4_nlp_analysis_schlaak_weise

August 23, 2020

Von Pascal Schlaak, Tim Weise - Natural Language Processing (SoSe 20)

1 Transformation der Daten

In diesem Notebook wird eine Clusteranalyse unserer zuvor vorverarbeiteten Daten durchgeführt, um inhaltlich ähnliche Filme einem Genre zuzuordnen. Basierend darauf soll eine Ähnlichkeitsanalyse innerhalb jedes Clusters durchgeführt werden, um Vorschläge für ähnliche Titel geben zu können.

1.1 Module importieren

Zur Verarbeitung der Datenbasis werden folgende Module benötigt und müssen zuerst importiert werden:

[1]:
```
import spacy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

1.2 Daten einlesen

[2]:
```
PATH_TO_DATA = '../data/movies.json'
# JSON Daten in Dataframe lesen
data = pd.read_json(PATH_TO_DATA)
```

[3]:
```
nlp = spacy.load("en_core_web_sm")
```

1.3 Datenbereinigung

Entfernen der Dateneinträge ohne Zusammenfassung.

[4]:
```
disclaimer = 'It looks like we don\'t have a Synopsis for this title yet.'
for index, movie in data.iterrows():
    if disclaimer in movie['synopsis']:
```

```
        data = data.drop(index)
print(len(data))
```

237

1.4 Vorverarbeitung der Daten

Lemmatisierung der Token, sowie entfernen von Stoppwörtern, Eigennamen und Verben. Speichern der Bow als DataFrame.

[5]:
```
processed_data = []
for index, movie in data.iterrows():
    # Vorverarbeiten der Zusammenfassungen
    processed_data.append({'title': movie['title'], 'bow': ' '.join([str(token.
 ↪lemma_).lower() for token in nlp(movie['synopsis']) if not token.ent_type_␣
 ↪and not token.is_stop and not token.is_punct and token.pos_ != 'VERB'])})
data = pd.DataFrame(processed_data)
```

Das DataFrame sieht nun folgendermaßen aus:

[6]:
```
# Nur erste fünf Einträge anzeigen
data.head()
```

[6]:

	title	bow
0	The Shawshank Redemption	banker wife lover golf pro state death penalty…
1	The Dark Knight	movie gang man clown mask bank mob large porti…
2	The Godfather	guest wedding reception daughter connie head f…
3	The Godfather: Part II	godfather ii parallel storyline chief event mo…
4	Pulp Fiction	restaurant young couple pro con bank versus li…

1.5 Feature-Matrix

Damit wir die vorverarbeitenden BoWs als Eingabe für den k-Means-Clusteralgorithmus verwenden können, erstellen wir eine sogenannte Feature-Matrix: Dabei generieren wir eine Matrix der Häufigkeit aller Wörter unserer Dateneinträge mithilfe des `CountVectorizer` gefolgt von einer Normalisierung mittels `TfidfTransformer`. Wir nutzen hierbei die *Inverse Dokument Frequency* (inverse Dokumentenhäufigkeit): Die inverse Dokumenthäufigkeit misst die Spezifität eines Terms für die Gesamtmenge der betrachteten Dokumente. Ein übereinstimmendes Vorkommen von seltenen Begriffen ist für die Relevanz aussagekräftiger als eine Übereinstimmung bei sehr häufigen Wörtern [Quelle].

[7]:
```
# Matrix von Token mit Frequenz plus Normalisieren mittels␣
 ↪"Inverse-document-frequency" (IDF)
vectorizer = TfidfVectorizer(max_df=0.9, min_df=0.2, ngram_range=(1,3))
# Lernen des Vokabulars und IDF
X = vectorizer.fit_transform(data['bow'])
```

Wir verwenden weitere Parameter bei der Vektorisierung und Normalisierung der Daten. Diese haben sich besonders während der Analyse des Clustering-Algorithmus bewährt:

- max_df: Ignoriert Token mit Frequenz > Wert
- min_df: Ignoriert Token mit Frequenz < Wert
- ngram_range: Definiert n-Gramme der spezifizierten Größen, die benutzt werden

Unsere Feature Matrix besteht somit aus 237 Einträgen, die jeweils über 196 Wörter/Token verfügen. Nicht jeder Dateneintrag besitzt jedes Wort und hat daher bei unbenutzten Token eine Frequenz von 0.

[8]:
```
X.shape
```

[8]:
```
(237, 196)
```

Wir erhalten folgende Matrix:

[9]:
```
X.toarray()
```

[9]:
```
array([[0.0353196 , 0.02406543, 0.        , …, 0.02499595, 0.07219628,
        0.01581857],
       [0.04885937, 0.02219394, 0.04252334, …, 0.        , 0.02219394,
        0.02917682],
       [0.        , 0.        , 0.        , …, 0.        , 0.        ,
        0.12084285],
       …,
       [0.        , 0.        , 0.04800141, …, 0.        , 0.        ,
        0.06587104],
       [0.        , 0.        , 0.        , …, 0.07059206, 0.        ,
        0.13402154],
       [0.0483163 , 0.06584177, 0.        , …, 0.        , 0.        ,
        0.        ]])
```

Wir erhalten somit 196 Wörter, welche das für dieses Projekt benötigte Vokabular darstellen. Bei einer Optimierung dieses Projekts könnte man beispielsweise dieses Vokabular (erweitert mit Stoppwörtern, Satzzeichen) in einem der von spaCy zur Verügung gestellten Sprachmodelle nutzen.

[10]:
```
# Vokabular
feature_names = vectorizer.get_feature_names()
feature_names
```

[10]:
```
['able',
 'action',
 'actually',
 'alive',
 'angry',
 'apartment',
 'apparently',
 'area',
 'arm',
 'attack',
 'attempt',
```

```
'attention',
'away',
'bad',
'bar',
'battle',
'bed',
'big',
'black',
'blood',
'body',
'book',
'boy',
'brother',
'building',
'business',
'car',
'case',
'chance',
'charge',
'child',
'city',
'close',
'company',
'completely',
'control',
'conversation',
'couple',
'crime',
'dark',
'daughter',
'day',
'dead',
'death',
'despite',
'different',
'doctor',
'door',
'earlier',
'end',
'entire',
'escape',
'event',
'eventually',
'eye',
'face',
'fact',
'family',
```

'far',
'father',
'fight',
'film',
'final',
'finally',
'fire',
'floor',
'food',
'foot',
'force',
'free',
'friend',
'game',
'girl',
'good',
'great',
'ground',
'group',
'guard',
'gun',
'hand',
'happy',
'hard',
'head',
'help',
'high',
'home',
'hospital',
'house',
'husband',
'idea',
'immediately',
'information',
'inside',
'instead',
'job',
'large',
'late',
'later',
'leg',
'letter',
'life',
'light',
'like',
'line',
'little',

```
'local',
'long',
'love',
'low',
'man',
'meeting',
'member',
'mind',
'moment',
'money',
'mother',
'movie',
'murder',
'near',
'nearby',
'nearly',
'new',
'news',
'night',
'note',
'number',
'office',
'officer',
'old',
'open',
'order',
'outside',
'paper',
'parent',
'party',
'past',
'people',
'person',
'phone',
'picture',
'piece',
'place',
'plan',
'point',
'police',
'power',
'prison',
'question',
'quickly',
'ready',
'real',
'reason',
```

```
 'relationship',
 'rest',
 'return',
 'right',
 'room',
 'scene',
 'school',
 'short',
 'shot',
 'simply',
 'sister',
 'situation',
 'slowly',
 'small',
 'soldier',
 'son',
 'soon',
 'station',
 'story',
 'street',
 'suddenly',
 'table',
 'thing',
 'time',
 'town',
 'train',
 'tree',
 'true',
 'unable',
 'voice',
 'wall',
 'war',
 'water',
 'way',
 'well',
 'wife',
 'window',
 'woman',
 'word',
 'work',
 'world',
 'wrong',
 'year',
 'young']
```

[11]:
```
len(feature_names)
```

[11]: 196

1.6 k-Means-Clustering

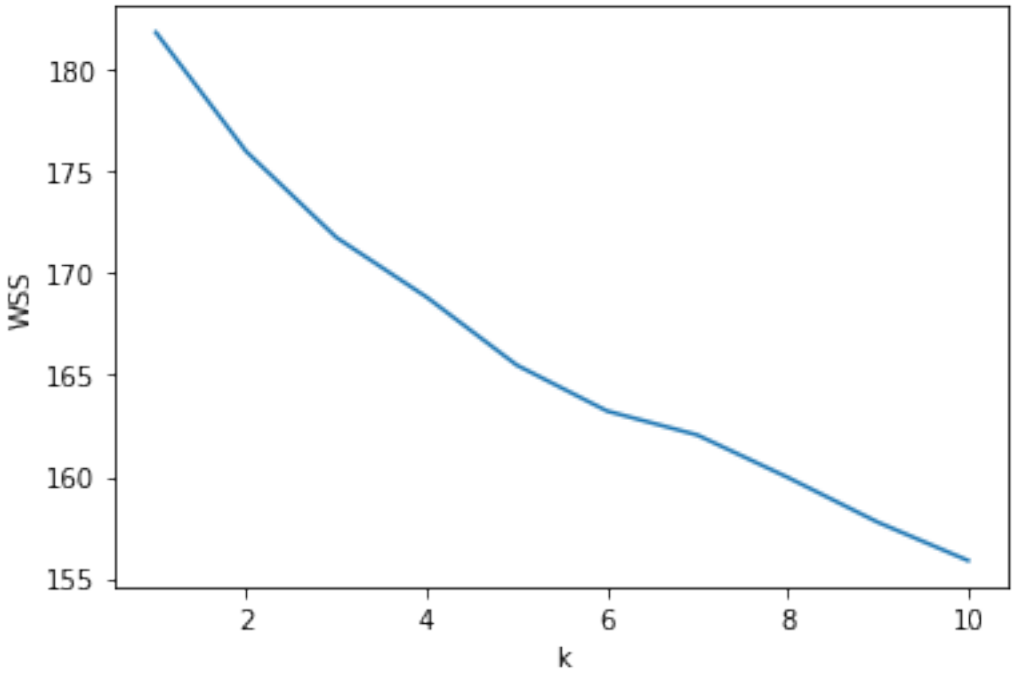
Nachdem wir nun valide Inputs generiert haben, können wir mit der Modellierung unseres Clustering Algorithmus fortfahren. Wir nutzen hierfür den sogenannten `k-Means`-Algorithmus, der aus einer Menge von ähnlichen Objekten eine zuvor definierte Anzahl an Clustern bildet. Hierfür nutzen wir darüberhinaus `sklearn`.

Ein gängiger Ansatz bei der Evaluierung von Clustering-Algorithmen ist es, den Algorithmus in einer Schleife bis zu einem bestimmten Schwellwert an Clustern auszuführen. Man nutzt dabei eine Metrik, um die Performanz des Algorithmus zu messen. Sklearns `inertia` bietet eine Schnittstelle zur Berechnung der `within-cluster sum-of-squares (WSS)` zur Berechnung der Summe der quadrierten Abweichungen von den Cluster-Schwerpunkten. Valide Werte für die Anzahl an Clustern bei unseren Daten sind: 1, …, 237. 237 Cluster sind dennoch nicht sinnvoll, da wir so jeden Dateneintrag einem eigenen Cluster zuordnen würden. Wir definieren unsere maximale Anzahl an Clustern, welche für uns die verschiedenen Genres der Filmhandlungen darstellen sollen, als 10.

[36]:
```
wss = []
limit = 11
for k in range(1, limit):
    model = KMeans(n_clusters=k, max_iter=100)
    model.fit(X)
    wss.append(model.inertia_)
```

Um eine Aussage zum Wählen einer bestimmten Anzahl an Clustern treffen zu können plotten wir mithilfe von Matplotlib alle WSS. Wir wählen dabei eine Clusteranzahl, welche einen “Knick” / niedrigere Steigung aufweist.

[37]:
```
plt.plot(range(1, limit), wss)
plt.xlabel('k')
plt.ylabel('WSS')
plt.show()
```



1.7 Ergebnisevaluierung

Der Algorithmus wird auf den Daten fortgehend mit einer Anzahl von 7 Clustern/Genren evaluiert.

[38]:
```
k = 7
model = KMeans(n_clusters=k, init='k-means++', max_iter=100)
model.fit(X)
```

[38]:
```
KMeans(max_iter=100, n_clusters=7)
```

Der Algorithmus terminiert nach 8 Iterationen.

[39]:
```
model.n_iter_
```

[39]:
```
8
```

WSS aktueller Cluster:

[40]:
```
model.inertia_
```

[40]:
```
160.88774168891274
```

Wir fügen alle Cluster-Zuordnungen als neue Spalte im DataFrame an, damit man eine direkte Zuordnung zwischen Filmtitel und Cluster hat.

[41]:
```
data['cluster'] = model.labels_.tolist()
```

[42]:
```
data.head()
```

[42]:

	title	bow	cluster
0	The Shawshank Redemption	banker wife lover golf pro state death penalty…	4
1	The Dark Knight	movie gang man clown mask bank mob large porti…	5
2	The Godfather	guest wedding reception daughter connie head f…	3
3	The Godfather: Part II	godfather ii parallel storyline chief event mo…	3
4	Pulp Fiction	restaurant young couple pro con bank versus li…	2

1.8 Clustermerkmale

Man kann nun die Eigenschaften eines Clusters untersuchen, in dem man sich die relevantesten Token der einzelnen Cluster anschaut. Dafür wird über jedes einzelne Cluster iteriert und eine bestimmte Anzahl an relevantesten Tokens ausgegeben, welche am nächsten zum Mittelpunkt des Clusters sind. Darüberhinaus können für jedes Cluster mehrere Filme ausgeben werden, um das Cluster genauer beschreiben zu können.

[43]:
```
true_k = np.unique(data['cluster']).shape[0]
# Indizes von Tokens pro Cluster nach Relevanz sortieren
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
# 10 wichtigsten Tokens von jeweiligem Cluster ausgeben
for i in range(true_k):
    print("\nCluster {}:\n{}".format(i, ', '.join(feature_names[ind] for ind in␣
 ↪order_centroids[i, :15])))
```

```
Cluster 0:
life, film, wife, husband, time, friend, relationship, story, love, letter,
book, child, son, mother, young

Cluster 1:
father, school, boy, parent, mother, home, letter, friend, child, new, family,
time, daughter, life, train

Cluster 2:
apartment, girl, fight, door, room, home, man, car, house, away, gun, head,
woman, face, time

Cluster 3:
family, house, father, police, business, man, son, car, dead, war, home, local,
member, daughter, gun
```

```
Cluster 4:
man, guard, town, woman, soldier, time, away, room, wife, black, life, gun,
prison, people, brother

Cluster 5:
car, police, man, money, room, house, time, home, phone, away, mother, job,
wife, officer, father

Cluster 6:
soldier, battle, war, son, man, attack, officer, fire, power, father, city,
force, time, group, hand
```

1.9 Clusterzuordnungen

[44]:
```
for i in range(len(set(model.labels_))):
    indices = np.where(data['cluster'] == i)[0].tolist()
    print('\nCluster {}: {} Filme\n{}'.format(i, len(indices), ', '.
 ↪join([data['title'][j] for j in indices[:10]])))
```

```
Cluster 0: 32 Filme
Anand, Casablanca, The Intouchables, Avengers: Endgame, The Hunt, Good Will
Hunting, American Beauty, Eternal Sunshine of the Spotless Mind, Witness for the
Prosecution, Life Is Beautiful

Cluster 1: 34 Filme
12 Angry Men, Forrest Gump, Grave of the Fireflies, Cinema Paradiso, Whiplash, 3
Idiots, Dangal, Once Upon a Time in America, Amélie, Children of Heaven

Cluster 2: 29 Filme
Pulp Fiction, Fight Club, Léon: The Professional, Joker, The Shining, The Lives
of Others, Rear Window, Snatch, Toy Story, Toy Story 3

Cluster 3: 10 Filme
The Godfather, The Godfather: Part II, Goodfellas, The Pianist, Coco, Parasite,
Drishyam, Hotel Rwanda, Gran Torino, Gangs of Wasseypur

Cluster 4: 44 Filme
The Shawshank Redemption, Schindler's List, The Good, the Bad and the Ugly, Once
Upon a Time in the West, American History X, The Prestige, Django Unchained,
Memento, Princess Mononoke, Oldboy

Cluster 5: 45 Filme
The Dark Knight, Inception, Psycho, Back to the Future, Terminator 2: Judgment
Day, The Usual Suspects, The Departed, It's a Wonderful Life, The Dark Knight
Rises, Andhadhun
```

```
Cluster 6: 43 Filme
The Lord of the Rings: The Return of the King, The Lord of the Rings: The
Fellowship of the Ring, Hamilton, The Lion King, Gladiator, Seven Samurai,
Avengers: Infinity War, Spider-Man: Into the Spider-Verse, Raiders of the Lost
Ark, WALL·E
```

Wir können dann versuchen jedes Cluster durch ein Genre zu beschreiben:

- Cluster 0: Drama, Romance
- Cluster 1: Drama, Emotional
- Cluster 2: Drama, Misterös, Krimi
- Cluster 3: Drama, Krimi, Mafia
- Cluster 4: Drama, History, Gewalt
- Cluster 5: Action, Krimi
- Cluster 6: Action, Abenteuer, Scifi, Krieg

Was ebenfalls für die Clusteraufteilung spricht, ist der Fakt, dass Sequels (Fortsetzungsfilme) mit ähnlicher Handlung dem gleichen Cluster zugeordnet wurden (siehe beispielsweise: Der Herr der Ringe, der Pate). Trotzdem gibt es ein paar Filme die sich zumindest subjektiv innerhalb des gleichen Clusters unterscheiden, beispielsweise Pulp Fiction und Toy Story. Man könnte hierfür eine genauere Analyse der zwei Filme durchführen, um zu entscheiden, ob deren Handlungen wirklich ähnlich sind.

Die Aufteilung der Filme ist dennoch sehr zufriedenstellend, was die Qualität der Datenvorverarbeitung verdeutlicht.

1.10 Ähnlichkeitsanalyse der Filme eines Clusters

Basierend auf dieser Aufteilung aller Filme in Clustern/Genres, kann nun eine Ähnlichkeitsanalyse durchgefürt werden: Es bietet sich an, nur Filme innerhalb eines Genres zu vergleichen, da sich diese inhaltlich am ähnlichsten sind.

spaCy bietet hierfür eine Methode namens `similarity()`, welche die Kosinus-Ähnlichkeit zwischen zwei Vektoren (z.B. Token oder Dokumenten) berechnet. spaCy setzt hierfür voraus, dass man ein Sprachmodell verwendet, welches über Wort-Vektoren verfügt. Wir importieren dafür zunächst ein umfangreicheres Sprachmodell:

[45]:
```
# Erweitertes Sprachmodel laden (enthält Vekoren)
nlp = spacy.load("en_core_web_lg")
```

Der folgende Algorithmus berechnet somit für jeden Film in jedem Cluster mit jedem weiteren Film des Clusters dessen Kosinus-Ähnlichkeit und speichert diese gerundet in einer Confusion-Matrix. Es ergibt sich folgende Tabelle, wobei jede Reihe einen Film darstellt und jede Spalte dessen Änlichkeit zu einem weiteren Film.

Hinweis: Die Berechnung der Confusion-Matrix dauert sehr lange.

[46]:
```
values = []
for i in range(k):
    indices_cluster = data.index[data['cluster'] == i].tolist()
    for j in range(len(indices_cluster)):
```

```
        row = {'title': data['title'][indices_cluster[j]]}
        for y in range(len(indices_cluster)):
            row[data['title'][indices_cluster[y]]] =␣
↪round(nlp(data['bow'][indices_cluster[j]]).
↪similarity(nlp(data['bow'][indices_cluster[y]])), 3)
        values.append(row)
similarity = pd.DataFrame(values)
```

[47]:
```
similarity
```

[47]:
```
                    title  Anand  Casablanca  The Intouchables  \
0                   Anand  1.000       0.882             0.887
1              Casablanca  0.882       1.000             0.921
2       The Intouchables  0.887       0.921             1.000
3      Avengers: Endgame  0.887       0.931             0.895
4              The Hunt  0.879       0.900             0.919
..                    …    …          …                 …
232        Hacksaw Ridge    NaN         NaN               NaN
233            Downfall    NaN         NaN               NaN
234   Howl's Moving Castle    NaN         NaN               NaN
235  Judgment at Nuremberg    NaN         NaN               NaN
236           Metropolis    NaN         NaN               NaN

     Avengers: Endgame  The Hunt  Good Will Hunting  American Beauty  \
0                0.887     0.879              0.935            0.916
1                0.931     0.900              0.944            0.938
2                0.895     0.919              0.928            0.935
3                1.000     0.881              0.942            0.924
4                0.881     1.000              0.934            0.928
..                 …       …                  …                …
232              NaN       NaN                NaN              NaN
233              NaN       NaN                NaN              NaN
234              NaN       NaN                NaN              NaN
235              NaN       NaN                NaN              NaN
236              NaN       NaN                NaN              NaN

     Eternal Sunshine of the Spotless Mind  Witness for the Prosecution  …  \
0                                    0.913                        0.859  …
1                                    0.888                        0.902  …
2                                    0.877                        0.883  …
3                                    0.907                        0.864  …
4                                    0.880                        0.879  …
..                                     …                          …  …
232                                  NaN                          NaN  …
233                                  NaN                          NaN  …
234                                  NaN                          NaN  …
235                                  NaN                          NaN  …
```

```
236                                    NaN                           NaN  …

     The General  Gone with the Wind  Barry Lyndon  How to Train Your Dragon  \
0            NaN                 NaN           NaN                       NaN
1            NaN                 NaN           NaN                       NaN
2            NaN                 NaN           NaN                       NaN
3            NaN                 NaN           NaN                       NaN
4            NaN                 NaN           NaN                       NaN
..           …                   …             …                         …
232        0.818               0.932         0.924                     0.828
233        0.861               0.960         0.943                     0.871
234        0.893               0.939         0.903                     0.931
235        0.717               0.896         0.893                     0.760
236        0.837               0.960         0.947                     0.882

     Harry Potter and the Deathly Hallows: Part 2  Hacksaw Ridge  Downfall  \
0                                             NaN            NaN       NaN
1                                             NaN            NaN       NaN
2                                             NaN            NaN       NaN
3                                             NaN            NaN       NaN
4                                             NaN            NaN       NaN
..                                            …              …         …
232                                         0.869          1.000     0.938
233                                         0.881          0.938     1.000
234                                         0.929          0.843     0.899
235                                         0.789          0.890     0.893
236                                         0.885          0.898     0.942

     Howl's Moving Castle  Judgment at Nuremberg  Metropolis
0                     NaN                    NaN         NaN
1                     NaN                    NaN         NaN
2                     NaN                    NaN         NaN
3                     NaN                    NaN         NaN
4                     NaN                    NaN         NaN
..                    …                      …           …
232                 0.843                  0.890       0.898
233                 0.899                  0.893       0.942
234                 1.000                  0.780       0.906
235                 0.780                  1.000       0.892
236                 0.906                  0.892       1.000

[237 rows x 238 columns]
```

1.11 Min-Max-Normalisierung

Wie man zuvor erkennen konnte, sind die Ähnlichkeiten aller Filme eines Clusters relativ hoch. Dies liegt an der größeren Anzahl an gleichen Wörtern (vor allem in deren Grundform) in jeder

Zusammenfassung. Es ist deshalb sinnvoll alle Ergebnisse zu normalisieren, um ein breiteres Spektrum an Ähnlichkeiten zu erhalten. Die sogenannte Min-Max-Normalisierung bietet sich hierfür an.

Es wird der minimale- und maximale Wert der Confusion Matrix benötigt:

[48]:
```
minima = min(similarity.min(axis = 1).values)
minima
```

[48]: 0.683

[49]:
```
maxima = max(similarity.max(axis = 1).values)
maxima
```

[49]: 1.0

Anschließend wird vorrübergehend die Spalte der Titel entfernt, damit alle Werte einer Zahl entsprechen und dementsprechend normalisiert werden können.

[50]:
```
titles = similarity['title']
similarity = similarity.drop(['title'], axis=1)
```

Anhand der folgenden Formel können unsere Daten normalisiert werden:

[51]:
```
similarity = (similarity - min(similarity.min(axis = 1).values))/
 ↪(max(similarity.max(axis = 1).values) - min(similarity.min(axis = 1).values))
```

Wir konkatinieren wiederrum unsere Titel-Spalte mit den nun normalisieren Ähnlichkeitswerten.

[52]:
```
similarity = pd.concat([titles, similarity], axis=1)
similarity
```

[52]:

	title	Anand	Casablanca	The Intouchables	\
0	Anand	1.000000	0.627760	0.643533	
1	Casablanca	0.627760	1.000000	0.750789	
2	The Intouchables	0.643533	0.750789	1.000000	
3	Avengers: Endgame	0.643533	0.782334	0.668770	
4	The Hunt	0.618297	0.684543	0.744479	
..	…	…	…	…	
232	Hacksaw Ridge	NaN	NaN	NaN	
233	Downfall	NaN	NaN	NaN	
234	Howl's Moving Castle	NaN	NaN	NaN	
235	Judgment at Nuremberg	NaN	NaN	NaN	
236	Metropolis	NaN	NaN	NaN	

	Avengers: Endgame	The Hunt	Good Will Hunting	American Beauty	\
0	0.643533	0.618297	0.794953	0.735016	
1	0.782334	0.684543	0.823344	0.804416	
2	0.668770	0.744479	0.772871	0.794953	

```
3             1.000000  0.624606           0.817035         0.760252
4             0.624606  1.000000           0.791798         0.772871
..                 …       …                  …                …
232               NaN       NaN                NaN              NaN
233               NaN       NaN                NaN              NaN
234               NaN       NaN                NaN              NaN
235               NaN       NaN                NaN              NaN
236               NaN       NaN                NaN              NaN

     Eternal Sunshine of the Spotless Mind  Witness for the Prosecution  …  \
0                                 0.725552                     0.555205  …
1                                 0.646688                     0.690852  …
2                                 0.611987                     0.630915  …
3                                 0.706625                     0.570978  …
4                                 0.621451                     0.618297  …
..                                     …                            …  …
232                                 NaN                          NaN  …
233                                 NaN                          NaN  …
234                                 NaN                          NaN  …
235                                 NaN                          NaN  …
236                                 NaN                          NaN  …

     The General  Gone with the Wind  Barry Lyndon  How to Train Your Dragon  \
0            NaN                 NaN           NaN                       NaN
1            NaN                 NaN           NaN                       NaN
2            NaN                 NaN           NaN                       NaN
3            NaN                 NaN           NaN                       NaN
4            NaN                 NaN           NaN                       NaN
..           …                   …             …                         …
232     0.425868            0.785489      0.760252                  0.457413
233     0.561514            0.873817      0.820189                  0.593060
234     0.662461            0.807571      0.694006                  0.782334
235     0.107256            0.671924      0.662461                  0.242902
236     0.485804            0.873817      0.832808                  0.627760

     Harry Potter and the Deathly Hallows: Part 2  Hacksaw Ridge  Downfall  \
0                                             NaN            NaN       NaN
1                                             NaN            NaN       NaN
2                                             NaN            NaN       NaN
3                                             NaN            NaN       NaN
4                                             NaN            NaN       NaN
..                                            …              …         …
232                                      0.586751       1.000000  0.804416
233                                      0.624606       0.804416  1.000000
234                                      0.776025       0.504732  0.681388
235                                      0.334385       0.652997  0.662461
236                                      0.637224       0.678233  0.817035
```

```
     Howl's Moving Castle  Judgment at Nuremberg  Metropolis
0                     NaN                    NaN         NaN
1                     NaN                    NaN         NaN
2                     NaN                    NaN         NaN
3                     NaN                    NaN         NaN
4                     NaN                    NaN         NaN
..                    …                      …           …
232              0.504732               0.652997    0.678233
233              0.681388               0.662461    0.817035
234              1.000000               0.305994    0.703470
235              0.305994               1.000000    0.659306
236              0.703470               0.659306    1.000000

[237 rows x 238 columns]
```

Alle Werte wurden somit auf einen Wertebereich zwischen 0 und 1 skaliert, wobei 0 einer Ähnlichkeit von 0.683 und 1 einer Ähnlichkeit von 1 entspricht.

1.12 Beispiel: Filmempfehlung

Ein letztes Beispiel soll nun unseren anfänglich definierten Anwendungsfall abbilden: Wir gehen davon aus, dass ein Nutzer den Film *Inception* angesehen hat, begeistert war und nun einen möglichst ähnlichen Titel wiedergeben möchte.

Wir identifizieren hierfür den Film *Inception* in unserer Datenstruktur (in einer realen Anwendung beispielsweise ein Eintrag in einer Datenbank). Anschließend transponieren wir den Datensatz bestehend aus einem Film und dessen Ähnlichkeiten zu weiteren Filmen aus dem gleichen Cluster, um einen Spaltenvektor zu erlangen. Schlussendlich sortieren wir unsere Liste der Ähnlichkeiten in absteigender Reihenfolge ("Order By"-Statement in einer Datenbankabfrage).

Achsen swappen

[58]:
```
inception = similarity.loc[similarity['title'] == 'Inception']
inception = inception.T
inception = inception.drop(inception.index[0])
```

[58]:
```
                          150
Anand                     NaN
Casablanca                NaN
The Intouchables          NaN
Avengers: Endgame         NaN
The Hunt                  NaN
…                         …
Hacksaw Ridge             NaN
Downfall                  NaN
Howl's Moving Castle      NaN
Judgment at Nuremberg     NaN
Metropolis                NaN
```

```
[237 rows x 1 columns]
```

Die Werte werden absteigend sortiert:

[60]:
```
inception.sort_values(by=[150], ascending=False)
```

[60]:
```
                              150
Inception                       1
The Truman Show          0.933754
Batman Begins            0.899054
The Wolf of Wall Street  0.886435
Back to the Future       0.886435
…                              …
Hacksaw Ridge                NaN
Downfall                     NaN
Howl's Moving Castle         NaN
Judgment at Nuremberg        NaN
Metropolis                   NaN

[237 rows x 1 columns]
```

Wir sind nun in der Lage, Vorschläge für inhaltlich ähnliche Titel zu geben. Hierfür könnte eine Reihe von Filmen (siehe Netflix-Beispiel in Übersicht) visualisiert werden oder ein einzelner Film ausgeben werden. Man muss beachten, dass man hierbei den gleichen Titel vernachlässigt. In einem realen Anwendungsfall könnte man ebenfalls alle Titel, welche der Nutzer bereits gesehen hat, ausschließen. Beispielsweise könnte man dem Nutzer als nächstes den Film: Die Truman Show empfehlen, insofern er diesen noch nicht gesehen hat. Dieser passt inhaltlich zu Inception, da es sich in beiden Filmen teilweise um eine Simulation dreht.

1.13 Fazit

Mithilfe einer durch einen Web-Crawler erstellten Datenbasis, geeigneten Vorverarbeitungen der Daten und dem k-Means-Clustering-Algorithmus konnten wir die Filmzusammenfassungen inhaltlich clustern. Die Kosinus-Ähnlichkeit ermöglichte uns darüberhinaus einen Vergleich aller Filme in einem Cluster. Zusammengefasst war es uns möglich, einen Teil eines realen Anwendungsfalls des Feldes Natural Language Processing und Maschinen Learning im Bereich Streaming-Anwendungen, abzubilden.

- Zurück zur Übersicht

[ ]:
```

```