UNSUPERVISED LEARNING IN PYTHON

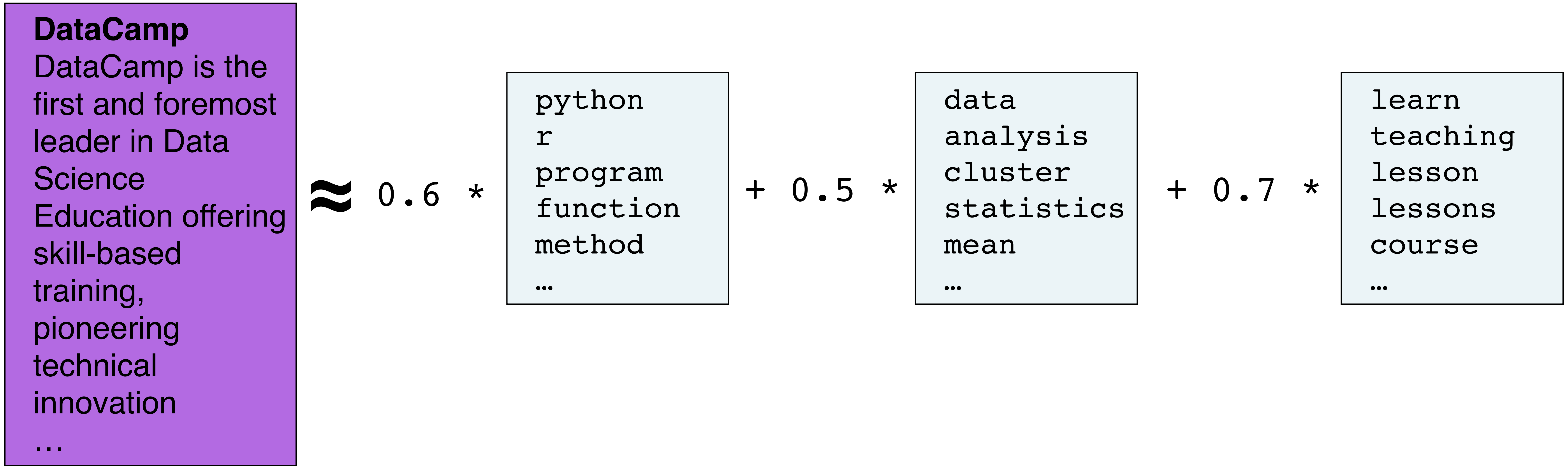
Non-negative matrix factorization (NMF)

Non-negative matrix factorization

- NMF = "non-negative matrix factorization"
- Dimension reduction technique
- NMF models are *interpretable* (unlike PCA)
- Easy to interpret means easy to explain!
- However, all sample features must be non-negative (>= 0)

Interpretable parts

- NMF expresses documents as combinations of topics (or "themes")



DataCamp
DataCamp is the first and foremost leader in Data Science Education offering skill-based training, pioneering technical innovation
…

≈ 0.6 *

```
python
r
program
function
method
…
```

+ 0.5 *

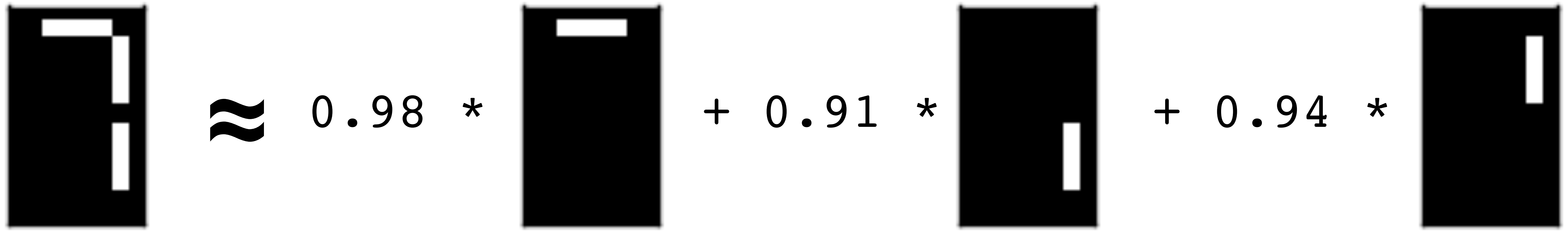
```
data
analysis
cluster
statistics
mean
…
```

+ 0.7 *

```
learn
teaching
lesson
lessons
course
…
```

Interpretable parts

- NMF expresses images as combinations of patterns



≈ 0.98 * + 0.91 * + 0.94 *

Using scikit-learn NMF

- Follows **fit() / transform()** pattern
- Must specify number of components e.g. **NMF(n_components=2)**
- Works with NumPy arrays and with **csr_matrix**

Example word-frequency array

- Word frequency array, 4 words, many documents
- Measure presence of words in each document using "tf-idf"
- "tf" = frequency of word in document
- "idf" reduces influence of frequent words

	course	datacamp	potato	the
document0	0.2,	0.3,	0.0,	0.1
document1	0.0,	0.0,	0.4,	0.1
...	...			

Example usage of NMF

- **samples** is the word-frequency array

```
In [1]: from sklearn.decomposition import NMF

In [2]: model = NMF(n_components=2)

In [3]: model.fit(samples)
Out[3]: NMF(alpha=0.0, ... )

In [4]: nmf_features = model.transform(samples)
```

NMF components

- NMF has components
- ... just like PCA has principal components
- Dimension of components = dimension of samples
- Entries are non-negative

```
In [5]: print(model.components_)
[[ 0.01  0.    2.13  0.54]
 [ 0.99  1.47  0.    0.5 ]]
```

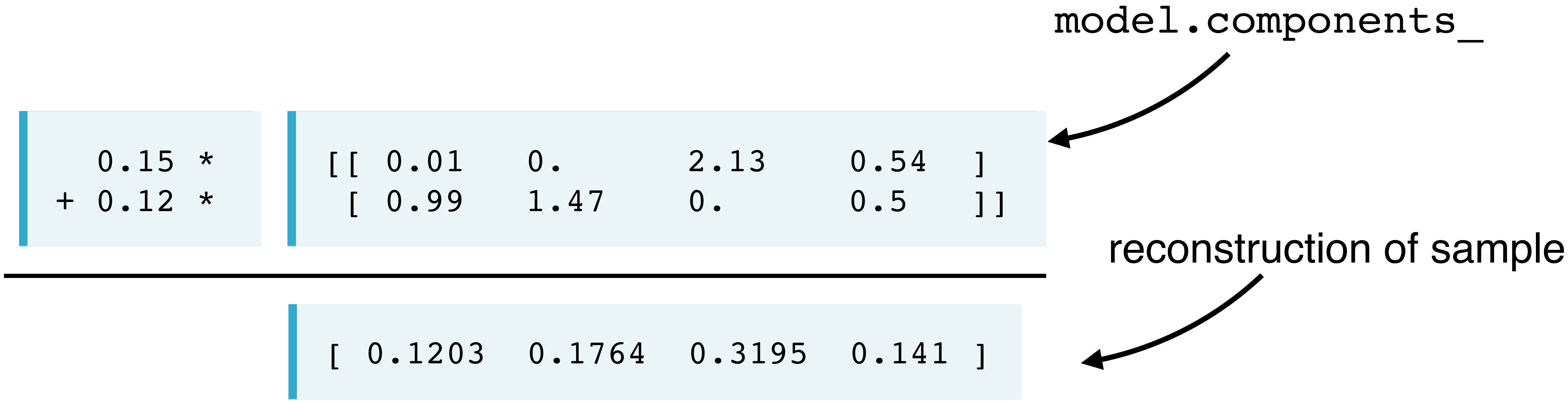
NMF features

- NMF feature values are non-negative
- Can be used to reconstruct the samples
- ... combine feature values with components

```
In [6]: print(nmf_features)
[[ 0.    0.2 ]
 [ 0.19  0.  ]
  ...
 [ 0.15  0.12]]
```

Reconstruction of a sample

```
In [7]: print(samples[i,:])
[ 0.12  0.18  0.32  0.14]

In [8]: print(nmf_features[i,:])
[ 0.15  0.12]
```



model.components_

```
  0.15 *
+ 0.12 *
```

```
[[ 0.01    0.      2.13    0.54  ]
 [ 0.99    1.47    0.      0.5   ]]
```

reconstruction of sample

```
[ 0.1203  0.1764  0.3195  0.141 ]
```

Sample reconstruction

- Multiply components by feature values, and add up
- Can also be expressed as a product of matrices
- This is the "**M**atrix **F**actorization" in "NMF"

NMF fits to non-negative data, only

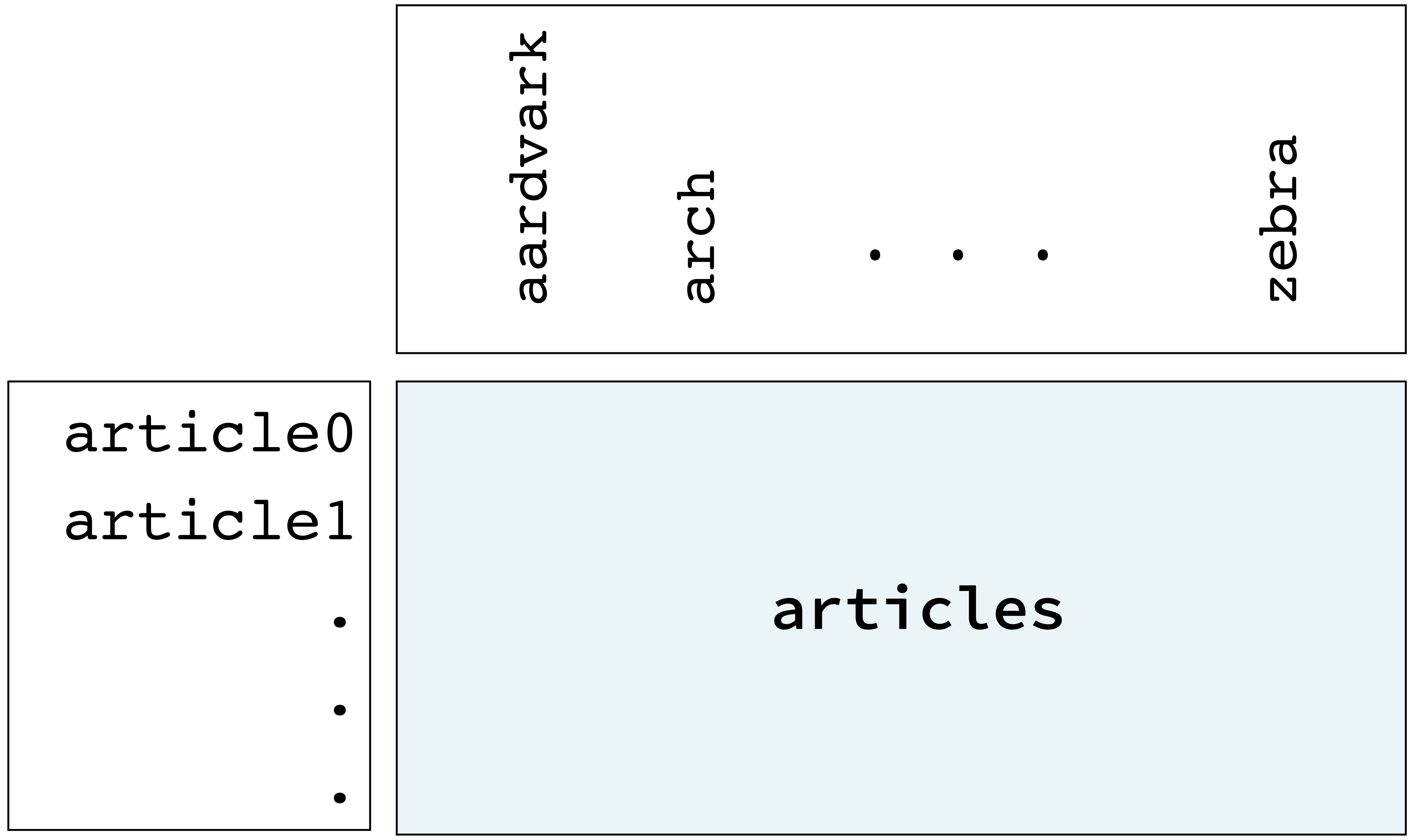
- Word frequencies in each document
- Images encoded as arrays
- Audio spectrograms
- Purchase histories on e-commerce sites
- ... and many more!

Let’s practice!

NMF learns interpretable parts

Example: NMF learns interpretable parts

- Word-frequency array **articles** (tf-idf)
- 20,000 scientific articles (rows)
- 800 words (columns)



aardvark arch . . . zebra

article0
article1
.
.
.

articles

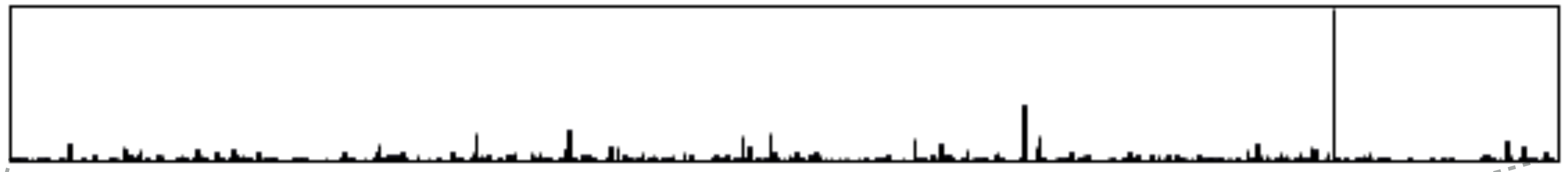
Applying NMF to the articles

```
In [1]: print(articles.shape)
(20000, 800)

In [2]: from sklearn.decomposition import NMF

In [3]: nmf = NMF(n_components=10)

In [4]: nmf.fit(articles)
Out[4]: NMF(alpha=0.0, ... )

In [5]: print(nmf.components_.shape)
(10, 800)
```

NMF components are topics

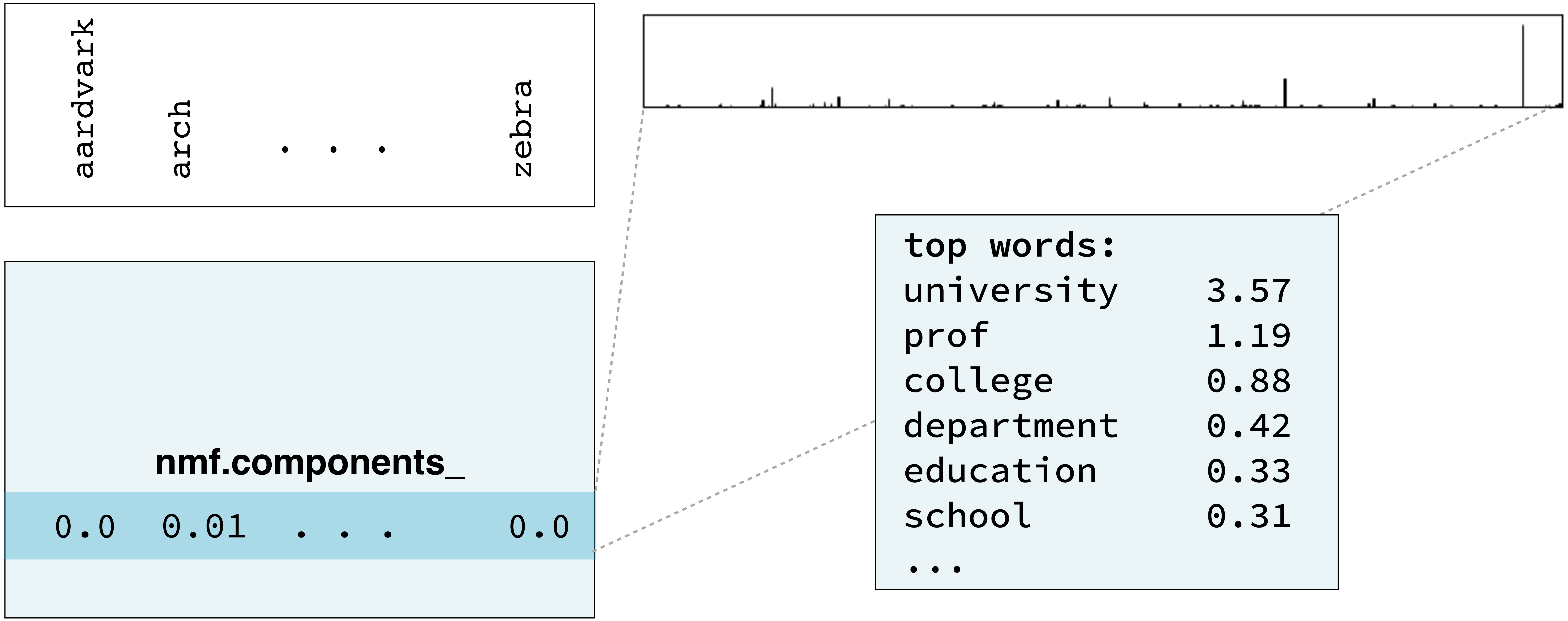
aardvark	arch	. . .	zebra
0.1	0.0	. . .	0.1

nmf.components_



```
top words:
species          2.95
plant            1.05
plants           0.78
genetic          0.58
evolution        0.53
life             0.53
...
```

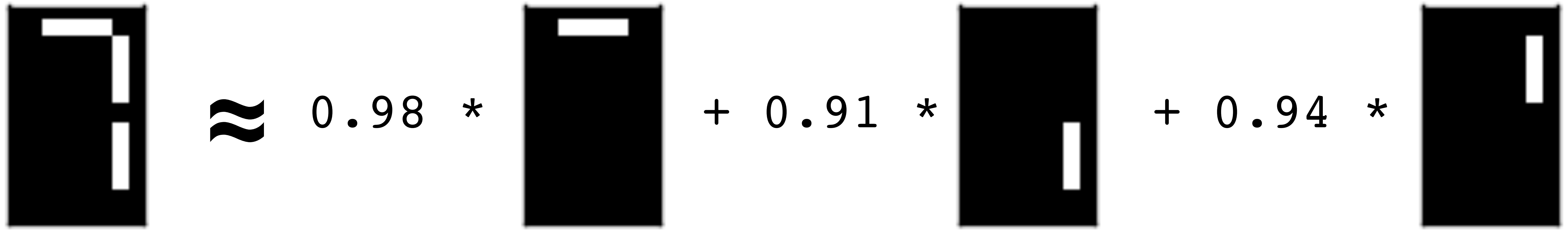
NMF components are topics



```
top words:
university     3.57
prof           1.19
college        0.88
department     0.42
education      0.33
school         0.31
...
```

NMF components

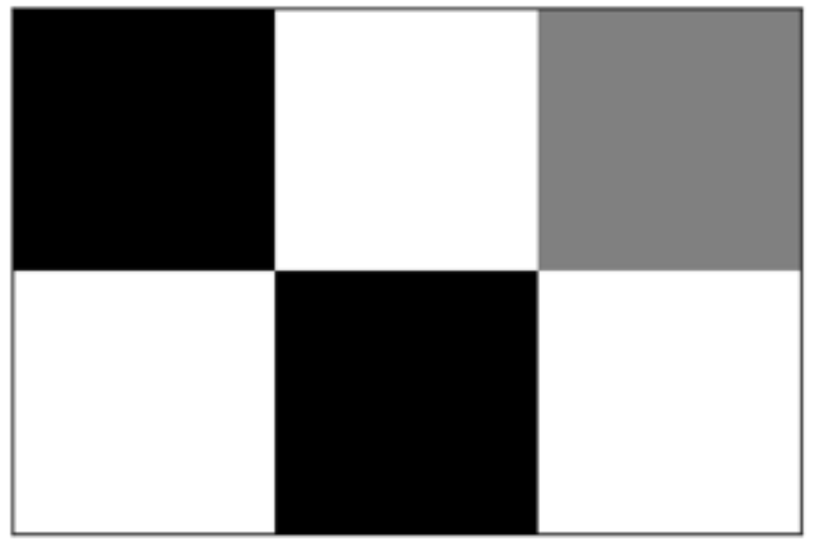
- For documents:
    - NMF components represent topics
    - NMF features combine topics into documents
- For images, NMF components are parts of images



≈ 0.98 * + 0.91 * + 0.94 *

Grayscale images

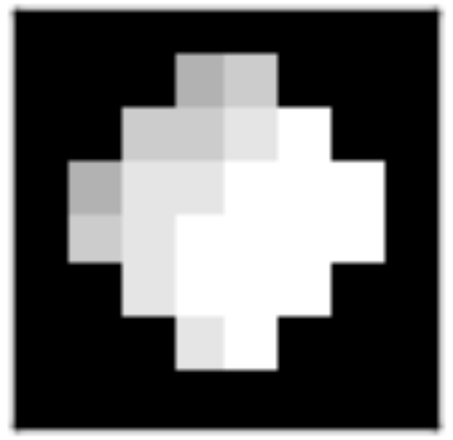
- "Grayscale" image = no colors, only shades of gray
- Measure pixel brightness
- Represent with value between 0 and 1 (0 is black)
- Convert to 2D array



```
[[ 0.   1.   0.5]
 [ 1.   0.   1. ]]
```

Grayscale image example

- An 8x8 grayscale image of the moon, written as an array



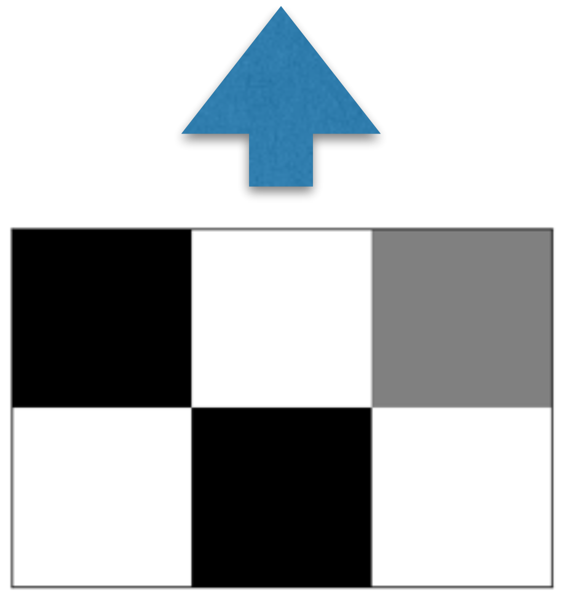
```
[[ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.7  0.8  0.   0.   0. ]
 [ 0.   0.   0.8  0.8  0.9  1.   0.   0. ]
 [ 0.   0.7  0.9  0.9  1.   1.   1.   0. ]
 [ 0.   0.8  0.9  1.   1.   1.   1.   0. ]
 [ 0.   0.   0.9  1.   1.   1.   0.   0. ]
 [ 0.   0.   0.   0.9  1.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]
```

Grayscale images as flat arrays

- Enumerate the entries
- Row-by-row
- From left to right

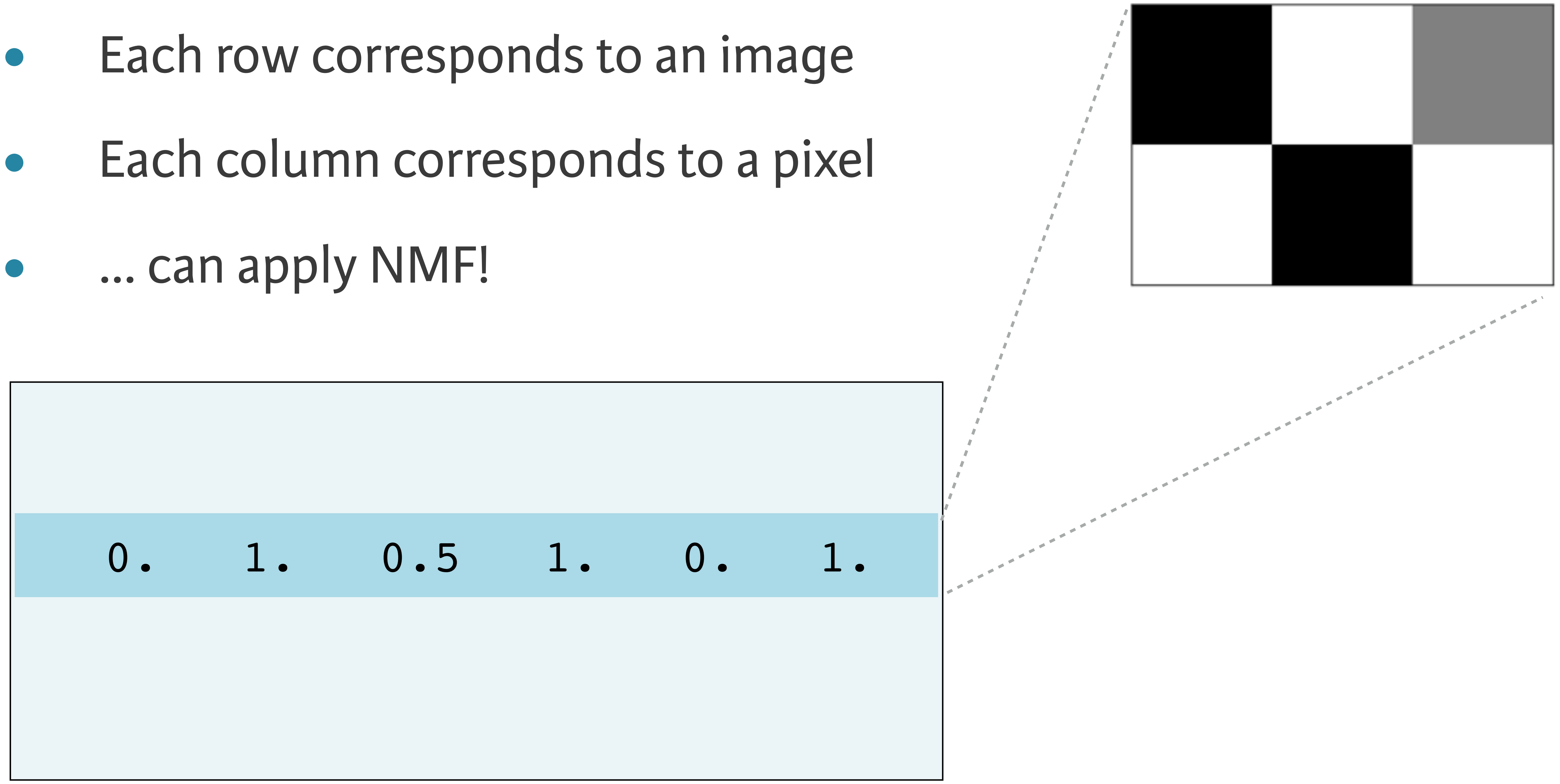
```
[[ 0.   1.   0.5]
 [ 1.   0.   1. ]]
```

```
[ 0.   1.   0.5   1.   0.   1. ]
```



Encoding a collection of images

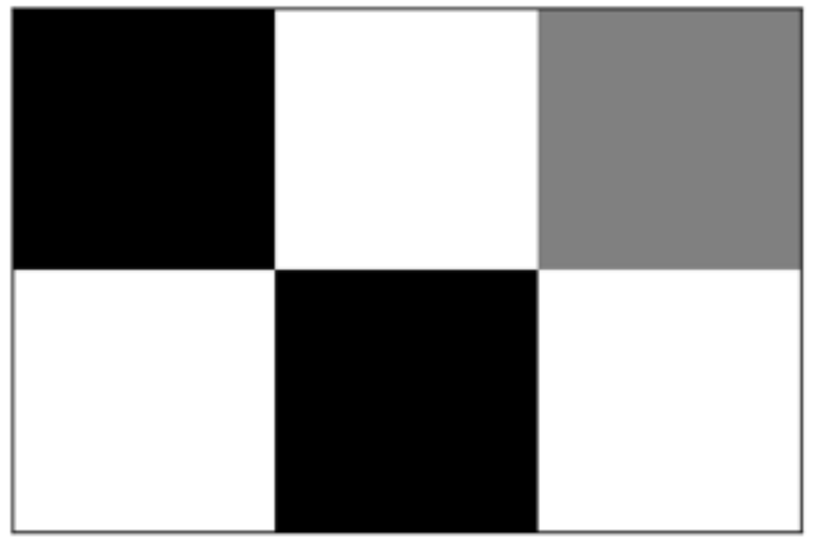
- Collection of images of the same size
- Encode as 2D array
- Each row corresponds to an image
- Each column corresponds to a pixel
- ... can apply NMF!



```
0.   1.   0.5   1.   0.   1.
```

Visualizing samples

```
In [1]: print(sample)
[ 0.   1.   0.5  1.   0.   1. ]

In [2]: bitmap = sample.reshape((2, 3))

In [3]: print(bitmap)
[[ 0.   1.   0.5]
 [ 1.   0.   1. ]]

In [4]: from matplotlib import pyplot as plt

In [5]: plt.imshow(bitmap, cmap='gray', interpolation='nearest')

In [6]: plt.show()
```



Let’s practice!

Building recommender systems using NMF

Finding similar articles

- Engineer at a large online newspaper
- Task: recommend articles similar to article being read by customer
- Similar articles should have similar topics

Strategy

- Apply NMF to the word-frequency array
- NMF feature values describe the topics
- ... so similar documents have similar NMF feature values
- Compare NMF feature values?

Apply NMF to the word-frequency array

- **articles** is a word frequency array

```
In [1]: from sklearn.decomposition import NMF

In [2]: nmf = NMF(n_components=6)

In [3]: nmf_features = nmf.fit_transform(articles)
```

Strategy

- Apply NMF to the word-frequency array
- NMF feature values describe the topics
- ... so similar documents have similar NMF feature values
- Compare NMF feature values?

Versions of articles

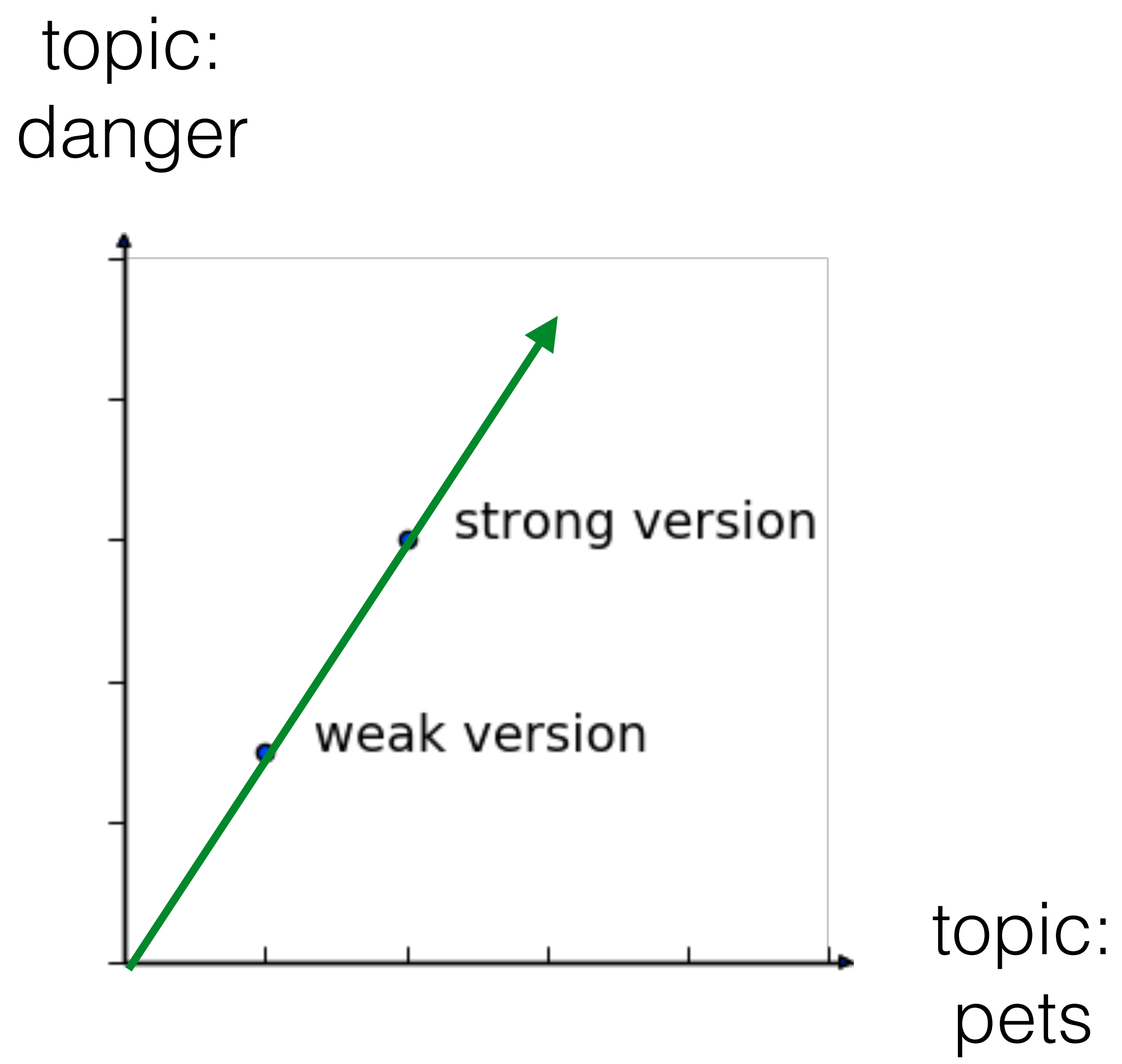
- Different versions of the same document have same topic *proportions*
- ... exact feature values may be different!
- E.g. because one version uses many meaningless words
- But all versions lie on the same *line* through the origin

strong version

Dog bites man!
Attack by terrible
canine leaves man
paralyzed...

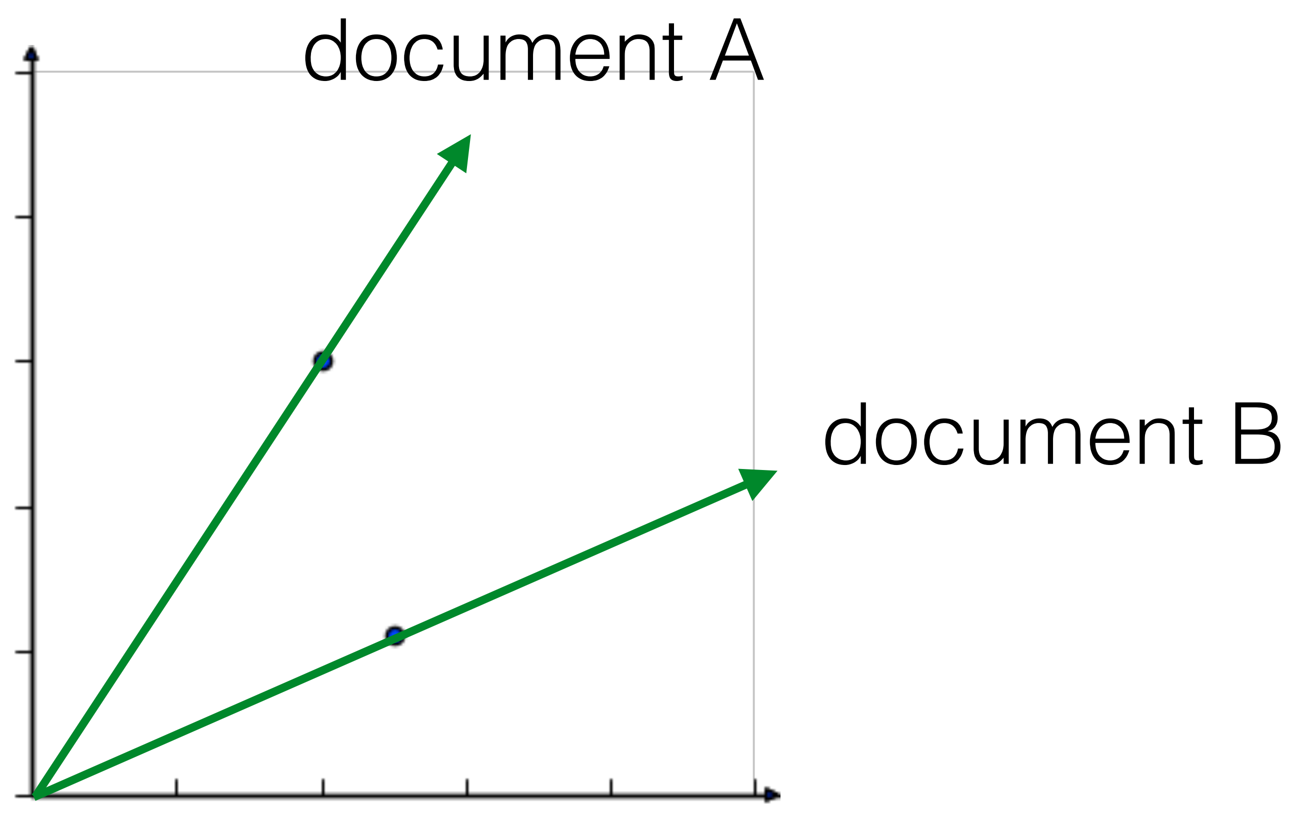
weak version

You may have heard,
unfortunately it seems
that a dog has perhaps
bitten a man ...



Cosine similarity

- Uses the angle between the lines
- Higher values means more similar
- Maximum value is 1, when angle is 0 °



Calculating the cosine similarities

```
In [4]: from sklearn.preprocessing import normalize

In [5]: norm_features = normalize(nmf_features)

In [6]: current_article = norm_features[23,:]  # if has index 23

In [7]: similarities = norm_features.dot(current_article)

In [8]: print(similarities)
[ 0.7150569   0.26349967  0.40210445 ...,  0.70462768  0.20323616
  0.05047817]
```

DataFrames and labels

- Label similarities with the article titles, using a DataFrame
- Titles given as a list: **titles**

```
In [9]: import pandas as pd

In [10]: norm_features = normalize(nmf_features)

In [11]: df = pd.DataFrame(norm_features, index=titles)

In [12]: current_article = df.loc['Dog bites man']

In [13]: similarities = df.dot(current_article)
```

DataFrames and labels

```
In [14]: print(similarities.nlargest())
Dog bites man                             1.000000
Hound mauls cat                           0.979946
Pets go wild!                             0.979708
Dachshunds are dangerous                  0.949641
Our streets are no longer safe            0.900474
dtype: float64
```

Let’s practice!

Building recommender systems using NMF

Finding similar articles

- Engineer at a large online newspaper
- Task: recommend articles similar to article being read by customer
- Similar articles should have similar topics

Strategy

- Apply NMF to the word-frequency array
- NMF feature values describe the topics
- ... so similar documents have similar NMF feature values
- Compare NMF feature values?

Apply NMF to the word-frequency array

- **articles** is a word frequency array

```
In [1]: from sklearn.decomposition import NMF

In [2]: nmf = NMF(n_components=6)

In [3]: nmf_features = nmf.fit_transform(articles)
```

Strategy

- Apply NMF to the word-frequency array
- NMF feature values describe the topics
- ... so similar documents have similar NMF feature values
- Compare NMF feature values?

Versions of articles

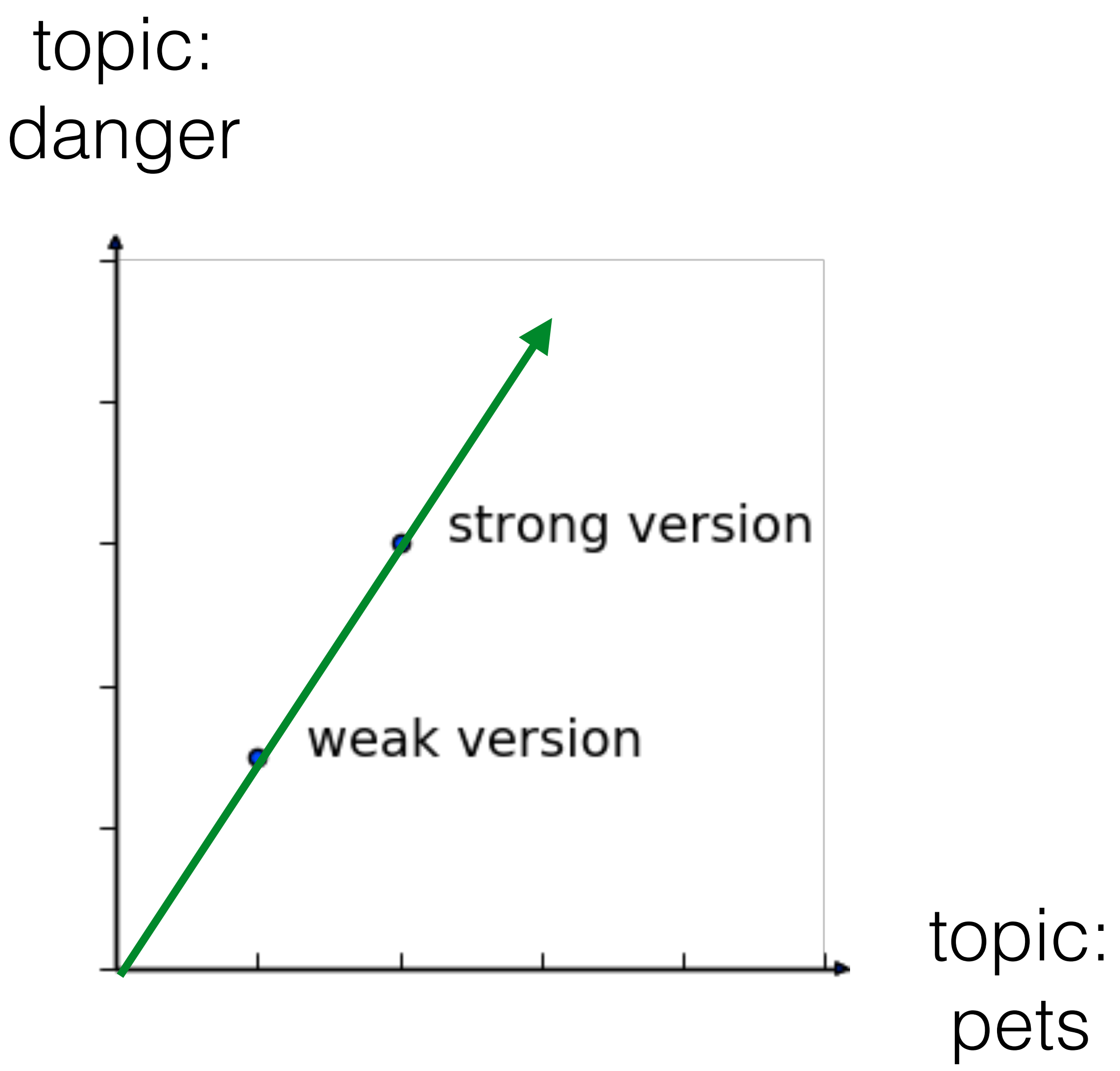
- Different versions of the same document have same topic *proportions*
- ... exact feature values may be different!
- E.g. because one version uses many meaningless words
- But all versions lie on the same *line* through the origin

strong version

Dog bites man!
Attack by terrible
canine leaves man
paralyzed...

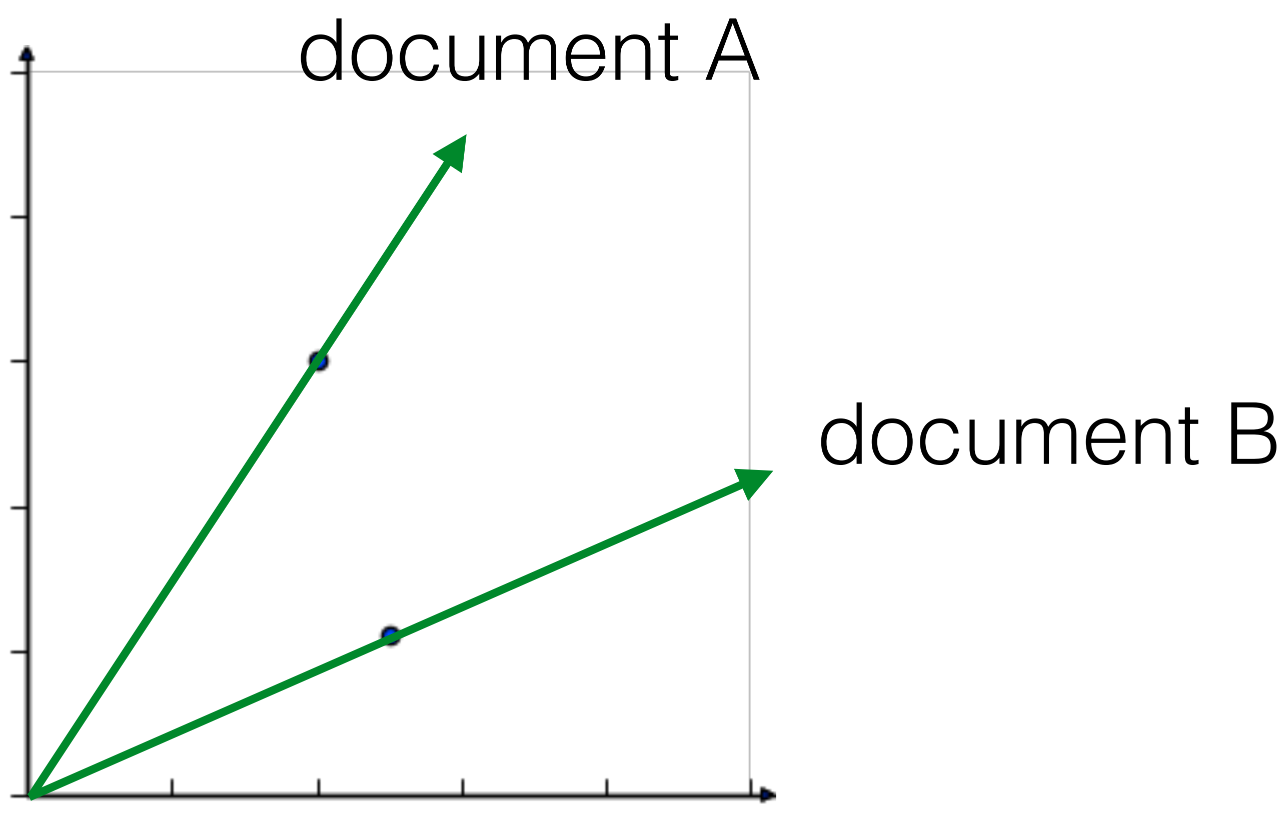
weak version

You may have heard,
unfortunately it seems
that a dog has perhaps
bitten a man ...



Cosine similarity

- Uses the angle between the lines
- Higher values means more similar
- Maximum value is 1, when angle is 0°



Calculating the cosine similarities

```
In [4]: from sklearn.preprocessing import normalize

In [5]: norm_features = normalize(nmf_features)

In [6]: current_article = norm_features[23,:]  # if has index 23

In [7]: similarities = norm_features.dot(current_article)

In [8]: print(similarities)
[ 0.7150569   0.26349967  0.40210445 ...,  0.70462768  0.20323616
  0.05047817]
```

DataFrames and labels

- Label similarities with the article titles, using a DataFrame
- Titles given as a list: **titles**

```
In [9]: import pandas as pd

In [10]: norm_features = normalize(nmf_features)

In [11]: df = pd.DataFrame(norm_features, index=titles)

In [12]: current_article = df.loc['Dog bites man']

In [13]: similarities = df.dot(current_article)
```

DataFrames and labels

```
In [14]: print(similarities.nlargest())
Dog bites man                            1.000000
Hound mauls cat                          0.979946
Pets go wild!                            0.979708
Dachshunds are dangerous                 0.949641
Our streets are no longer safe           0.900474
dtype: float64
```

UNSUPERVISED LEARNING IN PYTHON

Let’s practice!