# A Content-Based Movie Recommendation Engine

Ana Carolina Camargos Couto
*dept. of Applied Mathematics*
*Western University*
London, Canada
acamarg@uwo.ca

***Abstract*—This project is a result of the application of machine learning algorithms to a movie dataset, extracting meaningful relationships between the movies therein, and ultimately constructing a recommendation engine.**

***Index Terms*—recommender systems, support vector regression, principal component analysis**

## I. Introduction

This paper describes the development, implementation, and results of a movie recommendation engine based on a public dataset. Present-day media service providers benefit from utilizing recommender systems on their platforms as a way to improve the quality of their service and increase the number of sales. In particular, movie recommender systems attempts to successfully recommend relevant movies to those who watch them, i.e., users of media services.

This project uses Support Vector Regression to build a costumer specific recommendation model. The model is trained on movie attributes, and learns how to predict movie ratings. This paper is subdivided as follows: in Sec. II, the underlying principles of the main algorithms used in the project will be introduced. In Sec. III, other approaches from the literature will be discussed. In Sec. IV, the methodology designed for the development of the recommendation engine will be presented, and in Sec. V, the recommender results will be analysed.

## II. Background

In this section, we will introduce two of the core algorithms applied in the scope of this project, as well as the selected evaluation criteria.

### A. Singular Value Decomposition (SVD)

SVD is a matrix factorization algorithm that can be applied to dimensionality reduction problems by finding the largest singular values $\sigma_1, \sigma_2, ..., \sigma_r$ of an $m \times n$ matrix $A$, such that:

$$A = U\Sigma V^H \tag{1}$$

where $U$ and $V$ are unitary matrices and $\Sigma$ is an $m \times n$ non-negative diagonal matrix such that $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_p)$ with $p = min(m, n)$ [1]

Furthermore, we can represent the principal components of $A$ by the columns of $AV = U\Sigma$. The largest principal component of a matrix is the direction that maximizes the variance of the projected data [2], as shown in Fig. 1:

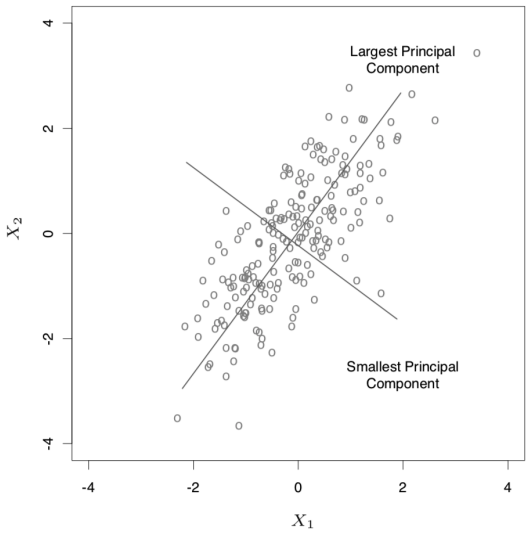


Fig. 1. Geometrical example of a principal component decomposition of 2D data components into 1D components. [2]. The new coordinates are the orthogonal distances between the data points and the largest principal component axis.

### B. Support Vector Regression (SVR)

SVR is a regression algorithm, and therefore it models the relationship between a dependent variable (labels) and independent variables (attributes) of a dataset. What characterizes SVR is that it converges by applying the same principles of a Support Vector Machine (SVM).

SVM is a classification method that attempts to find the fattest hyperplanes that separate the data into groups, or classes. Polynomial or radial basis kernels can be used when the segregation boundaries are not linear. Alternatively, SVR is a method that tries to find the thinnest hyperplane that *contains* the data points, sequentially fitting a curve through the points by finding the mean location of the hyperplane (see Fig. 2).

The SVR model can be specified by a set of hyper parameters: $epsilon$, $cost$, and $gamma$. $epsilon$ is the allowed distance between predicted points and actual values. If the predictions are made within the $epsilon$ boundary, no penalties are established. The $cost$ determines how large is the penalty when predictions are made outside of the $epsilon$ distance, and $gamma$ is a parameter that defines the kernel relationship.

E.g.: in the case of radial basis kernels, $gamma$ is the inverse of the standard deviation factor that measures the similarity between two points, and it is used to express the gaussian function that defines the kernel.

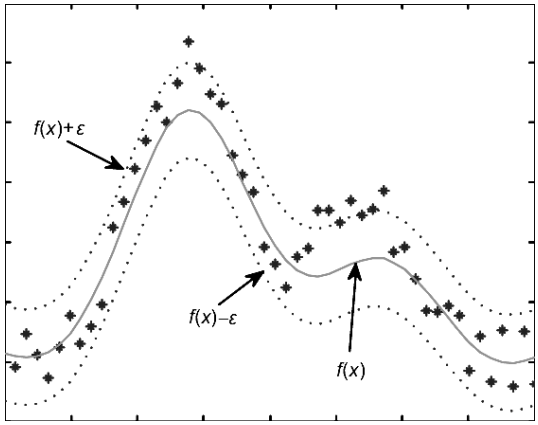


Fig. 2. Geometrical example of SVR with a polynomial kernel applied to a dataset where nonlinear relationships being modelled. [3]

### C. Evaluation

When applying SVD for dimensionality reduction purposes, it is usually desirable to transform the original dataset to obtain the smallest possible number of components without losing too much information. In order to evaluate the amount of lost information, the *variance explained* sum is analysed. The variance explained of one component is the ratio between the the variance from that component, and the total variance of the data in the original space. Ideally, the sum should be a value close to 1.

The goodness of fit of a regression model can be evaluated with a few different measures. Some examples include the Root Mean Squared Error (RMSE) and the R2 indexes:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2} \tag{2}$$

$$R2 = 1 - \frac{\sum_{i=1}^{N}(y_i - y_i')^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{3}$$

Where $y_i$ denote the real values of the independent variable, and $y_i'$ are the predicted values for them. $\bar{y}$ is the mean of the real values, and $N$ is the number of samples.

The RMSE index is a standard deviation of the residuals, and it can be interpreted as an error interval within which the real measurement can lie. The R2 index measures how well the model is predicting $y$ (in the ideal scenario, $R2 = 1.0$), and it also measures whether the model is constant, which is the case of a model that always predicts the expected value of $y$ (for those cases, $R2 = 0.0$). For the recommendation models, both the RMSE and R2 are being used as accuracy measures.

## III. Related Work

Recommender systems can be constructed using a variety of strategies. One of the most common strategy is the collaborative-based method. It works by grouping people into neighbourhoods of users with common interests, with the objective of recommending to users items that were ranked high by their neighbours.

Collaborative-based recommenders need only the rating history of a group of users in order to be applied. Because it doesn't take many resources, it has been widely used, including by top competitors of the *Netflix Prize data* challenge set by Netflix on Kaggle [4].

The weakness of this approach is the *cold start* problem, i.e., the challenges with recommending new movies on the platform, as they don't have ratings, and therefore lack data to make predictions as to which users they should be recommended to.

The strategy used in this paper is a content-based approach. It uses the descriptive attributes of movies (such as title, overview, release date, etc.) and the rating history of each costumer in order to recommend movies that are similar to the ones they rated high. Content-based methods work well with cold start problems, since a new movie will always have its attributes available to be explored.

However, content-based methods can be formulated in a number of different ways. The data available for these kinds of problems usually exhibit the shape from table I. The goal is to build a model that can predict the missing labels, i.e., a model that will say how a user will rate movies that they haven't rated yet.

TABLE I
EXAMPLE OF MOVIE DATA FOR CONTENT-BASED RECOMMENDERS. THIS TABLE SHOWS THE RATING HISTORY OF ONE COSTUMER.

| **Item ID** | **Attributes and labels** | | |
|---|---|---|---|
| | ***Attribute 1*** | ***Attribute 1*** | ***Label*** |
| movie 1 | title 1 | overview 1 | rating 1 |
| movie 2 | title 2 | overview 2 | rating 2 |
| movie 3 | title 3 | overview 3 | |
| movie 4 | title 4 | overview 4 | rating 4 |

One of the approaches presented by *Charu C. Aggarwal* in [5] is based on the k Nearest Neighbour Classifier (kNN). The idea of his approach is to first compute the pairwise similarity score between the movies, using, for instance, the cosine similarity index:

$$Cosine(\bar{X}, \bar{Y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}} \tag{4}$$

Where $\bar{X}$ and $\bar{Y}$ are any two column attributes, and $x_i$ and $y_i$ are their elements. The next step is to use the similarity scores to feed a kNN model, which will group the movies into neighbourhoods of similar items. The main challenge with this approach is its computational cost, since the nearest

neighbours of each item needs to be determined, and for each nearest neighbour determination, the time required is linear to the size of the number of items.

Another strategy that can be applied to content-based systems is fitting to the data a regression model, and this strategy is well suited for problems in which the labels are numerical quantities that exhibit a specified order. The ratings in the dataset selected for this project are numerical values from 1.0 to 5.0, where 1.0 expresses the lower end of the rating scale, and 5.0 expresses the upper end. Therefore, the method adopted by this paper was regression-based. Furthermore, once the model is trained, it can be effectively applied to new instances, with a low computational cost.

## IV. Methodology

The dataset used in this project is entitled "The Movies Dataset" and can be downloaded from Kaggle [6]. There are two main tables used in the dataset architecture: the movie table, with 45000 movies, where for each movie, there is a row of attributes:

- movieID: int
- budget: float
- genres: dictionary
- original_language: nominal
- title: string
- overview: string
- popularity: float
- release_date: datetime
- revenue: float
- runtime: float
- tagline: string
- vote_average: float
- vote_count: float

and the user table, which contains the movie ratings of 270,000 users from the database, with the following rows:

- userID: int
- movieID: int
- rating: float

In order to build a table similar to table I for each costumer, the user table was filtered for each userID, and the filtered user-specific tables were joined to the movie table on the movieID key. Through this process, we can obtain 270,000 user-specific tables which contain the movieID, the movie attributes and the rating given by the user.

Separating the data into user-specific tables is a necessary step for content-based recommenders, since each user is treated independently of the others, and for each user, the rating history of others is not needed for making recommendations. Therefore, the system developed in this project creates a separate model for each costumer. The model learns user-specific preferences in order to make user-specific recommendations.

The following sequence of steps describes the methodology applied to each costumer data: first, 15% of the data is held-out for testing purposes. the remaining 85% is the training data, and will be used for the learning stage. An SVR model is fit to the preprocessed training data. The model is then applied for making predictions on the test data, and the predicted ratings are compared with the real ratings in order for the model accuracy to be measured. Finally, The model is applied to the task of generating custom movie suggestions.

### A. Data Preprocessing and Feature Extraction

It is evident that The Movies Dataset contains a big variety of data types, which makes the preprocessing phase a challenge. Every type of column was treated in a specific way before it was fed to the regression model. First, the numerical columns were normalized in order for their elements to lie within the same range: [0,1] using *MinMax* normalization:

$$x_i = \frac{x_i - min(X)}{max(X) - min(X)} \tag{5}$$

Where min(X) and max(X) are the minimum and maximum values of the column X. The missing values in each column were replaced by the column average. Normalizing the data is a crucial step to the model's success, since regression algorithms are known to perform better when the variables are kept within a similar range. The original dataset contains values in different sets of ranges, e.g.: the $budget$ column contains values of order $10^9$, while the $vote\_avg$ column contains values of order $10^0$.

For the $release\_date$, which is a *datetime* type, only the year of release was selected. The dataset contains movies released from 1874 through 2020, and since the style of production does not change much throughout a year, only the year information is assumed to be necessary for this feature.

A vectorization method was applied to the non-numerical columns, which includes dictionaries and strings. Vectorization is a required step for those cases, since regression models can only understand language expressed in numbers, and therefore it cannot understand natural language, e.g., titles, names in genres, etc.

Dictionaries and strings were vectorized using different strategies. An example of a dictionary column is the $genres$ feature. Each movie has a set of descriptive genres, and they are stored in a list of dictionaries. E.g.: for the movie "Toy Story", the list of genres is stored as "['id': 16, 'name': 'Animation', 'id': 35, 'name': 'Comedy', 'id': 10751, 'name': 'Family']". A function with the purpose of extracting the names and writing them to a list was written, so for that movie, the genre list became:

['Animation','Comedy','Family']

After extracting the genre names of all movies in the dataset and writing them to a list with the function above, we can apply a one-hot-encoding technique in order to transform those lists into vectors. One-hot-encoded vectors are constructed using the space of classes as a basis (which, in this case, are the genre names). The vector length is equal to the total number of unique genres in the data corpus, and the presence of each genre is encoded in a binary representation. For the

example above, one of the possible vector representations for the genres in the movie "Toy Story" would be:

[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Where the $3^{rd}$, $4^{th}$, and $8^{th}$ positions of the vector indicate the presence of the genres from the list of names above. For The Movies Dataset, there are 20 unique genre names.

The vectorization of string features (title, overview, tagline), was carried out in a similar fashion, though for a corpus of unstructured text, there is a large number of unique words available. According to the experimental results obtained by [7] and described by [5], the number of words used for the construction of the vectorial space should be somewhere between 50 and 300 when the data corpus is large. The features in our dataset were better represented when 200 components were used. Noisy words often result in overfitting and should be removed. This includes *stop words*, i.e., words that are very frequent in the data corpus, albeit with no significant meaning, such as *the, a, an, of, this, in, at*, etc. For this reason, *stop words* from the most common languages found in movies, which are English, French, Spanish, and German, were removed from the corpus.

Statistically speaking, words that occur too often are less discriminative. But although it is worth removing the *stop words*, other frequent words carry some meaning, and we would be throwing out information by simply removing them. Instead, we can down-weight them using the *inverse document frequency* $(id_i)$ index:

$$id_i = \log(n/n_i) \tag{6}$$

Where $n$ is the total number of items (movies) in the collection, $n_i$ is the number of documents for which the $i^{th}$ word occurs, and thus $id_i$ is a decreasing function of $n_i$. So, instead of using the absolute number of occurrences, in this project, the $id_i$ is used for encoding the vector component of each word.

The vectorization of dictionaries and strings extended the number of features in the user dataset from 14 to 718, with every element of every vector being a new feature. However, those vectors are sparse, i.e., contain a great number of zero elements. Therefore, the 718 columns can be projected into a lower dimensional space without losing a lot of information. As mentioned in subsection II-A, the SVD algorithm can be applied for dimensionality reduction purposes, and the amount of lost information can be measured by summing over the *variance explained* index of each principal component obtained with the transformation. For this project, it was possible to reduce the 718 features to only 2 features without losing barely any information, as the variance explained of those 2 features summed to 0.9996.

### B. Validation Process

Since the models are user specific, each user model has its own accuracy measure. And it depends on the length of the rating history of the user, i.e., the amount of movies that they have watched and rated. If a user has not seen many movies, their model will not have a lot of supporting data for the learning stage, and consequentially, will probably have a smaller accuracy. The RMSE and R2 are computed for each user model for accuracy assessment. If we wish to evaluate the accuracy of the engine as a whole, which is the group of user models for all costumers in the database, we can average the RMSE and R2 over all of the trained models.

Additionally, the residual curve of the models can be assessed in order to provide a clearer analysis of the error rate for all the prediction values generated, giving a broader perspective of the model behavior on the dataset. the residual can be defined as $abs(y' - y)$.

Another way to evaluate the model's accuracy is to compare its predicting patterns with reality's behavior. Which is why an additional indicator is being used for validation purposes: the diversity measure. This indicator can be computed by counting the number of different genres being suggested in the $N_s$ most recommended movies for a user, and dividing through by 20, which is the total number of unique genres in the corpus:

$$diversity = \frac{number\_of\_genres}{20} \tag{7}$$

So, what values of novelty indexes should we expect if the models are satisfactory? If we measure the diversity index for various user models, and the predicted movie ratings accurately represents the actual ratings, intuition indicates that we will obtain a bell-shaped distribution. Different people have different tastes and preferences, some people are versatile and enjoy a fair variety of movie genres, while others enjoy just a few. It is equally improbable to find people who like all kinds of genres, and none. Most people would be located near the centre of the distribution.

As we increase the number of suggestions $N_s$, the number of recommended genres tends to increase. However, the expected distribution shape would still be a gaussian-like curve, and the effect of this parameter tuning would be to shift the location of the distribution mean. Therefore, we can set the number of suggestions to any number we like, and look for the distribution pattern.

In order to select the regression algorithm to be used, 3 different models were applied to the same problem and compared in terms of accuracy and performance. The selected models were SVR, Gradient Boosting Regression (GBR), and Linear Regression (LR). The 3 models are considerably different from one another in terms of the underlying mechanisms.

While LR works by applying the standard approach of Least Squares, GBR implements a group of decision trees (random forest) for carrying out the regression task, and converges by iteratively computing a loss parameter of the training predictions, and minimizing it by using the gradient descent of the loss function. The loss function, learning rate and number of trees in the random forest are free hyper parameters.

## V. Results

The implementation of the methodology presented in Sec. IV was carried out using the Python Scikit-learn library [8],

which offers open source Machine Learning algorithmic tools to the Python community. The project code was divided into two parts: preprocessing and modelling. The preprocessing part is the most computationally intensive one, and it needs to be completed beforehand so that it does not burden the model training part, which should be very efficient.

Support Vector Regression with a radial basis kernel, Gradient Boosting Regression, and Linear Regression algorithms were applied to the preprocessed data, and their accuracies were measured. A grid search method was applied to the algorithms in order to find the best hyper parameters. For SVR, the searched values were:

$$cost = [0.1, 1, 10],$$
$$epsilon = [0.1, 0.2, 0.5],$$
$$gamma = ['auto','scale']$$

where 'auto' is equal to the number of features, and 'scale' is equal to 1/((number of features)*variance(col)). For GBR, the searched values were:

$$n\_estimators = [100, 300, 500],$$
$$learning\_rate = [0.001, 0.01, 0.1, 1],$$
$$loss = ['ls','lad','huber','quantile']$$

where 'ls' and 'lad' stand for, respectively, least squares and least absolute deviation, 'huber' is a combination of the two, and 'quantile' allows quantile regression.

The grid search technique searches through every specified combination of parameters, and finds the best combination for tuning the model. The best parameters found for the SVR were: gamma = 'auto', epsilon = 0.2, cost = 0.1. And for GBR: learning_rate = 0.001, loss = 'ls', n_estimators=100. The grid search was applied to multiple user tables, and the most commonly generated parameters were selected.

The tuned models were independently trained on 1000 randomly picked user data frames, and the models' accuracy and performance were compared. Fig. 3 shows the results obtained for each model in terms of the calculated RMSE and R2 averages, and the measured runtime in $s \times 10^3$

From Fig. 3, we can see that the measured RMSE for the three models were roughly equal to the same value (from 0.9 to 0.95), with the SVR attaining the lowest error. The RMSE of each model tells us how far off we are from the real values. In the case of movie recommendations, it tells us how far off the predicted ratings are from the possible ratings given by the user. An error of 0.9 means that the actual rating y can be expressed in terms of the prediction y' by:

$$y = y' \pm 0.9 \tag{8}$$

Since the rating values range from 1.0 to 5.0, this error rate is acceptable, though there is a lot of room for improvement. When moving our attention to the R2 results, we can observe a surprising pattern. According to Eq. 3, a 100% accurate model has $R2 = 1.0$. And we can see that LR model outperforms the other models when this index is analysed, and GBR is still the least accurate model.

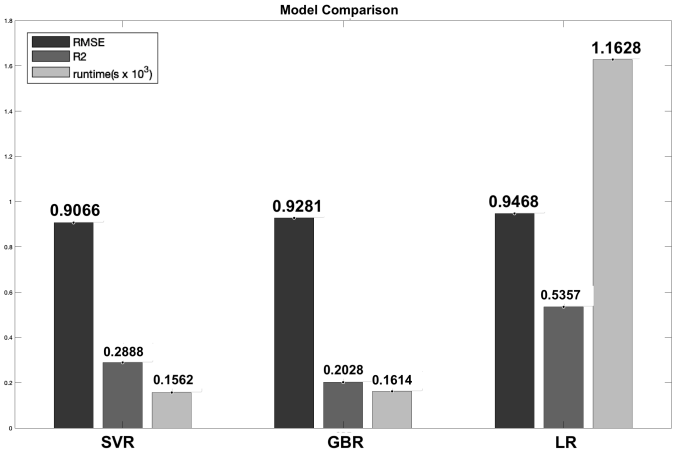


Fig. 3. Comparison of SVR, GBR, and LR in terms of accuracy and performance, averaged over 1000 user data frames.

The determining factor is the runtime. Although LR demonstrates a better performance than SVR in terms of the R2 index, it is expressively slower. Efficiency is a crucial factor to be taken into account for a recommendation model, since it has to provide suggestions to users in a fluid manner.

Finally, the SVR was selected as the main algorithm to train the user models, and applied for the task of making movie suggestions. To demonstrate the engine operation, 3 randomly picked users with a sufficiently large rating history were selected and the suggestions generated for them are summarized in tables II, III, and IV.

TABLE II
TOP 5 MOVIE SUGGESTIONS FOR USER 46. RATING HISTORY: 311 ITEMS

| **Title** | ***Genres*** |
|---|---|
| El Dorado | Adventure, History |
| Song of Freedom | Drama, Music |
| Trailin' West | Western |
| Treachery Rides the Range | Music, Western |
| A Message to Garcia | Drama, Romance |

TABLE III
TOP 5 MOVIE SUGGESTIONS FOR USER 24. RATING HISTORY: 287 ITEMS.

| **Title** | ***Genres*** |
|---|---|
| Darkness Falls | Thriller, Horror |
| The Recruit | Action, Thriller |
| The Singing Detective | Comedy, Music, Mystery, Crime |
| Blown Away | Drama, Action, Thriller |
| The Getaway | Drama, Action, Thriller |

Upon analysing the movie suggestions for user 46, 24, and 49, we can recognize a consistency of the models in

TABLE IV
TOP 5 MOVIE SUGGESTIONS FOR USER 49. RATING HISTORY: 174 ITEMS.

| **Title** | ***Genres*** |
|---|---|
| White Squall | Action, Drama |
| James and the Giant Peach | Adventure, Animation, Family |
| Extreme Measures | Drama, Thriller |
| My Best Friend's Wedding | Comedy, Romance |
| MouseHunt | Comedy, Family |

terms of the genres and types of movies suggested for each costumer. For example, it is evident that user 46 must have a preference towards History-themed movies, judging by the recommendations in table II. User 24, on the other hand, was suggested a fair amount of Horror movies, and user 49 was offered titles that can be considered popular and casual.

The residual curve of each user model is shown in Fig. 4, 5, and 6.

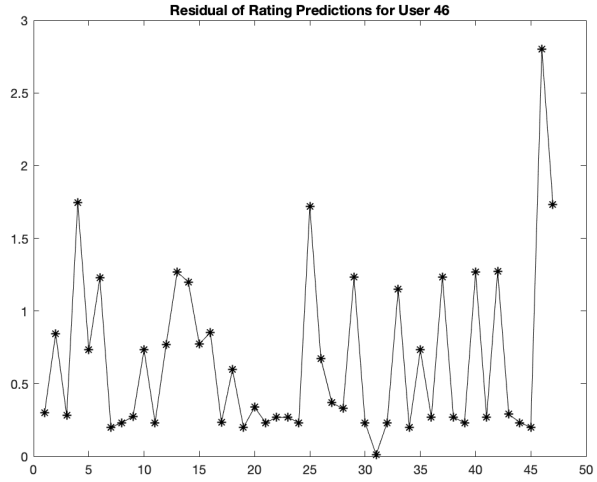


Fig. 4.

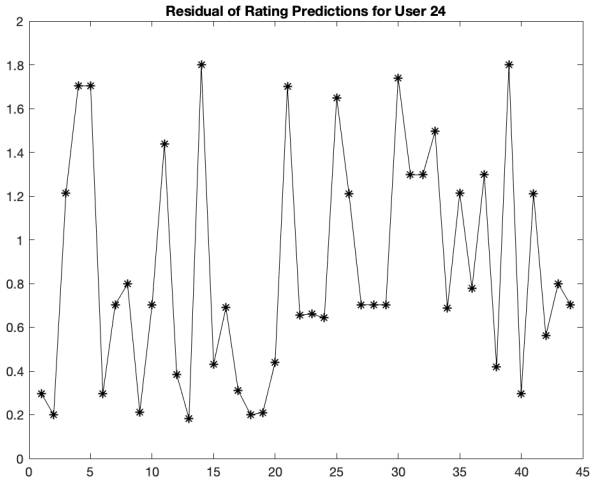


Fig. 5.

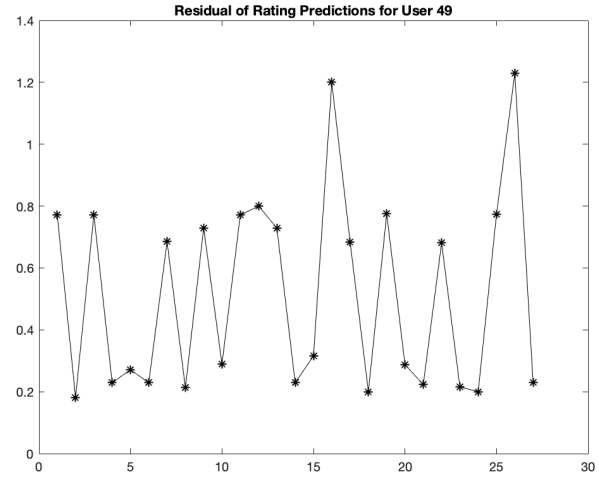


Fig. 6.

The residual can be computed by evaluating the difference between the predicted ratings and the actual ratings from the test set. The accuracy of the user model depends on the aspects of the training data. Some user data frames have more distinguishable patterns than others (e.g.: some users ratings are more precise than others), and when that is the case, the algorithm will have an easier time in the learning phase.

This observation becomes clear when we analyse the residual figures. Although there isn't much disparity between the curve ranges, the patterns are different. The model for user 49 displayed the best behaving residual curve, with the majority of points lying within the 0.8 bound. User 46 displayed a bigger amplitude in residual variation, with most points being bound by a 1.25 error rate, and user 24 exhibited the worst residual rates out of the three, with a few rates around 1.7 being measured.

In spite of the fact that user 49 has the fewest number of ratings in their data frame, the model for that user achieved the best prediction accuracies out of the three. The reason for this result may be because the *patterns* in user 49's data frame are more neatly laid out, either by the costumer him/herself of by the collection of movies they have watched.

If we wish to evaluate the accuracy of the engine as a whole, one way to do it would be to plot all residual values from the predictions obtained by all the user models. Alternatively, we could select a large enough random group of user models to represent the sample space. For efficiency purposes, the latter approach was selected. 1000 randomly picked user models were trained and their residuals were measures. Fig. 7 is a graph showing the resulting residual curve. We can see that the great majority of predicted values lie within the 1.25 error boundary, with a few values spreading out to 2.0, 3.0, and a rare few reaching 4.0. This pattern shows that the user models are being consistent in their predictions, which are not much distant from the real values.

In terms of diversity analysis, a similar approach can be

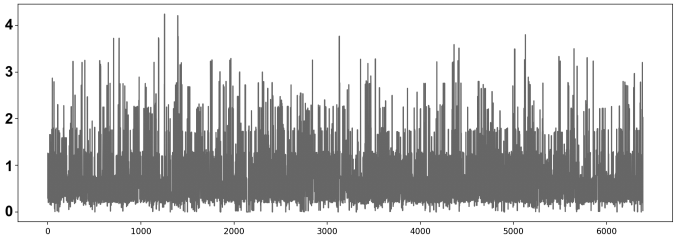


Fig. 7. Residual of 1000 randomly selected user models

used for sampling over the user models. By applying equation 7 to the recommendations found for 1000 randomly picked user models, and plotting the obtained data into a histogram, we can have an idea of the distribution of the suggested genre diversities. Fig. 8 shows the resulting histogram. As expected, the distribution has a bell shape, with the majority of occurrences being located in the centre. These results indicate a correspondence between the model and reality patterns.

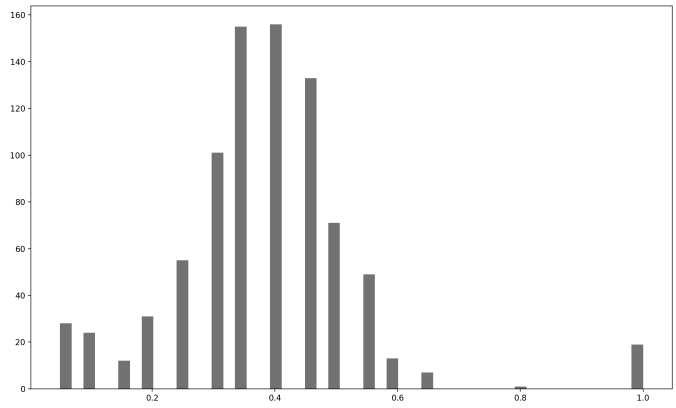


Fig. 8. Diversity histogram of recommendations for 1000 randomly selected user models.

## VI. Conclusion

This paper contains a full description of the development process of a content-based movie recommendation engine. In summary, a Support Vector Regressor (SVR) was trained on user data frames of a movie dataset. The obtained user models were applied to the task of relevant movie recommendation. As a content-based approach, the developed system relies on movies attributes and descriptions for training the regressor. The independent variables are the movie ratings, and therefore the trained models are able to predict the rating values.

Though not ideal, the accuracy results of the SVR user models were satisfactory. From Fig 3, we can see that the RMSE average of 1000 user models was equal to 0.9066. Since the rating range is given by [1.0, 5.0], this error value could mean, for example, that if we predict that a user will rate a movie 5.0, they can actually rate it 4.0, which is not so distant. The R2 results, however, were closer to 0.0 than 1.0, which mean that the model have some constancy, i.e., the predicted values are still very close to the expected values.

In order to make the system more distinguishable, other options of models could be considered in the future. Further analysis would have to be conducted to determine if this is an underfitting case. Underfitting can be remedied by considering more complex models, i.e., models that learn complex relationships between the data features. Some architectures of Neural Networks can be considered in future plans.

A hybrid approach can also be explored in future work. Hybrid recommenders combine the power of collaborative- and content-based systems to produce a more dynamic engine. While content-based systems can effectively tackle the cold start problem, collaborative-based systems are able to provide less predictable recommendations.

This is a problem that can be extended way beyond the scope of this paper, and it can be applied in a wide range of contexts in addition to the movie industry.

## References

[1] Corless RM, Fillion N. A graduate introduction to numerical methods. AMC. 2013;10:12.
[2] Hastie T, Tibshirani R, Friedman J. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media; 2009 Aug 26.
[3] Huang, Hua-juan & Ding, Shifei & Shi, Zhongzhi. (2013). Primal least squares twin support vector regression. Journal of Zhejiang University SCIENCE C. 14. 722-732. 10.1631/jzus.CIIP1301.
[4] https://www.kaggle.com/netflix-inc/netflix-prize-data/activity
[5] Aggarwal CC. Recommender systems. Cham: Springer International Publishing; 2016.
[6] https://www.kaggle.com/rounakbanik/the-movies-dataset
[7] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting Web sites. Machine learning, 27(3), pp. 313?331, 1997.
[8] https://scikit-learn.org/stable/