Identification and Utilization of Contextual Features Using Post-Filtering in an E-Commerce Context-Aware Recommender System

Stian Lohna

Master's Thesis
Master of Science in Media Technology
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2010

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Identification and Utilization of Contextual Features Using Post-Filtering in an E-Commerce Context-Aware Recommender System

Stian Lohna

1st July 2010

Abstract

Recommender systems are systems that provide recommendations to a user based on information gathered about that user or by finding other similar users. Only a few years ago, these recommender systems were solely dependent on either explicit information given by the user or implicitly gathered information such as user patterns- and actions. During these last few years we have seen a slightly increased interest incorporating context within recommender systems in order to provide better recommendations. There have been however, little or no contribution to find those contextual features that are useful and relevant to recommender systems that operates in an e-commerce setting. By exploiting more of the information that lies in the users' context, it's possible to generate more personalized recommendations related to the context. For instance, if a person at work visits an online store, the person might prefer getting work-related recommendations compared to personal-related. Applying contextual features into recommender systems could convert window-shoppers into buyers, increase cross-sell and customer satisfaction and ultimately generate more revenues to the company. In addition, the consequences of using the proposed contextual features in a fine-grained collection compared to a coarse-grained one will be studied and analyzed.

To start on the task of finding relevant contextual features and how they can affect the final recommendation outcome, a survey was conducted in companionship with one of Norway's biggest auction-based online stores, where over 35000 customers participated.

Preface

This Master thesis is the final work done at the Master in Media Technology study in the Faculty of Computer Science and Media Technology at Gjøvik University College. The topic was found during the first semester and was chosen because of high interest in electronic publishing and web technologies.
I want to thank my supervisor Rune Hjelsvold for his encouragement and helpful guidance during the last semester when I was writing this thesis.
I would also like to thank the great people from Netthandelen.no AS for their collaboration with the customer survey. I especially want to thank my good friend Espen Doknes for his active participation and for making this opportunity possible. I would also like to thank the company's owner Einar Øgrey Brandsdal that accepted to have competition prizes to encourage the customers to participate in the survey. Erik Jensen and Dag Øyvind Godtfredsen should also be thanked for your constructive meetings. The last person from Netthandelen.no AS I want to thank is Ingvar Orten for working overtime the evening before the survey launch. I hope you all liked the effect the survey provided - the best Wednesday in your company's history measured in income, customer visits and purchases.
And last, I want to thank my family for their support.

Stian Lohna, 1st July 2010

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic covered by the project

In the last decade we have experienced an incredible growth of the Internet and its users. The same has the number of online services like online shops that sell everything between rocks and tires to music and movies, and dedicated communities and social networks have also skyrocketed with seemingly no limitations. We use these services to buy and sell products worldwide with ease, hook up with friends or meet new people, watch movies and video clips from all over the world while at the same time getting the latest news from news sites that matches our diverse interests. All this activity has over the years resulted in an enormously amount of electronically stored information, and inconveniently lead to huge difficulties to find and present relevant personalized information that fit the user. However, all this information content can and should not be judged to be the cause of this problem, in fact, we are instead limited by the quality of the tools we use to search and find the information.

1.2 Keywords

Recommender systems, Context-aware, Contextual features, Collaborative filtering, Multidimensional data, Electronic commerce

1.3 Problem description

One of the biggest challenges in recommender systems is the context of the user. The content and artifacts a user might be interesting in, often depend on a specific situation: The current location, season, weather and activity are just some of the variables that could have a drastic effect on the user's goal. For example, if a female person in her twenties visit a global online travel agency during the winter, which recommendations should be offered? What if male person in his thirties visited the same travel agency at the same time from work, but is located on the other side of the planet where it is summer? This generation of recommender systems in use today typically does not take context into account, and are thus unaware of the user's actual goal.

1.4 Justification, motication and benefits

One of the golden rules in world of business is to know your customer. When a customer enter a physical shop, it takes little information from the person to make the seller understand the customer's needs, wishes and ultimately goals. However, if a person goes online and visit the online shop to the same company, the company will have severe problems to understand the person's goals, and is only able to treat the customer as any other anonymous customer. This may give the person the impression that this company is only providing commodities and services under the impression that one size fits all, and could be the reason why the company lost that sale.

By doing this master thesis, it is hoped to find relevant contextual features that can be used by recommender systems that operates in an e-commerce setting. In addition, how they can affect the final recommendations will also be analyzed. Using the right contextual features in the right circumstances, can contribute to convert window-shoppers into buyers, increase cross-sell and customer satisfaction and ultimately generate more revenues to the company.

1.5 Research questions

1. What are some of the contextual features that could be relevant to context-aware recommender systems that are used in e-commerce online stores that offers a great variety of products?
2. Using a post-filtering approach, how does a coarse-grained versus a fine-grained use of contextual features affect the final recommendation outcome?

2 Choice of methods

The different research questions requires different methodologies in order to be analyzed and evaluated properly.

The first research question that addresses on how to find some of the contextual features that could be relevant to context aware recommender systems that operates in an e-commerce setting, will require a combination of qualitative and quantitative methodology. The qualitative method regards the selection of some initial proposals of contextual features, which will be based on the most cited definition of context. A survey will be developed, hosted at Netthandelen.no and targeted towards their customers where they will among other things tag their previously bought products to the suggested contexts they consider being related in some way. By using a quantitative method, the tags and products will be analyzed to measure the popularity of each proposed contextual feature. In addition, a comparison between the number of tags versus the number of unique products they cover will be performed. This comparison will be performed on raw and filtered data, where the filter is a threshold that acts as a classification tool, based on the average number of tags of each contextual feature.

The second and last research question require a quantitative method in order to measure how the contextual features affect the final recommendation results. This will be accomplished by using the post-filtering approach, which means that the different contextual features are applied after every possible recommendation have been predicted for every customer using the Weighted Slope One algorithm. This algorithm will base its predictions from the data set generated from the survey that was developed, where the customer left a rating for each product describing how satisfactory it was. After post-filtering has been applied, the final recommendations will be analyzed for every customer, which will give the measurements of the exactness and completeness of the possible recommendations to the average customer.

3 Related work

3.1 Recommender Systems methods

A recommender system is a subtype of information filtering technique which offers specialized dynamic content adapted for normally a single user or a group of users. The content contain typically some kind of product or commodity, often referred to as items, like movies, music, books, images, web sites or computer hardware components.
There exists different types of techniques behind recommender systems, and a summarization of some of types that are most often used in recommender systems are listed below:

- ***Content-Based*** A recommendation is given based by estimating the similarity between the current item and the items the user earlier has shown some kind of interests of, like buying, viewing or commenting [2], or by analyzing the item descriptions to identify items that are of particular interest to the user [3]. However, systems that are only based on this setup have the disadvantage of having the user profile as the only source of information, and this type of information is of the kind that doesn't change too often. It's also normal to require the user to enter explicit personal information, like interests and demographic information.
- ***Collaborative filtering*** Recommender systems based on the collaborative filtering approach, predicts user preferences for items by analyzing the relationship between users and items. It's the most social one among the different recommender systems, where the recommendations are based on the feedback from the users. This feedback is often a rating describing the item in some way. The system can then try to predict which items that would be most liked by the users. The collaborative filtering versions are divided into two subtypes, and how the predictions are computed depends on the different types of neighborhoods. One neighborhood revolves around users and the other around items.
  The user-based neighborhood is based on that it's likely that a user belongs in a larger group of similar and like-minded users [4]. Items that are frequently purchased or liked by the various members of the group can be used to form a basis for recommended items. The system can predict a rating value for a item to a user. It tries to guess what the user would have rated the item, and the best guesses becomes the best recommendations. The rating can be computed as the mean of the ratings' values from the similar users. [5]
  The item-based neighborhood exploit the similarity among the items [5]. It is measured using other methods compared to content-based approaches. It isn't the content, like for instance the descriptions of the items that are analyzed, but the historical information that is analyzed to identify relations between the items.[4] The relationships are identified by looking into the set of items that the users have rated, to calculate the similarities between them and a given item.[6] It's then possible to compute predictions by taking for instance a weighted average of the target user's ratings on these similar items.

- ***Hybrid*** A hybrid system is a system that uses and combines different types of techniques, for instance, an combination of Content-Based and Collaborative Filtering techniques. [7] summarizes some examples:
  - *Cluster models* Maintains profiles for clustered users that have similar preferences. Recommended items are based on other users in the same cluster.
  - *Demographic filtering approaches* Recommends items based on demographic information as age, profession, education, nationality and geographical location.
  - *Knowledge-Based* Aggregates the users needs, preferences as well as location to recommend items.

3.2 Context interpretation

Recommender systems comes in various forms, and the type that most closely resembles context-aware recommender systems are the content-based ones.
The semantic web is in many ways relatively new and far from being a common element in the Internet as we know it today. However, we see signs that both public and commercial interest are increasing, and some call it Web 3.0. It is hoped that the incorporation of the semantic web will result in one of the next generation's new recommender systems.[8].
[9] presents a general framework that is for semantically enabled recommender systems that uses ontologies to improve the understanding of the context of the users, and gives examples how to deal with them and decide what to do. For instance, if their contextual sensor, which collects contextual information, finds out that it is a rainy day at the current user's location, then the system tries to avoid recommending outdoors events and concerts in that area. If Christmas holiday is coming up, and the user's last click was on a comedy movie, the framework will try to recommend (comedy) movies that are, in some way, related to Christmas. They are able to know that the recommended movie is a comedy that is related to Christmas, because all their item- and context-related information is incorporated by the system by using an ontology. They mention the example CD *[Garth Brooks - Beyond the Season]* $\rightarrow$ *[suitable for holiday]* $\rightarrow$ *[Christmas]*. This example shows one variation on how the inferring capabilities of semantic ontologies using knowledge representation languages like Web Ontology Language (OWL) can prove to be one kind of tool to recommender systems when it comes to dealing with context.

3.3 The search for contextual features

Before the search for which contextual features that matters in a recommender system that operates within the e-commerce field, it could be valuable to know what context actually means. There isn't no common technical definition of context that seemingly everyone agrees on. But ranked after number of citations, the most popular definition was proposed by Abowd et al.[10]:

> '"*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*"'.

Since we have x different unknown variables taken from this context cloud, we must find those contextual features y, that matters more than others. Abowd et al.[10] presented four

contexts that were stated being the four primary ones. Those four are location, identity, time and activity. Their tasks is to answer questions about who, what, when and where. These are considered to be the first level of contexts. Other context types can be derived from these, creating new context levels.

Adomavicius et al.[11] uses a contextual approach on recommender system that recommends movies, using the contextual features like:

- Time: Represents when the movie can or wants to be seen, time of day, day of week, month and year
- Place: Where the movie can be seen. For instance, in movie theaters, or home.
- Companion: If the person wants to see the movie alone or with somebody else, like friends, girlfriend/boyfriend, family, co-workers etc.

Hussein et al.[9] implements contextual features in a framework that suites a virtual shopping and leisure portal that involves DVDs, CDs, sport events and concerts. In their framework they focused on the users' location and surrounding areas at a given time and the weather situation at these locations.

In October 2008, Adomavicius and Tuzhillin stated that it exists a need to know more about what kind of contextual features that matters in different applications.[12] This statement seems to be still valid. No previous research about contextual features for context-aware recommender systems have been found prior or during the work of this thesis. Research about context-aware recommender system seem to focus on the recommendation performance and how to improve it. In the performance tests, a few, often simple contextual features are picked as examples to prove the different hypothesises. Frequently used contextual features used in the measurements are season and holiday, used for instance in [13] and [14].

3.4 Choosing an Collaborative Filtering algorithm

3.4.1 The search

There exists a great variety of different collaborative filterings algorithms, and [15] presents an overview over some of the different algorithms and areas. [15] states that the most well-known collaborative filtering algorithms are the nearest neighbor algorithms, but also probabilistic algorithms that involves Bayesian-network models are noteworthy.
However, [16] introduced a family of algorithms for collaborative filtering called *Slope One*. It is a purely item-based technique, the same kind that Amazon.com use [17], and the three proposed slope one schemes have predictors of the form $f(x) = x + b$, which pre-compute the average difference between the ratings of one item and another for users who rated both. [16] states their aim was to provide robust Collaborative Filtering schemes that are:

- Easy to implement and maintain, the algorithms should be easy to implement and test.
- Updateable on the fly: the addition of a new rating should change all predictions instanta-

User	Item 1	Item 2	Item 3
Jack	5	3	2
Bill	3	4	Didn't rate it
Chris	Didn't rate it	2	5

Table 1: Slope One scheme with three users and three items

neously.

- Expect little from first visitors: A user with few ratings should receive valid recommendations.
- Accurate within reason: the schemes should be competitive with the most accurate schemes, but a minor gain in accuracy is not always worth a major sacrifice in simplicity or scalability.

A simple demonstration of how the Slope One algorithm works is demonstrated in figure 1. Another example with three items is demonstrated in table 1.

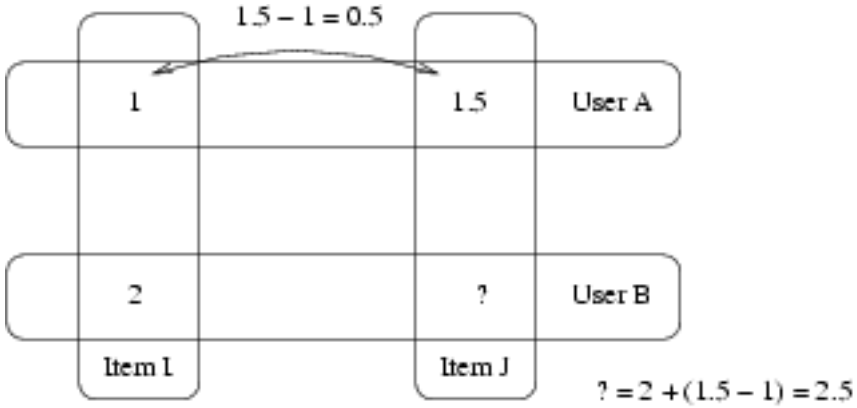


Figure 1: Basis of Slope One schemes: User A's ratings of two items and user B's rating of a common item are used to predict user B's unknown rating.

In table 1, the average difference between item 2 and 1 is $(2+(-1))/2 = 0.5$, thus on average, item 1 is rated above item 2 by 0.5. Also the average difference between item 3 and 1 is 3. If we try to predict the rating of Chris for item 1 using his rating for item 2, we get $2+0.5 = 2.5$. Using the same approach we can predict his rating for item 3 by calculating $5+3 = 8$.
[15] presents the the Slope One algorithm family as three different schemes, the SLOPE ONE Scheme, the WEIGHTED SLOPE ONE Scheme and the BI-POLAR SLOPE ONE. They are mentioned in order of complexity, where the Bi-Polar Slope One scheme is the most complex one.

3.4.2 The Weighted Slope One Scheme

In the prototype, the Weighted Slope One scheme was chosen for implementation. The advantage with this version compared to Slope One Scheme is that it considers the number of ratings observed. This could be very important when one popular item has many ratings, while another item has few, but is rated generally high. The weighting mechanism even out these differences accordingly. Given a training set χ, and any two items i and j with ratings u_j and u_i, respectively

in some user rating u (annotated as $u \in S_{j,i}(\chi)$), we can consider the average deviation of item i with respect to item j as:

$$dev_{j,i} = \sum_{u \in S_{j,i}(\chi)} \frac{u_j - u_i}{card(S_{j,i}(\chi))}$$

The symmetric matrix defined by $dev_{j,i}$ can be computed once and updated quickly when new data is entered.
We can now define the Weighted Slope One prediction as the following weighted average

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u)-\{j\}} (dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}}$$

where $c_{j,i} = card(S_{j,i}(\chi))$, and card($S_{j,i}iscardinality$.

3.5 Implementation of contextual information in Recommender Systems

Three approaches exists to create a context-aware recommender system. Those are pre-filtering, post-filtering and contextual modeling. These are visualized in figure 2/

Pre-filtering incorporates the context prior the computation of the predictions. Those items that were remaining after the pre-filtering, can be used to compute predictions, using for instance an traditional item-based approach. When the predictions are done, the items can be recommended to the user.

Post-filtering incorporates the context after the predictions have been calculated. The predictions have been based on all the original items, and traditional techniques can also be used to calculate them. Post-filtering are applied to the items of these predictions, and the remaining items are the recommendations.

Contextual modeling integrates the process into one combined model. It's a tight coupled approach compared to pre- and post-filtering. The contextual features are evaluated directly inside the recommendation-generating algorithms [14].

3.6 Measuring a Context-Aware Recommender System's performance

Precision and recall are common measurements in information retrieval and statistical classification tasks. This also applies to F-score (also known as F_1 score and F-measure) which is a measure of test's accuracy. It considers both precision p and the recall r of the test to compute the score which is the harmonic mean of precision and recall. This is one of the possible metrics used in recommender systems.[18][19]. These metrics have also been used in the initial and recent works [20] to measure the performance of context-aware recommender systems, where the goal is to compare the use of pre-filtering versus post-filtering of contextual information, use of folksonomy in combination with collaborative filtering recommendation [21], and in tag-aware recommender systems [22]. In other words, when a new variable is introduced that affects the predictions made by recommender systems by using some kind of classification, then these metrics are suited for these occasions.
Precision is the number of relevant documents retrieved divided by the number of retrieved documents, while Recall is defined as the ratio of the number of relevant documents that are

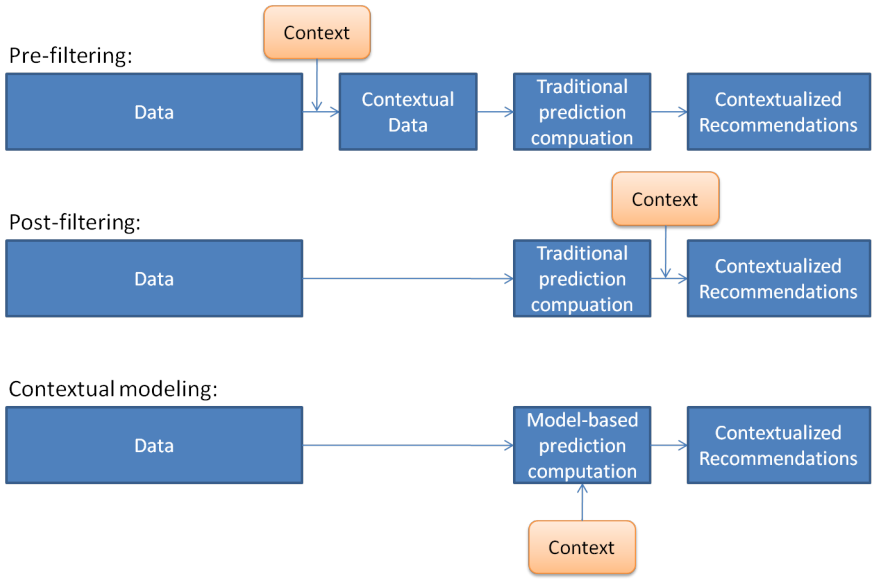


Figure 2: The different approaches that are used to get contextualized recommendations.

	Relevant	Nonrelevant
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

Figure 3: Shows how to find the different values for true positive, false positive and false negative. Image adapted from [1].

retrieved to the total number of relevant documents. [23][1]. Below is the formulas for each of them shown in the context of classification:

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

$$\text{Fscore} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The values for TP, FP and FN are found using the matrix in figure 3.

4 Contextual features

Since no related work regarding relevant contextual features were found prior or during this thesis, some initial contextual features had to be selected and proposed from scratch. The most cited definition of context [10] was used as a guideline, and the context types that Abowd et al. claimed to be the main primary ones were used as the foundation. Those primary types of context are location, identity, time and activity.

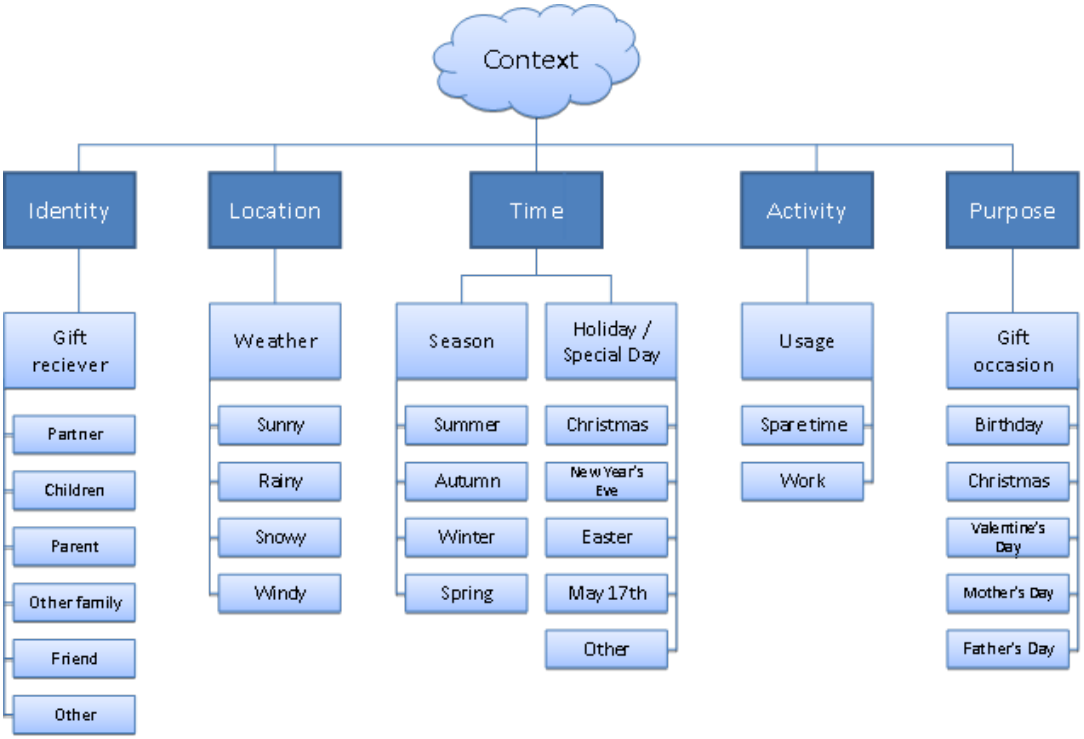
However, in the setting of recommender systems used in online stores, there was something that those primary types didn't cover. When customers visit an online store, they most likely have a reason for coming, a purpose. This is a part of the customers' context. A context-aware recommender system's goal is not only to sense the different contexts, but also try to infer some of the customers goals and needs. So a new primary type for context used in context-aware recommender systems is proposed and used during this thesis, and that is *intention*. Those five primary types now look at *who's*, *where's*, *when's*, *what's* and *why's*.
The proposed contextual features were put in a class hierarchy. The context class is something abstract and is the parent of the hierarchy. The class hierarchy is shown in figure 4.

The primary type *identity*, contains the main contextual feature *gift receiver*. Used as a default in the survey, identity is the participating customer. However, if the customer had purchased a product as a gift, then identity is used to describe the gift receiver.

Weather was the proposed contextual feature under location. Weather was selected because it's something that situates the customers and can influence their needs. If it's sunny, some customers might realize that they need a pair of sunglasses and will thus visit an online shop to find some. It's possible to identify users location with the accuracy down to city level or better by using geo targeting. It exists several services that offers this, and will probably gain popularity with W3C's Geolocation API which is included in the upcoming HTML5. The extracted location could then be used online weather services, which returns the weather for that particular moment or weather forecasts. And finally it's possible to recommend products related to the weather.

Time was divided into to main contextual features; *season* and *special day / holiday*. Season is something just like weather, it changes. Weather is also often considered as being related to season. Different seasons affects customers in different ways. If it's winter, an appropriate recommendation could be a vacation somewhere with warmer climate, or a pair of skis if the weather forecast say that snow is coming in addition. Recommendations for special day / holiday can be a music album with Christmas songs, safety goggles for fireworks, easter decorations or even the Norwegian flag and national customes[1].
The intended *usage* was chosen to belong to the activity type. Different types of recommendations can be given dependent of time of day, or by looking at the identity, if the customer is a personal

[1] May 17 is the Norwegian Constitution Day and is a national holiday observed each year.


Context
Identity
Location
Time
Activity
Purpose
Gift reciever
Partner
Children
Parent
Other family
Friend
Other
Weather
Sunny
Rainy
Snowy
Windy
Season
Summer
Autumn
Winter
Spring
Holiday / Special Day
Christmas
New Year's Eve
Easter
May 17th
Other
Usage
Sparetime
Work
Gift occasion
Birthday
Christmas
Valentine's Day
Mother's Day
Father's Day

account or a corporate account. If it's typical work hours and / or the customer is logged in using a corporate account, then it would feel natural to get work related products recommended.
And finally, the proposed primary type *purpose*, have the main contextual feature *gift occasion* as its child. This is used in this thesis to directly to describe the occasion for the gift. But this is just an example. Purpose is something more essential than this, it can be used to add additional information about all of the other contextual features. One might ask: *Why* do this customer come to this online shop? Because the customer needs to buy a gift. And *why* do the customer need to buy a gift? Because it's someones birthday. *Why* do you buy a pair of sunglasses as a gift? Because the person lost the old ones. *Why* do you buy a pair of skis? Because it will snow during the upcoming weekend.

These proposed contextual features will be used in the survey for data gathering, and later in contextualized post-filtering of recommendations to measure performance and evaluate their final impact of the recommendation outcome.

5 Survey prototype

5.1 Justification of prototype development

A prototype was developed during the project because of the many advantages it gives. First of all, this was done because it is a cost effective and quick way to get a better overview of the basic design and layout of the survey website. In addition it provides the opportunity to quickly identify which elements that were to be included and their associated functionality. It was also important to provide Netthandelen.no AS with a proof of concept to maintain and increase their interest in the project and thus their commitment to it. This encourages their active participation and results in a higher input and output from both parties.

5.2 Planning and preparations

There are many ways to make a prototype, ranging from simple paper drafts to full scale development. It was early decided that a interactive prototype was preferred, as this yields increased understanding of what the final system looks like. This key point applies to both the developer and the company who reviews the prototype. It does however require more effort from the developer compared to doing some sketches on paper. In order to deal with this seemingly inverse proportional problem, Microsoft Expression Blend 3 with Sketchflow were used. This combination offers an opportunity to create a interactive prototype that don't require too much development time. It resembles the Adobe Flash editor in many ways, but it offered some features that were considered as advantages for being used in this project, and hence, it was chosen for the prototype development. With Expression Blend it is possible to build the output into two different formats; Microsoft Silverlight and as an application based on Windows Presentation Foundation. The former format was chosen as this made it possible to present versions of the prototype remotely with ease by using a web browser if needed. It was also the most reasonable alternative since the final version of the survey would be a website, so by having the prototype in a browser it was created a stronger resemblance between those two.
It was also agreed upon that each customer was supposed to give information about maximum ten products. This limit was chosen as it seemed to be well balanced between what was fair and reasonable to expect from the customer, and what was considered enough data to be able to generate valid predictions.
A collaborative filtering algorithm is based on some kind of input from users, and will not work without it. Based on this fact, a rating describing the satisfaction had to be obligatory, or else the product entry would be considered useless. Everything else however was not obligatory, but it was hoped that the customers would tag relevant information about the product regardless of this. It was also known that not all of the products in the database are necessarily related to any of the given contexts. This decreases the chance for a product to be falsely tagged, but it does also increase the chance that customers skip the contextual tagging.

The user design is a very important aspect of a survey. A badly designed survey could lead to that fewer customers would complete it. To avoid this, is was planned to keep the survey as simple as possible. Questions should be clear and have enough space between each other to avoid giving the impression as being too crammed. The layout should be neat and minimalistic to avoid distractions, but at the same time not being boring. It was a goal trying to keep the height of each page of the survey little enough in order to avoid too much scrolling. And finally it was considered important to motivate the subject, and a solution to this was to inform the customer about how many products that were remaining.

5.3 Development

The development of the prototype went quite smoothly since some ground rules were set from the beginning. These ground rules acted like guide lines and provided the foundation of the prototype.
Microsoft Expression Blend 3 offers three ways to develop interactions. One way is purely based on using the graphical user interface, which may be the fastest way, but also the most limited. The second one is purely based on programming which is quite the opposite. The last one is a combination between these two and offers the best from both sides. As a result of this, this method was chosen.
A simple layout was designed and a random product was chosen to be the demonstrative product, as shown in figure 5. Programming was needed to create the color transitions that would happen when the mouse cursor was hovering above a star. More programming was also needed to demonstrate the functionality when a the product was tagged as being bought to somebody else other than the customer. In this case, a box would appear and provide the customer to answer additional contextual information about it, like shown in figure 5. Since it was bought to somebody else, it doesn't necessarily mean that it was a gift. This is why an additional question was needed: "Was it a gift from you?". Right below came the follow-up question: "If yes; what was the occasion?". These questions covered to who the product was bought to, if it was a gift or not, and finally the occasion if it was a gift.
The next contextual question regarded intention and activity. It was about the usage of the product, if it was supposed to be used during the customer's personal time, or at work. Both answers at the same was considered valid. After this, the next question was time related. It asked about if the product was related to a certain season. Alternatives were "No", "Summer", "Autumn", "Winter" and "Spring". Then the question about location came up, if the product was related to a certain type of weather. Alternatives were "No", "Sun", "Rain", "Snow" and "Wind". The last question was about time, if the product was related to a special day, for instance a holiday. The customer could choose between "No", "Christmas", "New Year's Eve", "Eastern", the Norwegian holiday"May 17th" and "Others / not given".

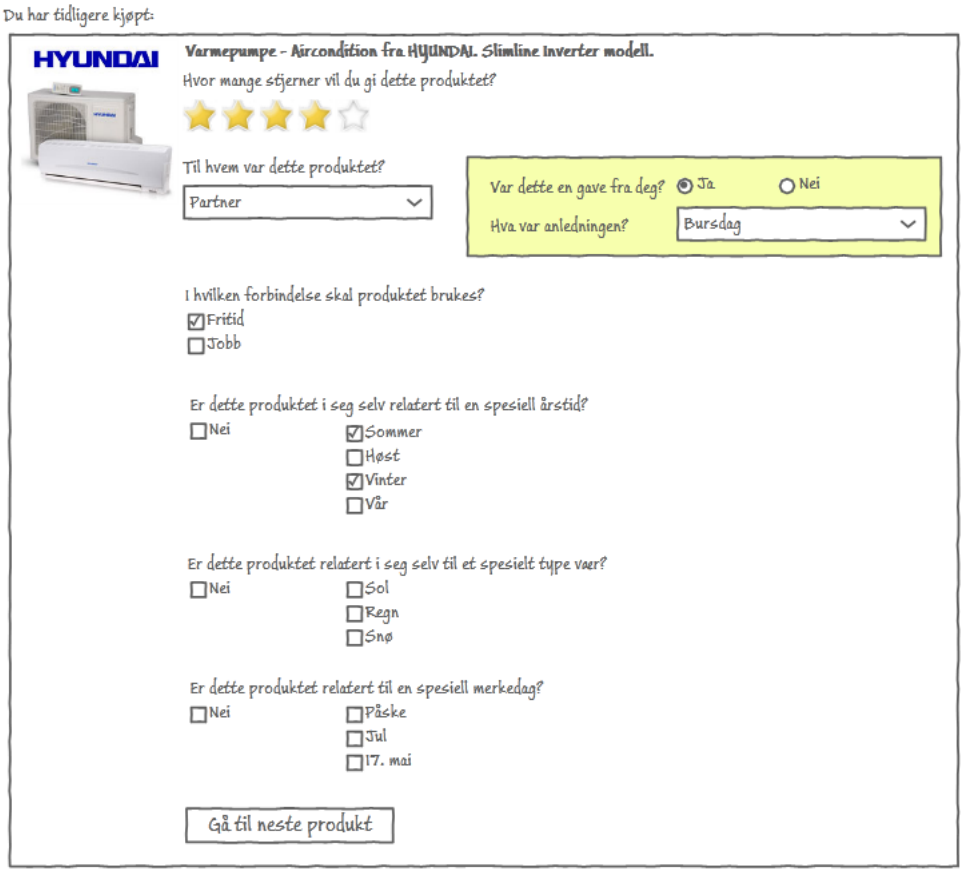


Figure 5: The prototype layout in Silverlight, developed in Microsoft Expression Blend 3 with Sketchflow. This particular product, a heat pump, is a gift to the customer's partner. The occasion was the partner's birthday. The heat pump is also tagged as being specially related to summer and winter.

6 Survey development

6.1 Planning and preparations

When the prototype was considered finished, an office was made available at Netthandelen.no AS, where the development of the survey system was undertaken. The survey was integrated into Netthandelens.no AS's existing system, so a set of conventions and guide lines had to be followed. The tools that were used for the development was Visual Studio 2010 and MS SQL Server 2008. The survey was implemented using ASP.NET C# 3.5. Since the survey directly used customer data, a database model was supplied. The model was used to create the database and tables with the necessary data that would be used locally during the development. Some data was provided, like product information and some non-sensitive customer information. Customer Id, postal code, gender and birth day were supplied to be used during the development. It wasn't possible to identify any customers from the data provided. The development took place in three stages. The first step was to use the prototype as template and start the development and test and debug during this period, which happened locally. The second step was to deploy the survey that was considered final to the test environment the Netthandelen.no AS used. The deployment of the survey to the test environment meant a feature freeze. By their convention, only debugging was allowed at this period of time. The final step was the deployment of the survey to the production servers, and at this step was considered as the final code freeze with limited test possibilities before the official launch. If it turned out that something didn't work at all or as expected, the launch would have been delayed at least one week, since new builds were deployed on the production server every Monday.

6.2 Reputation addon

In addition the the product data that were collected, some customer information was also important to gather to get a better understanding about those who had completed the survey. Netthandelen.no AS wanted to take advantage of this opportunity when reaching to its customers on this level. In addition to just letting the customer give feedback about the products, they also wanted to have the customers give feedback about the company itself. This meant that another module to the survey had to be implemented, which was named the reputation survey. This part would come right after the customer was finished about the product survey. This was made on purpose, so it wouldn't affect the product survey in any way since the customers were unaware about it until they were finished with the product survey. The reputation survey was divided into two parts. In the first part, the customer had to enter or update information about her gender and her birth date. In addition, it was a list that represented different types of interests. The user could choose to select any that were interested. In the second and last part of the reputation survey, five questions about Netthandelen.no AS were added. The customers had to answer by using the star system, just like they did with rating the products. The same concept was used

since it is a simple and uniform way to measure and the customer already knew how to use it.

6.3 Survey walkthrough

All pages had similar design and theme, and figures of the welcome page, product survey and reputation survey are in the appendix, figures 41, 42, 43 respectively. In addition, all the pages had a similar validation test.

Default.aspx: This page was the starting point of the survey. It was the page the customer landed on after clicking on the link in the email that was sent to them. From the customer's view, it only act like a introduction page. But in reality, it was more to it than that. In addition to validate the customer for being an actual customer, it also logged information about the time the customer first visited the survey. In also logged how many times the customer had visited the survey and if she already was finished with it.

It was important to make the page feel light-weight and seem overcoming in order to not scare the customer away already on this page.

ProductSurvey.aspx: When the customer clicked the "Start Survey" button on Default.aspx, she was redirected here. This is the main part of the survey. Exactly the same layout that was used in the prototype was applied to the web version. JavaScript was used for the color transition of the stars when the mouse cursor was hovering above one them them. It acted in the same way as the prototype; if for instance the cursor is above the third star, then the first and second star is also highlighted. When the user click on a star, then the stars stay that way. To improve consistency and clarity, a rating description was included just below the stars.

Small changes to some of the questions were made, after some discussion with Netthandelen.no AS on how to make them as clear as possible. In addition, the customer support staff reviewed the survey, which resulted to some additional small changes of the text information and questions in general. It was necessary to analyze this throughly with different people since it is common to interpret text and questions differently. Icons represented as question marks were also included with the questions about season, weather and special days, and gave an example of each of those when the mouse cursor was hovering above them. This was another initiative to reduce the chance that the customer didn't understand the question correctly. JavaScript and CSS were used in combination to show and hide the box that depended on who the product was bought to. This box contained the form inputs to give additional context information about if the product was a gift or not, and what the occasion was. In general, JavaScript was used extensively in order to create necessary interactivity, and also to provide the most seamlessly flow as possible.
As mentioned earlier, the only thing that was obligatory was to provide a rating to each product. But it was of course hoped that the customer did provide additional relevant information.
If a customer didn't rate the product, an error message get visible right above the rating stars to notify about this. To provide progress information to the customer, just right to the submit button it would say how many products that were remaining. The maximum limit

1. All in all, how satisfied are you with Netthandelen.no?
2. How were your experiences with the products compared to the expectations to them?
3. If you have been in contact with customer support, how would you rate this experience?
4. If you receive newsletters from us, how useful or relevant is the information in the e-mail?

Table 2: The questions used in the reputation survey

was set to ten, just like what was planned before the development started.
When at least a rating has been provided, two things could happen. If the customer have remaining unrated products, a new page will load with the next unrated product. If the customer don't have any remaining products, then the customer will be redirected to the reputation survey.

CustomerSurvey.aspx: This was the additional survey that Netthandelen.no AS wanted to include. On the top of the page were the customer information forms, like gender and birth, that was obligatory to submit. The interest section was also placed at the top, but wasn't obligatory. If the customer already had entered personal information about gender, birth date or interests, then this would be pre-selected at page load. The customer could change them even though they were pre-selected.

The reputation questions are shown in table 2, where the customer would rate the different things that were asked about in the same manner that they did with the products.

When the customer finally clicked the submit button, the last page was shown.

Takk.aspx: This page was the end screen and thanked the user for the participation. Behind the scenes, it marked the customer as having both of the surveys complete.

6.4 Optimalization

At the time of the execution of the survey, Netthandelen.no AS had over 900 000 registered customers. Their main web page was had everyday between 30 000 and 60 000 unique visitors. The e-mail about the survey was queued and sent to the customers in a relatively short time, meaning many customers could visit the survey at the same time. It was of utmost importance that the survey wouldn't become slow or even worse; shut down. A slow website leads to decreased user experience, thus increasing the chance that the customer doesn't complete the survey. Two things were possible to optimize, and that were the graphical content of the survey and the system itself. A typical survey isn't particular heavy in terms of content, pictures or images used for layout, they are rather the opposite, quite lightweight, and this survey was no exception. Applications and particularly the database and how those two things communicate are typically the bottlenecks in systems that suffers from bad performance. Even though Netthandelen.no AS had powerful servers with high capacity, no risk were taken when it came to this. With this in mind and also to do what many consider as best practice, stored procedures in the database were used instead of dynamic SQL. One of the advantages with stored procedures compared to dynamic SQL, is reduced network traffic between the application and the database server. For instance, instead of transmitting a series of complex SQL statements, this can be reduced to one stored

Customer ID	**Customer's personal URL**
Customer # 900000:	Default.aspx?customerID=900000
Customer # 900001:	Default.aspx?customerID=900001

Table 3: Customer's ID used directly in a query string for identification

procedure call. Another advantage is the possibility to encapsulate business logic at the database level if wanted.

6.5 Security

Security was another aspect that had to be dealt with. As little as possible of customer information should be exposed in order to identify the customer, and the risk for abuse should be minimal as possible. Netthandelen.no AS stored information about their customers in their database, where the customer profiles were identified by their customer IDs. There were several possible ways to identify the customers that visited the survey. One solution was that they could log on using their original credentials assigned to Netthandelen.no AS. The other solution was to identify the user by a query string from the URL. The first solution was not particularly considered as a possible choice because of its obvious disadvantages were considered more serious than the advantages. It was the most secure solution, as the customers have to log in manually and no customer information would have been exposed in any way. However it was also the most impractical and slowest solution for the customers. It would have given a decreased user experience, and an increased number of customers would not have chosen to participate in the survey. The other and more efficient alternative was to identify the customers by having some kind of identification code in the query string. The simplest method to achieve this would probably be to include the customer ID in this query string. However there would have been multiple implications by doing this. First and foremost, the customer IDs are generated in an increasing manner. One customer could have the ID of 900000 and another customer could have 900001. Table 3 demonstrates how the this would be like in a query string located in a relative URL.

However this exposes the customer ID and sometimes is this something people want to avoid. In this case it was particular important to use a different approach, because a customer could easily change the URL, changing the *customerID* to something else. This would have loaded the survey for another customer, which first of all would have logged and registered this customer referred in the query string as a participant of the survey, and would also expose up to ten products this customer had previously bought. It's easy to see that this alternative is not acceptable in any way. But the query string was nevertheless the way to go, but some modification had to be done. A new table in the database was created which consisted of two columns; *CustomerID* and *CustomerGUID*. The column CustomerID contained the original customer ID, and associated with this ID, a *Global Unique Identifier* (GUID) was randomly created for each customer. A GUID is a special type of identifier used in software applications to provide a unique reference number [24], and with the latest version of the GUID specification it is considered statistically impossible to create two identical GUID's. Once every CustomerID had a dedicated GUID, the GUID could be used as identification instead of the customer ID in the query string, as demonstrated in table

Customer ID	**Customer's personal URL**
Customer # 900000:	Default.aspx?guid=21EC2020-3AEA-1069-A2DD-08002B30309D
Customer # 900001:	Default.aspx?guid=3F2504E0-4F89-11D3-9A0C-0305E82C3301

Table 4: Customer's GUID used as identification instead of the customer ID

4.

This created URLs that were practically unique to each customer. It does exists numerous tools and methods to create unique codes, and many of them would probably have done the job equally as good in practical terms. However, GUID was used because of its simplicity when coming to the generation in most programming languages. With this solution implemented, nobody could tamper with the URL. In case somebody did, they would have been redirected to another page with an explainable error message.
On all pages there are several validation techniques to make sure that the person who is loading the survey page actually are a customer. In case a validation fails, the customer will be redirected to a page that explains the error. The first common test is to retrieve and parse the GUID from the query string. It has to be supplied or else an user friendly error is shown. The next validation is to check if the GUID has a valid GUID format. The third validation is to check if the supplied GUID is associated with a customer ID. Those three validations are the common ones that are on every page. There are also a few additional validation checks on some of the pages that varies:

ProductContextSurvey.aspx: At each page load, it checks if the customer has completed the product survey. This is to prevent from back button attacks, where the user knowingly or otherwise, tries to resubmit already submitted information (or changes to it) that shouldn't be resubmitted. In this a case, a duplicate rating of the product would have occurred with its corresponding contextual information, which also would have been duplicated. If it detects that the customer is completed with the the product survey then the system transfers the customer to the next survey; the reputation survey.

CustomerSurvey.aspx: This page is also set up that it isn't possible to submit data multiple times. If the customer already has completed this survey, then the system transfer the customer to the final page (Takk.aspx).

Takk.aspx: The final page checks on page load whether the customer has completed both the surveys or not. Both surveys have to be completed to view this page, or else the system will transfer the customer to the start page (Default.aspx).

7 Results

7.1 Survey preparations and response

7.1.1 Measures to increase the participation of the survey

Since a major part of this thesis is based on quantitative research, it was important to get as much data as possible to increase the validity of results and conclusions. To motivate the customers to participate, Netthandelen.no AS agreed to give all customers that completed the survey one exemption of the handling fee during their next purchase. In addition, three customers would win a gift card with a value of 3000 Norwegian kroner (NOK).

7.1.2 Candidate selection

Two criterias had to be fulfilled by the customers to be a valid candidate for the survey. Since the survey presented previously bought products, it was necessary that the customer had at least bought one product. The second criteria was that the products had to be bought by the customer at least 14 days prior the official launch of the survey. This would increase the probability that the product had arrived to the customer and that the customer have got an impression of it.
When this process was complete, a total of 430 000 customers had been selected as valid candidates.

7.1.3 Finding the best e-mail subject

After finding customers that were suitable candidates, the process of finding the best subject line that would be used in the survey e-mail started. Different subject lines can get different click-through rates. 30 000 randomly selected customers were divided into three groups. Each group got assigned one subject line each. The proposed subject lines are below, translated from Norwegian to English:

- Help us to get better!
- Competition: Win gift cards!
- Customer survey

In addition, a small group consisting of 1000 customers was also created and got its own subject line: “We need your help!”. The survey e-mail was first sent to these customers, to see if any problems with the survey occurred. Though it wasn’t the same population size as the other three groups, it was possible to get and indication about the click-through ratio here as well. Since no problems were reported after sending the survey invite to the customers in the small test group, the process of sending e-mails to the other groups started. After two hours the click-through rates were measured, and the best subject line was “Customer survey”. It was however a close tie to “Competition: Win gift cards!”.

Customers participated:	35615
Ratings:	261795
Unique products:	15790
Ratings on product average:	16.56
Ratings on customer average:	7.35

Table 5: Some statistics about the survey response.

7.1.4 Participants

Twelve days after the survey was launched, the data gathered was extracted for analysis. 430 000 e-mails had been sent, however 130 000 e-mails bounced back, indicating delivery errors. Most of the bounces were caused by that the e-mails no longer existed. This means that approximately 300 000 e-mails were successfully sent and delivered. As seen in figure 5, a total of 35615 customers participated in the survey. A histogram of the customers that participated and their age is shown in figure 7. They had rated 261795 products, of which 15790 were unique products. The two first days accounted for 83% of all the customer visits, and indications of this can be seen in figure 6. Even though new customers kept coming to the survey after the twelve first days, it was decided not to gather anymore for the analysis. It did however benefit Netthandelen.no AS as the customers continued to rate, tag products and update some of their personal information.

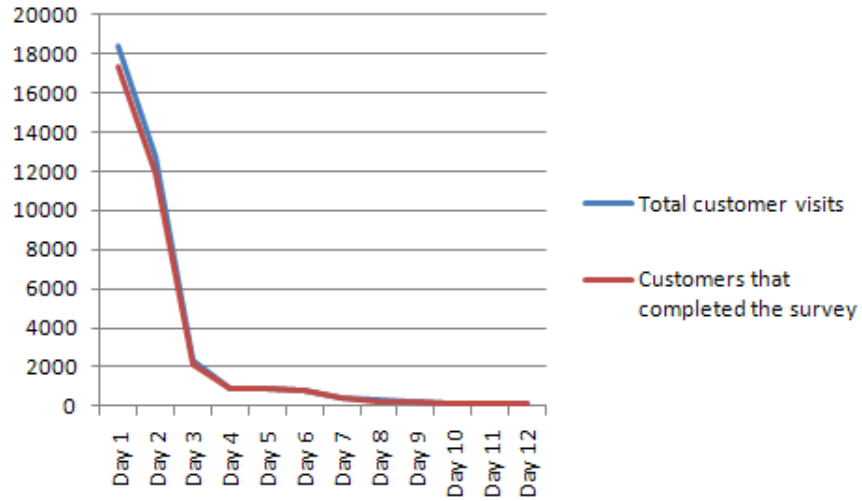


Figure 6: Showing the customer visits and completions of the survey during the twelve days.

7.2 Implementation of the Weighted Slope One algorithm

7.2.1 Precomputing Popularity Differentials

The Slope One algorithms work on the intuitive principle of a "“popularity differential"” between items for users. This allows us to determine how much better one item is liked than another. However, in a real setting on production level, the calculation of this popularity differential in real time with many items, users and ratings would be too costly. As briefly mentioned, the

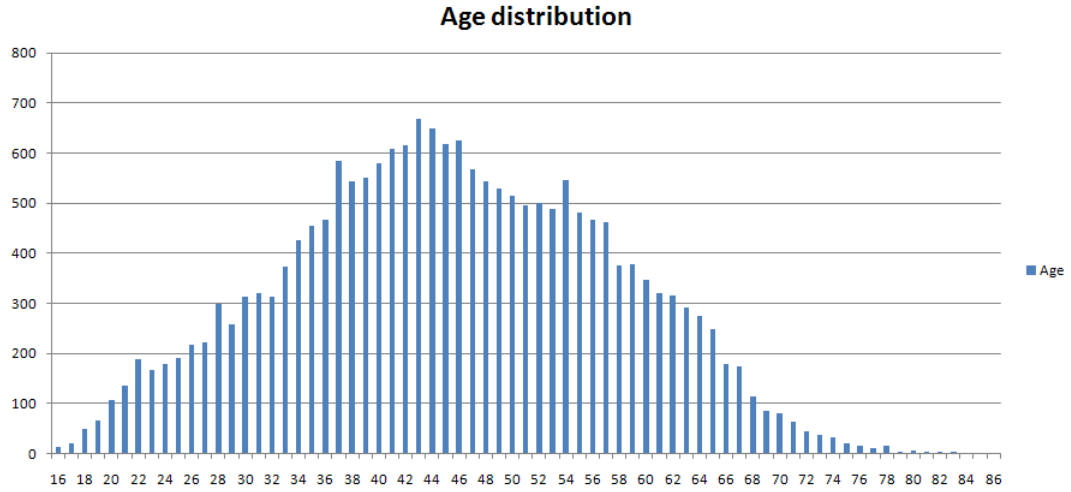


Figure 7: Histogram of the participating customers and their age.

symmetric matrix defined by $dev_{j,i}$ can be computed once and updated quickly when new data is entered. This means that once it's calculated, it will be required a lot less resources to generate predictions and recommendations for users compared to if it was needed to be calculated on the fly.

In order to implement the precomputing ability, we must generate a new database table to store this data. [25] presents a database-driven approach to item-to-item Collaborative Filtering in PHP and SQL that can support a full range of applications. The implementation of the Weighted Slope One algorithm was based on this, however a more efficient implementation was developed after thoroughly testing the first proposed approach, since [16] obviously tries to present their in an easy manner where optimization is not that important. [16] used PHP and SQL for their implementation, and since this project used C#, MATLAB and T-SQL (MS SQL Server) , the PHP and MySQL code had to be translated to C#, MATLAB and T-SQL respectively.

The database table *Deviation* that stores the calculated item-to-item matrix of the popularity differential where we compute by subtracting the average rating of the two items, is defined in listing 7.1.

Listing 7.1: he SQL that creates the deviation table

```
1 CREATE TABLE [Deviation](
2       [ItemID1] [int] NOT NULL,
3       [ItemID2] [int] NOT NULL,
4       [Count] [int] NOT NULL,
5       [Sum] [int] NOT NULL)
```

In listing 7.1 we see that we store only Count and Sum aggregates and not simply the average. This is a consequence of the fact that we can both add data and update the deviation table online.

The main implementation was as previously mentioned done in MATLAB. MATLAB was chosen because of it's considered as the *de facto* standard for implementing a wide variety of algorithms. The Weighted Slope One algorithm developed in MATLAB is a good match, as it generates

Customers	Ratings	Unique products
35614	254090	15789

Table 6: Rating data after data cleanup

Customers	Ratings	Unique products
20035	200350	14550

Table 7: Rating data after only selecting customers that have ten ratings.

large matrices. Fortunately, MATLAB is specially designed to operate with matrices, which usually results in a quicker development time. It also allowed for a more direct way when dealing with test and training set; the need to export the popularity differential data from one format into MATLAB was not necessary, since both the calculation of the popularity differentials and the generation of the test and training set were done at the same place.

When the implementation of the algorithm was completed, it was necessary to import the rating data from the database. As we know, this data was stored in a table containing the customer Id, product Id and finally, the rating itself. MATLAB do offer possibilities to connect to several type of databases, including MS SQL Server, however in this case, the most simple method was more than enough. The rating data was exported from the database to a text file, using tabulators to separate the columns and line breaks to separate the rows. Another MATLAB function was then developed to import this data, storing it as a matrix in memory and saved as a .MAT file for faster and easier import for later use.

To summarize the response represented in figure 5, 35614 customers completed the survey. Those customers generated a total of 261795 ratings distributed over 15790 unique products. Prior the popularity differentials were calculated, the data collected had to be checked for anomalies and invalid data in case it were any. This was done prior the export to MATLAB, and was done The results after the cleanup is presented in table 6.

Not too much data were removed at this stage. Even though 7705 ratings were removed, no customers were removed and only one unique product was removed. However, it was decided to filter out even more data. The item-to-item based collaborative filtering has a characteristic that it shares with most of the other types of recommender systems; the more ratings a user have, the more reliable and accurate are the predictions given[16]. The customers that participated in the survey rated between one to a maximum of ten products. In an ideal and realistic situation, there wouldn't be any maximum limit to the number of ratings the customer could give, but the survey was a special case and it was limited to the extent of how much work it was expected of the customer. Because of this, and to properly prepare the dataset for the creation of the test and training set, it was decided to remove the customers and their ratings for those customers who didn't have the maximum possible number of ratings, which was ten. The results after this cleanup is shown in table 7.

As we see in table 7 compared to table 5, 15579 customers have been removed with their 53740 ratings. 1239 unique products were also removed because of this. The decrease of data by removing customers that didn't have ten ratings may seem large at first, however the remaining

amount is still more than enough to get valid results. Prior the data cleanup, the average of ratings of each customer was ≈ 7.35, which explains why the reduction wasn't larger. The high average was also an advantage when creating the test and training sets.

During the testing of the Weighted Slope One algorithm in MATLAB, it became clear that significantly computational time was required in order to complete the whole dataset, because of the large amounts of data in it. To save time, it was decided to use computational power from the "cloud". Amazon Web Services offers a wide range of different kind of web services located in the cloud. One of the web services is Amazon Elastic Compute Cloud (EC2) which basically is a virtual server powered by Linux or Windows, controlled though an API or directly by remote desktop software. A virtual server, or instance as Amazon calls it, comes in different versions regarding processing power and system memory. The instance that offered the most powerful CPU was rented and MATLAB and Microsoft SQL Server were installed on it. The data were then exported to the server and set up, before finally starting the popularity differential calculation function in MATLAB. Once it was complete, it had created the deviation matrix that contained the popularity differential data. It was of the size 1418338 rows by 4 columns and in the same format as shown in listing 7.1, except for the part being stored in MATLAB. With this in place, the foundation for both non-personalized and personalized recommendations were set.

7.2.2 Implementing Non-Personalized Recommendations

As [25] states, a recommendation is "'non-personalized'" if it doesn't depend on a user profile. An example of this, a basic search engine provides a result based on the search query, and the result isn't personalized to a specific user. However, it can be useful to provide related items in addition to the result. This way we can help the user find items that the user likes as much or more than the current item. This approach is different from a similarity measure where we seek to present similar items, [25] states that item-to-item algorithms are not necessarily based on similarity across items.
A typical approach could be the following example: A user click on a item 1 from the item list to see more details about the product. A new page loads which contains this information. In addition, a list of related items is included. The list is generated by looking for items (called *ItemID2*) that:

- A sufficient large number of people (defined by a threshold) rated both item 1 and ItemID2,
- On average, we want ItemID2 to be rated as high as possible among users who also rated item 1.

Once the deviation table has been computed, the following SQL code in listing 7.2 ranks all the related items where we set the *count* threshold to 2 and the current item that are being browsed is *currentItemID*.

Listing 7.2: The SQL for Non-Personalized Recommendations

```
1 SELECT TOP 10 d.itemID2, d.Sum, d.Count, (CAST(d.Sum AS decimal) / d.Count) AS
      average
2 FROM Deviation d
```

```
3 WHERE d.Count > 2 AND ItemID1 = @currentItemID
4 ORDER BY (d.Sum / d.Count) DESC
```

After further expanding the SQL code to join in the product table from the prototype, and added some users that rated some different items, the online shop prototype now presents related products to the current product the user clicked on, shown in figure 8.

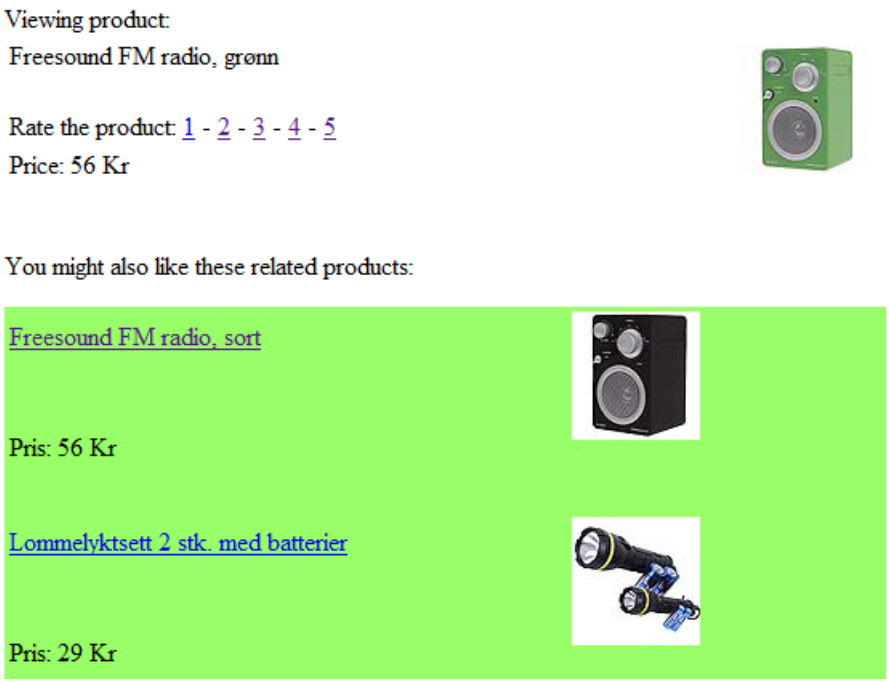


Figure 8: Product detail page with related products

7.2.3 Implementing Personalized Recommendations

Another typical scenario that involves item recommendations is for instance a list of products where the system with relatively high precision know that the user would like. This is done by letting the system try to predict the rating the user would have given an item, and optionally sort the list based on the highest rating predicted.
Listing 7.3 presents the function that only selects the best items that most probably are most interesting to the user. This SQL function was proposed in [25]. *currentUserID* is the ID of the current user that the recommendations are calculated for.

Listing 7.3: Function for presenting the items with the highest rating prediction

```
1 SELECT d.ItemID1 AS item, CAST(SUM(d.Sum + d.Count * r.RatingValue) AS decimal) /
      SUM(d.Count) AS avgrat
2 FROM Rating r, Deviation d
3 WHERE r.UserID = @currentUserID
4 AND d.ItemID1 <> r.ItemID
5 AND d.ItemID2 = r.ItemID
6 GROUP BY d.ItemID1
```

```
7 ORDER BY avgrat DESC
```

In the online shop prototype with personalized recommendations implemented, assuming that a certain user is logged in and that the user and multiple others has rated on some different items, a list containing the best predicted items for the current user will be presented, as shown in figure 9.

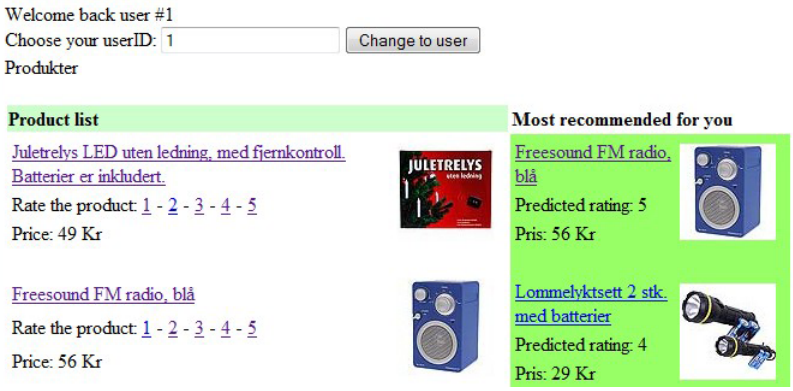


Figure 9: Product list with personalized recommendation. The predicted rating value is shown for demonstration.

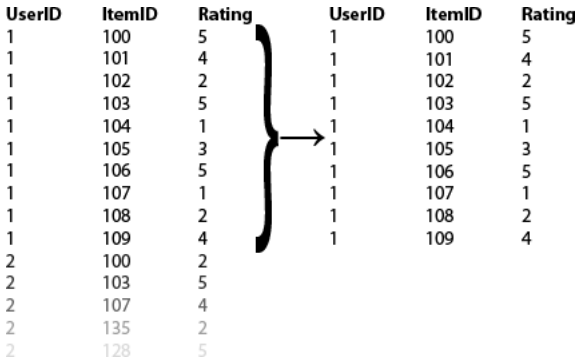
As we see in figure 9, the system has tried to predict the most interesting products to the current user, and for demonstration, the predicted value is included with each item. The system thinks that the current user would have given the blue radio a rating of 5, and the flashlight 4. This is why the blue radio is at the top, because statistically speaking, it is more interesting to the current user. It's also important to note that the Slope One algorithm schemes are able to generate recommendations after only given a few ratings, which significantly reduces the typical "“Cold-Start Problem"'. However, it's typical that the more ratings we have, the quality of recommendations increases. [7]

7.3 Non-contextualized recommandation performance

7.3.1 Setup and preparations

One of the most common methods to measure how collaborative filtering algorithms that calculates predictions perform, is to see how accurate the personalized predictions of the systems are. This was done by creating a training and test set from the rating matrix that contained all the ratings from the customers. To achieve a more stable and valid performance result, a ten-fold cross-validation was performed. This means that the rating matrix was partitioned into ten subsamples. This is illustrated in figure 10 for one user only, but the same procedure happens to each user. Of these ten subsamples, a single subsample is retained as the validation data for testing the model, and the remaining nine subsamples are used as training data. This is illustrated in figure 11. The cross-validation data is then repeated ten times, with each of the ten subsamples used exactly once in the validation data, and an error rate that defines the accuracy can be calculated by comparing the prediction against the actual value in the test set. This is shown in figure 12.

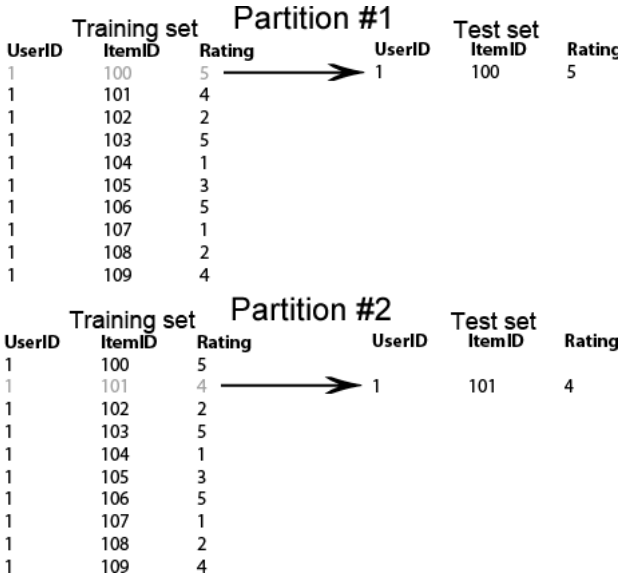
Finally, the ten error rates are averaged to produce a single estimation.



UserID	ItemID	Rating
1	100	5
1	101	4
1	102	2
1	103	5
1	104	1
1	105	3
1	106	5
1	107	1
1	108	2
1	109	4
2	100	2
2	103	5
2	107	4
2	135	2
2	128	5

UserID	ItemID	Rating
1	100	5
1	101	4
1	102	2
1	103	5
1	104	1
1	105	3
1	106	5
1	107	1
1	108	2
1	109	4

Figure 10: Example showing users with their ratings for items, before and after extraction a single user to prepare a k-fold cross-validation.



UserID	ItemID	Rating
1	100	5
1	101	4
1	102	2
1	103	5
1	104	1
1	105	3
1	106	5
1	107	1
1	108	2
1	109	4

UserID	ItemID	Rating
1	100	5

UserID	ItemID	Rating
1	100	5

UserID	ItemID	Rating
1	100	5
1	101	4
1	102	2
1	103	5
1	104	1
1	105	3
1	106	5
1	107	1
1	108	2
1	109	4

UserID	ItemID	Rating
1	101	4

Figure 11: Illustrates the partitioning to test and training set for one user. Only two partitionings are shown, partition 3 to 10 are not demonstrated.

The Statistics Toolbox in MATLAB was used for the partitioning and to create the test and training sets. A function was developed in MATLAB that iterated through a training set, and by aggregating the nine ratings for each user and using the personalized method for Weighted Slope One, a rating was predicted for the missing item. This rating was compared to the real rating that was stored in the test set, and the difference between them; the error rate e_i, was calculated by using the formula below.

$$e_i = |f_i - y_i|$$

where f_i is the predicted rating and y_i is the true rating. When the error rate had been computed for all predictions, it was averaged for the particular training and test set it worked on. Mean Absolute Error (MAE), Mean Square Error (MSE), Root Mean Square Error (RMSE) and the Normalized RMSE were used to calculate this for comparisons. MAE is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i|$$

where f_i is the prediction and y_i is the true value, which in this case is the true rating. These are also used during the calculation of the MSE, which is given by:

$$MSE(f, y) = \frac{1}{n}\sum_{i=1}^{N}(f_i - y_i)^2$$

The RMSE the square root of the MSE:

$$RMSE = \sqrt{MSE}$$

And the Normalized RMSE is given by:

$$Normalized\ RMSE = \frac{RMSE}{x_{max} - x_{min}}$$

where x_{max} is the highest error rate reported and x_{min} is the minimum. The Normalized RMSE is often expressed as a percentage, where lower values indicate less residual variance.

7.3.2 Results

Figure 13 presents the results from each training and test set using different error rate metrics. Detailed data are in table 11 in the appendix. The averages for MAE, MSE, RMSE and Normalized RMSE 0.2708, 0.2077, 0.4557 and 5.99% respectively.

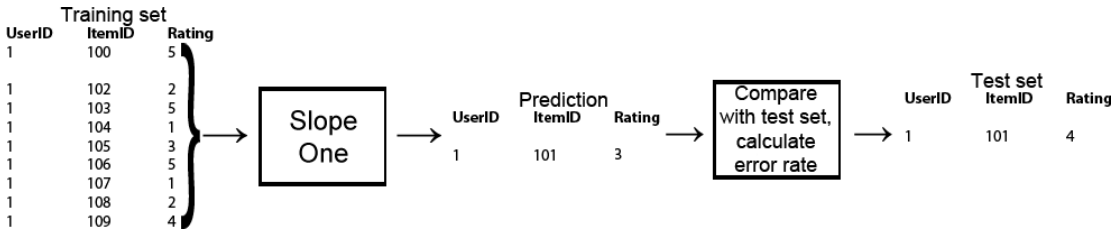


Figure 12: Illustrates the flowchart for predicting a rating for the missing item and by calculating an error rate one time for one user. The error rate in this example would be $|3-4|=1$.

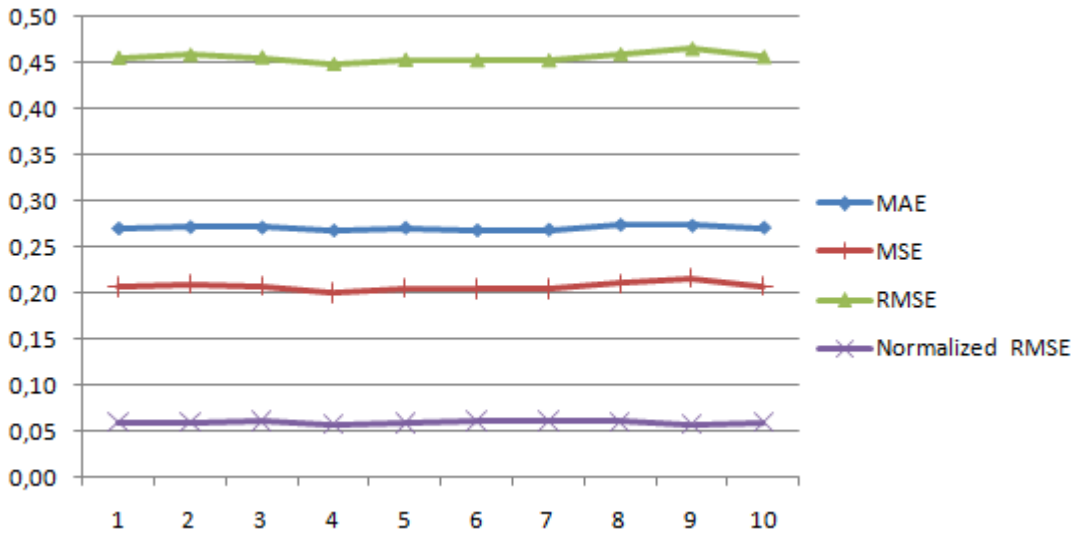


Figure 13: The results from the different metrics that were used to calculate the error rate on the ten training and test sets.

7.4 Tag distribution over the contextual features

7.4.1 Introduction

The first research question regards the process of trying to identify some relevant contextual features, and a set of them have been suggested and presented in this thesis. Customers have participated in the survey where they could tag some of their previously bought products to indicate that they are related to some of the given contextual features.
Trying to answer this research question, three things must be studied:

- What is the distribution of the tags over the different contextual features?
- For each contextual feature, how were the tags distributed over the products?
- The customers classified the products to related contexts using a social tagging model. Did they tag the products similarly, or was it large variations?

The distribution of the tags will directly show which contextual features that got most tags, but it doesn't say anything about the spread of the products - how many unique products that were tagged. The holiday Christmas could for instance have more tags than the season summer, but the actual number of different products that were tagged being related to Christmas might be very small, while the number of different products tagged being related to summer could be significantly higher. By including this ratio in the analysis, it's possible to get a better overview compared to only look at the tag distribution only.
Since the customers were given the task and responsibility to classify products being related to certain contextual features by tagging them, it's expected that there will be variations among the tags belonging to a product. Because of this, a kind of cutoff filter will be used, making sure that it's more likely that the products that were accepted by the filter are classified with increased validity. The filter will be based on the average tag count for the different contextual features. Those products who were tagged at least the equivalent times as the average tag count, are classified as being related to that particular contextual feature.

7.4.2 Setup

The tags were stored in the survey database. Each tag entry contained information to find the customer behind the tag and of course the product that was tagged. To directly find the general distribution of the tags over the different contextual features, a SQL script was written that returned the total tag count of the selected contextual features. This SQL script is included in listing 7.4 and shows how to return the total number of tags related to any kind of season. This means that the count is the total count of summer, autumn, winter and spring combined. A filter is also included, since it turned out that it existed some dummy products in the database which had their product Id being 0, and those were thus ignored during the count.

Listing 7.4: Returns the total number of tags related to any kind of season

```
1 SELECT COUNT(1) AS TotalTags FROM ProductSeasonRelated psr
2 JOIN FakturaLinjer fl ON fl.FakturaLinjeID = psr.FakturaLinjeID
3 WHERE fl.ProduktID > 0
```

This SQL script was executed for every main contextual feature class, which was the total count of all their subclasses:

- All kinds of usages
- All kinds of gift receivers
- All kinds of gift occasion
- All kinds of seasons
- All kinds of weather types
- All kinds of special days / holidays

In addition to get the overall distribution, it was also necessary to specify it, meaning that each of the subclasses of the main contextual features also had to be considered separately. A simple adjustment to the SQL script was enough to do this, shown in listing 7.5. In this example, we see that we have narrowed down the counting mechanism to only count tags belonging to the season summer, which had the corresponding Id of 1.

Listing 7.5: Returns the total number of tags related to the season summer

```
SELECT COUNT(1) AS TotalTags FROM ProductSeasonRelated psr
JOIN FakturaLinjer fl ON fl.FakturaLinjeID = psr.FakturaLinjeID
WHERE fl.ProduktID > 0 AND psr.SeasonID = 1
```

This SQL script was executed for each of the subclasses belonging to each of the main contextual features to get a finer distribution over all different available contextual features.

Now that the tag distribution have been mapped, the number of unique products that were covered for each of the main contextual features had to be found. Another SQL script was made to accomplish this, shown in listing 7.6, where season is the current contextual feature selected. To specify a certain season or any other contextual feature, only a small change is necessary to the code to find this.

Listing 7.6: Returns the total number of unique products related to any season

```
SELECT COUNT(1) AS UniqueProducts FROM (SELECT DISTINCT fl.ProduktID
  FROM ProductSeasonRelated psr --change tablename to the wanted contextual feature
  JOIN FakturaLinjer fl ON psr.FakturaLinjeID = fl.FakturaLinjeID
  WHERE fl.ProduktID > 0
  /* uncomment to find more specified results
  * AND psr.SeasonID = 1
  */
) AS derived
```

7.4.3 Raw data

Figure 14 is the result from executing the SQL in listing 7.4, with small changes to get the different types of the main contextual features. It shows the total, unfiltered tags for the different contexts given by the customers, and the number of unique products that was covered by those tags. In the figure, the contextual sub-features are grouped into their main types, except for usage which is shown separately in figure 15 because of its high values compared to the other ones. For instance, "Any Season" are the aggregated tags for summer, autumn, winter and spring.

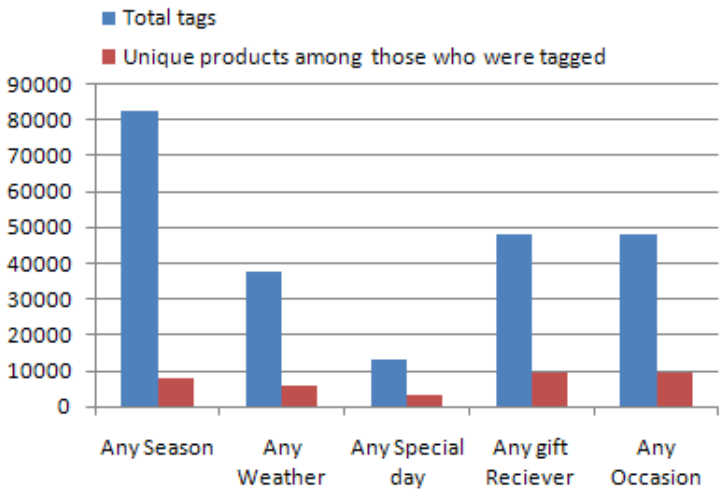
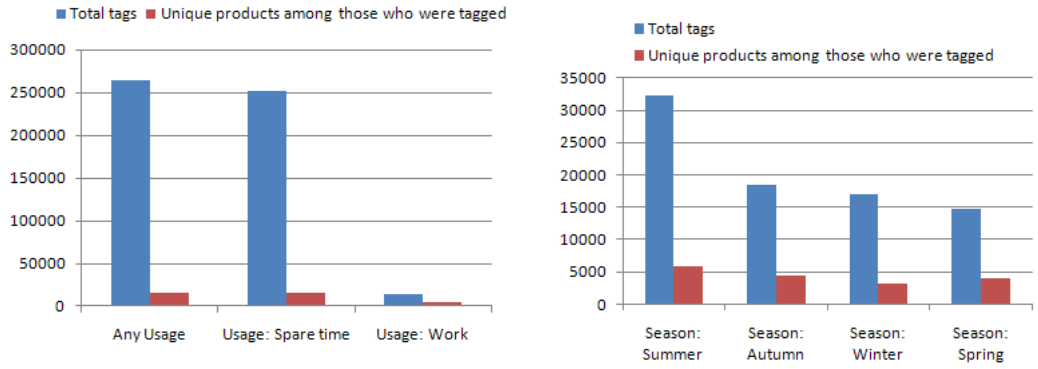


Figure 14: An overview over the total tags collected and the unique products of the different contextual groups except usage.

The contextual feature *usage* got most of the tags. It got a total of 265085 tags. Those tags was spread over 15739 unique products, which accounted for 99.68% of the overall number of unique products that were rated by the customers. Season came on second place with a total of 82643 tags and 7894 products. According to this, the customers meant that 49.99% of all the products that were rated, were in some way related to at least one kind of season. Next comes gift receiver and occasion, which always have the same count since they can't exist without each other. They have each 47937 tags spread over 9552 products. The last two are weather with 37630 tags and 6030 products, and special day / holiday with 13234 tags distributed over 3138 products.

Figure 15 presents the data for the different seasons. Summer is clearly the one with most tags and have most unique products, respectively 32307 and 5831. Despite this, autumn, winter and spring does not have significantly fewer unique products compared to summer. More detailed figures that show the differences between the other contextual sub-features are included in the appendix.



(a) Usage

(b) Season

Figure 15: Tags and the unique products covered for usage and season.

	Summer	**Autumn**	**Winter**	**Spring**
Total tags	32307	18554	16952	14830
Unique products	5831	4340	3175	4036
% of all season products	73.87 %	54.98 %	40.22 %	51.13 %
% of overall products	36.93 %	27.49 %	20.11 %	25.56 %

Table 8: Total tags and the unique products for season

7.4.4 Theshold selection and filtered data

Excluding the very general contextual feature *usage*, season clearly has the most tags among the different contextual features. Because of this, season will be used to demonstrate the selection process for for deciding a tag threshold that will act as a cutoff filter. This filter will do quality assurance, where the appointed products are classified with increased validity. In figure 15 the different tags for season are distributed over the different seasons. The tag count and unique products are shown in table 8.

Figure 16 shows how many products related to the different seasons that have a minimum and a certain tag count. A product can be tagged as being related to multiple seasons if the customer found this fitting. The plot shows that the majority of the products have very few tags. For instance, 44 % of all products related to summer have only been tagged once. 12 % have been tagged at least ten times and 0.5 % have been tagged at least 100 times. The products' tag counts where compared to each other to analyze the classification quality. It was clear that the more times they had been tagged, the more valid were the classifications.And those with very few tags, and especially those with only one tag, were very likely to be misclassified.

In order to get more reliable classifications, some kind a filtering had to be introduced. A common method was chosen, the *threshold*, being a number. Those products who were tagged at least the same number of times as the threshold, would be classified as being related to a given contextual feature. But what should the threshold be? Different methods were evaluated. One way was to set the threshold to a certain number, 20 for instance. But it could be a challenge

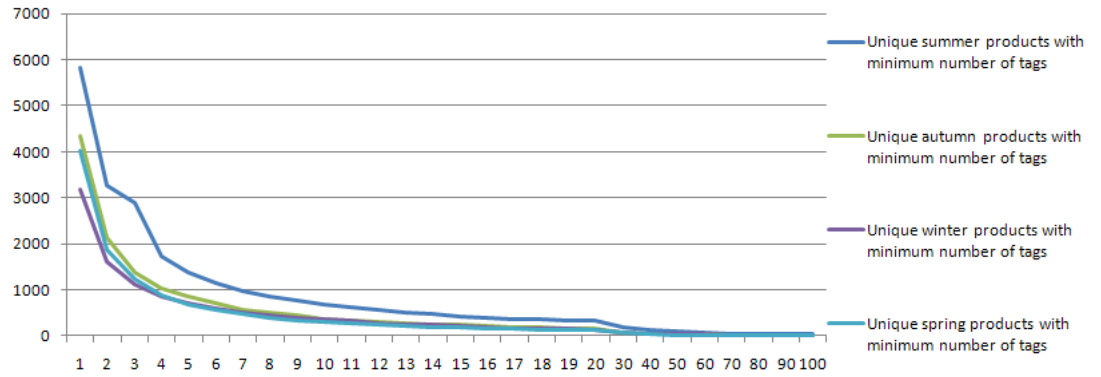
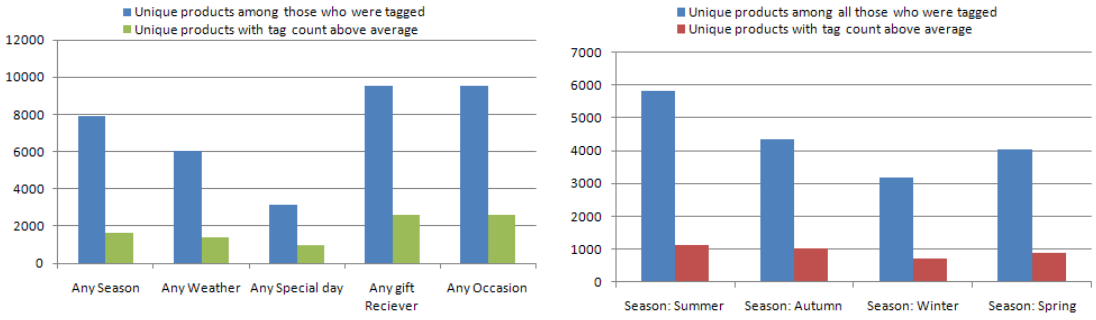


Figure 16: An overview over how many products that have at least been tagged n times.

if the same limit would be used across all of the different contextual features, since the number of tags and unique products varies among the contextual features. And the process of actually picking a certain number manually is highly subjective. So a more quantitative approach was chosen. The threshold was set to be the average tag count for the products related to a given contextual feature. This average value was rounded to the nearest integer. For instance, the products related to summer had an average tag count of 5.54. This means that products had to be tagged at least 6 times to be classified as summer related products. The different average tag counts are presented in figure 19 and were found using the SQL script in listing A.5 from the appendix. Other methods were also considered, for instance median and mode. They were both discarded because it returned too low values. Mode was always one for every contextual feature. Median showed slightly more variation, but still inadequate. The median for summer was 2, autumn 1, winter 1 and spring 1. The majority of all contextual features had medians being 1. The only exception was usage, with spare time. This one had a median of 5, the highest value retrieved. Work had a median of 1. It was concluded that the median values were too low to be used to classify products with higher validity. In fact, if the threshold was set to one, then it wouldn't have any practical use. There would be no difference between unfiltered and filtered results - those products that were tagged have at least been tagged once.
An overview over how many products that have least been tagged n times to seasons are showed in figure 16. For instance, almost 6000 products have been tagged to summer at least one time. When only considering products that have been tagged to summer at least two times, this number is reduced to approximately 3000. The precise reduction in percent is 56 %. Even though the mean also returned a few values of one for some contextual features, it was chosen as being the best alternative.
Having too many products classified would have increased the risk of misclassifications, and too few could have made it more difficult or even impossible to identify overall relevant contextual features, because of the data would have become too sparse. The mean average returned values somewhere in the middle.

Using the SQL script in listing A.7 in the appendix, the filtering resulted in a significant reduction in unique products, as seen in figure 17. Unique products season went from 7894



(a) (b)

Figure 17: Shows the number of unique products before and after the tag filtering.

to 1659, which gives a reduction of 79 %.

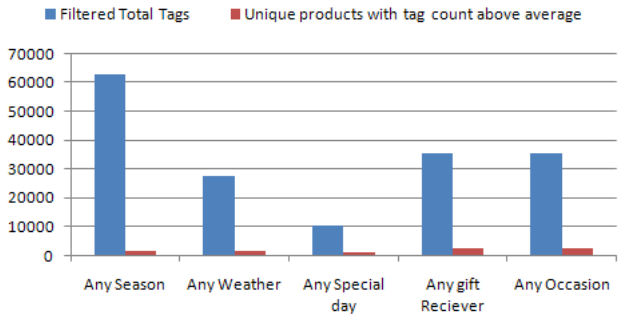


Figure 18: Shows an overview over the total tags and unique products of those products that have a greater tag count than the average, collected for the different contextual groups except usage. The unfiltered version is in figure 14.

Figure 20 show how many tags that was considered before and after the filter was introduced, while figure 21 compares the number of unique products before and after filtering for all of the different contextual features. This was found using the SQL in listing A.6 from the appendix.

7.4.5 Limitations

One thing that could have influenced the calculation of the tag count averages, was the lack of restriction that prevented customers from rating and tagging same products multiple times. For instance, a customer could hypothetically have bought ten garden hoses, which means that the customer would have rated and possibly tagged this product ten times. Let's say the customer tagged all of them being related to summer, and the average tag count for summer was 5.54. Since the garden hose had ten tags, it would have been classified as being related to summer. In this case this would be a correct classification, but if the customer tagged in a way that would have been considered wrong by the majority of customers, then a misclassification would have

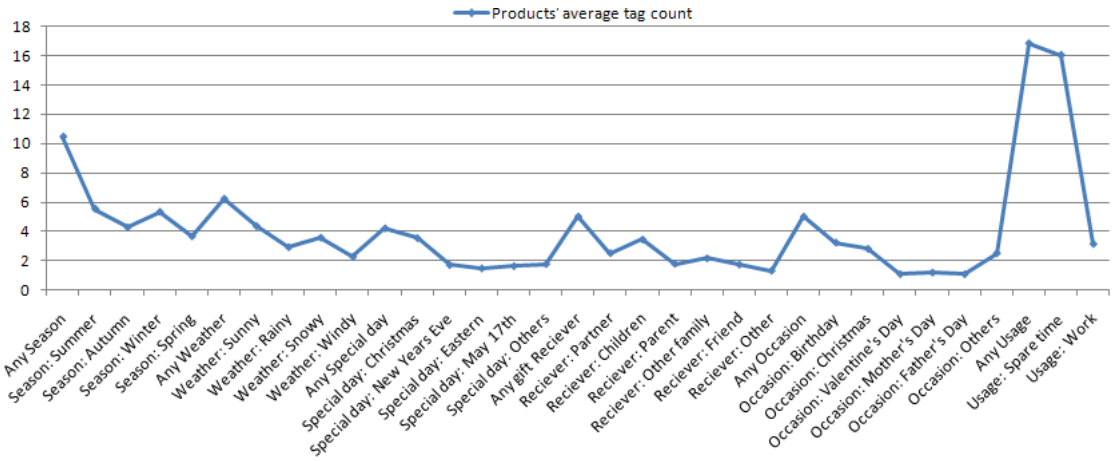


Figure 19: An overview over the average tag counts for the different contextual features.

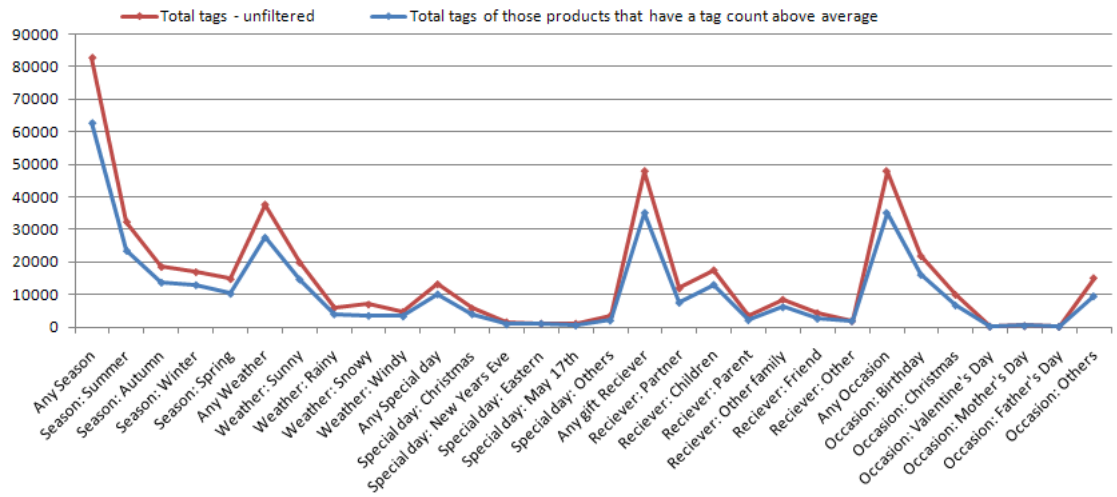


Figure 20: An overview over the tag counts for the different contextual features, unfiltered versus filtered (using average tag count). Numbers for usage is shown seperately in the appendix.

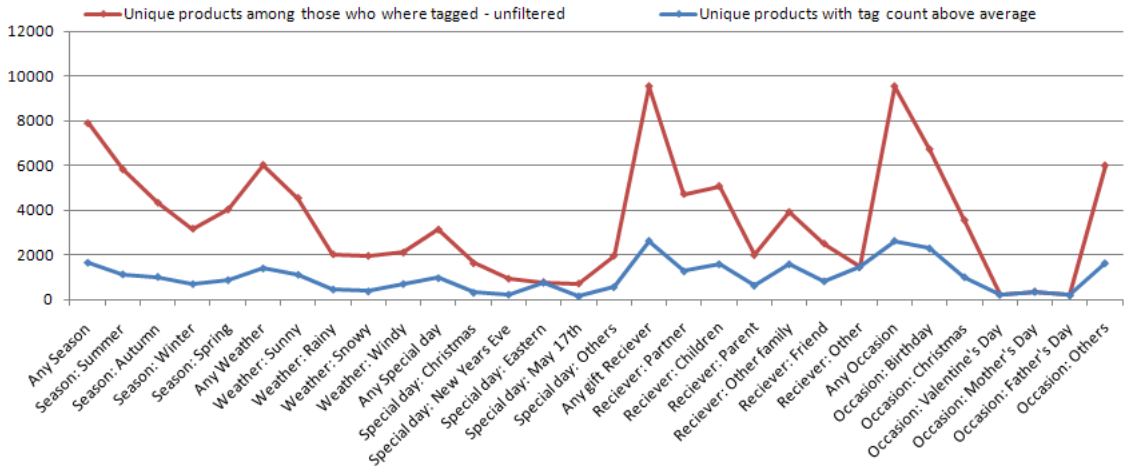
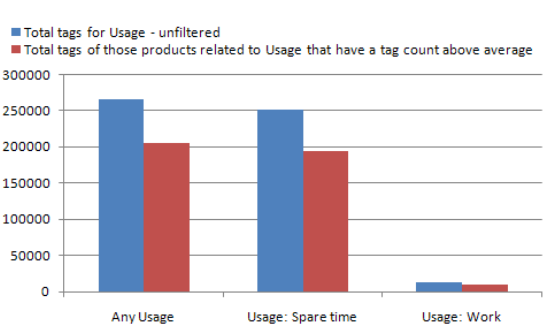
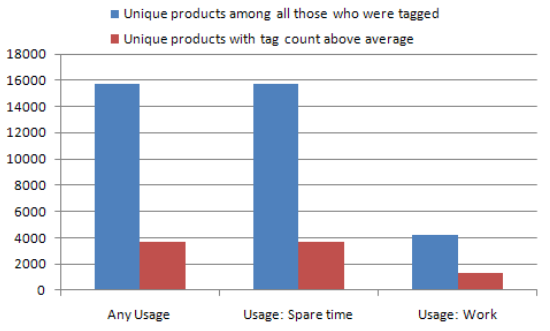


Figure 21: An overview over unique products related to the different contextual features, unfiltered versus filtered (using average tag count). Numbers for usage is shown seperately in the appendix.



(a) Total tags for Usage - unfiltered VS filtered



(b) Total unique products for Usage - unfiltered VS filtered

Figure 22: Tags and the unique products related to usage, unfiltered versus filtered (using average tag count).

occurred. The whole idea with the limit is that a classification is more trustworthy if ten different customers tagged the same product the same way only one time each, compared to that only one customer tags the same product ten times. The latter did in fact occur, and this was measured. Those contextual features that had products that were potentially misclassified are shown in figure 23, and those numbers wear found by using the SQL script in listing A.8. If one customer tagged one product multiple times (at least the average tag count number) the same way for that particular contextual feature, then this is recorded and contributes to the chart. However as mentioned, they are only potentially misclassified, they could very well be tagged in what's considered correct. In addition, some of the contextual features are less suitable to be classified than others. It's typically more natural to classify a product being related to summer compared to birthday, however, there exists products that are directly related to birthdays, like birthday cards for instance.

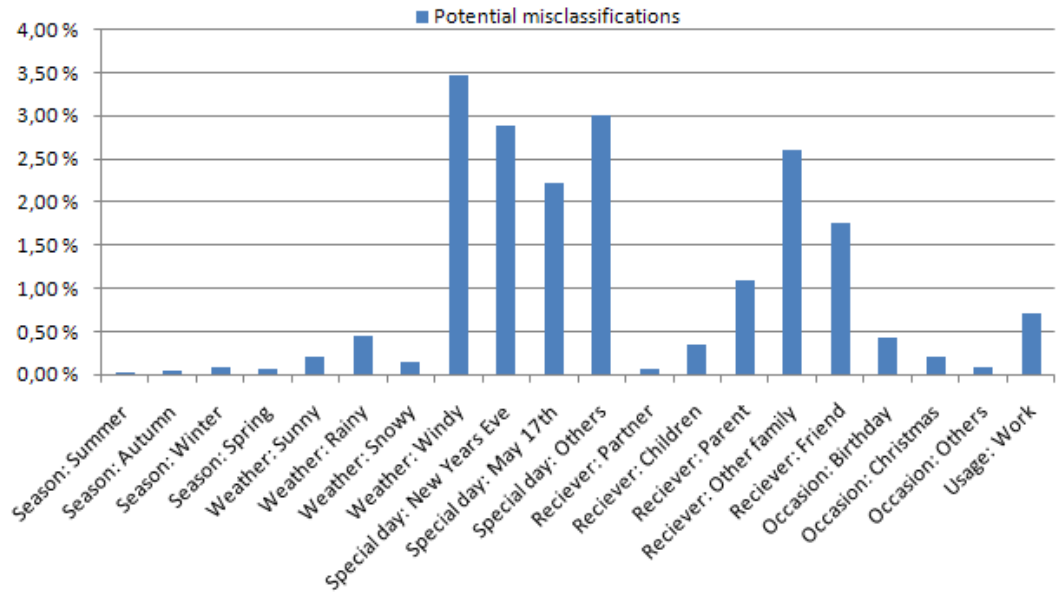


Figure 23: Percentage of possible unique products that potentially could have been misclassified. Only contextual features that were prone to have errors and with average tag count being at least 1 are included.

The second limitation that could have affected the overall tag distribution, was the sort mechanism that was used in the survey. Since products were sorted by their purchasing price in descending order, this could have resulted in less variations among the products that were chosen by the survey. There are some indications of this. A few products had over 200 tags within one contextual feature. Since these few products appeared that frequently in the survey, it means that they were some of the popular and expensive products that seemed to be relevant to one or more contextual features. It's challenging to measure how accurately this could have affected the outcomes, since Netthandelen.no AS is based on an auction based model, meaning that one customer could spend for instance 5000 NOK on a product, and another customer spend 2000

NOK on the same product. It's however, probable that this had a very low impact on the final outcome because of the large number of participating customers and the number of unique products that were included in the survey.

The third limitation that could have affected the overall tag distribution and thus the averages for each of the contextual features was that the customers couldn't revert and change some tags after they had submitted a particular product. This could hypothetically affect the outcome in case a customer had tagged and submitted one product, and later on realized that it was wrong, for instance caused by an initial misunderstanding.

The last limitation is regards the competition aspect. It's always a possibility that some of the customers feedback were somewhat affected by the competition itself. For instance, some could have believed that if they gave good feedback, this would increase the chance of winning. They could also been positively surprised by the initiative of the competition, and give better feedback. It's difficult to measure this potential effect.

7.5 Contextualized recommandation performance and outcome

7.5.1 Introduction

The second research question regards how the contextual features can affect the final recommendation outcome. To measure this, an experiment was conducted, that consisted of two parts. The first part focused on the performance of the context-aware recommender system that was implemented. This process consisted of the following:

1. Calculate all possible predictions for a customer
2. Using these predictions, only select those products that are classified as being related to one or multiple contexts. This is post-filtering, since predictions are generated prior filtering.
3. Calculate precision and recall and finally an F-score for the customer to measure the retrieval performance.

The second part regarded more on how the final recommendation outcome generally was affected by the different contextual features, by looking at the number of recommendations before and after using post-filtering. Different scenarios in forms of different contextual features with and with combinations were simulated to study the differences between them, and hopefully get some indication of which consequences that might happen when granularity level increases.

7.5.2 The choice of post-filtering instead of pre-filtering or contextual modeling

As described in the related work section, pre-filtering is used when contextual information is applied prior the calculation of recommendations. Post-filtering is the opposite, applied after the calculation of recommendations. Both pre- and post-filtering are methods that uses loose coupling. Contextual modeling are the integration and merging of contextual information and the prediction system becoming one tightly coupled model.

Post-filtering was used in the experiment because of the following reasons and explained below:

1. Recent research shows that post-filtering can lead to better performance compared to pre-filtering
2. Pre- and post-filtering seems to be the most popular and flexible alternatives for implementation, especially with existing systems
3. A contextual model requires to be implemented directly into recommendation algorithms
4. Pre-filtering is less suitable for model-based predictors, like the Slope One family
5. Pre-filtering is reported to increase the data sparsity, which basically leaves the predictor with less data to work with

Pre-filtering was considered, but post-filtering seemed to be a more logical choice. The reasoning behind this decision is that the Slope One family algorithms are model-based. Model-based recommender systems precompute parts of the necessary calculations off-line (the principle behind the deviation matrix used by Slope One), because they tend to be too computationally expensive and would have suffered if used to generate predictions real-time. Pre-filtering is more suited for memory-based recommender systems since they calculate predictions on the fly. This means in practical terms that if pre-filtering was used with contextual features, the deviation matrix would had to be calculated for each of the different contextual features that were used, which in general can't be considered feasible in realistic scenarios. And last, recent research show that pre-filtering often lead to greater sparsity from the very beginning, reducing the dataset the predictor can work with, and decreasing the overall performance compared to post-filtering.[14][20]

7.5.3 Setup

The deviation matrix was as mentioned before, pre-calculated in MATLAB. This matrix was exported to a text file and then imported to the table in the database, called Deviation. Using the personalized recommendation approach, all possible predictions were calculated for a customer in MS SQL Server this time, using the Weighted Slope One algorithm. To measure the performance of the contextual recommendations on a customer basis, three separate SQL scripts were used to filter out only those recommended products that were related to at least one given context and to calculate the F-score from precision and recall. This procedure was done on every customer from the reduced dataset, which were customers that had rated ten products. To have this procedure automated, two of the mentioned SQL scripts was converted to two separate scalar-valued functions in MS SQL Server. The third SQL script called those functions and saved the precision, recall and F-score for every customer in a table. The flowchart for this procedure is shown in figure 24.

Although this process were automated to calculate precision, recall and F-score for all customers, it was necessary to specify which context that should be used and enter the threshold (average tag count). The main SQL script where this values are set are shown in listing A.9, which calls the two scalar-valued functions in listings A.10 and A.11. The former function returns the number of products that are related to the specified context. The latter function return the number of true positives. This is the number of products related to the given context that are recommended to the customer. All possible predictions for the specified customer are calculated here, but only those products being related to the given context are considered. The calculation

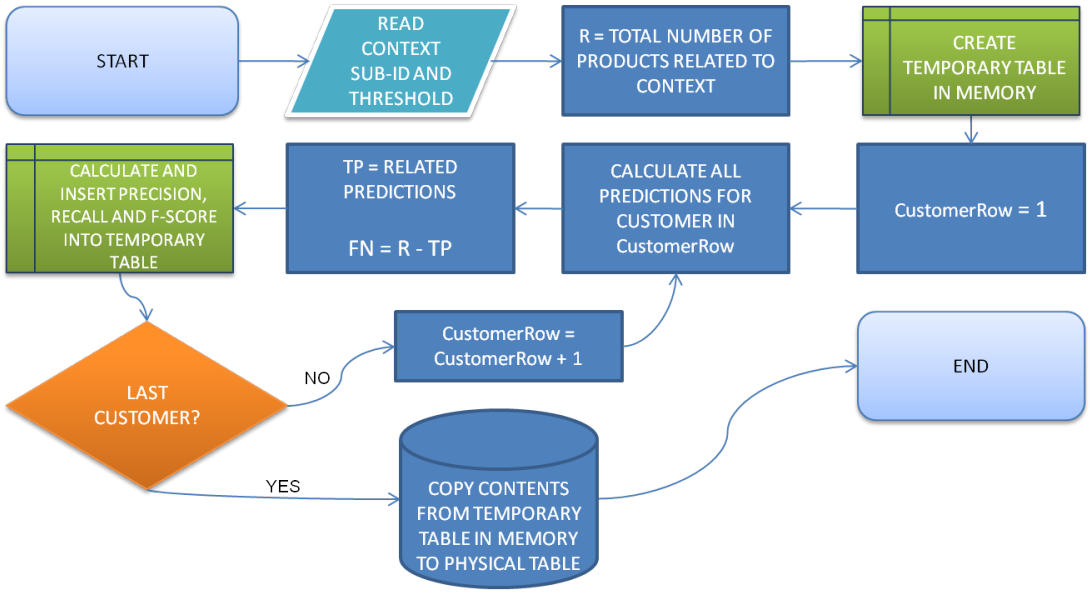


Figure 24: Flowchart for calculating precision, recall and F-score. This process was executed for each contextual feature.

of precision is in this application only based on the number of true positives, since either it returns some relevant and recommended products or it doesn't. This method does not return false positives. This means that if there at least one true positive, then precision will be one, or zero otherwise.

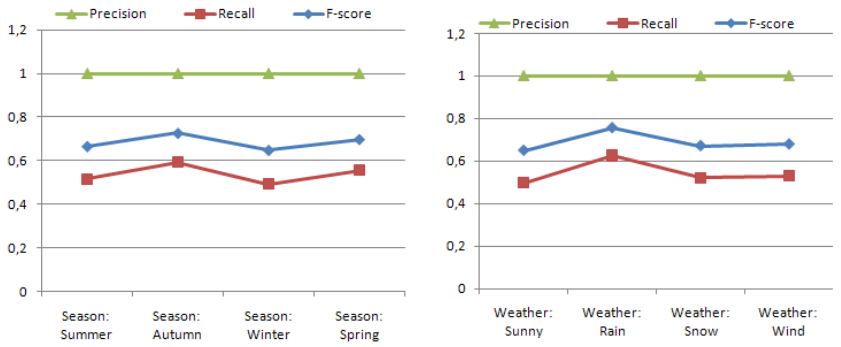
The calculations were computationally expensive. The calculation of *one* contextual feature, summer for instance, was measured and estimated to require over 200 hours. If only accounting single contextual feature scenarios, this would have required a total of 225 days on a dedicated modern physical server. When including scenarios using multiple contextual features, it would have required an estimated additional 100 days. It's probably possible to optimize the process further using different methods, but this would have required more time just to do this. The solution was to use cloud computing. Amazon EC2's most powerful (CPU wise) instance was rented. It did one calculation in an average of 30 minutes, instead of 200 hours. Different scenarios were set up. First, isolated scenarios were simulated, measuring all the different types of contextual features. After this, more complex simulations were initiated and measured. These scenarios used at least two different contextual features combined, up to maximum five. The results are saved in different tables, as described in figure 24. Then precision, recall and F-score is averaged across all the 20035 customers using the SQL script in listing 7.7.

Listing 7.7: This SQL gives the average precision & recall and F-score for all customers.

```
1 SELECT AVG(a.Prec) AS avgPrec, AVG(Recall) AS avgRecall, AVG(Fscore) AS avgFscore
2 FROM [NHSurveyFinal].[dbo].[PrecRecSeasonSummer] AS a
```

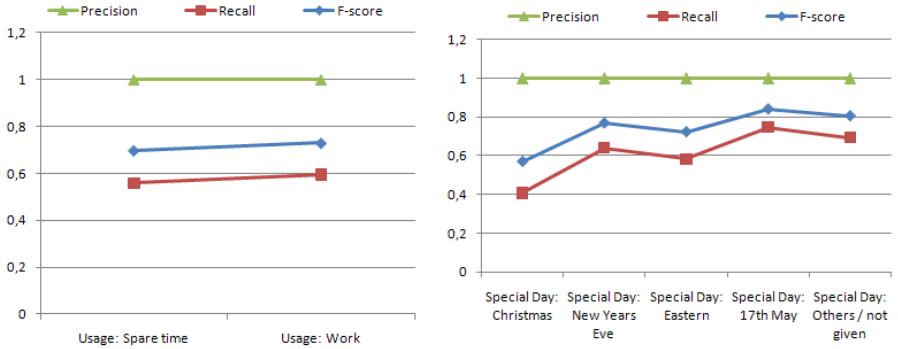
7.5.4 Isolated scenarios

The isolated scenarios used only one contextual feature each and measured separately for comparison between each other and against those scenarios that used multiple contextual features. On the performance graphs, the F-score and recall have similar patterns. The reason for this is that precision in all cases were 1, which means that recall is a variable with great influence. Regarding precision always being one; it's an either-or scenario. In this setting, all retrieved products is required to be relevant. So even if only one product is retrieved, it means that it's relevant, and hence precision is one. Figure 25 (a) show the performance for the different contextual features under season, and (b) shows the same for weather. Figure 26 (a) shows for usage, and (b) special day / holiday. The last figure 27 shows the performance for receiver (a) and occasion (b).



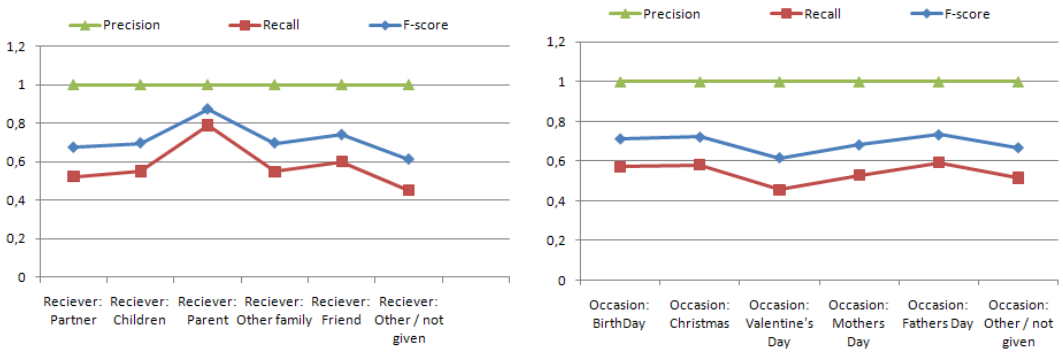
(a) Season (b) Weather

Figure 25: Precision, Recall and F-score for season and weather.



(a) Usage (b) Special Day / Holiday

Figure 26: Precision, Recall and F-score for usage and special day / holiday.



(a) Gift Reciever

(b) Gift Occasion

Figure 27: Precision, Recall and F-score for gift reciever and occasion.

7.5.5 Combined scenarios

To get some indication on what happens when multiple contextual features are included in the recommendation process, five scenarios were set up. In addition, each of those scenarios was divided into two sub-scenarios. The final choice of which contextual features that were used was based on a combination of those who had the largest tag count and matched each others in a natural and realistic way. For instance, the season winter and weather snowy is a natural couple compared to season summer and weather snowy. The five scenarios were the following:

Generalized contextual features:

Contextual feature parent class	Subclass
Scenario #1	
Season	Any
Scenario #2	
Season	Any
Special day	Any
Scenario #3	
Season	Any
Special day	Any
Receiver	Any
Scenario #4	
Season	Any
Special day	Any
Receiver	Any
Occasion	Any
Scenario #5	
Season	Any
Special day	Any
Receiver	Any
Occasion	Any
Weather	Any

Table 9: The scenarios that used only the general contextual features.

Specialized contextual features:

Figure 28 show the performance of the different scenarios that use generalized contextual features. The more contextual features that are used, the larger the F-score gets. At the same time, the numbers of unique products that are finally recommended to the customer after post-filtering with the different contextual features, are reduced when the granularity level of used contextual features increases.

However, the performance of specialized contextual features show less variation compared to the generalized one, seen in figure 30. The numbers of unique products in these specialized scenarios follow the same patterns as the generalized one, but results in even fewer recommendations. When using all five contextual features at once, the number of recommendations is only 27. These observations are of practical reasons explained in the discussion chapter.

Contextual feature parent class	Subclass
Scenario #1	
Season	Winter
Scenario #2	
Season	Winter
Special day	Christmas
Scenario #3	
Season	Winter
Special day	Christmas
Receiver	Partner
Scenario #4	
Season	Winter
Special day	Christmas
Receiver	Partner
Occasion	Christmas
Scenario #5	
Season	Winter
Special day	Christmas
Receiver	Partner
Occasion	Christmas
Weather	Snowy

Table 10: The scenarios that used the specialized contextual features.

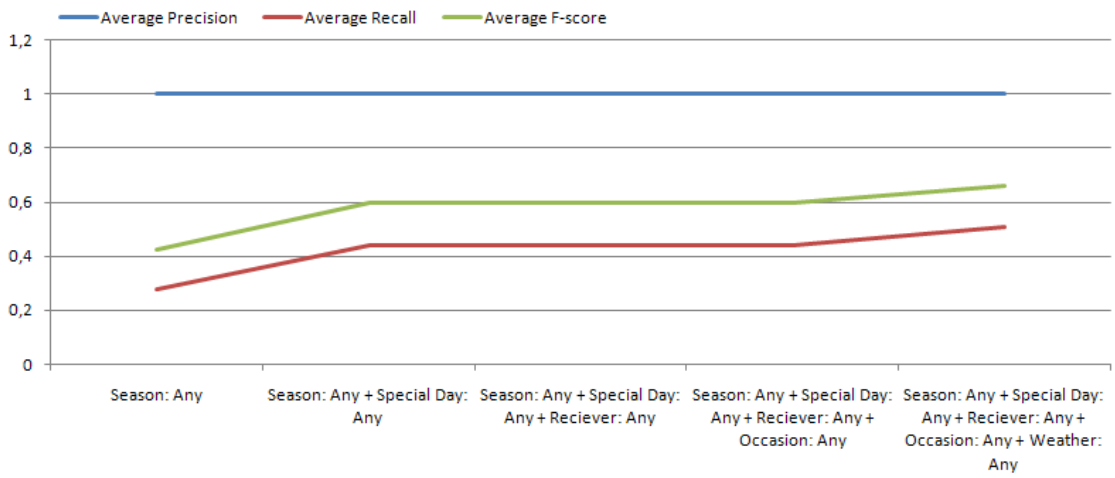


Figure 28: General contextual features: Precision, Recall and F-score versus different granularity levels.

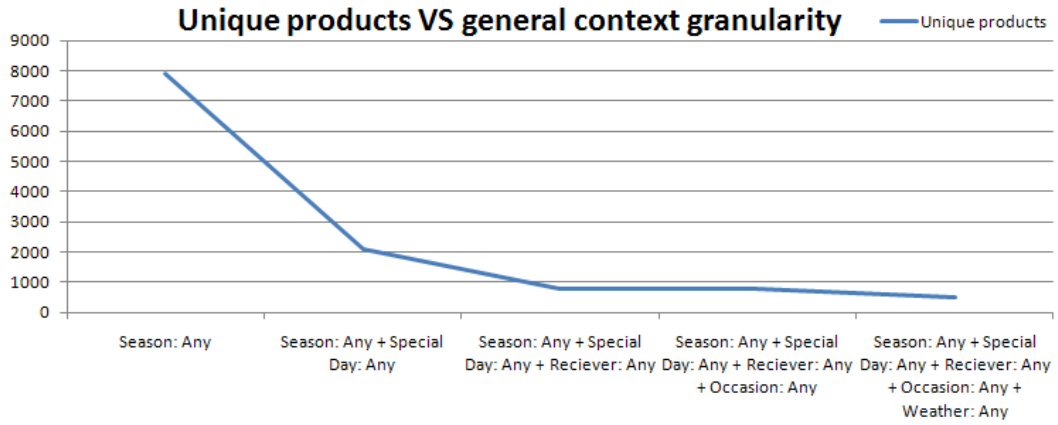


Figure 29: General contextual features: Unique products versus different granularity levels.

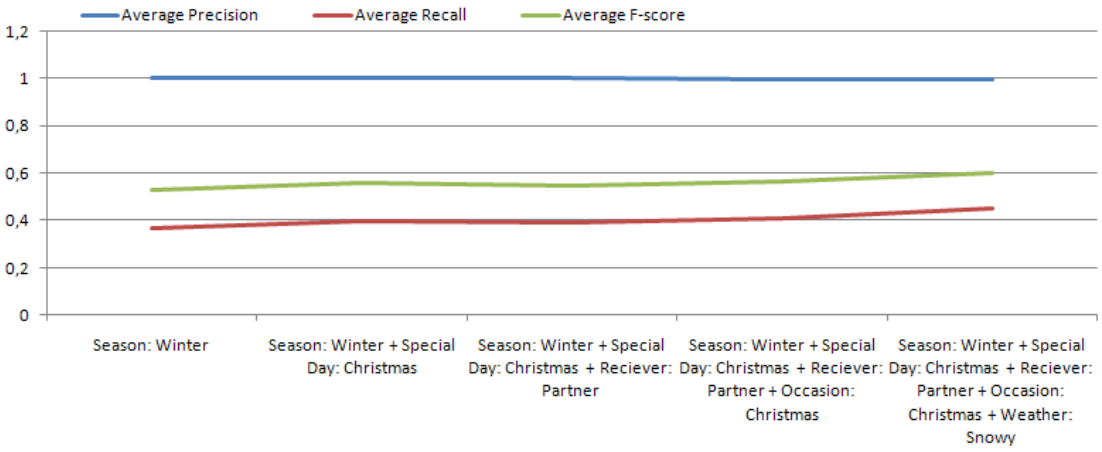


Figure 30: General contextual features: Precision, Recall and F-score versus different granularity levels.

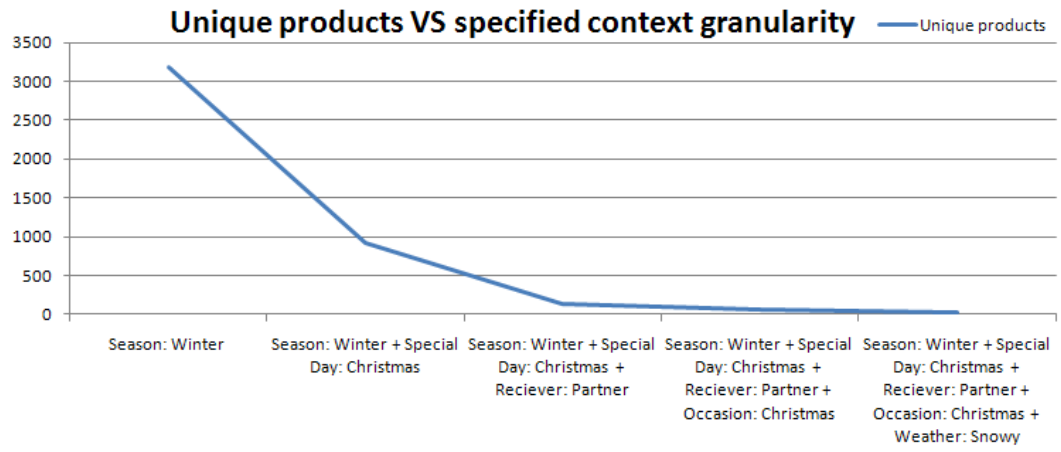


Figure 31: Specified contextual features: Unique products versus different granularity levels.

8 Discussion

8.1 Introduction

Suggesting contextual features that can be used in context-aware recommender systems is only one side of the story. The process of identifying the relevant ones another one. In this thesis some contextual features were suggested, applied and measured. The main classes *time, location, activity* and *identity* were based on Dey and Abowd's work on context-aware computing [10] and an additional contextual feature, *purpose*, was introduced in this thesis to be used in context-aware recommender systems. Some of the subclasses were based on previous work, as mentioned in related work. In 2008, Adomavicius and Tuzhilin agreed on that more research had to be conducted to find relevant contextual features that could be applied to context-aware recommender systems. Despite this, no previous research have been found that directly focused on the contextual features themselves, but some have been used as examples. For instance, a few were used to present context-aware recommender frameworks or measure the differences in performance between pre- and post-filtering. The contextual features themselves were one of the main things that this thesis focused on - to get some indications on which contextual features that could be used in a e-commerce setting.
It's indications because there is no right or wrong or exact science when it comes to identifying contextual features. Which ones to use in an e-commerce setting depends on the market and potential niches that an e-store operates in. Many e-stores sells products that are hard to classify as being related to a certain context. For instance, lets use a context-aware recommender system that senses which season it currently is. A store that sells nuts and bolts might not have that much use for this, while a travel agency could have benefited more from this.
While some e-stores have reduced need for context-aware recommender systems, others have an increased need. Especially e-stores that have taken advantage of the retail possibilities the Internet provides, can get increased value from context-aware recommender systems. Amazon.com is an example of an online store that leverages "The Long Tail" principle. The Long Tail refers to the statistical property that a larger share of population rests within the tail of a probability distribution than observed under a 'normal' or Gaussian distribution, as seen in figure 32.

Amazon take advantage of this; in addition to sell the few, but most popular products, they also offer a lot of niche products. In this jungle of products, it can be difficult for the customers to find the products they are looking for. Since a traditional recommender system only consider users and products when generating recommendations, it often contributes to the feedback loop, where popular products get the most feedback in form of ratings, which further increases the popularity. This means that unless the customer is a "gray sheep", a person with very atypical taste, then typically only the few and popular products get recommended to the majority of customers. A context-aware recommender system can affect this feedback loop by recommending products based on the context that situates the customer, which gives more variety. Products that normally

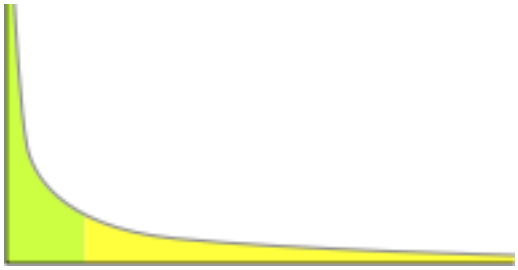


Figure 32: The Long Tail: An example of a power law graph showing popularity ranking. To the right is the long tail; to the left are the few that dominate. Notice that the areas of both regions match. Image courtesy of Wikipedia.com

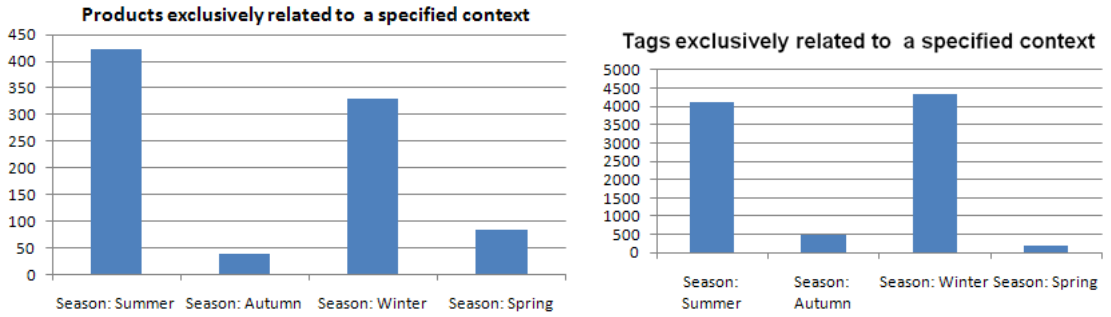
don't get recommended because they are aren't among the most popular ones, will have a greater chance of being recommended, and thus gain popularity.
Using the proposed contextual features, a context-aware recommender system can for instance sense that Christmas is upcoming, and start recommending products related to Christmas. If the system let the customers explain their purpose of their purchase in some way, then this information can be exploited to recommend gift ideas to different persons as well. Another scenario is being aware of the customers' location and season, and for instance get the weather forecast, and if it's summer and sunny; recommend a pair of waterskiis. If the customer is visiting during typical work hours or is a business customer, then it could recommend a desk fan instead.
Amazon.com and Ebay.com uses recommender systems as a targeted marketing tool in addition to the traditional way by just recommending products. They use recommendations for instance in newsletters in e-mail campaigns. It's possible that they in fact see recommender systems as being more valuable for the marketing purposes themselves, and that the accuracy of the recommendations is not necessarily top priority. McNee et al. claims that this race for developing the best algorithms that produces the best accuracies in different metrics have actually been detrimental to the field [26]. The popular "Netflix Price" is an example of this. This was a competition about developing the best recommender algorithm that would be used by the video streaming service company Netflix, where the winner would win one million dollar [27]. McNee et al. and Konstan further states that it's more important that a recommendation is meaningful to the user instead of just having a good accuracy on some metric. The users come to an online store because they reason for coming: they have a purpose [28]. This is a important aspect, and this is where context-aware recommender systems come into play: to provide more meaningful recommendations by taking the customers context into account.

8.2 Contextual features and their relevance

Season

Excluding usage, season was the contextual feature that got most of the tags. 82643 tags were collected, and divided over 7894 products. This accounts for 50 % of all the products in total. Using the average filter to get more valid classifications, the number of products is reduced to 1659 products (10.5 %). Summer was the season with most unique products. During the

analysis of the data, it seemed to exist a pattern that those products that were tagged as summer, often were tagged as spring and autumn additionally. 34 % of all seasons related products were actually tagged as spring, summer and autumn. Figure 33 shows the products and their tags for that are exclusively related to one specified season. A product is counted as exclusive if it's for instance tagged only as summer, and not as any other season.



(a) Products

(b) Products' tags

Figure 33: Products and their tags that are exlusively tagged to certain contexts. The "average filter" is applied.

What we see is that the majority of exclusive products and their tags are divided across two distinct classes with stark contrast to each other: summer and winter. This indicates that the level of granularity in season doesn't need to be more than two. A reduction means in this case increased generalization. The generalization will not have a large impact on season and it reduces complexity and source for disagreements when it comes to classifying the products.

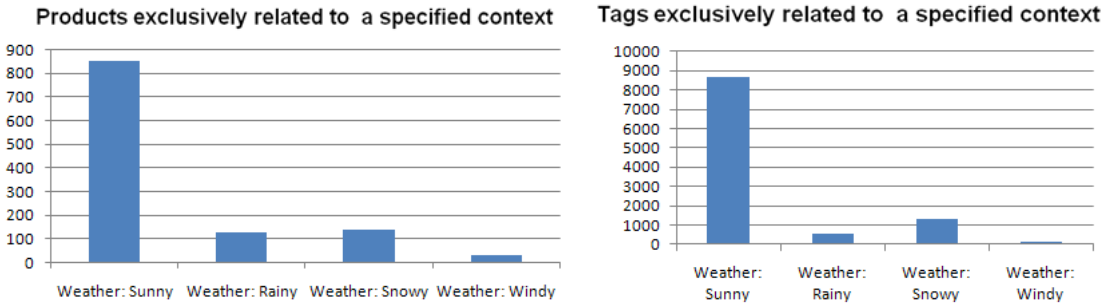
Weather

The contextual feature weather collected a total of 37630 tags across 6030 products. Unfiltered, this accounted for 38 % of all products. After the average filter is applied, the number of products is reduced to 1404 (9 %). The subclass sunny got the most the most tags, and had twice as many unique products compared to any other weather contexts. When it comes to exclusivity, figure 34 show that there is one subclass that stands out and that is sunny weather.

It seems logical to claim that the season summer and weather sunny are related, and can explain the large values for sunny. One might expect that this relationship would apply to winter and snowy in the same manner, but this isn't the case. This can possibly be explained that winter is more general than snowy. A heat pump for instance is related to winter, but not necessarily to snowy weather. When comparing the numbers for the different subclasses between season and weather, it seems to be only one weather subclass that shows a weak presence and that is windy. Sunny is the clear winner, but rainy and snowy could also be considered to be valid suggestions for contextual features.

Special day / Holiday

The contextual feature special day got the fewest tags in total, which was 13234, across 3138 products. Unfiltered, this accounted for 20 % of all products. After filtering, this number reduced



(a) Products (b) Products' tags

Figure 34: Products and their tags that are exlusively tagged to certain contexts. The "average filter" is applied.

to 971 (6 %). The subclass eastern had an average tag count of 1.48, and when obeying to the rounding rule, this was rounded down to 1 when used in the average filter. An average tag count of one is useless to the average filter since all tags are considered anyways. To be more comparable, eastern's average will be increased to two. What we see in figure 35 is that in this

(a) Products (b) Products' tags

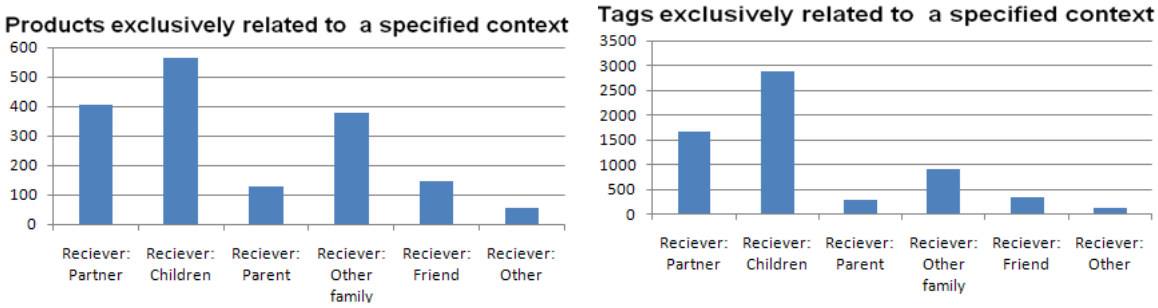
Figure 35: Products and their tags that are exlusively tagged to certain contexts. The "average filter" is applied.

case, two subclasses are considered popular, and that is Christmas and the generic class called others. The Christmas class can be useful in many cases. It exists an industry just for Christmas related products. The "other" class could be explored further, in case a higher granularity level can discover new useful contextual features that are now unknown. This being said, Christmas class do have more tags than "others", which means that the Christmas contextual feature offers higher validity in terms of being correctly classified.

Gift reciever

The gift receiver context explores more about the activity of the purchase. It gives information about to who this gift was. Moreover, we can see some indications about to which people do we buy gifts to. 60 % of all products have been gifted some someone at least one time. When finding

products that are exclusive to a certain receiver, we get information about to who we buy gifts to, but also some about the different niches that are popular for gifts. The subclass others have an tag count average of 1.33, and was adjusted to 2 for better comparisons. From figure 36 we



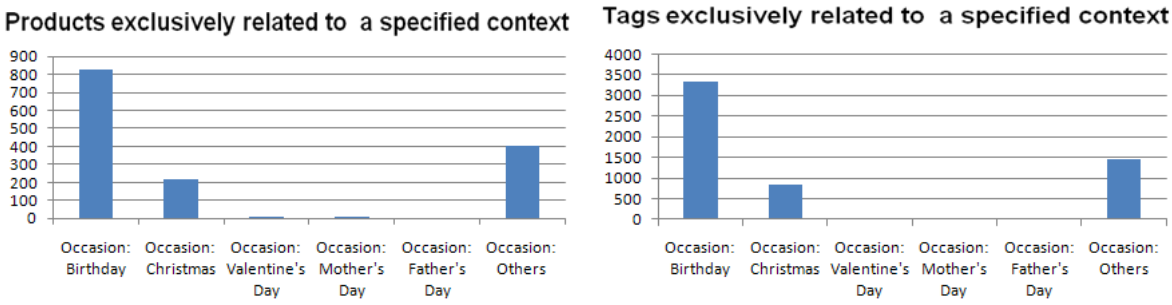
(a) Products (b) Products' tags

Figure 36: Products and their tags that are exlusively tagged to certain contexts. The "average filter" is applied.

see that three subclasses to receiver is noteworthy suggestions and those are partner, children and the generic subclass other family. "Others family" is comparable to partner in terms of number of products, but have only half the number of tags compared to partner, meaning less validity.

Gift occasion

The gift occasion context reveals the purpose about the gift. It gives the answer to why a customer bought a product to this other person. Of all the subclasses to occasion, it turns out that birthday is the most popular class. Second comes the generic "others" and third comes Christmas. Valentine's Day, Mother's Day and Father's Day have averages of 1.10, 1.21 and 1.09 respectively, which are not useful to be used in the average filter. Like the other contextual features in this position, they are set to 2 for comparable reasons. Even with and without filtering, figure 37



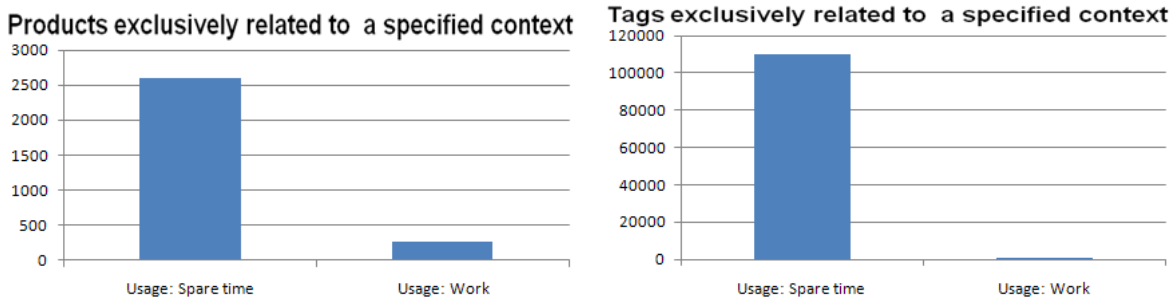
(a) Products (b) Products' tags

Figure 37: Products and their tags that are exlusively tagged to certain contexts. The "average filter" is applied.

shows that Valentine's Day, Mother's Day and Father's Day have practical no impact in this case and can't be considered being meaningful to be used in this setting.

Usage

Almost every product was tagged to one of the two usage types; spare time or work. When we look in figure 38 at those products that were exclusively tagged to the different types, we see that spare time was without doubt the most popular one.



(a) Products (b) Products' tags

Figure 38: Products and their tags that are exlusively tagged to certain contexts. The "average filter" is applied.

8.3 Granularity level in contextual features and its impact

Isolated scenarios

In the contextualized recommendation performance experiment, we saw how the context-aware recommender system performed using the different contextual features. Recall for the different contextual features ranged between 0.54 and 0.61, and a total average of 0.56 across all the contextual features. This means that a little more than half of all related products were retrieved. In recommender terminology; it was possible to generate recommendations using more than half of all related products that were available. This was however with customers that had ten ratings each. Since the Weighted Slope One algorithm is an item-based algorithm, it's in general able to generate more predictions for customers with many ratings compared to those with few. If all customers were used in the experiment, all logic based on item-based algorithms and results from experiments indicates that precision will be much more sensitive. Recall would also be lower, which finally results in a lower F-score. Even though only customers with ten ratings were used, the multiple scenario experiments showed that precision started to slightly decrease when multiple contextual features were used simultaneously. It then becomes clear that precision will get lower for customers with few available predictions even when only using one contextual feature. Figure 39 show the average number of predictions for the customers that have n ratings. The number of available predictions increased with the number of ratings a customer had.

Combined scenarios

The results from the experiment where different numbers of multiple contextual features were used, reveals a pattern regarding recall and thus F-score. They increase when we add more

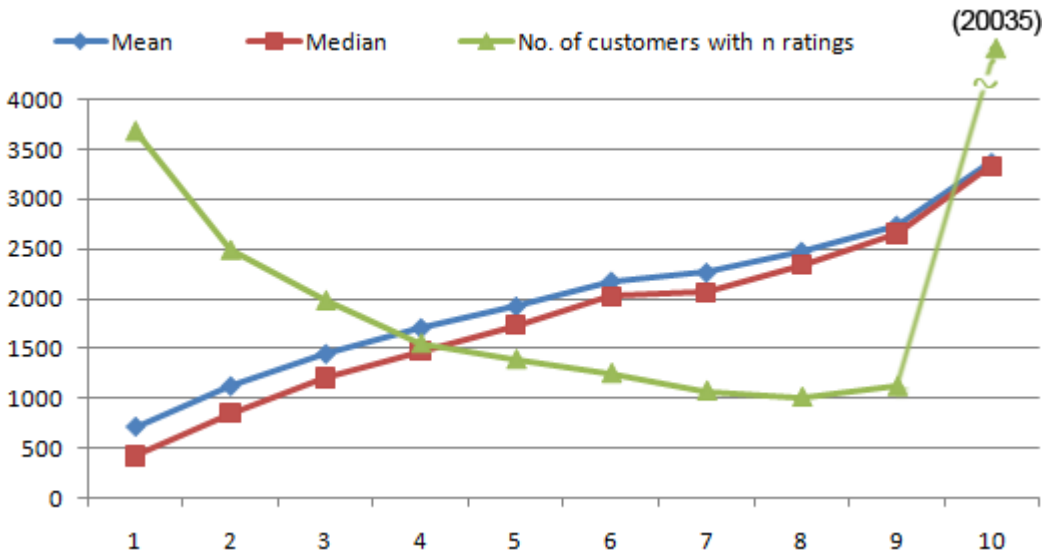


Figure 39: A graph showing the mean and median of predictions for the customers with n ratings.

contextual features that are used simultaneously. The reason for this is because for each contextual feature we add, fewer products that have been tagged and classified with that combination are available. The chance of retrieving more of them increases when the number of related products decreases.
If we only evaluate scenarios of the general types, where we for instance use all seasons, we start with 7894 products. Once we add another general type, the special day, this number decreases to 2088. Add receiver and occasion and it is reduced to 760. Add the last one to the combination, weather, and we end up with 481 products. Using all five in one combination, only 3 % of all products can be potentially recommended. And this is with customers with ten ratings. With customers with less ratings this number would have been lower.

Recommendation impact

A context-aware recommender system tries to suggest items based on the users situation. Customers coming to an online store have goals, they have a purpose behind the visit. Their purpose is based on their needs, which often is affected by the surrounding context. Season, location, weather, holidays and different types of relationships with others are some of the contexts that does this. Ideally, the recommender would offer recommendations that directly matches and fulfills the customers goals.
While using one contextual feature that filter out the recommendations, the customer can get a better impression on how the online store tries to help the customer. It doesn't presume that one size fits all. It can also be used as new way of using recommender systems in targeted marketing, for instance by sending an e-mail with Christmas related products that are recommended. This can be a benefit, targeted marketing becoming more automated than before, where it required sale staff manually picking off top of the line products which often led to a one size fits all approach. However, let's say a given context was the reason for a customer's visit in the first place

and is shown related recommendations. Just because the recommendations are related, doesn't mean that they are good recommendations. For instance, a customer get a book about gardening recommended because it summer. This book would never have appeared on the customer's list of recommended products, because it wasn't a good enough recommendation in the first place. But since the contextual feature works like a filter removing unrelated products, the book appears on the recommendation list because there aren't any better recommendations - the good ones are unrelated. And the more contextual features we use simultaneously, the greater this impact becomes. To counter-effect this, it's necessary to add more products with great variety. A greater quantity will offer more alternatives which could be better recommendations. The variety is good when it comes to which contextual features to use. These limitations is something one have to consider when developing and implementing a context-aware recommender system.
Since context-aware recommender systems considers the users context and thus increasing personalization, it increases the need to explain the recommendation given. Since contextual features reduces the number of available predictions, we may as mentioned end up with recommendations that may not be good, either at all or at first glance. The customer might not understand completely why this recommendation is present, the customer might not be aware of the intention behind the context-related recommendation. As Bilgic and Mooney states in [29], the goal of a good explanation should not be to "sell" the user on a recommendation, but rather to enable the user to make a more accurate judgment of the true quality of an item. This will improve user acceptance of the recommendations [30]. And this will lead to increased understanding of the recommendation, where the system explains that it tried to understand the user's goal by analyzing the context of the user.

9 Conclusion

Research Question 1: What are some of the contextual features that could be relevant to context-aware recommender systems that are used in e-commerce online stores that offers a great variety of products?

A set of contextual features were selected and studied to find out which ones that are reasonable to use in context-aware recommender systems. Since no previous work regarding this was found prior or during the work of this thesis, some initial contextual features had to be chosen from scratch. The foundation of the contextual features was based on the the work of Abowd et al.[10], who have the most cited definition of context and claim that the four most important primary context types are location, identity, activity and time. However, in the setting of recommender systems, it was found that those four weren't enough. They didn't describe the *intention*. For which purpose does a customer visit an online store? This thesis propose this new primary context type, and it was directly used as an example to explain why a customer gifted a given product to a given person.

The set of contextual sub-features started off with a relatively great variety. The customers tagged previously bought products to the different contextual features they found relevant. During the analysis, it became clear that something had to be done in order to get increased validity of the classifications. To do this in an objective way, the average tag count for each contextual feature was used as a cutoff filter. Those products who weren't tagged at least equivalent times as the average, were not classified as being related to the given contextual feature. The number of classified products was measured before and after using this method, including their tags. In addition, the number of exclusive classified products, which were products that only were classified to maximum one contextual sub-feature, was measured.

Which contextual features to actually use in a real life scenario, depends on the online store and the number and types of product categories. Some are however more general and relevant than others. The contextual features in figure 40 are the ones that stood out and had the biggest impact on the data gathered from Netthandelen.no, an auction-based online shop that sell products ranging over 190 categories.

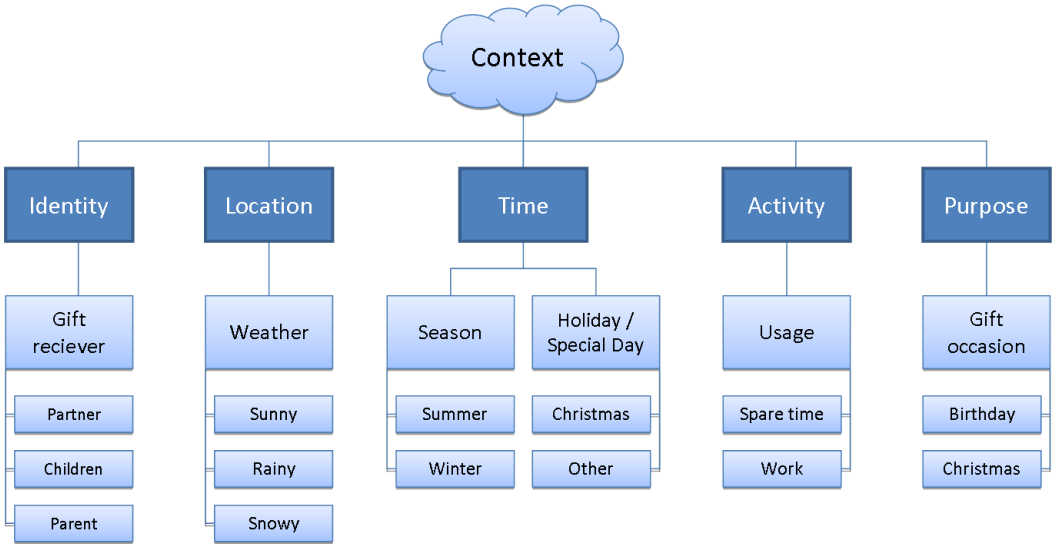


Figure 40: The conclusive contextual features with most impact are shown in this class diagram.

Research Question 2: Using a post-filtering approach, how does a coarse-grained versus a fine-grained use of contextual features affect the final recommendation outcome?

In this experiment, the results are based on customers with ten ratings. Those were selected because of two reasons: Most customers had ten ratings and to improve prediction performance. Scenarios using only one contextual feature showed acceptable performance. All customers could get context-related recommendations using any kind of contextual feature. On average, it was possible to generate recommendations using more than half of all related products available.
The scenarios that used multiple contextual features revealed implications. The higher granularity level of contextual features we use, the fewer recommendations becomes available. In addition, there is no guarantee that those recommendations that are available, are actual good ones. Online stores that sells a large range of products in many categories, can allow themselves to use a higher level of granularity for contextual features. The opposite applies to online stores with few products and categories.

10 Further work

In this thesis some initial contextual features were proposed to be used in context-aware recommender systems. To further increase the validity and relevance of them, a proposal for future work would be a survey. This survey can be divided into two parts. The first part would be to have some participants answering questions about how they would feel having contextual recommendations based on the proposed contextual features. Potentially new additions of contextual features could also be included. The second part could be an experiment where participants actually get some recommendations from a context-aware recommender system under different simulated scenarios using the proposed or new contextual features. They could then describe how they felt about them. Their answers from prior the experiment could then be compared to their answers after the experiment.

Another suggestion for future work is to directly compare prediction performance between a traditional recommender system and a context-aware one that uses a high level granularity of contextual features. This could help to see how better or worse the predictions are.

In this thesis, customers tagged the products to different contextual features during their participation of the survey. If it's wished that the customers should be responsible for the classifications on a regular basis, then some other possibilities should be considered. It would be challenging and perhaps unwise to ask, expect or demand that customers would do this explicitly, either directly at the online store website or by other contact methods like e-mail. Possibilities using explicit and implicit gathering methods could be considered. One implicit way to identify a product being bought as a gift, is to offer gift wrapping. If one want to know the receiver and the occasion for the gift, then it's possible to offer birthday cards as they often reveal both of these things.

Bibliography

[1] Manning, C. D., Raghavan, P., & Schtze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[2] Goy, A., Ardissono, L., & Petrone, G. 2007. Personalization in e-commerce applications. In *The Adaptive Web*, Brusilovsky, P., Kobsa, A., & Nejdl, W., eds, volume The Adaptive Web, chapter 16, 485–520. Springer-Verlag Berlin Heidelberg.

[3] Pazzani, M. & Billsus, D. 2007. Content-based recommendation systems. 325–341.

[4] Deshpande, M. & Karypis, G. 2004. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1), 143–177.

[5] Celma, O. *Music Recommendation and Discovery in the Long Tail*. PhD thesis.

[6] Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, 285–295, New York, NY, USA. ACM.

[7] Koutsabasis, P. & Darzentas, J. 2008. Item-based filtering and semantic networks for personalized web content adaptation in e-commerce. In *SETN '08: Proceedings of the 5th Hellenic conference on Artificial Intelligence*, 148–159, Berlin, Heidelberg. Springer-Verlag.

[8] Adomavicius, G. & Tuzhilin, A. April 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6), 734–749.

[9] Hussein, T., Linder, T., Gaulke, W., & Ziegler, J. 2009. Context-aware recommendations on rails. *CARS-2009*.

[10] Abowd, G. D., Dey, A. K., Brown, P., Davies, N., Smith, M., & Steggles, P. 1999. Towards a better understanding of context and context-awareness. In *HUC '99: Proceeding of the 1st international symposium on Handheld and Ubiquitous Computing*, 304–307. Springer.

[11] Adomavicius, G., Sankaranarayanan, R., Sen, S., & Tuzhilin, A. 2005. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems*, 23, 103–145.

[12] Adomavicius, G. & Tuzhilin, A. 2008. Context-aware recommender systems. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, 335–336, New York, NY, USA. ACM.

[13] Gorgoglione, M. & Panniello, U. 2009. Including context in a transactional recommender system using a pre-filtering approach: Two real e-commerce applications. In *WAINA '09: Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops*, 667–672, Washington, DC, USA. IEEE Computer Society.

[14] Panniello, U., Gorgoglione, M., & Palmisano, C. 2009. Comparing pre-filtering and post-filtering approach in a collaborative contextual recommender system: An application to e-commerce. In *EC-Web 2009: Proceedings of the 10th International Conference on E-Commerce and Web Technologies*, 348–359, Berlin, Heidelberg. Springer-Verlag.

[15] Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. 2007. Collaborative filtering recommender systems. In *The Adaptive Web*, Brusilovsky, P., Kobsa, A., & Nejdl, W., eds, volume The Adaptive Web, chapter 9, 291–324. Springer-Verlag Berlin Heidelberg.

[16] Lemire, D. & Maclachlan, A. 2005. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of SIAM Data Mining (SDM'05)*.

[17] Linden, G., Smith, B., & York, J. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 76–80.

[18] Breese, J. S., Heckerman, D., & Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. 43–52. Morgan Kaufmann.

[19] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. 2000. Analysis of recommendation algorithms for e-commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, 158–167, New York, NY, USA. ACM.

[20] Panniello, U., Tuzhilin, A., Gorgoglione, M., Palmisano, C., & Pedone, A. 2009. Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, 265–268, New York, NY, USA. ACM.

[21] Xiao, R., Hong, F., Xiong, J., Zheng, X., & Zhang, Z. 2009. Syncretizing context information into the collaborative filtering recommendation. *Database Technology and Applications, International Workshop on*, 0, 33–36.

[22] Tso-Sutter, K. H. L., Marinho, L. B., & Schmidt-Thieme, L. 2008. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, 1995–1999, New York, NY, USA. ACM.

[23] Raghavan, V., Bollmann, P., & Jung, G. S. 1989. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Trans. Inf. Syst.*, 7(3), 205–229.

[24] Globally unique identifiers (guids) - `http://msdn.microsoft.com/en-us/library/cc246025(v=PROT.10).aspx`. Accessed (May, 2010).

[25] Lemire, D. & McGrath, S. Implementing a rating-based item-to-item recommender system in php/sql. Technical Report D-01, Ondelette.com, January 2005.

[26] McNee, S. M., Riedl, J., & Konstan, J. A. 2006. Accurate is not always good: How accuracy metrics have hurt recommender systems.

[27] Netflix price - `http://www.netflixprize.com/`. Accessed (June, 2010).

[28] Joseph konstan on human-computer interaction - `http://www.acm.org/ubiquity/interviews/v6i10_konstan.html`. Accessed (June, 2010).

[29] Bilgic, M. & Mooney, R. J. 2005. Explaining recommendations: Satisfaction vs. promotion. In *In Proceedings of Beyond Personalization 2005, the Workshop on the Next Stage of Recommender Systems Research(IUI2005*, 13–18.

[30] Herlocker, J., Konstan, J. A., & Riedl, J. 2000. Explaining collaborative filtering recommendations. 241–250.

A Appendix

A.1 The survey website pages

Figure 41: The welcome page.

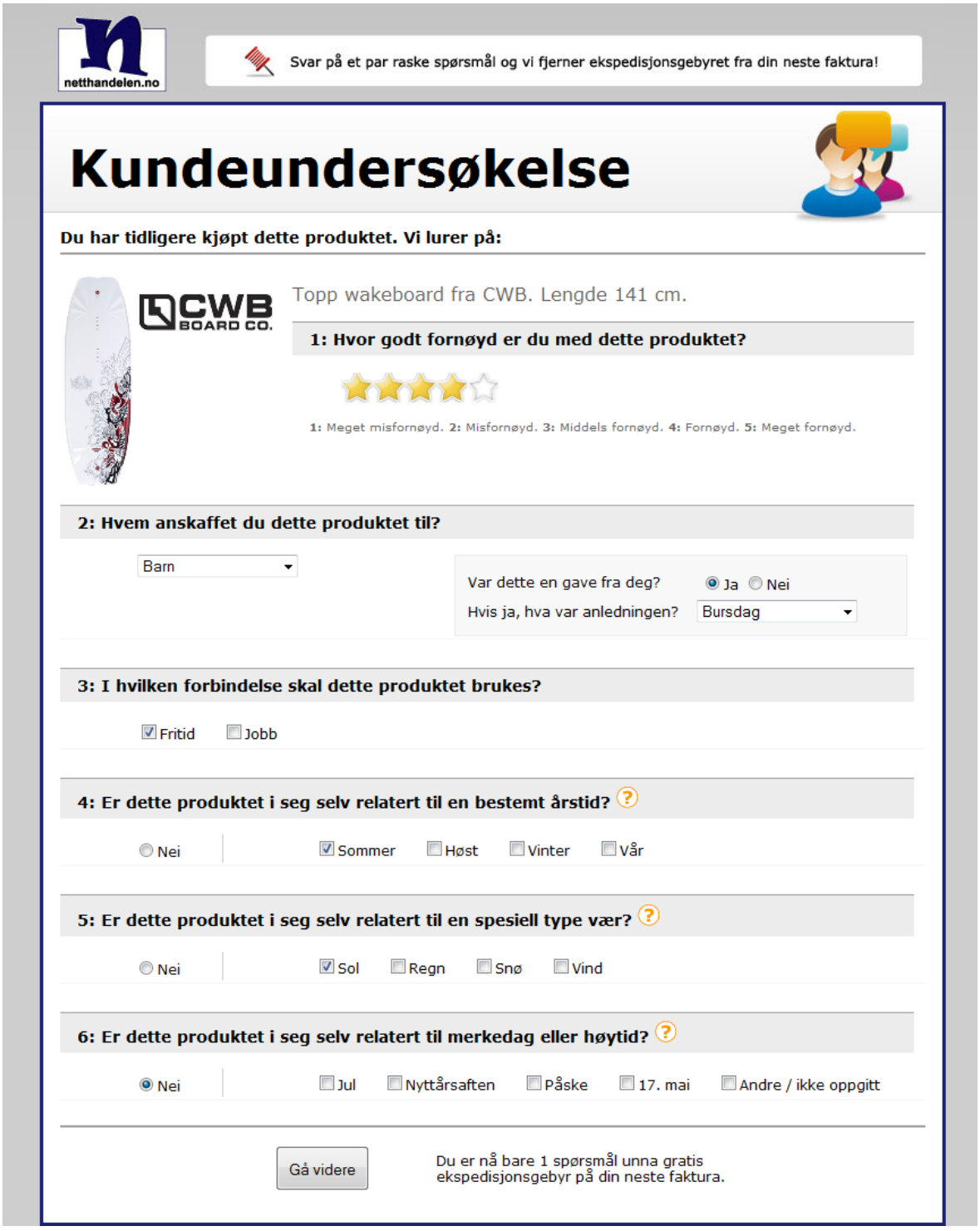


Figure 42: The product survey page.

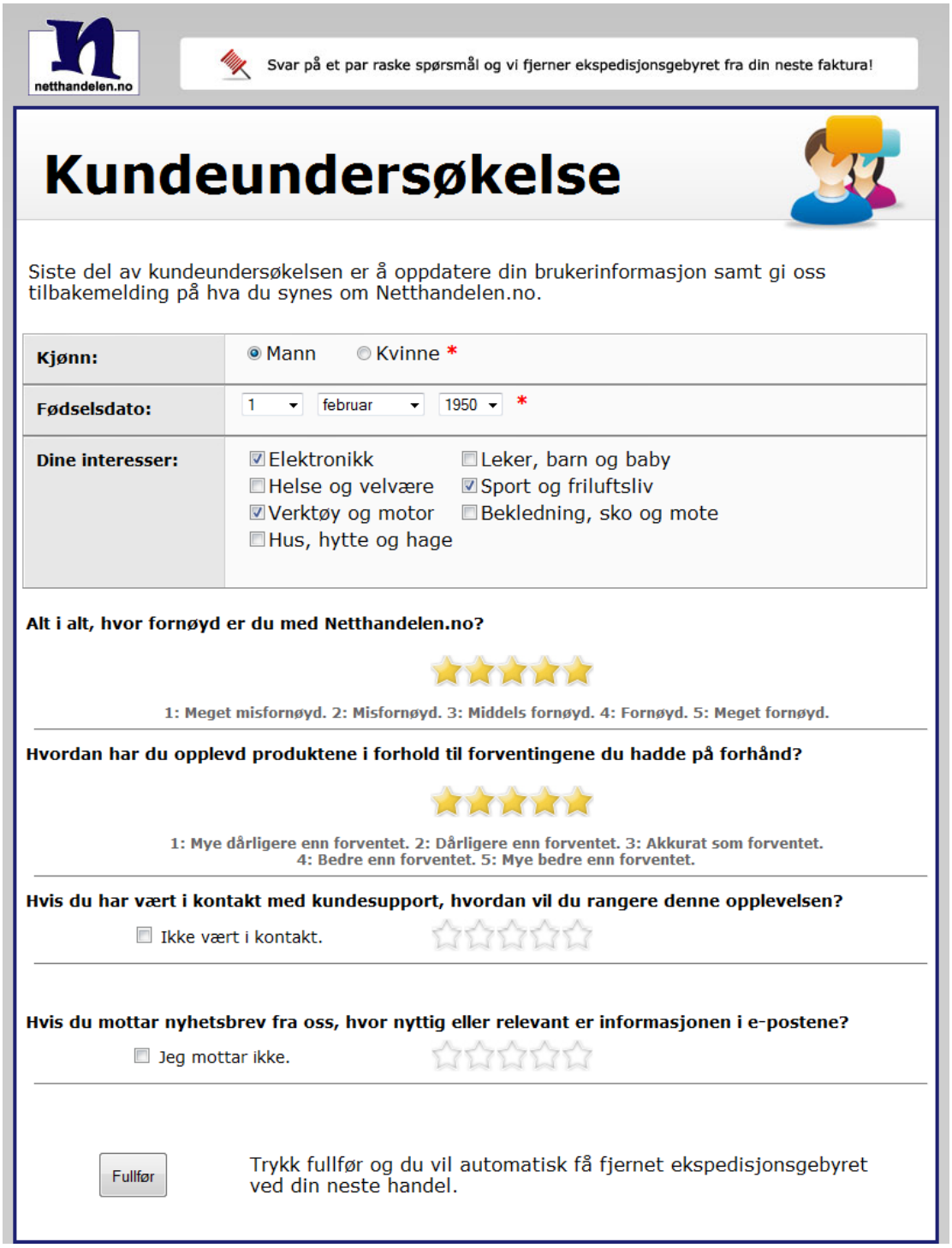
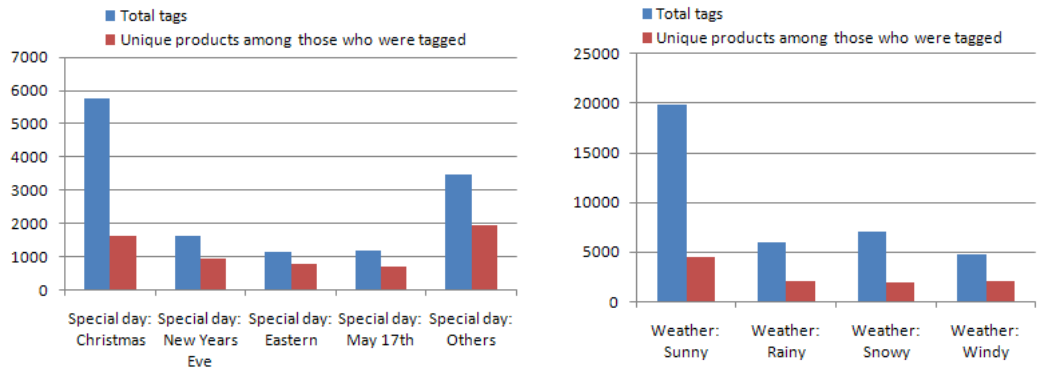


Figure 43: The reputation addon.



(a) Special Day / Holiday

(b) Weather

Figure 44: Tags and the unique products covered for special day / holiday and weather.

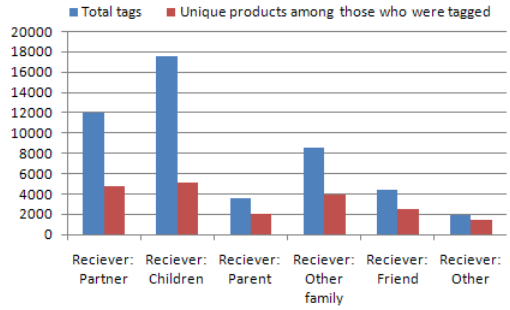
A.2 Weighted Slope One prediction performance

Test & training set #	MAE	MSE	RMSE	Normalized RMSE
1	0.2702	0.2074	0.4554	6.02 %
2	0.2719	0.2103	0.4585	6.01 %
3	0.2712	0.2079	0.4559	6.14 %
4	0.2678	0.2008	0.4481	5.79 %
5	0.2710	0.2054	0.4532	5.91 %
6	0.2680	0.2048	0.4525	6.11 %
7	0.2688	0.2046	0.4524	6.14 %
8	0.2744	0.2108	0.4591	6.04 %
9	0.2735	0.2168	0.4656	5.77 %
10	0.2712	0.2081	0.4562	5.96 %
Average	**0.2708**	**0.2077**	**0.4557**	**5.99 %**

Table 11: The Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and the Normalized RMSE for each test and training set and their averages.

A.3 Total tags and unique products

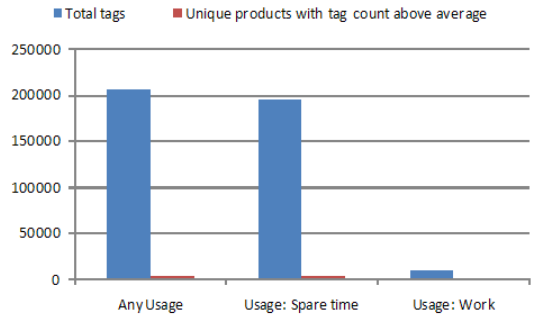
A.3.1 Unfiltered



(a) Gift Reciever

(b) Gift Occasion

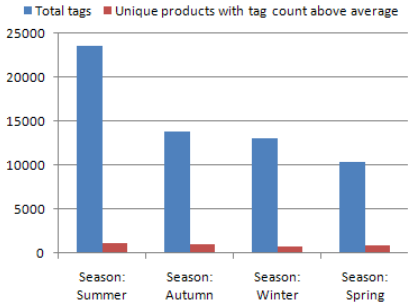
Figure 45: Tags and the unique products covered for gift reciever and occasion.



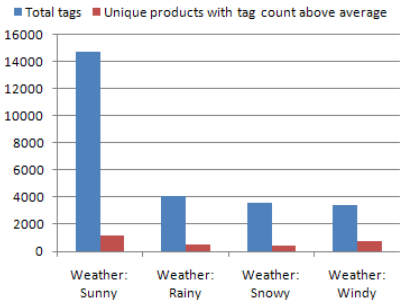
(a) Usage

(b) Special Day / Holiday

Figure 46: Tags and unique products related to usage and special day / holiday that have a greater tag count than the average.



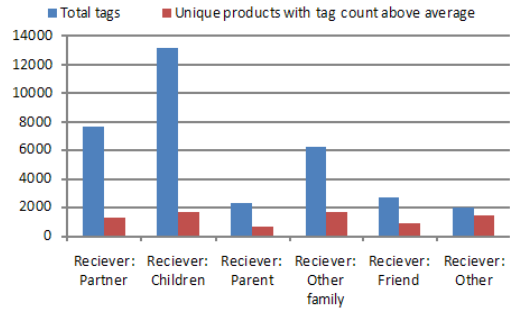
(a) Season



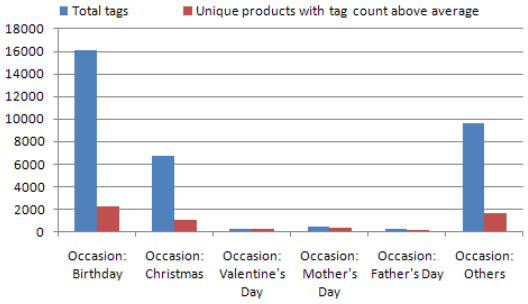
(b) Weather

Figure 47: Tags and the unique products related to season and weather that have a greater tag count than the average.

A.3.2 Filtered



(a) Gift Reciever



(b) Gift Occasion

Figure 48: Tags and the unique products related to gift reciever and occasion that have a greater tag count than the average.

A.3.3 Unfiltered tags

	Total tags	Unique products among all those who were tagged	% of total products
Any Season	**82643**	**7894**	**9,55 %**
Season: Summer	32307	5831	7,06 %
Season: Autumn	18554	4340	5,25 %
Season: Winter	16952	3175	3,84 %
Season: Spring	14830	4036	4,88 %
Any Weather	**37630**	**6030**	**7,30 %**
Weather: Sunny	19866	4549	5,50 %
Weather: Rainy	5952	2031	2,46 %
Weather: Snowy	7024	1961	2,37 %
Weather: Windy	4788	2104	2,55 %
Any Special day	**13234**	**3138**	**3,80 %**
Special day: Christmas	5781	1630	1,97 %
Special day: New Years Eve	1615	937	1,13 %
Special day: Eastern	1160	779	0,94 %
Special day: May 17th	1185	721	0,87 %
Special day: Others	3493	1964	2,38 %
Any gift Reciever	**47937**	**9552**	**11,56 %**
Reciever: Partner	11953	4726	5,72 %
Reciever: Children	17536	5086	6,15 %
Reciever: Parent	3587	1996	2,42 %
Reciever: Other family	8518	3923	4,75 %
Reciever: Friend	4384	2512	3,04 %
Reciever: Other	1959	1462	1,77 %
Any Occasion	**47937**	**9552**	**11,56 %**
Occasion: Birthday	21859	6732	8,15 %
Occasion: Christmas	10014	3553	4,30 %
Occasion: Valentine's Day	258	235	0,28 %
Occasion: Mother's Day	439	364	0,44 %
Occasion: Father's Day	220	202	0,24 %
Occasion: Others	15147	6007	7,27 %
Any Usage	**265085**	**15739**	**19,04 %**
Usage: Spare time	251711	15688	18,98 %
Usage: Work	13374	4240	5,13 %

Figure 49: Unfiltered data.

A.3.4 Filtered tags

	Average tag count	Total tags	Unique products with tag count above average	% of total products
Any Season	**10,467756**	**62824**	**1659**	**2,64 %**
Season: Summer	5,539622	23523	1140	1,81 %
Season: Autumn	4,274948	13793	1029	1,64 %
Season: Winter	5,340264	13018	709	1,13 %
Season: Spring	3,674597	10363	885	1,41 %
Any Weather	**6,239177**	**27620**	**1404**	**2,23 %**
Weather: Sunny	4,366974	14746	1124	1,79 %
Weather: Rainy	2,930049	4062	467	0,74 %
Weather: Snowy	3,582653	3588	380	0,60 %
Weather: Windy	2,27532	3396	714	1,14 %
Any Special day	**4,215492**	**10187**	**971**	**1,55 %**
Special day: Christmas	3,548189	4064	322	0,51 %
Special day: New Years Eve	1,720085	916	241	0,38 %
Special day: Eastern	1,489717	1159	778	1,24 %
Special day: May 17th	1,643055	622	159	0,25 %
Special day: Others	1,778909	2102	573	0,91 %
Any gift Reciever	**5,01717**	**35256**	**2631**	**4,19 %**
Reciever: Partner	2,529523	7595	1274	2,03 %
Reciever: Children	3,447984	13115	1595	2,54 %
Reciever: Parent	1,797493	2228	637	1,01 %
Reciever: Other family	2,171596	6188	1593	2,54 %
Reciever: Friend	1,744723	2701	831	1,32 %
Reciever: Other	1,334702	1950	1461	2,33 %
Any Occasion	**5,01717**	**35256**	**2631**	**4,19 %**
Occasion: Birthday	3,245877	16149	2295	3,65 %
Occasion: Christmas	2,818693	6773	1015	1,62 %
Occasion: Valentine's Day	1,09829	257	234	0,37 %
Occasion: Mother's Day	1,206043	439	364	0,58 %
Occasion: Father's Day	1,089108	220	202	0,32 %
Occasion: Others	2,521312	9584	1632	2,60 %
Any Usage	**16,820307**	**205026**	**3655**	**5,82 %**
Usage: Spare time	16,023203	194698	3647	5,81 %
Usage: Work	3,152158	9641	1316	2,09 %

Figure 50: Filtered data.

A.3.5 Exclusive classifications

		Unique products	Tags
Any Season			
Season: Summer	Season: Summer	423	4122
Season: Autumn	Season: Autumn	83	476
Season: Winter	Season: Winter	330	4333
Season: Spring	Season: Spring	40	178
Any Weather			
Weather: Sunny	Weather: Sunny	852	8662
Weather: Rainy	Weather: Rainy	125	555
Weather: Snowy	Weather: Snowy	137	1285
Weather: Windy	Weather: Windy	33	113
Any Special day			
Special day: Christmas	Special day: Christmas	183	1926
Special day: New Years Eve	Special day: New Years Eve	14	40
Special day: Eastern	Special day: Eastern	21	55
Special day: May 17th	Special day: May 17th	10	22
Special day: Others	Special day: Others	253	688
Any gift Reciever			
Reciever: Partner	Reciever: Partner	405	1676
Reciever: Children	Reciever: Children	565	2879
Reciever: Parent	Reciever: Parent	129	293
Reciever: Other family	Reciever: Other family	377	893
Reciever: Friend	Reciever: Friend	148	351
Reciever: Other	Reciever: Other	55	118
Any Occasion			
Occasion: Birthday	Occasion: Birthday	828	3347
Occasion: Christmas	Occasion: Christmas	219	855
Occasion: Valentine's Day	Occasion: Valentine's Day	2	5
Occasion: Mother's Day	Occasion: Mother's Day	3	6
Occasion: Father's Day	Occasion: Father's Day	0	0
Occasion: Others	Occasion: Others	406	1451
Any Usage			
Usage: Spare time	Usage: Spare time	2604	109702
Usage: Work	Usage: Work	273	1080

Figure 51: Products that were classified to maximum one subfeature within each main contextual feature.

A.3.6 Contextualized post-filtering performance

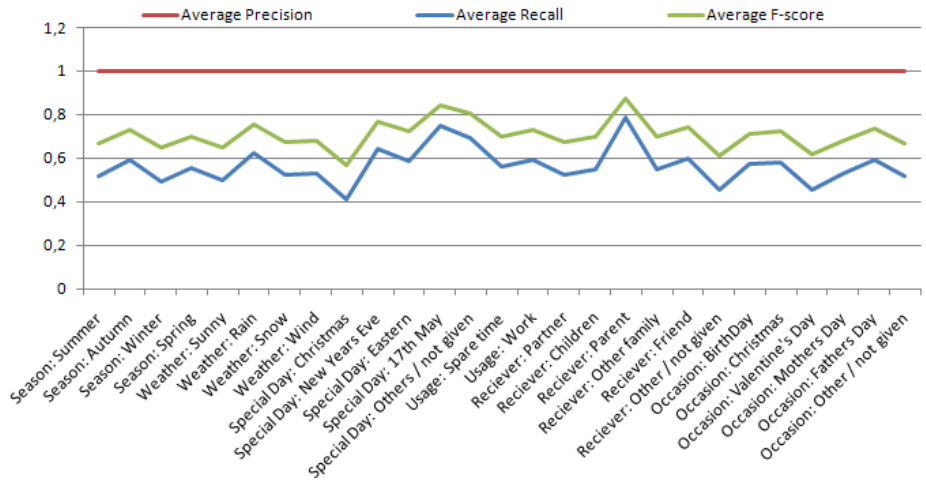


Figure 52: Performance for the different, single contextual features: Precision, Recall and F-score

A.3.7 Unique products vs minimum tag count

Unique products VS minimum tag count

Tag minimum count	Summer	Autumn	Winter	Spring
1	5830	4339	3174	4035
2	3268	2121	1615	1864
3	2882	1383	1122	1220
4	1726	1029	857	885
5	1367	841	709	690
6	1140	698	604	565
7	982	576	517	479
8	850	493	458	401
9	755	432	399	339
10	684	368	358	294
11	613	330	317	263
12	549	303	284	232
13	505	275	252	211
14	463	249	229	191
15	424	234	207	175
16	394	213	179	158
17	367	193	169	149
18	348	180	162	138
19	328	169	150	128
20	316	158	134	117
30	192	79	69	57
40	121	49	52	35
50	94	40	39	21
60	67	28	30	17
70	50	21	23	16
80	38	18	18	10
90	34	15	17	7
100	29	17	15	5
110	24	13	15	3
120	23	8	13	3
130	18	7	12	2
140	15	5	12	2
150	14	4	11	2
160	13	4	11	2
170	12	3	10	2
180	10	3	8	2
190	8	3	8	2
200	6	3	6	1
210	5	3	5	1
220	4	2	5	0
230	4	2	5	0
240	2	2	5	0
250	1	2	4	0

Figure 53: Number of unique products versus minimum tag count.

A.3.8 Contextualized recommendation performance

Context-Type ID	Average Tag Count, rounded	Unique products above average	Average Precision	Average Recall	Average F-score
Season: Summer	6,0	1045	1	0,516017352	0,66440864
Season: Autumn	4,0	821	1	0,593200632	0,7267821
Season: Winter	5,0	564	1	0,493060283	0,64653816
Season: Spring	3,0	964	1	0,556190774	0,69766082
Weather: Sunny	4,0	862	1	0,497119831	0,64880929
Weather: Rain	3,0	357	1	0,625800228	0,75478202
Weather: Snow	3,0	417	1	0,520309123	0,67061185
Weather: Wind	2,0	553	1	0,529082424	0,67820709
Special Day: Christmas	3,0	344	1	0,40739128	0,56873284
Special Day: New Years Eve	2,0	173	1	0,640614762	0,76909906
Special Day: Eastern	1,0	598	1	0,583768455	0,72343261
Special Day: 17th May	2,0	128	1	0,746421497	0,84192582
Special Day: Others / not given	2,0	409	1	0,694189081	0,80651524
Usage: Spare time	13,0	3454	1	0,558211592	0,69806648
Usage: Work	3,0	983	1	0,59402673	0,7297019
Reciever: Partner	2,0	1679	1	0,5223569	0,67321766
Reciever: Children	3,0	1280	1	0,550917418	0,69654368
Reciever: Parent	3,0	460	1	0,788970521	0,87342653
Reciever: Other family	2,0	1264	1	0,548893037	0,69652216
Reciever: Friend	2,0	640	1	0,599916084	0,73947736
Reciever: Other / not given	1,0	1205	1	0,451555338	0,61138365
Occasion: BirthDay	3,0	1813	1	0,57083207	0,71258106
Occasion: Christmas	3,0	810	1	0,581119337	0,72150457
Occasion: Valentine's Day	1,0	183	1	0,45499147	0,61510809
Occasion: Mothers Day	1,0	269	1	0,529911317	0,68140657
Occasion: Fathers Day	1,0	158	1	0,593694579	0,73311278
Occasion: Other / not given	2,0	2158	1	0,515477849	0,66693536

Figure 54: Precision, Recall and F-score for isolated scenarios.

Context-Type ID		Unique products that are tagged above average	Average Precision	Average Recall	Average F-score
Season	Season: Winter	3175	1	0,36957327	0,52842653
Season + SpecialDay	Season: Winter + Special Day: Christmas	925	1	0,39673731	0,5570535
Season + SpecialDay + ToWho	Season: Winter + Special Day: Christmas + Reciever: Partner	131	0,99995009	0,38800802	0,545149744
Season + SpecialDay + ToWho + Occasion	Season: Winter + Special Day: Christmas + Reciever: Partner + Occasion: Christmas	66	0,99910157	0,41145126	0,565447628
Season + SpecialDay + ToWho + Occasion + Weather	Season: Winter + Special Day: Christmas + Reciever: Partner + Occasion: Christmas + Weather: Snowy	27	0,9964562	0,44938025	0,600167129
Season	Season: Any	7894	1	0,27681578	0,424674937
Season + SpecialDay	Season: Any + Special Day: Any	2088	1	0,43961749	0,59844249
Season + SpecialDay + ToWho	Season: Any + Special Day: Any + Reciever: Any	760	1	0,44008454	0,599189655
Season + SpecialDay + ToWho + Occasion	Season: Any + Special Day: Any + Reciever: Any + Occasion: Any	760	1	0,44008454	0,599189655
Season + SpecialDay + ToWho + Occasion + Weather	Season: Any + Special Day: Any + Reciever: Any + Occasion: Any + Weather: Any	487	1	0,50627934	0,658711155

Figure 55: Precision, Recall and F-score for combined scenarios.

A.4 Source code snippets

A.4.1 SQL snippets

Stored procedures used in the survey

Below are a few of the stored procedures that were used by the survey application.

Listing A.1: Retrieves up to ten products bought by the customer.

```
/*
<summary>
Stored Procedure: usp_GetNextFakturaLinjeIDByMedlemsID

Usage: Returns a recordset containing maximum 10 FakturaLinjer (products).
If the customer has already rated some of the product then they are excluded.
Products that was purchased within the last 14 days are also excluded.
<param name="@MedlemsID">The customers MedlemsID</param>

Version: 0.1
</summary>
*/

CREATE PROC [dbo].[usp_GetNextFakturaLinjeIDByMedlemsID](
@MedlemsID int)

AS
SET NOCOUNT ON

BEGIN

DECLARE @CurrentCustomersRatingCount AS int


SELECT @CurrentCustomersRatingCount = COUNT(1) FROM dbo.ProductRating AS pr
  JOIN [NH].[dbo].[FakturaLinjer] AS fl ON pr.FakturaLinjeID = fl.FakturaLinjeID
  JOIN [NH].[dbo].[Fakturaer] AS f ON fl.FakturaID = f.FakturaID
  AND f.MedlemsID = @MedlemsID

-- Declare and set variables
DECLARE @Top AS int, @DateLimit AS datetime
SET @Top = 10 - @CurrentCustomersRatingCount --Customer will maximum rate about 10
    products
SET @DateLimit = GETDATE() - 14 --Customer will not rate products that was ordered
    within the last 14 days

SELECT TOP (@Top) fl.FakturaLinjeID, p.ProduktID, p.Navn, p.Bilde1
FROM [NH].[dbo].FakturaLinjer AS fl
LEFT JOIN [NH].[dbo].Produkter AS p
    ON fl.ProduktID = p.ProduktID
LEFT JOIN [NH].[dbo].Fakturaer AS f
    ON fl.FakturaID = f.FakturaID

WHERE f.MedlemsID = @MedlemsID
AND f.FakturaDato < @DateLimit
AND f.FakturaStatus in (4,7)
AND fl.ProduktType < 6

-- Make sure the customer hasn't already rated the product
AND NOT EXISTS (SELECT 1 FROM dbo.ProductRating AS pr
                    LEFT JOIN [NH].[dbo].[FakturaLinjer] AS fl2
                          ON pr.FakturaLinjeID = fl.FakturaLinjeID
                           LEFT JOIN [NH].[dbo].[Fakturaer] AS f2
```

```
52                                  ON fl2.FakturaID = f2.FakturaID
53                                WHERE fl.ProduktID = fl2.ProduktID
54                                 AND f2.MedlemsID = @MedlemsID)
55
56 -- Sort by descending purchasing price
57 ORDER BY fl.PrisEksMva DESC
58
59 END
```

Listing A.2: Insert or updates an existing record in the table that keeps track on the customers participation.

```
1  /*
2  <summary>
3  Stored Procedure: usp_CreateOrUpdateProductContextSurveyEntry
4
5  Usage: Insert or updates an existing record in the table ProductContextSurveyEntry,
       based on the input parameters.
6  <param name="@SurveyID">The ID of the current survey.</param>
7  <param name="@MedlemsID">The customers MedlemsID.</param>
8  <param name="@SurveyIsCompleted">If not null, it will tag the survey entry as
       completed</param>
9
10 Version: 0.1
11 </summary>
12 */
13
14 CREATE PROC [dbo].[usp_InsertOrUpdateProductContextSurveyEntry](
15 @SurveyID AS int,
16 @MedlemsID AS int,
17 @SurveyIsCompleted AS bit)
18
19 AS
20
21 /* Declare the ReturnValue. It will be to report the status back to the application
22  * 0 = Operation failed
23  * >0 = Operation success. Returns the ID of the record created or updated
24  */
25 DECLARE @ReturnVerdi AS int
26 SET @ReturnVerdi = 0
27
28 DECLARE @ExistingProductContextSurveyEntryID AS int
29
30 BEGIN
31 IF EXISTS (SELECT [ID] FROM [ProductContextSurveyEntry] WHERE [SurveyID] =
       @SurveyID AND [MedlemsID] = @MedlemsID)
32 BEGIN
33   -- Modes:
34   -- If @SurveyIsCompleted != null the survey is completed and we set the
         SurveyCompletedDate
35   -- If @IncrementProductsEvaluatedCount IS null AND @SurveyIsCompleted IS null,
         then increment the visitor count
36
37   IF @SurveyIsCompleted IS NOT NULL
38   BEGIN
39     UPDATE [ProductContextSurveyEntry]
40       SET [SurveyCompletedDate] = GETDATE()
41       WHERE [SurveyID] = @SurveyID AND [MedlemsID] = @MedlemsID
42   END
43   ELSE
44   BEGIN
45     UPDATE [ProductContextSurveyEntry]
```

```
46        SET [SurveyVisits] = [SurveyVisits] + 1
47        WHERE [SurveyID] = @SurveyID AND [MedlemsID] = @MedlemsID
48    END
49
50     IF @@ROWCOUNT IS NOT NULL
51     BEGIN
52      SET @ReturnVerdi = @@ROWCOUNT
53     END
54 END
55
56 ELSE
57
58 BEGIN
59    INSERT INTO [ProductContextSurveyEntry] ([SurveyID], [MedlemsID])
60         VALUES (@SurveyID, @MedlemsID)
61
62     IF SCOPE_IDENTITY() IS NOT NULL
63     BEGIN
64      SET @ReturnVerdi = SCOPE_IDENTITY()
65     END
66 END
67 RETURN @ReturnVerdi
68 END
```

Listing A.3: Check the customer's participating status.

```
1  /*
2  <summary>
3  Stored Procedure: usp_GetProductContextSurveyStatusForUserByMedlemsID
4
5  Usage: Returns a ReturnValue that describes the ProductContextSurvey status for a
       given customer
6  <param name="@MedlemsID">The customers MedlemsID</param>
7  <param name="@SurveyID">The ID of the ProductContextSurvey</param>
8
9  Version: 0.1
10 </summary>
11 */
12
13 CREATE PROC [dbo].[usp_GetProductContextSurveyStatusForUserByMedlemsID](
14 @MedlemsID int,
15 @SurveyID int)
16
17 AS
18
19 /* Declare the ReturnValue. It will be to report the status back to the application
20  *-1 = Couldn't find anything about the given customer
21  * 0 = Customer has not completed the survey
22  * 1 = Customer has completed the survey
23  */
24 DECLARE @ReturnValue AS int
25 SET @ReturnValue = -1
26
27 BEGIN
28
29   --Check first if the customer has activated this survey, regardless of completion
30   IF EXISTS(SELECT [ID] FROM [dbo].[ProductContextSurveyEntry] WHERE [MedlemsID] =
         @MedlemsID AND [SurveyID] = @SurveyID)
31   BEGIN
32     --Customer has activated the survey
33     --Check if customer has completed the survey or not
```

```
34      IF EXISTS(SELECT [ID] FROM [dbo].[ProductContextSurveyEntry] WHERE [MedlemsID]
            = @MedlemsID AND [SurveyID] = @SurveyID AND [SurveyCompletedDate] IS NOT
            NULL)
35      BEGIN
36        --Customer has completed the survey
37        SET @ReturnValue = 1
38      END
39      ELSE
40      BEGIN
41        --Customer has not completed the survey
42        SET @ReturnValue = 0
43      END
44
45    END
46
47    RETURN @ReturnValue
48
49  END
```

Listing A.4: Updates the customers' interests.

```
1  /*
2  <summary>
3  Stored Procedure: usp_InsertMedlemmerNyhetsGrupperFromCSV
4
5  Usage: Insert new rows in MedlemmerNyhetsGrupper, assigning a relationship between
       a customer and a NewsGroup, where the [NyhetsGrupper].NyhetsgruppeId
6  is contained in a CSV varchar which is parsed, and is identified by the MedlemsID.
       All pre-existing relationship records belonging to the user is deleted.
7  <param name="@MedlemsID">The MedlemsID</param>
8  <param name="@NyhetsgruppeIdCSV">The NyhetsgruppeId that the user selected,
       contained in a CSV varchar</param>
9
10 Version: 0.1
11 </summary>
12 */
13
14 CREATE PROC [dbo].[usp_InsertMedlemmerNyhetsGrupperFromCSV](
15 @MedlemsID int,
16 @NyhetsgruppeIdCSV varchar(8000))
17
18 AS
19
20 /* Declare the ReturnValue. It will be to report the status back to the application
21  * 0 = Operation failed
22  * 1 = Operation success.
23  */
24 DECLARE @ReturnValue AS int
25 SET @ReturnValue = 0
26
27 BEGIN
28
29   -- Create temp table in memory that will contain the NyhetsGruppeIds that the
         user selected
30   DECLARE @tmpNyhetsgrupperIds       TABLE (ID int identity(1,1) NOT NULL,
         NyhetsgruppeId varchar(80))
31
32   --Start the transaction
33   BEGIN TRANSACTION
34
```

```
35  -- Parse the CSV varchar that contains the NyhetsgruppeIds, seperated by a ',’.
        Insert the result into the temp table
36  INSERT INTO @tmpNyhetsgrupperIds
37  SELECT s FROM dbo.ufn_ParseString(@NyhetsgruppeIdCSV)
38
39  IF @@ERROR <> 0
40  BEGIN
41    -- Rollback the transaction
42    ROLLBACK
43    -- Raise an error and return
44    RAISERROR (                                                                 ,
          16, 1)
45    RETURN
46    END
47
48    --Delete all pre-existing records belonging to this customer in the
          MedlemmerNyhetsgrupper table
49    DELETE FROM [NH].[dbo].[MedlemmerNyhetsGrupper] WHERE [MedlemsID] = @MedlemsID
50  IF @@ERROR <> 0
51  BEGIN
52    ROLLBACK
53    RAISERROR (
                  , 16, 1)
54    RETURN
55    END
56
57
58  -- Get the last ID created from the temp table and store the id, which will be
        used in the while loop as the limit
59  DECLARE @LastTempTableID int
60  SET @LastTempTableID = SCOPE_IDENTITY()
61
62  -- Start while loop, iterating through the new NyhetsGruppeIds that we have
        parsed and extracted into the temp table from the CSV varchar input
63  -- But only start it if we actually got new NyhetsGrupper to insert
64  IF @LastTempTableID IS NOT NULL
65  BEGIN
66    DECLARE @i as INT, @NyhetsgruppeId as INT
67    SET @i = 1
68    WHILE @i <= @LastTempTableID
69    BEGIN
70      -- Get the NyhetsgruppeId
71      SELECT @NyhetsgruppeId = [NyhetsgruppeId] FROM @tmpNyhetsgrupperIds WHERE [ID
            ] = @i
72
73      -- Insert into MedlemmerNyhetsGrupper, the ReturnValue is 1 if successful, 0
            otherwise
74      EXEC @ReturnValue = usp_InsertMedlemmerNyhetsGrupper @MedlemsID = @MedlemsID,
            @NyhetsgruppeId = @NyhetsgruppeId
75
76      IF @ReturnValue = 0 OR @@ERROR <> 0
77      BEGIN
78        ROLLBACK
79        RAISERROR (
                                                                                , 16, 1)
80        RETURN
81      END
82
83      -- Increment loop iterator
84      SET @i = @i + 1
85    END
86  END
```

```
87    ELSE
88    BEGIN
89      -- The CSV was empty, thus no insertions were done. This is allowed.
90      SET @ReturnValue = 1
91    END
92
93
94   -- Everything OK, commit the transaction.
95   COMMIT
96
97   -- Last check to see if the transaction itself was successful
98   IF @@ERROR <> 0
99   BEGIN
100    SET @ReturnValue = 0
101  END
102
103  RETURN @ReturnValue
104  END
```

Total tags and unique products

Below are a few of the SQL scripts that were used during the analysis.

Listing A.5: Returns the average tag count for products related to summer

```
1 SELECT AVG(CAST(derived.theCount AS dec)) FROM (SELECT fl.ProduktID, COUNT(fl.
      ProduktID) AS theCount
2   FROM ProductSeasonRelated AS psr
3   JOIN FakturaLinjer AS fl ON fl.FakturaLinjeID = psr.FakturaLinjeID
4   WHERE psr.SeasonID = 1 AND fl.ProduktID > 0
5   GROUP by fl.ProduktID) AS derived
```

Listing A.6: Returns the total tag count for each of the products related to summer

```
1 SELECT fl.ProduktID, COUNT(fl.ProduktID) AS theCount
2 FROM ProductSeasonRelated AS psr
3 JOIN FakturaLinjer AS fl ON fl.FakturaLinjeID = psr.FakturaLinjeID
4 WHERE psr.SeasonID = 1 AND fl.ProduktID > 0
5 GROUP BY fl.ProduktID
6 /* Uncomment to find the total tags of products that have at least 10 tags
7 * HAVING COUNT(fl.ProduktID) >= 10
8 */
9 ORDER BY theCount DESC
```

Listing A.7: Returns the total number of unique products that have been tagged a certain amount of times for products related to summer

```
1 SELECT COUNT(1) FROM (SELECT DISTINCT fl.ProduktID
2 FROM ProductSeasonRelated AS psr
3 JOIN FakturaLinjer AS fl ON fl.FakturaLinjeID = psr.FakturaLinjeID
4 WHERE psr.SeasonID = 1 AND fl.ProduktID > 0
5 GROUP BY fl.ProduktID
6 HAVING COUNT(fl.ProduktID) = 1) AS derived
```

Listing A.8: Returns the total number of unique products that might have been misclassified as being related to summer.

```
SELECT COUNT(DISTINCT derived.ProduktID) FROM
(SELECT f.MedlemsID, fl.ProduktID, psr.SeasonID, COUNT(fl.produktid) AS
    DuplicateProductTags FROM ProductSeasonRelated psr
JOIN FakturaLinjer fl on fl.FakturaLinjeID = psr.FakturaLinjeID
JOIN Fakturaer f on f.FakturaID = fl.FakturaID
WHERE fl.ProduktID > 0 and psr.SeasonID = 1
GROUP BY f.MedlemsID, fl.ProduktID, psr.SeasonID
HAVING COUNT(fl.produktid) >= @averageTagCount) AS derived
```

Listing A.9: The main SQL script that starts the procedure to calculate precision & recall and F-score for all customers within the specified context and threshold.

```
SET NOCOUNT ON

DECLARE @Threshold int, @SeasonID int
SET @Threshold = 6
SET @SeasonID = 1

DECLARE @SeasonThresholdCount int
SELECT @SeasonThresholdCount = dbo.udf_Season_ThreshHold_Count(@SeasonID,
    @Threshold)

DECLARE @tabellen TABLE
(
  UserID int,
  TP float,
  FN float,
  Prec float,
  Recall float,
  Fscore float
)

INSERT INTO @tabellen (UserID, TP)
SELECT derived.UserID,
dbo.udf_Season_TP(derived.UserID, @SeasonID, @Threshold) as TP
FROM (SELECT DISTINCT rm.UserID FROM RatingMatrix AS rm) AS derived

UPDATE @tabellen SET FN = @SeasonThresholdCount - TP,
Prec = CASE WHEN TP > 0 OR TP < 0 THEN TP / TP ELSE 0 END,
Recall = TP / @SeasonThresholdCount
UPDATE @tabellen SET  Fscore = (CASE WHEN Prec + Recall > 0 OR Prec + Recall < 0
    THEN 2 * ((Prec * Recall)/(Prec + Recall)) ELSE 0 END)

INSERT INTO PrecRecSeasonSummer (UserID, TP, FN, Prec, Recall, Fscore)
SELECT tab.UserID, tab.TP, tab.FN, tab.Prec, tab.Recall, tab.Fscore
FROM @tabellen AS tab
```

Listing A.10: A scalar-valued function used to return the number of products being related to the specified context.

```
CREATE FUNCTION [dbo].[udf_Season_ThreshHold_Count]
(
@SeasonID int,
@ThreshHold int
)
RETURNS int
AS
BEGIN

```

```
10 DECLARE @SeasonThreshHoldCount int
11
12 SELECT @SeasonThreshHoldCount = COUNT(1) FROM (SELECT rm.ItemID, COUNT(rm.ItemID)
       AS summen
13 FROM ProductSeasonRelated AS psr
14 JOIN RatingMatrix AS rm
15 ON rm.FakturaLinjeID = psr.FakturaLinjeID
16
17 WHERE psr.SeasonID = @SeasonID
18
19 GROUP BY rm.ItemID
20 HAVING COUNT(rm.ItemID) >= @ThreshHold) AS derived
21
22 RETURN @SeasonThreshHoldCount
23
24 END
```

Listing A.11: A scalar-valued function used to return the number of true positives used in the calculation of precision and recall

```
1  CREATE FUNCTION [dbo].[udf_Season_TP]
2  (
3  @UserID int,
4  @SeasonID int,
5  @ThreshHold int
6  )
7  RETURNS int
8  AS
9  BEGIN
10 DECLARE @TruePositive int
11
12
13 SELECT @TruePositive = COUNT(1)
14 FROM (select derived.item FROM (SELECT d.ItemID1 AS item, CAST(SUM(d.Sum + d.Count
       * r.Rating) AS decimal) / SUM(d.Count) AS avgrat
15   FROM RatingMatrix r, Deviation d
16   WHERE r.UserID = @UserID
17   AND d.ItemID1 <> r.ItemID
18   AND d.ItemID2 = r.ItemID
19   GROUP BY d.ItemID1) AS derived
20   JOIN Produkter AS p
21   ON p.ProduktID = derived.item
22
23   JOIN (SELECT rm.ItemID
24     FROM ProductSeasonRelated as psr
25     JOIN RatingMatrix as rm
26     ON rm.FakturaLinjeID = psr.FakturaLinjeID
27     WHERE psr.SeasonID = @SeasonID
28     GROUP BY rm.ItemID
29     HAVING COUNT(rm.ItemID) >= @ThreshHold) AS derived2
30   ON derived2.ItemID = derived.item) AS finalDerived
31
32 RETURN @TruePositive
33
34 END
```

A.4.2 Survey application C# snippets

Listing A.12: The webpage in the survey where the customers rated and tagged the products.

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Web;
6  using System.Web.UI;
7  using System.Web.UI.WebControls;
8  using SurveyLib.ProductContextSurvey;
9  using SurveyLib.ProductContextSurvey.Entities;
10
11 /// <summary>
12 /// Page: ProductSurvey.aspx
13 /// Version: 0.3
14 /// </summary>
15 public partial class _ProductSurvey : System.Web.UI.Page
16 {
17
18     protected void Page_Load(object sender, EventArgs e)
19     {
20         //Get the GUID from the query string and get the MedlemsID. If it is null
               or empty, set local GUID var to null
21         string GUID = !string.IsNullOrEmpty(Request.QueryString["guid"]) ? Request.
               QueryString["guid"] : null;
22
23         //Only continue if we got a valid GUID
24         if (GuidUtility.IsGuid(GUID))
25         {
26             //GUID is valid
27             //Get information of the user that owns the GUID
28             Medlem medlem = new Medlem().Load(GUID);
29
30             //If the Medlem.ID is zero, it means that we didn't find any user with
                   the provided GUID.
31             if (medlem.ID > 0)
32             {
33                 //The user existed
34
35                 //Save the medlemsID into view state for optimization reasons, it
                       will be directly used in submit event. It will only be read
                       from viewstate directly on postback, and since we set this on
                       every page load, the chance of XSS or other tampering will be
                       dramatically reduced / eliminated
36                 ViewState["MedlemsID"] = medlem.ID;
37
38                 if (!IsPostBack) //First page load
39                 {
40                     //Check if the customer has previously completed the survey
41                     int surveyID = 1; //manually set for now
42                     bool? isProductContextSurveyCompletedForCustomer = new
                           UserSurveyTracking().
                           IsProductContextSurveyCompletedForCustomer(medlem.ID,
                           surveyID);
43                     if (isProductContextSurveyCompletedForCustomer != null && (bool
                           )(!isProductContextSurveyCompletedForCustomer))
44                     {
45                         List<FakturaLinje> fakturaLinjer = new FakturaLinje().
                               GetNextUnratedFakturaLinjer(medlem.ID);
46
47                         //Load first product for first page load
48                         //If for some reason no products were returned, then send
                               the customer to the error page with an explanation
49                         if (fakturaLinjer == null || fakturaLinjer.Count == 0)
```

```
50                          {
51                              Response.Redirect("Feil.aspx?error=" + Server.UrlEncode
                                    ("Beklager, det viser seg at du har mottatt denne
                                    spoerreundersoekelsen ved en feiltakelse. Vi har
                                    proevd aa filtrere saa godt som mulig paa forhaand,
                                     men en sjelden gang slipper noen faa gjennom. Vi
                                    beklager for forstyrrelsen. Ha en fin dag videre!
                                    Beskjeden nedenfor kan du se bort i fra."), true);
52                          }
53                          LoadProduct(fakturaLinjer[0].Produkt, fakturaLinjer[0].ID);
54
55                          //Count the number of uniquely bought products for the
                                customer. The product limit will be used in a TOP
                                clause
56                          SurveyStepStatus(fakturaLinjer.Count);
57                      }
58                      else if (isProductContextSurveyCompletedForCustomer != null &&
                            (bool)(isProductContextSurveyCompletedForCustomer))
59                      {
60                          //The customer has already completed this survey at an
                                earlier time!
61                          TransferToCustomerSurveyPage();
62                      }
63                      else
64                      {
65                          //isProductContextSurveyCompletedForCustomer is null, no
                                information about the survey for this customer was
                                found
66                          Response.Redirect("Feil.aspx?error=" + Server.UrlEncode("Du
                                 maa starte kundeundersoekelsen fra begynnelsen av."));
67                      }
68                  }
69              }
70              else
71              {
72                  //Response.Write("No user found!");
73                  Response.Redirect("Feil.aspx?error=" + Server.UrlEncode("Gjenkjente
                        ingen brukere med de opplysningene som ble gitt."));
74              }
75          }
76          else
77          {
78              //Not a valid GUID, the URL was probably altered by the user
79              //Response.Write("Not a valid GUID");
80              Response.Redirect("Feil.aspx?error=" + Server.UrlEncode("En ugyldig
                    identifikasjonskode var oppgitt."));
81          }
82      }
83
84      private int SurveyStepStatus(int productsLimit)
85      {
86          litProductsRemaining.Text = productsLimit.ToString();
87
88          return productsLimit;
89      }
90
91      protected void ResetControls()
92      {
93          //Reset controls
94          ddlGiftToWho.SelectedIndex = -1;
95          rblWasItAGift.SelectedIndex = 0; //Yes
96          ddlGiftOccasion.SelectedIndex = -1; //Bursdag
```

```
97          cblProductUsage.SelectedIndex = -1;
98          rbSeasonNo.Checked = false;
99          cblSeason.SelectedIndex = -1;
100         rbWeatherNo.Checked = false;
101         cblWeather.SelectedIndex = -1;
102         rbSpecialDayNo.Checked = false;
103         cblSpecialDay.SelectedIndex = -1;
104
105         hdRating.Value = "";
106     }
107
108     protected void TransferToCustomerSurveyPage()
109     {
110         Server.Transfer("CustomerSurvey.aspx");
111     }
112
113     protected void LoadProduct(Produkt produkt, int thisFakturaLinjeID)
114     {
115         //Load and view the product from FakturaLinjeID
116
117         imgProductImage.ImageUrl = "http://www.netthandelen.no/prodbilder/medium/"
                + produkt.Bildel;
118         imgProductImage.Height = 190;
119         imgProductImage.Width = 190;
120
121         litProductName.Text = produkt.Navn;
122
123         //Store the FakturaLinjeID for the product loaded, which will be used to
                save
124         hdThisFLID.Value = thisFakturaLinjeID.ToString();
125     }
126
127
128     protected void btnNextProduct_Click(object sender, EventArgs e)
129     {
130         if (string.IsNullOrEmpty(hdRating.Value))
131         {
132             pnError.Visible = true;
133             return;
134         }
135         else pnError.Visible = false;
136
137         //Save the entered information about the product
138         FakturaLinje fakturaLinje = new FakturaLinje();
139         fakturaLinje.ID = Convert.ToInt32(hdThisFLID.Value);
140
141         Produkt produkt = new Produkt();
142         ProductContext productContext = new ProductContext();
143
144         //Add the product rating
145         productContext.Rating = Convert.ToInt32(hdRating.Value);
146
147         //Only add gift context if this product was a gift
148         if (!ddlGiftToWho.SelectedItem.Text.Equals("Meg") && rblWasItAGift.
                SelectedItem.Text.Equals("Ja"))
149             productContext.AddGiftContext(Convert.ToInt32(ddlGiftToWho.
                    SelectedValue), Convert.ToInt32(ddlGiftOccasion.SelectedValue));
150
151         //Add selected usage context
152         foreach (ListItem item in cblProductUsage.Items) if (item.Selected)
                productContext.AddUsageContext(Convert.ToInt32(item.Value));
153
```

```
154         //Add selected season context
155         foreach (ListItem item in cblSeason.Items) if (item.Selected)
                productContext.AddSeasonContext(Convert.ToInt32(item.Value));
156
157         //Add selected weather context
158         foreach (ListItem item in cblWeather.Items) if (item.Selected)
                productContext.AddWeatherContext(Convert.ToInt32(item.Value));
159
160         //Add selected special day context
161         foreach (ListItem item in cblSpecialDay.Items) if (item.Selected)
                productContext.AddSpecialDayContext(Convert.ToInt32(item.Value));
162
163         //Add the product context object to the parent Produkt object
164         produkt.Context = productContext;
165
166         //Add the Produkt object to the parent FakturaLinje object
167         fakturaLinje.Produkt = produkt;
168
169         //Store the product context to the database
170         bool transactionSuccess = fakturaLinje.SubmitContextualProductData();
171
172
173
174         //Load the next products, if any. If Count = 0, then user is complete with
                the survey.
175         //First, get the customers MedlemsID from ViewState, it is set directly
                before this event is loaded, on every Page_Load
176         int medlemsID = 0;
177         if (ViewState["MedlemsID"] == null || !Int32.TryParse(ViewState["MedlemsID
                "].ToString(), out medlemsID) || medlemsID <= 0)
178         {
179             //ViewState containing MedlemsID is null or couldn't be parsed to int,
                    which means it somehow got tampered with
180             //TODO: ERROR & LOG HANDLING
181             //Response.Redirect("http://www.netthandelen.no", true);
182             Response.Redirect("Feil.aspx?error=" + Server.UrlEncode("En ugyldig
                    informasjon var forsoekt angitt."));
183         }
184
185         //Valid MedlemsID
186         List<FakturaLinje> fakturaLinjer = new FakturaLinje().
                GetNextUnratedFakturaLinjer(medlemsID);
187         if (fakturaLinjer.Count == 0)
188         {
189             //Confirm that the user has completed the survey.
190             bool sqlSuccess = new UserSurveyTracking().ProductContextSurveyEntry(1,
                     medlemsID, true);
191             TransferToCustomerSurveyPage();
192         }
193         else
194         {
195             //More products remains, survey is not completed.
196             //Reset the forms to default
197             ResetControls();
198
199             //Load next product
200             LoadProduct(fakturaLinjer[0].Produkt, fakturaLinjer[0].ID);
201
202             //Update navigation status
203             int productsLimit = fakturaLinjer.Count;
204             SurveyStepStatus(productsLimit);
205         }
```

```
206         }
207 }
```

Listing A.13: This class had the responsibility of retrieving products and storing contextual information about the products.

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Configuration;
5  using System.Data;
6  using System.Data.SqlClient;
7  using System.Linq;
8  using System.Transactions;
9  using System.Web;
10
11
12 namespace SurveyLib.ProductContextSurvey.Entities
13 {
14     /// <summary>
15     /// Class: FakturaLinje
16     /// Version: 0.1
17     /// </summary>
18     public class FakturaLinje
19     {
20         public int ID { get; set; }
21         public Produkt Produkt { get; set; }
22
23
24         private readonly string ConnString = new ConnectionStringUtility().
               GetConnectionString();
25
26
27         public FakturaLinje()
28         {
29             //
30             // TODO: Add constructor logic here
31             //
32         }
33
34         /// <summary>
35         /// Gets the number of number of products bought by the customer.
36         /// Maximum 10 products are counted, and products that have been ordered in
                the past 14 days are excluded from the count.
37         /// </summary>
38         /// <param name="medlemsID">The customers MedlemsID</param>
39         public int CountBoughtProductsForUser(int medlemsID)
40         {
41             using (SqlConnection conn = new SqlConnection(ConnString))
42             {
43                 //Add the product rating
44                 using (SqlCommand cmd = new SqlCommand("[dbo].[
                        usp_GetNumberOfBoughtProductsByMedlemsID]", conn))
45                 {
46                     cmd.CommandType = CommandType.StoredProcedure;
47
48                     //Input: @MedlemsID
49                     SqlParameter parameter = new SqlParameter("@MedlemsID",
                            SqlDbType.Int);
50                     parameter.Direction = ParameterDirection.Input;
51                     parameter.Value = medlemsID;
```

```
52                      cmd.Parameters.Add(parameter);
53
54                      //Output: @ReturnValue
55                      parameter = new SqlParameter("@ReturnValue", SqlDbType.Int);
56                      parameter.Direction = ParameterDirection.ReturnValue;
57                      cmd.Parameters.Add(parameter);
58
59                      conn.Open();
60                      cmd.ExecuteNonQuery();
61
62                      return Convert.ToInt32(cmd.Parameters["@ReturnValue"].Value);
63                  }
64              }
65          }
66
67          /// <summary>
68          /// Gets a List of FakturaLinje, containing up to 10 rows from
                FakturaLinjer. The rows are selected as long the FakturaLinjeID isn't
                in ProductRating, this means that the customer has rated the
                FakturaLinjeID.
69          /// We don't want the customer to rate the same FakturaLinjeID more than
                once. In addition, products that was bought within the last 14 days are
                 also excluded. And the recordset is ordered as FakturaLinje.PrisEksMva
                 DESC.
70          /// </summary>
71          /// <param name="medlemsID">The customers MedlemsID</param>
72          public List<FakturaLinje> GetNextUnratedFakturaLinjer(int medlemsID)
73          {
74              List<FakturaLinje> fakturaLinjer = new List<FakturaLinje>();
75
76              using (SqlConnection conn = new SqlConnection(ConnString))
77              {
78                  //Add the product rating
79                  using (SqlCommand cmd = new SqlCommand("[dbo].[
                        usp_GetNextFakturaLinjeIDByMedlemsID]", conn))
80                  {
81                      cmd.CommandType = CommandType.StoredProcedure;
82
83                      //Input: @MedlemsID
84                      SqlParameter parameter = new SqlParameter("@MedlemsID",
                            SqlDbType.Int);
85                      parameter.Direction = ParameterDirection.Input;
86                      parameter.Value = medlemsID;
87                      cmd.Parameters.Add(parameter);
88
89                      conn.Open();
90                      SqlDataReader dr = cmd.ExecuteReader();
91                      while (dr.Read())
92                      {
93                          fakturaLinjer.Add(new FakturaLinje()
94                                                  {
95                                                      ID = Convert.ToInt32(dr["
                                                            FakturaLinjeID"]),
96                                                      Produkt = new Produkt()
97                                                                  {
98                                                                      ID = Convert.
                                                                            ToInt32(dr
                                                                            ["ProduktID
                                                                            "]),
99                                                                      Navn = dr["Navn
                                                                            "].ToString
                                                                            (),
```

```
100                                                                     Bilde1 = dr["
                                                                            Bilde1"].
                                                                            ToString()
101                                                                 }
102                                                 });
103                     }
104                 }
105             }
106
107             return fakturaLinjer;
108         }
109
110         /// <summary>
111         /// Stores the given contextual data about a product from the user into the
                database
112         /// </summary>
113         public bool SubmitContextualProductData()
114         {
115                 using (SqlConnection conn = new SqlConnection(ConnString))
116                 {
117                     //Add the product rating
118                     using (SqlCommand cmd = new SqlCommand("[
                            usp_InsertProductRating]", conn))
119                     {
120                         cmd.CommandType = CommandType.StoredProcedure;
121
122                         //Input: @FakturaLinjeID
123                         SqlParameter parameter = new SqlParameter("@FakturaLinjeID
                                ", SqlDbType.Int);
124                         parameter.Direction = ParameterDirection.Input;
125                         parameter.Value = ID;
126                         cmd.Parameters.Add(parameter);
127
128                         //Input: @Rating
129                         parameter = new SqlParameter("@Rating", SqlDbType.Int);
130                         parameter.Direction = ParameterDirection.Input;
131                         parameter.Value = Produkt.Context.Rating;
132                         cmd.Parameters.Add(parameter);
133
134                         conn.Open();
135                         cmd.ExecuteNonQuery();
136                     }
137                 }
138
139                 //Only run usp_InsertProductGiftRelated and
                        usp_InsertProductGiftOccasionRelated if we got any data about
                        it
140                 if (Produkt.Context.GiftContext != null)
141                 {
142                     using (SqlConnection conn = new SqlConnection(ConnString))
143                     {
144
145                         using (SqlCommand cmd = new SqlCommand("[
                                usp_InsertProductGiftRelated]", conn))
146                         {
147                             cmd.CommandType = CommandType.StoredProcedure;
148
149                             //Input: @FakturaLinjeID
150                             SqlParameter parameter = new SqlParameter("
                                    @FakturaLinjeID", SqlDbType.Int);
151                             parameter.Direction = ParameterDirection.Input;
152                             parameter.Value = ID;
```

```
153                             cmd.Parameters.Add(parameter);
154
155                             //Input: @ToWhoID
156                             parameter = new SqlParameter("@ToWhoID", SqlDbType.Int)
                                    ;
157                             parameter.Direction = ParameterDirection.Input;
158                             parameter.Value = Produkt.Context.GiftContext.ToWhoID;
159                             cmd.Parameters.Add(parameter);
160
161                             conn.Open();
162                             cmd.ExecuteNonQuery();
163                         }
164
165                     }
166
167                     using (SqlConnection conn = new SqlConnection(ConnString))
168                     {
169                         using (SqlCommand cmd = new SqlCommand("[
                                usp_InsertProductGiftOccasionRelated]", conn))
170                         {
171                             cmd.CommandType = CommandType.StoredProcedure;
172
173                             //Input: @FakturaLinjeID
174                             SqlParameter parameter = new SqlParameter("
                                    @FakturaLinjeID", SqlDbType.Int);
175                             parameter.Direction = ParameterDirection.Input;
176                             parameter.Value = ID;
177                             cmd.Parameters.Add(parameter);
178
179                             //Input: @OccasionID
180                             parameter = new SqlParameter("@OccasionID", SqlDbType.
                                    Int);
181                             parameter.Direction = ParameterDirection.Input;
182                             parameter.Value = Produkt.Context.GiftContext.
                                    OccasionID;
183                             cmd.Parameters.Add(parameter);
184
185                             conn.Open();
186                             cmd.ExecuteNonQuery();
187                         }
188                     }
189                 }
190
191
192
193             //Add selected usage context
194                 foreach (ProductUsageContext productUsageContext in Produkt.Context
                        .ProductUsageContext)
195                 {
196                     using (SqlConnection conn = new SqlConnection(ConnString))
197                     {
198                         using (SqlCommand cmd = new SqlCommand("[
                                usp_InsertProductUsageRelated]", conn))
199                         {
200                             cmd.CommandType = CommandType.StoredProcedure;
201
202                             //Input: @FakturaLinjeID
203                             SqlParameter parameter = new SqlParameter("
                                    @FakturaLinjeID", SqlDbType.Int);
204                             parameter.Direction = ParameterDirection.Input;
205                             parameter.Value = ID;
206                             cmd.Parameters.Add(parameter);
```

```
207
208                             //Input: @UsageID
209                             parameter = new SqlParameter("@UsageID", SqlDbType.Int)
                                    ;
210                             parameter.Direction = ParameterDirection.Input;
211                             parameter.Value = productUsageContext.UsageID;
212                             cmd.Parameters.Add(parameter);
213
214                             conn.Open();
215                             cmd.ExecuteNonQuery();
216                         }
217                     }
218                 }
219
220                 //Add selected season context
221                 foreach (ProductSeasonRelatedContext productSeasonRelatedContext in
                         Produkt.Context.ProductSeasonRelatedContext)
222                 {
223                     using (SqlConnection conn = new SqlConnection(ConnString))
224                     {
225                         using (SqlCommand cmd = new SqlCommand("[
                                usp_InsertProductSeasonRelated]", conn))
226                         {
227                             cmd.CommandType = CommandType.StoredProcedure;
228
229                             //Input: @FakturaLinjeID
230                             SqlParameter parameter = new SqlParameter("
                                    @FakturaLinjeID", SqlDbType.Int);
231                             parameter.Direction = ParameterDirection.Input;
232                             parameter.Value = ID;
233                             cmd.Parameters.Add(parameter);
234
235                             //Input: @SeasonID
236                             parameter = new SqlParameter("@SeasonID", SqlDbType.Int
                                    );
237                             parameter.Direction = ParameterDirection.Input;
238                             parameter.Value = productSeasonRelatedContext.SeasonID;
239                             cmd.Parameters.Add(parameter);
240
241                             conn.Open();
242                             cmd.ExecuteNonQuery();
243                         }
244                     }
245                 }
246
247
248                 //Add selected weather context
249                 foreach (ProductWeatherRelatedContext productWeatherPelatedContext
                         in Produkt.Context.ProductWeatherRelatedContext)
250                 {
251                     using (SqlConnection conn = new SqlConnection(ConnString))
252                     {
253                         using (SqlCommand cmd = new SqlCommand("[
                                usp_InsertProductWeatherRelated]", conn))
254                         {
255                             cmd.CommandType = CommandType.StoredProcedure;
256
257                             //Input: @FakturaLinjeID
258                             SqlParameter parameter = new SqlParameter("
                                    @FakturaLinjeID", SqlDbType.Int);
259                             parameter.Direction = ParameterDirection.Input;
260                             parameter.Value = ID;
```

```
261                                 cmd.Parameters.Add(parameter);
262
263                                 //Input: @WeatherID
264                                 parameter = new SqlParameter("@WeatherID", SqlDbType.
                                        Int);
265                                 parameter.Direction = ParameterDirection.Input;
266                                 parameter.Value = productWeatherPelatedContext.
                                        WeatherID;
267                                 cmd.Parameters.Add(parameter);
268
269                                 conn.Open();
270                                 cmd.ExecuteNonQuery();
271                             }
272                         }
273                     }
274
275                     //Add selected special day context
276                     foreach (ProductSpecialDayRelatedContext
                            productSpecialDayPelatedContext in Produkt.Context.
                            ProductSpecialDayRelatedContext)
277                     {
278                         using (SqlConnection conn = new SqlConnection(ConnString))
279                         {
280                             using (SqlCommand cmd = new SqlCommand("[
                                    usp_InsertProductSpecialDayRelated]", conn))
281                             {
282                                 cmd.CommandType = CommandType.StoredProcedure;
283
284                                 //Input: @FakturaLinjeID
285                                 SqlParameter parameter = new SqlParameter("
                                        @FakturaLinjeID", SqlDbType.Int);
286                                 parameter.Direction = ParameterDirection.Input;
287                                 parameter.Value = ID;
288                                 cmd.Parameters.Add(parameter);
289
290                                 //Input: @SpecialDayID
291                                 parameter = new SqlParameter("@SpecialDayID", SqlDbType
                                        .Int);
292                                 parameter.Direction = ParameterDirection.Input;
293                                 parameter.Value = productSpecialDayPelatedContext.
                                        SpecialDayID;
294                                 cmd.Parameters.Add(parameter);
295
296                                 conn.Open();
297                                 cmd.ExecuteNonQuery();
298                             }
299                         }
300                     }
301
302                 return true;
303             }
304         }
305 }
```

Listing A.14: This class had contained the methods for adding contextual information to the products.

```
1
2 using System;
3 using System.Collections.Generic;
4 using System.Configuration;
5 using System.Data;
```

```
6  using System.Linq;
7  using System.Web;
8
9
10 namespace SurveyLib.ProductContextSurvey.Entities
11 {
12     /// <summary>
13     /// Class: ProductContext
14     /// Version: 0.1
15     /// </summary>
16     public class ProductContext
17     {
18         public int Rating { get; set; }
19         public ProductGiftRelatedContext GiftContext { get; set; }
20         public List<ProductUsageContext> ProductUsageContext { get; set; }
21         public List<ProductSeasonRelatedContext> ProductSeasonRelatedContext { get;
               set; }
22         public List<ProductWeatherRelatedContext> ProductWeatherRelatedContext {
               get; set; }
23         public List<ProductSpecialDayRelatedContext>
               ProductSpecialDayRelatedContext { get; set; }
24
25         private readonly string ConnString = new ConnectionStringUtility().
               GetConnectionString();
26
27
28         public ProductContext()
29         {
30             //Initiate a default ProductUsageContext list
31             ProductUsageContext = new List<ProductUsageContext>();
32             ProductSeasonRelatedContext = new List<ProductSeasonRelatedContext>();
33             ProductWeatherRelatedContext = new List<ProductWeatherRelatedContext>()
                   ;
34             ProductSpecialDayRelatedContext = new List<
                   ProductSpecialDayRelatedContext>();
35         }
36
37         /// <summary>
38         /// Adds gift context of the product.
39         /// </summary>
40         /// <param name="toWhoID">The description ID of the gift receiver.</param>
41         /// <param name="occasionID">The description ID of the gift occasion.</
               param>
42         public void AddGiftContext(int toWhoID, int occasionID)
43         {
44             ProductGiftRelatedContext ProductGiftRelatedContext = new
                   ProductGiftRelatedContext();
45             ProductGiftRelatedContext.ToWhoID = toWhoID;
46             ProductGiftRelatedContext.OccasionID = occasionID;
47
48             GiftContext = ProductGiftRelatedContext;
49         }
50
51         /// <summary>
52         /// Adds usage context of the product.
53         /// </summary>
54         /// <param name="usageID">The description ID of the usage type.</param>
55         public void AddUsageContext(int usageID)
56         {
57             ProductUsageContext productUsageContext = new ProductUsageContext();
58             productUsageContext.UsageID = usageID;
59
```

```
60              ProductUsageContext.Add(productUsageContext);
61          }
62
63          /// <summary>
64          /// Adds season related context of the product.
65          /// </summary>
66          /// <param name="seasonID">The description ID of the season.</param>
67          public void AddSeasonContext(int seasonID)
68          {
69              ProductSeasonRelatedContext productSeasonRelatedContext = new
                    ProductSeasonRelatedContext();
70              productSeasonRelatedContext.SeasonID = seasonID;
71
72              ProductSeasonRelatedContext.Add(productSeasonRelatedContext);
73          }
74
75          /// <summary>
76          /// Adds weather related context of the product.
77          /// </summary>
78          /// <param name="weatherID">The description ID of the weather type.</param>
79          public void AddWeatherContext(int weatherID)
80          {
81              ProductWeatherRelatedContext productWeatherRelatedContext = new
                    ProductWeatherRelatedContext();
82              productWeatherRelatedContext.WeatherID = weatherID;
83
84              ProductWeatherRelatedContext.Add(productWeatherRelatedContext);
85          }
86
87          /// <summary>
88          /// Adds special day related context of the product.
89          /// </summary>
90          /// <param name="specialDayID">The description ID of the special day.</
                param>
91          public void AddSpecialDayContext(int specialDayID)
92          {
93              ProductSpecialDayRelatedContext productSpecialDayRelatedContext = new
                    ProductSpecialDayRelatedContext();
94              productSpecialDayRelatedContext.SpecialDayID = specialDayID;
95
96              ProductSpecialDayRelatedContext.Add(productSpecialDayRelatedContext);
97          }
98      }
99  }
```

A.4.3 Weighted Slope One algorithm and cross-fold validation in MATLAB

Listing A.15: Function that is used when the ratings are parsed from text file into a matrix.

```
1  function ratingMatrix = addRating(ratingMatrix, userID, itemID, rating)
2
3  % Add the rating into the rating matrix
4  rowSize = size(ratingMatrix,1);
5  ratingMatrix(rowSize + 1,1) = userID;
6  ratingMatrix(rowSize + 1,2) = itemID;
7  ratingMatrix(rowSize + 1,3) = rating;
8
9  end
```

Listing A.16: Parses the text file containing all the ratings

```
clear all;

% Init the rating matrix and the deviation matrix
ratingMatrix = []; %[ActiveUserID, ItemID, Rating]
deviationMatrix = []; %[ItemID1, ItemID2, Count, Sum]

tStart = tic;

%numUsersLimit = 100;

fid=fopen(                                );
currUser=0;
numUsers=0;
while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
    numLine=sscanf(tline,    );
    activeUserID = numLine(1);
    itemID = numLine(2);
    rating = numLine(3);
    ratingMatrix = addRating(ratingMatrix, activeUserID, itemID, rating);
    %uncomment to activate user calculation limit
    %{
    if currUser~=numLine(1),
        numUsers=numUsers+1;
        currUser=numLine(1);
    end
    if numUsers>numUsersLimit, break;end
    %}
end

tElapsedLoadRatingMatrix = uint64(toc(tStart));
disp([                                                    ,  num2str(
    tElapsedLoadRatingMatrix)]);

tStart = tic;

currentDeviationSize = 0;

 for i = 1:size(ratingMatrix,1)
    [ratingMatrix, deviationMatrix, ratingDifferenceMatrix] =
        popularityDifferential(ratingMatrix, deviationMatrix, ratingMatrix(i,1),
        ratingMatrix(i,2), ratingMatrix(i,3));

    % Periodically display current computation time
    if (size(deviationMatrix,1) > currentDeviationSize + 99999)
        currentDeviationSize = size(deviationMatrix,1);
        tElapsedPopularityDifferential = uint64(toc(tStart));
        disp([                          ,  num2str(tElapsedPopularityDifferential)
            ]);
        disp([                                ,  num2str(currentDeviationSize)]);
    end

 end
 % Show calculation time
 tElapsedPopularityDifferential = uint64(toc(tStart));
 disp([                                                    ,  num2str(
     tElapsedPopularityDifferential)]);
```

Listing A.17: Calculates the popularity differentials that are stored in the deviation matrix.

```
function [ratingMatrix, deviationMatrix, ratingDifferenceMatrix] =
    popularityDifferential(ratingMatrix, deviationMatrix, activeUserID, itemID,
    rating)


% Get all of the user's rating pairs
% First, find all the indicies of the current user from the rating matrix
indActiveUserInRatingMatrix = find(ratingMatrix(:,1) == activeUserID);

% Generate the matrix that shows the rating differene to the other items
% the user has previously rated
ratingDifferenceMatrix = [ratingMatrix(indActiveUserInRatingMatrix,2), rating -
    ratingMatrix(indActiveUserInRatingMatrix,3)];

% Look though the ratingDifferenceMatrix
for i = 1:size(ratingDifferenceMatrix,1)
    % For every one of the user's rating pairs, update the deviation table
    otherItemID = ratingDifferenceMatrix(i,1);
    ratingDifference = ratingDifferenceMatrix(i,2);

    % If the pair (itemID, otherItemID) is already in the dev table, then
    % we want to update 2 rows
    indDeviation = [];
    if (size(deviationMatrix,1) > 0)
        indDeviation = find(deviationMatrix(:,1) == itemID & deviationMatrix(:,2)
            == otherItemID);
    end

    if (size(indDeviation,1) > 0)
       deviationMatrix(indDeviation,3) = deviationMatrix(indDeviation,3) + 1;
       deviationMatrix(indDeviation,4) = deviationMatrix(indDeviation,4) +
           ratingDifference;

       % We only want to update if the items are different
       if (itemID ~= otherItemID)
           indDeviation = find(deviationMatrix(:,1) == otherItemID &
               deviationMatrix(:,2) == itemID);
           deviationMatrix(indDeviation,3) = deviationMatrix(indDeviation,3) + 1;
           deviationMatrix(indDeviation,4) = deviationMatrix(indDeviation,4) -
               ratingDifference;
       end

    else %We want to insert 2 rows into the dev table
       %We only want to insert 2 rows into the dev table
       if (itemID ~= otherItemID)
          rowSize = size(deviationMatrix,1);
          deviationMatrix(rowSize + 1,1) = itemID;
          deviationMatrix(rowSize + 1,2) = otherItemID;
          deviationMatrix(rowSize + 1,3) = 1;
          deviationMatrix(rowSize + 1,4) = ratingDifference;
          %Note the +2 instead of +1, because we adding the "inverse"-row
          deviationMatrix(rowSize + 2,1) = otherItemID;
          deviationMatrix(rowSize + 2,2) = itemID;
          deviationMatrix(rowSize + 2,3) = 1;
          deviationMatrix(rowSize + 2,4) = -ratingDifference;
       end
    end


end
```

Listing A.18: Does the cross-fold validations and calculate the different error rates for the predictions that are calculated with the function below.

```
tStart = tic;

    % Create training sets, 10-fold
     c = cvpartition(ratingMatrix(:,1),       ,10);;

     % Set which fold to work on: 1-10
     fold = 1;

      % Get logical vector for the selected training set fold
     TRID = training(c,fold);

     % Get logical vector for the selected test set fold
     TEID = test(c,fold);

     % Find indicies that makes the selected training set fold
     indTRID = find(TRID(:,1) == 1);

     % Extract training set data
     train = ratingMatrix(indTRID,:,:);

     % Find indicies that makes the selected test set fold
      indTEID = find(TEID(:,1) == 1);

     % Extract test set data
     test = ratingMatrix(indTEID,:,:);

     %Prepare for MAE
      numberOfRatings = size(test,1);
      sum = 0;
      mseSum = 0;

     % Calculate predictions for all the missing items in the training set using
         Weighted Slope One
     predictionMatrix = zeros(numberOfRatings,3);
     uberMatrix = zeros(numberOfRatings,6);
     for i=1:numberOfRatings
         %temp, only for user 1, afterwards use loop in loop for all users
         %if (fictTest1(i,1) == 1)
            predictionMatrix(i, 1) = test(i,1);
            predictionMatrix(i, 2) = test(i,2);
            predictionMatrix(i, 3) = predictItemForUser(ratingMatrix,
                deviationMatrix, test(i,1), test(i,2));

            % All in one matrix:
% UserID - ItemID - True Rating - Predicted Rating - Error Rate - Variance
            uberMatrix(i, 1) = predictionMatrix(i, 1); %userid
            uberMatrix(i, 2) = predictionMatrix(i, 2); %itemid
            uberMatrix(i, 3) = test(i, 3); %true rating
            uberMatrix(i, 4) = predictionMatrix(i, 3); %predicted rating
            uberMatrix(i, 5) = predictionMatrix(i, 3) - test(i, 3); %error
            uberMatrix(i, 6) = uberMatrix(i, 5)^2; %variance

            % Calculate the sum that will be used in MAE
            sum = sum + abs(predictionMatrix(i, 3) - test(i, 3));

            % Calculate the sum that will be used in MSE
            mseSum = mseSum + uberMatrix(i, 6);
        % end
     end
```

```
58
59      % Calculate MAE
60
61      MAE = (1/numberOfRatings) * sum
62
63      % Calculate MSE
64      MSE = mseSum / size(uberMatrix,1)
65
66      % Calculate RMSE
67      RMSE = sqrt(MSE)
68
69      % Normalized RMSE
70      NormalizedRMSE = RMSE / (max(uberMatrix(:,5)) - min(uberMatrix(:,5)))
71
72     tElapsedPopularityDifferential = uint64(toc(tStart));
73     disp([                                  ,  num2str(
           tElapsedPopularityDifferential)]);
```

Listing A.19: Calculate the predictions using the Weighted Slope One algorithm.

```
1  function prediction = predictItemForUser(ratingMatrix, deviationMatrix, userID,
       itemID)
2
3  denom = 0; %denominator
4  numer = 0; %enumerator
5  k = itemID;
6
7  %Find all the ratings for the user except for the current item
8  indUserRatings = find(ratingMatrix(:,1) == userID & ratingMatrix(:,2) ~= itemID);
9  userRatings = ratingMatrix(indUserRatings,:,:);
10
11 for i = 1:size(userRatings,1)
12    %For all the items the user has rated
13    j = userRatings(i,2);
14    ratingValue = userRatings(i,3);
15
16    %Get the number of times k and j have both been rated by the same user
17    indUserItems = find(deviationMatrix(:,1) == k & deviationMatrix(:,2) == j);
18    userItems = deviationMatrix(indUserItems,:,:,:);
19
20    %Skip the calculation if it isn't found
21    if (size(userItems) > 0)
22        for n = 1:size(userItems,1)
23          count = userItems(n,3);
24          sum = userItems(n,4);
25
26          %Calculate the average
27          average = sum / count;
28
29          %Increment denominator by count
30          denom = denom + count;
31
32          %Increment numerator
33          numer = numer + (count * (average + ratingValue));
34        end
35    end
36 end
37
38 if (denom == 0)
39     prediction = 0;
40 else
```

```
41      prediction = numer / denom;
42  end
43
44  end
```

A.4.4 Database diagram

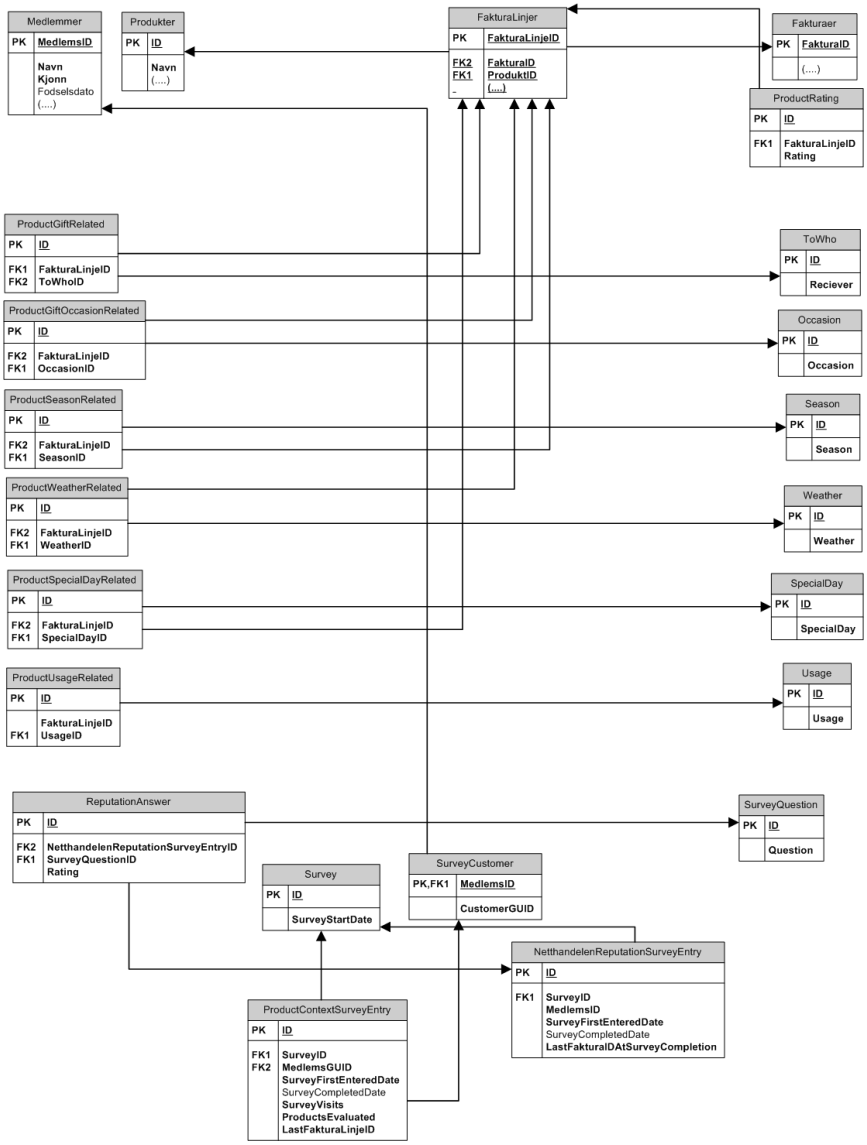


Figure 56: The database diagram.

A.5 Survey application evaluation

The survey can be tested in a sandbox environment for evaluation. The full source code can also be downloaded.
The URL is: `http://www.webwave.no/nhsurvey`.