What makes a good movie recommendation? Feature selection for Content-Based Filtering

Maciej Gawinecki[1], Wojciech Szmyd[1], Urszula Żuchowicz, and Marcin Walas[11]

Samsung R&D Institute Poland (SRPOL)
Bora Komorowskiego Street 25C, 31-416 Cracow, Poland
{m.gawinecki,w.szmyd,u.zuchowicz, m.walas}@samsung.com

Abstract. Nowadays, recommendation systems are becoming ubiquitous, especially in the entertainment industry, such as movie streaming services. In *More-Like-This* recommendation approach, movies are suggested based on attributes of a currently inspected movie. However, it is not obvious which features are the best predictors for similarity, as perceived by users. To address this problem, we developed and evaluated a recommendation system consisting of nine features and a variety of their representations. We crowdsourced relevance judgments for more than 5 thousand movie recommendations to evaluate the configurations of several dozen of movie features. From five embedding techniques for textual attributes, we selected Universal Sentence Encoder model as the best representation method for producing recommendations. Evaluation of movie features relevance showed that summary and categories extracted from Wikipedia led to the highest similarity on user perceptions in comparison to other analyzed features. We applied the feature weighting methods, commonly used in classification tasks, to determine optimal weights for a given feature set. Our results showed that we can reduce features to only *genres*, *summary*, *plot*, *categories*, and *release year* without losing the quality of recommendations.

Keywords: Recommender System · Content-Based Filtering · Feature Selection · Feature Weighting

1 Introduction

Selecting interesting movies in TV streaming services can be time-consuming for users due to the increasing amount of available content. Therefore, recommender systems are utilized broadly to assist users in handling information overload by suggesting new similar content. Conventional recommendation methods are classified into Collaborative Filtering (CF), Content-Based Filtering (CBF), and Hybrid systems, combining both methods. Recommending in CF is based on the similarity between users' preferences. In CBF, the retrieval process is driven by the characteristics of the products, providing items similar to items that the user selects or liked in the past. In this paper, we focused on the so-called *More-Like-This* (MLT) version of CBF recommenders that suggests more content similar

to particular items and does not consider user profile. A prerequisite for CBF is the availability of information about relevant content attributes of the items. Attributes in the movie domain mostly comprise of structured information, e.g., genres, release year, and unstructured information, such as plot. The relevance of each feature is not obvious in the context of the recommendation system. Typically, the features are chosen based on their relative usefulness at hand [5,17], not using some external heuristic, e.g. optimization techniques [19]. In a long run, using wrong features can lead to inaccurate recommendations and unnecessary engineering costs, like acquiring a source of data for a feature that does not pay off.

In this paper, we asked: (1) Which feature representation is most effective? (2) Which single feature provides the best recommendations? (3) Do we need all features to get good recommendations? (4) How to combine them to provide the most relevant recommendations? To answer those questions, we crowdsourced a large volume of recommendation relevance judgments. We released part of the collected dataset to the public as a benchmark for evaluation in the movie recommendation domain. For textual features, we compared the quality of recommendations from various novel embedding methods and found the most promising representations. Then, for all the features, we applied well-known feature selection algorithms to find which ones are relevant and which can be omitted without losing recommendation performance. We conclude with a summary of our findings.

2 Related works

There is a substantial body of work on *feature selection* algorithms for machine learning and statistics (see [26] for a survey). However, for most algorithms, it is unclear how to extend them to the case of a recommendation system. For instance, *filter methods* use similarity measures such as Spearman Correlation, to score features based on their information content concerning the prediction task. Yet, filter methods cannot be naturally extended to recommender systems, in which the prediction target varies because it depends both on the input item (selected movie in our case) and on the item under consideration (recommended movie). Ronen et al. [22] addressed this limitation by scoring not single items but the similarity between pairs of items.

The more frequent approach for feature selection in recommender systems is to use domain knowledge and non-systematic trial-and-error method that, as some authors like Colucci et al. [6] admit, is a naïve eyeball technique. Another approach is to run *online evaluation*, where users are asked to assess recommendations from multiple recommenders but are not told where each recommendation comes from [6,13]. While online evaluation can provide the most credible results by simulating real conditions and getting real user's decisions, it does not scale well for several dozen of possible recommender versions to evaluate. To address these limitations, recommenders are evaluated in offline setup against previously collected *relevance judgments*.

There is a number of public datasets with movie ratings, like Netflix [1] or MovieLens [10]. However, those datasets describe whether a movie is a good recommendation for a user rather than for a selected movie. One could argue that two movies are similar, if they were liked by same users. However, users tend to like a variety of films, e.g., both comedies and thrillers, and that does not automatically make them similar. To address this lack, authors of [6,13] collected two datasets[1] from online evaluation of multiple MLT recommenders. There is a risk that subsequent recommender systems generate recommendations that are not present in the dataset. The authors address this problem by ignoring movie pairs without relevance judgments when measuring performance. Research in information retrieval has shown that such an approach may lead to unfair performance results: it may happen that top search results should be treated as relevant but are considered as non-relevant if they were left unjudged, as reported by Webber et al. in [28]. However, it is impractical to obtain relevance judgments for all items. Tonon et al. [27] handle this problem by introducing *iterative pooling*: for each new retrieval system missing relevance judgments are obtained and added to the existing dataset.

In the context of movie recommenders, there has been little research on which features are best at predicting, and the results are often contradictory. For instance, in offline evaluation Soares et al. [24] found that *director* feature alone can provide better recommendations than other features like *actors* and *genres*, while the order starting from the most important (*title*, *genre*, *cast*, *screenwriter*, *director*, and *plot*) is suggested by Colucci et al. in [6]. There have been significantly more systematic research on which *feature representation* provides best recommendations, especially on representing textual content such as a *movie/book plot*. LSI and LDA were evaluated in [2], LSI and Random Indexing in [16], TF/IDF, Word2Vec, GloVe and Doc2Vec in [25], Word2Vec alone in [18] and Doc2Vec in [23]. Their results show that the quality of recommendations depends not only on the topic model used but also on the type and size of data used for training the embedding model.

3 Recommender system used in experiments

To perform experiments, we developed a prototypical recommendation system built using the dataset that we collected. Further details are described in this section.

3.1 Recommending approach

The system used in our experiments was designed to show five unordered recommendations next to a *selected movie* on a user's TV screen. Each movie is represented as a set of encoded features. The system calculates *similarity* between vectors for each feature separately, using suitable distance metrics, and

[1] `http://moviesim.org/`

takes a weighted sum of them. Weights enable us to control how much each feature contributes to the final output distance.

3.2 Movies Dataset

For experiments, we built the Movies Dataset of over 20K movies that were used as the input for the recommender. The movies came from intersecting internal Samsung dataset[2] and Wikipedia dump[3]. We also added additional movie ratings from MovieTweetings[4] [8]. For each title we extracted the following attributes: *release year*, *genre*, *language*, *screenwriter*, *director*, *summary* (merged Wikipedia introduction page and distributor description), *plot* (Wikipedia "Plot" section), *category* (from Wikipedia categories, e.g., "1990s black comedy films" or "Films about psychopaths"), and *popularity* (ratings from IMDb and metacritic.com). Three attributes in the dataset had incomplete values: language (1.5%), director (34.0%), and popularity (70.0%). One entry of Movies Dataset is included as an example in our public repository[5].

3.3 Features representations

For each movie attribute in the Movies Dataset, we developed the following *features representations* and *distance metrics*:

- **Release year**. The intuition is that when a user is looking for an old movie, we should recommend him/her a similarly old movie. However, when a user is looking for an old movie, it doesn't matter if it is from the 1930s or 1950s, but when looking for more contemporary movies, the subjective difference between the 2000s and 2020 movies may seem to be much bigger. To express this intuition, we represent it in a logarithmic form and use the Euclidean distance metric.
- **Language**. The expectation is that when a user is looking for a movie originally spoken in French, they might be interested in other French movies as well. Since a movie can have more than one language assigned, we used Jaccard distance to measure language overlap between two movies.
- **Genre**. For genres, we proposed two representations: a simple sparse label vector with a Jaccard distance metric and Word2Vec embeddings [11], trained over genres co-occurring in our dataset. The latter can capture the perceived similarity between genres, e.g., thriller can be considered to be more similar to action than to cartoon. We applied cosine distance between embedding vectors of movie genres, and we took a mean vector over embeddings for movies with multiple genres.

[2] Accessed on May 25, 2020.
[3] `https://dumps.wikimedia.org`, accessed on August 25, 2020.
[4] `https://github.com/sidooms/MovieTweetings`, accessed on August 27, 2020.
[5] `https://github.com/la-samsung-poland/more-like-this-dataset/sample_movie.json`

- **Category**. We extracted about 60K Wikipedia categories. To express similarity between categories we embedded them using Word2Vec with negative sampling [11]. The network used for machine learning was fed with movie titles as target words and corresponding categories as context words. We used the cosine metric to calculate the distance between vectors.
- **Director and screenwriter**. We suspected that directors and screenwriters often produce movies of a similar style and topic. To express similarity between movies we applied sparse vectors with the Jaccard distance.
- **Popularity**. A user may search for a blockbuster, a movie that is both highly rated and popular, or a niche movie, appreciated by critics but not so popular. To support such use cases, we constructed a vector of popularity indices and averaged ratings among users and critics, where distance is measured with the cosine metric.
- **Summary and plot**. For these textual features we experimented with a number of topic modelling approaches: Doc2Vec [11], LDA [3], LSI with TF-IDF [7], USE [4]. USE is a transformer-based model pre-trained on a variety of NLP tasks with large multi-domain datasets. We did not fine-tune the model on our dataset. Doc2Vec, LSI, LDA models were trained on the collected plot and summary.

4 Evaluation and feature selection methods

To find meaningful answers to questions posed in this paper we ran experiments with the tools and methods described below.

4.1 Comparing recommenders performance

We carried out an *offline evaluation*, where recommendations for a given input movie are compared against ground truth ratings (relevance judgments). For input movies, we have manually selected 153 movies from Movies Dataset (described in Section 3.2). We strove to achieve high diversity in content – from Marvel blockbusters through computer animations, European and Asian arthouse cinema to silent movies. Diversification of evaluation set allows to measure how recommender system performs for different inputs but also describes partially general diversity of recommender. List of all the movies from Evaluation Dataset is available publicly[6]

For each of the selected input movies, a recommender produced five recommendations. Input movies, together with their recommendations, were sent to annotation for collecting ratings. Given the relevance judgments, we were able to calculate metrics to assess and compare recommenders with various configurations of features and features weights.

The order of recommendations in our system was irrelevant because they were displayed in unordered series of tiles. Therefore, using any rank-based metric

[6] `https://github.com/la-samsung-poland/more-like-this-dataset/blob/main/evaluation_set.tsv`

was pointless. Thus, we decided to measure the performance of the system using *Precision@5* defined as:

$$Precision@5 = \frac{\#true\_positives}{\#true\_positives + \#false\_positives} \tag{1}$$

Because of the limited annotation budget and lack of agreement between annotators in some cases, we were not able to collect ratings for all possible pairs. Only explicitly labeled examples were considered during calculations. Hence, the denominator can sometimes be lower than 5. However, comparing systems using just this metric could lead to incorrect conclusions due to differences in the number of rated pairs. For this reason, we introduced another metric, *coverage*:

$$Coverage = \frac{\#(rated\_pairs \cap generated\_pairs)}{\#generated\_pairs} \tag{2}$$

For instance, coverage of 0.3 means that we know whether recommendation is good or bad for only 30% of recommendations returned by the recommender. It would assure us that the two systems were evaluated using a sufficient number of test examples. Low coverage might lead to erroneous interpretation of results. Features relevances can be inferred from comparing recommender systems with different sets of features and features weights.

4.2 Collecting relevance judgments

Evaluating recommender performance requires a binary label that denotes whether a recommendation is good or bad for the selected movie.

To collect relevance judgments, for each movie pair generated by the recommender, we submitted a rating task to the *crowdsourcing system*. To simulate real recommendation context, we started each task with the short introduction: *"Imagine you have been searching for a movie with Smart TV and the TV has recommended you another one in the 'More-Like-This' section. Would you be interested in watching the recommended movie, given the movie you were searching for?"*. For both selected and recommended movies, we showed only the information that would be present in the recommendation application: movie title, release year, summary, and poster.

We asked users to rate recommendations using 5-point scale of answers: *definitely interested*, *rather interested*, *rather uninterested*, *definitely uninterested* or *don't know*. 3-point scale, applied in similar studies [6,13], may result in losing some information due to a *rounding error* [12,20] and thus leave some less known movie pairs unrated.

Given the fixed budget, annotating all possible movie pairs upfront is impractical. To control the cost we employed *iterative pooling* [27]. Each time a new version of a recommender system was tested, we submitted recommendations, generated and not rated yet, to the crowdsourcing system. We also limited the number of annotators per movie pair to two. Those two users had to agree about the rating: whether they rated it positively or negatively. Answers *rather interested* and *definitely interested* were considered a positive rating, while *rather*

uninterested and *definitely uninterested* were considered a negative one. Only in case of disagreement, the movie pair was submitted for the third annotation to decide. If that did not help, i.e., when an annotator answered with *don't know*, the movie pair was not included in the relevance judgments. This strategy offered us a good trade-off between annotation cost and confidence in ratings.

4.3 Selecting optimal weights

To assess the relevance of features and to tune the recommender, we proposed three methods for determining an optimal set of features' weights. Since the order of recommendations is irrelevant, we can describe searching for nearest neighbors as a binary classification problem where the objective is to classify a pair of movies as good or bad recommendations. We experimentally showed that coefficients obtained by feature selection and classification algorithms can be used as weights in CBF. Input to algorithms were movie pairs represented by vectors of normalized distances between consecutive features of both movies.

Since there is no universal feature weighting method that works for all features configurations, we tried three different methods:

- Coefficients of a linear classifier such as Support Vector Machine (SVM) with linear kernel, perceptron, or logistic regression. In [15] similar procedure was developed for a classification task. This set of models works analogously to the presented recommender system. A linear combination of features is calculated. In the classification case, the calculated sum is compared against a fixed threshold to determine an output label. In our system, a linear combination is interpreted as a distance between movies in a pair, and only five movies closest to the input are selected. We used SVM as a representative of linear classifiers based on results from our initial experiments.
- ReliefF [21], a filter method for feature selection and feature weighting [29].
- Mean Decrease Impurity (MDI) [14] of variables in a random forest classifier.

4.4 Finding the best set of features

Our goal was to find a subset of features that are most relevant in predicting the target variable, i.e., whether a recommendation is good or bad. Evaluating recommendations for all possible feature subsets (2^N) is intractable. To address the problem, we used Recursive Feature Elimination (RFE) with SVM, originally proposed by [9] for feature selection in classifiers. RFE is a greedy algorithm that helps find a subset of a *given size* by recursively removing the least important features (i.e., the least informative during SVM classification). To find an *optimal size* of a subset, we combined RFE with 10-fold cross-validation. Since we optimized our recommender towards high precision rather than high recall, we used precision as a scoring function. For the same reason, we penalized the SVM training cost function for returning bad recommendations (false positives errors) more (4 times) than for missing good ones (false negative errors). Once we found an optimal features subset, we trained SVM over it and treated its

coefficients as *optimal weights* for our recommender. Finally, we validated those weights with a recommender against relevance judgments.

5 Experimental results

As a result of crowdsourcing, we collected 15334 annotations from 33 annotators for 6901 movie pairs. Out of those we got 5500 rated movie pairs (for the remaining ones users could not agree about the rating). The split between good and bad recommendations is 2988/2512 (54%/46%). We published 30% of collected ratings as MoreLikeThis dataset[7].

5.1 How to represent features effectively?

We reviewed the set of representation methods for the following features: *genre*, *plot*, and *summary*. We asked how the chosen representation affects recommendation performance, as different techniques have their own characteristics. In the first experiment, we evaluated how those selected features work together with other features (*features collectively*). The results of experiments are depicted in Table 1.

For *plot* and *summary* we tested the following document representation models: Doc2Vec [11], LDA [3], LSI with TF-IDF [7], and USE [4]. We collected ratings for output recommendations achieving coverage of about 0.6 for each model, which was acceptable for comparison of precision. The results showed that USE and Doc2Vec models provide higher precision (0.62 and 0.60 respectively).

In the second part of the experiment we analysed only textual features – plot and summary – evaluating representations *in isolation* as the single-feature recommender. We narrowed methods in this analysis to two the most promising semantic models – USE and Doc2Vec. Sufficient coverage was obtained only for analysis of summary feature (0.64 for Doc2Vec and 0.72 for USE). For remaining configurations (plot, and summary+plot), the gap in coverage between the two represenations was too be big to make any conclusions about them. Still, higher precision for USE in case of summary demonstrates that it outperforms Doc2Vec even though USE was trained on external data and was not even fine-tuned to our Movies Dataset.

All of the above confirms that transformer-based models pre-trained on large datasets are more powerful and generalizable.

Precision for Word2Vec embedding method applied to genre data was relatively lower than sparse vector (Table 1). Using not only exact genre matches, but also similar ones, probably increased the number of false positives.

Overall, our experiments showed that representation strategies can play a crucial role in the final recommendations.

[7] `https://github.com/la-samsung-poland/more-like-this-dataset`

Table 1: Performance of recommenders with various representations for a selected feature: (1) features collectively, (2) features in isolation. Weights were split evenly across features. In the first step, for uninvestigated features we used encoding methods depicted in 3.3, specifically USE representation for *plot* and *summary*, and Word2Vec representation for *genre*.

Feature(s)	Method	Precision	Coverage
Features collectively			
Summary +Plot	Doc2Vec	0.78	0.60
	USE	0.81	0.62
	LDA	0.80	0.57
	LSI	0.79	0.56
Genres	Simple Vector	0.86	0.62
	Word2Vec	0.81	0.62
Features in isolation			
Summary	Doc2Vec	0.45	0.64
	USE	0.86	0.72
Plot	Doc2Vec	0.52	0.19
	USE	0.71	0.71
Summary +Plot	Doc2Vec	0.56	0.24
	USE	0.90	0.61

5.2 Which features are relevant?

In the next experiments, we used only selected representations and distance metrics: the logarithm version of *release year* with the Euclidean distance, averaged Word2Vec vector with the cosine distance for *genre*, a vector of metrics with the Euclidean distance for *popularity*, USE model for textual features *plot* and *summary*, and sparse vectors with the Jaccard distance for *director*, *screenwriter*, and *language* features.

The main goal of this section is to gain an initial insight into the relevance of particular features. We ranked those features from the most relevant using the weight optimization procedure described in Section 4.3. Results are summed up in Figure 1.

All three algorithms are almost consistent – *summary*, *category*, *plot*, and *genre* were important according to all of them. Two of these features, *summary* and *category*, contain a wide range of information, e.g., brief storyline, cast, awards, or influence of the film on popular culture. We suspected them to be correlated with each other and with remaining features. Then, we looked for an optimal subset of features analyzing whether they are redundant (described in Section 5.4).

We evaluated recommender with three different sets of weights obtained by SVM, ReliefF and MDI, and one set of equal weights. Results are available in

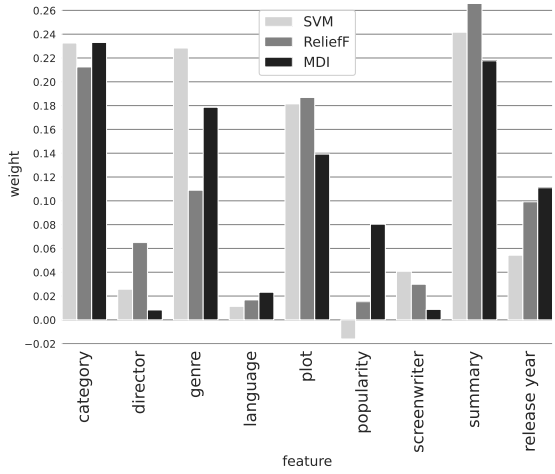

Fig. 1: Normalized (sum up to 1) weights for each feature obtained by SVM, ReliefF, and MDI.

Table 2. Coverage for each recommender was above 0.60 which allowed us to make further conclusions.

Precisions of recommenders with weights optimized by all methods are similar to each other and exceptionally higher than in the equal-weights scenario. That confirms the effectiveness of SVM, ReliefF, and MDI as weights selection algorithms.

Table 2: Evaluation of the recommender with weight obtained by SVM, ReliefF and MDI compared with setting all weights to the same value.

	Precision	**Coverage**
SVM	0.92	0.65
ReliefF	0.91	0.63
MDI	0.90	0.66
Equal weights	0.81	0.62

Some features turned out to be more informative, i.e., they are better at *discriminating* good recommendations from bad ones. However, we do not know which provide better recommendations.

5.3 Which single feature provides best recommendations?

To answer this question, we evaluated six recommender systems, each consisting of a single movie feature. We intentionally did not evaluate recommendations created by *language*, *popularity* and *release year* features. By common sense, we assumed that these features might be useful but cannot create standalone systems. Performance of single-feature recommenders is shown in Table 3.

Table 3: Precision and coverage for single-feature recommenders.

	Precision	**Coverage**
Plot	0.72	0.70
Genre	0.77	0.39
Category	0.87	0.53
Summary	0.87	0.72
Director	0.64	0.37
Screenwriter	0.65	0.35

The results confirmed our findings from Section 5.2: *category* and *summary* are great standalone features while *plot* and *genre* give slightly worse but still decent performance.

5.4 Do we need all features to get good recommendations?

We used RFE with cross-validation (Section 4.4) to find the smallest subset of features. The results are shown in Figure 2. It can be observed that RFE classification precision for five and nine features is almost the same (0.828 and 0.820 respectively). It demonstrates that we can remove four features and still get the best recommendations for a given setup. The best subset contains the following features (together with their optimal weights): *genre* (0.37), *plot* (0.20), *summary* (0.20), *categories* (0.20), and *release year* (0.03). The validation of this configuration with a recommender got 0.93 precision for 0.55 coverage. The validation of configuration with all features got the precision of 0.92 as well with coverage of 0.65.

If we compare the recommendation performance of this combination with the performance of single features (see Section 5.3), we can see that no feature alone can achieve such a high precision. The best single features, *summary* or *category*, have precision of 0.87. The results also indicate that it pays off to engineering features with more complex representations (*summary*, *plot*, and *categories*).

The remaining features: *director*, *languages*, *popularity* and *screenwriter* were found redundant.

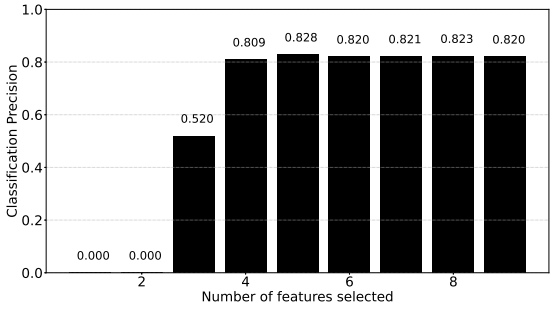

Fig. 2: The results for RFE with SVM and 10-fold cross-validation over relevance judgments.

5.5 Are relevance judgments credible?

We asked whether collected relevance judgments were *credible* and *consistent*.

We pushed random movie pairs and regular recommendations to the same group of annotators. The distributions of ratings for both recommendation types are presented in Figure 3. It showed that random recommendations were mostly rated as bad ones, whereas regular recommendations as good ones. That is in line with our intuition and demonstrates that our annotators provide credible ratings. That gives us trust in the results of the experiments.

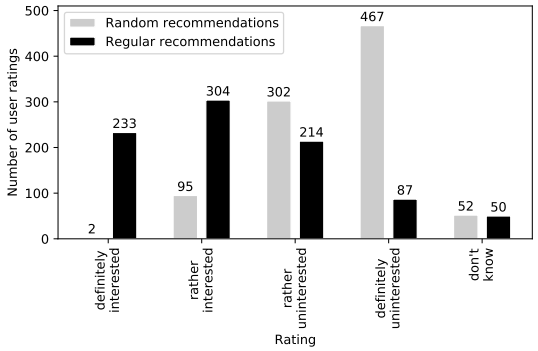

Fig. 3: Distribution of user ratings for regular and random recommendations in the first round of ratings collection.

Additionally, we checked whether users rated the same movie pair consistently and found that 70% of movie pairs had a complete agreement, i.e., all annotators agreed that a recommendation is good or bad. Remaining movie pairs had $\frac{2}{3}$ agreement. This demonstrates that ratings were relatively consistent across users.

6 Conclusions

In this paper, we asked what features of a movie make it a good recommendation for another movie.

We researched methods for feature representation. We found that various representation algorithms applied to the same movie attribute may result in different recommendation performance. For instance, changing movie *summary* or *plot* representations from Doc2Vec or LSI into Universal Sentence Encoder can increase recommendation precision significantly.

We also studied which single feature is most informative and can provide the best recommendations. We found that users rate recommendations generated solely from a movie *summary* or *category* higher than when using other features. We also found that combining these features with others improves recommendations quality, but not significantly. This suggests there is potential for single-feature recommenders if the feature is represented properly.

We also looked for the smallest set of features with high recommendation performance. We found that using only five features (*genres*, *plot*, *summary*, *categories*, and *release year*) we can provide as good recommendations as using all nine proposed features. Surprisingly, other features such as movie *director*, *screenwriter*, or *language*, often mentioned in the literature [6,13], were found redundant. Removing those features can shorten recommender response time, save storage space, and cut costs of acquiring data for those features.

Machine learning has a long-standing list of methods for feature ranking, weighting, and selection. We have shown that by representing a recommendation task as a classification problem, we can apply those methods for content-based recommenders. We also collected and released a large dataset of movie-recommendation ratings. We hope the dataset will encourage and enable future research in this domain.

Acknowledgments

We would like to thank Wojciech Smołka for his substantial help in implementing the recommender and Kaja Pękala for work in building datasets. We are also grateful to Artur Rogulski for editorial support.

References

1. Bennett, J., Lanning, S., et al.: The Netflix Prize. In: Proceedings of KDD cup and workshop. vol. 2007, p. 35. New York (2007)
2. Bergamaschi, S., Po, L.: Comparing LDA and LSA topic models for content-based movie recommendation systems. In: International Conference on Web Information Systems and Technologies. pp. 247–263. Springer (2014)
3. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. Journal of Machine Learning Research **3**(Jan), 993–1022 (2003)

4. Cer, D., Yang, Y., Kong, S.y., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al.: Universal Sentence Encoder. arXiv preprint arXiv:1803.11175 (2018)
5. Chen, H.W., Wu, Y.L., Hor, M.K., Tang, C.Y.: Fully content-based movie recommender system with feature extraction using neural network. In: 2017 International Conference on Machine Learning and Cybernetics (ICMLC). vol. 2, pp. 504–509. IEEE (2017)
6. Colucci, L., Doshi, P., Lee, K.L., Liang, J., Lin, Y., Vashishtha, I., Zhang, J., Jude, A.: Evaluating Item-Item Similarity Algorithms for Movies. In: Proceedings of the 2016 CHI conference extended abstracts on human factors in computing systems. pp. 2141–2147 (2016)
7. Deerwester, S., Dumais, S., Landauer, T., Furnas, G., Beck, L.: Improving information-retrieval with latent semantic indexing. In: Proceedings of the ASIS Annual Meeting. vol. 25, pp. 36–40 (1988)
8. Dooms, S., De Pessemier, T., Martens, L.: Movietweetings: a Movie Rating Dataset Collected from Twitter. In: Workshop on Crowdsourcing and Human Computation for Recommender systems, CrowdRec at ACM RecSys. vol. 2013, p. 43 (2013)
9. Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene Selection for Cancer Classification using Support Vector Machines. Machine Learning **46**(1-3), 389–422 (2002)
10. Harper, F.M., Konstan, J.A.: The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) **5**(4), 1–19 (2015)
11. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning. pp. 1188–1196 (2014)
12. Lehmann, D.R., Hulbert, J.: Are Three-Point Scales Always Good Enough? Journal of Marketing Research **9**(4), 444–446 (1972)
13. Leng, H., De La Cruz Paulino, C., Haider, M., Lu, R., Zhou, Z., Mengshoel, O., Brodin, P.E., Forgeat, J., Jude, A.: Finding Similar Movies: Dataset, Tools, and Methods (2018)
14. Louppe, G., Wehenkel, L., Sutera, A., Geurts, P.: Understanding variable importances in forests of randomized trees. In: Advances in Neural Information Processing Systems. pp. 431–439 (2013)
15. Mladenić, D., Brank, J., Grobelnik, M., Milic-Frayling, N.: Feature selection using linear classifier weights: interaction with classification models. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 234–241 (2004)
16. Musto, C., Semeraro, G., de Gemmis, M., Lops, P.: Learning Word Embeddings from Wikipedia for Content-based Recommender Systems. In: European Conference on Information Retrieval. pp. 729–734. Springer (2016)
17. Nasery, M., Elahi, M., Cremonesi, P.: Polimovie: a feature-based dataset for recommender systems. ACM (2015)
18. Nguyen, L.V., Nguyen, T.H., Jung, J.J.: Content-Based Collaborative Filtering using Word Embedding: A Case Study on Movie Recommendation. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems (ACM RACS). pp. 96–100. ACM (2020)
19. Odić, A., Tkalčič, M., Tasič, J.F., Košir, A.: Predicting and Detecting the Relevant Contextual Information in a Movie-Recommender System. Interacting with Computers **25**(1), 74–90 (2013)
20. Preston, C.C., Colman, A.M.: Optimal number of response categories in rating scales: reliability, validity, discriminating power, and respondent preferences. Acta Psychologica **104**(1), 1–15 (2000)

21. Robnik-Šikonja, M., Kononenko, I.: An adaptation of Relief for attribute estimation in regression. In: Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97). vol. 5, pp. 296–304 (1997)
22. Ronen, R., Koenigstein, N., Ziklik, E., Nice, N.: Selecting content-based features for collaborative filtering recommenders. In: Proceedings of the 7th ACM Conference on Recommender Systems. pp. 407–410 (2013)
23. Singla, R., Gupta, S., Gupta, A., Vishwakarma, D.K.: FLEX: A Content Based Movie Recommender. In: International Conference for Emerging Technology (INCET). pp. 1–4. IEEE (2020)
24. Soares, M., Viana, P.: Tuning metadata for better movie content-based recommendation systems. Multimedia Tools and Applications **74**(17), 7015–7036 (2015)
25. Suglia, A., Greco, C., Musto, C., De Gemmis, M., Lops, P., Semeraro, G.: A Deep Architecture for Content-Based Recommendations Exploiting Recurrent Neural Networks. In: Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization. pp. 202–211 (2017)
26. Tang, J., Alelyani, S., Liu, H.: Feature Selection for Classification: A Review. Data classification: Algorithms and Applications p. 37 (2014)
27. Tonon, A., Demartini, G., Cudré-Mauroux, P.: Pooling-based continuous evaluation of information retrieval systems. Information Retrieval Journal **18**(5), 445–472 (2015)
28. Webber, W., Park, L.A.: Score adjustment for correction of pooling bias. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 444–451 (2009)
29. Wettschereck, D., Aha, D.W., Mohri, T.: A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. Artificial Intelligence Review **11**(1-5), 273–314 (1997)