Machine Learning

Jetze Schuurmans

December 2017

Abstract

In this report we create a Recommender System for movies using different type of Machine Learning methods. First, we employ unsupervised learning to extract different types of users and check the stability of the found clusters. Secondly, we apply Matrix Orientated Path Finding on an information network to extract the number of paths, for different path types. Thirdly, we use Random Forests and Support Vector Machines to learn which path types are relevant and how their counts might contribute to the recommendations. Finally, we assess the performance in a Top-N recommendation by means of the One-Plus-Random methodology. We conclude that the path type created by applying unsupervised learning on the users, result in better recommendations. However, as it does not add new information, using all other path types give similar results. Furthermore, the best Recommender System is based on Random Forest, rather than Support Vector Machines.

Contents

1 Introduction

The amount of information available (online) continues to grow exponentially. We are beyond the point where one person can assess everything, let alone study everything. This is where Recommender Systems (RS) come in. A RS is an algorithms that suggest items to users. The goal of the RS is to find and suggest the most relevant items for a particular user.

In order to train a RS we need data. Luckily the information stream goes both ways, i.e. with the internet we also get more information about the user. In this report we try and leverage this information by forming clusters of similar users. Our research question is whether this information adds value to the recommendations made. We analyze a movie data set and our goal is to suggest the most relevant movies to users. We use the MovieLens 1M data set for the users and the ratings. For the information about the movies we use the MovieLens 20M dataset.

Our approach is similar to that of Wever and Frasincar (2017), however it differs significantly in the following ways. First, we include information about the user. Apply the unsupervised learning methodology of Dolnicar and Leisch (2010) on the user information, to create reproducible clusters and analyze if we can leverage this information to create better recommendations. Secondly, we include information about disliked movies. Thirdly, we apply several Support Vector Machine (SVM) based techniques to learn ranking functions and compare their results with Random Forest (RF) based ranking functions.

In the next section we describe the data collection, cleaning and transforming techniques used, together with the construction of the training and test sets. In Section 3 the methods used to analyze the data, construct the RS and evaluate the performance are discussed. Thereafter, we present the results. In Section 5 we discuss our conclusion and reflect on our report with some discussion points.

2 Data

The data used in this report is open source data from MovieLens. In this section we start with the way we gathered the data. Followed by discussing the way we cleaned, transformed and merged the data. Closing with the way we create a training and a test set.

2.1 1M MovieLens

The first sets of data contains the user information and ratings, this is the 1M MovieLens datasets. Which can be collected from GroupLens[1], a project from the University of Minnesota. It contains 1,000,209 ratings of approximately 3,900 movies made by 6,040 MovieLens users. There are two data sets, one on the users (`UserID`, `Gender`, `Age`, `Occupation`, `Zip-code`) and one on the ratings the user gave to certain movies (`UserID`, `MovieID`, `Rating`, `Timestamp`). These can easily be merged by the `UserID`. Before we do this we dismiss the variables, `Timestamp` and `Zip-code`, and discuss the variables. The variables `Gender`, `Age`, `Occupation` are categorical variables, where `Age` is grouped in seven categories and `Occupation` in 21 classes.

The `Rating` is transformed in a new variable `Like`, which is +1 if the rating was 4 or 5 (the highest rating) and -1 if the rating was 1 (the lowest rating), 2 or 3. When `Like` equals 0, the movie is unrated and do not know whether the user likes this movie or not. This means we can recommend this movie to the user, but be cannot verify the relevance of the recommendation.

2.2 20M MovieLens

The second set of data contains the basic information about the movies rated by the users. It is collected from GroupLens as well.[2] The data set contains the following information: `MovieID`, `Title`, `Genre`. The variable `Genre` is a concatenation of all relevant genres for the respective movie. We transform this variable such that for each genre a new column is created that equals one if it is relevant for a certain movie. The `Title` also contains the year of release. This information is separated from the `Title` and put in its own column. The variable `MovieID` is used to merge the movie data set with the user data set.

[1] grouplens.org/datasets/movielens/1m/
[2] grouplens.org/datasets/movielens/20m/

We use the ratings of the 1M data set, because the information about the user is more extensive in this set than it is in the 20M set. The 20M set however, contains more up-to-date description of the Movies. Therefore we use the 20M set for the movies.

2.3 Transformations and sampling

In this section we discuss how we transform the data such that we can use unsupervised learning to cluster similar users together and how we construct the training and test set for the supervised learning algorithms.

2.3.1 Unsupervised Learning

The unsupervised learning method used to categorize users. Therefore, we aggregate the data per user. For each user we know its `Gender`, `Age group`, `Occupation type` and which movies he or she likes and dislikes. We aggregate the `Like` variable by taking the average of all transformed ratings, -1 and +1. Furthermore, we count per genre how many liked and disliked movies the user has for that genre. Resulting in a net score ($= \#liked - \#disliked$) per genre. This might add users together who like similar genres. Finally, we standardize the data by representing each category as a separate column, with a 1 if the user belongs to the respective category and scale the other variables with their respective standard deviation, such that Euclidean distances are comparable between variables.

2.3.2 Supervised Learning

In order to get the top-N recommendations we use supervised learning to create ranking functions. We employ two types of supervised learning algorithms, Random Forests and Support Vector Machines. The RF ranking functions are created per user, while the SVM ranking functions is a general ranking functions. In order to learn the ranking functions we create a training set using only users who have 25 or more positive ratings (leaving 4732 users).

For the RF, we start with a set in which the disliked movies ($s_{ui} = -1$) are combined with the unrated movies ($s_{ui} = 0$). The test set used for the One-Plus-Random (OPR) methodology consists of 10 liked and 1000 unliked randomly selected movies, per user. We include all remaining liked movies in the training set. Ostuni et al. (2013) showed that the number of disliked movies in the training set needs to be proportional to the number of liked movies. The optimal ratio is two disliked movies for every liked movie.

When categorizing the disliked and unrated movies as one class, we lose information in two ways. First, the information about disliked movies is lost. Secondly, an unrated movies might be liked. Grouping this movie together with disliked movies might distort the model. Therefore, we construct a training set where all disliked movies keep their label -1. The training set contains disliked movies as well as all remaining liked movies. The number of unrated movies in the training set remains two for every liked movie. The results of Ostuni et al. (2013) indicates that a slight increase in the ratio of liked to unliked movies still has a decent performance. The model is tested on a different test sets. The test set for the Dislike RF, is constructed with the ratio of 1:100 with liked and unranked movies.

The SVM ranking functions is trained and evaluated on the sets where we aggregate the disliked and unrated movies in the same class. The test set is exactly the same as the first test set used for the RF ranking function. However, the training set is smaller, due to computational limitations. We follow Coussement and Van den Poel (2008) in creating a balanced training set (sampling 5000:5000) and using an unbalanced test set (1:100, the same used for most RF). Previous applications of SVM show decent results when training on balanced sets and empirical scenarios having unbalanced cases.

3 Methodology

3.1 Clustering

We start the methodology by describing the way we form clusters of similar users. The data discussed in the previous section is used, which contains information about the user, its general rating behaviour and the genres he or she likes.

Based on this information we try to find clusters of similar users. However, we do not know how many clusters there might be. It might even be possible that there are no natural clusters in the data set. We follow the methods proposed by Dolnicar and Leisch (2010) to determine what the optimal number of clusters is and whether the proposed clusters are natural clusters or reproducible clusters. Results of cluster algorithms depend on the the sample randomness and algorithm randomness. We try to minimize the influence of both by bootstrapping the data set and using several random starts of the cluster algorithm.

We start by drawing 2B bootstrap samples where each sample equals the training sample size. Following Dolnicar and Leisch (2010) we use 200 bootstrap samples. For each bootstrap sample the clustering algorithm, k-means, is applied with several values of k, the number of clusters. For each we use s=10 random starts and keep the best clustering. As a first performance measure the Calinski-Harabasz index is calculated for different number of clusters. The Calinski-Harabasz index is widely used to assess the performance of clusterings. It equals the variance between cluster centers divided by the sum of within cluster variances. Since we want to maximize the variance between clusters and minimize the variances within clusters, we want to maximize the Calinski-Harabasz index. The boxplots of the Calinski-Harabasz give us an indication of the distribution of the indices.

Additionally the adjusted Rand index is used to evaluate the stability of the clusterings. The former is done by pairing the 2B bootstraps and calculating the adjusted Rand index on these pairs, resulting in B indices. We consider each bootstrap sample only once in the calculation of the adjusted Rand index in order to guarantee independence. In order to evaluate the reproducibility of the clusters, the boxplots of the adjusted Rand indices is created. Additionally the kernel density estimates of the adjusted Rand indices are plotted. The adjusted Rand index is an agreement measurement between two cluster results. We prefer to use this measure, above for example Kullback-Leibler divergence or Euclidean distance between centers, because the adjusted Rand Index makes no further assumptions about the segmentation algorithm. We use the adjusted Rand Index, because this measure corrects for random agreements between clusters. The adjusted Rand Index is an agreement measure and for reproducibility we want to see similarity between the bootstrap results, therefore we want a larger adjusted Rand Index with its density close to 1.

3.1.1 Initialization

In order to apply the bootstapping methodology of Dolnicar and Leisch (2010) we must have an idea of how many clusters might exist in the data. We start with agglomerative hierarchical clustering. Which starts with each point in a cluster and merges the clusters closest together until all points are in one cluster. The process can be visualized with a dendrogram, from which we can infer what number of clusters might make sense. As input we need to calculated the distances between the points, we use euclidean distances over the standardized data. Ward's minimum variance method is used to find compact, spherical clusters. After merging two clusters, Lance–Williams dissimilarity update formula is used.

Once we have a range of possible clusters we employ the methodology discussed above. After we get the optimal number of clusters with respect to reproducibility, we use k-means with the euclidean distance and 100 random starts to get the final clustering. The number of random starts needs to be sufficiently large, because a bad initialization might end up in local optima. We use euclidean distances and means as centers, because this makes sense given the data and applied transformations discussed in the previous section.

3.2 Recommender Systems

The purpose of a Recommender System is to suggest relevant items to users. In order to increase the performance we make use of entities, information about the users or the items. We base our RS on the methodology of Ostuni et al. (2013), which tries to find the number of connections between items following predefined paths in a graph. Therefore, we make use of graph theory to store and analyze connection between users, items, and entities. In the information graph, the users, items, and entities are represented by nodes, while all connections are represented by edges. The information graph is used to find similarities between items and users. In order to search for similarities in an efficient way, we need to know which notes are connected by what type of edges. To do this, we employ Matrix Oriented Path Finding, previously used by Sun et al. (2011) for semantically enhanced web search.

3.2.1 Matrix Orientated Path Finding

The Matrix Orientated Path Finding (MOPF) approach is used to find all path instances in the information network belonging to each defined path type in the information schema. MOPF makes use of the adjacency matrices, $W_{A_iA_j}$, with n number of rows corresponding to nodes of type A_j and m number of columns corresponding to nodes of type A_j. The elements of an adjacency matrix, w_{kl}, equals one if there is an edge between nodes k and l, otherwise it is zero.

The commuting matrix M_P contains the number of path instances $p \in P$, where $P = A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_j$ is a specific meta path. $M_P = W_{A_1A_2}W_{A_2A_3}\cdots W_{A_{l-1}A_l}$, where $M_P(i,j)$ indicates how many path instances exist between node $x_i \in A_1$ and $y_j \in A_l$, following meta path P. We combine commuting matrices to get the path count matrix, which represents the number of paths between users and items.

$$PC = [vec(M_P^1) \cdots vec(M_P^o)]$$

Each row $pc_{u,i}$, with row number $(U * (u-1) + i)$, in this vector represents the path counts from user u to item i, for the different path types. We can group this in the set $[s_{u,i}, pc_{u,i}]$, where $s_{u,i}$ equals one if user u likes item i, zero if we have no information, and -1 if the user dislikes the item.

We implement MOPF instead of breadth-first search, depth-first search or other traditional path finding techniques, because MOPF has several advantages. The first being, that only path instances of predefined path types can be found. Secondly, it does not keep track of which nodes have already been visited. Finally, it is computationally very easy to find paths this way, resulting in decreased computation time.

In general, two nodes can be similar if there exist one or more connections between them in the information network. Nodes can be connected by a direct link, or a concatenation of multiple links called a meta path. Note that a meta path defines a concatenation of links on the schema level, whereas a path denotes an instance of a meta path. To show how meta paths can be used to find promising movies for user i using the information network we give the following example:

$$User2 \xrightarrow{\text{likes}} Movie1 \xrightarrow{\text{attributed}} Genre3 \xrightarrow{\text{attributed to}} Movie4$$

The fact that User 2 and Movie 4 are connected by a meta path implies that Movie 4 might be of interest to user 2. The extent to which this path is useful for recommendations is estimated using Random Forest and Support Vector Machines.

3.3 Random Forest

Given the training set we want to find a ranking function: $f : \mathbb{R}^m \rightarrow \mathbb{R}$, such that $f(pc_{u,i}) \approx s_{ui}$. In order to create a ranking function f, we make use of the Random Forest methodology, proposed by Breiman (2001) and the Support Vector Machines, proposed by Cortes and Vapnik (1995). We create a unique ranking function f_i for each individual using RF, while we construct one general ranking function f when applying SVM. In this subsection we discuss Random Forests and thereafter we discuss SVM.

A RF is an ensemble method, which has shown to be successful on many occasions. Each tree in the RF is a CART model, which are prone to overfitting. While individual CART models can be improved by stopping the splitting early, or pruning back the tree after fitting, RF take a different approach. RF reduce overfitting by limiting the number of possible splits at each node. To do this, each time a split has to be made, the model randomly selects a subset of size *mtry* from the independent variables and chooses the optimal split within this subset. This creates an opportunity to find other, maybe less pronounced relations in the data. Additionally, RF makes use of bagging. In this procedure, multiple bootstrap samples with replacements are drawn from the data set. Each time, a CART model is learned on the bootstrapped sample, and at the end all models are aggregated. This reduces overfitting on the training set, because each time the model is learned on a different subset of the data. The in-bag subset has a size two thirds of the full training set. This means, that one third of the observations in the training set are not used for growing the tree. These observations are defined as out-of-bag (OOB), and provide a good development set for monitoring key statistics.

3.3.1 Variable importance

One of the most important statistics is the variable importance. Variable importance is a difficult concept, because the importance of a variable may be conditional on its interaction with other variables (Liaw et al., 2002). In RF, variable importance is estimated by randomly permuting values of an independent variable in the OOB sample, and assessing the change in prediction performance when all other independent variables are untouched. The rationale behind this technique is that if the independent variable had a lot of explanatory power, this is removed by the permutations and the decrease in performance should be large. In contrast, if the explanatory variable had not much predictive power before the permutations, the decrease in performance should also be small. The variable importance of a variables in a tree is estimated as the increase in the within Means Squared Error (MSE) after permuting the value in the variable. The overall variable importance is defined as the increase in within MSE over the whole forest. The variable importance values have no direct meaning and there are no hard rules for determining when a variable should be included or when it should be excluded (Strobl and Zeileis, 2008). In other words, some interpretation is required in order to select variables based on variable importance scores.

3.4 Support Vector Machines

Support Vector Machines are known for their great theoretical generalization result. We make use of two types of SVM algorithms. First we use a similar approach as we took with Random Forest, where we used regression trees. The SVM methodology also has a regression variant, Support Vector Regression (SVR). The benefit of aggregating regression trees is that, because the training values of the dependent variable are -1, 0 or 1, the predictions are also between 0 and 1. This is not the case in SVR, where the outcomes are real values. Because we are interested in the ordering of the out of sample predictions we do not expect this to be a problem. Furthermore, we use another SVM method, that enables us to compare results. The second method is the SVM classification methodology. Again, we are interested in orderings not in class predictions. Therefore, we use the decision values to approximate the probability of belonging to a class.

3.4.1 Tuning Hyperparameters

When applying any SVM algorithm there is the choice of the kernel function and the need to tune the hyperparameters. For the kernel function we make use of the Radial Basis Function (RBF) kernel for the same reasons Coussement and Van den Poel (2008) give. First and foremost, the RBF kernel gives us the possibility of exploiting non-linearities in the data. Secondly, Keerthi and Lin (2003) report that the linear kernel has similar performance as the RBF kernel and Lin and Lin (2003) conclude that the sigmoid kernel behaves like the RBF kernel for certain parameters. Thirdly, the number of hyperparameters is larger for the polynomial kernel than for the RBF kernel. Thereby, requiring more cross-valitions to tune the hyperparameters. Finally, the polynomial kernel might encounter numerical difficulties. For a large degree it can get stuck at 0 or wonder off to infinity.

When using the RBF kernel there is the γ and in any soft-margin SVM we have the cost parameter, C, which need to be optimized. We use k-fold cross-validation in order to tune these hyperparameters. For SVR we use MSE as performance measure and for classification we use the accuracy rate. We apply a grid-search on the values $\gamma = (2^{-16}, 2^{-14}, 2^{-12}, 2^{-10}, 2^{-8})$ and $C = (2^{-2}, 2^{0}, 2^{2}, 2^{4}, 2^{6})$ with 5-fold cross-validation. The values of γ are chosen around $\frac{1}{n}$, where n equals the number of training observations, and the values of C are chosen such that they are close to the values which worked well for Coussement and Van den Poel (2008) and our previous applications.

SVM does not have a clear way to interpret the variables or assess their importance. Therefore, the type of paths used in the final model depend on the variable importance and model performance measured for the Random Forests.

3.5 Evaluation measure

We evaluate the performance of the RS using the One-Plus-Random methodology (Cremonesi et al., 2010; Bellogin et al., 2011). We consider all four and five star ratings to be positive, while all other ratings are regarded as unobserved. Let T be the set of all user-item ratings. First, the positive ratings

from T are split into a train set, T_{train}, and a test set, T_{test}. We select ten positive observations from T to be in T_{test} and put all other observations in T_{train}.

After training the model on the training set, it is tested by evaluating how good the model can select a relevant item from T_{test} for a specific user when random irrelevant items for that user are added. To do this, we iterate over each user-item combination in T_{test}. Each time, we create a temporary set consisting of the selected item from T_{test}, together with 100 randomly selected irrelevant items for that specific user. These must be items that do not appear in the training set.This results in a sample of 101 items, of which we assume that only the test item is relevant. The items are ranked according to their predicted score, and the top-N items are recommended to the user. If the test item is among the recommended items, it is counted as a hit. The performance is measured using recall@N (R@N):

$$\text{R@N} = \frac{\#\text{relevant items recommended}}{\#\text{relevant in } T_{test}} = \frac{\#hits}{|T_{test}|},$$

where N represents the number of items recommended to the user. This metric measures how much of the relevant items in T_{test} have been recommended to the user. We average the R@N over all users. It is worth mentioning that this methodology tempts to underestimate the performance of the RS, because some of the unliked items might actually be relevant to the user (Cremonesi et al., 2010). We try to decrease this bias by including some disliked movies that are not in the training set, to be in the test set. Note that for testing the Dislike RF we are including all disliked movies in the training set. Therefore, they cannot be in the test set. Making this test set more biased towards zero (worse top-N recommendations). The results of the ranking functions can therefore not be compared with the Dislike ranking function. However, the results of the Dislike ranking function might shed some light on the bias in the OPR methodology when only unrated movies are included.

4 Results

4.1 Clustering

We start the clustering with determining the range of sensible number of clusters. As discussed in the previous section we use agglomerative hierarchical clustering.The result can be plotted in a dendrogram, see Figure 1. The vertical axis in a dendrogram represent the dissimilarity between two clusters. The larger the height between two merges, the more stable the solution. We can see in Figure 1 that around height of 20 the clusters changes with a smaller height difference.As a result we use the bootstrap methodology on 2 till 13 clusters.

The stability and reproducibility of the number of clusters is determined by the bootstrapping method discussed in Section 3.[3] The bootstraps give us 100 adjusted Rand indices and 200 Calinski-Harabasz indices. In Figure 2 we plot the boxplots of the adjusted Rand indices, Figure 3 shows the density kernels. From these figures we can see that the optimal number of clusters with regards to the reproducibility (adjusted Rand Index densities skew towards 1) is 4 or 10.

Figure 4 presents the boxplots of the Calinski-Harabasz indices. When we want to select the optimal number clusters based on dense and well separated clusters, we are looking for the largest Calinski-Harabasz index. The highest mean Calinski-Harabasz index is acquired by using k-means with 9 or more clusters.

When combining these results we can choose either 4 or 10 clusters. We choose to work with 10 as this might segregate the users more and the probability that the users within the same cluster likes similar movies is larger. Now that we have determined the number of clusters we can run the k-means algorithm. After running k-means with 100 random initializations we the optimal clustering is chosen. The results of the final clustering are discussed in the Appendix, Section 6.

[3]We implement this using `flexclust` package in combination with `parallel`. The latter enables us to use multiple cores. While the default in `R` is only a single core, we use seven on our machine. This should significantly decrease the computation time and enables us to use the same number of bootstrap samples and double the number of random starts for k-means (from 5 to 10), compared to Dolnicar and Leisch (2010).

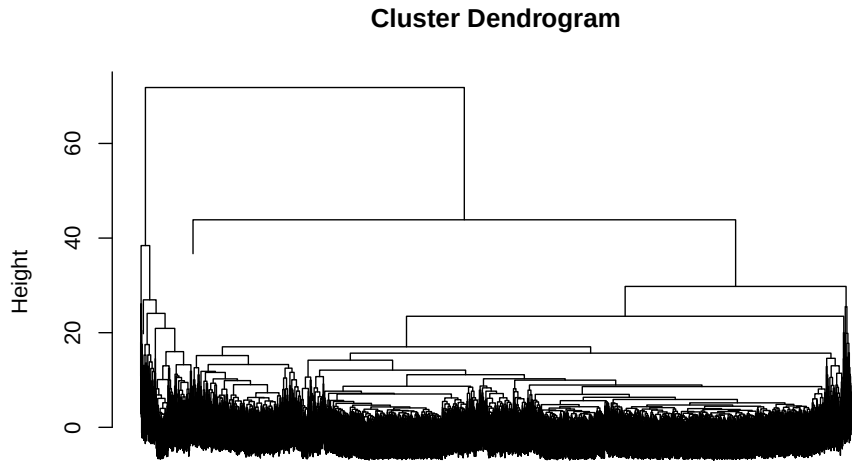

Figure 1: Dendrogram on the users

4.2 Random Forest

4.2.1 Variable importance

We start the implementation of Random Forest with the variable selection.[4] After training an RF for each user we obtain the percentage increase in Mean Squared Error (MSE) for each variable and average these over all users. The obtained results are presented in the first column of Table 1. We clearly see a distinction between two groups of variables with different orders of importance. The variables Genre and Year seem to be less important than the other variables. The number of paths of path type Year is clearly not a good indication whether a user likes a movie, since there are different types of movies released within a year. However, users might like vintage movies, which could be considered as a type of movie. For future research we suggest to try and group these movies, e.g. by decade. The irrelevance of the path type Genre comes more to a surprise. This conveys more information about the type of movie and can highlight similarities between movies. However, this information might already be incorporated in the path type Group. Since we used information about the liked genres in combination with other information about the user.

Variable	% Increase MSE	% Increase MSE Aero
Likes	8.81	9.53
Genre	2.23	-
Year	3.80	-
Group	7.28	7.55
Gender	6.73	6.90
Age	7.10	7.06
Occupation	7.03	7.19

Table 1: Average Increase in MSE in OOB predictions when variable is permuted.

The path types Genre and Year might add more noise than signal to the model. Therefore we consider a model without these variables. We refer to this model as Aero, because it is a more streamlined version of the complete model. After excluding Genre and Year we calculate the average percentage increase in MSE and tabulate the results in the second column of Table 1. Furthermore,

[4]Throughout this paper the `R` package `randomForest` is used in combination with `parallel` for more efficient code.

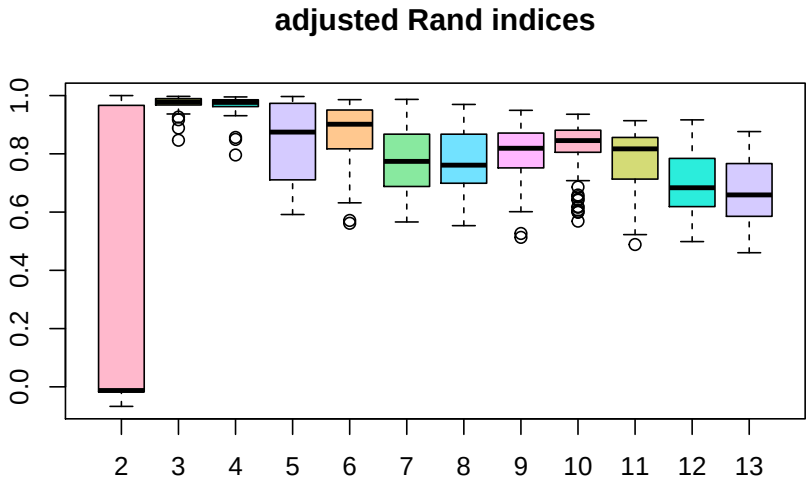

Figure 2: Boxplots of the adjusted Rand indices

we can see that in both models the path type Likes is the most important. We also construct the Collaborative Filtering (CF) model, as a baseline model. In the next section we compare the performance of the Complete and the Aero models with the baseline CF model.

4.2.2 Recall@N

In Table 2 we can see that the exclusion of variables results is worse performance for the Random Forest ranking functions. The speed with which the Recall@N increases with N is similar for the first five RFs. These RF ranking functions show us that the best results are obtained if we utilize all path types. Therefore, we use all path types for the Dislike RF and SVM ranking functions. The Group ranking function incorporates information about the likes (used in CF) and groups (obtained from cluster analysis). We can see that its results are in between those of Aero and CF. When we exclude the group type of paths from the Complete ranking function we can see that there is hardly a drop in performance. Concluding that the groups add information, but no new information. Making the path type created with unsupervised learning a substitute not a complement for existing information.

	Random Forest						Support Vector	
N	Complete	Aero	CF	Group	excl. Group	Dislike	Classification	Regression
5	59.0	47.5	39.6	42.6	58.9	48.5	47.5	45.6
10	75.6	65.4	53.4	58.6	75.5	61.0	65.4	64.1
15	83.6	75.5	63.2	69.1	83.4	67.2	75.8	74.4
20	88.5	82.3	71.3	76.5	88.4	70.8	82.6	81.4
25	91.5	87.0	78.3	82.2	91.6	73.1	87.4	86.3

Table 2: Recall@N in percentage for the RF and SVM ranking functions

The Dislike column in Table 2 presents the results of the RF ranking function where we labeled disliked movies as -1. The test set of Dislike only contains liked and unrated movies. Although we might expect that including this addition information in the training set might give a better Recall@N, we can not draw conclusion when comparing the values of the Dislike RF with the other ranking functions, as it is created using a different test set. When Dislike is tested on the second data set, it works for small N. However the Recall does not increase with N as fast as it does for the other algorithms. It might be possible that the Dislike algorithm become better at recognizing disliked movies. Recall@N only cares about the one liked movie. As mentioned earlier, Recall@N is

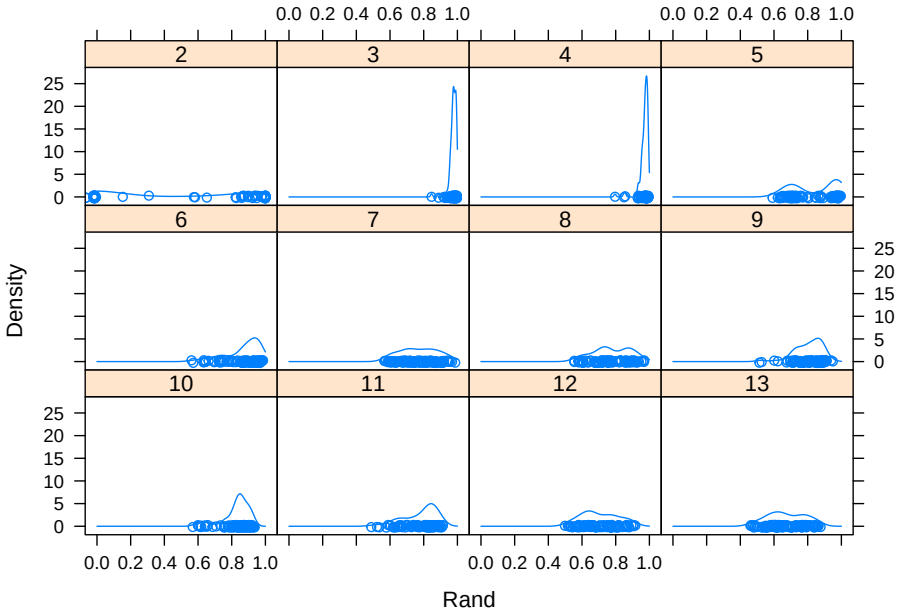

Figure 3: Kernel densities of the adjusted Rand indices

negatively biased in this setup as it is possible that the unrated movies are better suggestions than the rated movie. We expect that this bias becomes larger for the Dislike ranking function. In other words, it is possible that Dislike RF recognizes that several unrated movies are good suggestions as well.

4.3 Support Vector Machines

As discussed previously, we train our SVM algorithms on all path types and a subset of the total training set. We start by presenting the results of tuning the hyperparameters. For classification we used the accuracy and for regression we used MSE. The cross-validation results are tabulated in Table 3. For both the classification and the regression algorithms we obtain the best results when using $\gamma = 2^{-8}$ and $C = 2^6$.[5]

Accuracy	C				
γ	2^{-2}	2^0	2^2	2^4	2^6
2^{-16}	50.34	58.50	69.95	74.91	77.06
2^{-14}	58.51	69.90	74.91	77.08	77.96
2^{-12}	69.94	74.92	77.11	78.10	78.41
2^{-10}	74.98	77.28	78.19	78.67	79.02
2^{-8}	11.50	78.43	78.88	79.41	79.48

MSE	C				
γ	2^{-2}	2^0	2^2	2^4	2^6
2^{-16}	.335	.259	.220	.213	.213
2^{-14}	.259	.220	.212	.209	.198
2^{-12}	.218	.209	.197	.176	.169
2^{-10}	.196	.176	.169	.166	.161
2^{-8}	.169	.163	.156	.153	.151

Table 3: Accuracy and MSE results for 5-fold cross-validation

Therefore, we use the hyperparameters $\gamma = 2^{-8}$ and $C = 2^6$ when training the final SVM and SVR model. The results of the ranking functions based on classification and regression are reported in Table 2. Both SVM ranking functions have similar results. However, the ranking function based on the decision values of classification performs slightly better than the regression based method, specially for a small N. When comparing SVM with the RF ranking functions, we conclude that the classification SVM is comparable with the Aero RF. For lower N the Recall is slightly higher for the Aero RF and for larger N the Recall is slightly higher for the classification SVM. However, both SVM ranking functions get outperformed by the Complete RF ranking functions. The Complete ranking

[5]The training of the SVM models is done with the `R` package `e1071`, i.c.w. `parallel` for the cross-validation.

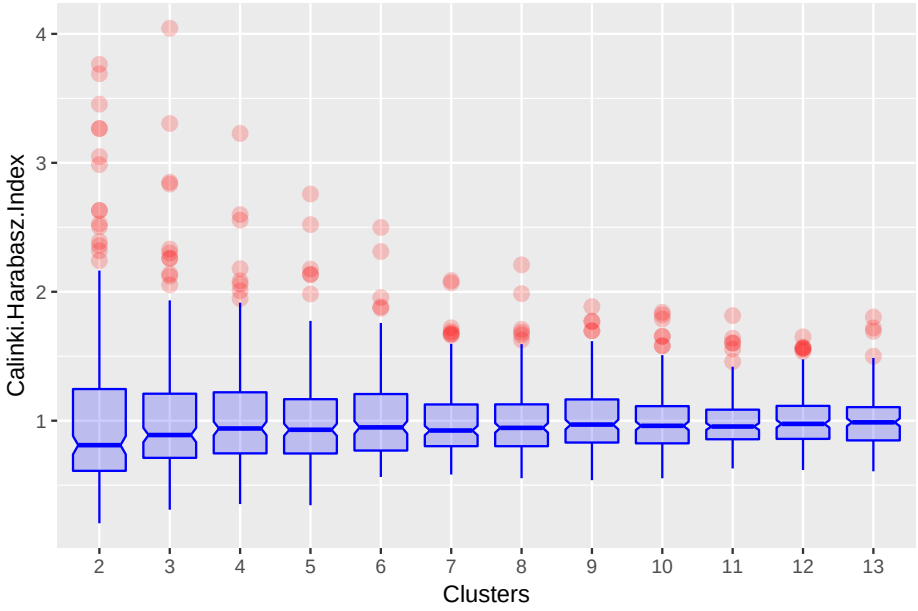

Figure 4: Boxplots of the Calinski-Harabasz indices

function is created training a RF for each individual, while there is one ranking function trained for SVM and SVR trained using only a subset of all available training data.

5 Conclusion and Discussion

Our research question is whether we can improve recommendations by including unsupervised learning. The relatively high variable importance of the Group type paths in the Random Forest ranking functions and the higher performance of the models where the Group type path is included show that the information from unsupervised learning improves the recommendation algorithms. However, we show that this method does not incorporate new information. Furthermore, it might be possible to incorporate information from several variables into the Group variable. In our application we see this happening to Genre. A variable which we expect to have high explanatory power, but it's information might be captured by the Group variable. For future research it is interesting to check if this form of unsupervised learning can be used variable reduction technique.

The goal of this report is to suggest the most relevant movies to users. The Recall@N results show that we can improve Collaborative Filtering recommendations with additional path types, i.e. information about the users and about the movies. In agreement with Wever and Frasincar (2017) we find that information about the movie adds to the performance of the recommendations. However, it is interesting to note that the variable importance of the path types related to the information about the movie is smaller than the variable importance of the user related information. Again, some information about the movie might be capture by the Group variable, the performance of the recommendations increase when we include the information and we did not extend our information about the movies with Linked Open Data.

Although including explicit information about dislike movies, by distinguishing them from unrated movies might increase performance. We cannot conclude this using this setup. However, we can see that there are several unrated movies the Dislike algorithm deems interesting. This shows the bias in Recall@N when only unrated movies are compared with a liked movie. Using OPR and Recall@N might thus not be the best performance measure for testing recommendations. It might be interesting to include other criteria, e.g. one that takes disliked movies into account. As suggesting these items might be harmful for the customer experience.

When using SVM for creating ranking functions we see a slightly better performance for the classification methodology, which uses the decision values for the rankings. Specially if N is small. However, SVM is still outperformed by the Complete RF ranking function. The Complete ranking

function is created training a RF for each individual, while there is one ranking function trained for SVM and SVR trained using only a subset of all available training data. For future research it might be interesting to train the SVM ranking functions on the set which distinguishes disliked from unrated movies. As well as the application of SVM Rank, a kernel based ranking algorithm. Since our data consisted of (-1,) 0 and 1 we used Classification, and Regression is used because its approach is similar to the regression trees used in the RFs.

Finally, it is interesting to note that the grid-search found optimal values on the edge of the parameter range. We would advise to extend the value range in the direction of the optimal values for future work. Furthermore, it would also be interesting to use other performance measures for the cross-validation. The final model is assessed on its ability to retrieve a single relevant item, while the accuracy and MSE are used for the cross-validation. The accuracy and MSE look at the specific values not the respective ranking. The same reasoning could be applied to the use of MSE in the calculation of variable importance in the RF. This might explain why the Aero model is outperformed by the Complete model, when measuring the performance of the top-N recommendations with the OPR methodology. Another explanation might be the limited number of path types considered. We would advise future work to consider a larger number of path types.

6 Appendix

The results of the final clustering are interpreted based on the means per cluster. We can compare these values with the overall average. These descriptive statistics are plotted in Table 4. We can see that more male users than female users in the whole data set. Clusters 8 and 9 contain mostly, if not only female users. While clusters 6 and 10 contain only Male users, 7 contains mostly male users. Cluster 5 comes the closest to represent the ratio of the whole population, meaning that for cluster 5 gender is not important.

When looking at the age categories we can see that category 5 consist of only users who are younger than 18 years. The distribution of age categories of the other clusters follow the full sample, cluster number 1 contains slightly more mature users, 2 is relatively peaked in the middle and 4 contains younger adults.

While occupations are more spread out across the clusters we can see that cluster 1 contains relatively more creative occupations. Since most K-12 students are under 18, almost all of them are in cluster 5.[6]

The average like gives an indication what type of ratings the users gave. The average of 0.25 indicates that the average user rates more movies with 4 or 5 stars than with 1, 2, or 3. This can be a result of some users only rating movies they liked or because they like more movies in general. We can see that clusters 1, 3, 7, 8 and 10 have given more positive ratings, while users in clusters 2 and 4 have a more negative rating.

This also reflects back in the average net scores per genre. Cluster number 1 seems to like more comedy, romance, drama and a bit more mystery, crime and war. The users in clusters 2 and 3 seem to be opposites, as the magnitudes are roughly equal, but the net scores have opposite signs. Cluster 4 seems to dislike mostly adventure, comedy, action, and sci-fi. Although cluster 5 contains only under 18 year olds, they seem to dislike children movies. With regards to the genres, cluster 6 behaves like the overall average. Cluster 7 behaves much like cluster 1, rating movies even higher and adding the genres adventure and sci-fi. Given the high average rating of group 8 and 9 their preferences towards genres are relatively average. Although users in cluster 10 give on average a higher rating, it does not show in their net genre scores.

[6]Note that there are no farmers in our sample, this leaves us with 19 occupations

Cluster	1	2	3	4	5	6	7	8	9	10	Average
Female	0.13	0.11	0.16	0.18	0.35	0.00	0.04	0.98	1.00	0.00	0.28
Male	0.87	0.89	0.84	0.82	0.65	1.00	0.96	0.02	0.00	1.00	0.72
Age under 18	0.01	0.00	0.01	0.01	1.00	0.00	0.00	0.00	0.00	0.00	0.04
18-24	0.07	0.16	0.15	0.29	0.00	0.20	0.18	0.15	0.19	0.19	0.18
25-34	0.37	0.45	0.41	0.43	0.00	0.36	0.38	0.38	0.33	0.36	0.35
35-44	0.22	0.24	0.19	0.15	0.00	0.20	0.22	0.25	0.20	0.22	0.20
45-49	0.13	0.08	0.11	0.06	0.00	0.09	0.06	0.10	0.12	0.09	0.09
50-55	0.12	0.05	0.08	0.05	0.00	0.08	0.10	0.09	0.09	0.08	0.08
56+	0.09	0.02	0.05	0.02	0.00	0.08	0.04	0.03	0.07	0.07	0.06
Occupation not specified	0.14	0.19	0.16	0.16	0.12	0.11	0.09	0.16	0.13	0.09	0.12
educator	0.12	0.04	0.07	0.10	0.02	0.08	0.04	0.14	0.12	0.08	0.09
artist	0.08	0.05	0.05	0.05	0.01	0.04	0.03	0.05	0.05	0.04	0.04
admin	0.04	0.02	0.05	0.01	0.00	0.02	0.03	0.07	0.06	0.01	0.03
student	0.05	0.17	0.16	0.17	0.01	0.12	0.14	0.12	0.15	0.14	0.13
customer service	0.02	0.02	0.02	0.03	0.00	0.02	0.03	0.01	0.02	0.02	0.02
health care	0.03	0.02	0.07	0.02	0.00	0.03	0.04	0.04	0.07	0.03	0.04
managerial	0.09	0.14	0.10	0.09	0.01	0.14	0.13	0.08	0.09	0.12	0.11
homemaker	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.06	0.00	0.02
K-12 student	0.01	0.00	0.00	0.02	0.73	0.01	0.00	0.01	0.00	0.01	0.03
lawyer	0.04	0.02	0.01	0.01	0.00	0.02	0.02	0.01	0.02	0.04	0.02
programmer	0.07	0.01	0.02	0.06	0.00	0.08	0.08	0.02	0.03	0.10	0.06
retired	0.04	0.00	0.02	0.01	0.00	0.03	0.00	0.01	0.02	0.02	0.02
sales / marketing	0.03	0.03	0.04	0.06	0.01	0.05	0.07	0.05	0.05	0.05	0.05
scientist	0.01	0.01	0.05	0.03	0.00	0.03	0.03	0.03	0.01	0.03	0.02
self-employed	0.04	0.08	0.08	0.04	0.00	0.04	0.07	0.04	0.03	0.04	0.04
engineer	0.04	0.08	0.01	0.06	0.01	0.11	0.12	0.03	0.03	0.12	0.08
tradesman	0.01	0.00	0.00	0.02	0.00	0.02	0.01	0.00	0.00	0.02	0.01
unemployed	0.00	0.04	0.02	0.02	0.06	0.01	0.01	0.00	0.01	0.01	0.01
writer	0.12	0.07	0.04	0.05	0.00	0.04	0.05	0.07	0.04	0.04	0.05
like	0.40	-0.50	0.52	-0.30	0.20	0.18	0.52	0.45	0.25	0.45	0.25
Adventure	6.74	-65.77	53.31	-21.59	3.57	1.66	36.09	16.25	1.87	12.06	4.47
Animation	3.32	-10.83	19.24	-2.77	1.00	0.38	8.23	8.78	0.75	2.78	1.69
Children	2.00	-34.44	24.48	-13.34	-0.60	-0.27	10.37	12.52	0.72	2.93	0.79
Comedy	27.55	-145.34	113.92	-48.64	5.92	1.67	50.73	38.94	4.53	16.83	6.79
Fantasy	4.17	-34.04	28.46	-11.85	0.74	0.15	13.08	9.98	0.92	3.91	1.44
Romance	19.41	-60.20	66.19	-17.27	3.22	1.18	27.93	29.25	4.53	9.02	5.61
Drama	72.92	-113.81	189.93	-18.20	9.47	6.20	78.54	54.04	10.88	29.89	19.22
Action	1.90	-104.46	53.33	-35.46	3.69	1.33	45.21	12.01	1.34	14.50	3.33
Crime	23.13	-37.26	53.34	-7.34	3.21	2.10	27.65	12.11	2.76	10.80	5.92
Thriller	15.06	-102.49	69.88	-28.93	2.67	0.66	41.34	13.12	1.68	12.85	3.90
Horror	-0.01	-53.63	15.60	-18.37	-0.08	-0.81	8.65	1.04	-0.34	2.05	-1.15
Mystery	11.21	-22.14	28.04	-4.13	1.57	0.80	12.69	7.29	1.38	5.12	2.75
SciFi	1.45	-62.22	32.46	-23.63	1.69	0.30	23.98	7.65	0.44	8.12	1.41
IMAX	0.21	-0.19	1.69	-0.04	0.30	0.05	1.24	1.31	0.13	0.56	0.29
Documentary	4.57	-0.70	6.08	0.39	0.13	0.17	0.67	0.94	0.23	0.50	0.57
War	11.50	-7.52	24.07	1.40	2.06	1.64	13.43	6.61	1.67	5.89	3.74
Musical	4.93	-14.09	22.78	-4.29	1.00	0.06	8.19	11.66	0.99	2.44	1.69
Western	2.99	-7.03	10.23	-1.61	0.21	0.17	6.55	1.43	0.17	1.77	0.85
FilmNoir	8.45	0.32	9.63	1.05	0.51	0.48	2.51	1.82	0.54	1.63	1.35

Table 4: Mean of different variables per cluster and average over all users

References

Bellogin, A., Castells, P., and Cantador, I. (2011). Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 333–336. ACM.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Coussement, K. and Van den Poel, D. (2008). Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert systems with applications*, 34(1):313–327.

Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM.

Dolnicar, S. and Leisch, F. (2010). Evaluation of structure and reproducibility of cluster solutions using the bootstrap. *Marketing Letters*, 21(1):83–101.

Keerthi, S. S. and Lin, C.-J. (2003). Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Lin, H.-T. and Lin, C.-J. (2003). A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. *submitted to Neural Computation*, pages 1–32.

Ostuni, V. C., Di Noia, T., Di Sciascio, E., and Mirizzi, R. (2013). Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 85–92. ACM.

Strobl, C. and Zeileis, A. (2008). Danger: High power!–exploring the statistical properties of a test for random forest variable importance.

Sun, Y., Han, J., Yan, X., Yu, P. S., and Wu, T. (2011). Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003.

Wever, T. and Frasincar, F. (2017). A linked open data schema-driven approach for top-n recommendations. In *Proceedings of the Symposium on Applied Computing*, pages 656–663. ACM.