A Recommender System Based on Omni-Channel Customer Data

Matthias Carnein, Leschek Homann, Heike Trautmann and Gottfried Vossen
Department of Information Systems, University of Münster
Münster, Germany
Email: {Matthias.Carnein, Leschek.Homann, Heike.Trautmann, Gottfried.Vossen}@wi.uni-muenster.de

***Abstract*—Recommender systems aim to provide personalized suggestions to customers which products to buy or services to consume. They can help to increase sales by helping customers discover new and relevant products. Traditionally, recommender systems use the purchase history of a customer, e.g., the purchased quantity or properties of the items. While this allows to build personalized recommendations, it is a very limited view of the problem. Nowadays, extensive information about customers and their personal preferences is available which goes far beyond their purchase behaviour. For example, customers reveal their preferences in social media, by their browsing habits and online search behaviour or their interest in specific newsletters. In this paper, we investigate how information from different sources and channels can be collected and incorporated into the recommendation process. We demonstrate this, based on a real-life case study of a retailer with several million transactions. We discuss how to employ a recommender system in this scenario, evaluate various recommendation strategies and describe how to incorporate information from different sources and channels, both internal and external. Our results show that the recommendations can be better tailored to the personal preferences of customers.**

***Keywords*-Recommender Systems; Omni-Channel; Machine Learning;**

I. Introduction

In the past, retailers were mostly passive actors which offered numerous products to customers. Based on the increasing catalogue size, growing competition and digitization, a new approach to actively recommend products became more popular. Its goal is to help customers sift through the catalogue of products by recommending items that a customer might like. Nowadays, the success of a retailer is closely related to a well implemented recommender system [1]. Modern recommender systems essentially recommend products that users with similar preference liked.

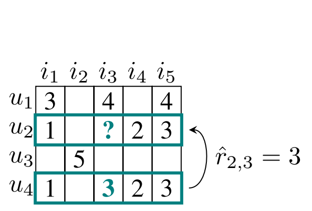
In this paper, we show how to implement a recommender system based on transactional data from a real-life case study of an on- and offline retailer with several million transactions. In particular, we show that there is much more information available beyond the purchase history which can improve the recommendation process. For example, we infer preference of a customer by analysing its search and browsing behaviour on the company's website. The underlying idea is that searching for products and browsing product pages already indicates preference. Similarly, we utilize newsletter subscriptions and click-rates as well as data from social media sources. Specifically, the customer's like, share and post behaviour helps to understand what products he or she might be interested in. Beyond this, we also propose how to improve the timing of recommendations by detecting when users might be interested. We demonstrate that our approach allows to extract better preference indicators and makes accurate recommendations. We term our approach an omni-channel recommender system since it integrates and uses information from all available channels and data sources.

The structure of this paper is as follows: Section II introduces the relevant literature for recommender systems. Then, Section III outlines our case study and shows how information from various sources can be collected and used to build an improved recommender system. In Section IV, we perform extensive experiments and evaluate our approach on a large real-life dataset. Additionally, Section V proposes how to improve the timing of recommendations. Finally, Section VI concludes the paper and gives an outlook on possible improvements.

II. Background on Recommender Systems

Recommender systems are typically based on user feedback where a user expresses to what degree he or she likes a product. A typical example of this are movie ratings where the user gives a rating between one and five stars. Recommender systems then try to estimate the number of stars for unwatched movies and generate a list from most suitable to least suitable recommendations.

Among the most popular approaches for recommender systems are Content-based and Collaborative Filtering (CF) approaches [2], [3]. Content-based approaches recommend items that have similar properties as items that a user previously liked. As an example, for movie recommendations, a content-based recommender system could recommend other movies of the same genre or similar actors. Collaborative Filtering approaches, however, recommend items that users with similar preference liked. Therefore, this uses the collective purchase patterns of users rather than item properties. We focus on this approach, since it has generally


(a) User-based CF

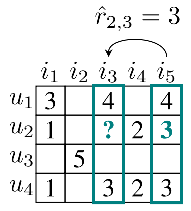

(b) Item-based CF

Figure 1. K-Nearest Neighbour example using $k = 1$.

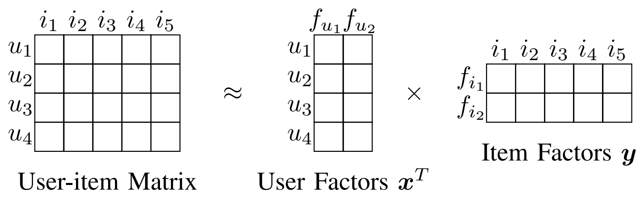


Figure 2. Approximation of the user-item matrix using two user and item factors.

shown to produce more accurate results [4]. Famous retailers and content providers such as Amazon and Netflix have implemented such systems with great success [5], [6]. For more information on recommender systems we refer to [2], [7], [8]. An overview of the evaluation of recommender systems can be found in [9].

In the following, the indices u and v are reserved for users and i and j for items. When a user u rates an item i the rating is denoted as $r_{u,i}$ and all ratings are stored in a large but sparse user-item matrix. A rating prediction is denoted as $\hat{r}$.

A. k-Nearest Neighbour

An example of Collaborative Filtering is the k-Nearest Neighbour (k-NN) approach [2]. The basic idea is to search for "like-minded" users and use their rating of an item for the prediction. This approach is usually referred to as user-based Collaborative Filtering. More specifically, the algorithm computes a similarity score s between users to identify the k users with the most similar rating behaviour. The prediction is then calculated as the weighted average of the rating from these neighbours:

$$\hat{r}_{u,i} = \frac{\sum_{v \in \boldsymbol{V}} s_{u,v} \cdot r_{v,i}}{\sum_{v \in \boldsymbol{V}} s_{u,v}}, \tag{1}$$

where $\boldsymbol{V}$ is the set of k-nearest neighbours of u. Figure 1(a) shows a simple example for $k = 1$. As an extension it is also possible to include biases which factor in the average rating behaviour of u and v:

$$\hat{r}_{u,i} = \overline{r}_u + \frac{\sum_{v \in K} s_{u,v} \cdot (r_{v,i} - \overline{r}_v)}{\sum_{v \in K} s_{u,v}}. \tag{2}$$

A problem with the user-based approach is that it requires a similarity comparison between all pairs of users. This is not scalable to a scenario of reasonable size due to the large number of users and the complexity of computing all pairwise similarities. However, a simpler formulation of the same idea is possible by computing a similarity score between items based on whether they have been rated similarly by users (Figure 1(b)). This is called item-based Collaborative Filtering and since the number of items is usually much smaller than the number of users, it is computationally less demanding [10]. Again, the prediction is computed as the weighted average of the rating from neighbouring items:

$$\hat{r}_{u,i} = \frac{\sum_{j \in J} s_{i,j} \cdot r_{u,j}}{\sum_{j \in J} s_{i,j}}, \tag{3}$$

where J is the set of k-nearest items of i. A popular approach to compute the similarities between items or users is the adjusted cosine similarity [3], [2], e.g., for item-based CF:

$$s_{i,j} = \frac{\sum_{u \in \boldsymbol{U}} (r_{u,i} - \overline{r}_u)(r_{u,j} - \overline{r}_u)}{\sqrt{\sum_{u \in \boldsymbol{U}} (r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{u \in \boldsymbol{U}} (r_{u,j} - \overline{r}_u)^2}}. \tag{4}$$

where $\boldsymbol{U}$ is the set of users that rated both items i and j and $\overline{r}_u$ is the average rating of user u. Possible alternatives are the closely related Pearson's correlation coefficient or Spearman's rank correlation coefficient.

B. Singular Value Decomposition

One of the most popular algorithms for recommender systems is Singular Value Decomposition (SVD). SVD is a matrix factorization approach which can decompose the large matrix of user-item ratings in two much smaller matrices (Figure 2) [11]. Specifically, the idea is that there exist some unknown factors that explain how a user rates an item. These factors could be the colour of a product, genre of a movie or aspects much more abstract and harder to interpret. Subsequently, the user preference captures how much interest a user has in that aspect. The user preference x_u and degree to which the item exhibits this property y_i can then estimate the rating using the inner product:

$$\hat{r}_{u,i} = x_u^T y_i. \tag{5}$$

As a result, the prediction does not require the full user-item matrix but only a user chosen number of latent factors for prediction.

In order to obtain the latent factors, an optimization task is used which minimizes the regularized squared prediction error:

$$\min_{x^*, y^*} \sum_{r_{u,i} \in R} (r_{u,i} - \hat{r}_{u,i})^2 + \lambda(\|x_u\|^2 + \|y_i\|^2), \tag{6}$$

where R is the set of all available ratings and λ a regularization factor. While the first term minimizes the squared prediction error, the second term prevents overfitting.

As an improvement, biases are often introduced which include whether users generally give higher or lower ratings:

$$\hat{r}_{u,i} = \overline{r} + b_u + b_i + x_u^T y_i \tag{7}$$

Here, $\overline{r}$ is the average rating of all users. The user and item biases b_u and b_i capture whether user u or item i generally receive higher or lower ratings than the average user or item. Note that the squared user and item biases also need to be added to the regularization term in Equation 6. In order to obtain the latent factors, an optimization task is used which minimizes the squared prediction error. The optimization problem is often solved by using a stochastic gradient descent optimization [11], [7], [8].

Thus far, all algorithms require that the user gave explicit feedback about its preference in the past (such as movie ratings). Ratings that are not available are assumed to be missing. Recommender systems based on explicit feedback are generally well studied and commonly used in practice. This approach is not directly transferable to transactional data since explicit ratings are often not available. In this case, preference needs to be inferred from the user's actions, e.g., the purchase of a product. In this case, the rating $r_{u,i}$ denotes the number of times a user purchased an item. If no preference was observed, the rating for the item is assumed to be zero. Alternatively, a binary preference value $p_{u,i}$ whether the user purchased an item can be used [12], [4]:

$$p_{u,i} = \begin{cases} 1 & r_{u,i} > 0 \\ 0 & r_{u,i} = 0. \end{cases} \tag{8}$$

A key difference of this is that missing preferences are now assumed to express disfavour. This causes the user-item matrix to grow considerably in size and often contain billions of entries. Traditional algorithms usually do not scale well to such large matrices.

An extension of the SVD approach specifically for this case was proposed in [4]. The authors utilize a binary preference matrix and further introduce a confidence for each preference value. The core idea is that if a user has shown interest in an item several times, it is more likely that the preference assumption is correct. The authors propose to encode this confidence as

$$c_{u,i} = 1 + \alpha r_{u,i}, \tag{9}$$

where α is a scaling factor, typically set to $\alpha = 40$.

With these modifications, the SVD minimization task can be written as:

$$\min_{x^*,y^*} \sum_{u,i} c_{u,i} \left(p_{u,i} - x_u^T y_i\right)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right). \tag{10}$$

This formulation is very similar to the minimization problem of traditional SVDs (cf. Equation 6) with the addition of the confidence value. However, a key difference is that the optimization task involves all possible user-item pairs and not only the observed ratings since combinations where no preference was detected are assumed to be zero. Considering that the number of items and users in any reasonable real life scenario will be several thousand each, the user-item matrix often contains billions of entries. Traditional algorithms usually do not scale well to such large datasets and optimization, e.g., using a stochastic gradient descent optimization, can be very time consuming. The authors avoid this problem by noting that the cost function becomes quadratic when user factors or item factors are fixed. Therefore its global minimum can be easily computed. As a consequence the authors propose an Alternating Least-Squares (ALS) approach where user and item factors are optimised alternatingly until they stabilize. This vastly improves the training time.

C. Co-clustering

Co-clustering [13] is a simple alternative to k-NN. It simultaneously attempts to identify groups of similar users and items, called user-clusters C_u and item-clusters C_i. Based on these groups, co-clusters are identified which relate users and items. The underlying idea is similar to k-NN, where similar users or items are first identified and their rating is used to generate the prediction. In Co-clustering, the prediction can be calculated from the average rating of the assigned clusters:

$$\hat{r}_{u,i} = \overline{C}_{u,i} + \left(\overline{r}_u - \overline{C}_u\right) + \left(\overline{r}_i - \overline{C}_i\right). \tag{11}$$

Here, $\overline{C}_{u,i}$ is the average rating in the co-cluster, $\overline{C}_u$ the average rating in the user-cluster and $\overline{C}_i$ the average rating in the item-cluster.

The cluster assignment strategy is similar to the popular k-Means algorithm [14]. First, users and items are randomly assigned to a user or item cluster. The co-clusters are formed implicitly between all pairs of item- and user-clusters, e.g., all users in the first user-cluster and all items in the first item-cluster form one co-cluster. The number of user-clusters k_u and item-clusters k_i are user chosen and the number of co-clusters is $k_u \cdot k_i$. Afterwards, the cluster assignment for users and items is iteratively updated such that the squared prediction error is minimized.

III. Omni-Channel Recommender System

In general, recommender systems based on explicit feedback have been well studied. However, less attention has been given to implicit feedback where user preference needs to be inferred. In the following, we first show how to implement a recommender system based on purchase history. We discuss important design decisions such as how to capture the preference from the purchase history, how to handle returns and whether to recommend individual products or product categories. Afterwards, we turn to other data sources and show how to collect user preference and use it to improve the recommendations.

The vast majority of recommender systems with implicit feedback considers the usage of a service as an indication of preference. As an example, [12] uses the purchase history of customers and [4] uses the watching time of television programs. In the following, we use the example of purchase history but it is equally applicable to other types of service delivery. The purchase history is typically available as a long list of transactions which contain the purchased items, quantity and date of purchase for a user. For recommendations, it is necessary to transform this list into a user-item matrix where rows denote a user and columns an item. As a cell content, the sum of purchases aggregated per user and item can be used.

Generally, this already allows to apply traditional recommender approaches such as k-NN algorithms. However, a problem is that there is no indication of disfavour with this approach. Usually, the lack of disfavour is solved by assuming that all missing user-item pairs are zero. This is sometimes referred to as all-missing-as-negative [15], [12]. This is a strong assumption, since a lack of purchase could also indicate that the user has not discovered the product yet or not purchased it for another reason, despite a preference for it.

Strictly speaking, purchases also do not necessarily indicate a positive rating of the product. For example, customers might regret their purchase and could even return it. However, a user might also return products for other reasons, e.g., because a piece of clothing was not in the correct size. For this reason, we still consider the initial purchase as an indication of preference. Returned products are usually stored as a separate transaction with negative quantity. Therefore, we ignore those transactions and only work on transactions with positive quantities.

An additional problem is that implicit feedback is merely an indication of preference rather than a rating value. This means that there is no scale for the ratings and the values can vastly differ between users and items. To mitigate the lack of scale and its influence, the user-item matrix can be transformed into a binary matrix of preference or no preference. This has been shown to produce better results [12], [4] which we could confirm during the development of our approach. For this reason, we adopt this approach in our algorithm. Note the special case of SVD for implicit feedback using ALS (cf. Section II-B), where the algorithm uses a confidence value for every user-item rating even with a binary preference matrix.

Another important choice is whether to recommend previously purchased items to a user. On the one hand, recommender systems should help customers discover new items. On the other hand, customers are known to repurchase items, e.g., to replace a broken item or have multiple items in the same design. This is particularly true for our case study in the textile & home dećor business where customers purchase the same item repeatedly in order to have a consistent design. In our dataset roughly 8.5% of transactions are repurchases of the exact same item. For this reason, we allow recommendations of already purchased items.

Due to their complexity many recommender systems are implemented on product categories which vastly reduces the number of user-item pairs. However, individual products are arguably more helpful to the user and can be much more tailored to his or her preference. For this reason, we focus on product-level in the following.

A. Conceptual Extension of Purchase History

While the purchase history of a customer can already yield good recommendations, there is much more information available in order to improve the recommendations. As an example, when a user visits a product page on the website or searches for a product in the online store, we can infer that the user is interested in the products, even if no purchase is made. Similarly, if a user clicks a link in a newsletter, or likes a picture of a product in social media it is likely that the user has an interest. These indicators can be extracted from weblogs, external social media data, or newsletter click rates and used for the recommendations. This approach can be considered as an omni-channel recommender system since it can cover all available data sources and channel information in order to build an improved model [16], [17]. This allows to collect much more information about the customer and personalize recommendations better. In this approach, each channel produces a set of preference indicators which can be combined as shown in Figure 3. In our scenario, we use the sum across all matrices, but it is also possible to weight the information by importance. For example, the purchase history could be considered more relevant than data from social media.

In the following, we explain how to extract preference indicators from all available data sources in our case study of a retailer in the home furnishings and textiles sector. The data contains more than 2.2 Million transactions for $13,000$ different products from June 2014 to December 2017. Each transaction is characterized by its time of purchase, quantity of items and where it was performed. Sales in brick and mortar stores vastly outnumber online shoppers which constitute only 2.8% of the total sales. In total, more than 4.8 Million units of $13,000$ different products were sold. Additionally, the retailer offers a loyalty program to customers which we focus on here. Almost $800,000$ customers joined the loyalty program and more than $500,000$ made a purchase in our dataset. For these customers, additional demographic information such as age and gender is available. Roughly 90% of loyalty members are female and the mean age of customers is 49.6 years.

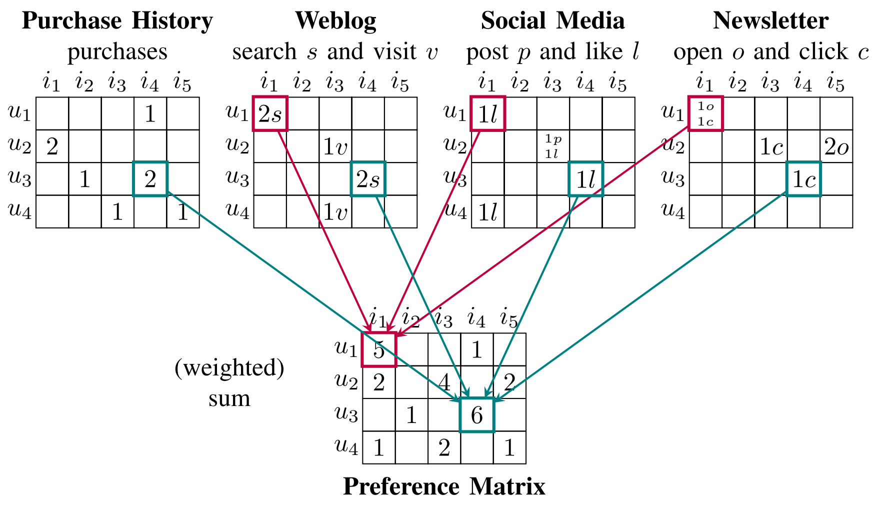


Figure 3. Example of preference collection from purchase history, weblogs, social media and newsletters with equal weighting.

B. Weblogs

These days, many retailers maintain an online store which produces a large number of information, often captured as a log file about the user's browsing behaviour. This information can be mined to improve the recommendation process. As an example, online stores usually offer a search field where customers can search for products or attributes of products that they are interested in. This is a strong indication of interest in a specific product or feature which can be used to improve recommendations.

However, this approach comes with several challenges. First, website visitors are generally anonymous and are only losely identified by their `IP`-address or the user-agent of their browser. This makes it difficult to associate an observed search query with a specific customer. Only after a user logs in or makes a purchase, it is possible to associate a real customer with the browsing session. To overcome this problem, websites typically use `HTTP`-cookies which are placed on a visitors machine and are automatically transferred to the website upon visit. In our scenario, we can place a cookie with a unique identifier on the visitors machine and associate that identifier with a user account upon login. This allows to uniquely identify visitors even if they are not logged in until the cookie is removed.

As a second problem, it is necessary to identify products that match the search query of the user. Ideally, we can use on the built-in search algorithm of the website and use the results to identify matching products. Unfortunately, the search algorithm and results of the website are not available in our case. Therefore, it is necessary to match the search query to the available products separately, e.g., by using a string matching approach.

A natural choice to compare the similarity of strings is the Levenshtein distance [18]. The Levenshtein distance is the minimal number of insert, delete and replace operations that is required to transform one string $\boldsymbol{X}$ into another string $\boldsymbol{Y}$. However, it is not directly applicable in our scenario since it does not handle strings of different lengths or different order of words well. Therefore, we use a partial string matching strategy called token set ratio [19]. The core idea is to tokenize the string into individual words and split them into different sets: an intersection and two remainders. Each set is sorted alphabetically in order to handle cases where words are in different order. Specifically, the string t_0 is the ordered intersection of words in $\boldsymbol{X}$ and $\boldsymbol{Y}$. Based on this, t_1 and t_2 are the string t_0 concatenated with the sorted remainder of $\boldsymbol{X}$ or $\boldsymbol{Y}$ respectively:

$$t_0 = \mathrm{sort}(\boldsymbol{X} \cap \boldsymbol{Y}) \tag{12}$$
$$t_1 = t_0 + \mathrm{sort}(\boldsymbol{X} \setminus t_0) \tag{13}$$
$$t_2 = t_0 + \mathrm{sort}(\boldsymbol{Y} \setminus t_0) \tag{14}$$

For each pair, a ratio of relative similarity is computed as $2M/T$ where M is the number of matches and T the total number of characters. The final matching score can then be calculated by taking the maximum similarity of all pairs. The underlying idea is that the similarity is large if $\boldsymbol{X}$ makes up a larger portion of $\boldsymbol{Y}$ or if the remainders are similar.

As a simple example let us assume that the search query is $\boldsymbol{X}$ = "red towel" and one of the product names is $\boldsymbol{Y}$ = "green towel". In this case the three strings are as follows: $t_0 = \text{towel}, t_1 = \text{towel red}, t_2 = \text{towel green}$. The pairwise similarity scores yield $\mathrm{sim}(t_0, t_1) = 0.71, \mathrm{sim}(t_0, t_2) = 0.62$, $\mathrm{sim}(t_1, t_2) = 0.80$. The similarity between t_1 and t_2 for example is calculated as $(2 \cdot 8)/(9 + 11)$ where 8 is the number of matching characters "towel re" and the length of the strings is 9 and 11 respectively. Therefore, the final similarity score is 0.8, i.e. the maximum of the three pairs.

To decide whether a product matches the search query, we compare its similarity to its product name, category, and description and employ a similarity threshold of 80%. Since our approach performs a partial string matching, we are not confined to compare product names. Instead, we also look for descriptions and product categories that match the search. If the similarity of the search query with any attribute is sufficiently high, we assume a preference for that product.

In our dataset, a total of 9 Million weblogs are available, recorded between June 2017 and February 2018. The weblog contains all `HTTP GET` requests made to the website during this period and allows to draw conclusions about product pages a user visited or what the customer searched for. Note that despite the rather short time-period, weblogs form one of the largest data sources available in our study. A total of $16,000$ weblogs could be matched to a specific customer of the loyalty program by identifying the cookie on their machine. In our dataset, search queries can be identified from the requested `URL` since each search query is prefixed using "`/q=`" after the toplevel domain. This allowed us to identify $180,000$ different search queries and 687 of them can be attributed to a specific customer of the loyalty program by identifying the cookie on their machine. For all search queries, we attempt to match the search query to one or more products in our database using the above string-matching scheme and add the identified preferences to our preference matrix. An exemplary search query, the corresponding log file and similarity to products is shown in Figure 4.

Additionally, weblogs can also be analysed to observe click-streams and product pages a customer visited. The underlying assumption is that if a user looked at the product page, he or she has a general interest in that product. In an ideal case, it should be known which products are associated with a specific `URL`. Unfortunately, this information is not available in our dataset which makes it necessary to match the `URL` against products in our database again (Figure 5). For this we extract the name of the requested `HTML` site and utilize the string matching scheme from above. If the name matches an entry in our database, we again record it as a preference for that product and add it to the user-item pair in the preference matrix.

C. Social Media

On top of online stores, many companies maintain a social media account on platforms such as Facebook, Twitter or Instagram. Typically, these accounts post news about products, run marketing campaigns, engage with users and provide customer service [20]. Social media platforms usually come with a feedback system such as likes, shares or comments. This makes them a valuable source to identify preferences of the users. If the post mentioned a specific product, it can also indicate preference for that product. Assuming that a company posted a facebook picture of a new product and the user liked and commented the picture, there is a high probability that the user is interested in the product.

To capture the user preference from such information, it is first necessary to identify which product was mentioned in a post. In an ideal case, this information is stored by the social media team and readily available. Since this is not available in our case, we propose several strategies. The most straightforward approach uses text-based posts or description of videos and photos. For this type we can use the text to perform the partial string matching strategy from above. The same strategy can be used for videos and pictures which usually come with description texts.

A special form of this is possible on Facebook, where products in a photo can be tagged by the company. If a user selects one of the tagged products, a page with item price and link to the online store is shown. For this special case we use the Facebook-API and query the name of the tags. These are usually abbreviations of product names and can be used for string matching. Additionally, companies sometimes post links to their website, e.g., to product categories or specific product pages. These links can be analysed with the same strategy as previously described for weblogs.

As before, another challenge is to associate the largely anonymous accounts in social media with real customers in the database. To solve this, a simple name matching strategy can be used. In our dataset, users are identified based on a match of their first and last names. Note that this is bound to miss some cases due to the large number of accounts with fake, abbreviated or modified names on social media. Additionally, it might match customer or social media accounts with popular names multiple times. More elaborate string matching could try to counter name obfuscations with the risk of misidentifying users and could prevent double matches. We apply the described approach to a total of $5,000$ user comments and $120,000$ user likes from the company's Facebook page. The posts range from the start of the account in 2011 until the end of 2017.

D. Newsletters

More indicators of preference can be obtained from interest in newsletters. Many companies regularly send out newsletters either to inform customers about new products or provide discounts as an incentive to make a purchase. They are typically sent by email and their usage can give feedback about what customers are interested in. The underlying idea is that if a customer clicks a link in a newsletter, he or she is likely to have a preference for the advertised item.

Our dataset contains information about almost $800,000$ newsletters sent roughly every 2–7 days between June and July 2017. $176,000$ mails were opened and $29,000$ click-throughs were recorded. Unfortunately, no detailed information is available in our dataset regarding which products were advertised in a newsletter. The only available attribute is the header of the email. While we recognize that

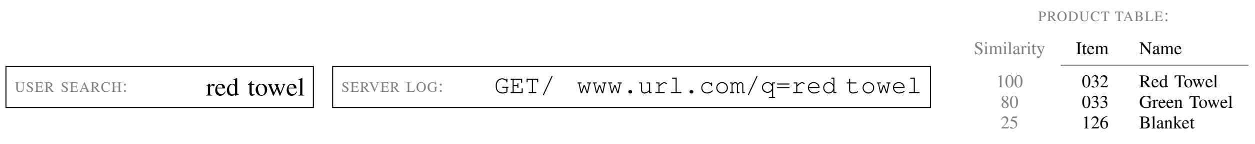


Figure 4. Matching search queries to product information.

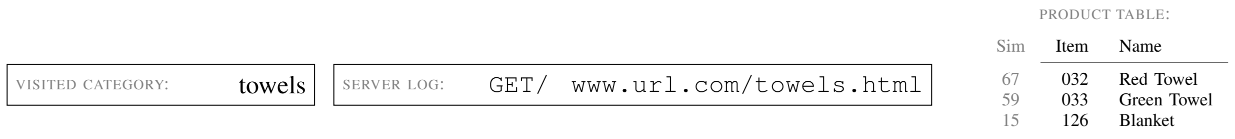


Figure 5. Matching page visits to product information.

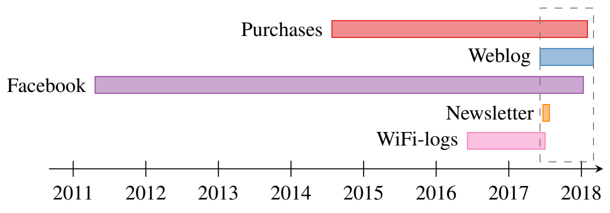


Figure 6. Timeline of available data sources.

this makes it difficult to relate newsletters to products, we match the header with the string matching from above. In the future, the content of newsletters should be stored more consistently to improve this approach.

IV. Evaluation

A. Experimental Setup

The goal of our recommender systems is to recommend the most suitable products per user based on the available information. Figure 6 summarizes all data sources and available time-ranges in our dataset. As indicated, the largest overlap of available data is between June 2017 and January 2018. In the following, we perform our experiments on this time-span since it is most relevant for our approach.

To evaluate a recommender system, we predict the user preference for every user and item combination in our dataset. We can then order the predicted scores from most suitable to least suitable product per user to obtain a list of recommendations. We evaluate Singular Value Decomposition (SVD), its extension using Alternating Least Squares (ALS), Co-Clustering as well as Item-based CF. For SVD we use 100 latent factors, 20 epochs and set the parameters $\gamma = 0.005$ and $\lambda = 0.02$. Note, to achieve a faster runtime of the SVD approach we parallelized its execution by removing any restrictions regarding the access of shared data structures to store the user and item factors. This allows race-conditions but during our tests with smaller datasets, the results show no negative impact on the quality of the results. For Co-Clustering we use 5 user and item clusters and 20 epochs and for Item-based CF we use $k = 10$ Nearest Neighbours. These settings are fairly standard but parameter tuning might improve the algorithms further. For comparison, we also include a baseline algorithm which randomly samples values with the same probability as observed in the training set.

To validate the list of recommendations we use a train-and-test strategy where a training set is used to train the model and a test set is used to evaluate it. In traditional supervised machine-learning, the entire dataset can be split in order to achieve this. For recommender systems, however, the full user-item matrix needs to be available. As an alternative, a popular strategy masks certain elements from the user-item matrix by setting them to zero. For purchase data this means that some of the purchases are removed from the training set. The algorithm is then trained on this data and asked to generate a list of recommendations for every user. Ideally, the purchases that were masked from the training data are then recommended as the most suitable items.

An example of this train-and-test strategy is shown in Figure 7. In the following, we randomly mask 20% of the data set. Even though we use a random masking strategy, we observed the results to be stable regardless of which ratings were masked. Other approaches, such as masking all purchases after a specific date are equally applicable.

Recommender systems based on explicit feedback can be evaluated by comparing the rating estimate with the true rating. As an example, they try to estimate the number of stars a user would give to a movie. Divergence from the true rating can then be measured by using the Root Mean Square Error (RMSE) or the Mean Absolute Error (MAE). Algorithms that minimize either measure are desirable. Unfortunately, this approach is not directly transferable to implicit feedback where only preference indicators are available. As a consequence, the values either have no fixed scale or are binary which makes precision-based metrics hardly applicable [4].

Instead, the quality of recommendations can be evaluated by determining whether more suitable items are more likely to be recommended. In our setup, we employ the approach

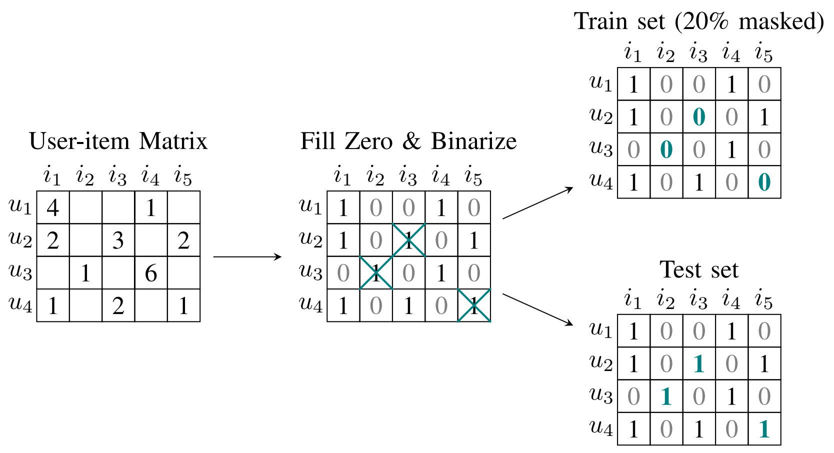


Figure 7. Masking strategy in our evaluation. All ratings are binarized and missing ratings are assumed to be zero. The training set masks some ratings to be zero.

Table I
EXAMPLE OF USING THE PERCENTILE RANK OF CORRECT RECOMMENDATIONS AS A QUALITY MEASURE.

Product	$rank_{u,i}$	$r_{u,i}$	$r_{u,i} \cdot rank_{u,i}$
Red Towel	0%	1	0%
Green Towel	3%	1	3%
Blue Towel	6%	1	6%
		⋮	
Red Blanket	97%	0	0%
Blue Blanket	100%	0	0%

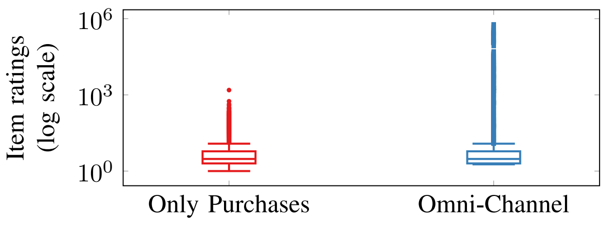


Figure 8. Boxplots of item ratings per user on a logarithmic scale using purchase data as well as all available sources.

from [4] and determine whether items of interest rank higher in the list of recommendations. In this evaluation strategy, $rank_{u,i}$ denotes the percentile-rank of an item in the list of recommendations. $rank_{u,i} = 0\%$ means that a product is the first item of the recommendation list, i.e. deemed most suited for the user, and $rank_{u,i} = 100\%$ that it is the least suited for the user. To evaluate all recommendations, the weighted average of percentile-rankings can be used.

$$\overline{rank} = \frac{\sum_{u,i} r_{u,i} \cdot rank_{u,i}}{r_{u,i}} \tag{15}$$

The underlying idea is that if a customer has shown interest in a product, we want that product to rank high in the list of recommendations. An example of this idea is shown in Table I where $r_{u,i}$ is the true rating taken from the test-set, i.e. whether the user purchased the item. By multiplying it with the respective percentile-rank, we can assess whether good recommendations rank higher in the list. A good recommendation algorithm will therefore yield a lower $\overline{rank}$, whereas a random algorithm will yield values close to 50%, i.e., placing the relevant items in the middle of the recommendation list.

B. Results

In our experimental evaluation, we aim to show that our approach is able to extract better indication of preference from the user than traditionally available in the purchase history. Figure 8 compares how many item-ratings are available per user for the traditional approach using purchase history and our omni-channel approach. It is obvious that we were able to extract vastly more preference indicators which allows us to make more tailored recommendations. For some users in particular, a lot more information could be gathered as highlighted by the increased number of outliers in the boxplot. In addition, there are fewer customers where no information is available as shown by the positive shift of the lower whisker. Note that this also includes users which have not made a purchase in the past and would usually be unknown to the recommender system. Figure 9 highlights how much each channel contributed to the improved collection of preference indicators. In total 3.87 million preference indicators could be collected in our approach which is almost 6 times more than from purchase data alone. 1 Million of these preferences were previously unknown, i.e., zero. The most information could be extracted from the website visits which we also believe to be among the more accurate channels.

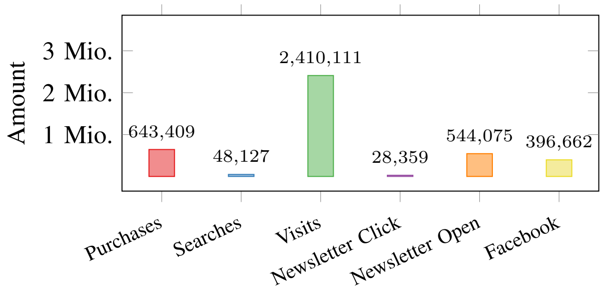


Figure 9. Preference indicators per channel.

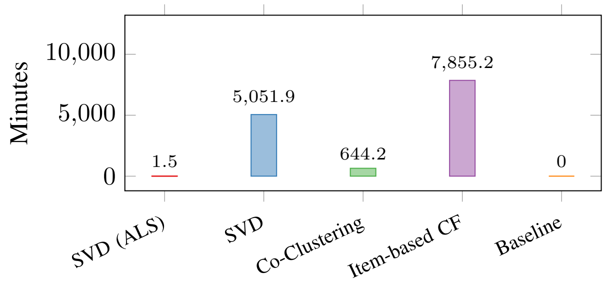


Figure 11. Training time for all evaluated algorithms.

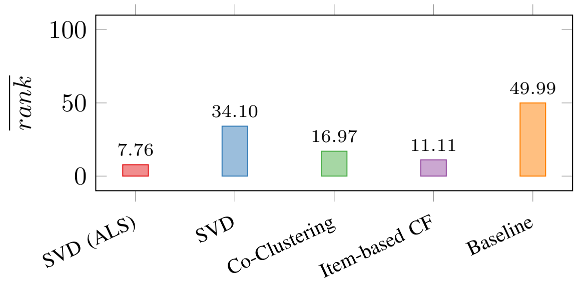


Figure 10. $\overline{rank}$ measure for all evaluated algorithms.

Additionally, we want to validate that the algorithms are recommending the right products to customers. For this, Figure 10 shows the $\overline{rank}$ measure for all evaluated algorithms. The results show that SVD using ALS yields the best recommendation result and outperforms all remaining algorithms with $\overline{rank} = 7.76\%$. This is a very good result and shows that the vast majority of relevant items also rank high in the recommendations list. As expected, the baseline algorithm yields roughly $\overline{rank} = 50\%$ by randomly recommending items. While traditional SVD shows the weakest performance in our study, its rank is still better than random guessing.

Item-based CF yields slightly worse results than SVD using ALS but takes considerable longer to compute. To highlight this, Figure 11 shows the training time of each algorithm. Since we parallelized all algorithms we show the total runtime on all CPUs, including some parallelization overhead. Again, SVD using ALS is the fastest of the tested algorithms and vastly outperforms traditional SVD and Item-based CF.

V. Timing of Recommendations

While recommendation accuracy is important, the timing of a recommendation is also of interest. For example, there is little value in recommending winter clothes during the summer and there are times where a customer is more welcoming to recommendations. One way to improve the timing is by recommending products to customers who seem to have interest in the brand. For example, when a user posts on social media or regularly opens newsletters, it is likely that the customer enjoys the products. It is a good idea to target these users in the recommendation process.

Another way to improve the timing is to detect when customers visit a store but leave without purchase. This could indicate that the user was interested but did not find a suitable product. This can be easily done in the online store but is also possible in brick and mortar stores. For this, the stores' `WiFi` routers can be configured to monitor `WiFi` probe and authentication requests. Such requests are sent from devices such as mobile phones in order to identify whether there is a known network available. These requests allow to identify `MAC`-addresses of all devices in close proximity to the store. Note that this practice might be subject to regulations in some legislatures.

In order to match a `MAC`-address with a specific customer, a simple matching scheme can be used. When a loyalty card is used while a `MAC`-address is in proximity, it is more likely that the device belongs to the customers. If this combination of `MAC`-address and loyalty card occurs repeatedly, it can be assumed that the device belongs to the customer. Our dataset contains a total of $4,800$ `MAC`-addresses that could be associated with a customer. When a customer is now recognized in the store but leaves without purchase, it can be a good opportunity to trigger a recommendation and send the customer a voucher or advertisement for products that are most suitable.

VI. Conclusion

Traditionally, recommender systems use the the purchase history in order to recommend suitable products. In this paper, we have shown that it is possible to use additional information such as online search behaviour, website visits, newsletter open and click rates as well as likes and comments from social media data in order to obtain better indication of preference. In a real-life case study of a retailer we were able to obtain almost six times more preference indicators, many of which were previously unknown. Our approach also allows us to make recommendations for new customers and thus can remove the usual cold-start problem of recommender systems.

We demonstrate how to infer preference from each channel, e.g., by analysing weblogs or crawling social media data. On top, we also propose several strategies on how to time recommendations, i.e. when to send recommendations to the customer. Our results show, that the recommendations are very accurate. Nevertheless, the performance measurement of recommender systems remains difficult. We used standard measures to ensure that the system recommends items that we deem interesting to the user. However, ultimately, recommender systems need to be evaluated in practice where turnover rates are compared for different strategies.

An open challenge remains the identification of customers across the different channels. The current approach is based on string matching of social media names or `HTTP`-cookies. These approaches are prone to errors and users might be misidentified, e.g., because of similar names. Additionally, it is sometimes difficult to link the user behaviour to their preference. For example, it can be difficult to attribute a like on an image to one specific product, e.g., because multiple items are shown in the image. Additionally, other sophisticated string matching approaches need to be considered in the future. Finally, other types or recommender systems such as hybrid or content-based approaches could be explored.

Acknowledgment

The authors acknowledge support from the European Research Center for Information Systems (ERCIS) as well as the "ERCIS Omni-Channel Lab – powered by Arvato".

References

[1] J. B. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM Conference on Electronic Commerce*, ser. EC '99. New York, NY, USA: ACM, 1999, pp. 158–166. [Online]. Available: http://doi.acm.org/10.1145/336992.337035

[2] B. Liu, *Web Data Mining*. Springer Berlin Heidelberg, 2011.

[3] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction*, 1st ed. New York, NY, USA: Cambridge University Press, 2010.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *8th IEEE International Conference on Data Mining*, 12 2008, pp. 263–272.

[5] B. Smith and G. Linden, "Two decades of recommender systems at amazon.com," *IEEE Internet Computing*, vol. 21, no. 3, pp. 12–18, 5 2017.

[6] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems*, vol. 6, no. 4, pp. 13:1–13:19, 12 2015.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 8 2009.

[8] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*. Springer, Boston, MA, 2011.

[9] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, Jan. 2004. [Online]. Available: http://doi.acm.org/10.1145/963770.963772

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 285–295. [Online]. Available: http://doi.acm.org/10.1145/371920.372071

[11] S. Funk. Netflix update: Try this at home. [Online]. Available: http://sifter.org/simon/journal/20061211.html

[12] B. Pradel, S. Sean, J. Delporte, S. Guérif, C. Rouveirol, N. Usunier, F. Fogelman-Soulié, and F. Dufau-Joel, "A case study in a recommender system based on purchase data," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: ACM, 2011, pp. 377–385.

[13] T. George and S. Merugu, "A scalable collaborative filtering framework based on co-clustering," in *Proceedings of the Fifth IEEE International Conference on Data Mining*, ser. ICDM '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 625–628.

[14] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theor.*, vol. 28, no. 2, pp. 129–137, 9 2006.

[15] R. Pan and M. Scholz, "Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 667–676.

[16] M. Carnein, M. Heuchert, L. Homann, H. Trautmann, G. Vossen, J. Becker, and K. Kraume, "Towards efficient and informative omni-channel customer relationship management," in *Proceedings of the 36th International Conference on Conceptual Modeling (ER '17)*. Springer International Publishing, 2017, pp. 69–78.

[17] H. Trautmann, G. Vossen, L. Homann, M. Carnein, and K. Kraume, "Challenges of data management and analytics in omni-channel crm," European Research Center for Information Systems, Münster, Germany, Tech. Rep., 2017.

[18] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[19] SeatGeek, *FuzzyWuzzy Python Library*, 2017. [Online]. Available: https://github.com/seatgeek/fuzzywuzzy

[20] M. Carnein, L. Homann, H. Trautmann, G. Vossen, and K. Kraume, "Customer service in social media: An empirical study of the airline industry," in *Proceedings of the 17th Conference on Database Systems for Business, Technology, and Web (BTW '17)*, 2017, pp. 33–40.