IMDB Movie Clustering and Recommender System

by M. Boyd-Vasiliou, A. Weber, Z. Zaidi

Background

Our employer has acquired access to a full set of movies to stream to new customers. However, our customers have no watching history that we can build upon to provide recommendations, so we have to start with suggesting movies by similarity to each other. To do this we will use machine learning to cluster the movies along available details, including actors, genre, director and ratings. We will create a simple interface so customers can select a movie that they're interested in, then be shown further recommendations to choose from.

Proof-of-concept

Key Metrics and Functionality

Performance
- Search must be able to generate at least 10 appropriately related movies
- Appropriateness will be measured by application 'stickiness', that customers perform at least one click-through more than 50% of the time

Data Capture
- System must log its inputs and click-throughs, for performance analysis and building customer-specific data for modeling

Privacy
- Logged customer data must be stored and analyzed anonymously, and limited in corporate visibility to the scope of providing future lookups

Data Overview

The IMDB dataset contains information on over 85,000 movies, from 1894 to the present, including cast information and rating scores. The data is obtained from here: https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset.

The data is broken down into four files:

IMDb movies.csv

- All of the movies with summary data, including cast and ratings, one record per movie
- 85,855 movie records with 22 different features

IMDb title_credits.csv

- Cast and primary crew, by movie, one row per person. Individuals are identified by a name key
- 835,513 credit records with 6 different features

IMDb names.csv

- Detail of individuals by name key, including full name, height, birthdate, short bio [* First four names: Fred Astaire, Lauren Bacall, Bridget Bardot, John Belushi]
- 297,705 name records with 17 different features

IMDb ratings.csv

- Summary of customer movie ratings by movie, one row per movie, with ratings breakdown by demographics (gender, age)
- 85,855 ratings records (one per movie) with 49 different features

We used the first two files, which we will refer to further as “Movies” and “Principals”.

Data Exploration

Data dictionary “Movies”

Column Name	Short Description	Notes for our application
imdb_title_id	Unique Movie ID	
title	Name of Movie	Sometimes a translation, dropped
original_title	Name of Movie	A few duplicates, to be expected with remakes
year of release	year movie released	One bad record with extra text, fixed. All converted to numeric, and used for grouping
date of release	Actual release date	As above, but to day and month. Not used.
genre	Classification into by category	25 categories in all (Action, Adventure, Comedy, etc) Movies may be in more than one, for example Musical and Comedy
duration	Running time in minutes	Not currently used for grouping, used for display
country	Country where movie produced	Vast majority of titles are from USA, so poor assistance for grouping. Field retained for filtering and display
language	Language of movie	Sometimes multiple. Field used for filtering and display
director	Director(s) of the movie	Usually Single name, excellent usage for grouping movies. Swapped in actual Name Keys from Principals, to avoid possible spelling/formatting anomalies
writer	Writer(s)	Single or multiple. Not used for grouping but available for display

production_company	Movie studio	Not used for grouping but available for display
actors	List of primary actors	Multiple field, excellent for grouping. We used the top 2, as ranked in the Principals file, and swapped in actual Name Keys, to avoid possible spelling/formatting anomalies
description	Short synopsis of the story	Not currently used for grouping, future possibility of using language processing here.
avg_vote	User simple rating, range 1 - 9.9	Used for filtering (recommend higher rated movies), not used for grouping
votes	Number of user votes for the above score	Could be used to provide a confidence measure in above, but that rating is not used for grouping. Available for display
budget	Movie budget	Possible usage for grouping, but most movies (70%+) have no value here, and values not adjusted for inflation. Available for display
usa_gross_income	Box office, USA	As above
worlwide_gross_income	Box office, world	As above
metascore	IMDB blended score based on actual user and critic reviews (as distinguished from simple user scores used for avg_vote)	Possible usage for grouping, but most movies (84%) have no value here, so not used for grouping. Available for display
reviews_from_users	Number of user reviews	Could be used to provide a confidence measure in above, but that rating is not used for grouping. Available for display
reviews_from_critics	Number of critic reviews	As above

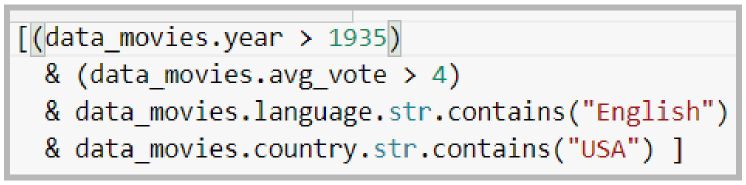
Dataset “Principals”

We use this file to obtain unique ID’s for actors and directors

Column Name	Short Description	Notes for our application
imdb_title_id	Unique Movie ID	Link (key) to Movies dataset
ordering	Rank in cast	Allows us to choose actor by importance of role, we take the first two
imdb-name_id	Unique Name Key, link to Names file with individuals details	We use this identifier instead of given names for the grouping algorithm, to avoid textual aberrations
category	type of work on the movie	This field allows us to select actor and director roles
job	Finer description of work above	Not used
characters	Names of characters as played by actors	Not used

Data Preprocessing

We filtered the dataset to make a more manageable set, and more relevant for our customers



```
[(data_movies.year > 1935)
  & (data_movies.avg_vote > 4)
  & data_movies.language.str.contains("English")
  & data_movies.country.str.contains("USA") ]
```

- Movies prior to 1936 not included. 1937 brings Technicolor and “Robin Hood” to the world, so that was our cutoff
- Movies are all in English. This was to aid us in building and testing the recommendation model. For North America there would be no reason not to include Spanish, French, Japanese, Cantonese and so on moving forward.
- Movies are American, again this narrows the range of actors and directors to help us observe the clustering with a relatively small dataset, as the builds and plotting/analysis operations take substantially longer than with classification. This is exacerbated when we normalize the dataset below with regards to actors and genres.
- Movies Dataset now contains ~22k rows.

Data Normalization

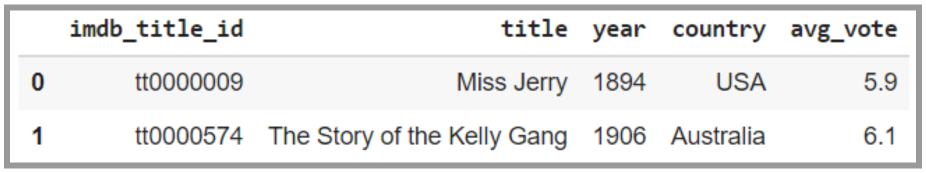
1. Actors and Directors

   The first thing we did was we used the Principals dataset to replace actor and director names with unique keys. This helped us avoid any possible issues with multiple spellings, or other data entry, and also high probability of simple duplicate names.

   The more important decision came around the issue of two actors being identified with a movie, let's call them star and co-star. Since they're both actors we want the program to cluster by either name, ie. if Nicolas Cage is the star of one movie and co-star in another, they're both Nicolas Cage movies. If we keep them in two fields then they only associate with other movies when Nicolas Cage has the same role level (star, co-star) as he does in this one. If you combine both actors into one column then it only matches when he's co-starring with the same person, in the same order. If you want to cross-match on two columns you need two rows to do that anyhow, so it's better to simply have one actor per movie record. Therefore, any movie with a listed co-star will now have two movie records (we only take the first two actors, by ranking in the Principals file). This has implications for clustering, which we'll explain further below.

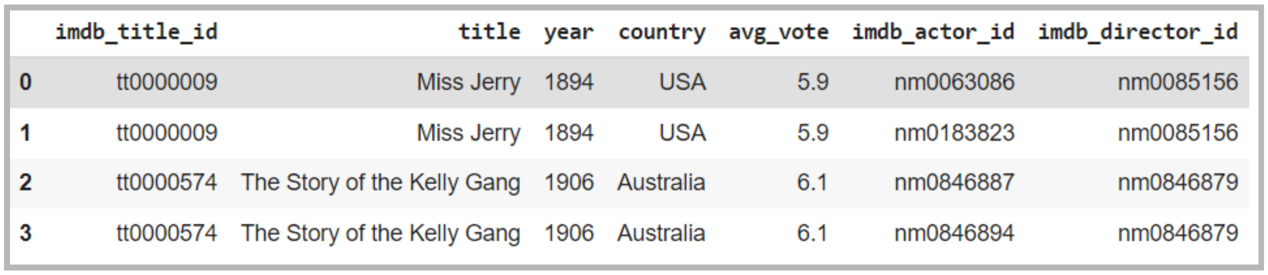
   Replacing Names and Normalizing Actors

Before:



	imdb_title_id	title	year	country	avg_vote
0	tt0000009	Miss Jerry	1894	USA	5.9
1	tt0000574	The Story of the Kelly Gang	1906	Australia	6.1

After:



	imdb_title_id	title	year	country	avg_vote	imdb_actor_id	imdb_director_id
0	tt0000009	Miss Jerry	1894	USA	5.9	nm0063086	nm0085156
1	tt0000009	Miss Jerry	1894	USA	5.9	nm0183823	nm0085156
2	tt0000574	The Story of the Kelly Gang	1906	Australia	6.1	nm0846887	nm0846879
3	tt0000574	The Story of the Kelly Gang	1906	Australia	6.1	nm0846894	nm0846879

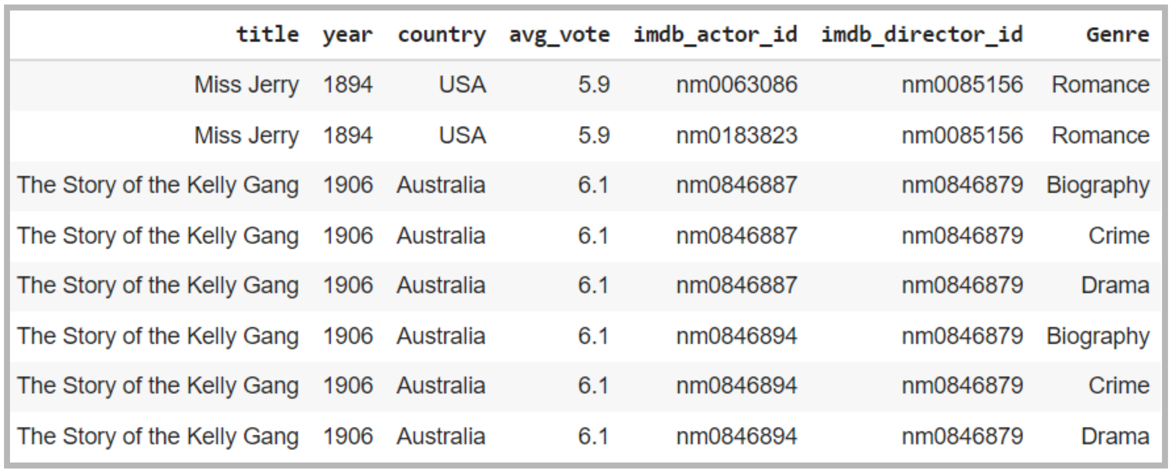
2. Genres

“Genres” has the same normalization problem that we have with actors, that we want to group movies by genre, and a movie may be indicated with multiple genres. However, the list of genres is reasonably short, so the first thing we did was try one-hot encoding. Here is the list of all genres, counted in all occurrences with the full 85k dataset:

```
Genre
Action         12948
Adult              2
Adventure       7590
Animation       2141
Biography       2377
Comedy         29368
Crime          11067
Documentary        2
Drama          47110
Family          3962
Fantasy         3812
Film-Noir        663
History         2296
Horror          9557
Music           1689
Musical         2041
Mystery         5225
News               1
Reality-TV         3
Romance        14128
Sci-Fi          3608
Sport           1064
Thriller       11388
War             2242
Western         1583
```

We were able to roll these up to present all genre tags in a single record, with a column per genre (we also combined some rows for balance and natural relation, Music and Musical, Sci-Fi and Fantasy). This model actually “clustered better” by plots and metrics than the one we chose, but this exercise brought us to another understanding for our use-case: clustering is a two-edged sword. It’s trying to group things together, while at the same time it’s trying to find boundaries and separate these groups from each other. If we search on Raiders of the Lost Ark, should we find ourselves amongst adventure movies, or Harrison Ford movies, or Steven Spielberg movies? And the converse is true, do we want to be recommended to Raiders of the Lost Ark by association with the genre, actor or director? The right answer is we want to find it all three ways, and the way to do this is to have the movie land in multiple clusters, *which is what happens when we enter it with multiple records*. What looked like a bug - how is it going to cluster properly with several entries for each movie? - is in fact the feature that enables a multi-variant lookup, which is what our customers expect to see.

This is what the first two movies look like when fully expanded by normalization on actor and genre as described above:



title	year	country	avg_vote	imdb_actor_id	imdb_director_id	Genre
Miss Jerry	1894	USA	5.9	nm0063086	nm0085156	Romance
Miss Jerry	1894	USA	5.9	nm0183823	nm0085156	Romance
The Story of the Kelly Gang	1906	Australia	6.1	nm0846887	nm0846879	Biography
The Story of the Kelly Gang	1906	Australia	6.1	nm0846887	nm0846879	Crime
The Story of the Kelly Gang	1906	Australia	6.1	nm0846887	nm0846879	Drama
The Story of the Kelly Gang	1906	Australia	6.1	nm0846894	nm0846879	Biography
The Story of the Kelly Gang	1906	Australia	6.1	nm0846894	nm0846879	Crime
The Story of the Kelly Gang	1906	Australia	6.1	nm0846894	nm0846879	Drama

3. Year and Avg Vote

   Year looked like an attractive field to include in the clustering data, you'd select a movie from a certain period and see several come back from the same era. However, this would be a second-order sort of criteria, and we were not able to prevent the clustering algorithm from prioritizing it far too heavily. In addition, directors and actors lend a certain period of activity range to the search, so an era is broadly captured that way without explicitly selecting for it. Average vote likewise tended to overwhelm the actual movie criteria, which is what our search lookup is about. A different search could simply look for highest-ranked movies. Also, if a search is made on a movie that happens to have low scores, it seems counterproductive to return more lousy movies. We obviously want our customers to be pleased, so we're only going to return a highly ranked set to choose from.

4. Final Paring Down

   With the data normalizations above, our 22k record set from the initial filtering ballooned to over 100k, which would be impossible to process at all in Colab (memory), even if one were able to wait it out. As it was, we then did a random sample of the filtered set, 20% which worked out to 5528 movies. After normalization, that became 33,899 rows to feed into the clustering model. The order of operations is important here, that we pared the set down before normalization, rather than taking a random sample of the normalized data, which would have returned incomplete record sets from more movies.

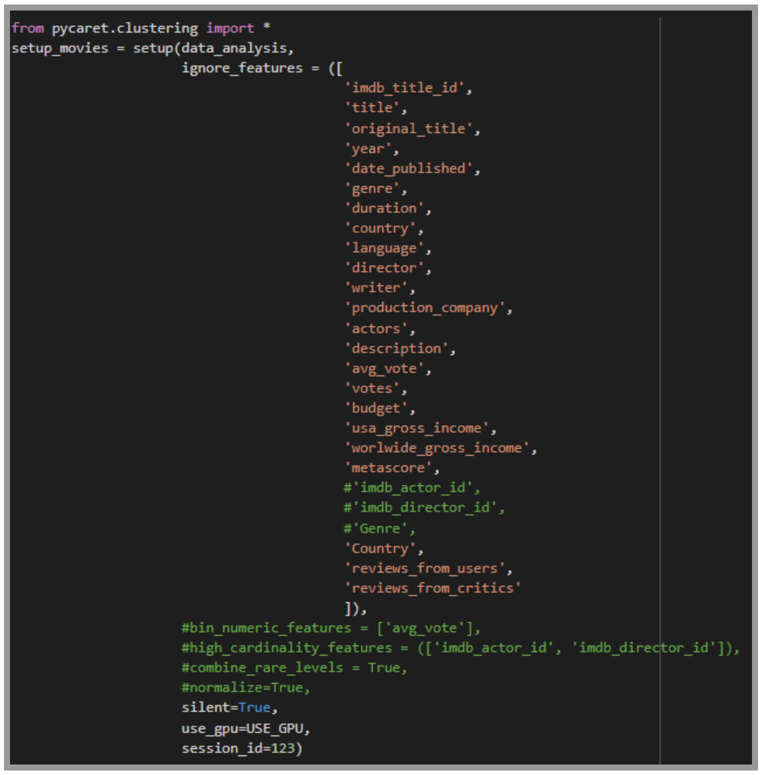
Model Generation

Choice of Clustering Algorithm

Pycaret offers several algorithms for clustering besides Kmeans, including Affinity Propagation, Agglomerative Clustering and Kmodes, which is supposed to work better with categorical data. in the end though, Kmodes never finished a model before crashing, and other clustering models that did work had limited ability to be plotted with the tools we understand at this time. So we ran with the one that completed its work, and gave us something we could interpret. When generating models for comparison, several setup options were tried:

- combinations of: year, avg_vote, metascore, reviews from users, reviews from critics, country
- tried binnings, normalization
- one-hot encoding of genre

In general we found that Kmeans was highly sensitive to numeric data, and tended to land hard on a particular category, in whatever setup we chose, it never found middle ground anywhere. So it fell on year, or it fell on Avg vote, or it picked genres. Our final setup we left to Genre, as we deemed that the most natural user expectation of the three, and the setup environment looked as follows:

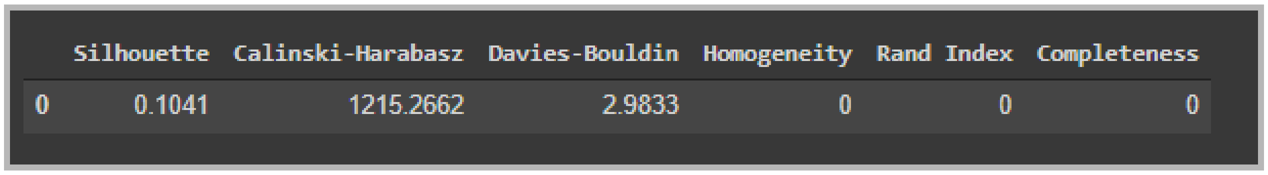


```
from pycaret.clustering import *
setup_movies = setup(data_analysis,
                     ignore_features = ([
                                         'imdb_title_id',
                                         'title',
                                         'original_title',
                                         'year',
                                         'date_published',
                                         'genre',
                                         'duration',
                                         'country',
                                         'language',
                                         'director',
                                         'writer',
                                         'production_company',
                                         'actors',
                                         'description',
                                         'avg_vote',
                                         'votes',
                                         'budget',
                                         'usa_gross_income',
                                         'worlwide_gross_income',
                                         'metascore',
                                         #'imdb_actor_id',
                                         #'imdb_director_id',
                                         #'Genre',
                                         'Country',
                                         'reviews_from_users',
                                         'reviews_from_critics'
                                         ]),
                     #bin_numeric_features = ['avg_vote'],
                     #high_cardinality_features = (['imdb_actor_id', 'imdb_director_id']),
                     #combine_rare_levels = True,
                     #normalize=True,
                     silent=True,
                     use_gpu=USE_GPU,
                     session_id=123)
```

Model Summary and Findings

We used K-means as the algorithm for our model as it is used to find groups and confirm business assumptions about what types of groups exist or to identify unknown groups. Within our model we examined a significantly large variation in type and number of features input to the model. Our final model which would support a recommendation system, used the primary identifiers most people use for movies: Genre, Actor, Director. Additional components such as writer could be incorporated

Model Score



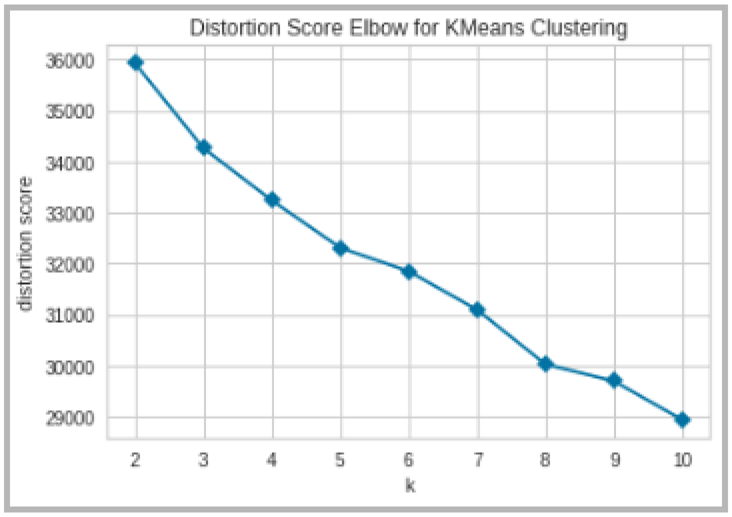
	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.1041	1215.2662	2.9833	0	0	0

Our Silhouette score is between -1 and 1 with 1 indicating highly dense clustering. Our Silhouette score of 0.10 indicates that there are overlapping clusters in our model.

The Calinksi-Harabasz score of 1215.2662 is not nearly as high as other models we created. The higher the score the more dense and well separated the clusters are. In our case the clustering is quite low indicating overlapping and less dense clusters. The higher scores were typically achieved when adding in year, or avg_vote into the clustering mix. While the model clustered better, it's clustering around year or avg_vote, did not match our business vision for the recommender system.

The Davies-Bouldin score of 2.9833 is on the higher side of average in our model experiments. Our scores typically ranged from 0.5 to 4.3. The higher score indicates a model with less separation between the clusters.

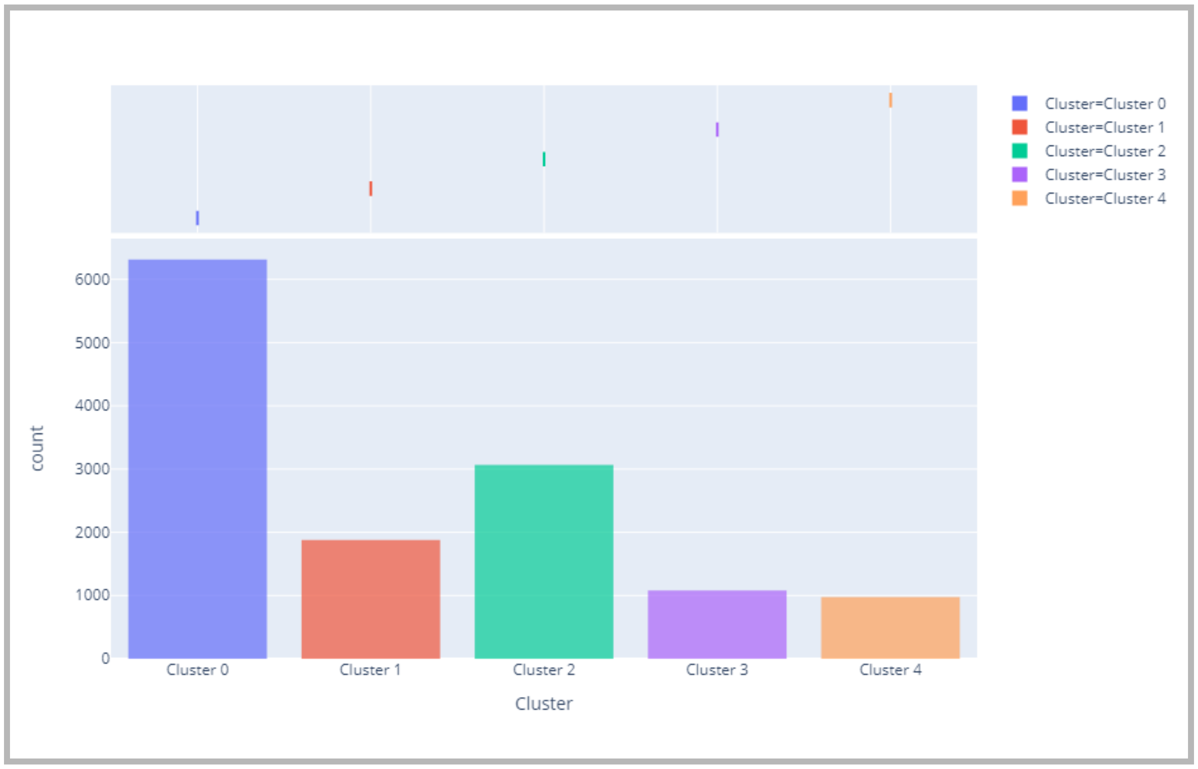
Elbow Plot



We generated an elbow plot to determine the ideal number of clusters. The relatively straight line of the elbow plot, with no apparent bend which indicates the lack of an obvious ideal cluster size within the range given. Cluster size experiments were performed and confirmed this finding. A final cluster size of 5 was chosen as it matched the breakout of our movie genres, and matched the business intentions.

Distribution Plot

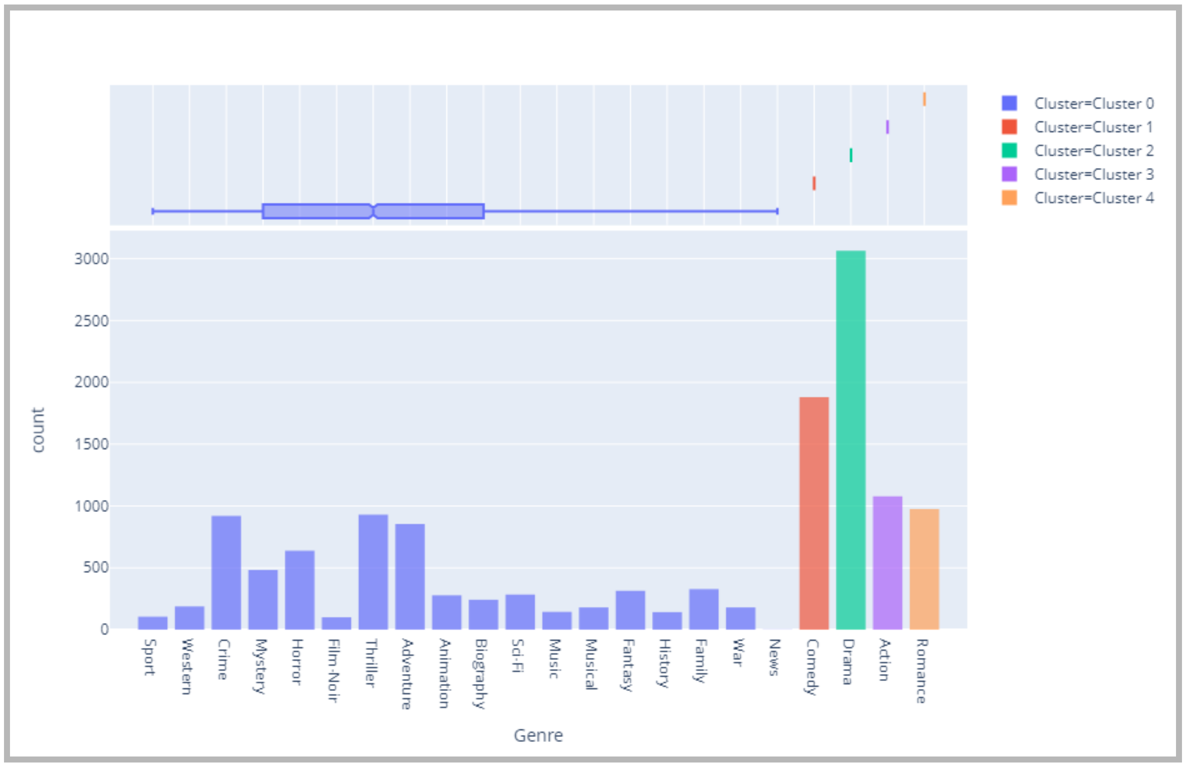
Our distribution plot showed a wide variation in the size of the clusters, with one cluster doubling the size of the next largest. While it would be nice to have similar size clusters for the separation of more data, the disparity in sizes works well when considering the distribution plot by genre, the key differentiator in our model.



Distribution Plot by Genre

Examining the distribution plot by Genre allows us to see that the model has separated the movies out into clusters based on genre. When examining the data there were high instances of Comedy, Drama, Action, Romance. While there are movies that are defined with only one genre, the most frequent being the aforementioned genres, more movies are considered combination genres and typically include one of the primary four. Instances of “Action, Comedy”, “Crime, Drama”, “Comedy, Romance” abound in the data set.

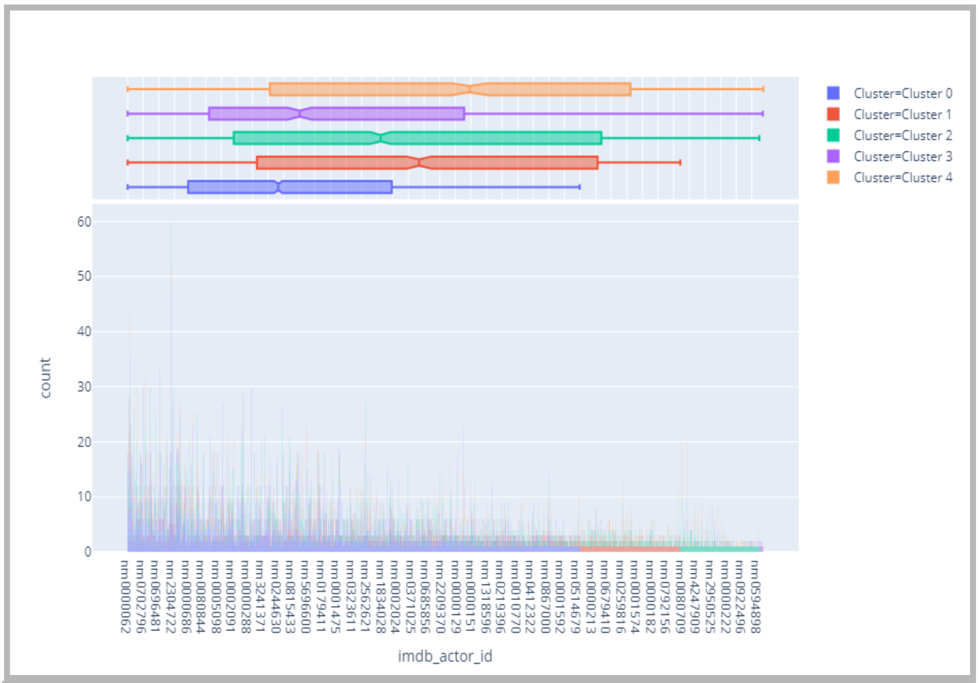
Since the data was normalized, the majority of movies will appear in at least 2 clusters, their primary, plus a secondary, or tertiary one.

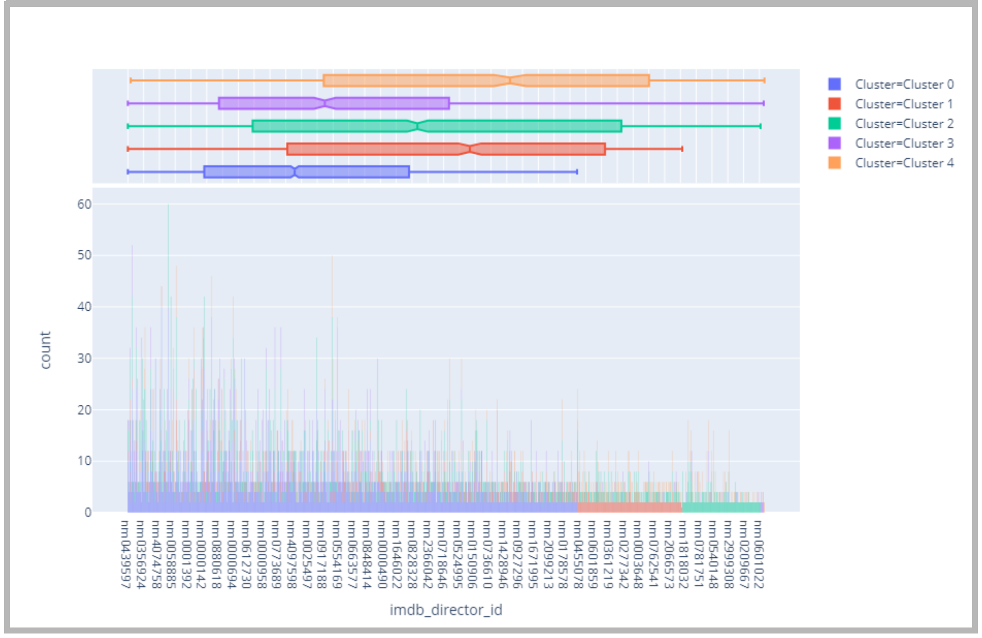


Distribution Plots - Actors/Directors

The distribution plots for the imdb_actor_id, and the imdb_director_id, show very little differentiation between clusters and rather show significant overlap. Both imdb_actor_id and imdb_director_id were used in our model. The cardinality difference between actors and genres makes it easy to understand why the model would focus on clustering genre.

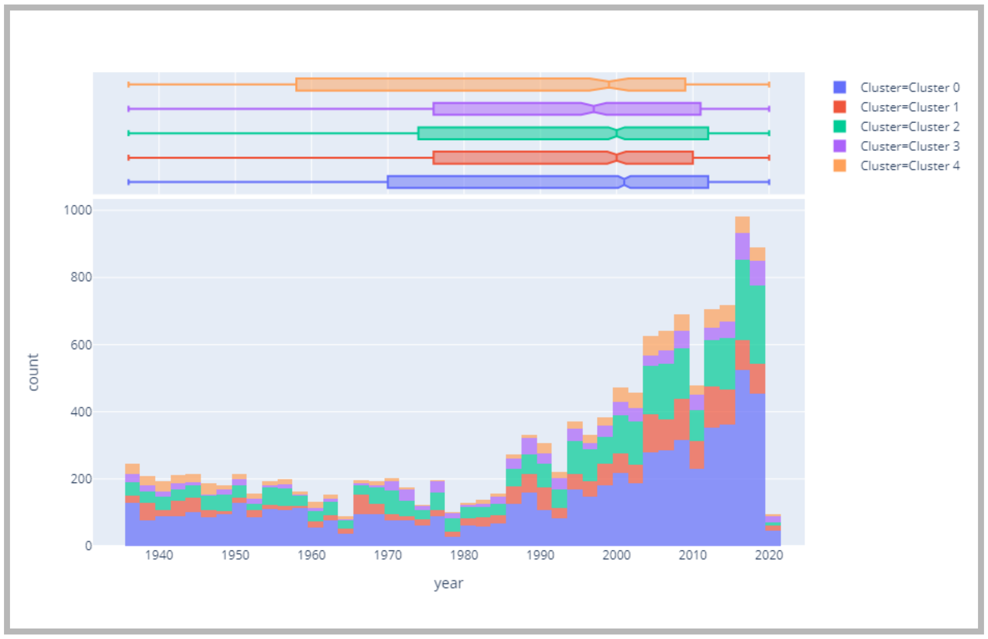
Another component to note with any actor/director differentiation in clusters is to consider that generally most directors adhere to similar movie genres, a small subset of genres, and typically deviate outside that very little.

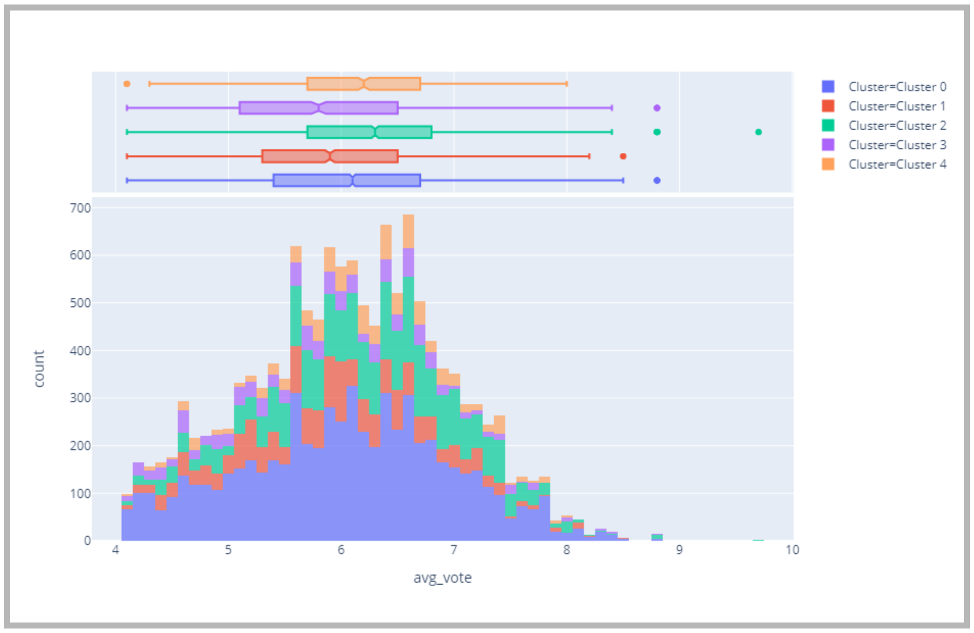




Distribution Plots - Non-Model Features

The distribution plots for other features within our dataset but not used within our model showed wide variation with few noticeable or intriguing patterns.





Recommender Functionality

The movie clustering model we have chosen lines up quite closely on the category of genre. Within each cluster, however, are movies with other genres indicated, as it is a multivariate field. Our recommender program then interrogates the cluster set to find matches on the other primary fields, and returns relatively high-scoring titles. This is typically how people refer to and/or recommend movies to others. "You liked The Sting? You gotta see Butch Cassidy. Same group of people and creativity, but different genre."

User selects a movie they wish to view or see recommendations on. The recommender takes that movie, uses our model to determine it's cluster, and along with other key criteria actors, directors, and genres generates a list of recommended movies you may like. An important factor to understand about this clustering is that it is not based on user scores, which were not available in our data set, but rather more concrete movie information.

The subjective component of classifying movies into genres does have an impact on potential recommendations. For example most Star Wars films are considered "Action, Adventure, Fantasy", while some are considered "Action, Adventure, Sci-Fi".

Since the data has been normalized for analysis purposes, each movie will have multiple rows associated with it, and each of those rows will be used within the recommender. As well since each movie may have multiple genres, it may exist in more than one cluster.

Each row representing the movie to be recommended upon will be run through the Recommendation Engine and return a number of movies (each with more than one row), that match the criteria.

Criteria 1: cluster
Criteria 2: cluster, genre
Criteria 3: cluster, genre, actor
Criteria 4: cluster, actor
Criteria 5: cluster, director
Criteria 6: cluster, director, actor
Criteria 7: cluster, genre, director

Let's look at an example movie "The Sting" starring Robert Redford, and Paul Newman. The recommender will examine all the available record combinations of cluster, genre, actor, and director for The Sting, and using it's criteria, if matches are found the top scoring results (sorted by avg_vote)

ther matching could be included, such as removing cluster, and adding in matches for all "actor, director" in any cluster. This could be one of many intuitive additions to the algorithm as many actors and directors have made multiple movies together.

For example: If "Butch Cassidy and the Sundance Kid" were in the same clusters, then the movie record featuring Robert Redford in "The Sting" would match on the movie record for "Butch Cassidy" under the above Criteria 4. A similar situation would occur when the movie record for "The Sting" and "Paul

Newman” was analyzed, it would add an additional instance of “Butch Cassidy” due to a match in Criteria 4.

Since “Butch Cassidy” is also the same director for “The Sting” there would be additional matches by the recommender under Criteria 5 and Criteria 6.

“Butch Cassidy has a movie genre of “Biography, Crime, Drama”, and “The Sting” has a movie genre of: “Comedy, Crime, Drama”. The two movies match on two of the three genres. This will add in matches for Criteria 2, 3, and 7. These two movies would have guaranteed matches on six of the 7 criteria.

Since there could be a large number of matches, the recommender only takes the top movies by avg_vote. As such note that Criteria 1 would only be met if “The Sting” were a top rated movie in it’s cluster.

With “Butch Cassidy” having:

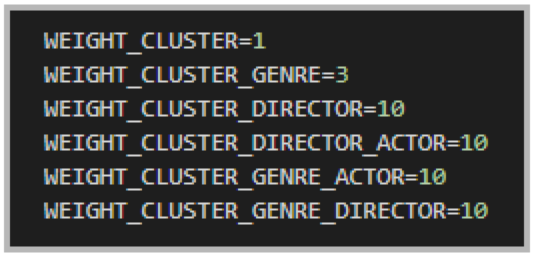
- 2 actors
- 3 genres
- 1 director

It would normalize to 6 distinct records. Assuming the two movies are clustered the same and since there are 2 actors, 2 genres, and 1 director as a match between Butch Cassidy and “The Sting” there would be 4 records the Recommender would match. Since the two movies match on 6 of the 7 criteria as noted previously, and there are 4 records the Recommender is matching on, there would 24 record matches determined by the Recommender

During it’s search the Recommender will find 100’s and 100’s of matches across the entire dataset. These matches are collected together, and combined into a single aggregated data set. The aggregated data set will include all movie fields, in addition to the normalized fields created during our data construction phase, as well as the clustering information provided by the machine learning model. This single data set is then sent for weighting and top movie selection.

A weighting mechanism is embedded within the Recommender to add tweaking functionality to increase/decrease weighting on various match combinations. This has the effect of improving or worsening search results based on their particular category. It would be possible to increase the weighting for actors, creating a recommendation list that was more dominated by actor matching than other criteria. Similar weighting alterations could improve or diminish a matching criterias appearance in the final recommendations. For example it would be possible to ensure that if there is a match on actor, genre, and director to weight it heavily ensuring it’s place at the top of the recommendation list.

This weighting allows more granular ability to customize the search results as more information is obtained, and/or to better align with changing business priorities.



```
WEIGHT_CLUSTER=1
WEIGHT_CLUSTER_GENRE=3
WEIGHT_CLUSTER_DIRECTOR=10
WEIGHT_CLUSTER_DIRECTOR_ACTOR=10
WEIGHT_CLUSTER_GENRE_ACTOR=10
WEIGHT_CLUSTER_GENRE_DIRECTOR=10
```

The Recommender receives this dataset of 100's of matches to other movies and then through an aggregation process it reduces these hundred of record matches (more than one per movie), to a single record for each movie which includes it's aggregate and weighted score. The top 10 movies based on weighting are then returned as a recommendation set for the end user.

The aggregator currently focuses on frequency based weighting mechanism, however during the aggregation phase, additional weighting elements could be added based on other available criteria such : metascore, budget, or number of ratings. This would allow a more nuanced configuration of the recommendation engine.

In the User Interface it would be recommended to add in a feedback mechanism for users to agree or disagree with the movie recommendations. Collecting this data and including it within the data model would assist in further differentiating movie recommendations and create new and stronger clustering.

The recommender and it's weighting mechanism does not rely on encoded logic, rather it looks at the strength of the matches through calculation of frequency of occurrence.

Because the recommender function bases it's weighted recommendation on the occurrences of titles in it's recommendation set, it's important to note that there is a bias towards actors/directors who have been associated with a larger number of movies. This was intended based on multiple matching criteria being available between the movies. After a sufficient feedback mechanism is put in place, additional weighting metrics should be added during the recommender rollup phase to help provide a more operationally tuned recommendation system.

Ethical Considerations

Simply based on the variety of stakeholders, the ethical considerations when developing and implementing this solution can be significant. The first tier stakeholders we can identify in this scenario, would be the users of the system, the company providing the system, and the movie content owners.

Let's first look at the users of the recommender. The current data set does not include categories for movie rating (Restricted, Family, Parental Guidance, …) and as such it becomes possible to start including recommendations that are inappropriate for the target audience. Were a very young child looking up the animated film "Chicken Run", it would be problematic to recommend "Braveheart" simply because Mel Gibson is in both.

If we were to enhance the recommender to start including a history of movies that the person has searched for we would be capturing a level of personal data. While this data could benefit the user to improve search results, it could also be used to steer advertising or bombard the user with targeted recommendations. The company's desire for profit and website traffic are at odds with a users expectation of privacy and a non-manipulative recommendation engine. Care should be taken to ensure the users security, privacy, and information are maintained.

Another area of potential concern is if the content owners are allowed to advertise by way of altering the ranking algorithms. It’s not difficult to increase the weighting of certain actors/directors/production companies so that they appear in the “recommended” movies more often. This would be transparent to the users and similar to the push/pull of expectations described above.

A similar situation could occur where there is active demotion - blacklisting - of some movies/actors/directors to ensure they no longer appear in search results, this could come from public or private pressure.

With search results(recommendations) being a key component of advertising on the site this would have an impact on viewing, popularity, and reviews. This could lead to favored treatment of some content owners at the expense of others.

Conclusions

The created model works as intended and allows for strong and relevant search results. Based on our own experience with Spotify, a good recommender system introduces you to a disparate catalog that is both suited and new to you. Now the entry point is a 3-minute song and not a 90-minute movie, but the idea is it opens your horizons, and hopefully puts you onto things as a fan and customer that you *wouldn’t have found yourself.* That’s a win for the content provider and the consumer. And the artist, for that matter.

What enhancements could be made to make this model better?

Hindsight suggests a few things we could try to make our model more efficient:

- We didn’t experiment with fields like writer or studio, those could be interesting.
- Also, our random paring down from 22k to 5k movies for the model build would have thinned the density of director and actor matching from ‘normal’, another pass at that we would try to use a tighter range of years to shrink the size but maintain the density. The generated model then runs against the whole set anyways, but might have found a better balance of weighting between personnel and genre.
- Within the cluster it would have been useful to be able to find our ‘near neighbors’. There is a way to [x,y] plot the cluster based on PCA, if we could feed that simplified data into an algorithm like https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html, then we should be able to quickly grab close matches, even though our set is not a nearest neighbors model.

However, the model could always be tweaked to find some notion of a perfect match, but the recommender makes some adjustments for that, and in the end the business case doesn't really require it. A good list of 10 is generated, and if one of those looks interesting enough to click on, another 10 can be popped up and the customer can follow the path wherever their own interest leads them. This system would be a precursor and eventual adjunct to a system based on clustered customer viewing history, and finding unseen titles from people with similar patterns. That history takes time to build, on both a new customer, and in our case on all our customers. Adding user level movie preferences into the analysis would dramatically improve the recommendations and also provide much stronger clustering opportunities.

Bibliography

Medium.com. Haitian Wei. How to measure clustering performances when there are no ground truth? URL: https://medium.com/@haataa/how-to-measure-clustering-performances-when-there-are-no-ground-truth-db027e9a871c

Towards Data Science. T. Sarkar. Clustering metrics better than the elbow-method. URL: https://towardsdatascience.com/clustering-metrics-better-than-the-elbow-method-6926e1f723a6

Oracle AI & Data Science Blog. Andrea Trevino. Introduction to K-Means Clustering. URL: https://blogs.oracle.com/ai-and-datascience/post/introduction-to-k-means-clustering

Towards Data Science. Ashish Rana. Build Your Own Clustering Based Recommendation Engine in 15 minutes!! URL: https://towardsdatascience.com/build-your-own-clustering-based-recommendation-engine-in-15-minutes-bdddd591d394

Paul DeSalvo's blog. Paul DeSalvo. How to Break up a Comma Separated String in Pandas Column. URL: https://www.pauldesalvo.com/how-to-break-up-a-comma-separated-string-in-a-pandas-column/