03 - TF-IDF and Dimensionality Reduction

Stephan Heule, stephan.heule@fhnw.ch

2025-02-28

University of Applied Sciences and Arts Northwestern Switzerland

Lecture Motivation

- still representing words/tokens with numbers
- vector similarity
- co-occurrence, TF-IDF
- PCA, SVD
- Application: Latent Semantic Analysis

Sparse representation

Words into numbers

- The methods so far included (except for naive Bayes):
  - using dot-product $\theta^T \mathbf{x}$ where θ are some learnable parameters from data. $\mathbf{x}$ is a feature vector (i.e. tokens in our case)
- recall that in general we include a *bias*, i.e. $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ with $x_0 = 1$. Often this is done automatically when you call a library/framework.
- Questions: how can we turn our features into numbers (i.e. vectors)
  - Bag of Words is one possibility[1]

1. See later lectures for more powerful deep-learning-approaches that conserve semantic and context of a word or a sentence

Term document matrix

Given $d = \text{ number of documents}$ and $|V| = \text{ vocabulary size over all documents}$

we build the **Term Document Matrix** TD as follows:

$$\begin{array}{ccccc}
 & \text{doc}_1 & \text{doc}_2 & \dots & \text{doc}_d \\
\text{word}_1 & C_{11} & C_{12} & \dots & C_{1d} \\
\text{word}_2 & C_{21} & C_{22} & \dots & C_{2d} \\
\text{word}_3 & C_{31} & C_{32} & \dots & C_{3d} \\
\dots & \dots & \dots & \dots & \dots \\
\text{word}_{|V|} & C_{|V|1} & C_{|V|2} & \dots & C_{|V|d}
\end{array}$$

where C_{ij} is simply the count of occurrences of word i in document j.

- this is an “old” concept and dates back to the 1960
- a document could be a book, an email, a sentence, a manual, a wikipedia article, …

Word vector from TD

The $\mathrm{TD\text{-}Matrix}$ allows to express a word (or a document) as a vector:

$$\mathrm{word}_i = (C_{i1}, \ldots, C_{id}) \in \mathbb{N}^d$$

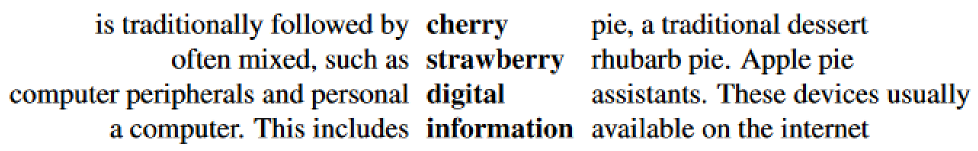
or

$$\mathrm{doc}_k = (C_{1k}, \ldots, C_{|V|k}) \in \mathbb{N}^{|V|}$$

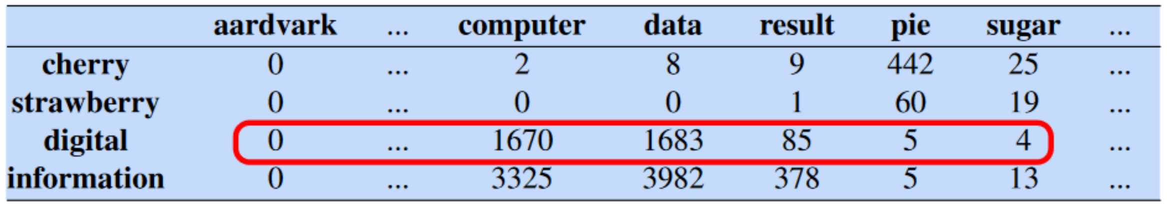
Remark:

- usually $|V| \gg d$
- often very sparse (i.e. many zeros)

Word-word matrix example



is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet



	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Chapter 6 in Speech and Language Processing

Here:

- 4 tokens to the left and right of *digital* we often see words like *computer, data, result* and not so often *pie, sugar*
- this 4 is called *context size*
- still very sparse representation

Word-word matrix

- the matrix representation is of dimension $|V| \times |V|$
- we select a context-size[1] c, e.g. $c = 4$ in the example before
- if you want to ignore beginning and end of sentences, paragraphs, etc. is your modelling choice

1. we will see this strategy later on in word2vec once again

Vector similarity

How would you measure if two vectors v_1 and v_2 are close or very different?

- any metric will do (e.g. L_2) → distance between v_1 and v_2
- cosine → angle between v_1 and v_2

cosine-similarity[1]:

$$\cos\angle(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2}$$

where $\angle(\mathbf{x},\mathbf{y})$ is the angle between $\mathbf{x}$ and $\mathbf{y}$ and $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{N} x_i^2}$

1. *cosine-similarity* turns out to be very important in various tasks (e.g. document retrieval, question answering, etc.)

Vector similarity

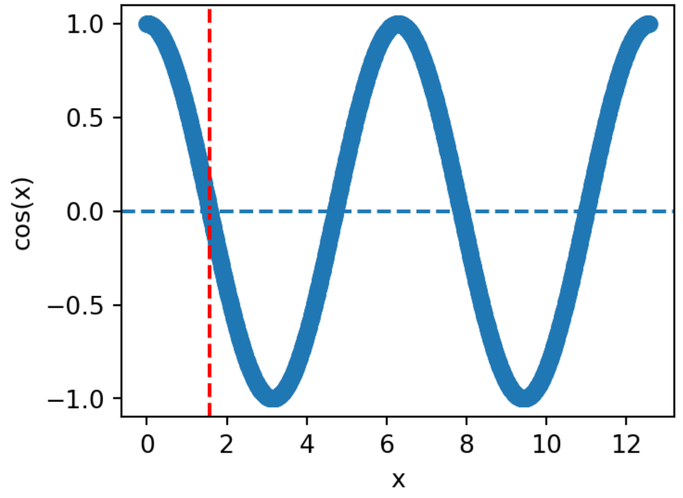
Remarks:

- two orthogonal vectors have cosine similarity of 0 and two vectors pointing in the same direction have a cosine similarity of 1
- **cosine-distance**[1] is often used: $\text{cosine distance}(\mathbf{x}, \mathbf{y}) := 1 - \cos \angle(\mathbf{x}, \mathbf{y})$
- important: this is not the same as a distance. A real distance follows this property $\text{dist}(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$

1. this is not a real distance!

Cosine plot

▶ Code

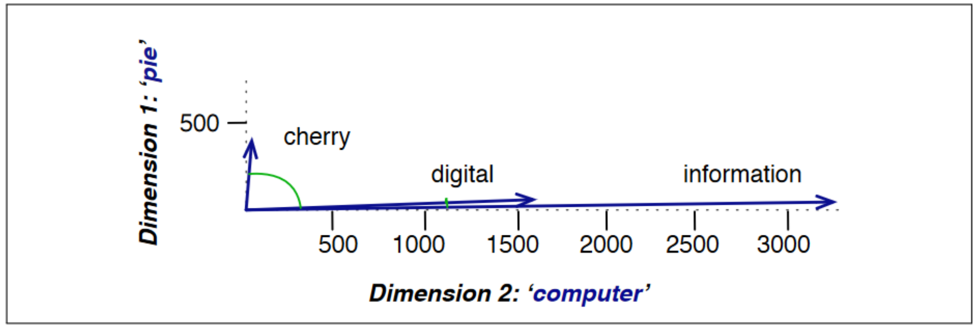


Remarks:

- cos is 2π-periodic
- for the representations we discussed so far: why is the $\angle(\mathbf{x}, \mathbf{y}) \in [0, \pi/2]$? What does this mean?

Cosine similarity example

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325



Very high similarity between *digital* and *information*. Almost orthogonal is *cherry*. Images taken from Chapter 6 in Speech and Language Processing

Co-occurrence challenges

- co-occurrence methods are based on counts
- problematic:
  - there are many words that are shared in all the documents
  - there are a few words that are very specific to a document
- we want to balance between rare (but important) words and words that are generic and unspecific
- → TF-IDF helps

TF-IDF

- **TF-IDF** is short for *term frequency inverse document frequency*
- important weighting scheme for term document matrices
- given a count of words (either TF or word-word) we want to:
  1. reduce the importance of generic words that are not specific
  2. give more weight to word counts that are rare but very specific in a document

TF-IDF idea

Recall:

$$\begin{array}{lcccc} & \text{doc}_1 & \text{doc}_2 & \dots & \text{doc}_d \\ \text{word}_1 & C_{11} & C_{12} & \dots & C_{1d} \\ \text{word}_2 & C_{21} & C_{22} & \dots & C_{2d} \\ \text{word}_3 & C_{31} & C_{32} & \dots & C_{3d} \\ \dots & \dots & \dots & \dots & \dots \\ \text{word}_{|V|} & C_{|V|1} & C_{|V|2} & \dots & C_{|V|d} \end{array}$$

- generic words have big counts in many documents: applying a $\log$-transformation reduces its weights
- document specific words only appear in certain documents but not in all
- we combine the two in a product to “down-weight” generic and “up-weight” specific words

TF-IDF method

1. transform the counts C_{ij}:

$$\tilde{C_{ij}} = \log(1 + C_{ij})$$

2. normalize by **inverse document frequency**

$$\begin{aligned} \mathrm{df}_{\mathrm{word}_i} &= \#\text{ of documents including word}_i \\ \mathrm{idf}_{\mathrm{word}_i} &= \log \frac{\#\text{ number of documents}}{\mathrm{df}_{\mathrm{word}_i}} \end{aligned}$$

3. finally the TF-IDF:

$$\text{tf-idf} = \tilde{C_{ij}} \times \mathrm{idf}_{\mathrm{word}_i}$$

TF-IDF matrix

Plugging this into the matrix:

$$\begin{array}{cccccc} & \text{doc}_1 & \text{doc}_2 & \dots & \text{doc}_d \\ \text{word}_1 & \text{tf\_idf}_{11} & \text{tf\_idf}_{12} & \dots & \text{tf\_idf}_{1d} \\ \text{word}_2 & \text{tf\_idf}_{21} & \text{tf\_idf}_{22} & \dots & \text{tf\_idf}_{2d} \\ \text{word}_3 & \text{tf\_idf}_{31} & \text{tf\_idf}_{32} & \dots & \text{tf\_idf}_{3d} \\ \dots & \dots & \dots & \dots & \dots \\ \text{word}_{|V|} & \text{tf\_idf}_{|V|1} & \text{tf\_idf}_{|V|2} & \dots & \text{tf\_idf}_{|V|d} \end{array}$$

where

$$\text{tf-idf}_{ij} = \tilde{C}_{ij} \times \text{idf}_{\text{word}_i}$$

Remarks TF-IDF

- the chosen logarithm does not matter[1]
- it is a heuristic approach, i.e. instead of $\log(1+x)$ you can also use $\log x \text{ if } x > 0 \text{ else } 0$. There are many variants of TF-IDF weighting strategies [2]
- one very prominent weighting strategy is called Okapi BM25
- the $\log$ in the $\text{idf}_{\text{word}_i}$ is also chosen to accommodate for large number of documents
- the specific transformations depend on your problem – there is not one true approach

1. the basis does not matter (up to a constant) since: $\log_k x = \log_k \left(b^{\log_b x}\right) = \log_b x \cdot \log_k b$
2. TF-IDF on Wikipedia

Document Scoring

How to find top documents to a query q in a bunch of documents $d_1, d_2, \ldots$?

$$\begin{aligned} \text{query} &= \text{Where is the zoo?} \\ \text{doc}_1 &= \text{The zoo is close to the train station ...} \\ \text{doc}_2 &= \text{Doctor Dolittle was ...} \\ \text{doc}_3 &= \text{Many animals are ...} \\ \text{doc}_4 &= \text{Eating soup ...} \end{aligned}$$

Idea: more relevant documents are closer to the query!

Retrieval

- What is close?

$$\text{score}(q, d) = \cos \angle(q, d) = \frac{q}{\|q\|_2} \cdot \frac{d}{\|d\|_2}$$

- How to choose a vector representation of q and d? → TF-IDF!
  - represent query q also in the TF-IDF space
  - q then is shaped like a column in the TF-IDF matrix
  - calculate the dot product

Practical considerations

- the idf term in the q and d part is always the same - calculate it only once:

$$\text{tf-idf}(\text{word}, \text{doc}) = \text{tf-idf}(i, j) = \tilde{C_{ij}} \times \mathrm{idf}_{\text{word}_i}$$

- query q is most often shorter than any of documents d_i
- thus the count of most words $w \in q$ will be 0 since $\tilde{C_{ij}} = \log(1 + C_{ij})$ and $C_{ij} = 0$ for most words
- turns out TF-IDF approaches can be a very strong baseline

Dense representation

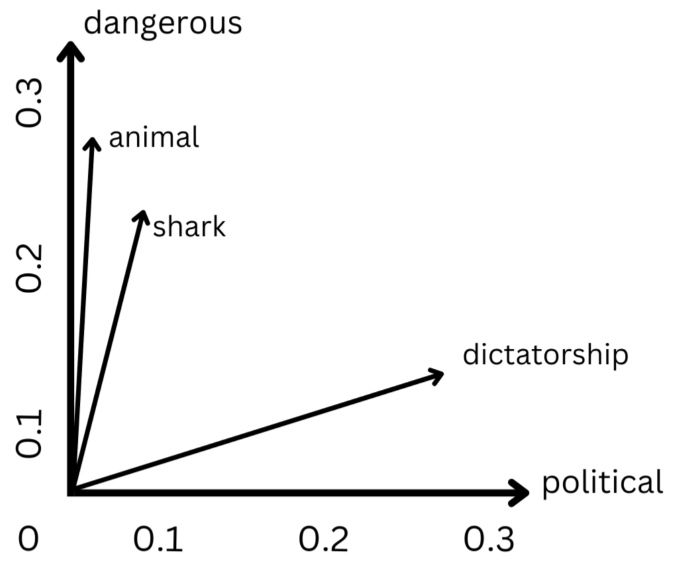
Motivation

- our vector representation of words so far resulted in very sparse matrices or vectors
- this is problematic:
  - we need too much memory to do efficient calculations
  - sparse dimensions hold (probably) very little useful information
- What should be closer to the vector representation of $\overrightarrow{\text{apple}}$: $\overrightarrow{\text{tree}}$, $\overrightarrow{\text{banana}}$ or $\overrightarrow{\text{car}}$?
  - $\overrightarrow{\text{tree}}$ and $\overrightarrow{\text{banana}}$ are related by the concept of plant, fruit, …
  - but depending on the documents we choose the closest representation could be $\overrightarrow{\text{car}}$
  - context is lost [1]

1. we learn about how to encode context shortly

What we actually want

Distributional hypothesis: “words that are used and occure in the same context tend to purport similar meanings”



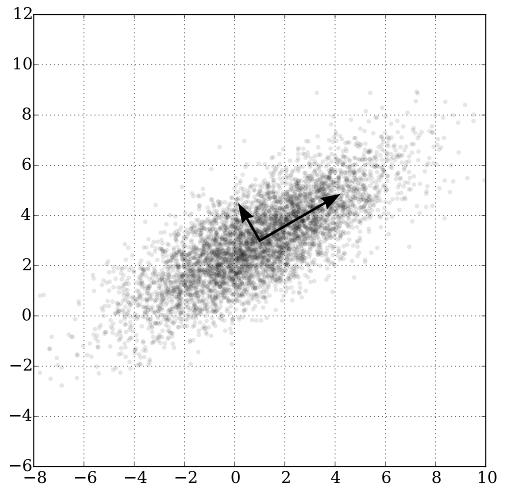
Wiki Distributional Semantics

PCA - principal component analysis

- is a data reduction algorithm
- given N observations $\mathbf{x} \in \mathbb{R}^D$ we want to find a “good” representation in a lower dimensional vector space $\mathbb{R}^M$ with $M < D$
- here *good* means: a lower dimensional space that maximizes the projected variance
- the lower dimensional basis (→ linear algebra) are called *principle components*
- it turns out that [1]:
  - this results in an orthogonal basis
  - the basis vectors are the eigenvectors of the covariance matrix of $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$
- easiest seen visually

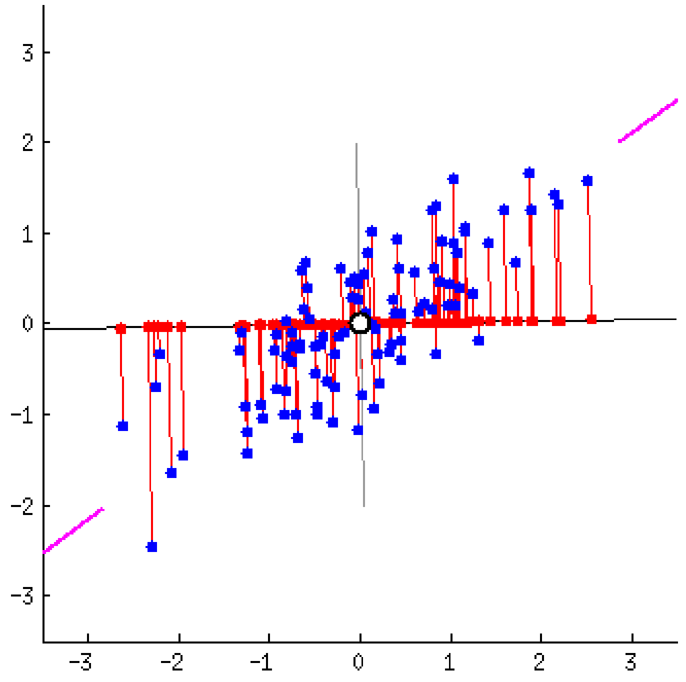
1. there is also a minimal error formulation: Pattern Recognition

PCA-plot



PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors (i.e. the 1st and 2nd principal components) of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean. Image source Wikipedia

PCA animation



Read this excellent answer

Animation source

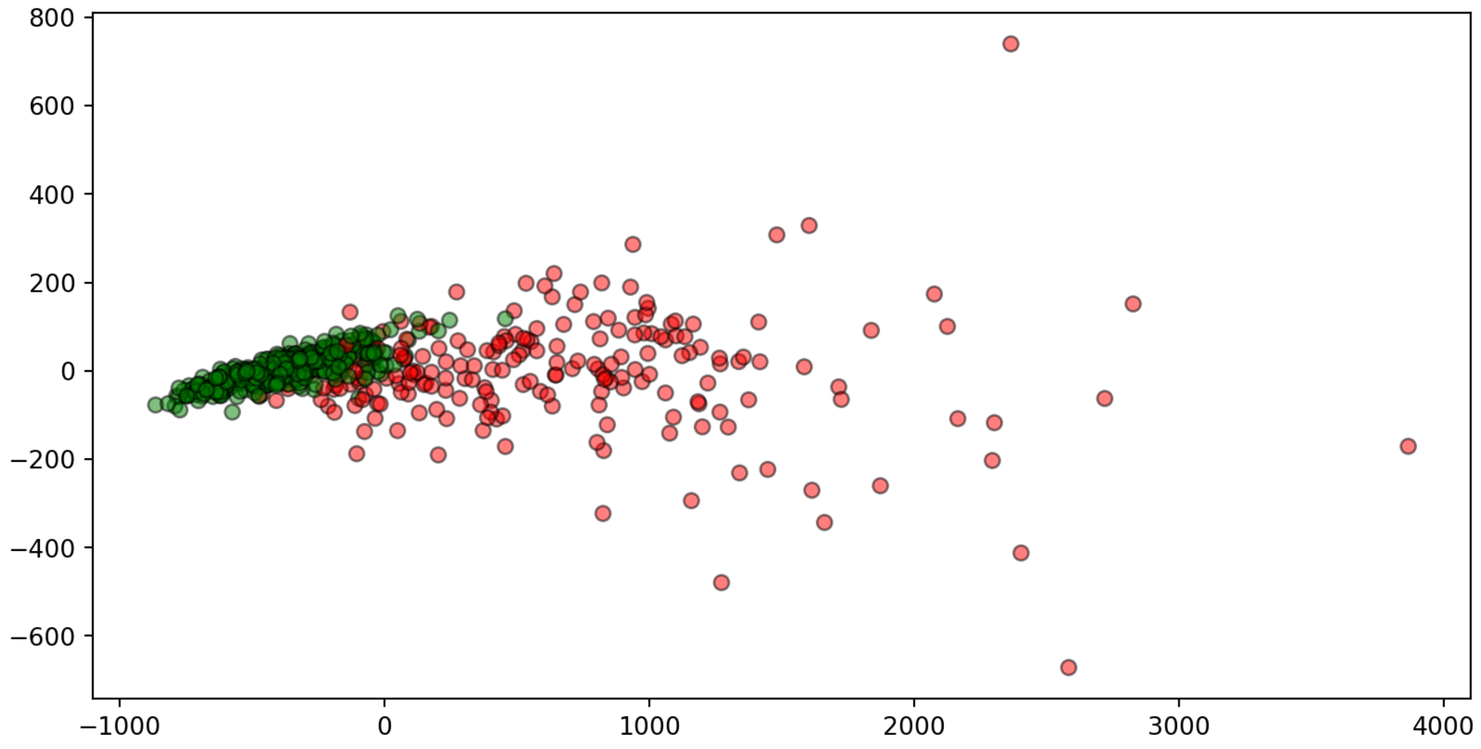
Example: PCA

Dimension reduction: we have 30 features of cancer observations of 569 patients and reduce it onto 2 dimensions. The colors are *malignant*, *benign*.

```python
from sklearn import datasets, decomposition
import matplotlib.pyplot as plt

data = datasets.load_breast_cancer()
X, y = data.data, data.target

print(f"{X.shape = }")
print(X[:2, :])
```

```
X.shape = (569, 30)
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]]
```

```python
pca = decomposition.PCA(n_components = 2)
X_pca = pca.fit_transform(X)

color_map = {0: 'red', 1: 'green'}
color = [color_map[label] for label in y]

plt.cla()
plt.scatter(x=X_pca[:,0], y=X_pca[:, 1], cmap=plt.cm.nipy_spectral, edgecolor="k", c=color, alpha=0.5)
plt.show()
```



If you reduce the data on the two first principal components (x- and$y$-axis), then the *malignant* and *benign* observations are already quit separated!

Singular Value Decomposition SVD

- under the hood the *PCA* relies on the **singular value decomposition** of a data matrix $\mathbf{A}$:
- given a matrix $\mathbf{A}$ we can always decompose it into:

$$\mathbf{A} = \mathbf{U}\,\Sigma\,\mathbf{V}^T$$

  where the columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and Σ is a diagonal matrix with positive real entries (the **singular values** σ_i)
- $a_i, a_j \in \mathbb{R}^n$ are *orthonormal* $\Leftrightarrow \langle a_i, a_j\rangle = \delta_{ij}$ (δ_{ij} is the Kronecker delta)
- we order the singular values with decreasing order $\sigma_1 \geq \sigma_2$ and chose the decomposition s.t. all $\sigma_i \geq 0$
- $\mathbf{SVD}$ is omnipresent in machine learning with many applications (pseudo inverse, linear regression, low-rank approximation, recommender engines, etc.)

SVD properties

SVD dimensions for $\mathbf{A} \in \mathbb{R}^{m \times n}$:

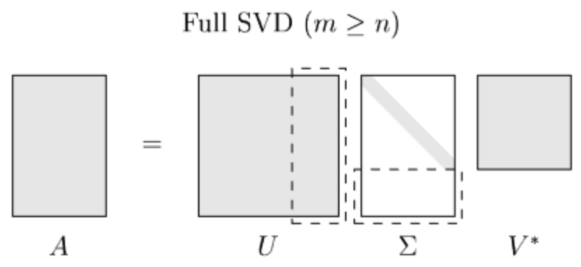
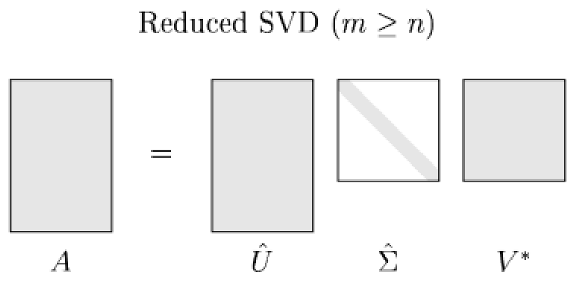


Image source: Trefethen-Bau



Often we have $\sigma_i = 0 \; \forall \, i \in \{k+1, \ldots, n\}$ then we use the reduced singular value decomposition

$\mathbf{SVD}$ properties continued

- the singular values σ_i of the $\mathbf{SVD}$ and the eigenvalue of the PCA are tightly connected[1]
- there is always a $\mathbf{SVD}$ to any matrix $\mathbf{A}$
- if $\mathbf{A}$ is square then the σ_i are distinct

> ! Make sure you understand how to apply the (reduced) SVD

1. PCA-SVD connection

SVD for matrix approximation

Suppose you decided to use the first k components. Then we can do a *low rank approximation*:

$$\mathbf{X} = U\,\Sigma\,V^T \approx U_k\,\Sigma_k\,V_k^T$$

where

- U_k are the first k columns of U
- Σ_k is the square diagonal matrix with the largest k $\text{singular values } \sigma_k$ and
- V_k^T are the first k rows of V^T

Note

What is the dimension of $\mathbf{X}$ and what are the dimensions of U_k, Σ_k and V_k^T?

What is a good k?

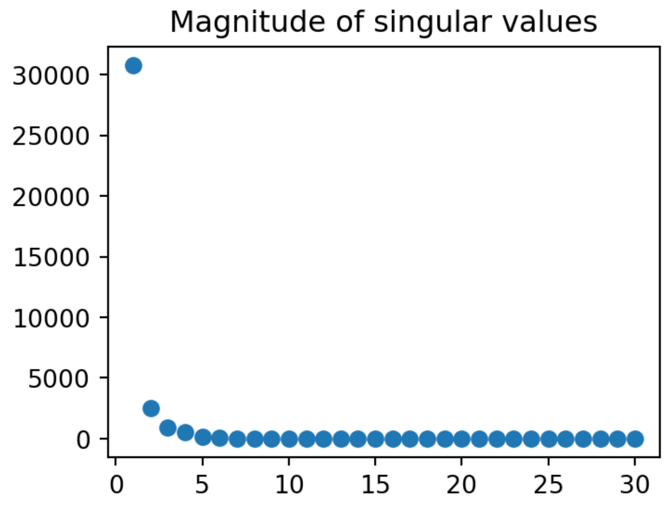
How many singular values $\sigma_1, \ldots, \sigma_k$ do we keep?

- there are restrictions wrt. memory, speed, etc.
- treat it as a hyperparameter and do crossvalidation
- use elbow heuristic[1] (if there is an elbow)
- …

1. Elbow Method

Elbow example

```python
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt

data = datasets.load_breast_cancer()
U, S, Vh = np.linalg.svd(data.data, full_matrices=True)

fig = plt.figure()
fig.set_size_inches(4, 3)
plt.scatter(np.arange(len(S)) + 1, S)
plt.title("Magnitude of singular values")
plt.show()
```



Remarks:

- in practice you might prefer to do a reduced $\mathbf{SVD}$
- here you might argue to use at least the first two components (but it is not wrong to use the first three or four ....)

Application: Latent Semantic Analysis

- Given a co-occurance matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ where m is the number of terms and n the number of documents
- for instance $\mathbf{X}$ could be the TF-IDF matrix
- then the angles between words of $\mathbf{X}$ and documents are given by[1]:

$$\begin{aligned} XX^T = & \quad U\Sigma\Sigma^T U^T \\ X^T X = & \quad V\Sigma^T\Sigma V^T \end{aligned}$$

1. recall that U and V are orthonormal basis, i.e. $U^T U = I$ (similar for V)

Latent Semantic Analysis

Thus with $X = U\,\Sigma\,V^T$ we can calculate similarity-scores as:

- document similarity of documents d_1, d_2:

$$\text{score}(d_i, d_j) = \Sigma_k V_{\bullet,i} \cdot \Sigma_k V_{\bullet,j}$$

- term similarity of terms w_i and w_j:

$$\text{score}(w_i, w_j) = \Sigma_k U_{i,\bullet} \cdot \Sigma_k U_{j,\bullet}$$

where:

$X_{i,\bullet}$ stands for the row vector i of matrix X, i.e. $[x_{i,1}, \ldots, x_{i,n}]$. Similarly for column vector j we write $X_{\bullet,j}$

Latent Semantic Analysis

Goal: given a new query vector $q \in \mathbb{R}^m$ we want to calculate the $\mathrm{score}_{\mathrm{approx}}(q, d_j)$ with each document embedding d_j. But the score is now approximated by the partial $\mathbf{SVD}$. Thus we have to apply the same transformation to our query q:

- Observe that

$$\begin{aligned} \mathbf{X} &\approx U_k\, \Sigma_k\, V_k^T \\ \iff\ & U_k^T \mathbf{X} \approx U_k^T U_k \Sigma_k V_k^T\ = \Sigma_k V_k^T \\ \iff\ & \Sigma_k^{-1} U_k^T \mathbf{X} \approx V_k^T \end{aligned}$$

- Thus to land in the column space of V^T we apply:

$$q \approx \underbrace{\Sigma_k^{-1}\, U_k^T\, q}_{=\hat{q}}$$

- Now you can calculate $\mathrm{score}_{\mathrm{approx}}(q, d_j) = \Sigma_k\, \hat{q}\, \cdot\, \Sigma_k V_{\bullet, j}$

Summary

- what kind of co-occurrence matrices do you know? How do you represent a document or a word?
- what is the idea behind the TD-IDF transformation?
- how can you measure the vector similarity?
- how can you do a document score, i.e. find the appropriate documents to a query?
- what is the distributional hypothesis and why are we aiming for that?
- what can you do with PCA and SVD? What is the reduced SVD?
- how do you choose an “appropriate” number of singular values or principle components?
- how can you approximate a matrix and why is this a useful operation?
- describe latent semantic analysis