Determination of Attribute Weights for Recommender Systems Based on Product Popularity

Article · July 2009

Source: RePEc

CITATIONS
2

READS
421

3 authors, including:

Martijn Kagie

14 PUBLICATIONS **133** CITATIONS

SEE PROFILE

Patrick J F Groenen

Erasmus University Rotterdam

290 PUBLICATIONS **15,049** CITATIONS

SEE PROFILE

Martijn Kagie *, Michiel van Wezel, Patrick J.F. Groenen

Econometric Institute, Erasmus University Rotterdam

Determination of Attribute Weights for Recommender Systems Based on Product Popularity

Abstract

In content- and knowledge-based recommender systems often a measure of (dis)similarity between products is used. Frequently, this measure is based on the attributes of the products. However, which attributes are important for the users of the system remains an important question to answer. In this paper, we present two approaches to determine attribute weights in a dissimilarity measure based on product popularity. We count how many times products are sold and based on this, we create two models to determine attribute weights: a Poisson regression model and a novel boosting model minimizing Poisson deviance. We evaluate these two models in two ways, namely using a clickstream analysis on four different product catalogs and a user experiment. The clickstream analysis shows that for each product catalog the standard equal weights model is outperformed by at least one of the weighting models. The user experiment shows that users seem to have a different notion of product similarity in an experimental context.

Key words: Recommender Systems, Attribute Weights, Poisson Regression, Boosting, Dissimilarity Measures, Evaluation.

1 Introduction

Over the past fifteen years, a very large number of approaches has been proposed which can be used to recommend products from a product catalog to users. These so-called recommender systems [29] can be subdivided in three groups [1]: Collaborative, content-based, and hybrid recommenders. Where collaborative filtering, the most popular method, recommends products based

* Corresponding author. Phone: +31 10 4088943. Fax: +31 10 4089031.
Email addresses: kagie@ese.eur.nl (Martijn Kagie), mvanwezel@acm.org (Michiel van Wezel), groenen@ese.eur.nl (Patrick J.F. Groenen).

on similarity of the user's taste with the taste of other users, content-based recommenders use characteristics of products to base their recommendations on. Hybrid recommenders combine both approaches.

Although collaborative recommendation methods seem to be the most popular type in practice and in the literature, content-based methods are far more useful in certain areas of electronic commerce, such as consumer electronics, real estate, and tourism. In these areas, products can usually be described by a well-defined set of attributes and it would be wasteful not to use these attributes when recommending. Moreover, often only very limited choice- or preference data are available, since customers buy these kinds of products infrequently. This makes it impossible to discover and exploit correlation structure in user preferences, as is typically done by collaborative recommenders. Furthermore, using attributes alleviates the cold start item problem: since attribute values are known from the start, new products can immediately be used in the recommendation cycle, whereas collaborative methods have to wait until sufficient co-preference data has been gathered. Finally, attribute-based recommendations open the door for explaining to the user why certain recommendations are given, thus making the recommendation process more transparent. Various sources (see e.g., [16,36,38]) suggest that users favour transparent recommendations over nontransparent ones.

Many content-based recommender systems use some type of case-based reasoning or nearest neighbor retrieval [24,26,28]. These techniques rely heavily on some dissimilarity measure between different products for their recommendation strategy. Usually, this dissimilarity measure is based on the attributes of the products. However, not all attributes of an product are equally important to the user and, thus, this asymmetry should be reflected in the dissimilarity measure. Therefore, to avoid a mismatch of the system and the perceived similarity, the dissimilarity measure should use a suitable attribute weighting, thus avoiding wrong recommendations by the system.

Although weights are generally specified by experts, some work has been done on recommender systems that automatically learn these weights on a per-user basis, such as systems based on MAUT-based preference models [19]. For example, Schwab et al. [35] learn user specific weights for binary features using significance testing assuming normal distributions. When the user selects items having a specific attribute value more often, that is, there is a significant effect, this attribute got a higher weight. Arslan et al. [2] use the number of times an attribute was used in the query of the user to learn these attribute weights. Branting [3] uses an algorithm that changes weights based on the set of items recommended and the selection of the user of one of these items. Finally, Coyle and Cunningham [9] compare the final choice of the user with the provided recommendations and learn the attribute weights from that.

All these approaches assume that the user gives the system time to let it learn his/her preferences in one or more sessions. However, in many e-commerce domains, such as durable goods, this is not a realistic assumption, due to the infrequent purchases and subsequent data sparsity mentioned above. Although it might be possible to adapt the weighting approaches above to a nonpersonalized setting so that the use of pooled data solves the data sparsity problem, these approaches remain incapable of handling mixed data (that is, consisting of both numerical and categorical values) and missing values, both of which are usually abundant in electronic commerce product catalogs.

In summary, attribute-based recommendation is relevant because it is useful in e-commerce domains with sparse preference data and rich attribute data, and it allows for more transparency in the recommendation process. It generally employs an attribute-based dissimilarity measure. Since it is likely that some attributes are more relevant than others in users' perceptions of product similarity, the dissimilarity measure used by the recommender should match the perceived attribute importances. Thus, the quest is to find (1) a way for determining attribute weights that match similarity perceived by users from the available catalog and choice data with their many missing values, their mixed attributes and their sparse observations, and (2) a way to evaluate a given set of attribute weights.

In this paper, we introduce two methods to determine attribute weights for dissimilarity in recommender systems. These methods only rely on some measure of product popularity as 'target' attribute, while the 'input' attributes are the product characteristics. We show how these weighting methods can be used with mixed data and missing values. We then evaluate the weights that are produced by these methods in two novel ways. The first evaluation method is based on clickstream analysis and the second one is based on a survey.

The first weighting method we discuss is based on Poisson regression [25,27] and was introduced in [21]. It uses multiple imputation [32] to handle missing values and dummy variables to handle categorical attributes. The weights are determined using t-values of the regression coefficients. The second weighting method is based on boosting [13] and has some advantages over the Poisson regression method, since it handles missing values and categorical attributes in a more natural way. Also, there is a straightforward method to determine attribute importance for boosting [13] and it is more flexible.

In the first evaluation, we use clickstream logs of four real life electronic commerce product catalogs, to see which products are often visited together in a session. These co-visits give rise to empirical similarities between product pairs: we assume that products frequently visited together are considered to be similar by users. Therefore, a dissimilarity measure based on attributes should also identify these as similar and thus a vector with attribute weights can be

evaluated by considering how well the attribute-based similarities match the observed empirical similarities. To measure this 'match', we introduce a measure called the mean average relative co-occurrence (MARCO).

The second evaluation method is based on a user experiment consisting of an online survey. In this survey, respondents were given a reference product. Then, they were asked to indicate which products they considered to be relevant recommendations given the reference product. The weighted dissimilarity measure should be able to recommend these relevant products. To measure the amount of overlap between the recommendations by the dissimilarity measure and the relevant recommendations, we propose the mean average relative relevance (MARR).

We use the MARCO and MARR measures to evaluate the weights yielded by both the Poisson regression- and the Boosting based weighting methods. The performance of both methods is benchmarked against the naive (but often used) method of weighting each attribute equally.

The remainder of the paper is organized as follows. In the next section, we introduce the notion of dissimilarity in recommender systems and the dissimilarity measure we use in this paper. Then, in Section 3, we introduce the two weighting approaches based on product popularity. In Section 4, these two approaches are applied to four real life electronic commerce product catalogs. Sections 5 and 6 discuss both evaluation approaches based on clickstream logs and a survey. Finally, we draw conclusions in Section 7.

2 The Use of Dissimilarities in Recommender Systems

A large group of content-based recommender systems, the so-called case-based recommender systems [28] or case-based reasoning recommender systems [24] rely on a dissimilarity measure based on the attributes of the products to provide recommendations. In general, we can define such a measure as

$$\delta_{ij} = f\left(\sum_{k=1}^{K} w_k \delta_{ijk}\right) \quad , \tag{1}$$

where δ_{ij} is the dissimilarity between product i and j, w_k is the attribute weight for attribute k, δ_{ijk} is a dissimilarity score measuring the difference between product i and j on attribute k, and $f(\cdot)$ is a monotone increasing function. Note that often $f(\cdot)$ is the identity function.

In case-based recommender systems, the use of a dissimilarity measure has the advantage that it can be used for recommendation in two different situations.

First, when a reference product is available (for example, the product the user is looking at or has selected in some way), recommendation is based on the dissimilarity between this product and other products. A second way to use a dissimilarity for recommendation is when the user specifies a search query in the form of an (incomplete) ideal product specification, we can provide recommendations. These recommendations are based on the dissimilarity between the ideal product specification and the products in the product catalog.

Although our approach to determine attribute weights can be used with every dissimilarity measure that can handle linear weights, it is vital to apply a good dissimilarity measure, that is, one that is close to the user's notion of dissimilarity. In the case of electronic commerce product catalogs, a dissimilarity measure should be able to handle missing values and attributes of mixed type. Often used (dis)similarity measures, like the Euclidean and Hamming distance, Pearson's correlation coefficient, and Jaccard's similarity measure, lack these abilities. Many dissimilarity measures in the domain of knowledge-based recommender systems need domain knowledge [8,9,10,26]. We would like to avoid this, such that the approach is more flexible.

To our knowledge, only two dissimilarity measures previously used in electronic commerce applications can handle both mixed attribute types and missing values and do not need any domain knowledge. The first are measures based on the heterogeneous Euclidean overlap metric (HEOM) [39] as used in [2,30,33]. The second is a modified version of the Gower's general coefficient of similarity [14], which has been used in [20,21,22].

HEOM and the adapted Gower coefficient both compute dissimilarity scores for all attributes separately and finally combine these. The computation of these scores differ between attribute types. When HEOM is not able to compare two attribute values, because one or both of them is missing, it will treat them as dissimilar from each other. The adapted Gower coefficient ignores the attribute and computes the dissimilarity based on the nonmissing dissimilarity scores, which is, in our opinion, a better approach. Also, the adapted Gower coefficient has the advantage that the dissimilarity scores are normalized such that each attribute is equally important in the dissimilarity measure. We will use this dissimilarity measure in the remainder of this paper. For more details on the definition of the adapted Gower coefficient, we refer to Appendix A.

3 Determining Attribute Weights

In this section, we will introduce two methods to determine attribute weights based on product popularity, such as sales or pageviews. The first method is based on Poisson regression in which we use multiple imputation to handle

missing values. Weights are based on the t-values of the Poisson regression coefficients. We consider a full Poisson regression model and a stepwise model that has a built-in method for attribute selection.

The second method is based on a new boosting algorithm, PoissonTreeBoost, that optimizes Poisson deviance. The relative importance measure is used to determine weights. This method is more flexible than Poisson regression. Also, it has built-in methods to handle categorical attributes and missing values.

In these two methods, the popularity counts for the I products are used as dependent/target attribute vector $\mathbf{y} = (y_1, y_2 \ldots, y_I)$. This target attribute is due to its nature a count variable being discrete and nonnegative and, therefore, different models than ordinary least squares regression models should be used. As independent variables we use, naturally, the attribute vectors $\{\mathbf{x}_i\}_1^I$, that is, $\mathbf{x}_i = (x_{i1}, x_{i2} \ldots, x_{iK})$, with K the number of attributes. The counts in $\mathbf{y}$ and the attributes will be used to determine weights $\{w_k\}_1^K$ for the product attributes in some dissimilarity measure.

3.1 Poisson Loglikelihood and Deviance

Models for count data, such as Poisson regression [25,27] which we will use in Section 3.2, maximize the loglikelihood of the Poisson distribution

$$\log \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^{I} [-\hat{y}_i + y_i \log \hat{y}_i - \log y_i!] \ , \tag{2}$$

where $\hat{y}_i$ are the model predictions and $y_i!$ is the factorial of y_i. An alternative approach, which is also used by the Poisson regression tree [37] and the boosting method we discuss in Section 3.3, is minimizing the Poisson deviance, which is defined as

$$\begin{aligned} D(\mathbf{y}, \hat{\mathbf{y}}) &= -2(\log \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) - \log \mathcal{L}(\mathbf{y}, \mathbf{y})) \\ &= 2 \left(\sum_{i=1}^{I} \left[y_i \log \left(\frac{y_i}{\hat{y}_i} \right) - (y_i - \hat{y}_i) \right] \right) \\ &= 2 \sum_{i=1}^{I} L^{PD}(y_i, \hat{y}_i) \ , \end{aligned} \tag{3}$$

which equals minus two times the loglikelihood of the model minus the loglikelihood of a model perfectly fitting the data. In this equation, L^{PD} denotes the Poisson deviance loss function. Note that $D(\mathbf{y}, \hat{\mathbf{y}})$ has the advantage that it is a minimization problem and a deviance of zero represents a perfect model (on the training data).

3.2 Poisson Regression

In [21], we introduced one way in which attributes can be determined based on product popularity, namely using a Poisson regression model, which is a member of the generalized linear model (GLM) framework [25,27]. To be able to estimate a Poisson regression model on our data, we set up the matrix $\mathbf{X}$ of predictor variables as follows. Categorical attributes are replaced by series of dummy variables as is common in the literature on GLMs. Every categorical attribute is represented by L_k dummies $x_{ik\ell}$, which are 1 for the category where the product belongs to and 0 for all other attributes. To avoid multicollinearity, the last category is not represented by a dummy, so that L_k is the number of different categories for attribute k minus one. For multi-valued categorical attributes the same approach is used, only now all categories are represented by the L_k dummies. For numerical attributes we use the original value. Hence, $x_{ik} = x_{ik1}$ and $L_k = 1$. We collect all $x_{ik\ell}$ for product i in vector $\mathbf{x}_i$. Also, an intercept term x_{i0} is incorporated in this vector, which equals 1 for all products, so that $\mathbf{X}$ has I rows and $(1+\sum_{k=1}^{K} L_k)$ columns. The Poisson regression model is defined as

$$y_i \approx \exp(\mathbf{x}_i'\mathbf{b}) \ , \tag{4}$$

where y_i is a dependent count variable (in our case product popularity or sales) and $\mathbf{b}$ is a $(1 + \sum_{k=1}^{K} L_k)$ by 1 vector of regression coefficients. Additionally, y_i is assumed to have a Poisson distribution having expectation $E(\exp(\mathbf{x}_i'\mathbf{b}))$. The regression coefficients can be estimated in this model by adapting the loglikelihood equation (2) to

$$\log \mathcal{L}(\mathbf{b}) = \sum_{i=1}^{I} [-\exp(\mathbf{x}_i'\mathbf{b}) + y_i \mathbf{x}_i'\mathbf{b} - \log y_i!] \ , \tag{5}$$

which is often maximized using an iteratively reweighted least squares procedure. This algorithm produces both the model parameters $b_{k\ell}$ and their standard errors σ_{kl}.

A disadvantage of Poisson regression models (and GLMs in general) is that they lack a built-in way to handle with missing values. In a recent paper, Ibrahim et al. [18] compared different methods to handle missing values in combination with GLMs and found that multiple imputation (MI) [32] was among the best methods to be used in this situation. MI methods create a number of 'complete' data sets in which values for originally missing values are drawn from a distribution conditionally on the nonmissing values. There are two methods to create these imputed data sets: Data augmentation [34] and sampling-importance-resampling [23]. Both lead to results of identical quality,

while the latter does this much faster. Therefore, we use that algorithm, which is available in the Amelia II package [17] for the statistical software environment R, in our approach. For a more detailed discussion on how regression coefficients and standard errors are computed using MI, we refer to [21].

The weights to be used in the dissimilarity measure are based on the regression coefficients $b_{k\ell}$. However, we cannot use these coefficients directly as weights for several reasons. First, all attributes are on different scales and this also holds for the coefficients. Second, any uncertainty about the coefficient is not taken into account. Although a coefficient value can be reasonably large, this does not necessarily mean we are also certain about the value of this coefficient. Finally, coefficients can also be negative, while weights should always be positive. The first two problems can be overcome by using the t-value of $b_{ik\ell}$ which is defined as

$$t_{k\ell} = \frac{b_{k\ell}}{\sigma_{k\ell}} \ , \tag{6}$$

while the second can be solved by using the absolute value $|t_{k\ell}|$.

However, since we had to use dummy variables for (multi-valued) categorical attributes, the weight for these attributes is based on the average over the corresponding t-values. Hence, we can define a 'pseudo' t-value v_k as

$$v_k = \frac{1}{L_k} \sum_{\ell=1}^{L_k} |t_{k\ell}| \ . \tag{7}$$

For numerical attributes v_k just equals the corresponding absolute t-value. Finally, we normalize the weights to have a sum of 1

$$w_k = \frac{v_k}{\sum_{k'=1}^{K} v_{k'}} \ . \tag{8}$$

Note that, instead of using t-values other approaches could be taken to determine weights based on a Poisson regression model. For example, one could first normalize all attributes and then use the absolute values of $\mathbf{b}$ to determine the weights. Preliminary experimentation did not show improvements over using (7). Therefore, we do not pursue this approach any further.

Besides a Poisson regression model using all attributes, also a stepwise Poisson regression model was discussed in [21] to determine attribute weights. Stepwise models use the statistical significance of the attributes to do attribute selection. Each time the most insignificant attribute (based on the t-values) is deleted from the model specification and a new model is estimated until

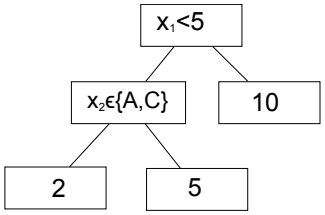


Figure 1. Example of a Regression Tree: The root divides the data in two parts using numerical attribute x_1. Instances having a x_1 value smaller than 5 turn left, the other instances right. The instances that have turned right get a y-value of 10, which is the sample mean of training instances belonging to this node. The instances that have turned left are again split using categorical attribute x_2. When x_2 of an instance belongs to category A or C, this instance turns left.

only significant attributes remain. In our evaluation, we will consider both the stepwise and the complete Poisson regression approach.

3.3 Boosting Poisson Deviance

Although the procedure based on Poisson regression is simple to use, it has some drawbacks. Poisson regression models have no integrated way to handle missing values (while product catalogs, in practice, contain a lot of missing values), due to which a computational intensive imputation techniques (see [32]) need to be applied. Furthermore, (multi-valued) categorical attributes need to be preprocessed prior to be included into the model. Finally, Poisson regression models cannot handle interaction effects (without dramatically increasing the attribute space or using some kernel method) and, therefore, may possibly miss attributes that are important, but only in combination with other attributes.

Therefore, we introduce a model that has integrated ways to handle missing values, categorical attributes, and interactions and is also able to handle a count variable as target attribute. This method is based on boosting [11,12], a method to combine a series of base models, which are often decision trees [7]. First, we briefly introduce these decision trees.

Decision trees are a popular method for both classification and regression, where trees applied to the latter are often called regression trees. Regression trees minimize a squared loss function

$$B = \arg\min_{B} \sum_{i=1}^{I} L^S(y_i, B(\mathbf{x}_i)) = \arg\min_{B} \sum_{i=1}^{I} (y_i - B(\mathbf{x}_i))^2 \ , \tag{9}$$

where $B(\mathbf{x})$ denotes the prediction of tree B for input vector $\mathbf{x}$ and L^S is the squared loss function. This function is minimized in such a way that this results

in a tree with non-terminal nodes containing split criterions and terminal nodes with predicted y-values. An example of such a tree is shown in Figure 1. Regression trees have the ability to deal with categorical attributes as is shown in Figure 1. In addition, regression trees are able to automatically handle missing values. Different methods exist to handle missing values in decision trees, see for example [15,31]. A convenient way to handle them is using surrogate splits [15], introduced by Breiman et al.[7]. We use this method to handle missing values in our application.

A drawback of decision trees is their instability. The implemented model depends heavily on the data set used for model creation, and a small change in the data may have large consequences for the model. Ensemble methods, such as bagging [4], random forests [6], and boosting [11,12], have a stabilizing effect by averaging over a number of decision trees.

Optimizing a squared loss function works well in most regression problems. However, when y is a count variable as in our problem, assuming squared loss is not valid. Therefore, a Poisson regression tree [37] has been introduced that minimizes Poisson deviance (see (3)). However, Poisson regression trees suffer from the same problem as ordinary regression trees: instability. Therefore, we introduce a boosting method minimizing the Poisson deviance.

Boosting is a method to combine multiple models, called base learners, into a single model. All base learners are of the same type and each subsequent model tries to minimize the error made by the previous base learners. Boosting was originally developed for classification problems by Freund and Schapire [11,12]. Friedman [13] developed a framework called GradientTreeBoost which can be used to use boosting in combination with all kinds of loss functions, such as squared, absolute, and Huber loss.

GradientTreeBoost [13] is inspired by gradient based optimization techniques. Similar to these techniques GradientTreeBoost takes steps in the negative gradient direction. Traditional gradient based model fitting does this by refining a parameter vector, and thus indirectly improving the outputs of a model with that parameter vector. In contrast, boosting directly improves the performance of a (composite) model by adding an additional model to it, such that the fit of the composite model on the training data improves.

Often, the base learners are regression trees. These regression trees are ordinary regression trees minimizing a squared loss function, irrespective of the loss function that is minimized by the GradientTreeBoost algorithm. In practice, this means that the base learners are fitted on data sets $\{\mathbf{x}_i, \tilde{y}_i\}_1^I$ in which $\tilde{y}_i$ is a pseudo target replacing the original target y_i. This $\tilde{y}_i$ is determined by the negative gradient of the loss function given the current estimate of y_i (provided by the previous base learners) and the real value of y_i. GradientTreeBoost al-

gorithms have two parameters that need to be set: the number of base learners N and a learning rate parameter ν which indicates how much the algorithm should learn from the base learners. Often, these parameters are determined using cross validation. For a more detailed explanation of GradientTreeBoost, we refer to [13].

To create a boosting model for a count variable, we use a new member of GradientTreeBoost family minimizing the Poisson deviance loss function L^{PD} introduced in (3). The derivation of this method which we call PoissonTree-Boost is shown in Appendix B.

We introduced the PoissonTreeBoost model as a way to determine weights for attributes, which can subsequently be used in a dissimilarity measure. To specify these weights we need a measure that indicates how large the influence of a certain attribute is on the predictions of the model. For this, we use an importance measure for GradientTreeBoost introduced by Friedman [13], which is based on the importance measure developed for CART [7]. Since this importance measure is applicable to all members of the GradientTreeBoost family (all implementing different loss functions), it is also valid in our case, when using the Poisson deviance loss function. It is based on the quantity (how many times is the attribute used to split) and quality (does it split the data in two almost equally sized parts) of the splits based on a certain attribute. For a detailed description of this importance measure, we refer to [13]. For numerical and categorical attributes this procedure works fine. For multi-valued categorical attributes (which can have multiple values for one attribute), a binary attribute is incorporated in the boosting model for each category. The weight is then based on the total importance of all these categories.

4 Determining Weights for Four Real-Life Product Catalogs

In this section, the two approaches to determine attribute weights for our dissimilarity measure are applied to four product catalogs of an electronic commerce website. We first give a short description of the product catalogs. Then, we compare the performance of the PoissonTreeBoost algorithm with two benchmark models to justify our model choice followed by a description of the weights resulting from our approach. In Sections 5 and 6, we evaluate these weights in two ways: The first is based on a clickstream analysis, while the second is based on a user experiment.

Table 1
Description of the product catalogs used in this paper.

Product Catalog	MP3 Players	Digital Cameras	Notebooks	Microwave Ovens
Number of Products (I)	228	283	206	221
Number of Outclicks per Product				
Mean	109	321	72	23
Median	14	78	16	9
Maximum	5652	7871	1417	295
Number of Attributes				
Numerical	16	21	14	14
Categorical	20	12	10	14
Multi-valued categorical	9	7	4	2
Total (K)	45	40	28	30
Missing Values				
Percentage Missing	63%	39%	51%	39%

4.1 Data

Both the product catalogs and the clickstream log files were made available to us by the Dutch internet company 'Compare Group' that hosts, among other European price comparison sites, the Dutch price comparison site `http://www.vergelijk.nl`. The product catalogs used are based on a database dump of this site (October 2007). The log files are used to count how many times users clicked on a link to an internet shop to buy a certain product, which is called an 'outclick'. We counted the outclicks during two months from July 15 until September 15, 2007. These outclicks are used as product popularity in this paper, but other measures for product popularity can be used as long as they are count variables, such as, for example, sales or pageviews. We have used product catalogs containing four types of products: MP3 players, Digital Cameras, Notebooks, and Microwave Ovens. Some characteristics of these catalogs are described in Table 1. As can be seen in this table all these product catalogs have about the same size, which is a typical size for catalogs of these kinds of products. All have a reasonable high number of attributes of which a substantial number are (multi-valued) categorical and a lot of these attributes contain many missing values.

4.2 Model Performance

In this section we compare the model fits of our regression models, both in-sample and out-of-sample. It is worth repeating that the aim of this paper is not to find a well-fitting model but rather to find attribute weights that match user-perceived attribute importances. This requires a different evaluation than considering model fit, and we will turn to that in Section 5 and 6. However,

Table 2
Performance of the different models on training and test data.

	MP3 Players				Digital Cameras			
	Training		Test		Training		Test	
Method	Deviance	MAE	Deviance	MAE	Deviance	MAE	Deviance	MAE
Poisson regression	39.45	44.39	69.20	77.10	116.55	221.57	390.74	416.95
Poisson regression tree	24.02	42.42	83.47	94.68	249.76	299.97	407.71	390.66
PoissonTreeBoost	2.02	14.64	52.16	78.16	33.06	109.47	302.79	328.43
	Notebooks				Microwave Ovens			
Poisson regression	21.57	41.62	35.26	53.18	9.50	16.26	17.67	21.08
Poisson regression tree	50.42	63.46	60.43	68.21	8.16	14.72	14.96	20.51
PoissonTreeBoost	9.14	26.40	56.13	66.34	1.57	6.32	11.84	18.17

it is reasonable to assume that a model that fits the popularity data well gives rise to an attribute weight vector that closely resembles user-perceived attribute importances. To investigate this hypothesis, we included the current section on model performance.

On all of the four product catalogs we have fitted the Poisson regression models and the PoissonTreeBoost algorithm. For the Poisson regression model, we use the same procedure as in [21]. To be able to estimate the missing values by multiple imputation, we delete attributes having more than 50% missing values and dummies with less than 10 ones. Furthermore, we used 25 imputations of the data. As with all members of the GradientTreeBoost family, three parameters need to be set in the PoissonTreeBoost algorithm: the number of base learners N, the learning rate ν, and the size of the regression trees used. Following [13], we set the learning rate ν to 0.1. Lowering the learning rate (and correspondingly increasing the number of base learners) did not lead to better results. We chose to use decision trees having a maximum depth of 6. This lead to better results than when using smaller trees, while using larger trees did not substantially improve results. The number of base learners was determined using cross validation. Note that, since the GradientTreeBoost algorithm does not depend on multiple imputation, we can use all attributes for this algorithm.

To test the accuracy of these models, we compared them to each other and to a single Poisson regression tree model [37]. The Poisson regression tree, for which we have used the Poisson option of the `rpart` package, is included to see whether our models fit the data better than a single tree. For the Poisson regression model, we only show results for the complete model, since in general accuracy does not differ much between a complete and a stepwise model.

To determine the in-sample and out-of-sample performance of the three algorithms on the four product catalogs, we divided the data 100 times randomly in a training (90%) and test set(10%), which is a common way to test model performance in boosting literature (see e.g. [5]). The in-sample performance

Table 3
Weights of the ten most important attributes (out of in total 45 attributes)for the MP3 player catalog using PoissonTreeBoost and a stepwise and complete Poisson regression model.

	Stepwise Poisson Regr.		Complete Poisson Regr.		PoissonTreeBoost	
	Attribute	Weight	Attribute	Weight	Attribute	Weight
1.	Brand	.242	Memory Size	.125	Color	.220
2.	Memory Size	.176	Brand	.119	Audio Formats	.186
3.	Audio Formats	.131	Voice Memo	.093	Brand	.090
4.	Battery Type	.106	Height	.090	Interfaces	.086
5.	Width	.090	Depth	.081	Photo Formats	.080
6.	Operating System	.086	Color	.057	Weight	.049
7.	Color	.084	Extendable Memory	.052	Price	.042
8.	Memory Type	.084	Operating System	.049	Memory Size	.035
9.			Width	.048	Height	.030
10.			Battery Type	.045	Memory Type	.026

is averaged over the 100 training sets, while out-of-sample performance is averaged over the test sets. Performance is measured by Deviance

$$Deviance = \frac{1}{I}\sum_{i=1}^{I}\left[y_i \log\left(\frac{y_i}{\hat{y}_i}\right) - (y_i - \hat{y}_i)\right] \tag{10}$$

and the mean absolute error (MAE).

As can been seen in Table 2, the PoissonTreeBoost algorithm is much better than the other two algorithms in minimizing both the deviance and MAE on the training data. On three data sets this also lead to a lower deviance on the test data, what in two cases is accompanied by a lower MAE than both other algorithms. Only on the MP3 player data, the MAE of the Poisson regression model is slightly lower. The product catalog on which the PoissonTreeBoost algorithm, despite a better in-sample performance, had both a higher out-of-sample deviance and MAE, is the notebooks catalog. In general, the boosting approach fits the data better than the Poisson regression model. However, more evaluation is needed to see whether this also leads to better weights on the data sets.

4.3 Determined Weights

Table 3 shows the ten attributes getting the highest weights for the MP3 player catalog according to the three methods: stepwise Poisson regression, complete Poisson regression, and PoissonTreeBoost. Although some attributes are considered to be important by all three methods (Brand, Memory Size, and Color are in all three top 10's), there are quite some differences between the weights

determined by the different methods. There is high correlation between many attributes in the data and the methods then have a preference for different attributes. For instance, boosting is somewhat biased to categorical attributes having a lot of values, since they have a higher probability to provide a good split in a tree. For the other three product catalogs similar patterns exist. In general, the difference in weights seems to be the largest between PoissonTreeBoost and both Poisson regression models.

5 Clickstream Evaluation

Remember that our aim was to find a weighted attribute-based dissimilarity measure which matches perceived (subjective) dissimilarities as closely as possible. The performance evaluation in Section 4.2 is no proper evaluation of the degree in which this aim was achieved because it merely considers goodness-of-fit. So, it is clear that we need an evaluation method that takes user perceptions into account. We performed two such evaluations: one using a clickstream analysis and another one based on a user experiment. We first describe the clickstream evaluation, while the user evaluation is deferred to Section 6.

Both evaluation methods are based on evaluation of top P recommendations of products given that the user is looking at another product (which we will call the reference product) in the product catalog. Such a list could one describe as: "When you are interested in this product, you would maybe also like these products". In the clicktream analysis, we determine good recommendations as products that are relatively often visited together in a session with the reference product. Then, we determine the overall quality of a top P of recommendations as the average of the relative co-occurrence of the products in this top P with the reference product. Finally, we average over all reference products, that are all products in the product catalog, to determine the overall quality of the recommendations. These top P's are determined in this evaluation using the different weighting schemes discussed earlier. However, this evaluation approach can be used to evaluate all kind of dissimilarity measures and other types of recommendations algorithms.

5.1 Mean Average Relative Co-Occurrence

In more detail, we looked for sessions in the clickstream log in which two or more products out of one of the four product catalogs were visited. We defined a visit of a product as a visit of the details page of a product. Since people generally look for a relative specific type of product, products that a

user visits during a session are probably considered to be similar by the user. Generalizing this, we can assume that products that are visited together in many sessions should be products that are considered to be similar by users in general. Therefore, we counted in the clickstream log for each pair of products in the catalog how often they co-occurred in sessions, which resulted in a product co-occurrence matrix **C**.

Consider the following hypothetical co-occurrence matrix

$$\mathbf{C} = \begin{pmatrix} 20 & 10 & 2 & 7 \\ 10 & 17 & 3 & 6 \\ 2 & 3 & 6 & 4 \\ 7 & 6 & 4 & 10 \end{pmatrix} , \tag{11}$$

where on the diagonal the total occurrence of a product is reported. We normalized this matrix by dividing each row i by the number of times product i was visited in a session (sessions in which only one product was visited were not taken into account), such that position ij in the matrix contains the relative occurrence of product j given that product i was visited in a session. Hence, we create a relative co-occurrence matrix **R** having elements $r_{ij} = c_{ij}/c_{ii}$. For the example, this results in the following matrix

$$\mathbf{R} = \begin{pmatrix} 1.00 & 0.50 & 0.10 & 0.35 \\ 0.59 & 1.00 & 0.18 & 0.35 \\ 0.33 & 0.50 & 1.00 & 0.67 \\ 0.70 & 0.60 & 0.40 & 1.00 \end{pmatrix} . \tag{12}$$

Since we assume that a high relative co-occurrence of two products implies that these two products are considered to be similar by users, we can use this matrix to evaluate a dissimilarity measure in the following way. First, we use the dissimilarity measure that we want to evaluate to compute a dissimilarity matrix between the products in the catalog and we transform the dissimilarity values into their row-wise ranks. For instance, assume that this yields

$$\mathbf{D} = \begin{pmatrix} 0 & 1 & 3 & 2 \\ 1 & 0 & 2 & 3 \\ 3 & 2 & 0 & 1 \\ 2 & 3 & 1 & 0 \end{pmatrix} . \tag{13}$$

In this example, $\mathbf{d}_4$ indicates that for product 4, the most similar product is product 3, followed by product 1 and product 2. Hence, a case-based recommender based on the distance measure used to construct $\mathbf{d}$ would recommend products in this order to a user that bought, or has otherwise shown interest in, product 4.

Supposedly, the recommendation of the top ranked product 3 is good if this product has a high session co-occurrence with product 4, as indicated by the value in matrix $\mathbf{R}$. In general, under the assumption that relative co-occurrence is a good measure for perceived similarity, a high average value of the relative co-occurrences for the top-ranked recommendations for all reference products is desirable. (The average is over all reference products.)

Since this average depends on the ranks in matrix $\mathbf{D}$, and these depend on the dissimilarity measure used, the average value can be used to evaluate dissimilarity measures. We term this measure it MARCO, which is an acronym for Mean Average Relative Co-Occurrence. Instead of taking the average over all top-1-ranked recommendations, one can also take the average over all recommendations with rank P and lower. In this case, the average relative co-occurrence is computed over all best, 2-nd best, ..., P-th best recommendations for each reference product. We call the corresponding measure $MARCO_P$.

In the example, the value for $MARCO_1$ is

$$MARCO_1 = \frac{.50 + .59 + .67 + .40}{4} = .54 \ . \tag{14}$$

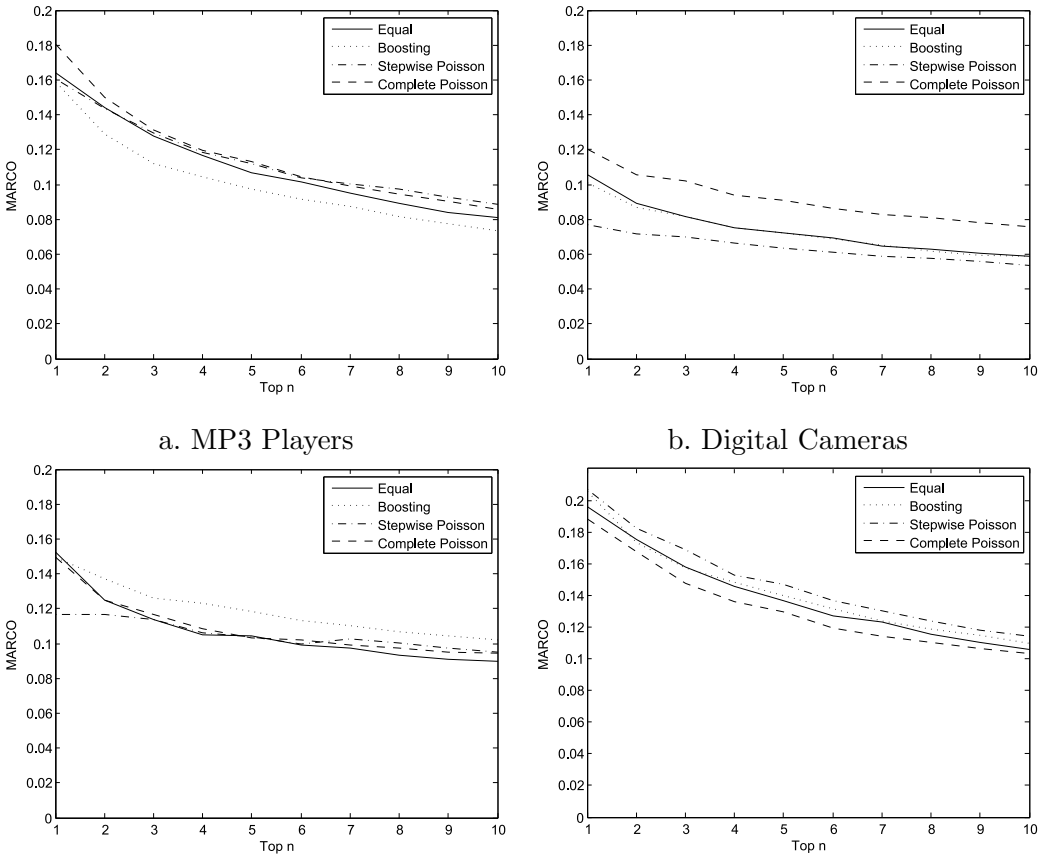
and the value for $MARCO_2$ is

$$\begin{aligned} MARCO_2 &= \frac{(.50 + .35) + (.59 + .18) + (.67 + .50) + (.40 + .70)}{8} \\ &= .49 \ . \end{aligned} \tag{15}$$

Expressed as an equation, $MARCO_P$ is

$$\begin{aligned} MARCO_P &= \frac{1}{I \times P} \sum_{i=1}^{I} \sum_{p=2}^{P+1} r_{i,\text{rankindex}(i,p)} \\ &= \frac{1}{I \times P} \sum_{i=1}^{I} \left(c_{ii} \sum_{p=2}^{P+1} c_{i,\text{rankindex}(i,p)} \right) \end{aligned} \tag{16}$$

where $\text{rankindex}(i, p)$ is the index of the p-th most similar product for reference



a. MP3 Players b. Digital Cameras

c. Notebooks d. Microwave Ovens

Figure 2. Mean average relative co-occurrence (MARCO) for top 1 until 10 for the different weighting methods.

product i. Note that $1 \leq P \leq I - 1$, and in the summation p starts at 2 so that relative co-occurrences of products with themselves are omitted.

Note that the MARCO is always between 0 and 1, where higher values indicate a higher recommendation quality. An optimal MARCO value could be obtained by directly use co-occurrences to provide recommendations. However, co-occurrences cannot be used when recommending new products or recommending products given a search query, which are situations in which (weighted) dissimilarity measures based on product attributes can be used. Additionally, although we use co-occurrences here as measure of evaluation, they may not be present in other systems in which a measure of popularity is present.

Table 4
p-Values of one-sided paired t-tests testing whether a weighting scheme performs better than using equal weights on a product catalog.

Product Catalog	Stepwise Poisson Regr.	Complete Poisson Regr.	PoissonTreeBoost
MP3 Player	**0.000**	**0.002**	1.000
Digital Camera	0.999	**0.000**	0.645
Notebook	0.054	0.079	**0.000**
Microwave Oven	**0.000**	0.854	0.051

5.2 Evaluation Results

We applied this approach on the four product catalogs described earlier and tested four dissimilarity weighting methods: equal weights, PoissonTreeBoost, and stepwise and complete Poisson regression. The approaches were applied to the adapted Gower coefficient measure (see Appendix A). The MARCO for different top P's from top 1 up to top 10 for four product catalogs is shown in Figure 2. As can be seen in Figure 2a, both Poisson model based dissimilarity measures generally perform better than the dissimilarity with equal weights on the MP3 player catalog, while the boosting approach does worse. In fact, the complete Poisson regression model is overall the best model followed by stepwise Poisson regression for this data set. Also for the digital camera catalog of which the MARCO plot is shown in Figure 2b, the complete Poisson model based dissimilarity measure does best. However for this data set, there is not much difference between using equal weights or weights determined by boosting and the stepwise Poisson regression performs worse than all other methods. The MARCO plot for the notebook catalog is shown in Figure 2c. For the top 3 and higher all weighting methods perform better than the equal weights approach. On this catalog boosting performs best, followed by both Poisson regression approaches. Equal weights only perform well when recommending a very small number of products. Finally, the plot for the microwave oven catalog is shown in Figure 2d. For this catalog the stepwise Poisson regression method is generally the best performing method followed by boosting both performing better than the method using equal weights. The complete Poisson regression model based dissimilarity measure performs worst on these data.

We also tested whether methods performed significantly better than using equal weights over all products in a catalog. To do so, we used a one sided paired t-test on the average relative co-occurrence of top 10's testing whether the model based method performs better than the equal weighted dissimilarity measure. The p-values of these tests are given in Table 4. On all product catalogs there is at least one method performing significantly better (at a 0.05 significance level) than the equal weighted dissimilarity. However, which method this is, is different among the product catalogs. Hence, although weighting of dissimilarity based on product popularity may improve recommendations

Figure 3. Screenshot of the online survey.

which method should be used changes among product catalogs.

6 User Experiment

We also evaluated the different weighting methods in an user experiment in the form of an online survey. In this survey, we collected data on which products users found to be relevant recommendations when they were looking at another product. These results could then be used to evaluate the different weighting methods.

6.1 Online Survey

People were asked to participate in this experiment via the Vergelijk.nl newsletter. In total, 70 people completed the survey. This experiment consisted of an online survey having three parts. In the first part, we asked participant to do the following. We gave them a randomly selected product, which they should treat as a reference product of which they wanted to see the details. Then, we also provided six other randomly selected products that served as potentially relevant products with respect to the reference product. The respondents were asked to select those products they thought to be a relevant recommendation for the reference product. A screenshot of this task is shown

Table 5
Relative weights of the different weighting schemes used in the experiment.

Attribute	Equal Weights	PoissonTreeBoost	Poisson regression	Average User Importance
Color	0.143	0.265	0.145	0.121
Brand	0.143	0.237	0.128	0.124
Price	0.143	0.191	0.238	0.167
Type	0.143	0.129	0.259	0.128
Volume	0.143	0.102	0.120	0.158
Max. Power	0.143	0.048	0.059	0.164
Model	0.143	0.030	0.051	0.139

in Figure 3. We repeated this procedure twice for each reference product. Every respondent had to perform this task for three reference products. Thus, per reference product, a maximum of twelve relevant products can be chosen by a respondent. Subsequently, we asked the respondents directly how important they judged the different attributes of the products on a five point scale. To avoid any order effect, the order in which the attributes were listed was also randomized per respondent.

For this experiment, we used the microwave oven catalog. However, to make the exercise manageable for the participants, we only selected seven attributes, that were, the seven attributes having the least number of missing values: Brand, Price, Color, Type, Volume, Max. Power, and Model. Only these seven attributes are shown to the participant. We also constrained the product catalog used in the experiment to the 25 most popular microwave ovens that did not have any missing value on one of these seven attributes. This was done, so that we could obtain relatively many observations for all considered products for the first part of the experiment.

The four weighting approaches were applied to only these seven attributes, which also means that the weights were determined using models only having these seven attributes as input. The four weighting schemes we evaluate are: equal weights for the seven attributes, PoissonTreeBoost, a complete Poisson regression model, and one using the average importance stated by the respondents (normalized to have sum of one). The weights of these four methods are shown in Table 5. The last column contains average weights specified by the users themselves in the survey.

6.2 Mean Average Relative Relevance

To evaluate the different weighting schemes, we used a similar approach as taken in the clickstream analysis in Section 5. We again evaluate top P's given a reference product. Only now, we define a good recommendation as a product that is considered to be a relevant recommendation by the respondents given

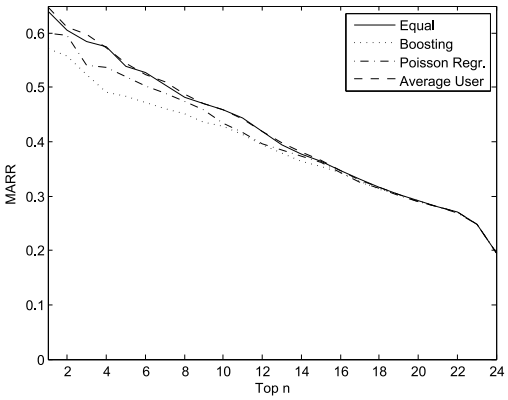


Figure 4. Mean average relative relevance (MARR) of four weighting schemes.

the reference product.

Therefore, we first counted how many times each product j was considered to be relevant given a reference product i and store this in a matrix $\mathbf{A}$. Contrary to the co-occurrence matrix used in the previous section, this matrix is not symmetric. Another difference with the co-occurrence matrix in the previous section is that the number of times a combination of a reference product and a potential relevant product is presented to a respondent is random. Therefore, we also counted how often each product j was shown to a respondent given reference product i and store this in a matrix $\mathbf{B}$. By dividing these two numbers (that is a_{ij}/b_{ij}), we get the relative relevance of products j given reference product i. By doing this for all products, a 25×25 matrix $\mathbf{R}$ is obtained. Then, a 25×25 dissimilarity matrix $\mathbf{\Delta}$ is computed for each of the four weighting schemes using the weights shown in Table 5 using the adapted Gower coefficient described in Appendix A. Note that Table 5 contains relative weights that sum to one. Similar to as was done in the clickstream evaluation, we use the rankorders in a row i of $\mathbf{\Delta}$ to order the relative relevances in $\mathbf{r}_i$. Based on this ordering, we can compute top the average relative relevance for different top P's using $\mathbf{R}$ and average this over all 25 products in a similar way as in (16). To acknowledge the difference in constructing $\mathbf{R}$, we will refer to this measure as the mean average relative relevance (MARR), defined as

$$\begin{aligned} MARR_P &= \frac{1}{I \times P} \sum_{i=1}^{I} \sum_{p=2}^{P+1} r_{i,\text{rankindex}(i,p)} \\ &= \frac{1}{I \times P} \sum_{i=1}^{I} \sum_{p=2}^{P+1} \frac{a_{i,\text{rankindex}(i,p)}}{b_{i,\text{rankindex}(i,p)}} \end{aligned} \tag{17}$$

where $\text{rankindex}(i,p)$ is the index of the p-th most similar product for reference product i.

6.3 Evaluation Results

Figure 4 shows the MARR for the four weighting schemes on the user experiment data. As can be seen in this plot, both model-based weighting schemes did not perform well in this experiment, since they did worse than the equal weighted dissimilarity measure. The dissimilarity measure using the weights based on the average importance according to users only performs slightly better than the equal weights.

It seems that respondents acted differently than in the clickstream evaluation. The limited number of attributes used in the experiment may have been a factor why the performance of the model based methods is quite poor. One thing the weighting methods can do quite well is giving very unimportant attributes very low or zero weights. However, these seven attributes were all considered relatively important attributes by the users, since they were have a mean value higher than 3 on a 5 point scale.

7 Summary, Conclusions & Discussion

Finding accurate attribute weights for case-based recommender systems is relevant because it may help to increase both accuracy and transparency of recommendations. In this paper, we have attempted to find methods for both identifying attribute weights, as well as for evaluating a set of attribute weights.

We have presented two methods to determine attribute weights, both methods modeling product popularity as a function of attribute values. The first method was based on a Poisson regression model, while the second method was based on a new boosting algorithm minimizing Poisson deviance. The latter model has the advantage that it has built-in methods to handle categorical attributes and missing values, which are common in e-commerce data. Also, it fits the data better, since it is more flexible and also able to handle interaction effects.

The methods and their resulting attribute weights were evaluated in a number of ways. The first evaluation used was aimed at model performance in terms of goodness-of-fit. In this evaluation, we used data from four real life commercial product catalogs with a corresponding clickstream log, and we evaluated how well the Poisson regression and boosting models could predict the number of 'outclicks' on a product. The latter model performed better in general, both in- and out-of-sample.

The second and third evaluations were aimed at determining how well dissimilarity measures using the resulting attribute weights matched user-perceived product dissimilarities. To this end, we extracted attribute weights from the the Poisson and Boosting models. We used those weights in the adapted Gower coefficient, and we developed two methods for determining how well the weighted Gower coefficient matches the user perceived similarities.

The first of these analyses was based on clickstream data. We developed a new measure, called MARCO, for assessing the quality of the weights in the Gower coefficient. Surprisingly, the only catalog on which the boosting based weights significantly outperformed the other models was the notebook catalog, which was also the only catalog on which the out-of-sample performance of boosting was *worse* than of Poisson regression. Hence, prediction performance of a method seems not to be a good predictor of whether a method provides good weights. Reasons for this might be the correlation between attributes and the bias of models for certain types of attributes.

Furthermore, in the evaluation on clickstream data, there was at least one model-based weighting scheme for each product catalog that was significantly better than using equal weights. This implies that attribute weighting based on product popularity improves recommendations. However, at this stage, it is not clear which model should be used in which situation and this is a topic for future research. Also, it may be the case that there exists another model based on product popularity that could improve recommendations on all product catalogs.

The second evaluation approach we used was a user experiment in the form of an online survey. Surprisingly, in this evaluation, the model-based weighting approaches performed worse than using equal weights, implying that users choose products differently in an experimental setup than on a real website. On one hand this result can be driven by the fact that the website setup leads users to certain products. On the other hand people may compare products in a different way during an experiment than they would do in real life. Another reason why the weighting models did not perform well in the user experiment, is that the number of attributes used was limited to only seven relative important attributes. The true contribution of the weighting methods may be the mere selection of these important attributes and not so much the difference in weights between these important attributes.

We see several directions in which this research can be extended or used. First of all, our results are inconclusive with respect to the single best method for determining attribute weights. Improving upon the proposed evaluation method, e.g. by deriving new features from the clickstream data, may alleviate this problem.

Both the clickstream analysis and the user experiment framework are set up in such a way that they can be used for evaluation of various types of (dis)similarity measures and weighting schemes. Such evaluations may potentially lead to a combination of a dissimilarity measure and weighting scheme that outperforms other approaches in both evaluations.

Also, we would like to study the use of the attribute weights in our previous work on map based recommender systems [20,22]. We hope that the usability of these systems will benefit from attribute weights. Finally, we think that the PoissonTreeBoost model can be a competitive alternative for Poisson regression models in different application fields. We believe that the weighting approaches combined with the dissimilarity measure can be a powerful addition for recommender systems.

Acknowledgements

We thank Compare Group for making their product catalog and clickstream data available to us and for distributing our survey via the Vergelijk.nl newsletter.

A Adapted Gower Coefficient

In the adapted Gower coefficient framework [20,21,22], the dissimilarity δ_{ij} between products i and j is defined as the square root of the weighted average of nonmissing dissimilarity scores δ_{ijk} on the K attributes

$$\delta_{ij} = \sqrt{\frac{\sum_{k=1}^{K} w_k m_{ik} m_{jk} \delta_{ijk}}{\sum_{k=1}^{K} w_k m_{ik} m_{jk}}} \ , \tag{A.1}$$

in which the w_k's are the weights for the different dissimilarity scores and, hence, for the different attributes. The binary indicator m_{ik} has a value of 1 when attribute k is nonmissing for product i. The weights w_k specify how important the different attributes are in the computation of the dissimilarity measure and, hence, in the application. In Section 3, we discuss two methods to determine these weights.

The computation of the dissimilarity scores δ_{ijk} in (A.1) is dependent on the type of the attribute. For numerical attributes, the dissimilarity score δ_{ijk} is

the normalized absolute distance

$$\delta_{ijk}^{N} = \frac{|x_{ik} - x_{jk}|}{\left(\sum_{i<j} m_{ik}m_{jk}\right)^{-1} \sum_{i<j} m_{ik}m_{jk}|x_{ik} - x_{jk}|} \ , \tag{A.2}$$

where $x_i k$ is the attribute value of product i for attribute k. For categorical attributes, the dissimilarity score δ_{ijk} is defined as

$$\delta_{ijk}^{C} = \frac{1(x_{ik} \neq x_{jk})}{\left(\sum_{i<j} m_{ik}m_{jk}\right)^{-1} \sum_{i<j} m_{ik}m_{jk}1(x_{ik} \neq x_{jk})} \ , \tag{A.3}$$

where 1() is the indicator function returning a value of 1 when the condition is true and 0 otherwise.

However, in many product catalogs, as will also be the case in the catalogs used in this paper, a third type of attributes exists, which we call multi-valued categorical attributes. Where a product can have only one value for a categorical attribute such as, for example, its brand, it can have multiple values for a multi-valued categorical attribute. For instance, an MP3 player can have an attribute called 'supported audio formats', which can contain the values MP3 and WMA at the same time.

In recommender systems, the products are often compared to a query and not to each other. When a user, for example, queries for an MP3 player that supports the audio formats MP3 and WMA, it does not matter for this user that a product also supports other audio formats, but when MP3 or WMA is not supported it does matter. Therefore, we use the following dissimilarity score for multi-valued categorical attributes in our framework

$$\delta_{ijk}^{M} = \frac{\sum_{s=1}^{S} 1(x_{iks} \notin x_{jk})}{\left(\sum_{i\neq j} m_{ik}m_{jk}\right)^{-1} \sum_{i\neq j} m_{ik}m_{jk}(\sum_{s=1}^{S} 1(x_{iks} \notin x_{jk}))} \ , \tag{A.4}$$

where x_{iks} denotes one of the in total S values product x_i has for attribute x_{ik}. Note that we normalize over all dissimilarities where $i \neq j$, since the dissimilarity score is not symmetric anymore.

B Derivation of PoissonTreeBoost

The PoissonTreeBoost algorithm can be derived by applying the Gradient-TreeBoost [13] framework to optimize the Poisson deviance loss function in

(3) . Boosting estimates target values as a sum of the predictions of the base learners

$$F_N(\mathbf{x}) = F_0 + \sum_{n=1}^{N} B_n(\mathbf{x}) \ . \tag{B.1}$$

The initial estimate F_0 can be determined by

$$\begin{aligned} F_0(\mathbf{x}) &= \arg\min_{\rho} \sum_{i=1}^{I} L(y_i, \rho) \\ &= \arg\min_{\rho} \left(\sum_{i=1}^{I} \left[y_i \log\left(\frac{y_i}{\rho}\right) - (y_i - \rho) \right] \right) \\ &= \frac{1}{I} \sum_{i=1}^{I} y_i \ , \end{aligned} \tag{B.2}$$

which implies that F_0 should equal the mean values of y. Then, we have to determine pseudo targets $\tilde{y}_i$ on which a base learner, that is, a regression tree, should be fitted by determining the negative gradient direction

$$\begin{aligned} \tilde{y}_i &= -\frac{\partial L(y_i, F_{n-1}(\mathbf{x}_i))}{\partial F_{n-1}(\mathbf{x}_i)} \\ &= -\frac{\partial \left[y_i \log\left(\frac{y_i}{F_{n-1}(\mathbf{x}_i)}\right) - (y_i - F_{n-1}(\mathbf{x}_i)) \right]}{\partial F_{n-1}(\mathbf{x}_i)} \\ &= \frac{y_i}{F_{n-1}(\mathbf{x}_i)} - 1 \ . \end{aligned} \tag{B.3}$$

On these pseudo targets we train an ordinary regression tree using a squared loss function. Finally, we have to replace the values of the terminal nodes by

$$\begin{aligned} \gamma_{pn} &= \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{pn}} L(y_i, F_{n-1}(\mathbf{x}_i) + \gamma) \\ &= \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{pn}} \left(y_i \log\left(\frac{y_i}{F_{n-1}(\mathbf{x}_i) + \gamma}\right) - (y_i - (F_{n-1}(\mathbf{x}_i) + \gamma)) \right) \ . \end{aligned} \tag{B.4}$$

Unfortunately, this minimum can often not be found analytically and therefore a line search in each node needs to be performed. Finally, we add our new base learner, a regression tree with terminal nodes R_{pn}, which have values γ_{pn}, to the boosting model

$$F_n(\mathbf{x}) = F_{n-1}(\mathbf{x}) + \nu \sum_{p=1}^{P} \gamma_{pn} 1(\mathbf{x} \in R_{pn}) \ . \tag{B.5}$$

procedure PoissonTreeBoost($\{\mathbf{x}_i, y_i\}_1^I, N, \nu$)
$F_0(\mathbf{x}) = \frac{1}{I}\sum_{i=1}^{I} y_i$
for $n = 1$ to N **do**
$\left\{\tilde{y}_i = \frac{y_i}{F_{n-1}(\mathbf{x}_i)} - 1\right\}_1^I$
Train tree B_n having terminal nodes $\{R_{pn}\}$ using $\{\mathbf{x}_i; \tilde{y}_i\}_1^I$
$\gamma_{pn} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{pn}} \left(y_i \log\left(\frac{y_i}{F_{n-1}(\mathbf{x}_i)+\gamma}\right) - (y_i - (F_{n-1}(\mathbf{x}_i) + \gamma))\right)$
$F_n(\mathbf{x}) = F_{n-1}(\mathbf{x}) + \nu \sum_{p=1}^{P} \gamma_{pn} 1(\mathbf{x} \in R_{pn})$
end for
end procedure

Figure B.1. PoissonTreeBoost algorithm

Note that there is also a shrinkage parameter ν incorporated in the GradientTreeBoost framework which is known as the learning rate. Setting the learning rate smaller than 1 implies that the results of a base learner are only partially incorporated in the model. Setting the learning rate ν very small, that is $\nu \leq 0.1$ leads to the best results [13,15]. The algorithm is summarized in Figure B.1.

References

[1] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, IEEE Transactions on Knowledge and Data Engineering 17 (2005) 734–749.

[2] B. Arslan, F. Ricci, N. Mirzadeh, A. Venturini, A dynamic approach to feature weighting, Management Information Systems 6 (2002) 999–1008.

[3] L. K. Branting, Learning feature weights from customer return-set selections, Knowledge and Information Systems 6 (2004) 188–202.

[4] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123–140.

[5] L. Breiman, Arcing classifiers, Annals of Statistics 26 (2) (1998) 801–824.

[6] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32.

[7] L. Breiman, J. H. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Chapman & Hall, New York, 1983.

[8] R. Burke, Knowledge based recommender systems, in: J. E. Daily, A. Kent, H. Lancour (eds.), Encyclopedia of Library and Information Science, vol. 69, Supplement 32, Marcel Dekker, New York, 2000.

[9] L. Coyle, P. Cunningham, Improving recommendation rankings by learning personal feature weights, in: Advances in Case-Based Reasoning; 7th European Conference, ECCBR 2004. Proceedings., vol. 3155 of Lecture Notes in Computer Science, Springer, Heidelberg, 2004, pp. 560–572.

[10] L. Coyle, D. Doyle, P. Cunningham, Representing similarity for CBR in XML, in: Advances in Case-Based Reasoning; 7th International Conference, ECCBR 2004. Proceedings., vol. 3155 of Lecture Notes in Computer Science, Springer, Heidelberg, 2004, pp. 119–127.

[11] Y. Freund, R. E. Schapire, Experiments with a new boosting algorithm, in: L. Saitta (ed.), Proceedings of the 13^{th} International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 1996, pp. 148–156.

[12] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.

[13] J. H. Friedman, Greedy function aproximation: A gradient boosting machine, Annals of Statistics 29 (5) (2001) 1189–1232.

[14] J. C. Gower, A general coefficient of similarity and some of its properties, Biometrics 27 (1971) 857 – 874.

[15] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, Springer, New York, 2001.

[16] J. L. Herlocker, J. A. Konstan, J. Riedl, Explaining collaborative filtering recommendations, in: Proceedings of the 2000 ACM conference on Computer supported cooperative work, ACM Press, New York, 2000, pp. 241–250.

[17] J. Honaker, G. King, M. Blackwell, Amelia II: A Program for Missing Data, r package version 1.1-27, available at `http://gking.harvard.edu/amelia` (2008).

[18] J. G. Ibrahim, M.-H. Chen, S. R. Lipsitz, A. H. Herring, Missing-data methods for generalized linear models: A comparative review, Journal of the American Statistical Association 100 (469) (2005) 332–346.

[19] A. Jameson, R. Schäfer, J. Simons, T. Weis, Adaptive provision of evaluation-oriented information: Tasks and techniques, in: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, 1995, pp. 1886–1893.

[20] M. Kagie, M. Van Wezel, P. J. F. Groenen, Online shopping using a two dimensional product map, in: G. Psaila, R. Wagner (eds.), E-Commerce and Web Technologies; 8th International Conference, EC-Web 2007. Proceedings., vol. 4655 of Lecture Notes in Computer Science, Springer, Heidelberg, 2007, pp. 89–98.

[21] M. Kagie, M. Van Wezel, P. J. F. Groenen, Choosing attribute weights for item dissimilarity using clickstream data with an application to a product catalog map, in: Proceedings of the 2nd ACM Conference on Recommender Systems, ACM Press, New York, 2008, pp. 195–202.

[22] M. Kagie, M. Van Wezel, P. J. F. Groenen, A graphical shopping interface based on product attributes, Decision Support Systems 46 (1) (2008) 265–276.

[23] G. King, J. Honaker, A. Joseph, K. Scheve, Analyzing incomplete political science data: An alternative algorithm for multiple imputation, American Political Science Review 95 (1) (2001) 49–69.

[24] F. Lorenzi, F. Ricci, Case-based recommender systems: A unifying view, in: B. Mobasher, S. S. Anand (eds.), Intelligent Techniques for Web Personalization, vol. 3169 of Lecture Notes in Computer Science, Springer, Heidelberg, 2005, pp. 89–113.

[25] P. McCullagh, J. A. Nelder, Generalized Linear Models, vol. 37 of Monographs on Statistics and Applied Probability, 2nd ed., Chapman & Hall, Boca Raton, 1989.

[26] D. McSherry, A generalised approach to similarity-based retrieval in recommender systems, Artificial Intelligence Review 18 (2002) 309–341.

[27] J. A. Nelder, R. W. M. Wedderburn, Generalized linear models, Journal of the Royal Statistical Society. Series A (General) 135 (3) (1972) 370–384.

[28] D. O'Sullivan, B. Smyth, D. Wilson, Understanding case-based recommendation: A similarity knowledge perspective, International Journal on Artificial Intelligence Tools 14 (1–2) (2005) 215–232.

[29] P. Resnick, H. R. Varian, Recommender systems, Communications of the ACM 40 (3) (1997) 56–58.

[30] F. Ricci, F. Del Missier, Supporting travel decision making throug personalized recommendation, in: Designing Personalized User Experiences in eCommerce, vol. 5 of Human-Computer Interaction Series, chap. 4, Springer, Netherlands, 2004, pp. 231–251.

[31] D. B. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, 1996.

[32] D. B. Rubin, Multiple Imputation for Nonresponse in Surveys, Wiley, New York, 1987.

[33] J. Sandvig, R. Burke, Aacorn: A CBR recommender for academic advising, Tech. Rep. TR05-15, DePaul University (2005).

[34] J. L. Schafer, M. K. Olsen, Multiple imputation for multivariate missing-data problems: A data analyst's perspective, Multivariate Behavioral Research 33 (4) (1998) 545–571.

[35] I. Schwab, W. Pohl, I. Koychev, Learning to recommend from positive evidence, in: Proceedings of the 5th International Conference on Intelligent User Interfaces, ACM Press, 2000, pp. 241–246.

[36] R. Sinha, K. Swearingen, The role of transparency in recommender systems, in: CHI '02 extended abstracts on Human factors in computing systems, ACM Press, New York, 2002, pp. 830–831.

[37] T. M. Therneau, E. J. Atkinson, An introduction to recursive partitioning using the RPART routines, Tech. Rep. 61, Mayo Foundation (1997).

[38] N. Tintarev, J. Masthoff, A survey of explanations in recommender systems, in: V. Oria, A. Elmagarmid, F. Lochovsky, Y. Saygin (eds.), Proceedings of the 23rd International Conference on Data Engineering Workshops, IEEE Computer Society, Los Alamitos, 2007, pp. 801–810.

[39] D. R. Wilson, T. R. Martinez, Improved heterogeneous distance functions, Journal of Artificial Intelligence Research 6 (1997) 1–34.