Duale Hochschule Baden-Württemberg
Mannheim

Data Exploration Projekt Dokumentation

Recommender Systems

Studiengang Wirtschaftsinformatik

Studienrichtung Data Science

Verfasser/in:	Simon Schmid, Andreas Dichter, Elisa Dandin
Matrikelnummer:	9917195, 6104795, 3636997
Studienfach:	Data Exploration
Kurs:	WWI19DSB
Studiengangsleiter:	Prof. Dr. Drabant
Dozenten:	Herr Poll & Herr Schön
Bearbeitungszeitraum:	11.05.2021 – 13.07.2021

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Als Thema aus den wurde ein „Recommender Systems“ ausgewählt. Im Zuge dessen wurde beschlossen, einen Algorithmus zu entwickeln, welcher darauf abzielt, dem Benutzer relevante Inhalte, in diesem Falle passende Filme vorzuschlagen.

1.1 Konzept

Es wurden 2 Konzepte gewählt, um die verschiedenen Herangehensweisen aufzuzeigen. Das erste Konzept, welches entwickelt wurde, nutzt das Prinzip der Content Based Recommendation. Das bedeutet, dieses System nutzt charakteristische Informationen der jeweiligen Filme. Ein Benutzer erhält Empfehlungen auf Grundlage des Inhalts der Schauspieler oder des Regisseurs eines Filmes. Bei dem Content Based Algorithmus geht es folglich darum, dass Filmvorschläge aufgrund von einem bestimmten Film, den der jeweilige Nutzer auf Netflix ausgesucht hat, erfolgen.
Zum anderen wird das Konzept der Rating Based Recommendation angewendet. Dieser Algorithmus wird hier verwendet, um allgemeine Vorschläge auf dem jeweiligen Netflix Profil zu erhalten. Für diese Empfehlung werden alle Filme, die der Nutzer bisher angesehen hat, genutzt, um eine allgemeine Empfehlung liefern zu können. Der Algorithmus sucht Filme, die Nutzer, die den Film positiv bewertet haben, ebenfalls positiv bewertet haben.

1.2 Wirtschaftlicher Vorteil

Der wirtschaftliche Vorteil eines Film Empfehlungssystems kommt auf den Kontext an. Im Fokus stand die Verwendung bei Streaminganbietern, wie zum Beispiel Netflix. Der wirtschaftliche Vorteil ist hier, dass die Film Empfehlung einen festen Bestandteil des Servicemodells solcher Anbieter ausmacht. Die Kunden verwenden den Service, um ohne zusätzliche Kosten Filme zu konmsumieren. Dennoch wird erwartet, dass Netflix ihnen weitere Inhalte vorschlägt, um die Suche nach weiteren Filmen und Serien zu verkürzen. Je besser dieser Service ist desto eher haben die Kunden neuen Kontent vor sich und sind nicht gelangweilt. Ist der Nutzer der Meinung, dass er nichts ansprechendes mehr entdecken kann kündigt er den Dienst. Dies gilt es zu vermeiden.
Handelt es sich beispielsweise um einen Service wie Itunes, kann so das Kaufverhalten

des Kunden gefördert werden. Werden ihm in beiden besagten Szenarien (Suchergebnisse oder anhand von bewerteten/gekauften Filmen) für ihn ansprechende Empfehlungen geliefert, ist die Wahrscheinlichkeit höher, dass er einen weiteren Kauf tätigt.

1.3 Auswahl des Datasets

Das Dataset beinhaltet Metadaten für alle 45000 Filme, welche in dem Full MovieLens Dataset gelistet sind. Er beinhaltet außerdem alle Informationen der Filme über Besetzung, Crew, Plot-Schlüsselwörter, Budget, Einnahmen, Plakate, Veröffentlichungsdaten, Sprachen, Produktionsfirmen, Länder, TMDB-Stimmenzahlen und Stimmendurchschnitte.

Die verwendeten Dateien des Datensatz sind im folgenden aufgelistet:

- movies_metadata.csv: enthält Informationen zu 45.000 Filmen, sowie zu den jeweiligen Postern, Kulissen, Budget, Einnahmen, Veröffentlichungsdaten, Sprachen, Produktionsländer, Firmen, etc.
- keywords.csv: besteht aus den Schlüsselwörtern über die Filmhandlung
- credits.csv: enthält die Cast- und Crew-Informationen
- ratings_small.csv: besteht aus der Teilmenge von 100.000 Bewertungen von 700 Benutzern zu 9.000 Filmen

2 Rating Based Recommendation

Als erstes werden alle relevanten Packages und Daten importiert. Damit die Daten importiert und gelesen werden können ist pandas fundamental.

```
import numpy as np
import pandas as pd
import sklearn
from scipy import sparse
import matplotlib.pyplot as plt
import seaborn as sns
import collections
import random
import copy
sns.set_style('whitegrid')
#matplotlib inline
```

2.1 Implementierung

Importieren und Analysieren

Nachdem alle Packages importiert wurden, werden die Daten importiert (ratings.csv und movies.csv).

```
ratings = pd.read_csv (r'./../data/ratings.csv')
movies = pd.read_csv (r'./../data/movies.csv')
```

Der inhaltliche Aufbau der Tabellen ist im folgenden zu erkennen.

Viewing data

In [314]:
```
ratings.head()
```

Out[314]:

	userId	movieId	rating	timestamp
0	1	110	1.0	1425941529
1	1	147	4.5	1425942435
2	1	858	5.0	1425941523
3	1	1221	5.0	1425941546
4	1	1246	5.0	1425941556

In [315]:
```
movies.head()
```

Out[315]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Folgernd wurde zu jedem Film das durchschnittliche Rating berechnet sowie die Menge an Ratings. Mit der folgenden Funktion wurde eine Tabelle erstellt, die diese Informationen enthält:

```
rating_info = pd.DataFrame(ratings.groupby('movieId')['rating'].mean())
rating_info['count'] = pd.DataFrame(ratings.groupby('movieId')['rating'].count())
rating_info.head()
```

Algorithmus

Im ersten Schritt wurde eine Pivot Tabelle erstellt, bei welcher alle userIds mit den movieIds in Relation gesetzt werden. Sie zeigt die Bewertungen der User zu den jeweiligen Filmen an.

```
rating_pivot = pd.pivot_table(ratings, index='userId', columns='movieId', values='rating')
rating_pivot.head()
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Abbildung 2.1: Pivot Tabelle

Die Abbildung 2.2 visualisiert die Pivot Tabelle. In den einzelnen Zellen der Tabelle steht von jedem Nutzer zu jedem Film die Bewertung. Wenn ein Nutzer für einen Film keine Bewertung abgegeben hat, wird dieses Feld mit „NaN“ ausgefüllt.

Im folgenden wird die Hauptfunktion definiert:

```
def recommendation(movie_id):
    #calculating correlation to other users
    correlation = rating_pivot.corrwith(rating_pivot[movie_id])
    #creating table
    corr_movie = pd.DataFrame(correlation, columns=['correlation'])
    #dropping movies non values => no correlation
    corr_movie.dropna(inplace=True)
    #adding total count of ratings for each movie to the table
    corr_movie = corr_movie.join(rating_info['count'])
    #listing movies by correlation score and just using movies with more then the 90th percentile of the total count of ratings per movie
    cutoff = rating_info['count'].quantile(0.90)
    corr_movie = corr_movie[corr_movie['count']>cutoff].sort_values('correlation', ascending=False)
    #deleting movie that the recommendation is based on from recommendation list
    if movie_id in corr_movie.index:
        corr_movie = corr_movie.drop([movie_id])
    #changing index to ranking and movieId to column
    corr_movie['movieId'] = corr_movie.index
    corr_movie = corr_movie.set_index(pd.Index(list(range(len(corr_movie)))))
    return corr_movie
```

Abbildung 2.2: Hauptfunktion

Im ersten Schritt wird mit Hilfe der corr.with Funktion von Pandas[1] die Spalte der übergebenen movieID aus der Pivot Tabelle mit der gesamten Pivot Tabelle korreliert.

[1] Vgl. *pandas.DataFrame.corr - Documentation* 2021.

Dann wird eine Tabelle mit allen Film IDs und den Korrelations Scores zu jedem Film erstellt. Nun werden alle NAN Felder (Filme ohne Korrelation) aus der Tabelle gelöscht. Anschließend wird die jeweilige Zahl an insgesamt abgegeben Bewertungen von jedem Film zur Tabelle hinzugefügt. Die Zahl der Bewertungen ist wichtig, da ein von vielen Benutzern bewerteter Film eine aussagekräftigere Empfehlung ist. Dementsprechend sind Filme, die weniger Bewertungen haben, für die Recommendation von geringerem Wert. Um eben diese auszuschließen, wird ein Cutoff verwendet. Der Cutoff ist die Anzahl an Ratings, die mindestens gegeben sein muss, damit der jeweilige Film für eine Empfehlung in Erwägung gezogen wird. Diese Berechnung erfolgt anhand des neunzigsten Perzentils der Anzahl an Ratings für die einzelnen Filme. Das bedeutet, der Film muss eine höhere Zahl an Ratings erhalten haben als 90% der anderen Filme. Bei dem verwendeten Datensatz beträgt der Cutoff so 27 Bewertungen.
Ein Problem gilt es noch zu lösen: Der Film, der die größte Korrelation mit dem ausgewählten Film hat, ist er selbst. Aufgrund dessen steht dieser in der Tabelle mit den Recommendations auch an oberster Stelle. Um nicht den gleichen Film, von dem die Recommendation ausgeht, vorzuschlagen wird dieser also aus der Liste gelöscht.
In einem letzten Schritt wird die Tabelle angepasst, indem der Index der Tabelle in eine Platzierung von 0 bis X umgewandelt wird. Der ursprüngliche Index (die Film Ids) wird als normale Spalte angefügt. Das Ergebnis der Recommendation Funktion ist in der folgenden Abbildung zu sehen.

	correlation	count	movieId
0	1.0	52	6870
1	1.0	31	2953
2	1.0	87	2011
3	1.0	48	2019
4	1.0	58	34162
...	...	...	...
400	-1.0	83	1101
401	-1.0	40	2541
402	-1.0	90	1610
403	-1.0	39	2746
404	-1.0	59	3535

Abbildung 2.3: Ausgabe Recommendation Funktion

Ziel des Algorithmus ist es, eine Recommendation für mehrere Filme (IDs) zu erstellen. Um dies zu gewährleisten und den Recommendation Score für mehrere Filme zu erweitern, wird die folgende Funktion verwendet:

```
def multi_recommendation(movie_ids):
    #function to get recommendation as an array of titles for multiple inputs
    rec_movies_list = []
    rec_movies_dict = {}
    for i in movie_ids:
        global rec_count
        rec = recommendation(int(i))
        #skip if no correlating movies
        if rec.empty:
            continue
        #adding correlation score for top 5 recommended movies to dictionary
        for k in range(5):
            id = rec.iloc[k].array[2]
            #checking if id has already been recommended
            if id in rec_movies_list:
                #if already recommended the correlation score of both cases are added up
                rec_movies_copy = copy.deepcopy(rec_movies_dict)
                rec_movies_copy.update({id: rec_movies_dict.get(id) + rec.iloc[k][0]})
                rec_movies_dict = rec_movies_copy
            else:
                rec_movies_dict[rec.iloc[k][2]] = rec.iloc[k][0]
                rec_movies_list.append(id)

    return sorted(rec_movies_dict, key=rec_movies_dict.get, reverse=True)[:10]
```

Abbildung 2.4: Recommendation für mehrere Ids

Die Funktion erwartet als Eingabe ein Array aus Film D, zu denen eine gesamtheitliche Filmempfehlung berechnet werden soll. Dieses wird im Jupyter Notebook zur Demonstration fest vorgegeben.
Im ersten Schritt wird die oben beschriebene Recommendation Funktion für jede eingegebene ID durchgeführt, und die Ergebnisse werden anschließend in der Variable „rec“ gespeichert. Falls für einen Film keine Recommendation geliefert werden kann, wird diese übersprungen. Zu jedem der Filme werden die Top 5 Vorschläge verwendet. Es wird die Film Id und der zugehörige Korrelations Score in dem Dictionary „rec_movies_dict“ gespeichert. Zusätzlich wird die Id in einem Array gespeichert, um eine reine Liste mit den vorgeschlagenen Ids zu erhalten.
Bei der Eingabe mehrerer Film Ids für die Recommendation besteht die Möglichkeit, dass ein Film mehrfach vorgeschlagen wird. Ist dies der Fall, wird der Korrelations Score zu dem schon bestehenden Score des Films hinzuaddiert. So wird den mehrfach vorgeschlagenen Filmen eine höhere Gewichtung zugeteilt.
Abschließend gibt die Funktion ein Array aus den Ids der Top 10 vorgeschlagenen Filme mit den höchsten Scores in absteigender Reihenfolge zurück.

2.2 Problembehandlung

Während der Arbeit an dem Algorithmus wurde zuerst nur mit den IDs gearbeitet, da das ratings Dataset keine Titel beinhaltet. Daher ist es erst im Nachhinein aufgefallen, dass bei der Ausgabe der eigentlichen Titel der vorgeschlagenen Filme Probleme aufgrund des Datensatzes auftreten. Aufgefallen ist, dass einige der Ids aus dem ratings Dataset nicht im Metadatenset, indem die Titel gespeichert sind, vorkommen.

Um zu untersuchen, wie gravierend des Problems ist, wurde sowohl für den kleinen als auch für den großen ratings Datensatz überprüft, wie viele Movie IDs aus dem ratings Dataset, mit den IDs aus den Metadaten übereinstimmen und wie viele nicht.
Im Folgenden wird die entsprechende Analyse veranschaulicht.

```
from IPython.display import clear_output

a=1
length = str(len(ratings.movieId))
not_in_meta = []
in_meta = []

for i in ratings.movieId:
    clear_output(wait=True)
    print(str(a)+'/'+length)
    a+=1
    if not movies_metadata.loc[movies_metadata['id'] == str(i), 'original_title'].array:
        not_in_meta.append(i)
    else:
        in_meta.append(i)

print('ratings:')
print('Not in: ', len(not_in_meta))
print('In: ', len(in_meta))
```

```
26024289/26024289
ratings:
Not in:  14587721
In:  11436568
```

Abbildung 2.5: Analyse der Übereinstimmung der IDs - ratings.csv

Die Abbildung 2.5 zeigt, dass in ratings.csv, dem großen Datensatz, von 2602489 Zeilen die IDs von 14287721 Zeilen nicht mit den Metadaten übereinstimmen.

```
from IPython.display import clear_output

not_in_meta_small = []
in_meta_small = []
a=1
length = str(len(ratings_small.movieId))

for i in ratings_small.movieId:
    clear_output(wait=True)
    print(str(a)+'/'+length)
    a+=1
    if not movies_metadata.loc[movies_metadata['id'] == str(i), 'original_title'].array:
        not_in_meta_small.append(i)
    else:
        in_meta_small.append(i)

print('ratings_small:')
print('Not in:', len(not_in_meta_small))
print('In:', len(in_meta_small))
```

```
100004/100004
ratings_small:
Not in: 55015
In: 44989
```

Abbildung 2.6: Analyse der Übereinstimmung der IDs - ratings_small.csv

In der Abbildung 2.6 ist zu erkennen, dass in ratings_small.csv, dem kleineren Datensatz, von 100004 Zeilen die IDs von 55015 Zeilen nicht in den Metadaten enthalten sind.

In dem nächsten Schritt wurde die Anzahl der voneinander verschiedenen Movie IDs der jeweiligen Datei berechnet, um bestimmen zu können, wie viele verschiedene Filme keine entsprechende Movie ID im Metadatensatz haben.

```
def get_missing_ids(list):
    list_count = collections.Counter(list)
    return sorted(list_count, key=list_count.get, reverse=True)
```

```
print('Small:')
print('Not in meta:', len(get_missing_ids(not_in_meta_small)))
print('In meta:', len(get_missing_ids(in_meta_small)))
print('Big:')
print('Not in meta:', len(get_missing_ids(not_in_meta)))
print('In meta:', len(get_missing_ids(in_meta)))
```

```
Small:
Not in meta: 6236
In meta: 2830
Big:
Not in meta: 37550
In meta: 7565
```

Abbildung 2.7: Analyse der Anzahl der IDs

Diese Analyse zeigte, dass in dem kleinen Datensatz von insgesamt 9066 IDs 6236 IDs nicht in den Metadaten enthalten sind. In dem großen Datensatz sind von insgesamt 45115 IDs 37550 IDs nicht in den Metadaten enthalten sind. Folglich ist der Datensatz nicht verwendbar für die Rating-based Recommendation. Daher wird ein neuer, verwandter Datensatz verwendet, welcher sich nur auf die Ratings bezieht.
Der Datensatz beinhaltet die Datei ratings.csv, welche die Bewertungen der Nutzer zu diversen Filme enthält und movies.csv, welche die Informationen zu den Filmen bereitstellt. Die gleiche Analyse wurde auch auf diesem Datensatz durchgeführt und sie ergab, dass keine Id von ratings.csv in movies.csv fehlt. Der neue Datensatz ist folglich verwendbar.

3 Content Based Recommendation

Als erstes wurden alle notwendigen Packages importiert.

```
import numpy as np
import pandas as pd
import sklearn
from ast import literal_eval
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import linear_kernel

import difflib
```

Abbildung 3.1: Packages Content Based Recommendation

Da für die Empfehlung besonders der Inhalt von Bedeutung ist, wurde das ursprüngliche Dataset benutzt. Dabei wurden die Dateien credits.csv, keywords.csv und movies_metadata. importiert:

```
#ratings = pd.read_csv (r'./../data/ratings.csv', low_memory=False)
#ratings_small = pd.read_csv (r'./../data/ratings_small.csv', low_memory=False)
credits = pd.read_csv (r'./../data/credits.csv', low_memory=False)
keywords = pd.read_csv (r'./../data/keywords.csv', low_memory=False)
#links = pd.read_csv (r'./../data/links.csv', low_memory=False)
#links_small = pd.read_csv (r'./../data/links_small.csv', low_memory=False)
metadata = pd.read_csv (r'./../data/movies_metadata.csv', low_memory=False)
```

Abbildung 3.2: Imports Content Based Recommendation

Mittels des Befehls

```
metadata = metadata[~metadata.id.str.contains("-")]
```

Abbildung 3.3: Datenvorbereitung Content Based Recommendation

werden die Daten bereinigt, dies ist notwendig, da ein paar Einträge in den Spalten verrutscht sind. Durch Testen konnte herausgefunden werden, dass in der ID-Spalte nun das Veröffentlichungsdatum im Format dd-mm-yyyy war. Deswegen konnte man diese Zeilen entfernen, indem man in der Spalte „id“ nach „-“ sucht und die jeweiligen Zeilen dann entfernt. Da das nur bei 3 Einträgen der Fall war, bleibt die Auswirkung auf den Datensatz gering.

Daraufhin können die Daten aus der Tabelle der keywords sowie aus der Tabelle der credits in die Metadatentabelle übertragen werden. Außerdem wird eine Spalte „features“ erstellt, welche alle Infos beinhaltet.

```
# Convert IDs to int. Required for merging
keywords['id'] = keywords['id'].astype('int')
credits['id'] = credits['id'].astype('int')
metadata['id'] = metadata['id'].astype('int')
```

```
# Merge keywords and credits into your main metadata dataframe
metadata = metadata.merge(credits, on='id')
metadata = metadata.merge(keywords, on='id')
```

```
# Parse the stringified features into their corresponding python objects
features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    metadata[feature] = metadata[feature].apply(literal_eval)
```

Abbildung 3.4: Datenbereinigung und Featuredefinition Content Based Recommendation

Weiterführend ist die Spalte „crew“ eine JSON-Datei, in welcher unter anderem die Daten über den jeweiligen Regisseur (director) enthalten sind. Daher wird eine neue Spalte „director“ in den Metadaten, durch Anwendung der **get_director** Funktion auf der „crew“-Spalte, erstellt
Mittels der Funktion **get_list** werden maximal drei Einträge der „cast“ (Schauspieler), „keywords“ und „genres“ gewählt, um die Datenmenge ein wenig zu verkleinern, was den Algorithmus schneller macht.

```python
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan

def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If no, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

```python
# Define new director, cast, genres and keywords features that are in a suitable form.
metadata['director'] = metadata['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    metadata[feature] = metadata[feature].apply(get_list)
```

Abbildung 3.5: Featurebearbeitung Content Based Recommendation

Im folgenden Schritt müssen alle Strings lower-case werden, sowie der jeweilige Vor- als auch der Nachname werden als ein Wort zusammengefasst. Dies geschieht, um später beim Erstellen der „CountVectorizer“-Matrix Schauspieler mit dem gleichen Vornamen unterscheiden zu können. Somit wird aus Tom Hanks „tomhanks“ und aus Tom Cruise „tomcruise“, was nun die beiden Schauspieler deutlich differenziert, während sie ursprünglich aufgrund des Vornamens „Tom“ vom CountVectorizer als sehr ähnlich klassifiziert worden wären.

```python
# Function to convert all strings to lower case and strip names of spaces
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''

# Apply clean_data function to your features.
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
    metadata[feature] = metadata[feature].apply(clean_data)
```

Abbildung 3.6: Datenbereinigung 2 Content Based Recommendation

Als letzter Schritt der Datenvorbereitung wird innerhalb der Metadaten eine weitere Spalte „soup“ definiert, welche für jeden Film eine Kette mit allen wichtigen Informationen (keywords, cast, director und genres), darstellt.

```
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])

# Create a new soup feature
metadata['soup'] = metadata.apply(create_soup, axis=1)

metadata[['soup']].head(2)
```

Abbildung 3.7: Datenbereinigung 3 Content Based Recommendation

Auf diese „soup“ wird nun der sogenannte CountVectorizer[2] angewandt. Dieser erstellt für jeden Film einen Vektor anhand der darin (soup) vorkommenden Worte. Dabei gehen die Einzelvektoren für „king“ und „man“ in eine ähnliche Richtung, während „princess“ und „girl“ in eine andere Richtung gehen, unter sich allerdings ebenfalls in eine ähnliche. Dabei wird außerdem beobachtet, welche Wörter wie oft vorkommen und Wörter, die sehr oft vorkommen erhalten eine geringere Gewichtung. Des Weiteren können „stop_words“ definiert werden. In diesem Fall sind das die Englischen. Sie bestehen aus Wörtern wie „and“, „or“ oder „to“, welche keinen Nutzen für die Erkennung des Inhalts des Textes liefern. Folgend wird eine „count_matrix“ anhand von der „soup“ erstellt.

```
# create count matrix with CountVectorizer
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(metadata['soup'])
count_matrix.shape
```

Abbildung 3.8: CountVectorizer Content Based Recommendation

In dem nächsten Schritt wird anhand der „count_matrix“ die Kosinus Ähnlichkeit („cosine similarity“)[3] berechnet. Das bedeutet es wird für jeden Film, zu jedem anderen Film aus dem Datensatz die Änhlichkeit anhand der Vektoren aus der „CountVectorizer“-Matrix, welche anhand des „soup“ erstellt wurde, bestimmt.

```
# Compute the Cosine Similarity matrix based on the count_matrix
cosine_sim = cosine_similarity(count_matrix, count_matrix)
```

Abbildung 3.9: cosine similarity Content Based Recommendation

Die Indexe des DataFrames werden zurückgesetzt und es wird eine Serie („pandas.Series“) erstellt, was ein eindimensionales ndarray ist, welche die Indexe beinhaltet, deswegen wird sie auch „indices“ genannt.

[2]Kulkarni und Shivananda 2019.
[3]Selva Prabhakaran 2018.

```
# Reset index of your main DataFrame and construct reverse mapping
metadata = metadata.reset_index()
indices = pd.Series(metadata.index, index=metadata['id'])
```

Abbildung 3.10: Indexbearbeitung Content Based Recommendation

Im Folgenden wird die Funktion **get_recommendations** definiert. Innerhalb der Funktion sind die sogenannten „sim_scores“, wobei eine Liste aufgezählt wird mit allen Ähnlichkeits-Scores der „cosine_similarity“. Diese werden daraufhin sortiert, um folglich die 10 ähnlichsten Filme vorzuschlagen.

```
def get_recommendations(id, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[id]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]
```

Abbildung 3.11: Recommendation Funktion Content Based Recommendation

Mittels der Funktion „input_title“ kann der jeweilige Benutzer einen Filmtitel eingeben. Als Beispiel wurde „double trouble“ ausgewählt. Die Liste „possible_titels“ beinhaltet die jeweiligen Titel, die zu dem input-Titel ähnlich sind. Die Liste wird auf maximal 5 Title begrenzt.

```
input_title = input("Your title: ")
```

```
Your title: double trouble
```

```
possible_titles = difflib.get_close_matches(input_title, metadata['original_title'].tolist(), 5)
print(possible_titles)
```

```
['Double Trouble', 'Double Trouble', 'Triple Trouble', 'Triple Trouble', 'Monkey Trouble']
```

Abbildung 3.12: Richtiger Film 1 Content Based Recommendation

Anschließend wählt der Benutzer den Film aus, zu welchem er die Recommendation erhalten will.

```
input_number = int(input("What's your film? Input 0 to 4: "))
chosen_title = possible_titles[input_number]
```

```
What's your film? Input 0 to 4: 0
```

Abbildung 3.13: Richtiger Film 2 Content Based Recommendation

Da mehrere Filme den gleichen Titel haben können, muss nun noch die ID des gewählten Filmes ermittelt werden.

```
chosen_id = metadata.loc[metadata['original_title'] == str(chosen_title), 'id'].array[0]
```

Abbildung 3.14: Richtiger Film 3 Content Based Recommendation

Die Recommendation läuft folgernd über den ausgewählten Film. Die vorgeschlagenen Filme sind sowohl vom Inhalt, Genres, Regisseur als auch von den Schauspielern her ähnlich.

```
get_recommendations(chosen_id, cosine_sim)
```

```
3505                          Blue Hawaii
8174     It Happened at the World's Fair
5961                Girls! Girls! Girls!
3507                           G.I. Blues
8176                              Spinout
9911          The Boys from County Clare
6184           Down & Out With The Dolls
9909                           Girl Crazy
18630                What's Up, Scarlet?
23438                Made For Each Other
```

Abbildung 3.15: Recommendation Content Based Recommendation

Literaturverzeichnis

Kulkarni, Akshay und Adarsha Shivananda (2019). *Converting Text to Features. In: Natural Language Processing Recipes*. Apress, Berkeley, CA. ISBN: 978-1-4842-4267-4.

pandas.DataFrame.corr - Documentation (2021). URL: `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html` (besucht am 13. 07. 2021).

Selva Prabhakaran (22. Okt. 2018). *Cosine Similarity – Understanding the math and how it works (with python codes)*. URL: `https://www.machinelearningplus.com/nlp/cosine-similarity/` (besucht am 03. 07. 2021).

A Anhang

A.1 Wie wird der Code verwendet?

Der Code ist auf https://github.com/adichter-hpe/data_exploration vorhanden. Die beiden Recommendation Systems können über das jeweils eigene Jupyter Notebook ausgeführt werden. Diese sind in ihren eigenen Ordnern zu finden. Außerdem befindet sich im Projekt der Anfang einer Flask-Webapp, welche allerdings aufgrund von technischen Schwierigkeiten (RAM-Limitierung des Docker-Containers) noch work-in-progress ist. Dennoch kann sie mit Hilfe von „docker-compose up“ in der Comandline innerhalb des Projekts gestartet werden und ist auf „localhost:5001“ zu finden.