LSA_toyexample

November 26, 2020

1 Zur Analyse von unstrukturierten Daten

(siehe: https://shankarmsy.github.io/posts/pca-sklearn.html)

Ziel Abschnitt 1: Sie knüpfen an Ihr Vorwissen zu PCA/PFA (Hauptkomponenten- und Faktorenanalyse) an und verbinden dies mit einer "neuen" Datenform: Bildern, die in ML-Anwendungen große Bedeutung hat. Auf dem Planeten Erde gibt es momentan etwa 1600 Exabytes an Daten (Stand 2015). Schätzungen zufolge sind etwa 95% aller Daten im Internet unstrukturiert (https://arxiv.org/pdf/1608.08176.pdf) (siehe auch: "Dataquake").

Beispiele für unstrukturierte Daten sind vielfältig. Zum Beispiel unbeschriebene Bilder ...

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
plt.rcParams["figure.figsize"]=15,15
img = mpimg.imread('../data/Einstein_pic.png')
print( img.shape )
plt.axis('off')
plt.imshow(img)
```

```
(1725, 2167, 3)
```

[1]:
```
<matplotlib.image.AxesImage at 0x186d8984400>
```

Um für die Analyse zugänglich zu sein, sollten Daten in Matrix-Form dargestellt werden:

[2]:
```
img
```

[2]:
```
array([[[0.8509804 , 0.8235294 , 0.8       ],
        [0.85490197, 0.827451  , 0.8039216 ],
        [0.85490197, 0.827451  , 0.8039216 ],
        …,
        [0.8901961 , 0.87058824, 0.85490197],
        [0.8901961 , 0.87058824, 0.85490197],
        [0.8901961 , 0.87058824, 0.85490197]],

       [[0.8509804 , 0.8235294 , 0.8       ],
        [0.85490197, 0.827451  , 0.8039216 ],
        [0.85490197, 0.827451  , 0.8039216 ],
        …,
        [0.8901961 , 0.87058824, 0.85490197],
        [0.8901961 , 0.87058824, 0.85490197],
        [0.8901961 , 0.87058824, 0.85490197]],
```

```
[[0.8509804 , 0.8235294 , 0.8       ],
 [0.85490197, 0.827451  , 0.8039216 ],
 [0.85490197, 0.827451  , 0.8039216 ],
 …,
 [0.8901961 , 0.87058824, 0.85490197],
 [0.8901961 , 0.87058824, 0.85490197],
 [0.8901961 , 0.87058824, 0.85490197]],

…,

[[0.17254902, 0.12941177, 0.11372549],
 [0.2       , 0.14509805, 0.13333334],
 [0.2       , 0.14509805, 0.13333334],
 …,
 [0.8352941 , 0.8117647 , 0.7647059 ],
 [0.8352941 , 0.8117647 , 0.7647059 ],
 [0.8352941 , 0.8117647 , 0.7647059 ]],

[[0.18039216, 0.1254902 , 0.11372549],
 [0.2       , 0.14509805, 0.13333334],
 [0.2       , 0.14509805, 0.13333334],
 …,
 [0.8352941 , 0.8117647 , 0.7647059 ],
 [0.8352941 , 0.8117647 , 0.7647059 ],
 [0.8352941 , 0.8117647 , 0.7647059 ]],

[[0.1882353 , 0.13333334, 0.12156863],
 [0.19607843, 0.14117648, 0.12941177],
 [0.2       , 0.14509805, 0.13333334],
 …,
 [0.8352941 , 0.8117647 , 0.7647059 ],
 [0.8352941 , 0.8117647 , 0.7647059 ],
 [0.8352941 , 0.8117647 , 0.7647059 ]]], dtype=float32)
```

[3]:
```
rows, pix, depth = img.shape
img.shape
```

[3]: (1725, 2167, 3)

[4]:
```
img.size
```

[4]: 11214225

[5]:
```
1725*2167*3
```

[5]: 11214225

Um das Bild in 2d-Matrixdekomposition verarbeiten zu können, restrukturieren wir die Matrix. Vergleichbar ist niederdimensionalen Fall mit:

$$[a, b, c, d] \rightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

[6]:
```
img_r = np.reshape(img, (rows, pix*depth))
print( 'Neue Dimensionen: {}'. format( img_r.shape ) )
```

```
Neue Dimensionen: (1725, 6501)
```

Wenn wir in der Bildungsforschung mit Matrizen rechnen, sind diese häufig "dense". D.h., mehr als die Hälfte der Elemente der Matrix sind nicht Null. Berechnungen auf solchen Matrizen sind oft stabiler, da Phänomene wie `zero division`seltener auftreten, die Algorithmen abstürzen lassen.

Sind Matrizen von Bildern "dense" oder "sparse"?

[7]:
```
from scipy.sparse import issparse
issparse(img_r)
```

[7]:
```
False
```

2 Recap: Principal Component Analysis

Die Hauptkomponentenanalyse dient dazu, Dimensionen maximaler Variabilität in der Korrelationsmatrix verschiedener Items zu identifizieren. Damit können viele Items (Dimensionen) auf eine kleinere Anzahl an Dimensionen reduziert werden.

Inwieweit ist dieses Konzept übertragbar auf andere Datentypen wie Bilder oder Sprachdaten? (from: https://shankarmsy.github.io/posts/pca-sklearn.html)

Probieren wir es aus und wenden wir eine Hauptkomponentenzerlegung (PCA) auf das Bild an:

[8]:
```
from sklearn.decomposition import PCA
ipca = PCA(n_components=20, svd_solver='randomized').fit(img_r) # Die␣
 ↪Hauptkomponenten sind im Objekt 'ipca' gespeichert
img_c = ipca.transform(img_r) # Nun wird die Bildmatrix 'img_r' transformiert

print( 'Neue Dimensionalität nach der PCA: {} versus {}'.format( img_c.shape,␣
 ↪img_r.shape ) )
print( 'Erklärte Varianz in Bezug auf "img_r": {:.0f} %'.format( np.sum(ipca.
 ↪explained_variance_ratio_)*100 ) )
```

```
Neue Dimensionalität nach der PCA: (1725, 20) versus (1725, 6501)
Erklärte Varianz in Bezug auf "img_r": 88 %
```

Ein großer Anteil an Varianz kann also erklärt werden, mit nur 20 Dimensionen! (Im Gegensatz zu 6501 Dimensionen)

Nun kann die neue Matrix 'img_c' zurücktransformiert werden, sodass wir das Bild mit weniger Dimensionen darstellen können:

[9]:
```
temp = ipca.inverse_transform(img_c)
print( temp.shape )
temp = np.reshape(temp, (rows,pix,depth))
print( temp.shape )
```

```
(1725, 6501)
(1725, 2167, 3)
```

[10]:
```
plt.axis('off')
plt.imshow(temp)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

[10]:
```
<matplotlib.image.AxesImage at 0x186e3486208>
```

[11]:
```
print( 'We only use {:.1f}% of the original image size!'.format( img_c.size/
↪(img.size + img_c.size)*100 ) )
```

```
We only use 0.3% of the original image size!
```

[12]:
```
import ipywidgets as widgets

@widgets.interact(dimensionality=(0, 256, 50))
def matrix_output(dimensionality=50):
    ipca = PCA(n_components=dimensionality, svd_solver='randomized').fit(img_r)
    img_c = ipca.transform(img_r)
    temp = ipca.inverse_transform(img_c)
    temp = np.reshape(temp, (rows,pix,depth))
    plt.axis('off')
    plt.imshow(temp)
    print( 'We only use {:.1f}% of the original image size!'.format( img_c.size/
 ↪(img.size + img_c.size)*100 ) )
```

```
interactive(children=(IntSlider(value=50, description='dimensionality', max=256, step=50), Outp
```

3 Inwieweit können wir dieses Konzept auf Texte übertragen?

Ziel des Abschnittes: Sie erkennen, dass eine einfache Übertragung der PCA auf Textdaten nicht möglich ist. Sie lernen die Grundlagen sowie die Implementation der Singulärwertzerlegung als Alternative für die PCA kennen. Kreieren wir einen mehr oder weniger sinnvollen Beispieltext, bestehend aus vier korrekten deutschsprachigen Sätzen:

[13]:
```
Beispieltext = 'Das Wetter ist heute schlecht. \
Bei diesem schlechten Wetter sollte man nicht rausgehen. \
Auch die Nachrichten zum Coronavirus helfen nicht zu guter Laune. \
Gute Laune kann nur unabhängig vom Coronavirus (SARS-CoV2) entstehen.'
```

Stellen Sie sich vor, diese vier Sätze wären die Originaldaten, die Sie auf einer Ihrer Aufgaben erhalten haben. Die erste Überlegung ist, wie können diese Daten weiter strukturiert werden. Dabei gilt es zu überlegen, was Dokumente in bedeuten.

Wie auch in anderen Studien gemacht, werden wir der Einfachheit halber Sätze als kleinste Kodiereinheiten/Dokumente betrachten. Die Zerlegung in Sätze erfolgt in diesem Beispiel einfach durch den Befehl `split('. ')`, der nach dem Auftauchen des Argumentes sucht, das Element entfernt und den Text entsprechend trennt:

[14]:
```
Texts_tokenized = Beispieltext.split('. ')
Texts_tokenized
```

[14]:
```
['Das Wetter ist heute schlecht',
 'Bei diesem schlechten Wetter sollte man nicht rausgehen',
 'Auch die Nachrichten zum Coronavirus helfen nicht zu guter Laune',
 'Gute Laune kann nur unabhängig vom Coronavirus (SARS-CoV2) entstehen.']
```

Neben der Segmentierung in Sätze als elementare Kodiereinheiten, werden die Dokumente jetzt weiter zerlegt. Jedes Dokument wird in einzelne Worte/Tokens zerlegt. Warum ist dies erforderlich?

Kurz: Um überhaupt sinnvoll mit begrenzten Datensätzen arbeiten zu können.

Eine solche Zerlegung wird auch als Bag-of-Words-Modell bezeichnet. Intuitiv ist es in dieser "Tasch" egal, wie die Reihenfolge der Worte ist. Sie können sich vorstellen, dass dies eine starke Annahme ist, die viele Abhängigkeiten in der Sprache zerstört. Stellen Sie sich vor, man würde die Sätze als Ganzes betrachten. Da kaum zwei Sätze jemals exakt identisch sind, können Sie damit keine Statistik machen, da alle Dokumente nur einmalig auftreten im Datensatz. In vielen Modellierungen von Sprachdaten geht es aber um das gemeinsame Auftreten von Elemente (z.B.: Worte, Phrasen, …). Deshalb wird vereinfachend mit Bag-of-Words-Modellen der Dokumente weitergearbeitet.

In `sklearn` stehen dabei sogenannte `vectorizer` zur Verfügung. In diesem Fall der `CountVectorizer`. Sie füttern diesen mit ganzen Sätzen (können auch andere Dokumente sein), die als `strings`vorliegen. Der `CountVectorizer` gibt Ihnen eine Matrix zurück, in der Dokumente die Zeilen sind und alle vorkommenden Worte/Token die Spalten. Die einzelnen Zellen bedeuten dann das Auftreten eines Wortes/Tokens in einem Dokument (Häufigkeiten).

[15]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
german_stopwords = stopwords.words('german')
cv = CountVectorizer( stop_words = german_stopwords ) # cv wird als␣
↪CountVectorizer initialisiert
```

[16]:
```
cv # der CountVectorizer macht viele Dinge zusätzlich für uns, die nicht␣
↪explizit angegeben sind!
# Achtung, nicht explizite Annahmen!
```

[16]:
```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None,
                stop_words=['aber', 'alle', 'allem', 'allen', 'aller', 'alles',
                            'als', 'also', 'am', 'an', 'ander', 'andere',
                            'anderem', 'anderen', 'anderer', 'anderes',
                            'anderm', 'andern', 'anderr', 'anders', 'auch',
                            'auf', 'aus', 'bei', 'bin', 'bis', 'bist', 'da',
                            'damit', 'dann', …],
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

[17]:
```
text_cv = cv.fit_transform(Texts_tokenized )
text_cv
```

[17]:
```
<4x15 sparse matrix of type '<class 'numpy.int64'>'
        with 18 stored elements in Compressed Sparse Row format>
```

[18]:
```
cv.vocabulary_
```

[18]:
```
{'wetter': 14,
 'heute': 6,
 'schlecht': 11,
 'schlechten': 12,
 'rausgehen': 9,
 'nachrichten': 8,
 'coronavirus': 0,
 'helfen': 5,
 'guter': 4,
 'laune': 7,
 'gute': 3,
 'unabhängig': 13,
 'sars': 10,
 'cov2': 1,
 'entstehen': 2}
```

3.0.1 Die Dokument-Term-Matrix

[19]:
```
from sympy import Matrix
Matrix(text_cv.toarray())
```

[19]:
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Diese Matrix sieht sehr viel weniger ``bevölkert'' aus als im Falle des Bilder (oder einer Koorelationsmatrix von Items). Handelt es sich um eine ``sparse''-Matrix?

[20]:
```
# check sparsity:
np.mean( text_cv.toarray()>0 )
```

[20]: 0.3

[21]:
```
Matrix(np.round( np.corrcoef(text_cv.toarray()), 1 ))
```

[21]:
$$\begin{bmatrix} 1.0 & 0.2 & -0.4 & -0.5 \\ 0.2 & 1.0 & -0.4 & -0.5 \\ -0.4 & -0.4 & 1.0 & -0.1 \\ -0.5 & -0.5 & -0.1 & 1.0 \end{bmatrix}$$

… aber wenn man mit richtigen Sprachdaten arbeitet, werden Dokument-Term-Matrizen sowie Korrelationsmatrizen ``sparse'':

[22]:
```
import pandas as pd
dat = pd.read_csv('../data/df_train.csv').OMT_texts[:200]
cv_omtdata = CountVectorizer( stop_words = german_stopwords )
corr_mat = np.corrcoef( cv_omtdata.fit_transform(dat).toarray() )
```

```
np.round( corr_mat, 1 )
```

[22]:
```
array([[ 1. , -0. , -0. , …, -0. , -0. , -0. ],
       [-0. ,  1. , -0. , …, -0. ,  0.1, -0. ],
       [-0. , -0. ,  1. , …, -0. , -0. , -0. ],
       …,
       [-0. , -0. , -0. , …,  1. , -0. , -0. ],
       [-0. ,  0.1, -0. , …, -0. ,  1. , -0. ],
       [-0. , -0. , -0. , …, -0. , -0. ,  1. ]])
```

Nun ist es das Ziel, auch diese Matrix zu zerlegen. Gibt es latente Dimensionen, die viel Varianz in diesen Daten erklären und damit die Dokumente gut repräsentieren? Dies würde bedeuten, dass man die Dokumente zusammenfassen kann durch abstraktere Themen …

4 Singular Value Decomposition

Einer der wichtigsten Algorithmen der moderneren Mathematik ist die Singulärwertzerlegung (Singular Value Decomposition, SVD) einer Matrix.

Anwendung in Bioinformatik (Kabsch-Algorithmus) zur Identifikation der Proteinstrukturen.

Mittels SVD können (ähnlich wie die Faktorisierung einer Zahl in Primfaktoren) große Matrizen in kleinere Matrizen zerlegt werden, die wichtige Informationen der Originalmatrix enthalten.

``LSA assumes that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis).'' (en wiki)

Gegeben sei eine $m \times n$-Dokument-Term-Matrix, wobei m, die Zeilen, im vorliegenden Fall die Dokumentindices repräsentiert und n, die Spalten, das Vokabular (jedes Token im Vokabular bekommt eine Spalte).

Die Intuition für die Singulärwertzerlegung im 2-dimensionalen Fall ist, dass eine Matrix transformiert wird in einen niederdimensionalen Raum.

Die Erzeugungs-Vorschrift lautet: $M = U\Sigma V^*$ (siehe Wikipedia, sub verbo: Singulärwertzerlegung). Die Matrizes U, Σ und V haben spezifische Bedeutungen, auf die im Folgenden eingegangen wird.

Spezifisch in der Sprachanalyse ist: M… Dokument-Term-Matrix, $U …$ Dokument-Topic-Matrix, Σ… (diagonale) Matrix der Eigenwerte und V^*… Topic-Word-Matrix.

Bevor die SVD implementiert wird, hier einige Literaturhinweise, die in SVD einführen, sie herleiten und Zusammenhänge zur PCA aufzeigen: http://gregorygundersen.com/blog/2018/12/10/svd/#:~:text=The%20singular%20value%20decompositio (from intuition to definition, relationship PCA with SVD)

http://www.ams.org/publicoutreach/feature-column/fcarc-svd (very intuitive image examples of SVD)

http://cognitivemedium.com/emm/emm.html (essay with proof for feasability of 2x2 SVD)

https://nlp.stanford.edu/IR-book/html/htmledition/latent-semantic-indexing-1.html

https://en.wikipedia.org/wiki/Latent_semantic_analysis

[23]:
```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=4)
svd.fit(text_cv.toarray())
u = svd.transform(text_cv.toarray())
s = svd.singular_values_
vh = svd.components_
```

Sanity check: Kann die originale Matrix aus der Dekomposition erneut erzeugt werden? D.h., wir prüfen, ob $U\Sigma V^* = M$ ist.

[24]:
```
recovered_matrix = np.round( u@vh ) # Achtung: u ist in der Funktion␣
↪TruncatedSVD bereits u*s!
recovered_matrix = np.intc(recovered_matrix, type=int)
Matrix(recovered_matrix)
```

[24]:
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Es sieht so aus, als wäre dies tatsächlich die ursprüngliche Dokument-Term-Matrix M.

Zur exakten Überprüfung wird weiterhin algorithmisch überprüft, dass alle Werte exakt übereinstimmen: Hinweis: die Funktion np.all prüft, dass alle Werte True sind.

[25]:
```
np.all(recovered_matrix==text_cv.toarray())
```

[25]: True

5 Interpretation der erhaltenen Matrizen

Wir haben nun drei neue (kleinere) Matrizen, U, Σ und V^*, die es zu interpretieren gilt.

5.1 Die Dokument-Topic-Matrix

[26]:
```
import pandas as pd
doc_concept_association = pd.DataFrame(u,
                                       index = ['{}'.format(k) for k in␣
↪Texts_tokenized],
```

```
                                        columns = [ 'Latent Concept {}'.
↪format(n) for n in range(len(u))] )

np.round( doc_concept_association, 2 )
```

[26]:

	Latent Concept 0	Latent Concept 1	Latent Concept 2	Latent Concept 3
Das Wetter ist heute schlecht	0.00	1.41	-0.00	1.0
Bei diesem schlechten Wetter sollte man nicht r…	-0.00	1.41	-0.00	-1.0
Auch die Nachrichten zum Coronavirus helfen nic…	1.51	0.00	1.65	-0.0
Gute Laune kann nur unabhängig vom Coronavirus …	2.44	0.00	-1.02	-0.0

5.2 Die Singulärwertematrix

[27]:
```
Matrix( np.round( np.diag(s), 1 ) )
```

[27]:
$$\begin{bmatrix} 2.9 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.9 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.4 \end{bmatrix}$$

5.3 Die Topic-Term-Matrix

In der Topic-Term-Matrix sind die Assoziationen der Worte mit den Topics/Concepts dargestellt:

[28]:
```
import pandas as pd
df = pd.DataFrame({ 'Latent concept 0': vh.T[:,0],
                    'Latent concept 1': vh.T[:,1],
                    'Latent concept 2': vh.T[:,2],
```

```
                    'Latent concept 3': vh.T[:,3] }, index = cv.vocabulary_.
 ↪keys() )
np.round( df.sort_values('Latent concept 0'), 2 ).T
```

[28]:

	laune	sars	entstehen	coronavirus	unabhängig	rausgehen	\
Latent concept 0	-0.00	-0.00	-0.00	0.00	0.00	0.18	
Latent concept 1	0.35	0.35	0.71	0.35	0.35	0.00	
Latent concept 2	-0.00	-0.00	-0.00	0.00	0.00	0.44	
Latent concept 3	-0.50	-0.50	0.00	0.50	0.50	-0.00	

	nachrichten	guter	heute	schlecht	schlechten	gute	cov2	\
Latent concept 0	0.18	0.18	0.30	0.30	0.30	0.30	0.30	
Latent concept 1	0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	
Latent concept 2	0.44	0.44	-0.27	-0.27	-0.27	-0.27	-0.27	
Latent concept 3	-0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	

	wetter	helfen
Latent concept 0	0.48	0.48
Latent concept 1	0.00	0.00
Latent concept 2	0.17	0.17
Latent concept 3	-0.00	-0.00

Überprüfen wir Koorelationen in der Topic-Term-Matrix, denn die Korrelationen geben Aufschluss darüber inwieweit die Topics mit denselben Worten assoziiert sind.

[29]:
```
Matrix(np.round(np.corrcoef(vh),2))
```

[29]:
$$\begin{bmatrix} 1.0 & -0.8 & -0.09 & 0.0 \\ -0.8 & 1.0 & -0.05 & 0.0 \\ -0.09 & -0.05 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Die Korrelationsmatrix deutet darauf hin, dass zwei sehr gegensätzliche Topics/Concepts/Cluster die Dokumente beschreiben.

Die Matrix u gibt die Stärke der Zugehörigkeit eines Dokumentes zu den Latent Concepts an.

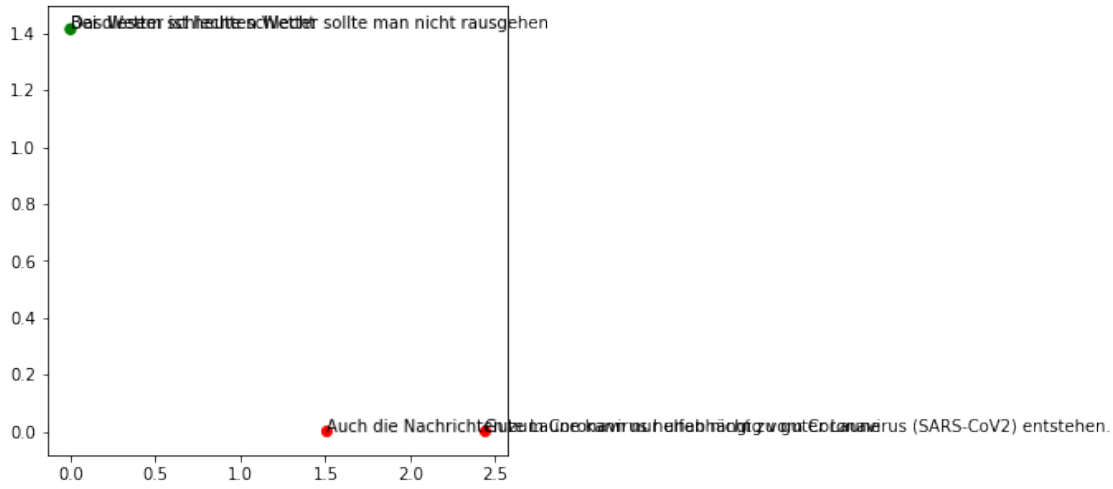
[30]:
```
colors = {'Latent Concept 0': 'red', 'Latent Concept 1': 'green'}
max_indices = np.apply_along_axis( np.argmax, 1, doc_concept_association.iloc[:
 ↪,:2] )
doc_colors = [ colors[concept] for concept in doc_concept_association.
 ↪columns[max_indices] ]
doc_colors
```

[30]: ['green', 'green', 'red', 'red']

[31]:
```
plt.rcParams["figure.figsize"]=5,5
import matplotlib.pyplot as plt

plt.scatter(u[:,0],u[:,1], c=doc_colors)
for x,y,name in zip(u[:,0],u[:,1],Texts_tokenized):
    plt.text(x,y,name)

    # add labels to axes
```



[32]:
```
vh
```

[32]:
```
array([[ 4.79599805e-01,  2.96408981e-01,  2.96408981e-01,
         2.96408981e-01,  1.83190825e-01,  1.83190825e-01,
         2.02962647e-16,  4.79599805e-01,  1.83190825e-01,
        -5.64652491e-16,  2.96408981e-01,  2.02962647e-16,
        -5.64652491e-16,  2.96408981e-01, -3.79904441e-16],
       [ 3.60822483e-16, -5.55111512e-17, -1.24900090e-16,
         0.00000000e+00,  3.88578059e-16,  3.88578059e-16,
         3.53553391e-01,  3.46944695e-16,  3.88578059e-16,
         3.53553391e-01, -1.38777878e-17,  3.53553391e-01,
         3.53553391e-01, -1.38777878e-17,  7.07106781e-01],
       [ 1.67476992e-01, -2.70983466e-01, -2.70983466e-01,
        -2.70983466e-01,  4.38460458e-01,  4.38460458e-01,
         8.32667268e-17,  1.67476992e-01,  4.38460458e-01,
        -4.99600361e-16, -2.70983466e-01,  8.32667268e-17,
        -4.99600361e-16, -2.70983466e-01, -2.77555756e-16],
       [-4.14598911e-16,  5.63785130e-17,  1.59594560e-16,
        -2.22044605e-16, -3.20923843e-16, -3.20923843e-16,
         5.00000000e-01, -3.76434994e-16, -3.20923843e-16,
        -5.00000000e-01, -2.42861287e-17,  5.00000000e-01,
```

```
       -5.00000000e-01, -2.42861287e-17,  1.52655666e-16]])
```

[33]:
```
for i in range(vh.shape[0]):
    print( 'Most important words Latent concept {}: {}'.format( i,
        [ list( cv.vocabulary_.keys() ).__getitem__(n) for n in np.
 ↪argsort(vh[i,:])[-5:] ] ) )
```

```
Most important words Latent concept 0: ['schlechten', 'gute', 'cov2', 'wetter',
'helfen']
Most important words Latent concept 1: ['laune', 'sars', 'coronavirus',
'unabhängig', 'entstehen']
Most important words Latent concept 2: ['helfen', 'wetter', 'rausgehen',
'nachrichten', 'guter']
Most important words Latent concept 3: ['heute', 'entstehen', 'schlecht',
'coronavirus', 'unabhängig']
```

``LSI can be viewed as soft clustering by interpreting each dimension
of the reduced space as a cluster and the value that a document has
on that dimension as its fractional membership in that cluster.''
(https://nlp.stanford.edu/IR-book/html/htmledition/latent-semantic-indexing-1.html)

6 Truncated Singular Value Decomposition

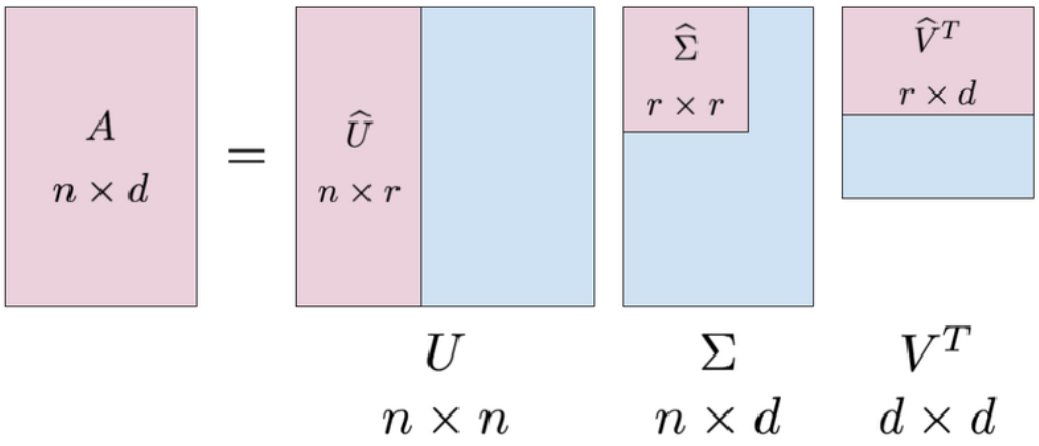
In der SVD kann die Anzahl der Singulärwerte voreingestellt werden. Dies ermöglicht es, vorab eine Anzahl an Dimensionen festzulegen, die maximal identifiziert werden soll.

In der folgenden Grafik werden statt n Singulärwerten nur r Singulärwerte berücksichtigt. (n entspricht hier der Anzahl an Dokumenten und d der Anzahl an Worten).

[34]:
```
# Quelle: https://intoli.com/blog/pca-and-svd/img/svd-matrices.png

img = mpimg.imread('../svd-matrices.png')
plt.rcParams["figure.figsize"]=15,15
plt.axis('off')
plt.imshow(img)
```

[34]:
```
<matplotlib.image.AxesImage at 0x186ebc11d68>
```



In sklearn ist dazu die Funktion TruncatedSVDimplementiert. Das Argument n_components legt die Anzahl der Dimensionen (latente Konzepte) fest, auf die die Matrizen reduziert werden:

[35]:
```
lsa = TruncatedSVD(n_components=2, random_state=42)
u = lsa.fit_transform( text_cv )
vh = lsa.components_
s = lsa.singular_values_

Matrix( np.round( u, 1 ))
```

[35]:
$$\begin{bmatrix} 0.0 & 1.4 \\ 0.0 & 1.4 \\ 1.5 & 0.0 \\ 2.4 & 0.0 \end{bmatrix}$$

Dadurch, dass nun nicht mehr alle Singulärwerte genutzt werden, wird die Dokument-Term-Matrix (M) nur noch approximiert. Dies kann man daran sehen, dass die Dokument-Term-Matrix nicht mehr komplett korrekt aus der Rückwärtsmultiplikation ($U\Sigma V^* = M$) rekonstruiert werden kann:

[36]:
```
import pandas as pd
df = pd.DataFrame({ 'Latent concept 0': vh.T[:,0],
                    'Latent concept 1': vh.T[:,1] }, index = cv.vocabulary_.
 ↪keys())
np.round( df.sort_values('Latent concept 1'), 2 ).T
```

[36]:
```
                 heute  schlechten  gute  cov2  schlecht  helfen  wetter  \
Latent concept 0   0.3         0.3   0.3   0.3       0.3    0.48    0.48
Latent concept 1  -0.0        -0.0  -0.0  -0.0      -0.0   -0.00   -0.00

                 rausgehen  nachrichten  guter  laune  sars  coronavirus  \
```

```
Latent concept 0       0.18          0.18   0.18   0.00  0.00         -0.00
Latent concept 1       0.00          0.00   0.00   0.35  0.35          0.35

                  unabhängig  entstehen
Latent concept 0       -0.00       0.00
Latent concept 1        0.35       0.71
```

6.1 Zuordnung ungesehener Beispiele zu den Latent Concepts, oder: Wie baue ich eine Suchmaschine?

Zunächst müssen wir den neuen Input in den Raum bekannter Wörter ``der Suchmaschine'' transformieren. Dazu benötigen wir das Objekt cv, in dem unser Vokabular gespeichert ist.

[37]:
```
neuer_input = cv.transform( ['Ist das Wetter heute schön?'] ).toarray()
# neuer_input = cv.transform( ['Infektionen mit dem Coronavirus können tödlich␣
 ↪enden.'] ).toarray()
# neuer_input = cv.transform( ['Komplett verrückt'] ).toarray()
neuer_input
```

[37]:
```
array([[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=int64)
```

Im Weiteren können wir mit dem Objekt lsa über die Topic-Term-Matrix die Nähe des neuen Inputs zu den Topics berechnen:

[38]:
```
np.round( lsa.transform( neuer_input ), 1 )
```

[38]:
```
array([[0. , 1.1]])
```

Es können auch ähnliche Dokumente in unserem Corpus ausgegeben werden:

[39]:
```
topic_signature = np.round( lsa.transform( neuer_input ), 1 )

np.round( u@topic_signature.T, 1 )
```

[39]:
```
array([[ 1.6],
       [ 1.6],
       [ 0. ],
       [-0. ]])
```