Providing Intelligent Assistance for Product Configuration in Manufacturing: A Learning-To-Rank Approach

Carsten Poggemeier[1], Matthias Hartung[2], and Philipp Cimiano[2]

[1]HARTING IT Services GmbH & Co. KG, Espelkamp, Germany,
Carsten.Poggemeier@HARTING.com
[2]CITEC, Bielefeld University, Germany, {mhartung,cimiano}@cit-ec.uni-bielefeld.de

Abstract

Configuring complex products can be a challenge due to the huge number of configuration possibilities. In this paper, our goal is to foster the development of intelligent configuration assistants that can support customers in configuring complex products. We formalize the task as a machine learning problem and in particular as a learning-to-rank problem. Given pairwise preferences elicited from experts, we show that we can train a model using support vector machines that ranks possible products according to their relevance to a given set of requirements specified by a user.

1 Introduction

Many industries produce complex products that consist of different sub-components and can be configured or assembled in many different ways to meet the needs of a specific customer or application. This is the case for the domain of manufacturing or engineering where products can be quite complex and display many configuration possibilities. Selecting the right product from all the possible configurations can be a significant challenge.

In this paper we are concerned with developing intelligent assistants in order to support the search for components or products that meet a certain set of requirements. We follow Hedin et al. [1] and define a product configurator as "a tool which supports the product configuration process so that all the design and configuration rules which are expressed in a product configuration model are guaranteed to be satisfied."

Instead of specifying the configuration rules by hand, we propose a machine learning approach that learns to score different configurations or products with respect to a set of requirements specified by a user. Our larger goal is to develop intelligent assistants that can engage with users in a dialogue and incrementally elicit their requirements, such that the set of possible configurations meeting these requirements is stepwise reduced.

We frame the task of supporting the search for products as a ranking problem in which a set of products or possible configurations needs to be ordered according to how well they match a set of user requirements. This approach is similar in spirit to research in the field of information retrieval and recommender systems. We assume user requirements to be specified in terms of certain values $v_1, .., v_n$ corresponding to a set of attributes $A_1, ..., A_n$. These attribute-value pairs $\langle A_1 = v_1, ..., A_n = v_n \rangle$ can be seen as a set of constraints that any solution needs to fulfill. However, given these requirements, two cases are thinkable:

- There is no product that meets all constraints at the same time, i.e., the constraints are not fully satisfiable.
- Given a set of requirements, there are still too many possible products and/or configurations that meet these requirements, i.e., the given requirements underspecify the product.

Our goal is to always present a ranked list of products and/or product configurations to a user. Using a machine learning approach overcomes both problems mentioned above in the sense that a learned approach will induce weights from training data that quantify how severe it is to violate a certain user requirement. In the case that the given set of requirements underspecifies a product, a machine learning approach can include other features or product characteristics in the form of priors to come up with a suitable ranking of products.

Using a machine learning approach and in particular a learning to rank approach has a number of advantages:

1. Arbitrary features can be flexibly incorporated into the model, without the need to write rules by hand. Priors can be incorporated as well in order to model that certain products are in stock, for instance, that certain products are more popular than others, or that the company has a certain priority to sell particular products.

2. Functions can be learned for specific application domains or even for specific customers, e.g., by interpolating the scores of different scoring functions.

3. Functions can be retrained continuously on the basis of information about which products customers actually bought.

In order to provide a proof-of-concept for our approach, we consider the domain of industrial connectors as products to be configured. Industrial connectors consist of an insert with a number of contacts and a housing that contains the insert and that is embedded into some appliance. Relevant parameters for the insert are number of contacts, amperage and voltage, among others. Relevant parameters for the housing are material, lever type and degree of protection, inter alia (see section 3.1).

As our main contribution in this paper, we present a learning-to-rank approach for complex technical products. We present results using different machine learning approaches as well as the methodology that has been followed in order to elicit training data from experts.

The paper is structured as follows: In the next section, we describe the problem more formally as well as the proposed formulation as a learning-to-rank problem. We further describe the features used and the methodology for acquiring training data. In Section 3, we describe our experimental settings and results. Before concluding, we discuss related work.

2 Ranking SVM Approach to Product Configuration

2.1 Problem Statement

Our problem can be formalized as follows: Given a set of product configurations P and a set of attributes $A = \{A_1, ..., A_n\}$ describing these products, our goal is to induce a function f that scores products in P according to the degree to which they satisfy the specific requirements of a user given as an n-tuple $(a_1, a_2, ..., a_n) \in A_1 \cup \{\perp\} \times A_2 \cup \{\perp\} \times ... \times A_n \cup \{\perp\}$.

Here, $\perp$ is used to denote that the user has not specified any preference with respect to the corresponding attribute. In the following we show how this problem can be formulated as a learning-to-rank problem.

2.2 Formulation as a learning-to-rank problem

Following Joachims [2], we formalize the task as a ranking problem. We define a user requirement as a tuple $r = (a_1, a_2, ..., a_n) \in A_1 \cup \{\perp\} \times A_2 \cup \{\perp\} \times ... \times A_n \cup \{\perp\}$, and a pairwise (preference) ranking $\geq_r$ of products in P such that $p_i \geq_r p_j$ if p_i matches the requirements specified in r better than p_j. Then, our goal is to induce a ranking function f such that:

$$\forall r \; \forall (i,j) \in P: \quad f(p_i, r) \geq f(p_j, r) \Leftrightarrow p_i \geq_r p_j$$

We approach this problem using the framework of Ranking SVM provided by Joachims [2]. The function f is assumed a linear model parametrized by a weight vector w that assigns a weight to all features:

$$f(p_i, r) = \vec{w}^T \; \Theta(p_i, r)$$

where $\Theta(p_i, r)$ is a feature vector that describes the match between p_i and r.

Given $R = \{(i,j) \text{ s.t. } p_i \geq_r p_j\}$, the model solves

$$\min(\overrightarrow{w}, \xi) = \frac{1}{2} \overrightarrow{w} \overrightarrow{w}^T + C \sum_{(i,j) \in R} \xi_{i,j}$$
$$\text{s.t. } \forall (i,j) \in R: (\overrightarrow{w}^T \Theta(p_i, r)) \geq (\overrightarrow{w}^T \Theta(p_j, r)) + 1 - \xi_{i,j} \quad (1)$$

with C > 0 and $\overrightarrow{w}$ as parameter vector. $\xi_{i,j}$ is the loss term, with $\xi_{i,j} > 0$.

2.3 Feature Extraction

In this section we describe the different types of features we use in our model. The features in principle describe the degree of match between a product $p \in P$ and a given user requirement r.

We distinguish between i) prior features, and ii) requirement-specific features. Intuitively, the prior features capture some aspect of a product independently of a user requirement. This captures the general a priori likelihood that some customer will be satisfied with a given product, no matter what his/her particular need is. The second type of features captures the degree of relevance of a product with respect to the particular requirements of a user. We describe these types of features in more detail in the following:

2.3.1 Prior features

Prior features capture the likelihood that a given customer will like product $p \in P$, given some intrinsic characteristic of the product that is independent of the given requirement.

A certain subset of these features is binary and specific for some product attribute A. The binary features $BF_A^v(p)$ can be formalized as follows:

$$BF_A^v(p) = \begin{cases} 1 & \text{if } p \text{ has value } v \text{ for attribute } A \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

2.3.2 Requirement-specific features

Requirement-specific features describe the degree of match between some product p and a given customer requirement r. We denote with $r[A]$ the value of attribute A as specified by a customer in his/her requirement r. We denote with $p[A]$ the value of attribute A for product p. Requirement-specific features capture the agreement or disagreement between the values $r[A]$ and $p[A]$ as follows:

$$RS_A^{v/v'} = \begin{cases} 1 & \text{if } r[A] = v \text{ and } p[A] = v' \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

For the special case of numeric attributes $N \subseteq A$, we define continuous requirement features $RS_N^{v/v'}$ as follows:

$$RS_N^{v/v'} = \Delta(v, v') \quad (4)$$

Here, $\Delta(v, v')$ is the distance between v and v' in some metric space. We use the signed L_1 distance: $\Delta_1(v, v') = v - v'$. As an example for a requirement-specific feature, we address the material of the housing. In our dataset, the material can have the following values:

1. metal
2. polyamide
3. high-grade steel

For example, if a user has indicated that she wants a metallic housing and the corresponding housing is metallic, then the feature $RS^{metal/metal}$ will have the value 1 while all other features $RS^{metal/i}$ with $i \neq metal$ will be 0. This models the case that the material of a housing matches perfectly the requirement of a user.

In contrast, if a user has indicated a preference for a metallic housing but the housing is made of polyamide, then the feature $RS^{metal/polyamide}$ will have a value of 1 and all the other features $RS^{metal/i}$ with $i \neq polyamide$ will have a value of 0.

In this way, the machine learning approach can induce how severe the violation of a certain constraint is and allow solutions which would not be possible if all constraints had to be met. Table 1 presents an overview of the features that are used in order to represent inserts and housings.

Table 1: Number of features for our products in relation to the different feature groups.

	Insert	**Housing**
Binary Features	12	17
Prior Features	12	24
Requirement-specific Features	18	93
RS_N Features	4	0
Total	42	134

2.4 Acquiring Training Data

In this section we explain how training data is elicited from domain experts who are product managers or salesforce personnel in the domain of industrial connectors. The procedure followed to acquire training data was as follows:

1. We interviewed 12 domain experts asking them to provide a specification of test cases including requirements that a user might have. This resulted in a set of 40 development cases D.
2. We asked a different set of domain experts to propose a best matching product configuration for a specific development case $d \in D$.
3. Then we asked a different set of domain experts to propose alternative matching products and assign a score to each product configuration along a scale ranging from 5 (best match) to 1 (lowest match).

Overall, using this methodology we acquired 783 ratings for inserts and 1849 for housings.

3 Experiments

3.1 Industrial Connectors: Background and Dataset

For the experiments reported in this paper, we rely on a dataset of *industrial connectors*. In mechanical

engineering, traffic engineering, automation technology or in the energy sector, for instance, industrial connectors are needed to transmit data, electricity or compressed air. One connector consists mainly of *two inserts* and *two housings*, each of which can be configured according to various individual attributes and rules of their combination. These will be described in the following. For this study, we do not consider additional material for connectors such as screws or cables.

Insert configurations An insert supplies the basis for the connector. The insert can transmit electricity or data or compressed air by a number of contacts. An insert can be specified by the following attributes:

- Number of contacts
- Voltage and amperage, if the inserts deals with electricity
- Termination technology
- Gender and size

In order to yield a viable configuration, inserts to be combined must be of the same size, number of contacts, voltage and amperage. Another rule is that inserts to be combined must be of opposite gender.

Housing configurations Housings are described by:

- Material
- Lever type
- Number of cable entries and their size
- Type and size
- Degree of protection

A housing configuration needs two housings which have the same material, lever type, degree of protection and size. In our dataset we have four different types of housings: One type is a top housing and three types are for the bottom part. Every housing configuration requires a combination of one top and one bottom part.

Combining inserts and housings into connectors In order to build an operable connector from inserts and housings, the insert configuration has to match the housing configuration with respect to their size.

In this dataset, we have about 3000 housing configurations and 300 insert configurations. Each housing and insert configuration is characterized by 46 and 40 attributes, respectively.

3.2 Evaluation Measures

We assess the quality of the ranked configurations produced by the system with regard to two aspects:

- **Overall ranking:** Is our ranking model capable of producing a relevant order in the sense that good results are ranked at the top of the list, whereas bad results have lower ranks?
- **k-best suggestions:** Under the assumption that customers will not be willing to explore the entire list of suggestions, this aspect focuses on the quality of those suggestions that are most likely to be considered as relevant by the customer, i.e., suggestions that are ranked at the k best ranks. With respect to these suggestions, we ask two questions: First, are the configurations that were assigned high expert ratings among the top-k suggestions provided by the model? Second, are the singular best configurations according to the experts among the top-k suggestions? Both these questions investigate the suitability of the model in an end-to-end recommendation system that is capable of replicating experts' recommendations.

To answer these questions, we use the metrics of *nDCG* (normalized discounted cumulative gain), *precision-at-rank* and *recall-at-rank* to evaluate the ranking. These metrics are typically used in information retrieval studies (see Manning et al. [3]). In all metrics, k is used to denote the rank that limits the number of configurations to be evaluated such that only suggestions at ranks $1, \ldots, k$ are considered. For instance, $k = 5$ means that we analyze the proposed configurations at ranks 1 to 5.

3.2.1 Normalized Discounted Cumulative Gain

nDCG considers relevance values and the order of the results list. The metric uses a function which progressively reduces the relevance value. This means that lower ranks have lower values (see Järvelin and Kekäläinen [4]). In a first step, the DCG_k (Discounted Cumulative Gain) is computed as follows:

$$\mathrm{DCG}_k = \sum_{i=0}^{k} \frac{2^{\mathrm{rel}_i} - 1}{\log_2(i+1)} \tag{5}$$

The term rel_i describes the relevance of the result at position i. In addition, we compute an ideal order of the list, called IDCG. To normalize the DCG, we have to compare the DCG values with the ideal values as follows:

$$\text{nDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k} \tag{6}$$

3.2.2 Precision-at-rank

Precision-at-rank indicates how many relevant suggestions are at the top of the ranking. In our approach, a configuration is considered relevant if its expert rating is at least 3 (cf. Section 2.4). All other configurations with values below 3 are considered irrelevant, as they do not meet the customer's needs to a sufficient extent. With respect to rank k, precision-at-rank is computed as follows:

$$\text{Pr}_k = \frac{1}{k} \sum_{i=0}^{k} \text{Correct}(i) \tag{7}$$

$$\text{Correct}(i) = \begin{cases} 1 & \text{if score}_i \geq 3 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

3.2.3 Recall-at-rank

We use a slightly non-standard version of Recall@k in which we check whether the best configuration identified by the experts is among the top k results.

3.3 Results

We report results using a 5-fold cross-validation regime (i.e., training on four folds and testing on the previously held-out fold). As ranking model, we use the RankingSVM that is available as part of the SVM$^{\text{rank}}$ toolbox with standard parameters (C = 0.01) and a linear kernel. We compare the performance of the RankingSVM approach in terms of the above mentioned measures to six other learning-to-rank approaches: RankBoost [5], Coordinate Ascent [6], AdaRank [7], ListNet [8] and Random Forests [9]. We use the versions of these learners implemented in the RankLib library version 2.5, also with standard parameters. We perform separate experiments for inserts and housings. Figures 1 and 2 show the results in terms of nDCG over increasing values of k for inserts and housings, respectively. Figures 3 and 4 show the results in terms of Precision@k over increasing values of k for inserts and housings, respectively. Figures 5 and 6 show the corresponding results for Recall@k.

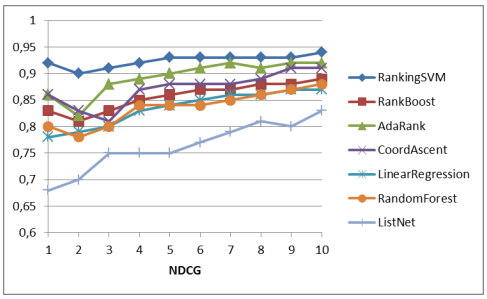

Figure 1: NDCG@k for inserts over different values of k.

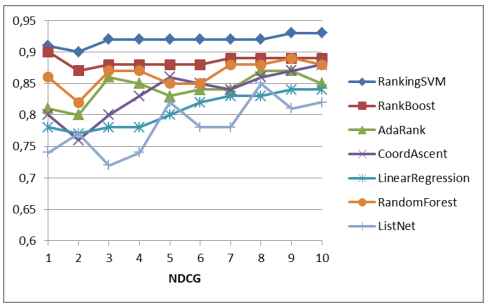

Figure 2: NDCG@k for housings over different values of k.

We discuss results with respect to the evaluation questions introduced in Section 3.2:

1) Are the configurations ranked in the correct order?

For both inserts and housings, the performance in terms of nDCG is remarkably high with values over 0.9 (see Figures 1 and 2), showing a strong correlation with the preference ordering provided by experts. In comparsion to the results of the other approaches, RankingSVM consistently stands out: For the inserts (see Figure 1), CoordinateAscent and AdaRank reach a value over 0.9 at $k = 10$, whereas ListNet only achieves a value of 0.83, which is 10% below RankingSVM. LinearRegression, RankBoost and RandomForest show strong overall performance as well, obtaining an nCDG between 0.85 and 0.9.

In case of housings (see Figure 2), the performance values of ListNet, CoordinateAscent and RandomForest strongly oscillate until $k = 10$. This is in contrast to inserts, where the values hold a level or slightly improve with an increasing k. RankBoost, RandomForest and CoordinateAscent reach a level of 0.88 at $k = 10$, whereas AdaRank, ListNet and LinearRegression stay below 0.84. In contrast, RankingSVM constantly performs beyond 0.9 for all k. This is the best result for housings across all learners. Therefore, we can conclude that, both for inserts and

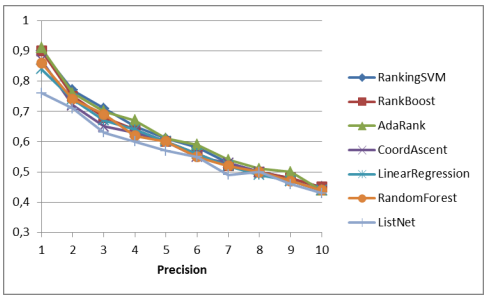

Figure 3: Precision@k for inserts over different values of k.

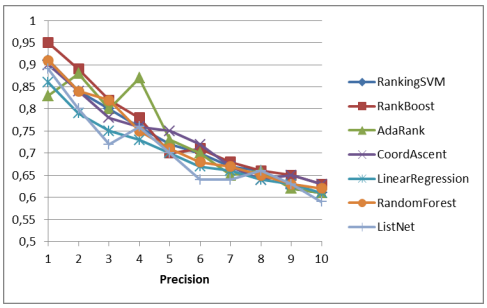

Figure 4: Precision@k for housings over different values of k.

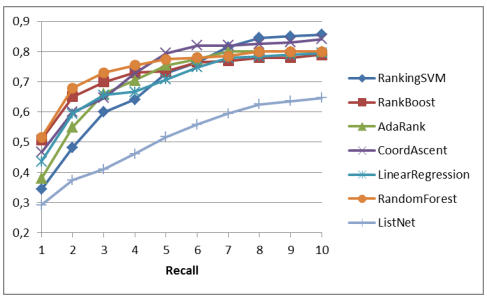

Figure 5: Recall@k for inserts over different values of k.

housings, the rankings produced by the RankingSVM approach contain the relevant product configurations at the top of the ranked list. Moroever, RankingSVM shows the most robust performance compared to all other approaches, i.e., the most stable results for different values of k.

2) Are the highest ratings at the top of the ranking?

Precision values start at 0.9 for inserts (see Figure 3) for all approaches except for ListNet and decrease quickly as k increases, as expected. At $k = 5$, the precision is higher than 0.6 for the RankingSVM, then decreases quickly until less than 0.5 at $k = 10$. The rapid decrease of precision for inserts is expected as the different inserts differ substantially in terms of

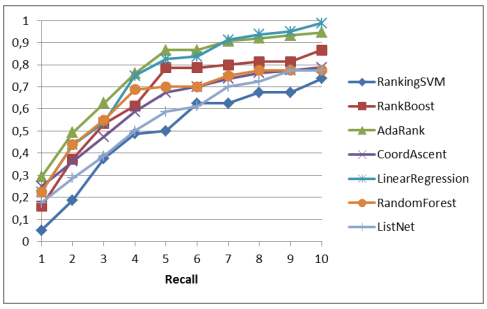

Figure 6: Recall@k for housings over different values of k.

their characteristics. Thus, there are only few inserts that are appropriate for each user-specified requirement.

For housings (see Figure 4), in contrast, the starting point differs between the different approaches. At $k = 1$, RankBoost obtains the highest value of 0.95, while the majority of the other approaches exhibits values between 0.85 and 0.91. The RankingSVM approach achieves a precision of 0.9 at $k = 1$, which decreases to 0.72 at $k = 5$ and 0.61 at $k = 10$. At $k = 10$, all approaches obtain similar performance between 0.59 and 0.63. Except for AdaRank (showing a positive outlier at $k = 4$), the precision curves are highly similar across all approaches, which suggests that learning-to-rank approaches generally bear a strong potential to provide highly precise recommendations at the top of the ranked lists.

3) Are the best configurations among the first results?

The results for recall on inserts are very remarkable (see Figure 5), with starting points between 0.4 and 0.5 for most of the approaches (and even 0.51 for RankBoost and RankingForest). Given that we investigate singular best results here, these values indicate that the best configuration is ranked at $k = 1$ in over 50% of the cases, which is a very good value.

Relative to all other approaches, only ListNet underperforms with a recall of 0.65 at $k = 10$. Departing from a relatively low recall of 0.34 at $k = 1$, RankingSVM achieves the overall best result across all approaches at $k = 8$. At $k = 10$, the RankingSVM yields a recall of 0.87, which means that the best solution can be found among the first ten recommendations in in 87% of the cases. The CoordinateAscent approach yields the best result at $k = 5$ (0.8), but is slightly inferior to RankingSVM at $k = 10$ (0.84 vs. 0.87).

On housings (see Figure 6), the RankingSVM obtains a rather low recall overall. At $k = 1$, AdaRank has the best performance of 0.3, while the other ap-

proaches achieve values between 0.15 and 0.25. It is interesting to note that AdaRank and LinearRegression reach values beyond 0.8 at $k = 5$. This is a very high value and was not expected, because our test data contains many similar housing configurations, on the one hand, and rather weak requirement specifications for housings, on the other. At $k = 10$, LinearRegression and AdaRank obtain a recall of more than 0.95, which means that in more than 95% of the cases we find the best result among the first ten suggestions. This is a very good result and shows that learning-to-rank approaches can effectively support product configuration by providing high-quality recommendations to customers.

3.4 Discussion

The results presented above point out two important variations: On the one hand, different performance levels can be observed for inserts and housings. On the other hand, the relative performance differs across the various ranking approaches evaluated here as well. This shows that not every ranking system can be expected to obtain the same performance for different products. For instance, while RankingSVM and CoordinateAscent stand out for inserts, they underperform on housing configurations. This suggests that, depending on the objectives underlying the particular configuration task, different ranking approaches may turn out effective.

In our case, we are mostly interested in a *consistent overall ranking* of product recommendations that (i) effectively reproduces experts' ratings and, at the same time, (ii) avoids patronizing the user by an immoderate focus on the top-ranked results only. We argue that some of the parameters that are relevant to the customer's final choice are possibly not fully exposed yet by the time of their specification (e.g., differences in price or shipment conditions); in such cases, it is of great importance that the recommendations presented to the user reflect a sound relative order, as this enables the customer to override individual suggestions by relying on the next-best alternative.

Moreover, in the configuration task investigated here, *inserts* are the main components of industrial connectors in that they have to meet the key requirements of the application intended by the customer, whereas housings potentially offer more degrees of freedom. For instance, in many electronic applications, customer requirements are primarily determined by specific constraints on the inserts. Apart from being compatible with the specification of the inserts, housings often do not impose any further constraints on the entire configuration in these cases. This is also reflected in our data set, which contains a considerable proportion of cases with only one or two requirements for housings.

Against this background, we consider RankingSVM most appropriate for the configuration task at hand, as this ranking approach clearly outperforms all others in terms of overall ranking performance and shows a superior recall curve for inserts (whereas in case of precision, all approaches are largely on a par).

4 Related Work

In information retrieval, the goal is to return a set of documents ranked by relevance with respect to the information need of a user which is typically expressed as a keyword query. Recently, the problem has been framed as a learning-to-rank problem and techniques from machine learning have been applied in order to learn a ranking function from relevance rankings of documents given a query (see Joachims [2], Robertson and Zaragoza [10], Piwowarski and Zaragoza [11]).

In the field of recommender systems, the goal is to learn a function that ranks items based on how likely a user will like them. The task of recommendation has also been formalized as a learning-to-rank problem, e.g., by Karatzoglou et al. [12].

Falkner et al. [13] apply classical recommendation techniques to the product configuration task. The authors propose different methods in order to support the configuration process, including exclusion/inclusion of features, ranking of features and entropy-based feature selection. In this case, the features describe decision criteria, i.e., the attributes a customer can use in order to specify her requirements. Falkner et al. focus on situations where no solution satisfies all constraints, using model-based diagnosis such as Max-CSP (Petit et al. [14]) or preferred explanations which can be solved by FastDiag (Felfernig et al. [15]). Beyond that, our approach presented in this paper also offers solutions in cases where the user requirements do not fully specify the desired product.

Tiihonen and Felfernig [16] show approaches of case-based recommendation techniques in relation to product configuration. They use an approach based on Naïve Bayes classification in order to suggest individualized product and service offers.

Apart from these efforts to integrate recommendation techniques into product configuration tasks, there is – to the best of our knowledge – no previous work that uses learning-to-rank models in this class of problems. Therefore, we consider our approach as a promising first step towards a practical combination

of recommendation and machine learning techniques in product configuration.

5 Conclusion and Future Work

We have addressed the task of supporting users in finding product configurations that meet their requirements. We have formulated the problem as an SVM optimization problem and proposed a learning-to-rank approach in order to induce a function that scores configurations according to how well they match the user requirements on the basis of a preference relation. We have described a methodology for eliciting preferences from domain experts and trained and evaluated our approach on a dataset consisting of 40 real use cases provided by domain experts from electrical engineering.

Comparing the performance of our SVMRank approach to the performance of six further classifiers, we have shown that the performance of our model is very positive, both in terms of correlation with the preference rankings provided by experts, as well as with respect to precision-at-rank and recall-at-rank metrics. Most remarkably, we have shown that the best configuration provided by experts for each use case can be retrieved within the top 10 results in 85% of the cases for connector inserts, and within the top 7 results in 90% of the cases for housings. These results show that learning-to-rank approaches bear a strong potential to be applied in end-to-end product configuration tasks.

In future work, we aim at integrating our learning-to-rank model into an intelligent assistant that engages with customers in natural language. This will pave the way towards an intuitive interface for non-expert customers in order to incrementally specify their requirements with regard to high-end technological products.

Acknowledgment

The second and third author acknowledge funding from the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277), Bielefeld University.

References

[1] G. Hedin, L. Ohlsson, and J. McKenna, "Product configuration using object oriented grammars," in *System Configuration Management*. Springer, 1998, pp. 107–126.

[2] T. Joachims, "Training linear svms in linear time," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 217–226.

[3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008.

[4] K. Järvelin and J. Kekäläinen, "Ir evaluation methods for retrieving highly relevant documents," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2000, pp. 41–48.

[5] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *The Journal of machine learning research*, vol. 4, pp. 933–969, 2003.

[6] D. Metzler and W. B. Croft, "Linear feature-based models for information retrieval," *Information Retrieval*, vol. 10, no. 3, pp. 257–274, 2007.

[7] J. Xu and H. Li, "Adarank: a boosting algorithm for information retrieval," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 391–398.

[8] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 129–136.

[9] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] S. Robertson and H. Zaragoza, "On rank-based effectiveness measures and optimization," *Information Retrieval*, vol. 10, no. 3, pp. 321–339, 2007.

[11] B. Piwowarski and H. Zaragoza, "Predictive user click models based on click-through history," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 175–182.

[12] A. Karatzoglou, L. Baltrunas, and Y. Shi, "Learning to rank for recommender systems," in *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 2013, pp. 493–494.

[13] A. Falkner, A. Felfernig, and A. Haag, "Recommendation technologies for configurable products," *AI Magazine*, vol. 32, no. 3, pp. 99–108, 2011.

[14] T. Petit, C. Bessiere, and J.-C. Régin, "A general conflict-set based framework for partial constraint satisfaction," *algorithms*, vol. 15, p. 13, 2003.

[15] A. Felfernig, M. Schubert, and C. Zehentner, "An efficient diagnosis algorithm for inconsistent constraint sets," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 26, no. 01, pp. 53–62, 2012.

[16] J. Tiihonen and A. Felfernig, "Towards recommending configurable offerings," *International Journal of Mass Customisation*, vol. 3, no. 4, pp. 389–406, 2010.