CHALMERS

Constructing a Context-aware Recommender System with Web Sessions

Master of Science Thesis in
Computer Science: Algorithms, Languages and Logic

ALBIN BRAMSTÅNG
YANLING JIN

Department of Computer Science & Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Constructing a Context-aware Recommender System with Web Sessions

ALBIN BRAMSTÅNG
YANLING JIN

Examiner: GRAHAM KEMP
Supervisor: OLOF MOGREN

Chalmers University of Technology
Department of Computer Science & Engineering
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science & Engineering
Gothenburg, Sweden June 2015

Abstract

During the last decade, the importance of recommender systems has been increasing to the point that the success of many well-known service providers depends on these technologies. Recommender systems can assist people in their decision making process by anticipating preferences. However, common recommender algorithms often suffer from lack of explicit feedback and the "cold start" problem.

This thesis investigates an approach of using implicit data only, to extract users' intent for fashion e-commerce in cold start situations. Markov Decision Processes (MDPs) are used on web session data to extract topic models. This thesis also explores how well the topic models can capture users intent and whether they can be used to produce good recommendations. The results show that this approach was able to accurately identify sessions topics, and in most cases the topics could successfully be translated to product recommendations.

Keywords: Recommender system, Context-aware, Topic models, E-commerce, Cold start, Markov decision process

Acknowledgments

The authors of this thesis would like to thank the following:

- Olof Mogren, for his support in this work.
- Jonas Hörnstein, for his guidance.
- 3Bits AB, for the opportunity to conduct this work and sharing data with us.

Contents

1 Introduction

Since the amount of information on the web is growing rapidly, information overload has become an increasing problem for users (Khoshneshin and Street 2010). For example, there exist e-commerce sites that offer millions of products (Linden, Smith, and York 2003). Consumers who need to select the right products quickly often lose themselves in the amount of options they have. Consequently, they often either miss the products they prefer or have a hard time making decisions. Recommender systems can be of great help when dealing with this challenge (Baltrunas and Amatriain 2009; Khoshneshin and Street 2010). Through anticipating information needs of users and providing suggestions, recommender systems may not only increase the overall user experience, but also link directly to revenue (Schafer, Konstan, and Riedl 1999; Yi et al. 2014).

The creation of new and better recommender systems has been an important field since the middle of the 1990s (Adomavicius and Tuzhilin 2005; Schafer, Konstan, and Riedl 1999), and it is still an active area of research and development today. There are scientific conferences dedicated exclusively to this topic, such as the ACM conference on Recommender Systems[1]. There have also been competitions offering tempting grand prizes, such as the Netflix Prize[2], in order to address the practical aspects of recommendation tasks.

Recommender systems are available across various domains, e.g. music, movies, blogs and news feeds (Adomavicius and Tuzhilin 2005; Schafer, Konstan, and Riedl 1999). In order to provide relevant recommendations in any domain, user interests and preferences towards the products need to be understood. This can be done through analyzing user interactions with products. User interactions are mainly classified into explicit and implicit feedback (Baltrunas and Amatriain 2009; Cho, J. K. Kim, and S. H. Kim 2002). Explicit feedback refers to the preferences that are learned without any assumptions, such as ratings. Implicit feedback is user preferences that are inferred from actions. For instance, in the domain of e-commerce, purchases and page views are indications of user interests.

Matrix-completion based collaborative filtering has become one of the most popular recommendation techniques with its success in the Netflix Prize competition (Aksel and Birtürk 2010; Khoshneshin and Street 2010; Yi et al. 2014). Collaborative filtering is a method of making predications about a user's preferences based on many other users who share the same taste (Cho, J. K. Kim, and S. H. Kim 2002; Ghazanfar and Prugel-Bennett 2010; Jambor and Wang 2010; Lu, Agarwal, and Dhillon 2009). It is stated by Campos Soto

[1] http://recsys.acm.org
[2] http://www.netflixprize.com

(2011) and Cho, J. K. Kim, and S. H. Kim (2002) that among all the possible information sources used by collaborative filtering, the most valuable is explicit rating. However, in most cases, users are rarely motivated to provide direct feedback to the service provider. The sparsity of explicit user ratings is thus a problem for recommendation tasks. Another problem that recommender systems are facing is the cold start problem (Adomavicius and Tuzhilin 2005; Braunhofer 2014; Gunawardana and Meek 2009; Ronen et al. 2013). This is referring to the cases where recommender systems have no information of the active user or items. For example, in the domain of e-commerce, some users log in to their accounts only at the moment of checkout. In this case, the recommender system has no information about previous purchases or visits by the users. This case is usually referred as "new user" problem. Another situation is the "new item" problem, which is related to new products being added to the inventory. In the collaborative filtering approach, products are recommended to the users who share the same preferences. This approach would fail to consider those new products which no one in the community has shown interest in.

1.1 Background

Lindex[3] is a European fashion chain that has over 480 stores in the world. Its fashion products are also sold online in all EU countries and Norway. Lindex online store contains a large number of products in various categories, where a recommender system can be of great assistance for customers.

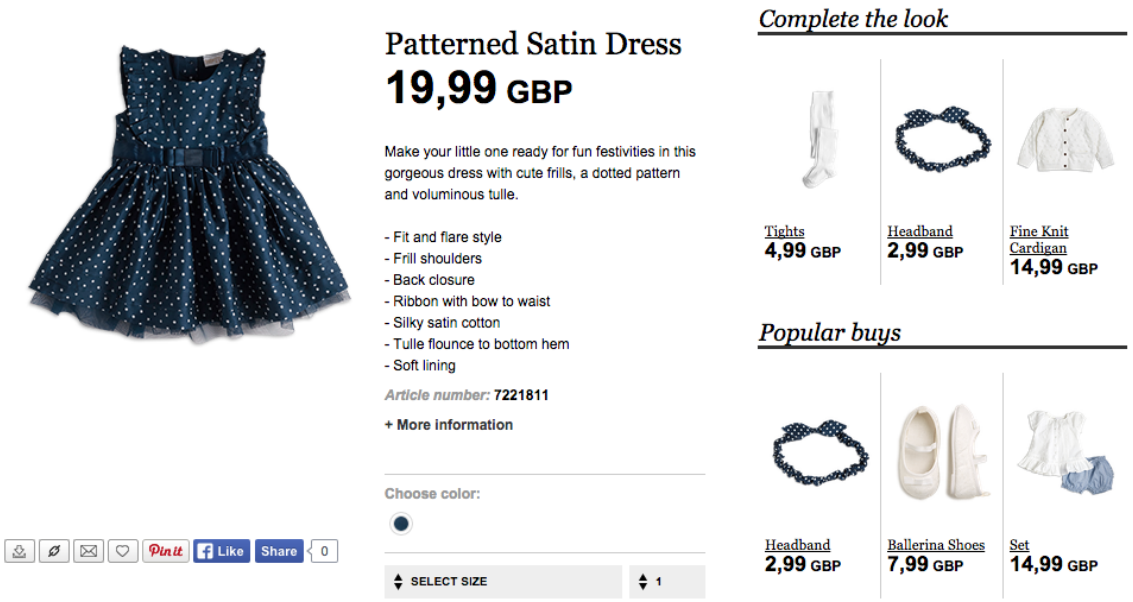

Figure 1.1: A Lindex product page.

Figure 1.1 shows a product page in Lindex online store. There are two sections to the right of the product, which are "*Complete the look*" and "*Popular buys*". *Complete the look* is reserved for products which are complements for the current product. It is edited

[3]http://www.lindex.com

manually by experienced Lindex staff. *Popular buys* are for products proposed by the current recommender system.

In 2014, Lundgren and Lindberg (2014) conducted a study on constructing and evaluating a recommender system in the domain of clothing e-commerce. Their thesis was supported by 3Bits Consulting[4] and used datasets provided by Lindex. They used on-going shopping cart data together with contextual information to find patterns using clustering and create recommendations with collaborative filtering. They mentioned that contextual information can be defined as time, location, weather, mood or device etc. Their proposed algorithm was better than a simple frequency algorithm, but was outperformed by the current recommendation system at Lindex. Due to time constraints, they performed simple contextual pre-filtering instead of a thorough investigation as planned. In the end, they concluded that the complexity of the solution might be unnecessary and a simpler solution could have sufficed. It was suggested by Lundgren and Lindberg (2014) that in order to increase recommendation accuracy, more complex context-awareness models should be investigated. Also, since data sparsity was one of the problems they faced, a suggestion was to use more data than just on-going shopping carts.

As the recommender system created by Lundgren and Lindberg (2014) did not match the initial expectations on prediction accuracy, this thesis aimed to create a new recommender system, with focus on more complex context-aware models and new data sources. This thesis was also supported by 3Bits Consulting, and the solution was tailored to datasets from Lindex.

1.2 Motivation

As mentioned, most recommender systems suffer from data sparsity or cold start problems. It is shown in figure 1.1 that there are "*Facebook Like*" and "*Add to Wish List*" options under the product from Lindex online store. However, the pre-study showed that the direct feedback collected by these features was too sparse to be usable as input for matrix factorization approaches. In this case, the implicit feedback was the only source that could be used due to its availability. Recent studies have suggested that web session data is an alternative to subjective user ratings (Alam et al. 2013; Cho, J. K. Kim, and S. H. Kim 2002; D.A., Z., and Y. 2014; Liu et al. 2010; White, Jose, and Ruthven 2001; Yi et al. 2014). Web session data records the visitor's path through a website and provides information essential to understanding shopping patterns or pre-purchase behaviors. The underlying assumption is that customers are interested in the products they visit. Moreover, Lindex web shop does not require a user to log in in order to make a purchase. Thus, in order to provide adequate recommendations, the recommender system for Lindex shall also deal with the cold start problem.

Generally, the effectiveness of recommender systems is measured in terms of prediction accuracy on user preferences (Desarkar, Sarkar, and Mitra 2010; Guy et al. 2010; Said and Bellogín 2014). Traditional recommender systems are designed to capture users' long term

[4]http://www.3bits.se

interests, which is under the assumption that users' preferences do not change (Tavakol and Brefeld 2014). However, in fashion industries, customer preferences are likely not consistent over time (Rendle, Freudenthaler, and Schmidt-Thieme 2010). The taste may be influenced by external factors such as the current fashion trend, or by time periods, such as Christmas. The unobservable internals such as mood changes and events happening in life can also influence the taste on clothing. Studies have shown that short term interests of users is valuable information to improve the quality of recommendations (Tavakol and Brefeld 2014). The aforementioned external and internal factors are usually referred to as "*context*", which can be used to capture a user's short term interests (Rendle, Freudenthaler, and Schmidt-Thieme 2010; Tavakol and Brefeld 2014).

In this thesis, web session data was used to detect users' preferences in cold start situations. Visited products in a session were used to generate topic models. Topic models that reflect the context can be seen as probabilistic semantics for retrieving information (Tavakol and Brefeld 2014). Evaluation was performed on the historical data provided by Lindex.

Recommender systems is not a new research field, there exist many studies on improving the prediction accuracy of a recommender system. However, a recommender system is normally designed and developed for specific datasets in a certain domain. For example, a recommender system for music recommendation, is not likely suitable for recommending real estate. The main contribution of this thesis is to choose the right approaches for designing and constructing a recommender system for fashion e-commerce. Moreover, as opposed to most existing solutions that focus on addressing users' long term interests, the approach in this thesis also aims to capture users' short term interest, which is another contribution.

1.3 Problem definition

Attribute	p1	p2	p3	p4	p5
Section	Shirt	Shirt	Shirt	Shirt	Shirt
Color	Blue	Black	Brown	Black	Black
Gender	Women	Women	Unisex	Women	Women
Size	Small	Medium	Small	Small	Small

Table 1.1: An example of user session.

Table 1.1 shows an example of a user session, which contains five visited products. The first item is a blue shirt for women with size small, followed by a black shirt for women with size medium and so on. Instead of approaching a product as a whole, we focus on each attribute independently. For example, with the *color* attribute, we get a sequence of *blue, black, brown, black, black*. The attribute *size* will return the sequence of *small, medium, small, small, small*, and so on. Each of the sequences gives an expectation of

associated value. For example, *shirt* is expected to be viewed together with *shirt* and *black* is expected to be viewed together with *black*.

To create a topic for the example session, a set of consecutive visited products is taken into account at each step. This results in sequences of attributes, e.g. (*blue, black*), (*black, brown*), (*brown, black*) and (*black, black*) for *color* with sequences of length two. The best value associated with each sequence is part of the topic for that sequence. This is done for all the attributes, so in each step in the session a tuple of best values can be produced and this tuple is defined as the topic at that moment. The topic is then used to retrieve products. The assumption is that with longer sequences the topic can reflect the whole session.

In this thesis, the aim is to identify the most probable occurrence given one set of attribute values using Markov decision processes (MDPs). The model is then used to identify the topic of a set of products within a session. Both value iteration and Q-learning are used for creating the model. Value iteration and Q-learning are two techniques to solve MDPs, which will be explained further in Section 2.3. Topics are used to capture user intent, which reflects the context the user is in. The recommendations are translated from topics.

1.4 Research Goals

Q1. *Given a set of items visited by a user, is it possible to detect the user's intent?*

As shown by multiple papers (Alam et al. 2013; D.A., Z., and Y. 2014; Liu et al. 2010), web session data can be used to improve the accuracy of recommendations. The idea is intuitive, since customer behavior on a web site should indicate the objective with the session, but to identify the goal is not a trivial task. An important problem is hence to isolate the parameters which describe the goal of the customer. More specifically, find traits of the products that the customer would purchase.

Q2. *Can topics created from user sessions be used to generate high precision recommendations?*

Research has shown that topic implies the context and context has an impact on product recommendations, and is thus worth investigating. The problem is to translate the topic to recommendations without losing its original meaning.

Q3. *Can a web session based approach with topic models be used to create a recommender system that handles cold start situations?*

We hypothesize that cold start situations can be solved by using topic models. The web session data can improve the topic accuracy adaptively with increasing knowledge on the products in the session. Since the topic is modeled as a list of attribute values of possible recommended products, the approach is expected to deal with the situation.

1.5 Outline

The thesis is structured as follows. Chapter 1 introduces the topic and motivates the problem setting. Chapter 2 reviews the related work on recommender systems and the recommendation algorithms. Chapter 3 introduces the data collected, technical contributions and presents the implementation details. Chapter 4 shows the empirical results, and reflects on the results. Chapter 5 provides conclusion by answering the research questions and recommends future work.

2 Related Work and Theory

This chapter provides an overview of existing recommender systems and recommendation algorithms. The aim of this chapter is to provide a relevant background about recommender systems. It also explains the related machine learning methods, Context-awareness and topic models. Since only existing methods and techniques are used in this work to solve a specific problem, this chapter serves as the foundation of the selected approach.

2.1 Recommender System Classification

As mentioned in the introduction, one of the challenges for online customers today is to process all the available information and make sensible decisions. Recommender systems are designed to help individuals deal with this problem and serve as an aid in the decision making.

In the literature, recommender systems are usually classified into three basic categories. These are content-based filtering, collaborative filtering and hybrid approaches. (Adomavicius and Tuzhilin 2005).

2.1.1 Content-Based filtering

In a content-based approach, relations between items is the core problem. How the relations are described depends on the problem, but in general items are compared based on their attributes. Movies, for example, can be described by genre, director, actors etc. Depending on how many of these attributes that match, it is possible to calculate similarity scores between all movies for a given service, and use these scores to find recommendations based on the viewing history of a user, or an explicit profile of preferences.

A benefit of using content-based filtering is that items that no one has visited or purchased still can be recommended. This is usable for newly added items in a system. A drawback however, is that since items are combined with an artificial score, and not by users, important connections may be overlooked. In most cases people's associations between items have higher relevance than a score created by the developers of the system (Adomavicius and Tuzhilin 2005).

2.1.2 Collaborative filtering

Collaborative filtering is the opposite of content-based filtering, since it uses similarities between users instead of between items. The general idea is to group users with similar behavior, e.g. purchases, and use these groups as a source for recommendations. When a user looks at a product, the user can be associated with one or more groups of users who have bought that product, and recommendations can be produced based on what these other users have purchased in addition to the current product.

Collaborative approaches usually suffer from data sparsity and the cold start problem, e.g. when a lot of users have bought very different products, it can be difficult to create groups of users with similar behavior. Another case is when no one has bought a particular product, and it gets visited by a user. In this case it is not possible to find users with similar behavior, because there simply are none. The pros of using a collaborative approach is that when lots of data is available, the recommendations become reliable, since they reflect actual user behavior (Adomavicius and Tuzhilin 2005).

2.1.3 Hybrid approaches

Combining a content-based and a collaborative approach is called a hybrid approach. This is a common method to counter the cons with both methods. The cold start problem in a collaborative approach can be solved by using similarities between items, and connections between items that has not been revealed by a content based approach can be detected by looking at similar users. How the approaches are combined depends on the specific problem (Adomavicius and Tuzhilin 2005).

2.2 Context-awareness

Context is an area in recommender system research which has become increasingly important. The idea comes from behavioral research in marketing, which has shown that decision making depends on the context of the decision (Adomavicius, Sankaranarayanan, et al. 2005). This means that customer preferences change, depending on the situation, and a recommender system which reflects these changes is called a context-aware recommender system. The exact definition of context is not fixed, but varies depending on the problem.

There have been multiple studies to test the influence of context. In 2005 a study was published which used a custom built web site to collect movie ratings and contextual factors, such as the time, the location and in what company the movie was watched (Adomavicius, Sankaranarayanan, et al. 2005). This data was used in a multidimensional collaborate-filtering recommender system. The name multidimensional means that the 2D user-item matrix was extended with the contextual information to form a multidimensional cube. The results showed that context can increase the precision of recommendations, but also that there are cases where it has no influence or even worsens the results. The

authors therefore claimed that selection of contextual factors should be done with care (Adomavicius, Sankaranarayanan, et al. 2005).

Another collaborative filtering method has been created by Baltrunas and Amatriain (2009), where the purpose was to recommend music. In this case contextual factors could not be incorporated beforehand, so the only available contextual data was time. The hypothesis was that people listen to different music depending on the time of day, and the results showed that using time as a contextual factor might increase the recommendation precision.

A bigger study by Domingues, Jorge, and Soares (2011) concluded that in order for context to have a significant impact is must add "rich contextual dimensions". Their experiments included data from three sources: two music web sites and one restaurant web site. For all three datasets, time and location were selected as contextual factors. In addition to this, genre, band and whether the music were instrumental or not was added to the context for the music data. For the restaurant data the customer's intention with the site visit was added. The datasets were then tested with an item-based collaborative filtering approach and association rules.

Higher level psychological surveys has also been performed. The authors Gorgoglione, Panniello, and Tuzhilin (2011) wanted to answer the questions how trust and purchase behavior depends on context. They worked together with a large Italian comic book company, to get access to real users for the purpose of A/B-testing. The company used non-personalized newsletters as a part of their marketing strategy, and the authors used this resource to ask users to participate in the survey. The users who agreed to participate were asked to rate a representative set of comic books to set up initial preference profiles. Three separate recommender systems were then used to produce recommendations: a content-based, a context-based and a random system. The random system was used as a baseline. For each set of recommendations, the users were asked to answer a set of questions, and also to specify two contextual parameters: the intention of a potential purchase and the current mood. All this information was used to update the profiles over time, and the result clearly showed that the contextual system outperformed the other two with regard to user trust, product diversity and the amount of sold products.

2.3 Topic detection

A problem with context-awareness is that contextual factors can be fully observable, partly observable or unobservable. If the factors are fully observable, modeling the context is not difficult, but in many real life situations this is not the case. An approach to solve this is topic detection, also called topic models. The hypothesis is that context affects the preferences of users, and therefore is reflected in their behavior. If this is true, then context does not have to be modeled directly, but can be captured implicitly with topics.

An attempt to test this hypothesis was performed by Hariri, Mobasher, and Burke (2012). They created a music recommender system, which used a social music tagging service to create latent topics in listening sessions. Topic prediction was performed by analyzing

the last n songs for a user, and perform sequential matching on a set of existing topics. If the predicted topics had a probability above a certain threshold, then those topics were considered as recommendation candidates. The system was compared to three other methods, and in general it performed better. Another result was that using topics can be a good approach to counter the cold start problem. New songs can be difficult to recommend, because they do not have any connections to other songs, but it is almost certain that they will fit into existing topics.

Tavakol and Brefeld (2014) also made a study about topic detection. The goal of their study was to investigate how well topic could be predicted in web sessions for a large e-commerce company in the clothing business. Their reason for choosing the topic based approach was that user intent is much easier to look for, than contextual factors. As others have argued (Hariri, Mobasher, and Burke 2012), the intent of the user can reflect the context the user is in. The method used in the paper is called a factored Markov decision process (fMDP), and their idea to model each topic as a separate fMDP was quite successful. They achieved prediction precision up to around 90%. They also manage to use the topics in a recommender system which outperformed several baseline collaborative filtering methods.

2.4 Markov decision process

A Markov decision process is a method for solving stochastic planning problems. These are planning problems extended with an uncertainty for each action. Because of the uncertainty the goal also differs from regular planning. Instead of searching for a particular goal state, the objective is instead to find an estimated goal. This means that the same set of actions in a given state space may produce different results. An example of where MDPs have successfully been used is game theory, and according to Otterlo and Wiering (2012) MDPs has become the *de facto* method for sequence based decision making.

Formally, an MDP is a tuple (S, A, T, R) of two sets and two functions. The sets are assumed to be finite for real applications, but in the general case infinite sets are allowed.

- S is the set of all possible states. A state consists of characteristics specific to the modeled problem.
- A is the set of all possible actions. Actions are used to switch between states.
- $T(s, a, s')$ is the conditional probability that action a in state s results in state s'. Can be rewritten as $P(s'|s, a)$.
- $R(s, a, s')$ is the reward for changing state from s to s' with action a.

The goal of an MDP is to collect the optimal amount of reward, and according to Otterlo and Wiering (2012) there are at least three definitions of optimality. In short these are: the sum of rewards in a finite time horizon, the sum of rewards in an infinite time horizon and the average reward in an infinite time horizon. These definitions are important because

they determine how the learning agent should handle future decisions in relation to the present.

To find the optimal reward in an MDP, the optimal policy π has to be found. A policy is a mapping between states and actions, that determines what action to take in each state. Optimal policies can be found in two ways: using V-functions or Q-functions. A V-function $V^\pi(s)$ describes the goodness of policy π in state s in terms of the expected reward. V-functions are used when both the transition function T and the reward function R are known. When T and R are unknown, Q-functions are used. They are also called state-action functions, because $Q^\pi(s, a)$ calculates the expected reward by performing action a in state s and then follow the policy π (Otterlo and Wiering 2012).

2.4.1 Solving using dynamic programming

For MDPs where the transition function T and the reward function R are known, there are two common algorithms for finding the optimal policy. These are called policy iteration and value iteration, and both are based on dynamic programming.

Policy iteration

Policy iteration has two steps for each iteration. The first is called policy evaluation, where the value function is calculated for the current policy. The second step is called policy improvement, where the next policy is calculated by maximizing over value functions. Both steps are repeated until convergence within some selected margin. The evaluation step can be formulated as:

$$V^\pi_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V^\pi_k(s')) \tag{2.1}$$

In the equation k depicts time steps, and γ is a discount factor that satisfies $0 \leq \gamma < 1$. The discount controls the influence of historical values.

The policy improvement step iterates through all actions to check if a better action can be found. The approach looks for the greedy policy π', based on the value function from the evaluation step:

$$\pi'(s) = \arg\max_a \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V^\pi(s')) \tag{2.2}$$

If the improvement step results in a new action for any state, the algorithm continues. Otherwise is stops, and the resulting policy is returned. With finite state and action sets, policy iteration can be shown to converge in finite time, but a bound on the number of iterations is not known (Otterlo and Wiering 2012).

Value iteration

Value iteration is similar to policy iteration, but the difference is that the evaluation and improvement steps are combined. Instead of converging using one policy and then evaluate

the policy, value iteration tries all actions directly, and then iterates to converge on optimal values for each state. The policy can then be calculated afterwards. The equation is:

$$V_{k+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_k(s')) \tag{2.3}$$

2.4.2 Solving using reinforcement learning

Reinforcement learning is an alternative when a model of the problem is not available beforehand, i.e. the transition function T and the reward function R. This problem can be solved with either direct or indirect reinforcement learning. An indirect approach interacts with the environment to approximate T and R, and then the dynamic programming methods from the previous section can be applied. However, most algorithms based on reinforcement learning have the direct approach. This means that no approximation of the model is performed, and instead the Q-function is estimated directly (Otterlo and Wiering 2012).

Another part of reinforcement learning is the selection of exploration policy. This policy controls the relation between exploration and exploitation. Exploration is required to test new actions, and exploitation is used to maximize the reward by selecting the currently best action. There exist many policies with different levels of sophistication, but one of the most common is the simple ϵ-greedy policy. It selects the best action with probability ϵ, and uniformly random among the other actions with probability $1 - \epsilon$ (Otterlo and Wiering 2012).

Q-learning

This is a common basic approach that follows a real session of requests, rather than exploring the whole state space. The Q-values for states and actions are calculated during the course of the session, based on observed state changes. The update formula is:

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha(r + \gamma \arg\max_{a'} Q_k(s',a') - Q_k(s,a)) \tag{2.4}$$

The Q-matrix is initiated with arbitrary values, e.g. $Q(s,a) = 0$, $\forall s \in S$ and $\forall a \in A$. The variables α and γ both satisfy $0 \leq \alpha, \gamma < 1$. α is called a learn rate parameter, and is usually low and fixed, or decreased in each iteration. Then, for each step k, an action is selected for the current state, based on the exploration policy. The next step s' is observed and the reward r is returned. With this experience $Q(s,a)$ is updated accordingly (Otterlo and Wiering 2012).

2.5 Evaluation methods

2.5.1 Online

In online testing of recommender systems, the purpose is to investigate how real users behave and react. It has been shown that user tests, in e.g. controlled lab simulations or surveys, can reveal important factors other than just accurate predictions. Herlocker et al. (2004) write that trust and confidence in the recommender system are important for users, and can be achieved by revealing the underlying reason for a recommendation. A/B-testing is another method used to monitor user behavior (Gorgoglione, Panniello, and Tuzhilin 2011) (Lundgren and Lindberg 2014). It is useful for comparing algorithms, since the users does not know that they are being surveyed.

2.5.2 Offline

Offline testing is an alternative to online testing, that is common in the recommendation system community. The general approach is to use historical data and split it up into a training set and a testing set. The split can be performed randomly or along a time axis. The training set is used to train the model, and the testing set is used to measure the prediction accuracy. It is important that these sets are disjoint, because testing on training data will in most cases make the results biased towards the training data, i.e. overfitting.

Offline testing is usually easier to perform than online testing, since no interaction with real users is required. On the other hand, with offline testing it can be difficult to capture other factors than prediction accuracy. Hence, the results can be used to compare algorithms, but should not be considered as representative for the whole problem.

The are two common approaches to measure recommendation accuracy: statistical and decision support (Campos Soto 2011). Two ways of measuring statistical accuracy is mean absolute error (MAE) and root mean squared error (RMSE).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \tag{2.5}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}} \tag{2.6}$$

Both methods give an error based on the difference between the predicted value p_i and the observed value a_i, but RMSE puts more emphasis on big errors because of the squaring. A lower error means higher accuracy. Since they measure the difference between scores, suitable applications are e.g. prediction of ratings (Campos Soto 2011) (Herlocker et al. 2004).

Decision support problems do not predict scores or ratings, but helps with making choices. Usually the task is to recommend a set of items, and the goal is to make as relevant

recommendations as possible. The most common approach to measure the accuracy of such recommendations, is precision and recall. Precision is defined as the ratio between the number of good predictions g and the total number of predictions t, which shows how many of the recommendations that actually are relevant (Herlocker et al. 2004).

$$precision = \frac{g}{t} \tag{2.7}$$

Recall is the ratio between the number of good predictions g and the number of all possible good predictions p, which shows how relevant the recommendations are overall (Herlocker et al. 2004).

$$recall = \frac{g}{p} \tag{2.8}$$

When both precision and recall are used for comparing algorithms, the F1-measure can be applied instead. It combines the two and returns the harmonic mean between them (Herlocker et al. 2004).

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.9}$$

Precision and recall depends on the definition of relevant items. Some researchers in the area of information retrieval have argued that relevance can be decided objectively. This means that if a user is searching for an item, either with a query or by viewing related items, the most relevant items are the ones which matches the query or viewed items the best. On the other hand, in a real situation, only the user can decide whether the recommended items feels relevant to the current objective. This turns it into a subjective problem, which shows that the concept of relevance may not be as useful in recommender systems as in traditional information retrieval. Another problem is that in most cases it is only possible to recommend a small number of items. If that is the case, precision is important to measure how many of the recommended items that are relevant to the user. Recall is less important, because the user probably does not care how many other relevant items there are (Herlocker et al. 2004).

3 Methods

This chapter describes the approaches and methods used in this thesis, including data collection, algorithm implementation and experimental setting. The aim of this chapter is to present the necessary means to realize the project.

3.1 Motivation of methods

As explained in the introduction, the goal of the thesis was to create a context-aware recommender system. In section 2.2 some examples of related work within this area was presented and the term context was described. Later, in section 2.3, some problems with context-awareness were addressed, and as the provided dataset from Lindex lacked explicit contextual factors, it was decided to use the topic based approach described by Tavakol and Brefeld (2014).

3.2 Data collection

The focus of this work is to extract meaningful information from web sessions from Lindex online store, in order to gain insights about customer preferences and interests on fashion products. Thus, primary data sources are web session data and product information. The datasets used in this thesis were collected from real life network traffic to Lindex online store. The data is unfortunately not public since there is a need to protect the company's business information and customer data based on the Swedish law of personal record.

Publicly available datasets were not chosen because recommender systems are data dependent. In order to conduct this work, traffic data from a real life fashion e-commerce site was needed. Due to the sensitivity of such information, we failed to find public datasets that fulfill our requirements.

3.2.1 Web session data

Web session data contains a sequence of network request-response transactions. The request messages contain actions that users performed on the resource. It is an alternative to explicit user feedback, since the user's intent can be derived from request messages.

Web session data is traditionally recorded by server logs. There are however a number of cloud services today for doing the same task of recording web logs, such as Google Analytics, which provides aggregated data on web traffic. As a strategy decision by Lindex, the current system is under transition to migrate from using traditional log files to cloud logging. The web session data used in this thesis is therefore retrieved from Application Insights[1] which is a service from Microsoft Azure.

Requirements and assumptions

Since we want to keep track of the products visited by real users who intend to make purchases in the online store, some requirements and assumptions were made to extract the data from the web sessions.

Req 1: A unique product id is required in order to identify products.
Req 2: A time stamp is required to identify when a product was visited.
Req 3: A session id is required to identify a session.
Req 4: The user who visited the product cannot be a bot.
Assump: One web session should contain between *2* and *50* visited products.

Pre-processing

The data collected by Application Insights can be viewed through Azure Portal[2]. However, in order to access it programmatically, the data needs to be exported to Azure storage[3]. The exported data is in JSON format and is stored as *blob* files. It is sorted based on request type and time. The types can be Request, Dependency, Exception etc[4]. The data that fulfills the specified requirements can be obtained from the Request type.

Figure 3.1 is an example of a request that is in JSON format. The request has a unique *id* that is 12356638698845046525, and it is a *HTTP GET* request. The url has a *base* which is /Assets/SiteV3/Pages/Product/Product.aspx and product *id* equals to 7231252. The request is made at *eventTime* 2015-05-14T03:57:26.4185891Z and it is from a *syntheticSource* called bingbot. The session has an *id* which is 0341b7260b86405f873ddf196d7c04e4.

C# was used to programmatically download the data from Azure Storage and Json.NET library was used to parse the JSON object, and extract the data fields according to the requirements. Specifically, the following fields were collected:

$$\lceil RequestId, SessionId, ProductId, EventTime \rceil$$

During the processing, the following steps were used to filter out unwanted data:

1. Filter based on request methods.

[1] http://azure.microsoft.com/en-us/documentation/articles/app-insights-get-started/
[2] https://portal.azure.com/
[3] http://azure.microsoft.com/en-us/services/storage/
[4] https://azure.microsoft.com/en-us/documentation/articles/app-insights-data-retention-privacy/

Figure 3.1: Request Example

The GET method from the HTTP specification is for retrieving a specified resource. The information in the resource contains the data relevant to the requirements, so a decision was made to only consider the GET requests.

2. Filter based on "base".

The example in figure 3.1 has *base* which is /Assets/SiteV3/Pages/Product/Product.aspx. It indicates that the resource the user is requesting is a product page. Since the goal is to get the visited products, other bases can safely be filtered out to remove pictures, scripts, etc.

3. Filter based on "syntheticSource".

The user who visited the product page cannot be synthetic, thus only data where syntheticSource is *null* is kept. Bots such as "bingbot", "YandexBot", "Googlebot", "Yahoo Bot" and "TwitterBot" are detected and will not be considered. The data shown in figure 3.1 is

thus disregarded.

All fields are stored in a database with $RequestID$ as the key, since it is unique. Database management was done through using Entity Framework, which provides the object-relational mapper that supports .NET framework with relational data using domain-specific object. It eliminates the need for writing most of the data-access code. As a concrete example, if the syntheticSouce in the data presented in figure 3.1 is null, the following entry would be added to the database.

$$\lceil 12356638698845046525, \quad 0341b7260b86405f873ddf196d7c04e4, \quad 7231252, \quad 2015-05-14T03:57:26.4185891Z \rceil$$

4. Clean the sessions according to request count in a session.

Based on the assumptions, the requests were grouped by sessionID, and the number of requests in each session was counted. Sessions with a length of less than 2 or more than 50, were removed from the databases.

Limitation

Since Application Insights is in the preview stage, up to 500 telemetry messages per second and up to 10 million page views or events per month are stored. The rest of the messages will be dropped. Hence, in case of a data peak, data might be lost.

3.2.2 Product information

Product information is retrieved directly from Lindex product database. A product can be described with a set of attributes, such as color, gender, size etc. In this thesis, the attributes are used to describe the topic of a session, so a full analysis of the product database was done.

Selected attributes

The attributes were selected empirically by observing the online store, product database tables and schemes. The following attributes were chosen:

- Gender
- Section
- Size
- Color

Section, size and color are self-explanatory. Gender in this thesis is not only referring to the state of being male or female, but also reflects the status of a person and the height. In total 16 genders, 102 sections, 315 sizes and 30 colors were identified. The sizes and colors were easily retrieved. But genders and sections required some preprocessing.

Process gender

By analyzing the products, a list of keywords could be obtained, and resulted in the following key-value dictionary.

Key	Value
44-68	0
56-86	1
86-122	2
128-170	3
girl	x
boy	y
women	X
men	Y
mom	Z
generous	+

Table 3.1: key-value dictionary

According to table 3.1, the distinct genders in table 3.2 were created.

Identifier	Gender
X	women
Y	men
X+	plus size women
XZ	mom
0	new borns
1	44-68cm children unisex
2	86-122cm children unisex
3	128-170cm children unisex
x1	44-68cm girl
y1	44-68cm boy
x2	44-68cm girl
y2	86-122cm boy
x3	128-170cm girl
y3	128-170cm boy
xy:	children unisex

Table 3.2: Genders

Process section

Since the section has many more keys than the gender has, it is not possible to manually create the keywords. For keyword generation, the entire product description and category description were used to search for the keywords dynamically. Words to ignore were identified, and the rest were normalized by transforming from plural to singular. This resulted 102 sections.

Limitation

There is a limitation in the attribute selection. Four attributes were selected to test the approach of using topic models on visited products. However, since the topic is formulated by attributes, the more attributes there are, the more detailed the topic is. Another limitation is in keyword accuracy. The keywords, that were selected manually for genders and dynamically for sections, are dependent on the product description. However, as mentioned by Lundgren and Lindberg (2014), some data fields were not consistent over time in Lindex database, such as the technical description of items that lack of uniqueness and was rather sparse in appearance.

3.3 Implementation

The MDP was implemented in two versions: one using indirect reinforcement learning and one using direct reinforcement learning. This section lists various key elements of the algorithms, and the algorithms themselves. A single MDP was used for each attribute. A state in the MDP was modeled to either contain 1, 2 or 3 consecutive requests.

3.3.1 Symbols and definitions

- S - The set of all states. A state is either a single attribute values, or a series of them.
- A - The set of all actions. To perform an action is to recommend an attribute. The action set is equal to the state set.
- U - A web session of requests for a single user. Each request is a state s.t. $U \subseteq S$.
- M - A matrix with dimensions $|S| * |S|$. Initialized with 0s.
- V - An array containing the value history for each state s.t. $V[s]$ is a chronological list of values for state s, with the most recent value last.
- X - An array s.t. $X[s]$ is the best action for state s according to the MDP.
- Q - A matrix s.t. $Q[s][a]$ is the value for choosing action a in state s.
- r - Reward. A numerical value used to promote good predictions.
- γ - Discount factor. Controls how much past values influence new ones.
- α - Learn rate. A factor used in Q-learning to control convergence speed.
- ϵ - The probability used in the EGreedy algorithm.

3.3.2 Indirect reinforcement learning

The indirect approach has two key parts: distribution approximation and value iteration. The distribution approximation is divided into algorithms 1 and 2. Algorithm 4 describes the value iteration, which uses algorithms 2 and 3.

The first part of the approximation consists of counting the number of times two attribute values have occurred together in the same session, for each session U in a certain training period. The result is an updated version of the M matrix.

Algorithm 1: Approximation of distribution

Input: U
Output: M

```
for u, u' ∈ U do
    M[u][u'] ← M[u][u'] + 1
end
```

The transition algorithm is a simple addition to the distribution matrix. It is used to calculate the conditional probability between two states s and s', based on the current M.

Algorithm 2: Transition

Input: $s, s' \in S$
Output: $P(s'|s)$

1 **return** $M[s][s'] / \sum_i M[s][s_i]$

The reward algorithm is used to reward good predictions. If a state s and a recommended state s' occurs in the same session, the recommendation is deemed successful, and the reward value is returned.

Algorithm 3: Reward

Input: $s, s' \in S$, U
Output: $r \in \mathbb{R}$

1 **if** $s, s' \in U$ **then**
2 | **return** r
3 **end**
4 **else**
5 | **return** 0
6 **end**

The value iteration algorithm is based on the value iteration described in section 2.4.1. The biggest formal difference is the removal of the summation. This relaxation can be done because the state space and the action space are equal. Additionally it reduces the complexity and increases the prediction precision, according to the tests performed during development.

The algorithm takes a session U and updates the values and actions for each request in U, until it converges. For each request u, all states in S are considered in the central equation at line 7. At the end of the inner loop at line 12, the best state to recommend for request u is stored in X and the value associated with that state is stored in V. Note that states to recommend are called actions. Convergence occurs when the difference between $V[u]_{k-1}$ and $V[u]_k$ is smaller than 1, for all requests in U. The variable k is the number of iterations performed until convergence.

Algorithm 4: Value iteration

Input: U
Output: V, X

```
converged ← false
while converged = false do
    for u ∈ U do
        bestAction ← null
        bestValue ← 0
        for s ∈ S do
            temp ← Transition(u, s) * (Reward(u, s, U) + γ * V[s])
            if temp > bestValue then
                bestValue ← temp
                bestAction ← s
            end
        end
        V[u] ← bestValue
        X[u] ← bestAction
    end
    for s ∈ S do
        converged ← true
        if V[s]_{k−1} − V[s]_k > 1 then
            converged ← false
        end
    end
end
```

3.3.3 Direct reinforcement learning

The selected algorithm for the direct approach was Q-learning. This algorithm does not use a probability distribution, but calculates the values for states and actions directly. It takes a session U and iterates through it once. For each request u, an action a is selected with the ϵ-greedy policy, the best action a_{max} for the next request u' is selected, and finally the new value for $Q[u][a]$ is calculated.

Algorithm 5: Q-learning

Input: U
Output: Q

for $u \in U$ **do**
  $a \leftarrow EGreedy(u)$
  $a_{max} \leftarrow FindMaxQAction(u')$
  $Q[u][a] \leftarrow Q[u][a] + \alpha * (Reward(u, a, U) + \gamma * Q[u'][a_{max}] - Q[u][a])$
end

The ϵ-greedy exploration policy is a simple but popular policy, that selects the best action for a given state with probability ϵ. It selects uniformly random between the other actions with probability $1 - \epsilon$.

Algorithm 6: EGreedy

Input: $s \in S$
Output: $a \in A$

$a \leftarrow FindMaxQAction(s)$
$rand \leftarrow$ generate random value, s.t. $0 \leq rand \leq 1$
if $rand < \epsilon$ **then**
  return a
end
else
  $a' \leftarrow$ find random action s.t. $a' \neq a$
  return a'
end

Algorithm 7 uses the values in the Q-matrix to find the best action for the given state. It iterates through all actions for the state, and selects the one with the highest value.

Algorithm 7: FindMaxQAction

Input: $s \in S$
Output: $a \in A$

```
bestAction ← null
bestValue ← 0
for a ∈ Q[s] do
    if Q[s][a] > bestValue then
        bestValue ← Q[s][a]
        bestAction ← a
    end
end
return bestAction
```

3.4 Performance analysis

The complexity of the value iteration is $O(I * |U| * |S| * (|S| + |U|) + C)$ where I is the number of iterations until convergence, $|U|$ is the number of states in the session U and $|S|$ is the number of states in the whole state space S. The complexity of the Q-learning is $O(|U| * (|A| + |U|) + C)$, where $|A|$ is the number of actions in the action space. In the the implementation $|S| = |A|$, and in a worst case scenario a session contains all possible states, i.e. $|U| = |S|$. This means that the complexity for value iteration can be rewritten as $O(I * |S| * |S| * |S| + C) = O(I * |S|^3 + C)$. The complexity for Q-learning can be rewritten as $O(|S| * |S| + C) = O(|S|^2 + C)$. This clearly shows that Q-learning is the fastest algorithm, but the drawback is that it requires more session data to converge, since it only iterates through each session once.

3.5 Evaluation

The tests were divided into three distinct parts: single attribute prediction, topic prediction and product prediction. The selected evaluation metric was precision. Recall was not included, mainly because of the reasons explained in section 2.5.

During the tests, the session data was divided into days. To simulate realistic behavior, the model was first trained on one day and tested on the next day. In the second run two days were used for training and the third day for testing. In this way the test data was gradually increased with one day at a time, and testing was always performed on the day after the last training day. This way of training and testing simulates the behavior of a real online system, where new data is added gradually. To test different convergence points, the training data was split in two periods.

3.5.1 Single attribute prediction

The single attribute test was performed with the attributes color, size, section and gender. In the test a session is a set of requests, and a request is a visited product and contains an attribute value. The test iterates through all sessions in the testing period, and produces recommendations. Each recommendation is based on either 1, 2 or 3 consecutive requests, and the pseudocode shows the case with 1 request, line 6. If the recommended attribute value occurs in the session, then the recommendation is successful.

Algorithm 8: Single attribute test

```
1  sessions ← select sessions in training period
2  train mdp with sessions
3  sessions ← select sessions in testing period
4  for session ∈ sessions do
5      for request ∈ session do
6          rec ← GetBestAction(mdp, request)
7          if rec ∈ session and rec ≠ request then
8              successful recommendation
9          end
10     end
11 end
```

3.5.2 Topic prediction

The purpose of the topic prediction test was to investigate the joint prediction rate of all 4 attributes together. Hence, a topic in this test is a tuple of attribute values. 4 MDPs are created, one for each attribute, and they are trained separately. A session is a set of requests, and a request contains all information about the visited product. Topics are created based on 1, 2 or 3 requests, and the pseudocode shows the case with 1 request, line 7. The if-statement at line 8 matches the recommended topic with the individual topic of all products in the session, and returns a result which depends on how well the topic matches. If all 4 attributes match, then the topic is 100% successful, but if e.g. only 2 of the attributes in the topic matches, then it is only a 50% match.

Algorithm 9: Topic test

```
1  sessions ← select sessions in training period
2  mdps ← create 4 MDPs for color, size, section and gender
3  train mdps with sessions
4  sessions ← select sessions in testing period
5  for session ∈ sessions do
6      for request ∈ session do
7          topic ← GetBestTopic(mdps, request)
8          if parts of or all of topic ∈ session then
9              successful recommendation
10         end
11     end
12 end
```

3.5.3 Product prediction

The product prediction is an extension of the topic detection, but instead of predicting the topic, the topic is used retrieve products. Similarly to the topic test, topics are selected based on 1, 2 or 3 consecutive requests, and the case with 1 request is shown at line 7. To get products, line 8, all products are first ordered by their overall popularity, and then the top products which matches the topic are selected. 1, 5 or 10 products are recommended, and a successful recommendation occurs if at least one of the retrieved products exist in the session, which is checked at line 9.

Algorithm 10: Product test

```
1  sessions ← select sessions in training period
2  mdps ← create 4 MDPs for color, size, section and gender
3  train mdps with sessions
4  sessions ← select sessions in testing period
5  for session ∈ sessions do
6      for request ∈ session do
7          topic ← GetBestTopic(mdps, request)
8          products ← GetProducts(topic)
9          if at least 1 product in products ∈ session then
10             successful recommendation
11         end
12     end
13 end
```

4 Results and Discussion

This chapter presents all the collected results, explains them and reasons about them. The results are divided into single attribute results, topic results and product results.

4.1 Experimental setting

The results presented in this section were collected using the MDP with value iteration and the MDP with Q-learning. The probability distribution used in the value iteration was also used directly, as a baseline. The whole time period for the tests was between 2015-02-10 and 2015-03-16. The period was split between February and March to test different convergence periods. The data from February is referred to as period 1, and the data from March is referred to as period 2.

In the graphs P is the probability distribution, V is the value iteration and Q is the Q-learning. The Y-axis shows the precision in percent, and the X-axis is divided into days. The actual dates are omitted to save space, and because they lack relevance. In order to capture the behavior in a session, multiple requests have to be considered when making predictions. The tests were therefore performed by taking 1, 2 and 3 requests into account. Because of performance reasons, value iteration was only used when taking 1 request into account. In the other cases only the probability distribution and Q-learning were used.

4.2 Individual attributes

The single attribute tests are divided into color, size, section and gender. Each test was performed for period 1 and period 2.

4.2.1 Color

The best average precision for color (60%) was achieved when only considering 1 request at a time. Both P and V succeeded with that (figures 4.1 and 4.2). Q on the other hand takes some time to converge, and averages 49%.

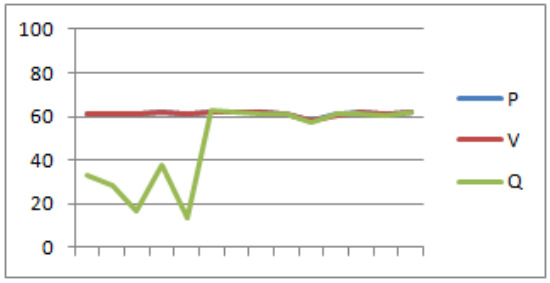

Figure 4.1: Color, 1 request, period 1.

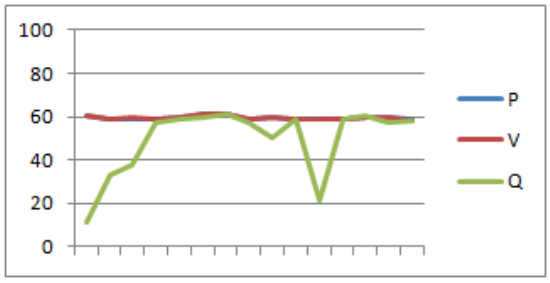

Figure 4.2: Color, 1 request, period 2.

With 2 requests (figures 4.3 and 4.4) there is almost no difference for P (59%), but Q converges directly to an average of 58%. The quicker convergence shows that by considering 2 requests the results are more stable.

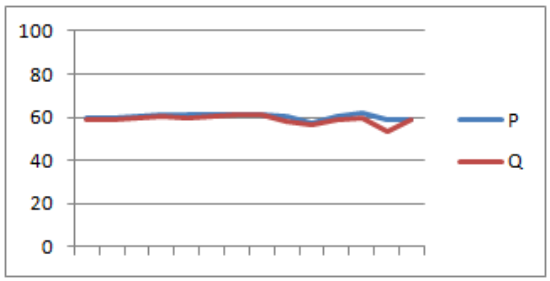

Figure 4.3: Color, 2 requests, period 1.

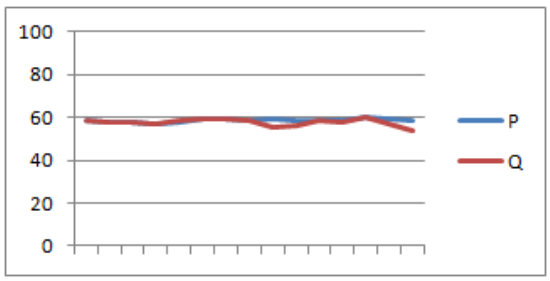

Figure 4.4: Color, 2 requests, period 2.

With 3 requests (figures 4.5 and 4.6) the precision drops for P to around 50% and Q drops to around 47%. The curve for Q also gets a bit bumpier.

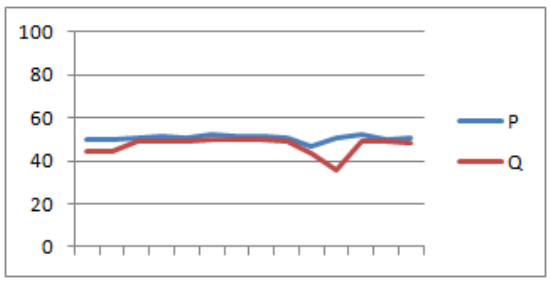

Figure 4.5: Color, 3 requests, period 1.

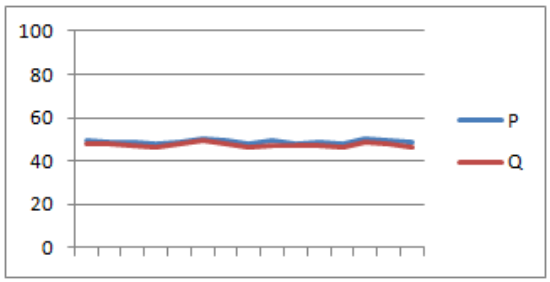

Figure 4.6: Color, 3 requests, period 2.

Overall the results show that color is difficult to predict. By looking at longer sequences does no seem to increase the precision, and a reason for that is probably that people look at different colors in different orders. This makes it more difficult to predict, and also indicates that color may not be that important for the topic. The reason why the results are not lower is because some colors are much more common that others, and therefore results in successful predictions because of high probability.

4.2.2 Size

Similarly to the color test, the best results were achieved with 1 request with P (64%) and V (64%) (figures 4.7 and 4.8). P and V are fairly stable, but Q never converges completely, which suggests that the periods are too short.

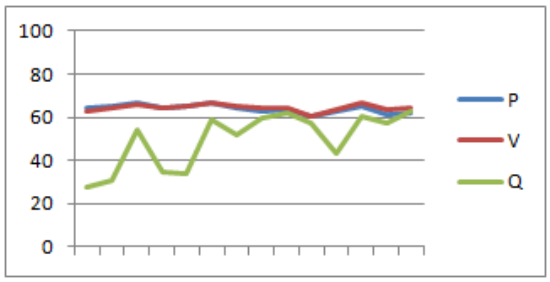

Figure 4.7: Size, 1 request, period 1.

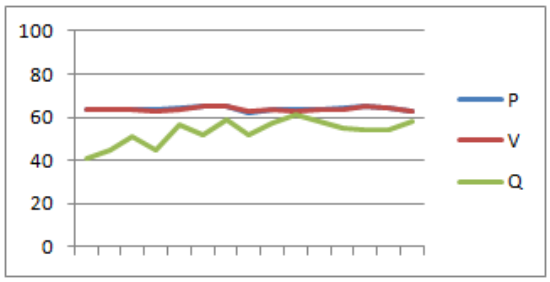

Figure 4.8: Size, 1 request, period 2.

With 2 requests (figures 4.9 and 4.10) the behavior is also similar to color, where Q is much more stable. Here P averages 62% and Q 58%.

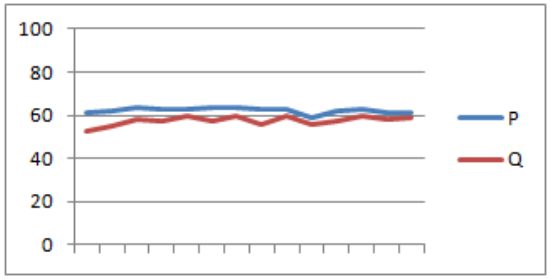

Figure 4.9: Size, 2 requests, period 1.

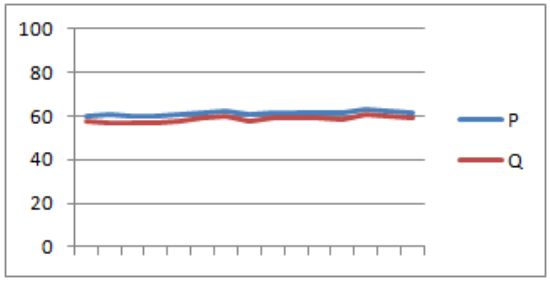

Figure 4.10: Size, 2 requests, period 2.

With size it was not possible to perform the test with 3 requests, because the state set in the MDP grew too big. The conclusion for size is that it also varies a lot, like color, but this has probably more to do with the fact that there are different sizes for different types of clothes.

4.2.3 Section

In the section tests P, V and Q performs more evenly in comparison to color and size. The results are also generally higher. The best results are achieved when considering 2 requests, P gets 94% and Q gets 93% (figures 4.13 and 4.14). These results are significantly better than the case with 1 request (figures 4.11 and 4.12), but only marginally better than the case with 3 requests (figures 4.15 and 4.16).

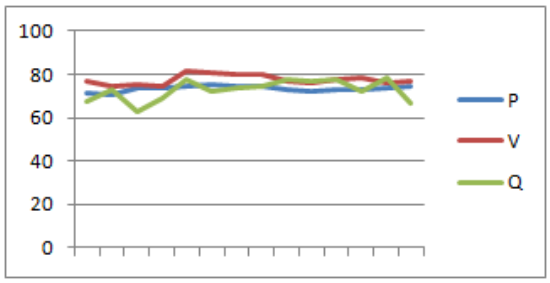

Figure 4.11: Section, 1 request, period 1.

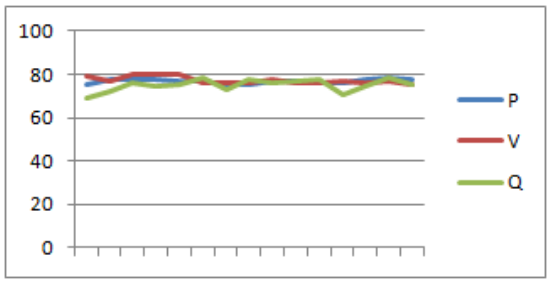

Figure 4.12: Section, 1 request, period 2.

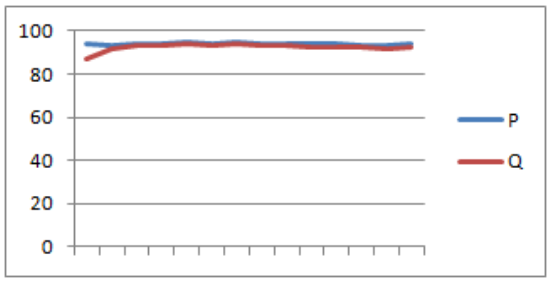

Figure 4.13: Section, 2 requests, period 1.

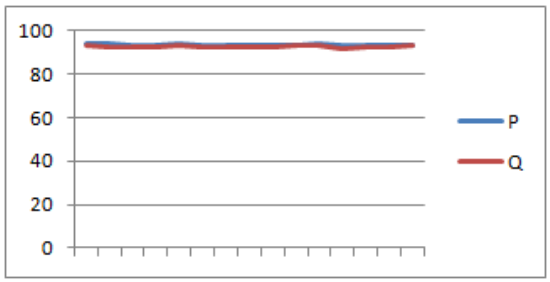

Figure 4.14: Section, 2 requests, period 2.

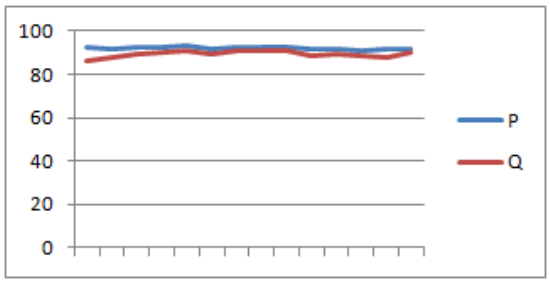

Figure 4.15: Section, 3 requests, period 1.

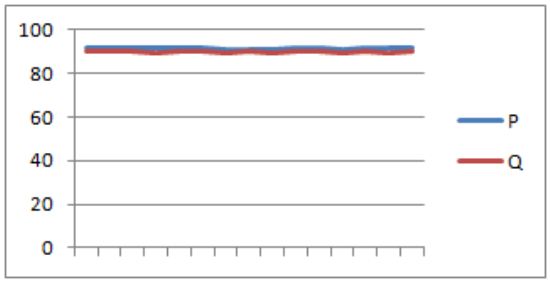

Figure 4.16: Section, 3 requests, period 2.

The general conclusion for section is that is seems to much easier to predict. There are two probable reasons for this. The first is that it is more likely that users stay within the same product section when looking for something to buy. This creates more predictable patterns, which is evident when considering the much faster convergence for Q in the case with 1 request. The precision also increase when considering 2 requests instead to 1, and only decrease slightly when considering 3, which supports this argument. The other reason may be related to the probability distribution for section. It may be the case that the distribution is very skewed towards a small set of categories, and therefore resulting in easier predictions.

4.2.4 Gender

The gender results are a bit surprising, since they decrease when taking more requests into account. Intuitively the target gender in a session should not vary, but apparently it does.

The decrease is quite significant, from around 85% (figures 4.17 and 4.18), to 78% (figures 4.19 and 4.20) to 71% (figures 4.21 and 4.22) for P, V and Q. One reason for this may be the way gender is defined in this project. It is a combination of sex and age, e.g. there is a difference between young girls and grown up women.

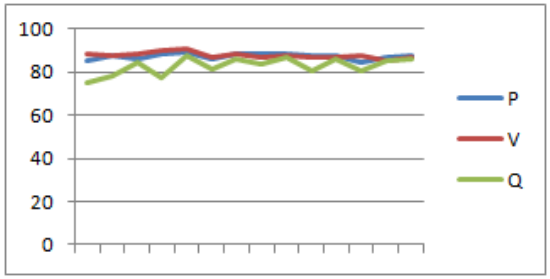

Figure 4.17: Gender, 1 request, period 1.

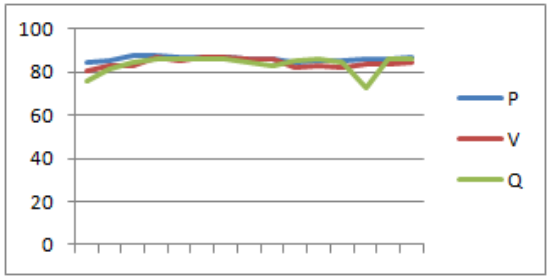

Figure 4.18: Gender, 1 request, period 2.

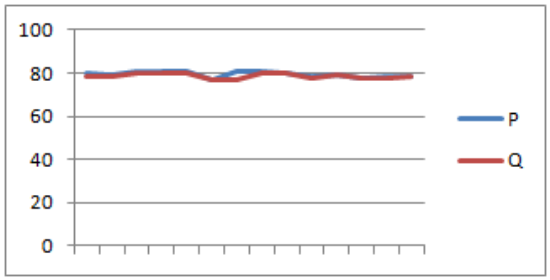

Figure 4.19: Gender, 2 requests, period 1.

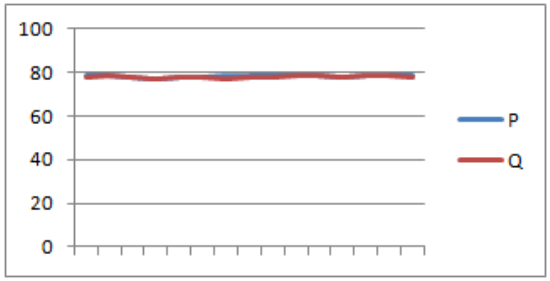

Figure 4.20: Gender, 2 requests, period 2.

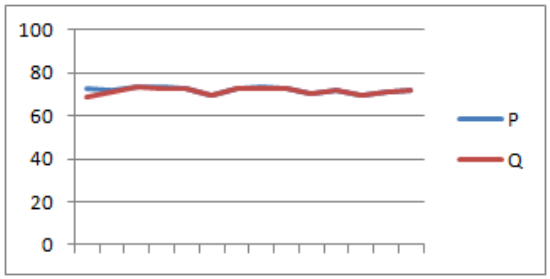

Figure 4.21: Gender, 3 requests, period 1.

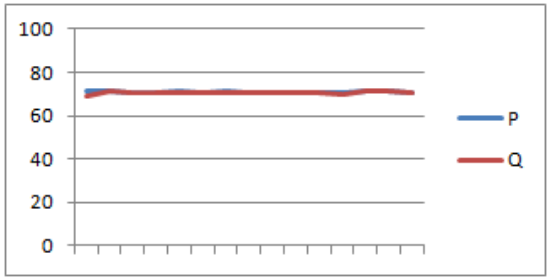

Figure 4.22: Gender, 3 requests, period 2.

4.2.5 Comparison with related work

When comparing the results from the single attribute tests with the results from Tavakol and Brefeld (2014) some similarities can be detected. The best results for color is 60% against their 93%, section has 94% against their 92% and gender has 87% against 93%. Tavakol and Brefeld (2014) did not perform tests on size. The biggest difference is color, but for section and gender the results are very similar. This reinforces their idea of using MDPs for topic detection, since the result could be recreated with another dataset.

4.3 Topic

The topic results are surprising, because Q performs better than P and V, with an average precision of 84% compared to 79% and 81%, in the case with 1 request (figures 4.23 and 4.24). The difference is not significant, but still better, considering that Q only performed worse than or equal to P and V in the single attribute tests. With 2 (figures 4.25 and 4.26) and 3 (figures 4.27 and 4.28) requests P increases to around 85% and Q decreases to around 82%. There is apparently no big difference between taking 2 or 3 requests into account. One thing to note however is that the topic with 3 requests does not contain size because of the previously stated performance issues.

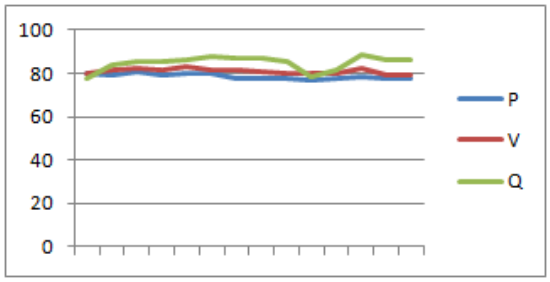

Figure 4.23: Topic, 1 request, period 1.

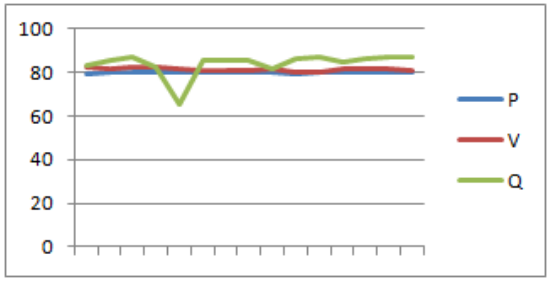

Figure 4.24: Topic, 1 request, period 2.

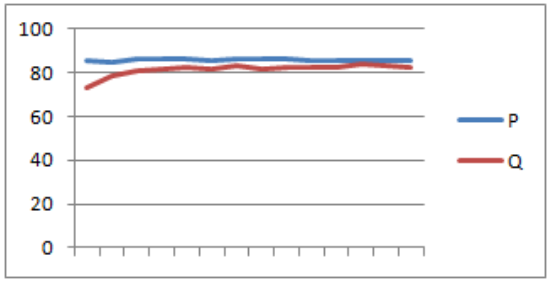

Figure 4.25: Topic, 2 requests, period 1.

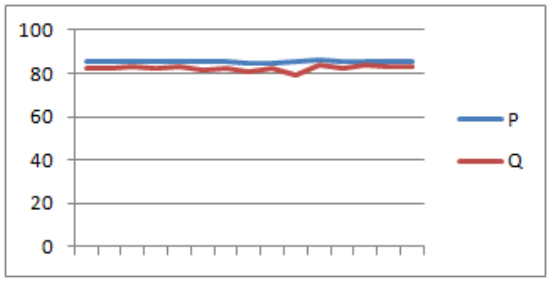

Figure 4.26: Topic, 2 requests, period 2.

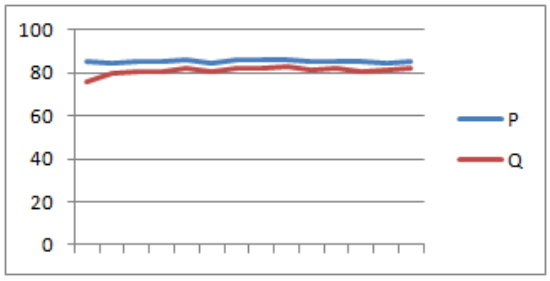

Figure 4.27: Topic, 3 requests, period 1.

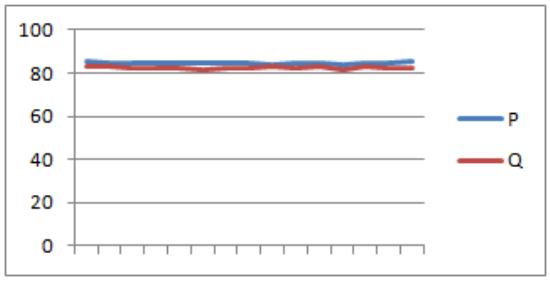

Figure 4.28: Topic, 3 requests, period 2.

4.4 Product

The product recommendations were made by creating a topic, and then select 1, 5 or 10 products to recommend. As explained in section 3.5 the precision was calculated on the whole set of recommended products, and not on individual products. As in the topic tests, size was not included in the tests with 3 requests.

4.4.1 10 recommendations

When making 10 recommendations based on 1 request (figures 4.29 and 4.30), there is a big difference between Q, P and V. This may be related to the topic results with 1 request (figures 4.23 and 4.24), where Q also performed better. But in this product test the difference is much larger, which is noteworthy. There is also a bigger difference between V and P.

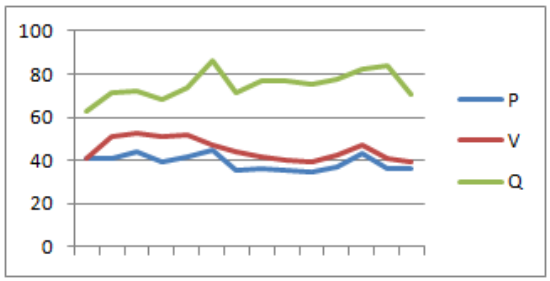

Figure 4.29: Products, 1 request, period 1.

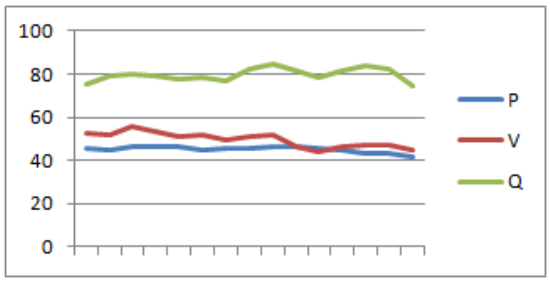

Figure 4.30: Products, 1 request, period 2.

In the case with 2 requests (figures 4.31 and 4.32), the precision for P and Q increases dramatically, to around 92% for P and 93% for Q. This is also the best result for 10 products.

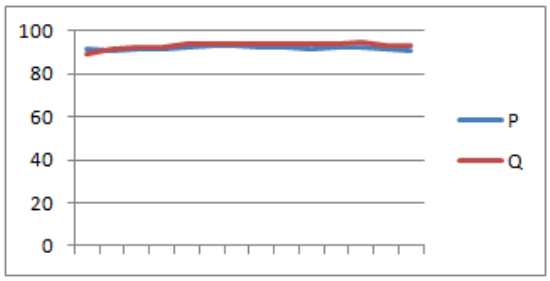

Figure 4.31: Products, 2 requests, period 1.

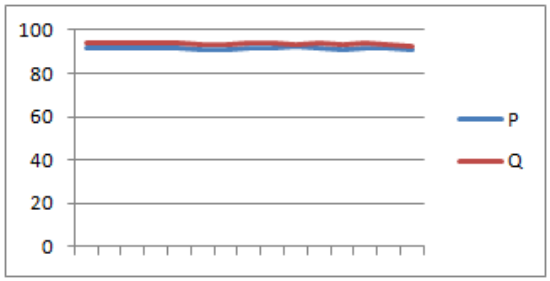

Figure 4.32: Products, 2 requests, period 2.

Using 3 requests (figures 4.33 and 4.34) produces worse results than using 2, 86% for P and 87% for Q, but still better than using only 1.

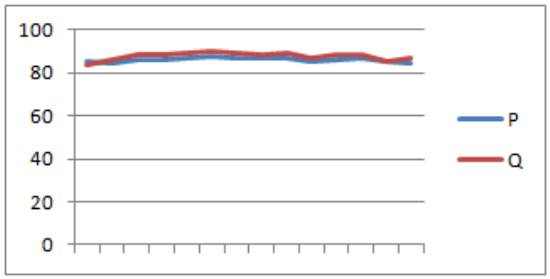

Figure 4.33: Products, 3 requests, period 1.

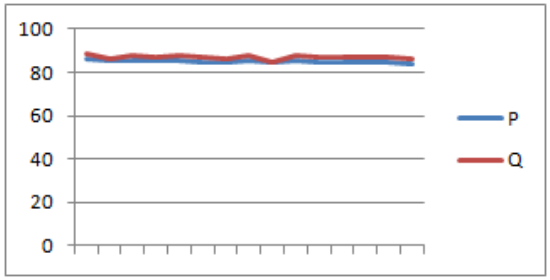

Figure 4.34: Products, 3 requests, period 2.

It is clear from these results that taking more than 1 request into account when making recommendations is better. The results are more stable, and the precision is higher. However, it is interesting to note that P and Q perform so similar. This suggests that the data is suitable for a probabilistic model, and that a more advanced machine learning algorithm like Q-learning may be unnecessarily complicated.

4.4.2 5 recommendations

Decreasing the number of selected products naturally lowers the precision, since the probability to get a match is decreased. The decrease is most obvious in the case with 1 request (figures 4.35 and 4.36), but there is a clear decrease for the other cases as well (figures 4.37, 4.38, 4.39 and 4.40). The relative difference between the algorithms does change, and the best results are achieved when using 2 requests.

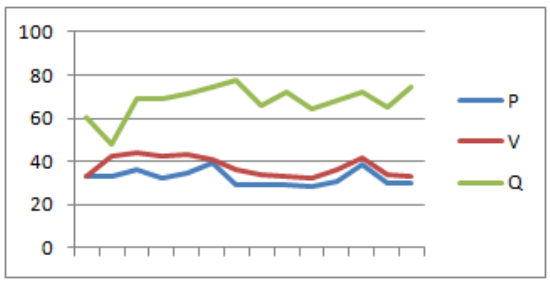

Figure 4.35: Products, 1 request, period 1.

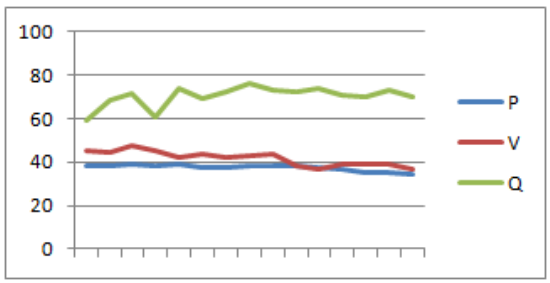

Figure 4.36: Products, 1 request, period 2.

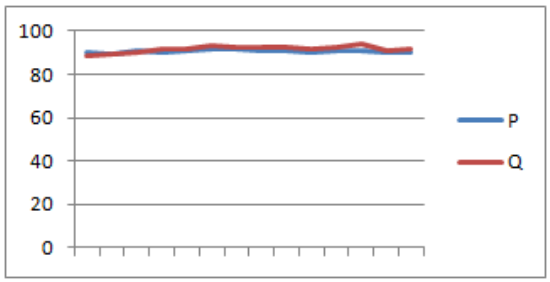

Figure 4.37: Products, 2 requests, period 1.

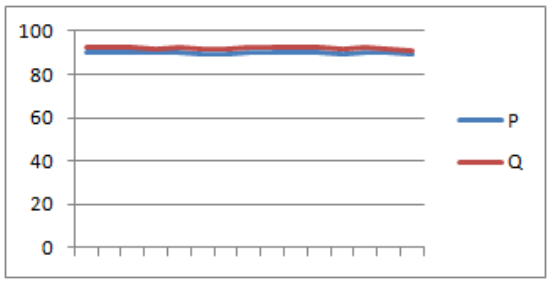

Figure 4.38: Products, 2 requests, period 2.

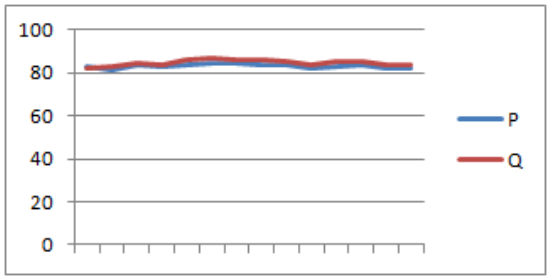

Figure 4.39: Products, 3 requests, period 1.

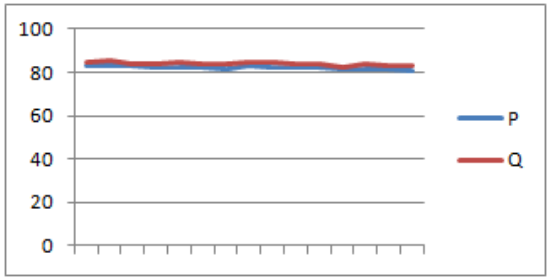

Figure 4.40: Products, 3 requests, period 2.

4.4.3 1 recommendation

The trend spotted for five products continues when only selecting one single product to recommend. The case with 1 request (figures 4.41 and 4.42) gets much lower precision, but even if there is a decrease in precision for the other cases with 2 (figures 4.43 and 4.44) and 3 (figures 4.45 and 4.46) requests, their results remain high. This seems a bit strange, and may depend on the data distribution more than the model itself.

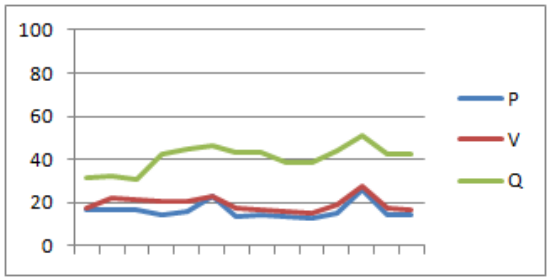

Figure 4.41: Products, 1 request, period 1.

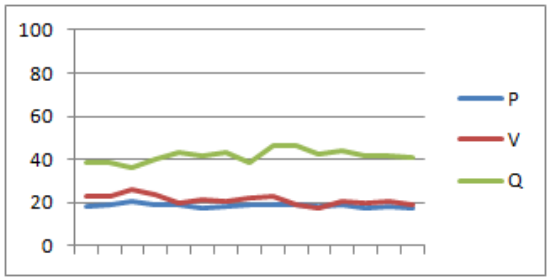

Figure 4.42: Products, 1 request, period 2.

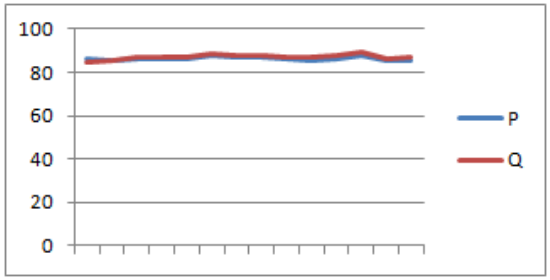

Figure 4.43: Products, 2 requests, period 1.

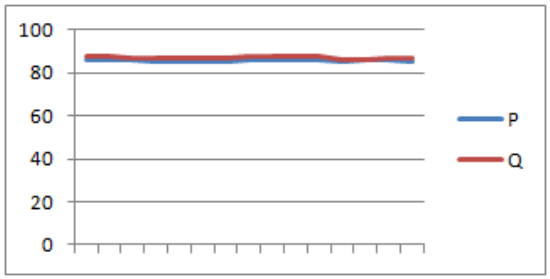

Figure 4.44: Products, 2 requests, period 2.

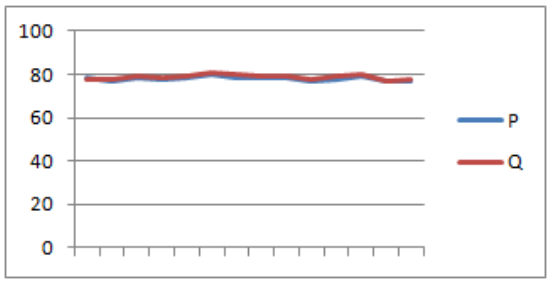

Figure 4.45: Products, 3 requests, period 1.

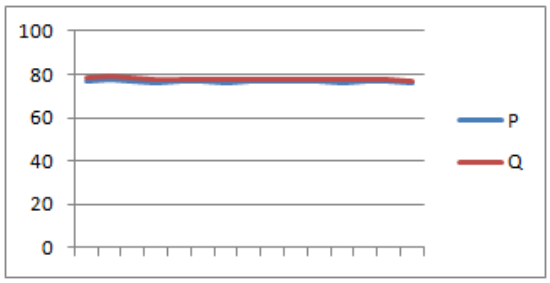

Figure 4.46: Products, 3 requests, period 2.

5 Conclusions and Future Work

This chapter summarizes the thesis by answering the research questions. Possible future work is also suggested.

5.1 Conclusion

Throughout this master thesis, we have developed a recommender system that uses topic models to predict users intent with implicit feedback only. The topic models were created using Markov decision processes on web session data, which in turn were translated to product recommendations. The results show that our approaches was able to accurately identify sessions topics. In most cases the topic of a session could successfully be translated to product recommendations.

The research questions defined in the beginning of this thesis were:
Q1. *Given a set of items visited by a user, is it possible to detect the user's intent?*

Q2. *Can topics created from user sessions be used to generate high precision recommendations?*

Q3. *Can a web session based approach with topic models be used to create a recommender system that handles cold start situations?*

Regarding Q1, the results show a positive answer to the possibility of using visited items to detect user intent. The topic accuracy is around 80% in general regardless the length of the sequence that is used to predict.

Regarding Q2, the results clearly show that it is possible to translate topics to product recommendations. When considering sequences of two requests, the average precision is above 85%, and for three requests it is just below 80%. Compared to the thesis by Lundgren and Lindberg (2014), our results are better in general. But it is difficult to give a decisive answer, since there are almost two years in between the data sets. The best way to measure the difference between our approaches would be to use A/B-testing on actual users.

Regarding Q3, our initial hypothesis about the success of countering the cold start problem had two reasons. First, the use of topic models makes it possible to recommend products to new users, because they reflect general user behavior. Second, the use of attributes to describe the topics can capture all products regardless if it has been viewed or bought

previously. Our solution in the end fits the first reasoning. However, our solution does not fit the second reasoning, even though the topic catches the new item, because during the translation, the recommended products were based on the most popular products that fits the topic.

5.2 Future work

With the presented results, this work has showed that using MDPs with web session data can provide adequate recommendations in terms of prediction accuracy. However, future work can be done on top of this concept. In this section, the future work is divided into algorithm modifications, additional extensions and further evaluation.

5.2.1 Algorithm modification

Selection on attributes

We have selected four attributes for products, which are color, gender, size and section. We argued in our work that the context is implied when using attributes for describing the session topic. However, more attributes can be chosen to further specify the topic, which can probably increase the prediction accuracy. Work can be done on attribute extraction from product databases. A thorough experiment can be done to determine which attributes are relevant for improving the topic accuracy.

State Size

In this work, we have defined a state in the MDP as one single attribute value, two attribute values and three attribute values. Through observation on the results, we found that the state size is important to the prediction accuracy, which is not surprising since the longer the state, the better it captures the entire web session. Additional experiments can be done on determining the perfect state size for creating accurate topic models. Also, a computational problem arises when the number of possible values for attributes increase, since the state size in the MDP grows quickly, because of combinatorics. This can result in a very memory heavy system.

Topic translation

Since our main focus of the thesis was to extract topic models which captures users intent, little effort has been done on translating the topic to recommended products. The current solution is to conduct a database search on the products that matches the topic, and choose the most popular ones. In most cases, there are more products than our designed recommendation space. Our current solution is sufficient enough according to our presented results. However, additional work can be done on finding other alternatives such as using ratings.

5.2.2 Additional extension

Novelty and diversity

Our solution nature limited us to recommend the products within the topic. Our assumption is that customers have a clear goal on what they want to purchase. But in real life, there are cases that people like to do window shopping, and purchase things that they do not know they would like to buy beforehand. A suggestion is to combine our solution with another recommendation algorithm which provides more diversity and novelty to recommended products.

Attributes Normalization

Effort was spent on pre-processing genders and sections in our work. We used the size and colors directly from the product database. This resulted with 315 sizes and 30 colors. In real life, it is usually hard to choose between a 36 size and 38 size, and color black and color dark brown do not matter much to some individuals. We suggest that further processing on the datasets can be done to improve the recommender system. For example, color black and color dark brown can be normalized to dark colors, etc.

5.2.3 Further evaluation

Different datasets

Our recommender solution was specifically tailored to the fashion e-commerce. We have only used datasets from Lindex, which is not public. Using publicly available datasets could certainly be benefit for the confidence of our approach.

AB testing

We have compared our results with the one presented by Lundgren and Lindberg (2014). However, the comparison is not made by confidence, since we were using different datasets within different time periods. The solution to make two approaches comparable is to preferably conduct an A/B-testing with both approaches and measure the number of actual purchases.

Bibliography

Adomavicius, Gediminas, Ramesh Sankaranarayanan, et al. (2005). "Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach". In: *ACM Trans. Inf. Syst.* 23.1, pp. 103–145. ISSN: 1046-8188. DOI: 10.1145/1055709.1055714.

Adomavicius, Gediminas and Alexander Tuzhilin (2005). "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions". In: *IEEE Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99.

Aksel, Fatih and Aysenur Birtürk (2010). "Enhancing Accuracy of Hybrid Recommender Systems through Adapting the Domain Trends". In: *Workshop on the Practical Use of Recommender Systems, Algorithms and Technologies (PRSAT 2010)*, p. 11.

Alam, Shafiq et al. (2013). "Analysis of Web Usage Data for Clustering Based Recommender System". In: *Trends in Practical Applications of Agents and Multiagent Systems*. Ed. by Javier Bajo Pérez et al. Advances in Intelligent Systems and Computing 221. Springer International Publishing, pp. 171–179. ISBN: 978-3-319-00562-1, 978-3-319-00563-8. URL: http://link.springer.com/chapter/10.1007/978-3-319-00563-8_21.

Baltrunas, Linas and Xavier Amatriain (2009). "Towards time-dependant recommendation based on implicit feedback". In: *Workshop on context-aware recommender systems (CARS'09)*.

Braunhofer, Matthias (2014). "Hybridisation Techniques for Cold-starting Context-aware Recommender Systems". In: *Proceedings of the 8th ACM Conference on Recommender Systems*. RecSys '14. New York, NY, USA: ACM, pp. 405–408. ISBN: 978-1-4503-2668-1. DOI: 10.1145/2645710.2653360.

Campos Soto, Pedro G. (2011). "Temporal Models in Recommender Systems: An Exploratory Study on Different Evaluation Dimensions". PhD thesis. Universidad Autónoma de Madrid.

Cho, Yoon Ho, Jae Kyeong Kim, and Soung Hie Kim (2002). "A personalized recommender system based on web usage mining and decision tree induction". In: *Expert Systems with Applications* 23.3, pp. 329–342. ISSN: 0957-4174. DOI: 10.1016/S0957-4174(02)00052-0.

D.A., Adeniyi, Wei Z., and Yongquan Y. (2014). "Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method". In: *Applied Computing and Informatics*. ISSN: 22108327. DOI: 10.1016/j.aci.2014.10.001.

Desarkar, Maunendra Sankar, Sudeshna Sarkar, and Pabitra Mitra (2010). "Aggregating Preference Graphs for Collaborative Rating Prediction". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. New York, NY, USA: ACM, pp. 21–28. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864716.

Domingues, Marcos A, Alípio Mário Jorge, and Carlos Soares (2011). "Using contextual information as virtual items on top-n recommender systems". In: *arXiv preprint arXiv:1111.2948*.

Ghazanfar, Mustansar and Adam Prugel-Bennett (2010). "An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering". In:

Gorgoglione, Michele, Umberto Panniello, and Alexander Tuzhilin (2011). "The Effect of Context-aware Recommendations on Customer Purchasing Behavior and Trust". In: *Proceedings of the Fifth ACM Conference on Recommender Systems*. RecSys '11. New York, NY, USA: ACM, pp. 85–92. ISBN: 978-1-4503-0683-6. DOI: 10.1145/2043932.2043951.

Gunawardana, Asela and Christopher Meek (2009). "A Unified Approach to Building Hybrid Recommender Systems". In: *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. New York, NY, USA: ACM, pp. 117–124. ISBN: 978-1-60558-435-5. DOI: 10.1145/1639714.1639735.

Guy, Ido et al. (2010). "Will Recommenders Kill Search?: Recommender Systems - an Industry Perspective". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. New York, NY, USA: ACM, pp. 7–12. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864713.

Hariri, Negar, Bamshad Mobasher, and Robin Burke (2012). "Context-aware Music Recommendation Based on Latenttopic Sequential Patterns". In: *Proceedings of the Sixth ACM Conference on Recommender Systems*. RecSys '12. New York, NY, USA: ACM, pp. 131–138. ISBN: 978-1-4503-1270-7. DOI: 10.1145/2365952.2365979.

Herlocker, Jonathan L. et al. (2004). "Evaluating Collaborative Filtering Recommender Systems". In: *ACM Trans. Inf. Syst.* 22.1, pp. 5–53. ISSN: 1046-8188. DOI: 10.1145/963770.963772. URL: http://doi.acm.org/10.1145/963770.963772.

Jambor, Tamas and Jun Wang (2010). "Optimizing Multiple Objectives in Collaborative Filtering". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. New York, NY, USA: ACM, pp. 55–62. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864723.

Khoshneshin, Mohammad and W. Nick Street (2010). "Collaborative Filtering via Euclidean Embedding". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. New York, NY, USA: ACM, pp. 87–94. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864728.

Linden, Greg, Brent Smith, and Jeremy York (2003). "Amazon.com recommendations: item-to-item collaborative filtering". In: *IEEE Internet Computing* 7.1, pp. 76–80. ISSN: 1089-7801. DOI: 10.1109/MIC.2003.1167344.

Liu, Nathan N. et al. (2010). "Online Evolutionary Collaborative Filtering". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. New York, NY, USA: ACM, pp. 95–102. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864729.

Lu, Zhengdong, Deepak Agarwal, and Inderjit S. Dhillon (2009). "A Spatio-temporal Approach to Collaborative Filtering". In: *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. New York, NY, USA: ACM, pp. 13–20. ISBN: 978-1-60558-435-5. DOI: 10.1145/1639714.1639719.

Lundgren, Alexander and Linus Lindberg (2014). "Constructing & Evaluating Context-Aware Recommender System in a case study with webshop carts and AB-testing". Master Thesis. Chalmers University of Technology.

Otterlo, Martijn van and Marco Wiering (2012). "Reinforcement Learning and Markov Decision Processes". In: *Reinforcement Learning*. Ed. by Marco Wiering and Martijn van Otterlo. Adaptation, Learning, and Optimization 12. Springer Berlin Heidelberg, pp. 3–42. ISBN: 978-3-642-27644-6, 978-3-642-27645-3. URL: http://link.springer.com/chapter/10.1007/978-3-642-27645-3_1 (visited on 03/23/2015).

Rendle, Steffen, Christoph Freudenthaler, and Lars Schmidt-Thieme (2010). "Factorizing Personalized Markov Chains for Next-basket Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. New York, NY, USA: ACM, pp. 811–820. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772773.

Ronen, Royi et al. (2013). "Selecting Content-based Features for Collaborative Filtering Recommenders". In: *Proceedings of the 7th ACM Conference on Recommender Systems*. RecSys '13. New York, NY, USA: ACM, pp. 407–410. ISBN: 978-1-4503-2409-0. DOI: 10.1145/2507157.2507203.

Said, Alan and Alejandro Bellogín (2014). "Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks". In: *Proceedings of the 8th ACM Conference on Recommender Systems*. RecSys '14. New York, NY, USA: ACM, pp. 129–136. ISBN: 978-1-4503-2668-1. DOI: 10.1145/2645710.2645746.

Schafer, J Ben, Joseph Konstan, and John Riedl (1999). "Recommender systems in e-commerce". In: *Proceedings of the 1st ACM conference on Electronic commerce*. ACM, pp. 158–166.

Tavakol, Maryam and Ulf Brefeld (2014). "Factored MDPs for Detecting Topics of User Sessions". In: *Proceedings of the 8th ACM Conference on Recommender Systems*. RecSys '14. New York, NY, USA: ACM, pp. 33–40. ISBN: 978-1-4503-2668-1. DOI: 10.1145/2645710.2645739.

White, Ryen W., Joemon M. Jose, and I. Ruthven (2001). "Comparing explicit and implicit feedback techniques for Web retrieval: TREC-10 interactive track report". In: *Proceedings of the Tenth Text REtrieval Conference (TREC 2001*.

Yi, Xing et al. (2014). "Beyond Clicks: Dwell Time for Personalization". In: *Proceedings of the 8th ACM Conference on Recommender Systems*. RecSys '14. New York, NY, USA: ACM, pp. 113–120. ISBN: 978-1-4503-2668-1. DOI: 10.1145/2645710.2645724.