactly. But as this is so much more computationally expensive, an approximate factorization is sufficient.

In the context of text analytics, both W and H have an interpretation. The matrix W has the same number of rows as the document-term matrix and therefore maps documents to topics (document-topic matrix). H has the same number of columns as features, so it shows how the topics are constituted of features (topic-feature matrix). The number of topics (the columns of W and the rows of H) can be chosen arbitrarily. The smaller this number, the less exact the factorization.

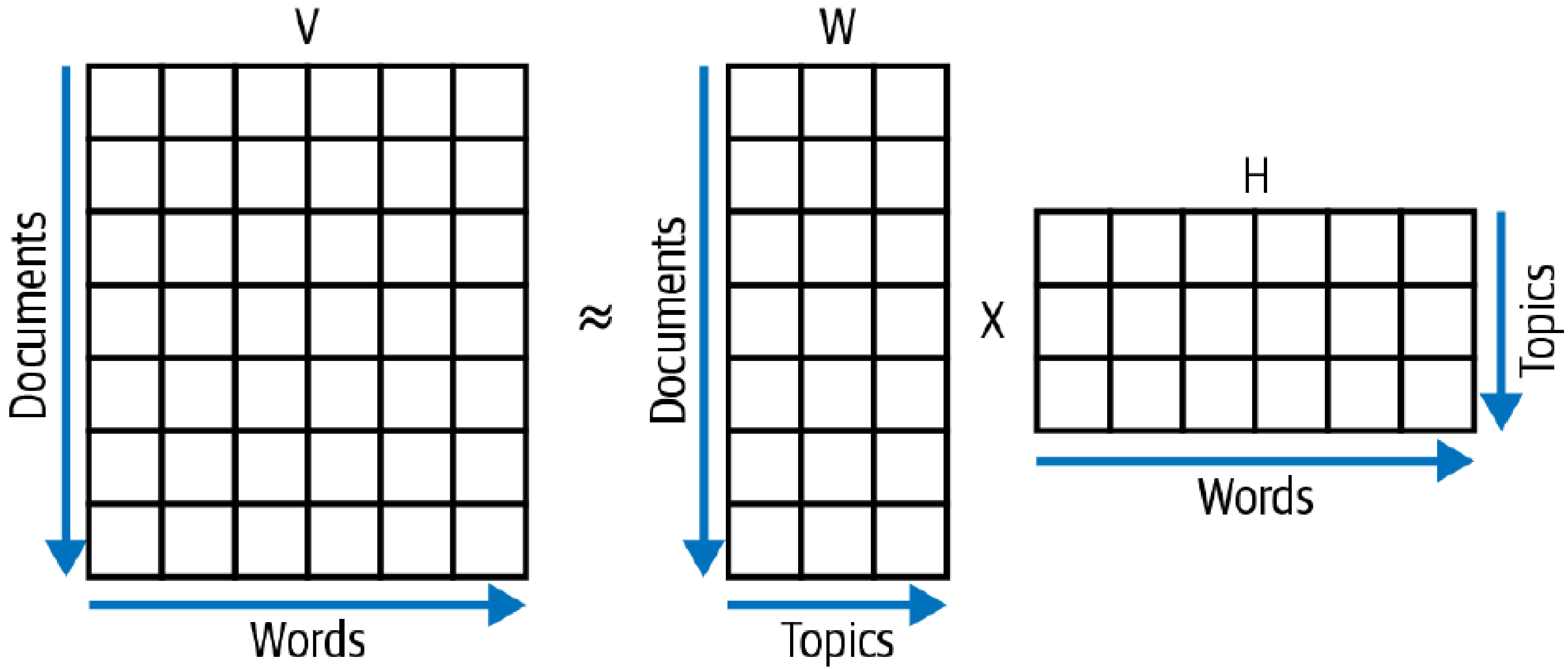


Figure 8-1. Schematic nonnegative matrix factorization; the original matrix V is decomposed into W and H.

File Edit View History Bookmarks Tools Help

sklearn.decomposition.NMF —

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition 150%

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.

Toggle Menu

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)

 $\sum_{i,j} abs(A_{ij})$ (Elementwise L1 norm)

components_ : *ndarray of shape (n_components, n_features)*

Factorization matrix, sometimes called 'dictionary'.

n_components_ : *int*

The number of components. It is same as the `n_components` parameter if it was given. Otherwise, it will be same as the number of features.

reconstruction_err_ : *float*

Frobenius norm of the matrix difference, or beta-divergence, between the training data `X` and the reconstructed data `WH` from the fitted model.

Toggle Menu

File Edit View History Bookmarks Tools Help

sklearn.decomposition.NMF —

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition 150%

```
>>> 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_
```

Methods

`fit`(X[, y])	Learn a NMF model for the data X.
`fit_transform`(X[, y, W, H])	Learn a NMF model for the data X and returns the transformed data.
`ames_out`([input_features])	Get output feature names for transformation.
eep])	Get parameters for this estimator.

Toggle Menu

File Edit View History Bookmarks Tools Help

sklearn.decomposition.LatentDi

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDir 150%

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]

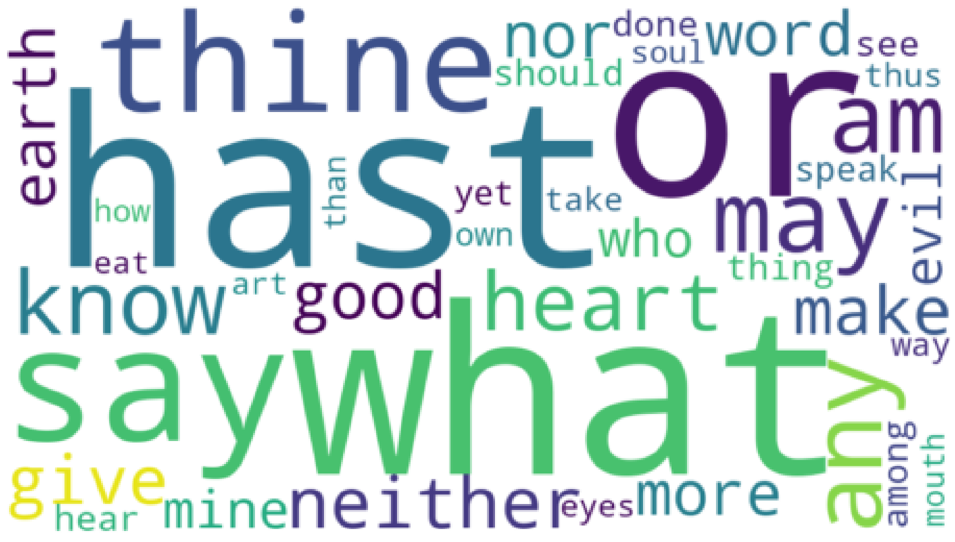
Latent Dirichlet Allocation with online variational Bayes algorithm.

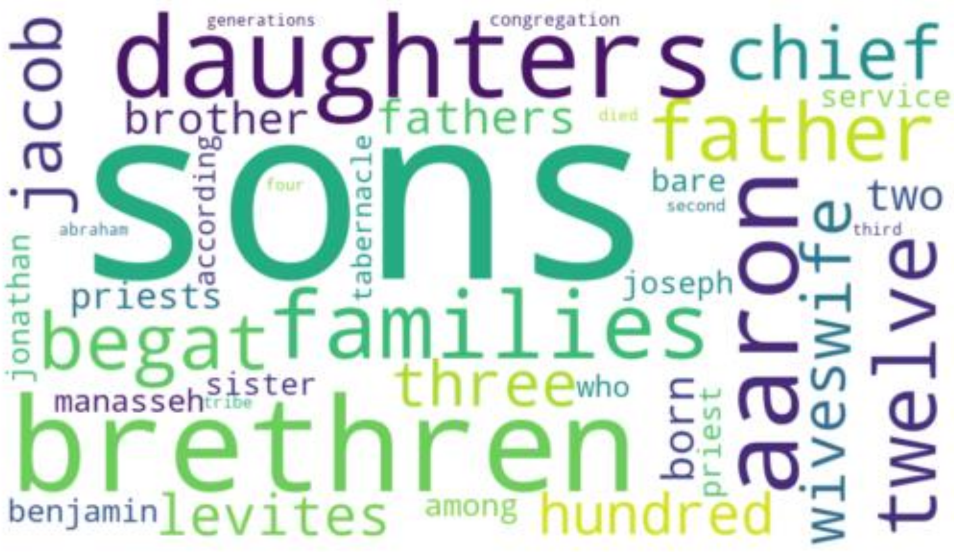
Toggle Menu

tation is based on [1] and [2].

```
# IR21A.py CS5154/6054 cheng 2022
# decompose the document-term matrix
# Usage: python IR21A.py

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from matplotlib import pyplot as plt

f = open("bible.txt", "r")
docs = f.readlines()
f.close()
N = len(docs)

vectorizer = TfidfVectorizer(max_df=1000, min_df=100)
X = vectorizer.fit_transform(docs)
words = vectorizer.get_feature_names()

nmf_model = NMF(n_components=5, init='nndsvda')
W = nmf_model.fit_transform(X).T
H = nmf_model.components_
```

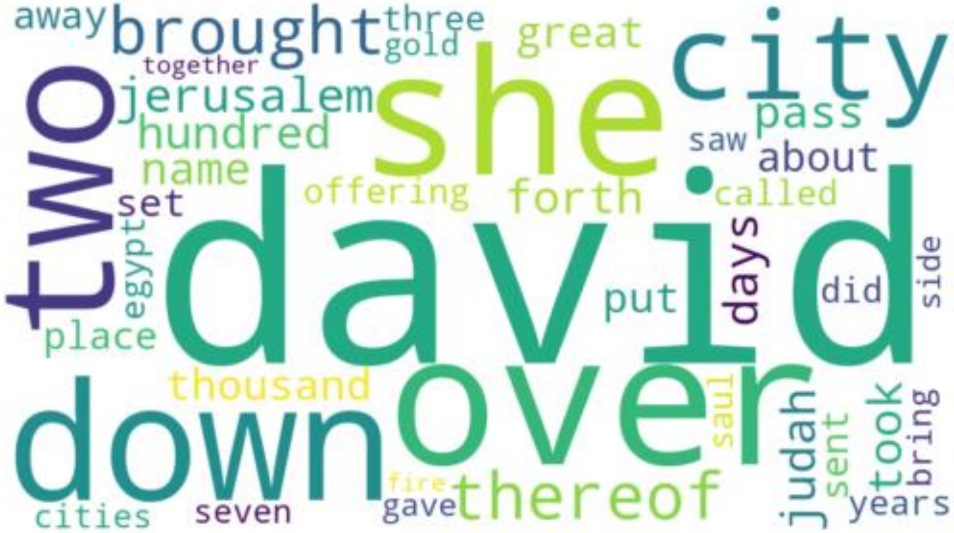
```
from wordcloud import WordCloud

for topic in range(5):
    size = {}
    largest = H[topic].argsort()[::-1]
    for i in range(40):
        size[words[largest[i]]] = abs(H[topic][largest[i]])
    wc = WordCloud(background_color="white", max_words=100, width=960, height=540)
    wc.generate_from_frequencies(size)
    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```




sons
daughters
brethren
families
aaron
begat
father
chief
jacob
twelve
wiveswife
three
hundred
levites
fathers
brother
priests
two
bare
joseph
manasseh
sister
born
benjamin
jonathan
according
tabernacle
priest
among
who
service
generations
congregation
abraham
second
third
four
died
tribe


our
jesus
christ
father
grace
brethren
answered
called
who
faith
glory
name
might
whom
disciples
spirit
through
peace
fathers
himself
love
heard
dead
paul
believe
being
should
brother
mother
abraham
world
jews
peter
again
kingdom
art
according
sins
body
saw


moses
spake
aaron
did
commanded
congregation
egypt
according
pharaoh
tabernacle
joshua
word
words
mount
servant
brought
law
heard
speak
voice
concerning
near
book
called
priest
tribe
stood
sight
sent
camp
written
commandment
jordan
gave
should
father
elders
spoken
levites
wilderness


david
she
over
down
two
city
brought
jerusalem
hundred
name
thereof
forth
great
three
gold
together
away
pass
about
saw
called
offering
set
egypt
place
put
did
days
side
thousand
judah
sent
took
bring
saul
years
fire
gave
seven
cities