Movie Recommendation System Using Linear Kernel and KNN

Garland Qiu CSC 44700 Section P

Movie Recommendation System Dataset and Build

For this movie recommendation system, the dataset used is the TMDB 5000 movie dataset, which contains the movies and its information such as the genres, overviews, taglines, etc., along with another TMDB 5000 dataset that contains the cast and crew of the movies. The prototype build for this movie recommendation system is a content-based filtering recommendation system. A content-based filtering recommendation system uses the contents or features of a movie, which are the columns in our dataset, to recommend a user a specific number of movies. What can be assumed is that the user will pick a movie to watch, and after watching the movie, that is where we can recommend a movie based on what they watched.

Cleaning and Formatting the TMDB 5000 Datasets for Use

Before the dataset can be used, there were a lot of cleaning that needed to be done. First thing to do was to merge the two datasets based on their movie ids. Their dataset had the same movies and ids, so we needed no further cleaning on the movie titles and its data. The next problem the dataset had was the JSON formatted data in specific columns, which are genres, keywords, production companies, production countries, spoken languages, cast, and crew. The data becomes very difficult for the learning algorithms to read in JSON format, so it was needed to convert those into strings that can be read by the algorithms. Another issue to deal with was the missing values for certain columns and movies. After checking what was missing, I decided that

those columns or features were not significant enough to use in the model and we can simply just fill those in as blanks.

Choosing the Features to Use in Recommending Movies

The next problem to tackle was to choose what specific features from the dataset to use to recommend the movies. When we think about recommending a movie, we would usually think of things such as a sequel or prequel to a movie, actors, and probably directors too in some cases. Of course, there are other factors such as genres and the descriptions of a movie that can come in play as well. For my KNN model, I would have the cast, director, genres, and keywords as the features, mainly due to how well they correlate when people usually recommend movies to each other. Due to how recommendation systems are as a concept, one of the challenges is figuring out what may or may not be a good recommendation when given a movie. However, the recommendation system should give a "reasonable" recommendation based on the features from the movie itself. Examples of "reasonable" recommendations would be prequels and sequels of a movie, or movies belonging to a "universal lore".

Some of the features from the dataset, such as budget, homepage, popularity, runtime, and production companies, might not make too much sense in the case of recommending movies. If we include these features, we can end up having some sort of bias when it comes to recommending movies, as the model may lean towards the heavy budget movies or the most popular movies. What we want to do is to recommend a movie no matter how popular or how much a budget may be, so long as it can be recommended due to its other features.

Another problem was deciding if I should take account on the vote averages and vote counts from each movie. One of the reasons I decided to not use it as features is mainly due to how we may have some bias arise as well, as such with the other features I decided not to use. Since I wanted to recommend movies that relate to each other in features, the movies with a lower rate average or count may not be considered since their vote averages are very weak. A lot of choosing which features to use and not use stems from how people would normally decide to watch another movie they previously have watched.

Determining Accuracy of Results in our Models

When determining the accuracy of the results found, the only realistic approach is to check the movies recommended manually and see how similar they are. This is mainly because we do not have any sort of data other than the dataset to rely on, so the models may recommend us something useful or not useful. The models would give us movies according to what they find to be the most similar in comparison, so some recommendations may look good in the learning process, but that does not necessarily mean it would be the perfect recommendation. As stated earlier, one of the ways to determine accuracy of results would be when we get prequels or sequels of a movie, or a movie belonging to a “universal lore”. This would let us know the models are at least going in some direction that we would like, rather than just recommending us random movies.

Using a Linear Kernel on Overviews to Recommend Movies

One model I made to try and recommend movies is a simple linear kernel model that uses the TMDB dataset's movie overviews. In short, a linear kernel is a learning algorithm that is a part of SVMs, or Support Vector Machines, that helps us with pattern analysis. What we want to do with the recommendation model is to use the overviews and for each word in the overview, it gets classified those words into a category, which we can pull out when we recommend the movies with similar overviews.

To efficiently use each word in the overview, we need a way of getting words with "significant values", or unique words such as "machine" and "learning", rather than getting words with "no significant values" or common words used, such as "in", "this", "the", etc. To do this, we use scikit-learn's built in function, TfidfVectorizer, which is mainly used to extract words and give weights on those words. The weight is a statistical measure on how important the word is to the description, or overview in this case, and the more a word appears, the more likely it is more significant, so it holds more weight. The TfidfVectorizer function also helps us in filtering out common words so that we get a more precise measure on the weight of the words in the overview. To use this function, the overview of every movie is taken and placed into the TfidfVectorizer function and the weights of the words are transformed into a matrix with each movie and weight on each word. The next step is to compute the linear kernel using the matrix and then using that linear kernel model to recommend movies based on their similarities in overviews.

Based on these results from Figure 6, this model is very weak when trying to recommend movies. Although the results from Figure 6 may show that it is very good when recommending movies, this model still only relies on the overview only, which may cause some sort of

“inaccurate” results. We can see that it may look good since it is recommending other Spiderman movies when Spider-Man 3 is chosen, but we start to see that other movies do not necessarily relate to the movie chosen, such as The Thing, which is a sci-fi horror movie, and would be a weird recommendation from a superhero movie. Another thing to consider when using the models are the plot on the movies. Not every movie will have the same plot, so some words are holding more weight for movies with prequels and sequels, however, this may not apply to every movie, as characters and plot do not necessarily carry over to the next or previous movies.

Using KNN with Features to Recommend Movies

Another model that can be used to recommend movies is a KNN model, or K-Nearest-Neighbors. KNN is another training algorithm that calculates the distances between objects and its features and uses that to determine similarities between each object and its features. This means that the larger the distance, the less similarities there are between two objects. We can imagine this in a graph with multiple points or objects, and to classify them, we line up two points randomly and calculate their distances. The closer they are, the more similar they are to each other. The purpose of KNN in the recommendation system is to find what features we can use to recommend the movies based on their similarities.

One of the issues in using KNN with this dataset is that some features are in text and not in numbers, which makes it difficult to use KNN in this case. Features, such as genre, cast, crew, and keywords, cannot really be measured as they are in text. One solution to this is to use these features and convert them into a list based on that feature. The list contains a unique word or

phrase from every movie in our dataset, and with that unique list, for each row, we can check to see if that feature has that unique word or phrase and store it into an array of 0s and 1s, where 0 represents that the movie does not have that unique word or phrase, and 1 represents that the movie does have that unique word or phrase. Figures 7 to 14 shows the unique values list from each feature and their binary lists. This method helps us in generating numbers that can be used for KNN model to find similarities between two movies.

The next step for our model is to calculate the distance between the features that we will be using, and to do this, we will be using cosine similarity as the similarity metric to recommend movies. The cosine similarity measures the cosine of the angle between two vectors, or in this case, our lists, and the smaller the angle, the higher the similarities. SciPy has a built-in function that we can use to calculate the cosine similarity as shown in Figure 15 and using the binary lists from those features alongside, we can get the distances between two movies and see if we can consider them good recommendations.

The last step is to build the recommendation function that takes in all these values from the dataset and recommends us movies sorting by their similarity distances starting from smallest distances.

Using Genre and Cast as Features

Using genres and cast as features for our model as shown in Figure 17, we can see from the results when given a random movie, the model tends to lead towards the same genres and the same cast. We can also see that some of these movies that alone with the same genre, there are a

few actors that play in more than one movie given, however, the genres are the main factors that keeps the similarity distances very low.

Using Genre and Director as Features

Using genres and director as features as shown in Figure 18, this model only relies on the genre to get its distances and is not using the directors as a factor as much. This is also because there are so few movies with the same director and the same genre. Most directors do not necessarily stick with the same genre in movie making, so this model would be very weak in terms of recommending movies.

Using Cast and Directors as Features

Based on the results for this model shown in Figure 19, it is quite good for people who may be interested in a director's or cast member's work in other movies, but when it comes to recommending movies for the content and story, this would not be the case. Many of the genre differs from each other, which is only reasonable as we did not include genres as a feature in this model.

Using Genre and Keywords as Features

Based on the results for this model shown in Figure 20, we can see that it does recommend other movies with the same genre, however, the movies in context do not necessarily relate to the other movies. From Figure 20, some movies that are straight horror films, such as Annabelle from the

model, should not really be in the same recommendations as Zombieland, which was more comedy with small bits of horror. This may be because the keywords provided was not enough for the model to realize that these do not necessarily relate to each other, other than the fact that they are horror films.

Using Cast and Keywords as Features

Based on the results shown in Figure 21, the recommendations given do not necessarily work with each other. Although we get small distances between the movie picked and the recommendations, they do not necessarily relate to each other when it comes to the context of the movies. We can also notice the distinct differences in genre, as some movies go from comedy to movies that are drama and thrillers. This can be used to recommend movies that have certain actors playing in it, but other than that, not much else.

Using Director and Keywords as Features

Based on the results from Figure 22, this model is not necessarily strong other than recommending movies with directors in it. There are not many similarities between the movies and the movie picked, and the similarity distances show that too, with distances greater than 1. Overall, a weak model that does not recommend much other than the directors for those movies.

Using Genres, Cast, Director, and Keyword as Features

Based on the results from Figure 23, this model comes the closest in recommending movies that closely relates to each other. From Figure 23, we can see that they relate a lot to each other, with some movies being prequels and sequels of the other movies. Because we put these 4 features into this model, we can see that we are getting more related movie recommendations, and generally, what we might see if we were to watch some of these movies and see their recommendations after watching. Although their similarity distances may be somewhat large, it is mainly due to how the similarity function calculates the distances by adding up the distances from each feature, so in perspective, these values are generally good when comparing the movies.

Conclusion/ Analysis

In conclusion, recommending a movie requires more features and even with more features, it can never be guaranteed that those movies would be the "perfect" recommendation and needs to be taken with a grain of salt. There was also no proper way of calculating any sort of accuracy, as we were simply outputting movies based on their features. One of the ways to "calculate" accuracy was to simply search up the movies and see if they have any similarities with each other. Another way to "calculate" accuracy was based on the movie's title, and if it is a prequel or sequel to the movie, or sharing a "universal lore", since that is one of the ways to know that we were getting the "accurate" results. The model using the four features did prove to be the most effective one, and adding anymore unnecessary features such as budget, homepage,

revenue, etc. would be unnecessary in recommending movies, as those values do not help in determining the right movies. One dataset I was hoping the TMDB dataset included or had was a list of users that voted and rated on certain movies instead of an average vote and vote count. With that, I would have liked to explore how that feature with users voting and rating on movies can improve on the recommendation system, but unfortunately, I was not able to find one. Overall, this was certainly an interesting topic in researching and making and it was nice to see the basics on prototyping a recommendation system, as there are many factors involved with developing one system.

Appendix

```
In [3]: # Checking if we're missing any null values intially
        tmdb5000_df1.isna().sum()
```

```
Out[3]: budget                     0
        genres                     0
        homepage                3091
        id                         0
        keywords                   0
        original_language          0
        original_title             0
        overview                   3
        popularity                 0
        production_companies       0
        production_countries       0
        release_date               1
        revenue                    0
        runtime                    2
        spoken_languages           0
        status                     0
        tagline                  844
        title                      0
        vote_average               0
        vote_count                 0
        dtype: int64
```

Figure 1: The missing values in the dataset by column.

```
# Based on the output, homepage, overview, release_date, runtime, and tagline have null values here.
# Homepage, release_date, and runtime will most likely not have any significant result in training and testing.
# We do want to fill in empty strings for the null values in our dataset.

tmdb5000_df1['homepage'] = tmdb5000_df1['homepage'].fillna('')
tmdb5000_df1['overview'] = tmdb5000_df1['overview'].fillna('')
tmdb5000_df1['release_date'] = tmdb5000_df1['release_date'].fillna('')
tmdb5000_df1['tagline'] = tmdb5000_df1['tagline'].fillna('')

# For the null runtimes, I will be using the runtimes' mean to fill.
tmdb5000_df1['runtime'] = tmdb5000_df1['runtime'].fillna(tmdb5000_df1['runtime'].mean())
```

Figure 2: Filling in the missing columns

```
# Looking at a random overview
df['overview'][np.random.choice(df.shape[0])]
```

```
"Set in 1920's New York City, this movie tells the story of idealistic young playwright David Shayne. Producer Julian Marx
finally finds funding for the project from gangster Nick Valenti. The catch is that Nick's girl friend Olive Neal gets the
part of a psychiatrist, and Olive is a bimbo who could never pass for a psychiatrist as well as being a dreadful actress. A
greeing to this first compromise is the first step to Broadway's complete seduction of David, who neglects longtime girl fr
iend Ellen. Meanwhile David puts up with Warner Purcell, the leading man who is a compulsive eater, Helen Sinclair, the gra
nd dame who wants her part jazzed up, and Cheech, Olive's interfering hitman / bodyguard. Eventually, the playwright must d
ecide whether art or life is more important."
```

Figure 3: A random movie's overview

```
# Keeping "important" words and removing common words.
tfidf = TfidfVectorizer(stop_words='english',analyzer='word')
tfidf_matrix = tfidf.fit_transform(df['overview'])
print(tfidf_matrix.shape)
tfidf_matrix
```

```
(4803, 20978)

<4803x20978 sparse matrix of type '<class 'numpy.float64'>'
        with 125840 stored elements in Compressed Sparse Row format>
```

Figure 4: Using TfidfVectorizer on each movie's overviews to get each significant words' weight.

```
similarity = linear_kernel(tfidf_matrix, tfidf_matrix)
print(similarity.shape)
similarity
```

```
(4803, 4803)

array([[1.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 1.        , 0.        , ..., 0.02160533, 0.        ,
        0.        ],
       [0.        , 0.        , 1.        , ..., 0.01488159, 0.        ,
        0.        ],
       ...,
       [0.        , 0.02160533, 0.01488159, ..., 1.        , 0.01609091,
        0.00701914],
       [0.        , 0.        , 0.        , ..., 0.01609091, 1.        ,
        0.01171696],
       [0.        , 0.        , 0.        , ..., 0.00701914, 0.01171696,
        1.        ]])
```

Figure 5: Using linear kernel on the matrix to classify each movie based on its overview.

```python
indices = pd.Series(df.index, index=df['original_title'])

def overview_recommendation(title, sim=similarity):
    idx = indices[title]
    similar = list(enumerate(sim[idx]))
    similar = sorted(similar, key=lambda x:x[1], reverse=True)
    similar = similar[1:11]
    m_idx = [i[0] for i in similar]
    return m_idx, idx


random_movie = df['original_title'][np.random.choice(df.shape[0])]
movie_index,idx=overview_recommendation(random_movie)
print("Movie Picked:", random_movie)
for i in movie_index:
    print(df['original_title'].iloc[i],'(content match rate):',similarity[idx][i])
```

```
Movie Picked: Spider-Man 3
Spider-Man (content match rate): 0.28269377551691705
Spider-Man 2 (content match rate): 0.2464720918482627
Arachnophobia (content match rate): 0.23671046045214342
The Amazing Spider-Man (content match rate): 0.22284657494531743
The Amazing Spider-Man 2 (content match rate): 0.21139629084201764
The Thing (content match rate): 0.17166864148468533
Bronson (content match rate): 0.10119023772910762
Not Easily Broken (content match rate): 0.09698331013627293
Raising Victor Vargas (content match rate): 0.0944122894618133
Def-Con 4 (content match rate): 0.09420458163538704
```

Figure 6: Recommendation function and recommended movies based on overview similarities.

```
# Checking the unique genres in the list
genre_list = []
for index, row in df.iterrows():
    genres = row["genres"]

    for genre in genres:
        if genre not in genre_list:
            genre_list.append(genre)

genre_list
```

```
['Action',
 'Adventure',
 'Fantasy',
 'ScienceFiction',
 'Crime',
 'Drama',
 'Thriller',
 'Animation',
 'Family',
 'Western',
 'Comedy',
 'Romance',
 'Horror',
 'Mystery',
 'History',
 'War',
 'Music',
 'Documentary',
 'Foreign',
 'TVMovie',
 '']
```

Figure 7: List of unique genres from every movie taken from the dataset.

```
df['cast']
```

```
0       CCHPounder,GiovanniRibisi,JoelDavidMoore,LazAl...
1       BillNighy,ChowYun-fat,GeoffreyRush,JackDavenpo...
2       AndrewScott,BenWhishaw,ChristophWaltz,DanielCr...
3       AnneHathaway,ChristianBale,CillianMurphy,GaryO...
4       BryanCranston,CiaránHinds,DominicWest,JamesPur...
                              ...
4798    CarlosGallardo,ConsueloGómez,JaimedeHoyos,Juan...
4799    CaitlinFitzgerald,DaniellaPineda,EdwardBurns,K...
4800    BenjaminHollingsworth,CrystalLowe,DaphneZuniga...
4801    AlanRuck,BillPaxton,DanielHenney,ElizaCoupe,Zh...
4802    BillDElia,BrianHerzlinger,CoreyFeldman,DrewBar...
Name: cast, Length: 4803, dtype: object
```

Figure 8: List of Actors from each movie.

```
# Getting unique directors
director_list = []
for i in df['crew']:
    if i not in director_list:
        director_list.append(i)

director_list
```

```
['James Cameron',
 'Gore Verbinski',
 'Sam Mendes',
 'Christopher Nolan',
 'Andrew Stanton',
 'Sam Raimi',
 'Byron Howard',
 'Joss Whedon',
 'David Yates',
 'Zack Snyder',
 'Bryan Singer',
 'Marc Forster',
 'Andrew Adamson',
 'Rob Marshall',
 'Barry Sonnenfeld',
 'Peter Jackson',
 'Marc Webb',
 'Ridley Scott',
 'Chris Weitz',
```

Figure 9: List of unique Directors extracted from cast column.

```
# All the unique keywords
keyword_list
```

```
['3d',
 'alien',
 'alienplanet',
 'antiwar',
 'battle',
 'cgi',
 'cultureclash',
 'future',
 'futuristic',
 'loveaffair',
 'marine',
 'mindandsoul',
 'powerrelations',
 'romance',
 'society',
 'soldier',
 'space',
 'spacecolony',
 'spacetravel',
```

Figure 10: List of unique keywords from all the movies in the dataset.

```python
# Forming a binary list of the genres that the movie contains to later use for our KNN model.
def binary_genre_lst(gl):
    lst = []

    for genre in genre_list:
        if genre in gl:
            lst.append(1)
        else:
            lst.append(0)

    return lst

# We can add this to our dataset
df['binary_genres'] = df['genres'].apply(lambda x: binary_genre_lst(x))
df['binary_genres']
```

```
0       [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2       [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3       [1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...
4       [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
                              ...
4798    [1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
4799    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, ...
4800    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, ...
4801    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4802    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: binary_genres, Length: 4803, dtype: object
```

Figure 11: Binary list of genres that are in every movie.

```python
# Forming a binary list of the cast that the movie contains to later use for our KNN model.
def binary_cast_lst(cl):
    lst = []
    for c in cast_list:
        if c in cl:
            lst.append(1)
        else:
            lst.append(0)

    return lst

# We can add this to our dataset
df['binary_cast'] = df['cast'].apply(lambda x: binary_cast_lst(x))
df['binary_cast']
```

```
0       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1       [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
2       [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, ...
3       [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, ...
                              ...
4798    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, ...
4799    [1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, ...
4800    [1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, ...
4801    [1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, ...
4802    [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, ...
Name: binary_cast, Length: 4803, dtype: object
```

Figure 12: Binary list of casts that are in every movie.

```
# Forming a binary list of the cast that the movie contains to later use for our KNN model.
def binary_directors_lst(dl):
    lst = []
    for d in director_list:
        if d in dl:
            lst.append(1)
        else:
            lst.append(0)
    return lst

# We can add this to our dataset
df['binary_directors'] = df['crew'].apply(lambda x: binary_directors_lst(x))
df['binary_directors']
```

```
0       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
                              ...
4798    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4799    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4800    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4801    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4802    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: binary_directors, Length: 4803, dtype: object
```

Figure 13: Binary list of directors that are in every movie.

```
# Putting into list and then converting to binary based on list
keyword_list = []
for index, row in df.iterrows():
    keyword = row["keywords"]

    for k in keyword:
        if k not in keyword_list:
            keyword_list.append(k)


def binary_keywords_lst(kw):
    lst = []
    for k in keyword_list:
        if k in kw:
            lst.append(1)
        else:
            lst.append(0)
    return lst


# We can add this to our dataset
df['binary_keywords'] = df['keywords'].apply(lambda x: binary_keywords_lst(x))
df['binary_keywords']
```

```
0       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4       [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
                              ...
4798    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4799    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4800    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4801    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4802    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: binary_keywords, Length: 4803, dtype: object
```

Figure 14: Binary list of keywords in every movie.

```
# Calculate the distance between two points using cosine similarity
# The more the distance, the less similarity they have
def genre_director_cosine_similarity(id1, id2):
    a = df.iloc[id1]
    b = df.iloc[id2]

    genres_a = a['binary_genres']
    genres_b = b['binary_genres']

    genresDistance = spatial.distance.cosine(genres_a, genres_b)

    director_a = a['binary_directors']
    director_b = b['binary_directors']
    directorDistance = spatial.distance.cosine(director_a, director_b)

    return genresDistance + directorDistance
```

Figure 15: Cosine Similarity function. The names of the features are changed accordingly.

```
# Recommend function
def recommend(name):
    # Match movie with a name similar to it from the dataset
    match_movie = df[df['original_title'].str.contains(name)].iloc[0].to_frame().T
    print('Selected Movie: {} \n'.format(match_movie.original_title.values[0]))
    def getNeighbors(original_movie, k):
        distances = []

        for index, movie in df.iterrows():
            if movie['index'] != original_movie['index'].values[0]:
                sim = gcdk_cosine_similarity(original_movie['index'].values[0], movie['index'])
                distances.append((movie['index'], sim))

        distances.sort(key=operator.itemgetter(1))
        # You can see the distances of all the movies comparing with the parameter below.
        #print("Distances:",distances)
        neighbors = []

        for x in range(k):
            neighbors.append(distances[x])
            print("Distance:",distances[x])
        return neighbors

    k = 10
    neighbors = getNeighbors(match_movie, k)

    print('\nRecommended Movies:\n')
    for neighbor in neighbors:
        print("Movie Name: {}\n Average Rating: {} \t Genre: {}".format(df.iloc[neighbor[0]][7],
                                                                       df.iloc[neighbor[0]][18],
                                                                       df.iloc[neighbor[0]][2]))
        print('\n')

    print('----------------------------------------------------------------------------------------------------------------------\n')


# Test recommend by providinig a random movie
# Looping to see more recommendations
for i in range(2):
    pick_movie = np.random.choice(df.original_title.sample())
    recommend(pick_movie)
```

Figure 16: Recommend function that sorts the cosine similarity values and recommend the movies based on those values.

```
Selected Movie: Certifiably Jonathan

Distance: (3546, 0.1760935250995478)
Distance: (616, 0.20145059049530956)
Distance: (2319, 0.20145059049530956)
Distance: (2090, 0.21328160616263325)
Distance: (1966, 0.21354381941320477)
Distance: (2231, 0.2255969073376336)
Distance: (639, 0.22592973018679008)
Distance: (1147, 0.22592973018679008)
Distance: (1173, 0.22592973018679008)
Distance: (2989, 0.22592973018679008)

Recommended Movies:

Movie Name: The Ten
 Average Rating: 4.9     Genre: ['Comedy']


Movie Name: Ted 2
 Average Rating: 6.2     Genre: ['Comedy']


Movie Name: Deuce Bigalow: Male Gigolo
 Average Rating: 5.5     Genre: ['Comedy']


Movie Name: Bringing Down the House
 Average Rating: 5.4     Genre: ['Comedy']


Movie Name: Old School
 Average Rating: 6.6     Genre: ['Comedy']


Movie Name: Strange Wilderness
 Average Rating: 4.7     Genre: ['Comedy']


Movie Name: Step Brothers
 Average Rating: 6.5     Genre: ['Comedy']


Movie Name: The Big Year
 Average Rating: 5.6     Genre: ['Comedy']


Movie Name: The Intern
 Average Rating: 7.1     Genre: ['Comedy']


Movie Name: Happy Gilmore
 Average Rating: 6.5     Genre: ['Comedy']
```

Figure 17: Using genre and cast as features to recommend movies.

```
Selected Movie: Sex and the City 2

Distance: (3977, 0.2928932188134524)
Distance: (4247, 0.2928932188134524)
Distance: (226, 0.5)
Distance: (438, 0.5)
Distance: (440, 0.5)
Distance: (681, 0.5)
Distance: (795, 0.5)
Distance: (809, 0.5)
Distance: (815, 0.5)
Distance: (822, 0.5)

Recommended Movies:

Movie Name: Boynton Beach Club
 Average Rating: 6.8     Genre: ['Comedy', 'Drama', 'Romance']


Movie Name: Me You and Five Bucks
 Average Rating: 10.0    Genre: ['Romance', 'Comedy', 'Drama']


Movie Name: How Do You Know
 Average Rating: 4.9     Genre: ['Comedy', 'Drama', 'Romance']


Movie Name: Something's Gotta Give
 Average Rating: 6.3     Genre: ['Drama', 'Comedy', 'Romance']


Movie Name: Four Christmases
 Average Rating: 5.3     Genre: ['Comedy', 'Romance', 'Drama']


Movie Name: The American President
 Average Rating: 6.5     Genre: ['Comedy', 'Drama', 'Romance']


Movie Name: Leatherheads
 Average Rating: 5.7     Genre: ['Comedy', 'Romance', 'Drama']


Movie Name: Forrest Gump
 Average Rating: 8.2     Genre: ['Comedy', 'Drama', 'Romance']


Movie Name: Hitch
 Average Rating: 6.4     Genre: ['Comedy', 'Drama', 'Romance']


Movie Name: Maid in Manhattan
 Average Rating: 5.6     Genre: ['Comedy', 'Drama', 'Romance']
```

Figure 18: Using genre and director as features to recommend movies.

```
Selected Movie: Waterworld

Distance: (1193, 0.20277289626738504)
Distance: (2142, 0.21067023706540877)
Distance: (2261, 0.2716043165649188)
Distance: (901, 0.28089879059716916)
Distance: (1483, 0.6319077068705881)
Distance: (1183, 0.6427491427487142)
Distance: (686, 0.6437190367316358)
Distance: (1236, 0.6499525629935172)
Distance: (2207, 0.6499525629935172)
Distance: (382, 0.6550610072153058)

Recommended Movies:

Movie Name: The Count of Monte Cristo
 Average Rating: 7.3     Genre: ['Action', 'Adventure', 'Drama', 'Thriller']


Movie Name: Risen
 Average Rating: 5.7     Genre: ['Action']


Movie Name: Rapa Nui
 Average Rating: 6.2     Genre: ['Adventure']


Movie Name: Robin Hood: Prince of Thieves
 Average Rating: 6.6     Genre: ['Adventure']


Movie Name: Step Up Revolution
 Average Rating: 6.7     Genre: ['Music', 'Drama', 'Romance']


Movie Name: The Mexican
 Average Rating: 5.8     Genre: ['Action', 'Comedy', 'Crime', 'Romance']


Movie Name: Hop
 Average Rating: 5.5     Genre: ['Animation', 'Comedy', 'Family']


Movie Name: Bless the Child
 Average Rating: 4.9     Genre: ['Drama', 'Horror', 'Thriller', 'Crime']


Movie Name: 12 Rounds
 Average Rating: 5.7     Genre: ['Action', 'Adventure', 'Drama', 'Thriller']


Movie Name: Seabiscuit
 Average Rating: 6.7     Genre: ['Drama', 'History']
```

Figure 19: Using cast and director as features to recommend movies.

```
Selected Movie: Bride of Chucky

Distance: (1693, 0.6795507883933044)
Distance: (3418, 0.7583707349886037)
Distance: (2749, 0.9330532904861591)
Distance: (1430, 1.0)
Distance: (1534, 1.0)
Distance: (1648, 1.0)
Distance: (1981, 1.0)
Distance: (1988, 1.0)
Distance: (2132, 1.0)
Distance: (2477, 1.0)

Recommended Movies:

Movie Name: Seed of Chucky
 Average Rating: 4.9     Genre: ['Drama', 'Horror', 'Comedy']


Movie Name: Annabelle
 Average Rating: 5.6     Genre: ['Horror']


Movie Name: Child's Play 2
 Average Rating: 5.8     Genre: ['Drama', 'Horror']


Movie Name: Cursed
 Average Rating: 5.1     Genre: ['Horror', 'Comedy']


Movie Name: Arachnophobia
 Average Rating: 6.2     Genre: ['Comedy', 'Horror']


Movie Name: Fright Night
 Average Rating: 6.0     Genre: ['Horror', 'Comedy']


Movie Name: Piranha 3D
 Average Rating: 5.3     Genre: ['Comedy', 'Horror']


Movie Name: Zombieland
 Average Rating: 7.2     Genre: ['Comedy', 'Horror']


Movie Name: Vampires Suck
 Average Rating: 4.2     Genre: ['Horror', 'Comedy']


Movie Name: Jennifer's Body
 Average Rating: 5.3     Genre: ['Comedy', 'Horror']
```

Figure 20: Using genre and keywords as features to recommend movies.

```
Selected Movie: Snow Flower and the Secret Fan

Distance: (1593, 0.5235714001926229)
Distance: (1733, 0.5372835669786271)
Distance: (975, 0.5569131466194652)
Distance: (3037, 0.5626424885623571)
Distance: (588, 0.5943925203124046)
Distance: (2528, 0.6158900184271223)
Distance: (2738, 0.6244692960364916)
Distance: (1268, 0.6261425520624524)
Distance: (345, 0.6366432188134524)
Distance: (2367, 0.6366432188134524)

Recommended Movies:

Movie Name: The Three Stooges
 Average Rating: 4.9     Genre: ['Comedy']


Movie Name: The Spy Next Door
 Average Rating: 5.5     Genre: ['Action', 'Comedy', 'Family']


Movie Name: The International
 Average Rating: 6.0     Genre: ['Drama', 'Thriller', 'Crime']


Movie Name: The Goods: Live Hard, Sell Hard
 Average Rating: 5.4     Genre: ['Comedy']


Movie Name: Wall Street: Money Never Sleeps
 Average Rating: 5.8     Genre: ['Drama', 'Crime']


Movie Name: Jackass Presents: Bad Grandpa
 Average Rating: 6.0     Genre: ['Comedy']


Movie Name: About Last Night
 Average Rating: 6.0     Genre: ['Comedy', 'Romance']


Movie Name: Soul Men
 Average Rating: 6.3     Genre: ['Comedy', 'Music']


Movie Name: Rush Hour 2
 Average Rating: 6.4     Genre: ['Action', 'Comedy', 'Crime', 'Thriller']


Movie Name: The Last Station
 Average Rating: 6.7     Genre: ['Drama', 'Romance']
```

Figure 21: Using cast and keyword as features to recommend movies.

```
Selected Movie: And When Did You Last See Your Father?

Distance: (2299, 1.0)
Distance: (3314, 1.0)
Distance: (1338, 1.2142857142857144)
Distance: (2539, 1.2327387580875757)
Distance: (3632, 1.2327387580875757)
Distance: (4505, 1.2327387580875757)
Distance: (4610, 1.2480236846605153)
Distance: (725, 1.272078847080724)
Distance: (4511, 1.272078847080724)
Distance: (954, 1.2817821097640076)

Recommended Movies:

Movie Name: Leap Year
 Average Rating: 6.4     Genre: ['Romance', 'Comedy']


Movie Name: Shopgirl
 Average Rating: 5.7     Genre: ['Comedy', 'Drama', 'Romance']


Movie Name: John Q
 Average Rating: 7.0     Genre: ['Drama', 'Thriller', 'Crime']


Movie Name: Snitch
 Average Rating: 5.8     Genre: ['Thriller', 'Drama']


Movie Name: 90 Minutes in Heaven
 Average Rating: 5.4     Genre: ['Drama']


Movie Name: Médecin de campagne
 Average Rating: 6.0     Genre: ['Drama', 'Comedy']


Movie Name: Pieces of April
 Average Rating: 6.4     Genre: ['Comedy', 'Drama']


Movie Name: The Shaggy Dog
 Average Rating: 4.5     Genre: ['Comedy', 'Family']


Movie Name: Fireproof
 Average Rating: 7.0     Genre: ['Drama']


Movie Name: The Judge
 Average Rating: 7.2     Genre: ['Drama']
```

Figure 22: Using directors and keywords as features to recommend movies.

```
Selected Movie: Final Destination 5

Distance: (1789, 1.4122848723742258)
Distance: (2673, 1.5856275943960212)
Distance: (1186, 1.6231954119378793)
Distance: (1164, 1.6830175537624341)
Distance: (1961, 1.7054048946907732)
Distance: (3835, 1.715929797867981)
Distance: (1219, 1.7166505481993597)
Distance: (2120, 1.7168376746502587)
Distance: (947, 1.7282563668587103)
Distance: (3601, 1.774676433517926)

Recommended Movies:

Movie Name: Final Destination 2
 Average Rating: 5.9     Genre: ['Horror', 'Mystery']


Movie Name: My Bloody Valentine
 Average Rating: 5.3     Genre: ['Mystery', 'Horror']


Movie Name: The Final Destination
 Average Rating: 5.4     Genre: ['Horror', 'Mystery']


Movie Name: Scream 3
 Average Rating: 5.7     Genre: ['Horror', 'Mystery']


Movie Name: Scream 2
 Average Rating: 6.1     Genre: ['Horror', 'Mystery']


Movie Name: The Witch
 Average Rating: 6.3     Genre: ['Mystery', 'Horror']


Movie Name: Scream 4
 Average Rating: 6.1     Genre: ['Horror', 'Mystery']


Movie Name: Orphan
 Average Rating: 6.7     Genre: ['Horror', 'Thriller', 'Mystery']


Movie Name: Silent Hill
 Average Rating: 6.3     Genre: ['Horror', 'Mystery']


Movie Name: April Fool's Day
 Average Rating: 5.8     Genre: ['Horror', 'Mystery']
```

Figure 23: Using genre, cast, director, and keyword as features to recommend movies.

Sources

- https://medium.com/analytics-vidhya/distances-in-machine-learning-ec8c6cebbc7f
- https://towardsdatascience.com/importance-of-distance-metrics-in-machine-learning-modelling-e51395ffe60d
- https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db
- https://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html
- https://www.machinelearningplus.com/nlp/cosine-similarity/
- https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system#Collaborative-Filtering
- https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a
- https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea
- https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c
- https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://towardsdatascience.com/beginners-recommendation-systems-with-python-ee1b08d2efb6
- https://heartbeat.fritz.ai/recommender-systems-with-python-part-ii-collaborative-filtering-k-nearest-neighbors-algorithm-c8dcd5fd89b2#:~:text=When%20a%20KNN%20makes%20a,the%20most%20similar%20movie%20recommendations
- https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761