

# Non-negative matrix factorization (NMF)

UNSUPERVISED LEARNING IN PYTHON

# 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 ( $\geq 0$ )

**Non-Negative Matrix Factorization (NMF)** is a dimensionality reduction and matrix decomposition technique that factors a non-negative matrix  $V$  into two lower-dimensional non-negative matrices,  $W$  and  $H$ .

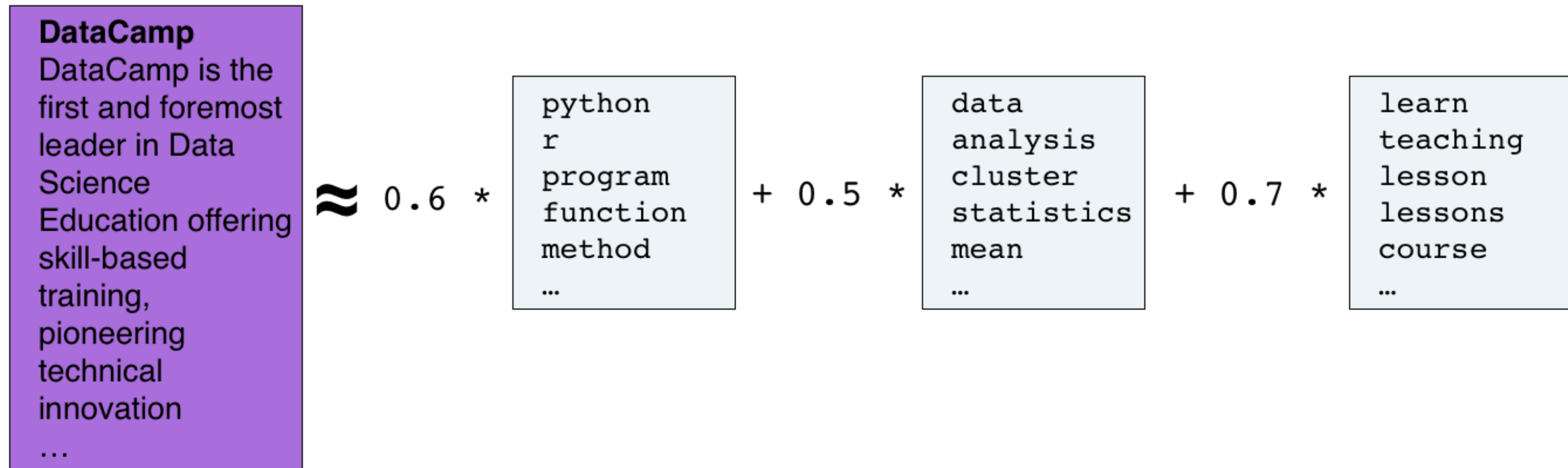
- It is widely used in data analysis and machine learning tasks where the data is inherently non-negative, such as text mining, image processing, and bioinformatics.

NMF is a powerful and interpretable dimensionality reduction technique that is well-suited for applications involving non-negative data.

- It excels in uncovering latent structures and patterns, making it a popular choice for tasks like topic modeling, image analysis, and recommendation systems.
- Its simplicity and ability to handle sparse data further enhance its utility in real-world scenarios.

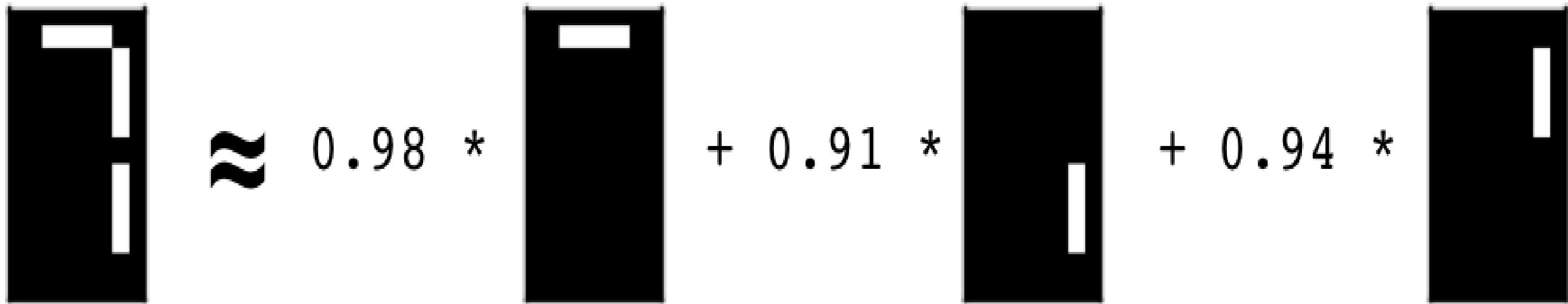
# Interpretable parts

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



# Interpretable parts

- NMF expresses images as combinations of patterns


$$\text{Image} \approx 0.98 * \text{Pattern}_1 + 0.91 * \text{Pattern}_2 + 0.94 * \text{Pattern}_3$$

# 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`

**csc\_matrix** (Compressed Sparse Row matrix) is a data structure used in numerical and scientific computing to store sparse matrices efficiently.

# 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
...			...	

Aspect	TF-IDF	Word Embeddings (e.g., Word2Vec)
Focus	Term frequency and document importance	Semantic meaning of words
Context	Ignores context	Captures context and relationships
Representation	Sparse vectors	Dense vectors
Use Cases	Text classification, clustering	NLP tasks requiring semantics (e.g., Q&A)

# Example usage of NMF

- `samples` is the word-frequency array

```
from sklearn.decomposition import NMF
nmf = NMF(n_components=2)
nmf.fit(samples)
```

**n\_components=2**: Specifies that the matrix will be reduced to **2 components** (or topics in text processing).  
**model.fit(samples)**: Decomposes the samples matrix into two matrices  $W$  (basis) and  $H$  (components).

```
nmf = NMF(n_components=2)
```

```
nmf_features = nmf.transform(samples)
```

**nmf\_features**: A matrix where each row corresponds to a sample, and each column represents its weight for one of the components.

**model.transform(samples)**: Computes the **feature matrix**  $W$ , which represents the input data in terms of the new components.

# NMF components

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

```
print(model.components_)
```

**model.components\_**: Retrieves the matrix HHH, where each row represents one of the **2 components**, and each column corresponds to the importance of a feature (e.g., a word) for that component.

```
[[ 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

```
print(nmf_features)
```

```
[[ 0.    0.2 ]  
 [ 0.19  0.  ]  
 ...  
 [ 0.15  0.12]]
```

# Reconstruction of a sample

```
print(samples[i,:])
```

Outputs the original word-frequency data for the ith sample for comparison.

```
[ 0.12    0.18    0.32    0.14]
```

```
print(nmf_features[i,:])
```

Displays the **weights** of the ith sample for the 2 components. Each value in this array indicates how much the sample contributes to each component.

```
[ 0.15    0.12]
```

The diagram illustrates the reconstruction of a sample using Non-negative Matrix Factorization (NMF). It shows the original sample data being decomposed into two components, which are then multiplied by the weights of the sample to reconstruct the original data.

The original sample data is shown as a 1x4 array: `[ 0.12 0.18 0.32 0.14]`.

The NMF components are shown as a 2x4 array, labeled `model.components_`:

$$\begin{bmatrix} 0.01 & 0. & 2.13 & 0.54 \\ 0.99 & 1.47 & 0. & 0.5 \end{bmatrix}$$

The weights of the sample for the two components are shown as a 1x2 array: `[ 0.15 0.12]`.

The reconstruction of the sample is shown as a 1x4 array, labeled "reconstruction of sample":

$$[ 0.1203 \quad 0.1764 \quad 0.3195 \quad 0.141 ]$$

The reconstruction is calculated as the product of the weights and the components:

$$\begin{bmatrix} 0.15 & 0.12 \end{bmatrix} \times \begin{bmatrix} 0.01 & 0. & 2.13 & 0.54 \\ 0.99 & 1.47 & 0. & 0.5 \end{bmatrix} = [ 0.1203 \quad 0.1764 \quad 0.3195 \quad 0.141 ]$$

# Sample reconstruction

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

Matrix Factorization in **Non-Negative Matrix Factorization (NMF)** refers to the process of decomposing a given non-negative matrix  $V$  into two lower-dimensional non-negative matrices  $W$  (basis matrix) and  $H$  (coefficient matrix) such that:

$$V \approx W \cdot H$$

This decomposition is achieved while ensuring that all elements in  $W$  and  $H$  are non-negative, preserving the non-negative nature of  $V$ .

## Key Components in Matrix Factorization

### 1. Input Matrix ( $V$ ):

1. A non-negative matrix of size  $m \times n$
2. Rows often represent features (e.g., words in text data).
3. Columns often represent samples (e.g., documents or items).

### 2. Basis Matrix ( $W$ ):

1. A non-negative matrix of size  $m \times k$ , where  $k$  is the number of components or latent factors.
2. Each column in  $W$  represents a latent feature, such as a topic in text or a feature in image data.

### 3. Coefficient Matrix ( $H$ ):

1. A non-negative matrix of size  $k \times n$ , where each column represents how much the corresponding latent feature contributes to the sample.

# 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!

# **Non-negative matrix factorization (NMF) learns interpretable parts**

UNSUPERVISED LEARNING IN PYTHON

# Example: NMF learns interpretable parts

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



# Applying NMF to the articles

```
print(articles.shape)
```

```
(20000, 800)
```

```
from sklearn.decomposition import NMF
nmf = NMF(n_components=10)
nmf.fit(articles)
```

```
NMF(n_components=10)
```

```
print(nmf.components_.shape)
```

```
(10, 800)
```

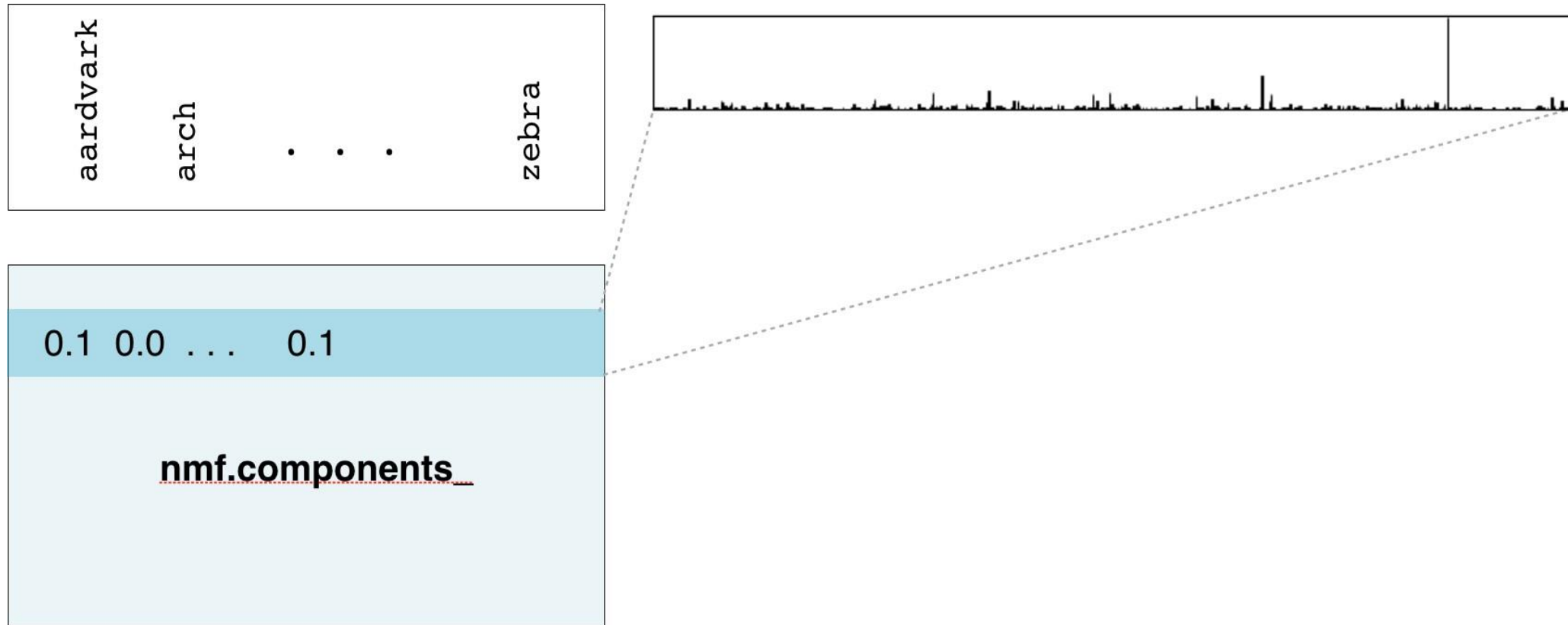
# NMF components are topics

aardvark	arch	.	.	.	zebra
----------	------	---	---	---	-------

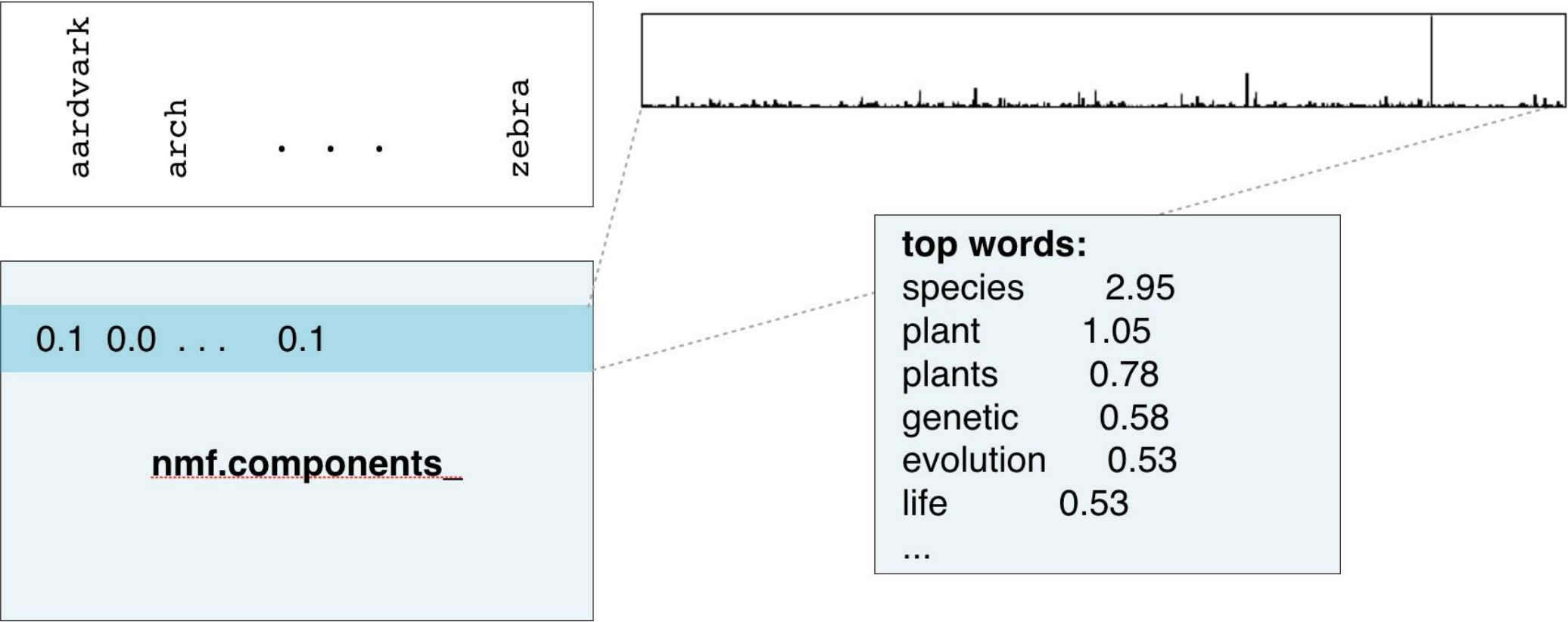
**nmf.components**



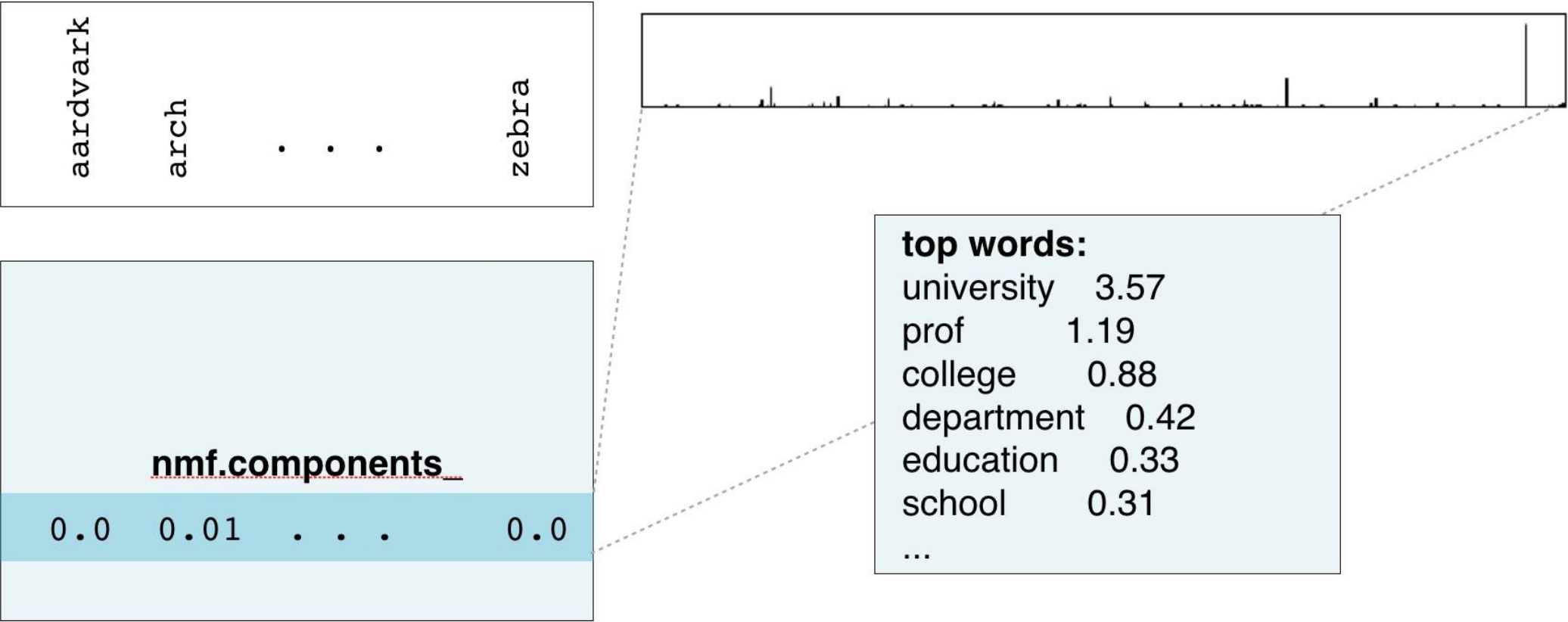
# NMF components are topics



# NMF components are topics

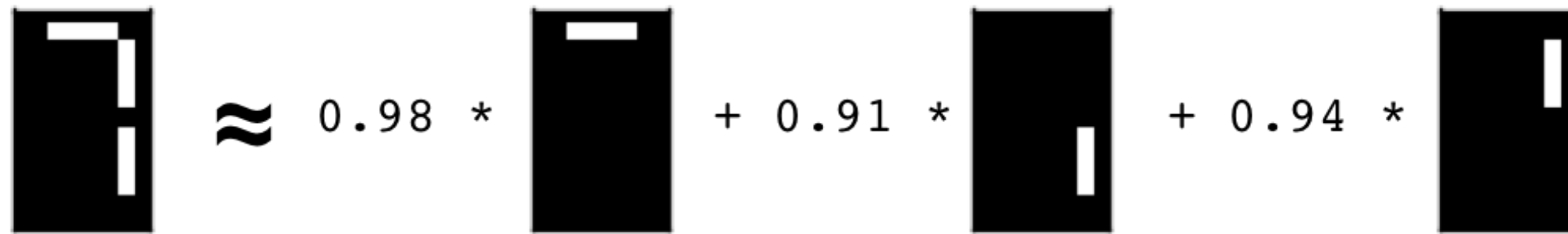


# NMF components are topics



# NMF components

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

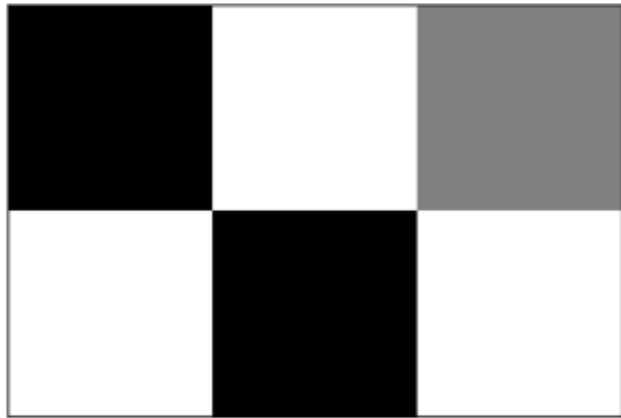


The diagram illustrates the NMF decomposition of an image. On the left is a target image of a black rectangle with a white 'H' shape. This is followed by an approximation symbol ( $\approx$ ). Then, three terms are shown, each consisting of a weight, a multiplication symbol, and a component image. The first term is  $0.98 *$  followed by a component image showing the top horizontal bar of the 'H'. The second term is  $+ 0.91 *$  followed by a component image showing the vertical stem of the 'H'. The third term is  $+ 0.94 *$  followed by a component image showing the right vertical bar of the 'H'.

$$\text{Image} \approx 0.98 * \text{Component 1} + 0.91 * \text{Component 2} + 0.94 * \text{Component 3}$$

# 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

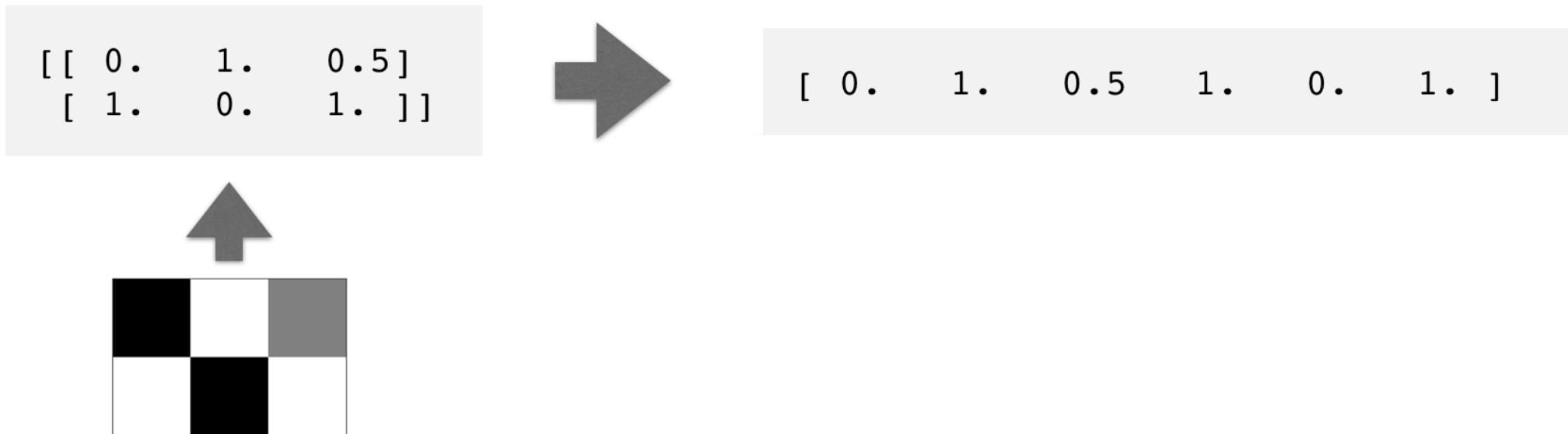
- An 8×8 grayscale image of the moon, written as an array



```
[ [ 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. ] ]
```

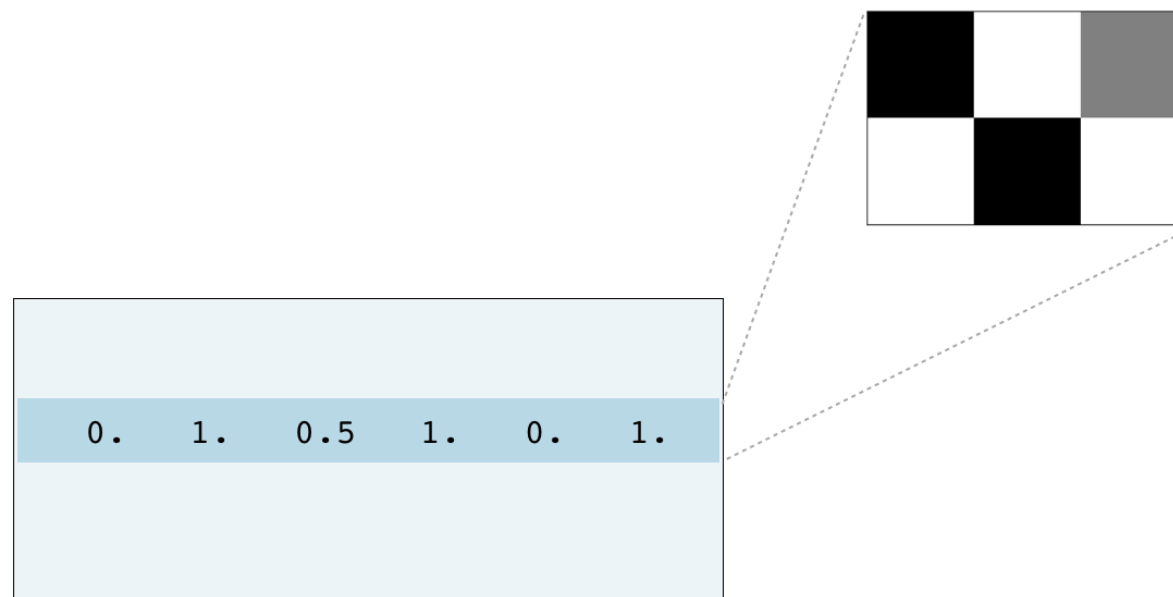
# Grayscale images as flat arrays

- Enumerate the entries
- Row-by-row
- From left to right, top to bottom



# 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!





# Visualizing samples

```
print(sample)
```

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

```
bitmap = sample.reshape((2, 3)) print(bitmap)
```

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

```
from matplotlib import pyplot as plt
plt.imshow(bitmap, cmap='gray', interpolation='nearest') plt.show()
```

# **Building recommender systems using Non- negative matrix factorization (NMF)**

**UNSUPERVISED LEARNING IN PYTHON**

# 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

```
from sklearn.decomposition      import  
                                NMF  
  
F nmf = NMF(n_components=6)  
nmf_features = nmf.fit_transform(articles)
```

# Versions of articles

- Different versions of the same document have same topic proportions
- ... exact feature values may be different!
- 
- 

## strong version

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

# 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
- 

## **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 ...

# Versions of articles

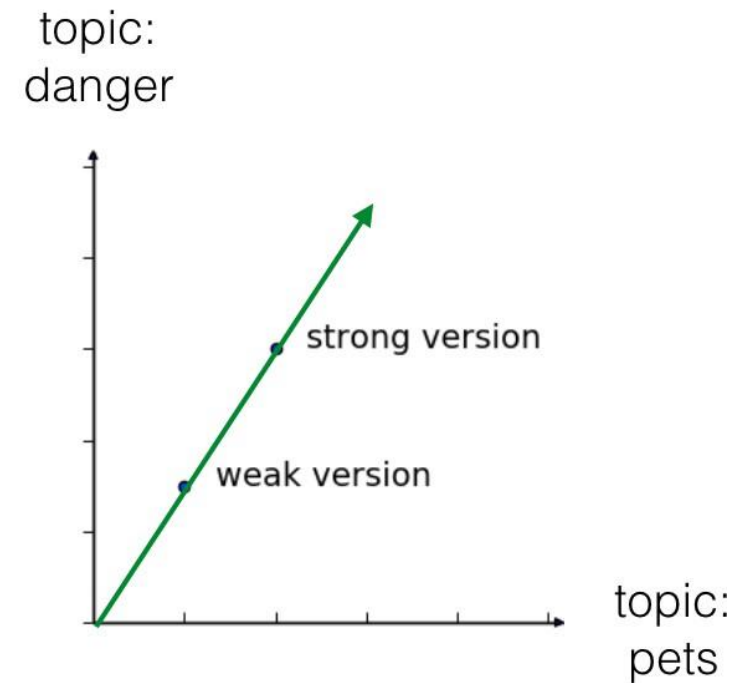
- Different versions of the same document have same topic proportions
- ... exact feature values may be different!
- E.g. because one version uses many meaningless words
- But all versions lie on the same line through the origin

## strong version

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

## weak version

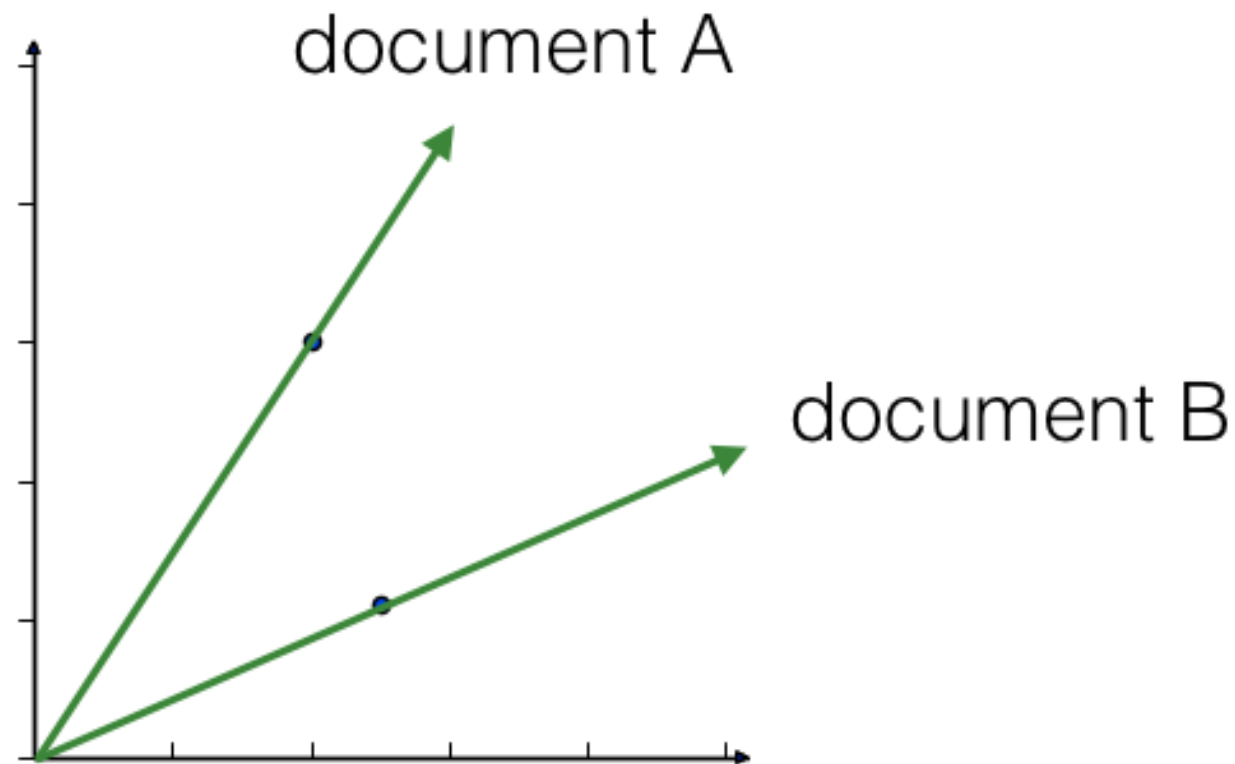
You may have heard,  
unfortunately it seems  
that a dog has perhaps  
bitten a man ...





# Cosine similarity

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



**Cosine similarity** is a metric used to measure the similarity between two non-zero vectors based on the cosine of the angle between them. It is widely used in applications like **text analysis**, **information retrieval**, and **machine learning** to determine how similar two pieces of data are in terms of their direction in a high-dimensional space.

Cosine similarity is a versatile and widely-used metric to compare vectors based on their orientation in space. Its emphasis on direction rather than magnitude makes it particularly useful for text similarity, clustering, and high-dimensional data analysis.

# Calculating the cosine similarities

```
from sklearn.preprocessing import normalize
norm_features = normalize(nmf_features)

# if has index 23
current_article = norm_features[23,:] similarities =
norm_features.dot(current_article) print(similarities)
```

```
[ 0.7150569          0.26349967 ..., 0.20323616          0.05047817]
```

## `norm_features.dot(current_article):`

- Computes the cosine similarity between the `current_article` vector and all other vectors in `norm_features`.
- Since the rows of `norm_features` are normalized, the **dot product** directly gives the cosine similarity.

## •Output:

- `similarities` is a 1D array, where each entry corresponds to the similarity score between the `current_article` and another article.
- Values range between -1-1-1 and 111:
  - **1**: Perfectly similar (identical direction).
  - **0**: Orthogonal (no similarity).
  - **-1**: Completely opposite direction (rare in normalized positive data like NMF).

**normalize(nmf\_features):** The `normalize` function from `sklearn.preprocessing` scales the rows of the input matrix (`nmf_features`) so that each row has a unit norm (length of 1).

## Purpose:

- Ensures that the magnitudes of the feature vectors do not affect the cosine similarity calculation.
- When vectors are normalized, cosine similarity becomes equivalent to the dot product of the vectors.

After normalization:

- Each row of `norm_features` is a unit vector.

**current\_article = norm\_features[23, :]**

This extracts the **23rd row** of the normalized feature matrix `norm_features`, representing the features of the target article.

**Shape:** `current_article` is a 1D vector corresponding to the 23rd article in the dataset.

# DataFrames and labels

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

```
import pandas as pd

norm_features = normalize(nmf_features)
df = pd.DataFrame(norm_features, index=titles)
current_article = df.loc['Dog bites man']
similarities = df.dot(current_article)
```

**df.dot(current\_article):** Computes the dot product between the feature vectors of all articles (rows of df) and the feature vector of the current article.

Since all rows are normalized, the dot product directly computes the **cosine similarity**.

**similarities:**

- ✓ A pandas Series where:
  - **Index:** Article titles.
  - **Values:** Cosine similarity scores between the corresponding article and "Dog bites man".

**normalize(nmf\_features):**

- Scales each row (representing an article) of the NMF feature matrix nmf\_features to unit norm (length 1).
- Normalization ensures that the magnitude of the vectors does not influence the cosine similarity calculation.
- After normalization:
  - ✓ The rows of norm\_features are unit vectors.
  - ✓ Cosine similarity becomes equivalent to the dot product of the vectors.

**pd.DataFrame():**

- Converts the normalized feature matrix into a pandas DataFrame for easier handling and visualization.
- **index=titles:**
  - ✓ Assigns the titles of the articles as row indices.
  - ✓ Each row in df corresponds to an article, and each column corresponds to a normalized feature.

**df.loc['Dog bites man']:** Retrieves the row corresponding to the article titled "Dog bites man".

**current\_article:**

- A 1D pandas Series containing the normalized feature vector for the article.

# DataFrames and labels

```
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
```