Predicting User Flight Preferences in an Airline E-Shop

Gaby Budel, Lennart Hoogenboom, Wouter Kastrop,
Nino Reniers, and Flavius Frasincar(✉)

Erasmus University Rotterdam,
PO Box 1738, 3000 DR Rotterdam, The Netherlands
{385330gb,388233lh,457571wk,373768nr}@student.eur.nl,
frasincar@ese.eur.nl

Abstract. With the continuous development of the Web, it is becoming increasingly important for e-shops to present customers with the most relevant products. The personalisation of product rankings is one of the problems associated with this development. Until now, the focus in the field of machine learning has mainly been on the ranking of documents The ranking of items in general asks for new types of features, that accurately describe the match between query and item. We propose the usage of cross-terms between item-specific and user-specific variables in the Ranking SVM algorithm. We apply these new features for the ranking of flights on the website of a company in the airline industry. For our data, the cross-terms improve the out-of-sample accuracy of the Ranking SVM with 2.11% points compared to a baseline. Due to the high amount of traffic on the Web, improvements like this can already have a big impact on users' purchase activity.

Keywords: Learning to rank · Implicit feedback
Personalised item ranking · User flight preferences · Web shop

1 Introduction

Due to continuous advancements in the digital world, Web commerce is becoming increasingly important for companies. This development stresses the companies' need of having a high quality website. The ongoing evolution of data analytics provides companies with more insights in the characteristics their customers' online behaviour [1]. These insights in behaviour can be used to provide customers with relevant product suggestions. Since customers usually only spend a limited amount of time on a website, it is crucial to present them with the most interesting offers during their visit. These personalised recommendations require a thorough understanding of the factors that drive a customer when purchasing a product. One way of providing customers with these offers is a relevance-based product ranking on individual level. In this study we aim to provide such a ranking for a company in the airline industry. In the past, various approaches have been proposed to solve the problem of making relevance predictions for products or related items. Many of these approaches were in the field of information retrieval, which originally focused on the ranking of documents. However, information retrieval has also been applied to the field of e-commerce [2,3].

Another class of methods that aims to provide a relevance-based ranking, is the recommender system [4,5]. These methods are mainly applied to constant product assortments. Although these methods have proven to be very accurate in predicting product relevance, they tend to suffer from so-called cold starts. These cold starts indicate that the model does not perform well for customers or items on which it has not gathered sufficient information yet. For some e-shops these are serious concerns, since new customers are introduced every day and product assortments change rapidly [6]. Failing to deliver personalised recommendations can lead to customers prematurely leaving the website, causing the company to miss out on revenue in current and potential future sales. We propose a method that avoids the problem of cold starts by obtaining relevancy rankings merely based on information retrieved during a customer's session on the website domain. If the customer closes the web page and decides to return at a later moment, recommendations will be solely based on information from that new session.

Another difficulty for e-shops, besides cold starts, is that a customer may have different motivations per website visit. To give an example from the airline industry, a customer can book a flight for both a business meeting and a holiday in the very same week. A well-personalised website is able to capture both motivations, requiring customer preferences to be updated during a website visit. To enable websites to capture these motivations based on real-time purchase behaviour, Latent Dirichlet Allocation (LDA) can be applied [7]. However, LDA has the disadvantage of not being able to update customer characteristics in real time.

For real-time updating of customer preferences more flexible methods are required, which can be found in the field of machine learning. In the last decade, the ranking of items gained more and more attention in machine learning algorithms, gathered under the name of learning to rank (LETOR). The purpose of these algorithms is to rank unseen items based on user preferences. Within LETOR there are three main approaches: the pointwise, pairwise, and listwise approach. While the pointwise approach evaluates each item individually, the pairwise approach focuses on item pairs. Both approaches implicitly optimise the item ranking by obtaining a relevance score for each item or pair, respectively. The pointwise approach needs implicit feedback on all items, where for the pairwise approach it is sufficient to have partial feedback on paired comparisons. The listwise approach solves the ranking problem by explicitly optimising the entire ranking. However, a drawback of the listwise approach is the complexity of its loss function. As the data set at hand contains partial feedback, we employ a pairwise approach.

We propose a novel method for Personalising A Ranking Using Cross-terms (PARUX) which determines the features that drive customers, and provides an item ranking function which can be updated in real-time. In the past, feature-based ranking functions could not incorporate user characteristics as ranking criteria, simply because the characteristics do not differ over the items to be ranked. We introduce the inclusion of user characteristics in the Ranking SVM methodology of [8] by setting up cross terms with the characteristics. To the best of our knowledge, this set-up has never been applied for the ranking of items. In the next section, we present the related work that gave rise to this study, followed by a framework for our method. Subsequently, we discuss an implementation in the airline industry, followed by the performance evaluation. Lastly, the conclusions are drawn and suggestions for further research are given.

2 Related Work

Although a lot of research has been done in the field of learning to rank (LETOR) [9–11], the main focus of this research area has been on the relevance ranking of documents or Web pages. In this context, the used algorithms aim to retrieve and rank documents that match a query specified by the user. To capture the similarity based on features, one can often make use of common feature weighting measures such as TF.IDF and BM25 [9]. However, once the application of the ranking task switches to e-commerce and purchase recommendation, no such common measures exist. One of the challenges of LETOR in e-commerce is to identify the user's preferences with a low cognitive load on the user.

Coyle and Cunningham were the first to apply the concept of LETOR to the online recommendation of flights [12]. To be able to present the users' most preferred flights in descending order, they proposed a so-called Case-based Reasoning. This method is based on similarities of previous accepted offers, as those are presumed to contain the user's preferences. These extracted preferences are used to construct feature weights. Once these feature weights are known, the ranking of the flights boils down to a straight forward multiplication of the features and their weights. Their method is shown to achieve a recommendation accuracy of at least 85% for each of the users. However, before the method is able to accurately approximate the preferences, several purchases must have been made. Moreover, since only one set of preferences is identified for each user, the approach cannot handle the fact that a customer might have different motivations when booking flights.

In 2013, Ostuni et al. proposed an entirely different approach to construct a personalised ranking based on positive implicit feedback hidden in Web data [13]. Their method aims to solve this task by integrating both the content-based and collaborative features in the same feature space. Representing this entire feature space in an extensive graph allowed them to use path-based feature extraction, which describes similarities between the items preferred by the user and the items to rank. However, the proposed algorithm still needs a relatively large amount of feedback from each individual customer. In order to avoid overfitting, they propose to drop all users with less than 20 observations. This reveals the downside of this method for our setting, as such a vast amount of feedback is rarely available in the airline industry.

Other approaches have tried to personalise the product recommendation based on similarities between users, which has the advantage that no data on past purchases of an individual are required. In 2003, Freund et al. set up an experiment to improve the ranking of meta-search results and movie recommendations [14]. They proposed a pairwise LETOR algorithm based on the concept of boosting called RankBoost. The idea behind RankBoost is to take a weighted average over many (weak) ranking functions, resulting in a much more powerful ranking rule. While the authors show that for the meta-search results RankBoost performed just as well as the best search strategy, for the movie recommendation RankBoost outperformed the standard regression and nearest-neighbour algorithms. For a large problem setting, as we encounter in our work, RankBoost achieves an accuracy of about 65%, exceeding the runner-up by almost 5%. Although RankBoost is a very flexible and powerful algorithm, the method gives little insight in why a certain item is preferred over another.

The method proposed by Joachims in 2002 is an approach that is able to uncover those insights and has inspired PARUX [8]. His LETOR implementation of the Support Vector Machine (SVM), called Ranking SVM, is a pairwise approach that optimises the retrieval quality of search engines in the document domain. The set-up of this method is analogous to the classical SVM. Joachims shows in his paper that Ranking SVM exceeds Google's `PageRank` algorithm in terms of retrieving quality, already after a couple of hundred training examples. Since the method has shown to obtain accurate results when trained on relatively small groups of users, it should also be able to capture the preferences of specific, infrequent types of users. Therefore the approach can be very well applied in our goal of personalising the ranking of items. We extend the work of [8] by implementing Ranking SVM in an e-commerce setting. This gives rise to the challenge of coming up with adequate features that capture the similarities between the search query and the items to rank.

Another challenge, as stated in [15], is the fact that pairwise LETOR approaches can be very sensitive to noise in the data. When a ranking function is learned from clickstream data on a previous, non-optimised ranking, the observed preferences can contain biased information. Items that appeared in the top of the non-optimised ranking may have enjoyed customers' interest without being truly preferred. To minimise this bias, our work exclusively uses click logs that have led to an actual purchase. We argue that with the large number of different flight options and airlines available these days, a customer will only purchase a flight when it truly meets his or her preferences.

3 Framework

In this section we present a framework for PARUX, which ranks items in search results based on revealed user preferences. It is based on the methodology of Ranking SVM as presented by Joachims [8] that we extend to the context of ranking items in general. In addition, we propose a novel method to incorporate user characteristics in the ranking of items.

3.1 Learning to Rank for Items

The goal of LETOR is to train a ranking model for a set of items. Most LETOR algorithms are originally designed to rank documents. Fortunately, a feature-based formulation of this problem makes it extensible to items in general. In that case the task remains to find meaningful features. The idea behind a feature approach is that it reflects the relevance of an item to a query. One can regard this query in the broadest sense of the word, as it might also include customer characteristics. For some query q, one of its associated items i can be represented by a feature vector $\mathbf{x}_i^{(q)} = \Psi(i, q)$. Here, Ψ is a feature extracting function. The problem is now reduced to learning a ranking function from the feature vectors of the queries and items in a training set. The ranking function can then be used to rank the associated items of a new query, by using their feature representations. The task remains to find a suitable ranking function.

3.2 Ranking SVM

Ranking SVM is a pairwise approach to LETOR [9]. That is, the method considers items in pairs, where two items are represented by their feature vectors. Although the method trains on item pairs, the final goal is to rank a list of individual items by relevancy. The workflow of Ranking SVM is shown in Fig. 1. In this figure the preferred item in each query is marked with an asterisk, and the final ranking order is denoted by the superscript.

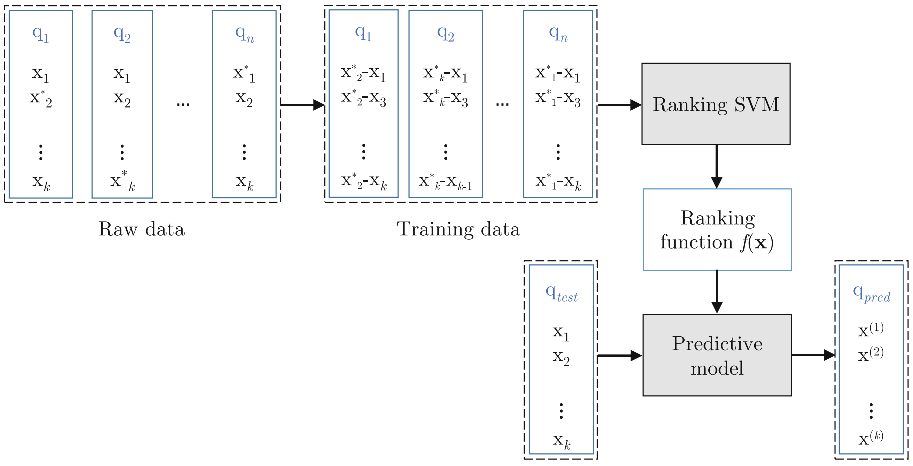

Fig. 1. Framework of ranking SVM

Ranking SVM requires ground truth labels for a target ranking. The goal is to minimise the number of pairs that are not in accordance with the target ranking, i.e., to minimise the number of discordant pairs. This minimisation can be shown to be equivalent to maximising Kendall's τ [8]. Here Kendall's τ is a measure for the quality of the ranking. Maximising τ is in turn equivalent to minimising the error of classifying the training sample. For a more detailed explanation on Kendall's τ, we refer to [8].

We introduce a (linear) scoring function and incorporate a margin in the ranking of the items. That is, for given weights we require that for every item i that is preferred over item j in query q it holds that

$$\mathbf{w} \cdot \mathbf{x}_i^{(q)} \geq \mathbf{w} \cdot \mathbf{x}_j^{(q)} + 1, \tag{1}$$

where $\mathbf{x}_u^{(q)}$ is the feature vector of item u and query q. Weight vector $\mathbf{w}$ indicates the importance of the features and has to be learned from the training data. The $+1$ term in (1) is the hard margin of the SVM, i.e., the constant in the linear combination. Note that we can obtain any margin size by multiplying the elements in $\mathbf{w}$ by a scalar. However, we follow the example of the regular SVM and restrict it to $+1$ [16]. The current formulation is not yet suitable for an SVM, instead we have to train it on the difference of the two feature vectors:

$$\mathbf{w} \cdot \left(\mathbf{x}_i^{(q)} - \mathbf{x}_j^{(q)}\right) \geq 1. \tag{2}$$

Once we have introduced slack in the constraints, we will show that it is not necessary to denote the -1 inequality, due to the pairwise differences. When using large data sets of user preferences, it is unlikely that these data can be perfectly ranked, causing the hard margin constraints to be too restrictive. We allow for misclassification in the training set by adopting the idea of a SVM with soft margins to the ranking context and by including a slack variable $\xi_{i,j}^{(q)}$ in the constraints [16]. We adapt (2) by allowing that the linear combination of the difference vector is less than or equal to $+1$ for some item pairs:

$$\mathbf{w} \cdot \left(\mathbf{x}_i^{(q)} - \mathbf{x}_j^{(q)}\right) \geq 1 - \xi_{i,j}^{(q)}. \tag{3}$$

Next, we should minimise the slack $\xi_{i,j}^{(q)}$. Using this formulation, it becomes apparent that Ranking SVM is equivalent to applying the regular classification SVM with soft margins to the difference of the feature vectors. In this case the classes are either item i is more relevant than item j, or the other way around. The total optimisation problem can now be formulated as follows:

$$\begin{array}{lll}
\min & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{q=1}^{n}\sum_{(i,j):y_{i,j}^{(q)}=1} \xi_{i,j}^{(q)} & \\
\text{s.t.} & \mathbf{w}\cdot(\mathbf{x}_i^{(q)} - \mathbf{x}_j^{(q)}) \geq 1 - \xi_{i,j}^{(q)} & \forall (i,j) : y_{i,j}^{(q)} = 1 \\
 & \xi_{i,j}^{(q)} \geq 0 & q = 1,\ldots,n
\end{array} \tag{4}$$

In this formulation $y_{i,j}^{(q)} = 1$ if item i is preferred over item j in query q according to the target ranking, and $y_{i,j}^{(q)} = -1$ if otherwise. Furthermore, slack variables

$\xi_{i,j}^{(q)}$ indicate the distance of the difference vectors to the margin requirement. Cost parameter C gives a penalty to a non-zero value of $\xi_{i,j}^{(q)}$ and allows for a trade-off between margin size and misclassification in the training sample. These slack variables are combined with the complexity control term $\frac{1}{2}\left\|\mathbf{w}\right\|^2$ into the minimisation objective function. Note that since we train on difference vectors, the constraint for the -1 group of the SVM boils down to the same constraint as for the $+1$ group:

$$(-1)\left(\mathbf{w}\cdot(\mathbf{x}_i^{(q)}-\mathbf{x}_j^{(q)})\right)\geq 1-\xi_{i,j}^{(q)} \iff \mathbf{w}\cdot(\mathbf{x}_j^{(q)}-\mathbf{x}_i^{(q)})\geq 1-\xi_{i,j}^{(q)}, \qquad (5)$$

with the only difference being that the indices i and j are swapped. Therefore, it is sufficient to denote only one type of constraint.

In Ranking SVM the margin is defined as the distance between the closest two projections onto the weight vector, out of all the projections of the items in the target ranking. Figure 2 illustrates this distance geometrically, denoted by δ, for an instance with 4 items. As discussed, we set the margin δ to 1. In the case with soft margins, this size is traded off against the number of misclassifications in the training sample. In econometric literature, the loss function in Ranking SVM is usually defined as a hinge loss on the item pairs. The hinge loss implies that when an item pair is classified correctly according to the target ranking with a distance between the projections of at least 1, there is no loss. Only when the distance is smaller than the margin or when the pair is incorrectly classified, the loss is equal to $\xi_{i,j}^{(q)} > 0$.

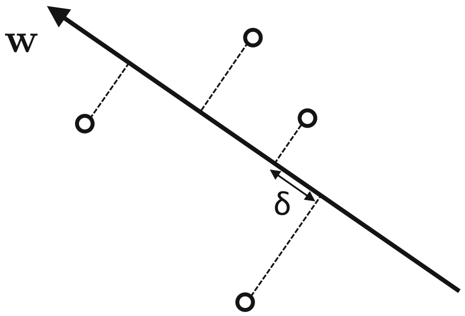

Fig. 2. Geometric interpretation of the margin in Ranking SVM

3.3 Ranking SVM Using Clickstream Data

In practice the complete target ranking of a query is rarely available. Users of a search tool only reveal their preference by selecting or buying the most relevant item or product. Therefore, clickstream logs contain only partial feedback from the user. However, Ranking SVM can be adjusted for a single 'winner' versus multiple 'losers' concept. The SVM is then trained on a subset of the true target ranking, namely item pairs containing the winner and one of the losers. It is sufficient to consider only a random subset of the losers, instead of creating all possible winner-loser pairs [17]. In this way, the SVM ignores interdependency among the losers, since the user did not reveal his preference among these items. The learned retrieval function from Ranking SVM can be used to construct the final ranking. From this point onwards, it is no longer necessary to consider the items in pairs, as the items can now be ranked according to the length of their projection on the weight vector $\mathbf{w}$ [8]. The length of the projection of $\mathbf{x}$ onto $\mathbf{w}$ is defined as

$$\|proj_{\mathbf{w}}\mathbf{x}\| = \|\mathbf{x}\| \cos(\theta_{xw}), \tag{6}$$

where θ_{xw} is the angle between $\mathbf{x}$ and $\mathbf{w}$. Remark that the definition of the dot product of $\mathbf{x}$ and $\mathbf{w}$ is

$$\mathbf{x}'\mathbf{w} = \|\mathbf{x}\| \, \|\mathbf{w}\| \cos(\theta_{xw}). \tag{7}$$

Since $\|\mathbf{w}\|$ is a constant and has the same value for all $\mathbf{x}$, ranking by the dot product is equivalent to ranking by the length of the projection. It is therefore sufficient to rank the items by the value of $\mathbf{x}'\mathbf{w}$.

The SVM requires kernel parameter tuning in order to obtain optimal results. A possible tuning method is a brute force approach in the form of an exhaustive grid search. However, since this method is computationally expensive, a more efficient method is necessary when the used data set is large. A more efficient method is the Bayesian Optimisation approach as proposed by Brochu et al. [18]. They introduce Bayesian models for predicting a target function $f(x)$. By starting off with evaluation in a random point and using the gathered information to predict the next point that should be evaluated, a probabilistic model for $f(x)$ is constructed. With each evaluation this model is updated and exploited in order to choose the best candidate point for the next evaluation. In such a way, the method is able to efficiently determine the optima for any continuous target function.

3.4 Personalisation of Rankings

For the personalisation of a ranking, it is important that the match between a query and an item is included in the criteria on which this ranking is based. In particular, it is important that the ranking function incorporates user characteristics. However, user characteristics do not differ across the items associated with a search query. It is therefore not possible to differentiate items based on user characteristics.

In order to solve the above problem, we propose a novel method for including characteristics in the ranking of items. By multiplying numeric user characteristics with item-specific variables, we combine the two effects. Depending on the value of the user characteristic, we amplify the effect of a property, or we reduce it. We call these kinds of feature weighting measures 'cross-terms'. It is possible to include all available combinations of variables and user characteristics, as collinearity is not an issue for an SVM [19]. One type of cross-term can simply be the variable itself, without multiplication factor.

We illustrate the idea of cross-terms by giving an example from the airline industry. Consider the cross-term for ticket price

$$\texttt{Price} \times \texttt{Number of children},$$

where `Number of children` is specified in the search query. This cross-term captures the possible effect that the number of children has on the price preference. When the user specifies no children, this variable has no effect on the ranking, as the values for all items in the query results are equal to zero. Note that we do not yet imply the sign of the effect, as the SVM can still give either a negative or a positive weight to this variable. However, when using this cross-term one does assume that the effect increases linearly with the number of children.

If both the number of variables and the number of user characteristics is large, the number of cross-terms becomes also large, as this is the multiplication of the two. It is necessary to perform feature selection for Ranking SVM, especially when using many cross-terms. Including too many features might cause overfitting of the SVM for the training sample. We perform feature selection such that we are left with a prespecified number of features r.

3.5 Feature Selection

In order to find the subset of r among d cross-terms ($r < d$) that optimise the performance of the ranking function, we use the SVM-RFE approach as proposed by [20]. This method utilises the properties of the weight vector $\mathbf{w}$. When each of the variables is scaled onto a continuous interval of $[0, 1]$, the magnitude of w_i gives an indication for the importance of cross-term i [21]. Therefore, we can perform backward feature selection on the weights. We iteratively remove the cross-term corresponding to the minimal value of w_i^2 and calculate a new weight vector with the remaining cross-terms, until a subset of size r remains.

3.6 Performance Measurements

Accuracy is used as the main evaluation measurement in this study. We calculate this by introducing a Success@N% performance measure. This measure considers a prediction to be correct when the selected item is ranked in the top N percent of the ranking. The top N percent is ceiled to the next integer. Furthermore, we test the model performance by comparing the results of PARUX with a benchmark. As a baseline model we use a naive ranking rule, which simply orders the items based on one single feature. The feature that correlates the most with the target ranking is chosen as the feature for the naive ranking rule.

3.7 Implementation

We use a number of different programming languages for this study. Due to the large size of the used data set, we perform pre-processing steps in Java, as Java is relatively fast in processing compressed data. We use an implementation of the Ranking SVM algorithm in Python, called SVM^{rank}. Before that, the data is shaped in R into the format that is required for the SVM^{rank} algorithm. R is also used to perform parameter tuning for SVM^{rank} and feature selection according to the SVM-RFE algorithm.

In Python we make use of the SVM^{rank} software provided by Joachims [8]. The `svm_rank_learn` command performs the Ranking SVM algorithm and gives importance weights to all included features. The binary `svm_rank_classify` leverages these weights and assigns a class label to items in a test set. In R we make use of the `rBayesianOptimization` package to find the optimal SVM parameters. Apart from these operations, we only use standard libraries in all of the programming languages.

4 Evaluation

In this section we delineate a detailed set-up of our experiment and evaluate the results obtained by PARUX. To train and test our proposed framework, we use a data set which comprises clickstream data, collected on the website of a European airline during the first quarter of 2016. For the sake of comparability among the click logs, we only consider users that have made a selection on both an outward and a return flight, i.e., retour flight selections only. As a final sweep through this sample, we select the 9.59% of users that have actually purchased the flight ticket corresponding to their selection, that is 16,239 queries. By taking only the converted flights we aim to minimise the bias caused by clicks that were made to just explore the site, and therefore might not contain actual information on customer preferences. To obtain a set of training data, the remaining sample is randomly permuted and 2/3 of the click logs are drawn without replacement, which gives $n = 10{,}826$ queries resulting in a conversion. Consequently, the test data is made up of 5,413 queries.

We merely focus on ranking the presented outward flights, since in the used data set the logged return price fully depends on the selected outward flight. Considered are four main features of outward flights, on which data is contained in the related search query on individual level. These features are price, journey duration, number of stops, and departure time. Since the latter has non-numerical values (time format), we transform the single variable into four disjunct categorical dummies: departure in the night (00:00–5:59 AM), morning (6:00–11:59 AM), afternoon (00:00–5:59 PM), and evening (6:00–11:59 PM). With one of the four features being transformed into respectively four different dummies, we are left with a total of seven flight-specific features.

Only queries for which 4 or more flights were presented to the customer are considered. We argue that in smaller sets of flights ranking is not relevant. For every query we select the flight which is chosen by the customer: the ‘winning’ flight. Subsequently, we construct 3 item pairs containing both the ‘winner’ and one randomly picked ‘loser’. We choose to take 3 ‘losers’ as we wish to include the same number of item pairs for each query. There is large variation in the prices and journey durations of flights to different destinations in the data, as both short distance trips and intercontinental flights occur. To make the queries for different destinations comparable, we transform price, flight duration, and the number of stops to Z-scores within the search results of a query. That is, we consider these variables relative to the values of the corresponding variables of other flights that were shown in the search results. We then create the input file for the SVM^{rank} algorithm with these new standardised variables. In this way, we are able train one model for search queries to all destinations in the data. After this standardisation, each of the features is then scaled onto the continuous interval $[0, 1]$ to allow for an unambiguous interpretation of the weight values that result from SVM^{rank}. Scaling is performed according to Min-Max normalisation over all observations:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{8}$$

As proposed in Sect. 3.4, we include cross-terms in our analysis to personalise the ranking. For the customer-specific features we incorporate variables such as the specified number of adults, selected travel class, whether the user is logged in on the website, and if he or she has visited a promotional Web page. A full overview of all features can be found in Table 1. `Economy` is a dummy which is equal to 1 if the specified travel class is economy, and 0 if business. The three variables containing the word "page" take on a value of 1 if a particular page on the website has been visited before the search. The `Apple` dummy indicates whether the flight is booked from an Apple device. `Europe` is equal to 1 if the selected flight origin and destination are both located within Europe. We include the variable `Europe` because the Europe-based airline considered in this study is interested in potential differences between continental and intercontinental flights. This gives rise to a total of 77 variables that can be included in the SVM^{rank} algorithm: the seven untouched flight-specific features, as well as every flight-specific feature multiplied by each of the ten customer-specific features (7×10) as presented in Table 1.

We decide to include a subset containing the top 25% most important cross-terms, which results in $r = 20$ variables. To identify these cross-terms, we perform a feature selection as described in Sect. 3.5. Later we will present a complete overview of the 20 included cross-terms. These variables are used to obtain the optimal parameters for SVM^{rank}.

As [22] states that $C_{\text{SVM}} = C_{\text{SVM}^{rank}}/n$, with n the number of queries in the training set, we can use a classical SVM to tune the kernel parameters. To determine which kernel function should be used, we compare them by running an SVM with different kernels on the same validation set. The validation set is randomly sampled from the total training set by means of 10-fold cross-validation. In each iteration we assess the accuracy of the four different kernel functions: linear, radial, polynomial, and sigmoid. For the kernel parameters we use the default values of the `e1071` package in R. After 1500 iterations the resulting accuracy levels per kernel are compared in a t-test, for which the results can be found in Table 2. It follows that a linear kernel is independently preferred over any of the three other kernels for a significance level of 0.01. Since the

Table 1. Overview of all used flight-specific and customer-specific features

Flight-specific features	Customer-specific features
Price	Number of adults
Number of stops	Number of children
Journey duration	Economy
Departure in the night	Log in
Departure in the morning	Business page
Departure in the afternoon	Promotional page
Departure in the evening	Booking orientation page
	Number of clicks
	Apple
	Europe

Table 2. p-values related to SVM kernel selection, indicating a preference for kernel 1 over kernel 2

Kernel 1	Kernel 2	t-value	p-value
Linear	Radial	3.886	0.000
Linear	Polynomial	1354.783	0.000
Linear	Sigmoid	2.682	0.007

non-linear kernels have the additional disadvantage of losing interpretation on the generated weights, we choose to apply SVM^{rank} with a linear kernel.

To tune the cost parameter C for the linear kernel, we perform Bayesian optimisation on a validation set, which is constructed by applying 10-fold cross-validation to the training sample. This results in an optimal cost parameter value of $C = 0.0316$ for a classical SVM with linear kernel. Given the relation stated in the previous paragraph, we obtain an optimal cost parameter value for SVM^{rank} of $C = 0.0316 \times 10,826 = 342.10$. With this optimal value the model gives the cross-term weights as presented in Table 3.

Regarding the absolute values of the weights, `Price` has the strongest effect on the determined ranking, with a weight value of -2.945. As an illustration, the ranking score of a flight will decrease if the price increases, assuming that all other variables are held constant. Similarly, given the negative weight for variable `Journey duration` of -2.700, a decrease in duration will improve the relative ranking position of the flight ceteris paribus. This yields that a trade-off between features can be constructed.

It should be noted that it is impossible to infer a ranking from direct feature values, as for the SVM^{rank} input each feature is first converted into a Z-score on query level, and then scaled over the entire feature range. The weights are

Table 3. SVM^{rank} output weights corresponding to the 20 most important cross-terms based on training data

Cross-term	Value
Journey duration	−2.700
Journey duration × Europe	0.607
Price × Economy	−2.644
Price	−2.945
Journey duration × Economy	−2.227
Number of stops	−1.879
Number of stops × Economy	−1.607
Price × Europe	−0.868
Number of stops × Booking orientation page	−0.217
Number of stops × Number of clicks	−0.262
Departure in the evening × Number of adults	−0.145
Departure in the morning × Number of adults	0.458
Departure in the evening × Europe	−0.533
Departure in the afternoon × Number of clicks	−0.093
Price × Business page	0.032
Number of stops × Log in	−0.341
Price × Number of clicks	−0.544
Journey duration × Booking orientation page	−0.206
Departure in the morning × Number of children	−0.032
Journey duration × Apple	−0.102

therefore only applicable to standardised units, clouding the explicit interpretation. Nonetheless, if the constructed cross-term values are standardised and scaled, an expected ranking can be constructed in the form of a linear combination by multiplying weights with cross-term values.

Apart from the feature `Price`, Table 1 also contains the `Price` × `Economy` cross-term. Since `Economy` is equal to 1 only if the customer has selected the economy travel class, the effect of this cross-term is only existent for economy flights. The negative weight of −2.644 implies that economy travellers are more sensitive to changes in price than business travellers, given that all other variables remain constant. However, there are more additional effects for economy travellers. Note the presence of variables `Journey duration` × `Economy` and `Number of stops` × `Economy`, for example. The main consequence is that the trade-off between variables becomes inherently different between economy and business travellers.

The weight for `Journey duration` × `Europe` has a positive sign, indicating that the duration of a flight is of less influence for customers travelling within Europe. The other cross-term weights can be interpreted in a similar way.

Table 4. Outcomes and performance measures of the SVM^{rank} classification step. When applying the model to the training data, only 3 'loser' flights are included per query. For the test set, all presented flights are included. Average loss is defined as the fraction of discordant pairs averaged over all rankings.

	Train data	Test data
Number of queries	10,826	5,413
Correct	8,240	3,099
Incorrect	2,586	2,314
Number of pairs	32,478	72,682
Number of discordant pairs	3,968	7,630
Success at 15%	76.11%	75.54%
Average loss	12.22%	10.50%

Given the model output as represented by the weights, classification of the test data by SVM^{rank} can be assessed by means of performance measures as shown in Table 4. We use Success@k% instead of Success@k, since the number of flights shown to a customer highly differs per search query. In specific, we choose the Success@15% performance measure. In this way, we require the chosen flight to be on top of the predicted ranking to count as a success, if the number of flights is 6 or less. However, when the number of shown flights is high, e.g. 30, we only require that the chosen flight is among the top 5 items of the model ranking. We consider this a reasonable measure, as we argue that website visitors do not investigate the entire list of search results, both for short and long lists of results.

Classification on the test data gives a total of 7,630 swapped pairs, resulting in an average loss value of 10.50%. Phrased differently, we are able to predict the winner correctly for 89.50% of all possible winner-loser pairs. This results in a Success@15% of 75.54%. We predict the selected flight to be at the top of the ranking for 3,099 queries, out of 5,413 on the test set.

If we run SVM^{rank} with the 7 flight-specific features only, the selected flight was predicted to be in the top 15% of all presented flights 73.43% of the time on the test data. Hence, using the top 20 cross-terms gives an increase in Success@15% of 2.11 percentage points. Although these increments appear to be relatively small, when viewed by a vast number of customers, a slightly more personalised ranking potentially increases click-through and conversion rates.

Additionally, we generate a naive ranking to evaluate the performance of PARUX. We choose the variable `Price` as the naive ranker, since this is the flight-specific feature that correlates the most with the target ranking. The naive model is evaluated on the same test set used for the SVM^{rank} algorithm. The Success@15% on this test set is 62.41%, which indicates that SVM^{rank} with 20 cross-terms outperforms the naive ranking model by approximately 12% points. This shows that additional information on customer preferences is captured by the cross-terms, and that SVM^{rank} is able to identify the trade-offs customers make up to a certain extent.

5 Conclusion

In this work, we have explained the necessity of presenting customers with the most interesting products when shopping online. We proposed a learning-to-rank framework for personalising the product ranking in an e-shop. Using Ranking SVM, we were able to beat a naive ranking rule by 10% points in success rate. We introduced PARUX, a model that leverages features that describe the match between item-specific variables and user characteristics. Compared to the Ranking SVM model with only flight-specific features, PARUX improves the Success@15% with 2.11% points. Due to the high amount of traffic on the Web, improvements like this can already have a big impact on users' purchase activity.

However, the actual impact of our framework should be investigated in an A/B-testing experiment. Note that the feature weights should be updated regularly, as customer preferences can change over time. As soon as the model is implemented in an actual item search engine, it would be interesting to train the model on the search behaviour in this optimised setting. We would like to see if the model performance improves if it is trained on this new data, as this data might contain a better representation of the user preferences, due to the fact that the top listed items are the most competitive ones.

As a suggestion for future work, we would like to extend PARUX such that it can incorporate past purchase behaviour. In this way, the framework is further personalised on the individual level. Another way to extend the model for the same purpose, would be to segment the users and train a model for each of the different segments. Additionally, it would be interesting to see if the use of cross-terms will prevail in areas other than the airline industry.

References

1. Van Rijmenam, M.: Big data trends for 2016: what you need to know, December 2015. https://execed.economist.com/career-advice/industry-trends/big-data-trends-2016-what-you-need-know. Posted 1 December 2015
2. Kobayashi, M., Takeda, K.: Information retrieval on the web. ACM Comput. Surv. (CSUR) **32**(2), 144–173 (2000)
3. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: Proceedings of the 2nd ACM Conference on Electronic Commerce, pp. 158–167. ACM (2000)
4. Vargas, S., Castells, P.: Rank and relevance in novelty and diversity metrics for recommender systems. In: Proceedings of the Fifth ACM Conference on Recommender Systems, pp. 109–116. ACM (2011)

5. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. (TOIS) **22**(1), 5–53 (2004)
6. Bernardi, L., Kamps, J., Kiseleva, J., Mueller, M.J.: The continuous cold start problem in e-commerce recommender systems. Eindhoven University of Technology (2015)
7. Jacobs, B.J., Donkers, B., Fok, D.: Model-based purchase predictions for large assortments. Mark. Sci. **35**(3), 389–404 (2016)
8. Joachims, T.: Optimizing search engines using clickthrough data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 133–142. ACM (2002)
9. Liu, T.Y.: Learning to Rank for Information Retrieval. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-14267-3
10. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: Proceedings of the 22nd International Conference on Machine learning, pp. 89–96. ACM (2005)
11. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: Proceedings of the 24th International Conference on Machine learning, pp. 129–136. ACM (2007)
12. Coyle, L., Cunningham, P., Hayes, C.: A case-based personal travel assistant for elaborating user requirements and assessing offers. In: Craw, S., Preece, A. (eds.) ECCBR 2002. LNCS (LNAI), vol. 2416, pp. 505–518. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46119-1_37
13. Ostuni, V.C., Di Noia, T., Di Sciascio, E., Mirizzi, R.: Top-n recommendations from implicit feedback leveraging linked open data. In: Proceedings of the 7th ACM Conference on Recommender Systems, pp. 85–92. ACM (2013)
14. Freund, Y., Iyer, R., Schapire, R.E., Singer, Y.: An efficient boosting algorithm for combining preferences. J. Mach. Learn. Res. **4**, 933–969 (2003)
15. Hofmann, K., Whiteson, S., de Rijke, M.: Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. Inf. Retrieval **16**(1), 63–90 (2013)
16. Burges, C.J.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Disc. **2**, 121–167 (1998)
17. Clegg, A.: Learning to rank: where search meets machine learning, Berlin Buzzwords, July 2016. https://berlinbuzzwords.de/16/session/learning-rank-where-search-meets-machine-learning
18. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
19. Dormann, C.F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carré, G., Marquéz, J.R.G., Gruber, B., Lafourcade, B., Leitão, P.J., et al.: Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. Ecography **36**(1), 27–46 (2013)
20. Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene selection for cancer classification using support vector machines. Mach. Learn. **46**(1–3), 389–422 (2002)
21. Rakotomamonjy, A.: Variable selection using SVM-based criteria. J. Mach. Learn. Res. **3**, 1357–1370 (2003)
22. Joachims, T.: SVM-rank: Support vector machine for ranking. Department of Computer Science, Cornell University (2009). https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html