kNN Versus SVM in the Collaborative Filtering Framework

Miha Grčar, Blaž Fortuna, Dunja Mladenič, and Marko Grobelnik

Jožef Stefan Institute,
Jamova 39, SI-1000 Ljubljana, Slovenia

Abstract. We present experimental results of confronting the k-Nearest Neighbor (kNN) algorithm with Support Vector Machine (SVM) in the collaborative filtering framework using datasets with different properties. While k-Nearest Neighbor is usually used for the collaborative filtering tasks, Support Vector Machine is considered a state-of-the-art classification algorithm. Since collaborative filtering can also be interpreted as a classification/regression task, virtually any supervised learning algorithm (such as SVM) can also be applied. Experiments were performed on two standard, publicly available datasets and, on the other hand, on a real-life corporate dataset that does not fit the profile of ideal data for collaborative filtering. We conclude that the quality of collaborative filtering recommendations is highly dependent on the quality of the data. Furthermore, we can see that kNN is dominant over SVM on the two standard datasets. On the real-life corporate dataset with high level of sparsity, kNN fails as it is unable to form reliable neighborhoods. In this case SVM outperforms kNN.

1 Introduction and motivation

The goal of collaborative filtering is to explore a vast collection of items in order to detect those which might be of interest to the active user. In contrast to content-based recommender systems which focus on finding contents that best match the user's query, collaborative filtering is based on the assumption that similar users have similar preferences. It explores the database of users' preferences and searches for users that are similar to the active user. The active user's preferences are then inferred from preferences of the similar users. The content of items is usually ignored.

In this paper we explore how two different approaches to collaborative filtering – the memory-based k-Nearest Neighbor approach (kNN) and the model-based Support Vector Machine (SVM) approach – handle data with different properties. We used two publicly available datasets that are commonly used in collaborative filtering evaluation and, on the other hand, a dataset derived from real-life corporate Web logs. The latter does not fit the profile of ideal data for collaborative filtering. Namely, collaborative filtering is usually applied to research datasets with relatively low sparsity. Here we have included a real-life dataset of a company in need of providing collaborative filtering recommendations. It turned out that this dataset has much higher sparsity than usually handled in the collaborative filtering scenario.

The rest of this paper is arranged as follows. In Sections 2 and 3 we discuss collaborative filtering algorithms and data quality for collaborative filtering. The three datasets used in our experiments are described in Section 4. In Sections 5 and 6 the experimental setting and the evaluation results are presented. The paper concludes with the discussion and some ideas for future work (Section 7).

2 Collaborative filtering in general

There are basically two approaches to the implementation of a collaborative filtering algorithm. The first one is the so called "lazy learning" approach (also known as the memory-based approach) which skips the learning phase. Each time it is about to make a recommendation, it simply explores the database of user-item interactions. The model-based approach, on the other hand, first builds a model out of the user-item interaction database and then uses this model to make recommendations. "Making recommendations" is equivalent to predicting the user's preferences for unobserved items.

The data in the user-item interaction database can be collected either explicitly (explicit ratings) or implicitly (implicit preferences). In the first case the user's participation is required. The user is asked to explicitly submit his/her rating for the given item. In contrast to this, implicit preferences are inferred from the user's actions in the context of an item (that is why the term "user-item interaction" is used instead of the word "rating" when referring to users' preferences in this paper). Data can be collected implicitly either on the client side or on the server side. In the first case the user is bound to use modified client-side software that logs his/her actions. Since we do not want to enforce modified client-side software, this possibility is usually omitted. In the second case the logging is done by a server. In the context of the Web, implicit preferences can be determined from access logs that are automatically maintained by Web servers.

Collected data is first preprocessed and arranged into a user-item matrix. Rows represent users and columns represent items. Each matrix element is in general a set of actions that a specific user took in the context of a specific item. In most cases a matrix element is a single number representing either an explicit rating or a rating that was inferred from the user's actions.

Since a user usually does not access every item in the repository, the vector (i.e. the matrix row), representing the user, is missing some/many values. To emphasize this, we use the terms "sparse vector" and "sparse matrix".

The most intuitive and widely used algorithm for collaborative filtering is the so called k-Nearest Neighbor algorithm which is a memory-based approach. Technical details can be found, for example, in Grcar (2004). The algorithm is as follows:

1. Represent each user by a sparse vector of his/her ratings.

2. Define the similarity measure between two sparse vectors. In this paper, we consider two widely used measures: (i) the Pearson correlation coefficient which is used in statistics to measure the degree of correlation between two variables (Resnick et al. (1994)), and (ii) the Cosine similarity measure which is originally used in information retrieval to compare between two documents (introduced by Salton and McGill (1983)).
3. Find k users that have rated the item in question and are most similar to the active user (i.e. the user's neighborhood).
4. Predict the active user's rating for the item in question by calculating the weighted average of the ratings given to that item by other users from the neighborhood.

The collaborative filtering task can also be interpreted as a classification task, classes being different rating values (Billsus and Pazzani (1998)). Virtually any supervised learning algorithm can be applied to perform classification (i.e. prediction). For each user a separate classifier is trained (i.e. a model is built – hence now we are talking about a model-based approach). The training set consists of feature vectors representing items the user already rated, class labels being ratings from the user. Clearly the problem occurs if our training algorithm cannot handle missing values in the sparse feature vectors. It is suggested by Billsus and Pazzani (1998) to represent each user by several instances (optimally, one instance for each possible rating value). On a 1–5 rating scale, user A would be represented with 5 instances, namely {A, 1}, {A, 2}, ..., {A, 5}. The instance {A, 3}, for example, would hold ones ("1") for each item that user A rated 3 and zeros ("0") for all other items. This way, we fill in the missing values. We can now use such binary feature vectors for training. To predict a rating, we need to classify the item into one of the classes representing rating values. If we wanted to predict ratings on a continuous scale, we would have to use a regression approach instead of classification.

In our experiments we confronted the standard kNN algorithm (using Pearson and Cosine as the similarity measures) with SVM classifier and SVM regression (Vapnik (1998)). In the case of SVM we did not convert the user-item matrix into the dense binary representation, as SVM can handle sparse data directly.

3 Sparsity problem and data quality for collaborative filtering

The fact that we are dealing with a sparse matrix can result in the most concerning problem of collaborative filtering – the so called sparsity problem. In order to be able to compare two sparse vectors, similarity measures require some values to overlap. Furthermore, the lower the amount of overlapping values, the lower the reliability of these measures. If we are dealing with high level of sparsity, we are unable to form reliable neighborhoods.

Sparsity is not the only reason for the inaccuracy of recommendations provided by collaborative filtering. If we are dealing with implicit preferences, the ratings are usually inferred from the user-item interactions, as already mentioned earlier in the text (Section 2). Mapping implicit preferences into explicit ratings is a non-trivial task and can result in false mappings. The latter is even more true for server-side collected data in the context of the Web since Web logs contain very limited information. To determine how much time a user was reading a document, we need to compute the difference in time-stamps of two consecutive requests from that user. This, however, does not tell us wether the user was actually reading the document or he/she, for example, went out to lunch, leaving the browser opened. There are also other issues with monitoring the activities of Web users, which can be found in Rosenstein (2000).

From this brief description of data problems we can conclude that for applying collaborative filtering, explicitly given data with low sparsity are preferred to implicitly collected data with high sparsity. The worst case scenario is having highly sparse data derived from Web logs. However, collecting data in such a manner requires no effort from the users and also, the users are not obliged to use any kind of specialized Web browsing software. This "conflict of interests" is illustrated in Figure 1.

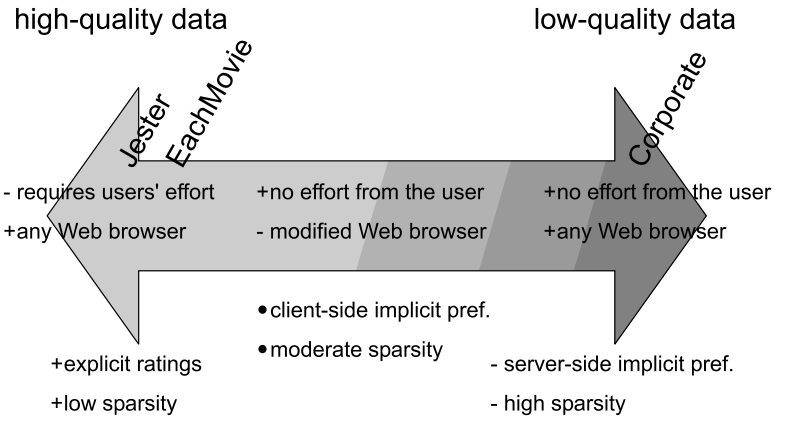


Fig. 1. Data characteristics that influence the data quality, and the positioning of the three datasets used in our experiments, according to their properties.

4 Data description

For our experiments we used three distinct datasets. The first dataset was EachMovie (provided by Digital Equipment Corporation) which contains ex-

plicit ratings for movies. The service was available for 18 months. The second dataset with explicit ratings was Jester (provided by Goldberg et al. (2001)) which contains ratings for jokes, collected over a 4-year period. The third dataset was derived from real-life corporate Web logs. The logs contain accesses to an internal digital library of a fairly large company. The time-span of acquired Web logs is 920 days. In this third case the user's preferences are implicit and collected on the server side, which implies the worst data quality for collaborative filtering.

In contrast to EachMovie and Jester, Web logs first needed to be extensively preprocessed. Raw logs contained over 9.3 million requests. After all the irrelevant requests (i.e. failed requests, non-GET requests, requests by anonymous users, requests for images, index pages, and other irrelevant pages) were removed we were left with only slightly over 20,500 useful requests, which is 0.22% of the initial database size. For detailed description of the preprocessing that was applied see Grčar et al. (2005). Note that only the dataset termed "Corporate 1/2/3/2" in Grčar et al. (2005) was considered in this paper.

Table 1 shows the comparison between the three datasets. It is evident that a low number of requests and somewhat ad-hoc mapping onto a discrete scale are not the biggest issues with our corporate dataset. The concerning fact is that the average number of ratings per item is only 1.22, which indicates extremely poor overlappingness. Sparsity is consequently very high, 99.93%. The other two datasets are much more promising. The most appropriate for collaborative filtering is the Jester dataset with very low sparsity, followed by EachMovie with higher sparsity but still relatively high average number of ratings per item. Also, the latter two contain explicit ratings, which means that they are more reliable than the corporate dataset (see also Figure 1).

Table 1. The data characteristics of the three datasets, showing the explicitness of ratings (explicit, implicit), size of the dataset and the level of sparsity.

	Ratings		Size			Sparsity		
	Explicit/ implicit	Scale	Num of users	Num of items	Num of ratings	%**	Avg # of r'tings/usr	Avg # of ratings/item
EachMovie	Explicit	Discrete 0–5	61,131	1,622	2,558,871	97.42	41.86	1,577.60
Jester	Explicit	Continuous −10 – +10	73,421	100	4,136,360	43.66	56.34	41,363.60
Corporate	Implicit	Discrete 1–3*	1,850	16,941	20,669	99.93	11.17	1.22

*after preprocessing
**computed as the number of missing values divided by the user-item matrix size (i.e. the number of rows times the number of columns)

5 Experimental setting

To be able to perform evaluation we built an evaluation platform (Grčar et al. (2005)). We ran a series of experiments to see how the accuracy of collaborative filtering recommendations differs between the two different approaches and the three different datasets (from EachMovie and Jester we considered only 10,000 randomly selected users to speed up the evaluation process). Ratings from each user were partitioned into "given" and "hidden" according to the "all-but-30%" evaluation protocol. The name of the protocol implies that 30% of all the ratings were hidden and the remaining 70% were used to form neighborhoods.

We applied three variants of memory-based collaborative filtering algorithms: (i) k-Nearest Neighbor using the Pearson correlation (kNN Pearson), (ii) k-Nearest Neighbor using the Cosine similarity measure (kNN Cosine), and (iii) the popularity predictor (Popularity). The latter predicts the user's ratings by simply averaging all the available ratings for the given item. It does not form neighborhoods or build models and it provides each user with the same recommendations. It serves merely as a baseline when evaluating collaborative filtering algorithms (termed "POP" in Breese et al. (1998)). For kNN variants we used a neighborhood of 120 users (i.e. k=120), as suggested in Goldberg et al. (2001).

In addition to the variants of the memory-based approach we also applied two variants of the model-based approach: SVM classifier, and SVM regression. In general, SVM classifier can classify a new example into one of the two classes: positive or negative. If we want to predict ratings, we need a multi-class variant of SVM classifier, classes being different rating values. The problem also occurs when dealing with continuous rating scales such as Jester's. To avoid this, we simply sampled the scale interval and thus transformed the continuous scale into a discrete one (in our setting we used $0.\overline{3}$ precision to sample the Jester's rating scale).

Although the work of Billsus and Pazzani (1998) suggests using items as examples, the task of collaborative filtering can equivalently be redefined to view users as examples. Our preliminary results showed that it is best to choose between these two representations with respect to the dataset properties. If the dataset is more sparse "horizontally" (i.e. the average number of ratings per user is lower than the average number of ratings per item), it is best to take users as examples. Otherwise it is best to take items as examples. Intuitively, this gives more training examples to build models which are consequently more reliable. With respect to the latter, we used users as examples when dealing with EachMovie (having on average 41.86 ratings per user vs. 1,577.60 ratings per item) and Jester datasets (having on average 56.34 ratings per user vs. 41,363.60 ratings per item) and items as examples when dealing with the corporate dataset (having on average 11.17 ratings per user vs. 1.22 ratings per item).

We combined several binary SVM classifiers in order to perform multi-class classification. Let us explain the method that was used on an example. We first transform the problem into a typical machine learning scenario with ordered class values as explained earlier in the previous paragraph. Now, let us consider a discrete rating scale from 1 to 5. We need to train 4 SVMs to be able to classify examples into 5 different classes (one SVM can only decide between positive and negative examples). We train the first SVM to be able to decide weather an example belongs to class 1 (positive) or to any of the classes 2–5 (negative). The second SVM is trained to distinguish between classes 1–2 (positive) and classes 3–5 (negative). The third SVM distinguishes between classes 1–3 (positive) and classes 4–5 (negative), and the last SVM distinguishes between classes 1–4 (positive) and class 5 (negative). In order to classify an example into one of the 5 classes, we query these SVMs in the given order. If the first one proves positive, we classify the example into class 1, if the second one proves positive, we classify the example into class 2, and so on in that same manner. If all of the queries prove negative, we classify the example into class 5. We used SVM classifier as implemented in Text-Garden (`http://www.textmining.net`). We built a model only if there were at least 7 positive and at least 7 negative examples available (because our preliminary experiments showed that this is a reasonable value to avoid building unreliable models).

SVM regression is much more suitable for our task than SVM classifier. It can directly handle continuous and thus also ordered discrete class values. This means we only need to train one model as opposed to SVM classifier where several models need to be trained. We used SVM regression as implemented in Text-Garden. As in the case of SVM classifier, we built a model only if there were at least 15 examples available.

Altogether we ran 5 experiments for each dataset-algorithm pair, each time with a different random seed (we also selected a different set of 10,000 users from EachMovie and Jester each time). When applying collaborative filtering to the corporate dataset, we made 10 repetitions (instead of 5) since this dataset is smaller and highly sparse, which resulted in less reliable evaluation results. Thus, we ran 100 experiments altogether.

We decided to use normalized mean absolute error (NMAE) as the accuracy evaluation metric. We first computed NMAE for each user and then we averaged it over all the users (termed "per-user NMAE") (Herlocker et al. (2004)). MAE is extensively used for evaluating collaborative filtering accuracy and was normalized in our experiments to enable us to compare evaluation results from different datasets.

6 Evaluation results

We present the results of experiments performed on the three datasets using the described experimental setting (see Section 5). We used two-tailed paired

Student's t-Test with significance 0.05 to determine if the differences in results are statistically significant.

We need to point out that in some cases the algorithms are unable to predict the ratings, as the given ratings do not provide enough information for the prediction. For instance, Popularity is not able to predict the rating if there are no ratings in the given data for a particular item. When calculating the overall performance we exclude such ratings from the evaluation, as we are mainly interested in the quality of prediction when available, even if the percentage of available predictions is low. We prefer the system to provide no recommendation if there is not enough data for a reasonably reliable prediction.

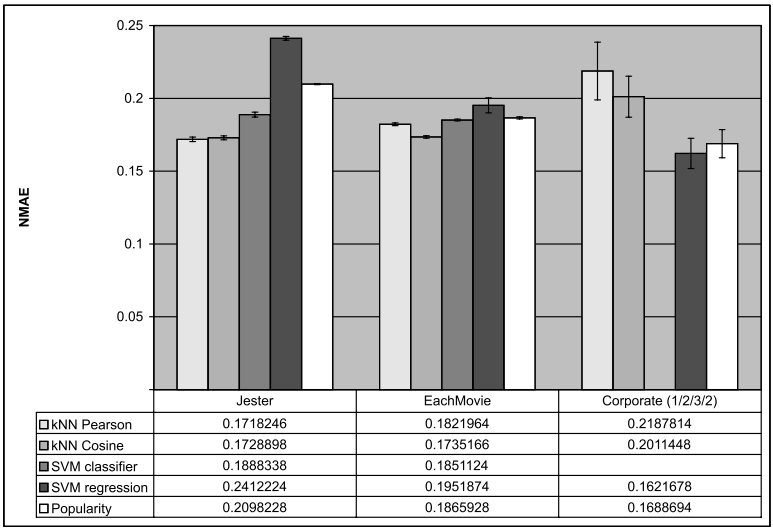
As mentioned earlier in Section 4, the three datasets have different characteristics that influence the accuracy of predictions. Jester is the dataset with the lowest sparsity and thus the most suitable for the application of collaborative filtering of the three tested datasets. We see that the kNN methods significantly outperform the other three methods. kNN Pearson slightly yet significantly outperforms kNN Cosine. SVM classifier also performs well, significantly outperforming SVM regression and Popularity. Interestingly, SVM regression performs significantly worse than Popularity.

EachMovie is sparser than Jester yet much more suitable for the application of collaborative filtering than the corporate dataset. Here kNN Cosine performs significantly better than kNN Pearson, followed by SVM classifier and Popularity. kNN Pearson slightly yet significantly outperforms Popularity. Again, SVM regression performs significantly worse than Popularity.

The corporate dataset is the worst of the three – it is extremely sparse and collected implicitly on the server side. It reveals the weakness of the kNN approach – lack of overlapping values results in unreliable neighborhoods. Notice that we do not provide the results of applying SVM classifier on this dataset, as the quality of the corporate dataset is too low for the classification setting. We see that SVM regression and Popularity perform best in this domain. The difference between them is not significant but they both significantly outperform the kNN approach. In this paper we are not concerned with the inability to predict but it is still worth mentioning that SVM regression can predict 72% of the hidden ratings, Popularity 23.7%, and the kNN approach only around 8% of the hidden ratings.

7 Discussion and future work

In our experimental setting we confronted the k-Nearest Neighbor algorithm with Support Vector Machine in the collaborative filtering framework. We can see that on our datasets, kNN is dominant on datasets with relatively low sparsity (Jester). On the two datasets with high to extremely high level of sparsity (EachMovie, the corporate dataset), kNN starts failing as it is unable to form reliable neighborhoods. In such case it is best to use a model-



	Jester	EachMovie	Corporate (1/2/3/2)
kNN Pearson	0.1718246	0.1821964	0.2187814
kNN Cosine	0.1728898	0.1735166	0.2011448
SVM classifier	0.1888338	0.1851124	
SVM regression	0.2412224	0.1951874	0.1621678
Popularity	0.2098228	0.1865928	0.1688694

Fig. 2. The results of the experiments.

based approach, such as SVM classifier or SVM regression. Another strong argument for using the SVM approaches on highly sparse data is the ability to predict more ratings than with the variants of the memory-based approach.

Interestingly, Popularity performs extremely well on all domains. It fails, however, when recommending items to eccentrics. We noticed that the true value of collaborative filtering (in general) is shown yet when computing NMAE over some top percentage of eccentric users. We defined eccentricity intuitively as MAE (mean absolute error) over the overlapping ratings between "the average user" and the user in question (greater MAE yields greater eccentricity). The average user was defined by averaging ratings for each particular item. This is based on the intuition that the ideal average user would rate every item with the item's average rating. Our preliminary results show that the incorporation of the notion of eccentricity can give the more sophisticated algorithms a fairer trial. We computed average per-user NMAE only over the top 5% of eccentric users. The power of the kNN algorithms over Popularity became even more evident. In the near future we will define an accuracy measure that will weight per-user NMAE according to the user's eccentricity, and include it into our evaluation platform.

In future work we also plan to investigate if the observed behaviour – that SVM regression outperforms the kNN approaches at a certain level of sparsity – holds in general or only for the three datasets used in our evaluation.

Also interesting, the Cosine similarity works just as well as Pearson on EachMovie and Jester. Early researches show much poorer performance of the Cosine similarity measure (Breese et al. (1998)).

Acknowledgements

This work was supported by the Slovenian Research Agency and the IST Programme of the European Community under SEKT Semantically Enabled Knowledge Technologies (IST-1-506826-IP) and PASCAL Network of Excellence (IST-2002-506778). The EachMovie dataset was provided by Digital Equipment Corporation. The Jester dataset is courtesy of Ken Goldberg et al. The authors would also like to thank Tanja Brajnik for her help.

References

BILLSUS, D., and PAZZANI, M. J. (1998): Learning Collaborative Information Filers. In: *Proceedings of the Fifteenth International Conference on Machine Learning.*

BREESE, J.S., HECKERMAN, D., and KADIE, C. (1998): Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence.*

CLAYPOOL, M., LE, P., WASEDA, M., and BROWN, D. (2001): Implicit Interest Indicators. In: *Proceedings of IUI'01.*

DEERWESTER, S., DUMAIS, S.T., and HARSHMAN, R. (1990): Indexing by Latent Semantic Analysis. In: *Journal of the Society for Information Science, Vol. 41, No. 6, 391–407.*

GOLDBERG, K., ROEDER, T., GUPTA, D., and PERKINS, C. (2001): Eigentaste: A Constant Time Collaborative Filtering Algorithm. In: *Information Retrieval, No. 4, 133–151.*

GRCAR, M. (2004): User Profiling: Collaborative Filtering. In: *Proceedings of SIKDD 2004 at Multiconference IS 2004, 75–78.*

GRCAR, M., MLADENIC D., GROBELNIK, M. (2005): Applying Collaborative Filtering to Real-life Corporate Data. In: *Proceedings of the 29th Annual Conference of the German Classification Society (GfKl 2005)*, Springer, 2005.

HERLOCKER, J.L., KONSTAN, J.A., TERVEEN, L.G., and RIEDL, J.T. (2004): Evaluating Collaborative Filtering Recommender Systems. In: *ACM Transactions on Information Systems, Vol. 22, No. 1, 5–53.*

HOFMANN, T. (1999): Probabilistic Latent Semantic Analysis. In: *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence.*

MELVILLE, P., MOONEY, R.J., and NAGARAJAN, R. (2002): Content-boosted Collaborative Filtering for Improved Recommendations. In: *Proceedings of the 18th National Conference on Artificial Intelligence, 187–192.*

RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., and RIEDL, J. (1994): GroupLens: An Open Architecture for Collaborative Filtering for Netnews. In: *Proceedings of CSCW'94, 175–186.*

ROSENSTEIN, M. (2000): What is Actually Taking Place on Web Sites: E-Commerce Lessions from Web Server Logs. In: *Proceedings of EC'00.*

SALTON, G., McGILL, M.J. (1983): Introduction to Modern Information Retrieval. McGraw-Hill, New York.

VAPNIK, V. (1998): Statistical Learning Theory. Wiley, New York.