Implikat -

A System for Categorizing Products using Implicit Feedback on a Website

Implikat -

Ett system för kategorisering av produkter med hjälp av implicit feedback på en webbsida

OLLE CARLQUIST

SANTOS BOSTRÖM LEIJON

Degree project in
Computer Science,
First level,15 hp
Supervisor from KTH: Reine Bergström
Examiner: Ibrahim Orhan
TRITA-STH 2014:24

KTH
The School for Technology and Health
136 40 Handen, Sverige

Abstract

Implicit feedback is a form a relevance feedback that is inferred from how users interact with an information retrieval system such as an online search engine. This degree project report describes a method of using implicit feedback to establish relevance judgments and rank products based on their relevance to a specified attribute. The report contains an overview of the benefits and limitations of implicit feedback, as well as a description on how those limitations can be mitigated.

A prototype that interpreted user actions as relevance votes and calculated a fair relevance score based on these votes with the help of an algorithm was developed. This system was then tested on a website with real users during a limited period of time. The results from the test period were evaluated and the system was concluded to be far from perfect, but that improvements could be made by making adjustments to the algorithm. The system performed better when looking at the algorithm's precision rather than its sensitivity.

Keywords
Implicit feedback, relevance judgments, ranking systems

Sammanfattning

Implicit feedback är en sorts relevansfeedback som sammanställs utifrån användares interaktion med ett informationsökningsssystem. Denna examensarbetesrapport beskriver ett sätt att använda implicit feedback för att skapa en bedömning av en produkts relevans till ett angivet attribut. Rapporten innehåller också en överblick av fördelarna och nackdelarna med implicit feedback, samt en beskrivning av hur dessa nackdelar kan hanteras.

En prototyp som översatte användarbeteende till olika relevansröster och beräknade ett relevansvärde baserat på dessa relevansröster med hjälp av en algoritm, utvecklades. Denna prototyp testades sedan på en hemsida med verkliga användare under en begränsad tid. Resultatet från denna testperiod analyserades och gav slutsatsen att prototypen inte var perfekt, men att resultaten kunde förbättras med hjälp av finjusteringar av algoritmen. Prototypens precision, med avseende på vilka produkter algoritmen valde ut som relevanta, var dock bättre än dess sensitivitet.

Nyckelord

Implicit feedback, relevansbedömning, rankningssystem

Innehållsförteckning

1 Introduction

Implicit feedback and its use in information retrieval (IR) systems, such as online search engines, have been studied since the early 1990s. By measuring how users interact with an IR system that generates rankings of items such as web pages or articles, signals of how relevant users found those items can be established. Most studies of implicit feedback has been made in the context of online search engines, but in this project implicit feedback will be used to rank products that are relevant to a specified category or attribute on a commercial website.

1.1 Problem statement

The idea of an automated system that could rank products on a website based on how relevant users perceived them came from the company Wall Cloud Productions. This system should remove the need of manually categorizing products and therefore reduce the workload on the company's employees. Since the task of placing products in different categories is a repetitive and tedious one, it would be desirable to make it unnecessary. By measuring how users interacted with a website and interpreting these interactions as relevance feedback, the system should be able to calculate how relevant a set of products are to a set of attributes.

To be able to replace the manual categorization done by the administrators on the website, the system needs to create relevance rankings in a reliable and precise way. This means that user interactions need to be measured in an accurate way, that these interactions are interpreted as useful feedback and that the system is not severely affected by irrational or malicious users. The system should not only measure user interaction, but also adjust the content on the website according to the rankings it generates.

1.2 Goals

The primary goal of this project was to develop a system that could use implicit feedback from a website to rank products according to their relevance to a specified attribute. This task was divided into the following goals:

1. Research the subject of implicit feedback in an online environment.
    a. Research and document a set of parameters that can be used to measure user behavior on the website used in this project.

    b. Research and document how those parameters can be measured from a technical standpoint.

    c. Research and document a number of algorithms that can be used to rank products based on implicit feedback.

2. Construct and test a number of prototypes.
    a. Conduct a number of simulations that documents and verifies the function of the algorithm that was chosen in the research stage of the project.

    b. Construct a number of prototypes that measure user behavior on the website used in this project and use the chosen algorithm to rank products.

    c. Implement the prototype on the website and test the prototype with real users during a limited period of time.

    d. Analyze and discuss the results from the prototype test.

3. Construct a finished end product.
    a. Construct a reusable module that can be implemented on multiple websites to measure user behavior to be used as implicit feedback.

    b. Construct a backend system that stores measurements of user behavior and create product rankings using the chosen algorithm.

    c. Construct a control panel where from the system can be managed.

1.3 Delimitations

To be able to complete the project within the allotted time and the given resources, a number of delimitations were given. Since the main goal of the project is to develop a system that uses implicit feedback to rank products on a website, and not to create the best possible rankings, basic functionality is more important than optimization.

The system only has to be implemented, tested and evaluated on a single website.

The system does not have to measure all possible parameters of users' behavior on the website. Only a set of suitable user actions that can be interpreted as either positive or negative implicit feedback needs to be measured.

The weights that are given to different parameters do not have to be optimized to give the best possible rankings, but rather to prove the basic functionality of the system.

1.4 Statement of authorship

The writing if this thesis, including the results from simulations and the tested prototype, have been undertaken as a mutual collaboration by Santos Boström Leijon and Olle Carlquist.

2 Theory

This chapter presents some basic theory behind ranking systems and implicit feedback. It describes three types of systems that have some similarities to the subject of this thesis. The benefits and limitations of implicit feedback are also described, as well as how implicit feedback can be classified and measured. A set of metrics that can be used to evaluate a ranking system is also presented.

2.1 Related systems

This section describes three systems that use data gathered from users' interaction with a website to rank various items. Recommender systems rank product recommendations based on ratings from users, social media sites like Reddit rank posts based on votes and search engines utilizing implicit feedback rank search results based on user behavior. Although two of these systems use explicit feedback in the form of ratings and votes, the mathematics behind them can still be applied to a ranking system using implicit feedback if an equivalent to a rating or vote is created.

2.1.1 Recommender systems

Online businesses such as Amazon and Netflix use recommender systems to give customers recommendations of products or items that might interest them [1]. These systems can use ratings, product descriptions, user demographics and other types of information to calculate what recommendations a specific user should be given. Recommender systems which use *Collaborative Filtering* base their recommendations on historical data gathered from users. This method can be used to find patterns in the ratings made by all users in the system and thereby predict which users will like which products. Recommender system can also use *Content-based recommending* which instead makes suggestions based on how similar the content of two or more items are, for example if a movie contains the same actors as another movie the user previously has rated highly. There are also systems which combine both of these methods to create a hybrid solution.

A recommender system which bases its recommendations on user ratings can use a matrix similar the one in *Figure 1* to represent the information gathered from the users. Each cell in this *user ratings matrix* contains the rating a user has given a specific item. Since it's unlikely that all users have rated all items there will be a large number of empty cells in the matrix.

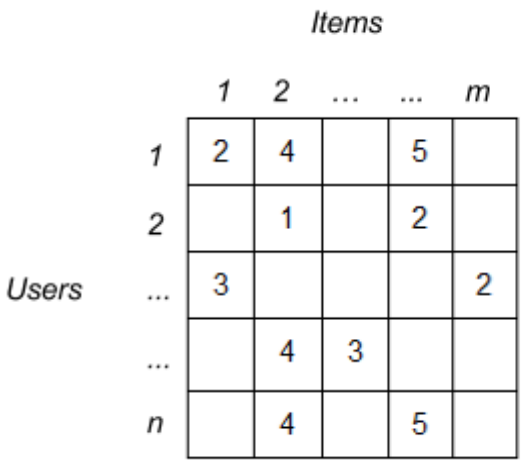

Figure 1: A user ratings matrix containing the ratings (1-5) that users have given items. The job of the recommender system is to predict what ratings the empty cells will contain.

The expected rating in one of these empty cells can be predicted using a simple weighted average and the *Pearson correlation coefficient,* or a similar correlation metric. The Pearson correlation is calculated using the equation below:

$$w_{i,j} = \frac{\sum_{u \in U} \left(r_{u,i} - \underline{r_i}\right)\left(r_{u,j} - \underline{r_j}\right)}{\sqrt{\sum_{u \in U} \left(r_{u,i} - \underline{r_i}\right)^2} \sqrt{\sum_{u \in U} \left(r_{u,j} - \underline{r_j}\right)^2}} \quad (1)$$

Where $w_{i,j}$ is the correlation between the items i and item j. U is a set of all users who have rated both i and j, with their ratings represented by $r_{u,i}$ and $r_{u,j}$. The overall average rating of items i and j across all users are $\underline{r_i}$ and $\underline{r_j}$. The predicted rating $p_{a,i}$ of user a on item i is calculated using a weighted average from a set of items K that have been rated by user a and have the highest correlation with item i.

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|} \quad (2)$$

Recommender systems are used to find items that are relevant to a specific user, often using explicit feedback in the form of ratings from users, although systems using implicit feedback also have been proposed [2]. If an equivalent to a rating is created from a collection of implicit feedback measurements, then the equations described in this section could be used in this project as well. Instead of predicting what rating a user will give to an item, the equations could be used to predict what level of relevance a product will have to an attribute.

2.1.2 Ranking algorithms using explicit feedback in social media

Websites like Facebook and Reddit use a system of likes, which is the case for Facebook, or upvotes and downvotes, in the case of Reddit, to rank content that is available for viewing by the user [3]. Unlike Facebook, which keeps it's algorithm for ranking content a secret, Reddit has their algorithm open to the public.

2.1.2.1 Reddit's story ranking

To rank the stories which users upload to Reddit, Reddit uses the ranking algorithm called "hot ranking", which is further explained below.

$$t_s = A - B \quad (3)$$

Where $t_s$is the time between the time of posting A and an arbitrary timestamp (2005-12-08 07:46:43)(B).

$$x = U - D \quad (4)$$

Where U being the number of “upvotes” and D being the number of “downvotes”, x becomes the difference between the two of them.

$$y = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x = 0 \\ -1 \text{ if } x < 0 \end{cases} \quad (5)$$

Where yis given a value of 1 if the story has a *negative* x, a value of -1 with a *positive* x and a value of 0 in the case of xbeing 0 as well.

$$z = \begin{cases} 1 \text{ if } |x| < 1 \\ |x| \text{ if } |x| \geq 1 \end{cases} \quad (6)$$

Here z has the absolute value of x, unless the case being that the value of x is 0, whereas z would get the value of 1. This check is done to scale the number of votes logarithmically in the function shown below.

$$f(t_s, y, z) = \log_{10} z + \frac{y * t_s}{45000} \quad (7)$$

The function $f(t_s, y, z)$, shown in (7) gives the value that is used when ranking the different stories that user may submit. The logarithmic part $\log_{10} z$ (7) evens out the variable of z when compared to different stories. It makes sure that a story with 100 upvotes and 0 downvotes is worth double in comparison to a story with 10 upvotes and 0 downvotes, as long as they were posted at the same time. The divisional part is used to make sure that older stories get a lower score than a newer story, in the event that they have the same amount of upvotes and downvotes.

2.1.2.2 Reddit's comment ranking

To rank comments on stories on the other hand, Reddit uses a different algorithm, the ranks achieved by this is called "best" ranking. To be more specific, Reddit uses the *lower bound of Wilson score confidence interval for a Bernoulli parameter* [4] to achieve this "best" ranking. The reason this is called "best" ranking is that it takes the number of votes into account when calculating a score, making the result fairer. This means that a comment with 1000 positive votes and 100 negative votes will rank higher than a comment with only 100 positive votes and 10 negative votes, despite the mean average of the votes being the same for both comments.

The Wilson score interval looks as follows, where p is a population of positive and negative votes, n is the number of votes and a is the confidence represented by a percentage, where the standard confidence is 95%:

$$\frac{\hat{p} + \frac{1}{2n}z^2_{1-a/2} - z_{1-a/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2_{1-a/2}}{4n^2}}}{1+\frac{1}{n}z^2_{1-a/2}} \qquad (8)$$

2.1.2.3 Relevance to this project

Ranking algorithms using explicit feedback can be useful when interpreting implicit feedback as well. As long as an equivalent to an explicit vote is created from measurements of implicit feedback the algorithms discussed in this chapter can be used as they are. The Wilson score interval algorithm is especially interesting since it takes the confidence of the result into account when calculating a score. Since time is not a relevant variable when categorizing products in the way it is when ranking news stories, Reddit's story ranking method is not as interesting for this project.

2.1.3 Search engines incorporating implicit feedback

The use of implicit feedback extracted from user behavior, such as counting the clicks on search results [5] and the rephrasing of search queries [6], in search engines has been studied several times during the last decade.

Agichtein and others [7] showed that incorporating implicit feedback into existing search engine algorithms that uses content- and link-based information to rank web pages can improve the accuracy of the results by as much as 31 % relative to the original performance. The authors measured a large number of different normal actions which users took when using a search engine, such as the number of results users clicked on for a each query and the amount of time users dwelled on a search results page. These measurements were then incorporated into a selection of existing ranking algorithms established in the industry. They were either incorporated directly into the algorithms or used to rerank the results which the original algorithms had produced. The result pages for each search query were then manually judged using human judges to compare the relevance of the results. The authors could conclude that using implicit feedback significantly improved the relevance of the results in all algorithms tested.

Dou and others [8] successfully used clickthrough data as an alternative to human judgments of relevance to train so called learning-to-rank algorithms which are used in search engines. Their experiments showed that clickthrough data from a large number of real-world users can achieve better search results than human judgments from a small number of judges. Clickthrough data was concluded to be especially reliable when the search queries were ambiguous or covered a broad topic.

Joachims and others [5] have highlighted the fact that clickthrough data from a search engine can be informative but that it is hard to measure absolute relevance with clicks alone. What results users click on is heavily dependent on what order the results are presented in and the overall relevance of the results presented. Users are more inclined to click on results that are listed highly on a search results page and are influenced by both the relevance of the clicked results itself and the results not clicked on.

Even though clickthrough data is an ineffective measurement of *absolute* relevance the authors conclude that clicks can be used to measure *relative* relevance with reasonable accuracy.

The studies conducted on the use of implicit feedback in search engines are of great interest to the subject of this thesis. These studies aim to solve the same problem as this thesis does, i.e. how to use user behavior on a website to measure the relevance of content. This project aims to solve the problem of deciding how relevant a particular product is to a particular attribute or category, while a search engine has the similar ambition of deciding how relevant a particular document is to a particular query. More findings from research on implicit feedback in search engines will be discussed in section 2.3 in this chapter.

2.2 Implicit feedback

Implicit feedback is a term used to describe relevance feedback that is inferred from how users interact with an information retrieval system [9]. This feedback is generally used to measure how relevant users found the content that was presented to them. In an online environment where the system in question is a website implicit feedback can consist of information about which pages users visited, how long they spent on each page and what links they clicked on. Explicit feedback refers to relevance feedback that comes from ratings which users or human judges have made of the performance of the information retrieval system.

2.2.1 The benefits of implicit feedback

The main benefit of implicit feedback over explicit feedback is that it can be gathered without having to interrupt the users' normal use of the system. Implicit feedback can be collected in larger quantities at a much lower cost compared to explicit feedback [5]. The fact that implicit feedback doesn't require users to be incentivized to give ratings or answer questionnaires gives it a clear advantage. Since this type of feedback measures the users' actual use of the system it also avoids the problem of users giving arbitrary or dishonest ratings which don't reflect their true opinions. Another advantage of implicit feedback is that it makes the system adapt to its users instead adapting to opinions of a smaller group of people who have given explicit feedback.

2.2.2 The limitations of implicit feedback

Even though implicit feedback has several interesting advantages it also has some significant limitations. Implicit feedback can be noisy and hard to draw useful conclusions from [10]. Individual users might behave irrationally or have malicious intents. Some users might be robots, and not actual real users, but still affect the information gathered. Useful conclusions can therefore only be drawn from the aggregated behavior of a large number of users, and not from the behavior of an individual user.

2.2.3 Classification of implicit feedback

In 2001, Oard and Kim [11] presented a framework for classifying the various kinds of user behavior that can be observed in an information

retrieval system. Even though their work is over ten years old by now, it was made with a general outlook on user behavior which still makes it useful today.

The classification scheme Oard and Kim proposed is illustrated in *Table 1*. The y-axis (*Examine*, *Retain*, *Reference* and *Annotate*) are different types of *Behavior Categories*. The x-axis (*Segment*, *Object* and *Class*) represent the *Minimum Scope* of the item that is acted upon. The scopes consist of *objects* which are documents of some kind, *classes* which are collections of objects and segments which are smaller parts of objects. The table contains examples of behaviors from Oard's and Kim's research, which are mostly relevant to information retrieval systems used for scientific research papers.

	Segment	**Object**	**Class**
Examine	View Listen	Select	
Retain	Print	Bookmark Save Delete Purchase	Subscribe
Reference	Copy-and-paste Quote	Forward Reply Link Cite	
Annotate	Markup	Rate Publish	Organize

Table 1: Classification of an example of user behavior that can be observed and used for implicit feedback

2.2.4 Interpreting viewing time

The time a user spends viewing a document has been shown to be an indication of how interesting that user finds that document. Morita and Shinoda [12] studied how measurements of user behavior can be used to improve an information retrieval system. One of the behaviors they studied was how much time users spent reading articles after they had retrieved them from an online news service. They compared explicit ratings, collected from users, of 8 000 articles with the time users spent reading those articles. The authors concluded that there was a strong tendency to spend a long time reading articles that users had rated as interesting and a significant tendency to not spend a long time reading articles rated as uninteresting.

Morita and Shinoda also studied factors that could affect the reading time. The length of the articles and the readability of the articles were measured, but was concluded to have an insignificant effect on reading time in the system they studied. The authors also compared different reading time thresholds for when an article could be predicted as interesting. The most effective threshold was found to be 20 seconds, which resulted in 30 % of the interesting articles being identified with 70 % precision.

2.2.5 Interpreting clicks

Joachims and others [5] studied clickthrough data from an online search engine and established that clicks are a reasonably accurate form of relevance judgment, but that they are biased in at least two ways. Users are more likely to click on results that rank higher on the results page and more likely to click on results that are more relevant than the other results on page, regardless of how relevant the clicked results are to the query in absolute terms. According to their research clickthrough data can, however, be used as a measurement of relative relevance, i.e. to establish if a document is more relevant to a specific query than another document. The authors also propose several strategies that can generate reliable implicit feedback by mitigating the position bias and quality bias that affect user behavior.

One of the strategies Joachims and the other authors present was called "*Click > Skip Above*". This strategy was used to generate *pairwise preferences* from clicks on search engine results pages. When a user skipped a higher ranked result and chose to click on a lower ranked result on the page, a pairwise preference was created that established that the lower ranked result was more relevant than the higher ranked result. This strategy was shown to be close in accuracy to explicit relevance judgments that had been made by human judges. A mathematical explanation of the *Click > Skip Above* strategy is shown below.

> ***Click > Skip Above* [5]:**
>
> *For a ranking* $(l_1, l_2, l_3, \ldots)$ *and a set* C *containing the ranks of the clicked-on links,*
>
> *extract a preference example* $rel(l_i) > rel(l_j)$ *for all pairs* $1 \leq j < i$, *with* $i \in C$ *and* $j \notin C$.

Despite the many similarities between the website used in this project and traditional document based search engines, there is one major difference that changes user behavior. Traditional search engines are text-based and present search results in a list. The presentation the website used in this project is more similar to that of an image search engine where thumbnail images are displayed in a grid. Instead of reading document titles and excerpts, users using an image search engine examine the results visually to make a judgment on their relevance.

The use of implicit feedback in image search engines has also been studied previously. Smith and Ashman [13] analyzed the available research on the topic and conducted their own experiments to conclude that clickthrough data from image search is more accurate in general than the same type of data from document search. Although the clickthrough data was reliable in general, when users had a low level of knowledge about the topic they were searching for and the results from the search engine were of poor relevance the data became unreliable. Since our implementation

is focused on categorizing products based on opinion and taste rather than knowledge, this is not likely to cause any issues for us.

2.2.6 Measuring viewing time

The task of measuring viewing time can either be given to the server or the client. In this section, the benefits and drawbacks of each of these approaches are discussed.

2.2.6.1 Measuring viewing time on the server

One way of measuring the time a user spends viewing a page is to examine the server logs. By comparing the time of a user's first page request with the time of the user's second page request one can estimate how long the user stayed on the first page. This method has the benefit of not burdening the user's web browser with the task of measuring viewing time. The measurements can be gathered after the user's visit has ended without affecting the user experience or allowing users to tamper with the data.

This server log method also has some major drawbacks. One of those is that it only measure the time between new requests to the server. When the content on the page is updated without a new request occurring, i.e. when JavaScript is used to hide or display content that already has been loaded, this method is not applicable. Another drawback is that it requires the user to visit several pages sequentially. If the user only visits one page, reads the content on it and then leaves without a new request to the server occurring the server has no way of knowing how long the user stayed on that page.

A third drawback of the server log method is that the log entries from all users are collected in the same log files. Each unique user will therefore have to be identified using IP addresses and other metadata, which can be unreliable [14]. A fourth drawback of this method is that it requires the measurement system to be developed around the web server software. The measurement system has to know where the server logs are located in the file systems, has to have the file system permissions to read those files

and has to know how to parse them. If the measurement system has to be built for a specific web server software its flexibility is reduced greatly.

Another way of measuring viewing time on the server is to use some kind of server-side programming language that keeps track of users and timestamps. The website used in this project was built using *Ruby on Rails* [15] which is a open-source web application framework for the programming language *Ruby*. This framework can easily be used to make estimations of viewing times in the same way as with server logs. The problem with identifying unique users in a reliable way can also be solved using cookies, which are supported by Ruby on Rails [16]. This method still requires users to visit several pages sequentially or trigger several requests to the server in another way. It will also affect the workload of the web server since the measurements have to be computed at the same time as web pages are generated.

2.2.6.2 Measuring viewing time on the client

Viewing time can also be measured using the user's web browser. By using JavaScript and its timer functionality [17] in the browser viewing time can be measured regardless of how many pages the user visits or how many requests to the server occurs. To store this information persistently the measurements would still have to be sent to the server, but there is no need for the system to be built for a specific web server and its server logs. There is also no ambiguity about the user's unique identify since the viewing time is measured in each user's individual web browser.

A drawback of the client method is that it burdens the web browser with extra tasks to perform besides loading and rendering the web page. This may hurt the user experience if these tasks require a significant amount of computer resources to be performed. Another drawback with making the web browser responsible for measuring viewing time is that users with malicious intents can tamper with the measurements. It is therefore important to make calculations that directly affect the ranking of products on the server and not on the client.

One limitation to keep in mind is that these methods do not generate a direct measurement of how long the user is actually looking at a web page, but rather a measurement of how long the content is displayed. It could be possible that the user has a web page open, but isn't looking at any of the content on it [14].

2.2.7 Measuring clicks

Clicks that result in a request to the server can, just like with viewing time, be measured by examining server logs. This has the previously described benefits of not affecting the user experience and not being vulnerable to tampering by users. But it also has the unattractive drawback of requiring the system to be built for a specific web server software and its logs. It also requires users to be identified with metadata, which as previously stated can be unreliable.

With a server-side programming language, clicks can be counted without regards for the web server's logs, as long as the web server supports the programming language in question. A drawback of this method is that the web server's workload will increase since the clicks have to be counted and stored in a database in real-time as the web page is generated. Since it is the server that does the computations it still has the benefit of affecting the performance of the user's web browser.

Both these methods can only measure clicks that result in a request to the server, i.e. clicks on regular hyperlinks that redirect users to a new page or clicks on elements that loads a document in the background using JavaScript and XMLHttpRequest [18]. If a click doesn't generate a request to the server it has to be captured using the user's web browser and JavaScript. When the click has been captured the measurement can then be delivered to the server using XMLHttpRequest so that it can be stored persistently. This method has the drawback of possibly affecting the performance of the user's web browser since it has to perform extra tasks.

2.2.8 Mitigating unreliable implicit feedback

Although implicit feedback can be a useful when treated correctly, it can be unreliable if certain behavioral biases and issues are not taken into

account. As Joachims and others [5] found in their research, users' behavior when interacting with an information retrieval system is biased in two major ways. There is a position bias that affect which results users chose to examine and a quality bias that affect how relevance judgments can be inferred from user behavior. There is also a concern about robots and users with malicious intents that can generate unwanted implicit feedback.

2.2.8.1 Mitigating position bias

Position bias, also called *trust bias*, is an observed behavioral pattern that makes users focus more of their attention on results that are ranked higher on a search results page than those ranked lower on the page [5]. Users are more likely to click on the top results even if the lower ranked results are more relevant since they have an inherent trust in the information retrieval system. Even though the existing research was done on search results displayed in a one-dimensional list, it seems reasonable that the same type of position bias exists when products are displayed in a two-dimensional grid as on the website used in this project.

One way of mitigating position bias is to quantify it and include it with the measurements of user actions. To quantify how big the position bias is on different positions in the grid systems automatic experiments could be conducted on the website with the website's normal visitors. A number that represents the bias could be established by calculating the click-through rate, i.e. the number of clicks divided by the number of page views, for each position in the grid. By displaying random products during these experiments the products' varying levels of relevance are cancelled out and only their position in the grid is affecting which products the user clicks on. A drawback with this approach is that these experiments are likely to hurt the user experience and irritate the users since random products are displayed instead of the most relevant ones.

Another way of mitigating position bias is to always randomize the order of the displayed products. The top most relevant products would still be displayed to the user, but where in the grid they are positioned would be random and not ordered from highest to lowest relevance. This is also

likely to hurt the user relevance since the most relevant product could happen to be positioned in the lowest row of the grid and the sixteenth most relevant product could be displayed at the absolute top. From a user experience point of view, this is still more attractive than having to browse through completely random products, all of which could possibly be irrelevant to the user, for the purpose of quantifying position bias.

2.2.8.2 Mitigating quality bias

As previous studies [5] have shown, it is difficult to interpret clicks as a form of *absolute relevance* judgments. This means that a product is not necessarily relevant to a specific attribute in absolute terms just because it has received many clicks from users. Clicks can, however, be interpreted as *relative relevance* judgments with reasonable reliability, i.e. that one product is more relevant than another product if both of these have been presented to the user. There is an inherent *quality bias* to users' behavior which means that users judge an item relative to the quality of the items around it rather than judging the item on its own merits alone.

The "*Click > Skip Above*" strategy, as proposed by Joachims and others [5], that was explained previously in this chapter solves the issues with position and quality bias. The problem with this strategy is that it was designed for search engine results pages where results are displayed in a one-dimensional list. In our implementation products are displayed in a two-dimensional grid where they are not only displayed from top to bottom, but also from left to right. It is therefore difficult to implement this strategy in our solution.

Quality bias can also be mitigated by including a limited number of randomly selected products in all sets of products displayed to users. This means that implicit feedback can be interpreted as absolute relevance judgments, but that these judgments are continually challenged by giving new products exposure that lets them received implicit feedback. Eventually, all products will have been displayed side-by-side in the same grid as all other products, and an indirect relative relevance judgment has therefore been made in the aggregate.

2.2.8.3 *Limiting user influence*

There is no guarantee that all users will use the website in the way it was intended. Some users might have malicious intents or behave irrationally which can lower the quality of the collected implicit feedback. It is therefore important to limit individual users influence on the relevance rankings. One way of limiting this influence is to only collect a certain number of user actions from each user. This could be implemented so that, for example, only the first ten clicks or views would be used as implicit feedback.

Another way of limiting user influence is to only use a certain number of user actions for a certain time period. The system could be designed to, for example, only measure ten clicks or views per minute from each user. Both of these methods would require the system to identify each user so that the number of actions could be tracked. This identification could be done with a cookie [19] or a "browser fingerprint" [20] identified by metadata from the user's web browser.

It is not only irrational and malicious users that can damage the relevance rankings on the website. There are also a large number of robots, or bots, which can generate false implicit feedback by interacting with the website. According to a study conducted by the cloud application company *Incapsula* [21] up to 61.5 percent of all website traffic online is created by bots. These bots have a variety of purposes. Some are used by search engines like Google to index the content on websites while others are used by hackers to identify security weaknesses. Bots that are open with their identify, like *Googlebot*, can be identified by reading the *User-Agent* field in the *Hypertext Transfer Protocol* (HTTP) header [22] while other bots that pretend to be real users need to be limited using the same techniques as used to limit normal users.

2.3 **Evaluating result sets**

To evaluate how accurate a ranking system is the result sets it generates need to be examined in a systematic way. There are four common metrics that are used to evaluate these kinds of result sets: *precision*, *recall*, *F-measure* and *Mean Average Precision* (MAP) [23]. These metrics require

all examined items in the system to be judges as *relevant* or *not relevant* for each evaluated query. For this project, a query is equivalent to an attribute.

Precision (P) is the fraction of retrieved items that have been judged as relevant.

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} \tag{9}$$

Recall (R) is the fraction of relevant items that have been retrieved out of all relevant items that exists in the system.

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} \tag{10}$$

The F-measure is used to combine these two numbers and find the *weighted harmonic mean* of precision and recall. This is used to generate a single value that considers both how many relevant items the result set contained and how many relevant items that were not included in the result set. The F-measure can be tuned to give a higher weight to one of the two metrics, but the default method is to use the *balance F-measure*, also called the *F1-measure*.

$$F_1 = \frac{2PR}{P+R} \tag{11}$$

3 Method

This chapter describes how the prototype of our measurement system was created. The user behavior on the website is described together with a discussion on how various behaviors can be interpreted as implicit feedback. The algorithm used to create a relevance score from the measured user behaviors is also described, as well as the technologies used to develop the system.

The word *attribute* is used multiple times in this section. An attribute can be a category, a tag or any other kind of characteristic that is shared among several products. The purpose of this measurement system is to measure how relevant products are to different attributes, i.e. to discover what products belong to what categories.

3.1 Interpreting user behavior

The website used in this project was built using a proprietary system called *ShowSpace*. This system is used to create websites that showcase a collection of products within a niche topic. A website could showcase products like masquerade costumes or coffee mugs. The commercial purpose of these websites is to market these niche products and generate sales for the online stores where they are sold.

The use case diagram in Figure 2 illustrates how users can interact with the website. Interactions that are deemed to be useful when measuring a product's relevance to a specific attribute are highlighted in yellow. These interactions are performed when a user has arrived to a page with the expectation of seeing products that are relevant to a specified attribute.

When users use the search function or filter products based on price they do not necessarily expect to see products relevant to any specific attribute, and that type of relevance can therefore not be measured based on their actions.

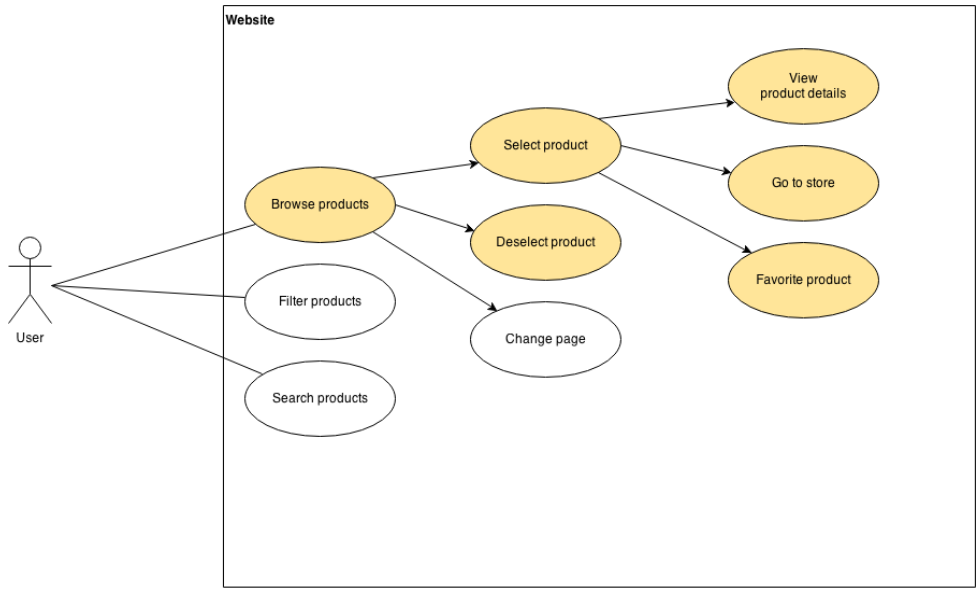

Figure 2: A use case diagram of how a user can interact with the website.

As illustrated in *Figure 3,* the products are displayed in a grid system on the website. When a user hovers over a product image the product's name and price are displayed together with an abbreviated description. Users can also go the next page of results as part of the normal browsing behavior.

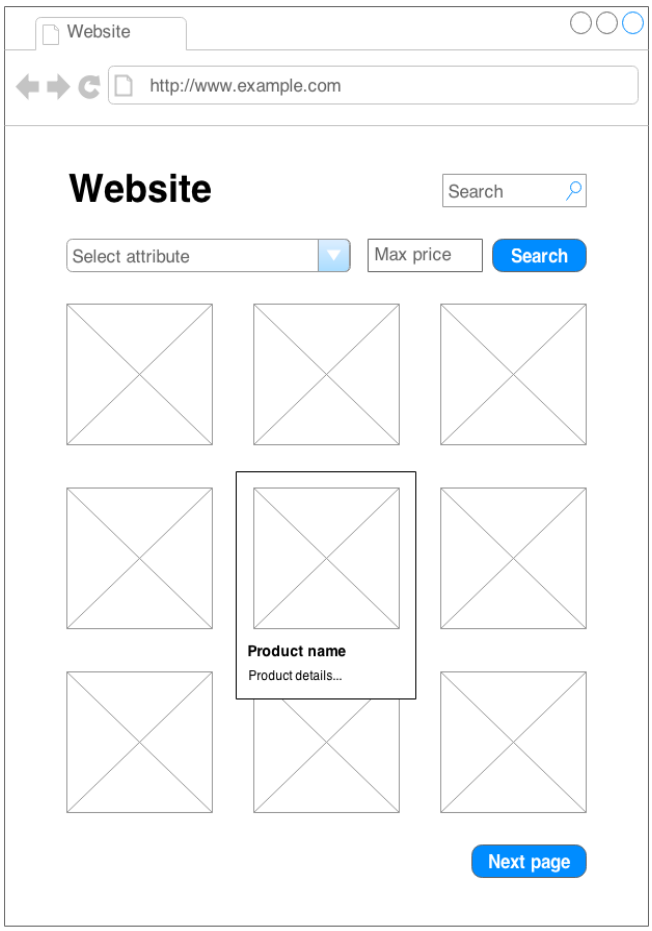

Figure 3: A mockup of the browsing features implemented on the website.

When a product has been selected a container with a larger image, a full description and other details are displayed. This is illustrated in *Figure 4*. The container also includes a link to the store where the product is sold and an option to add the product to the user's personal list of favorites.

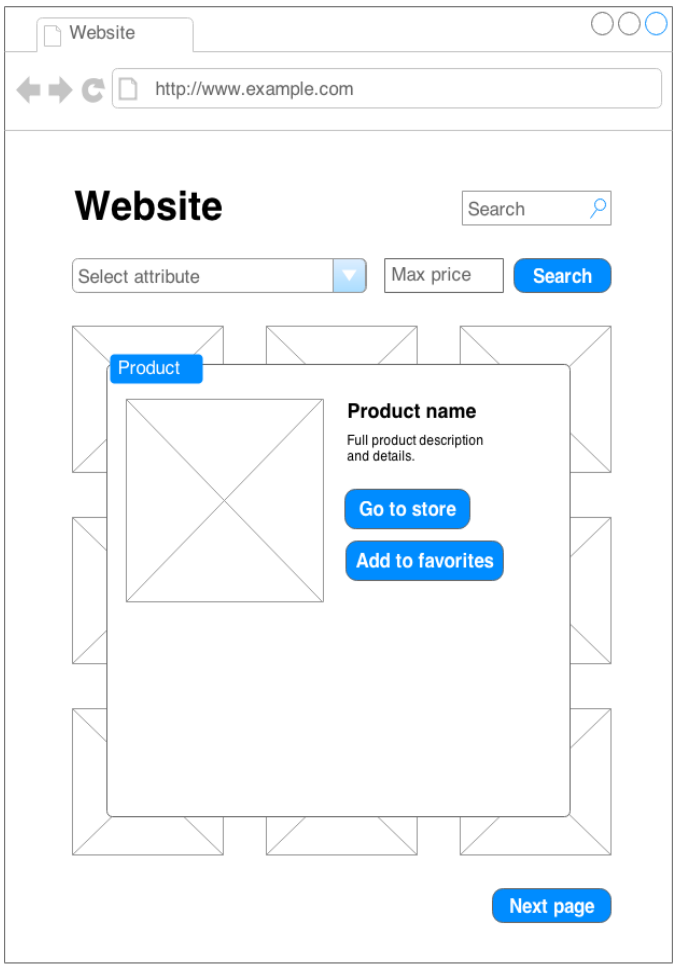

Figure 4: A mockup of the product selection feature on the website.

3.2 Classifying user behavior

In *Table 2* the user behaviors described in the previous section are classified using the framework from Oard and Kim [11], as described in chapter 2. These kinds of behaviors are a part of the normal use cases for the website in question. In this implementation, objects represent products, classes represent categories of products and segments represent product images and product descriptions. Why these actions were chosen was described in section 3.1 in this report.

	Segment	**Object**	**Class**
Examine	View	Select Deselect	
Retain		Go to store	
Reference			
Annotate		Favorite	

Table 2: Classification of user behavior that is relevant to our implementation.

3.2.1 Product viewing

As a part of the normal user behavior, users view product images and read product details displayed on the website. The decision to look at a certain image or description on a page is based on some kind of judgment from the user and is therefore an interesting form of implicit feedback. Unfortunately the eyes' focal point cannot be measured in a non-intrusive way without using cameras, so the act of viewing an item has to be triggered in another way.

The website used in this project contains product information and not the type of articles that has been studied previously by Morita and Shinoda [12]. It could be possible that users behave differently when viewing product images and reading product description than they do when reading news articles, but it stands to reason that the principle of users spending more time viewing content they find interesting and less time viewing content they find uninteresting is still applicable. The threshold of 20 seconds might not be optimal for our implementation since the short product descriptions can be read much faster than the longer news articles previously studied.

There are several different viewing behaviors that can be measured on the website. The time a user is viewing the container of product information that is displayed after a product has been selected on the website would be the equivalent of the reading times studied by Morita and Shinoda [12]. But it could also be possible to measure how long a user views a collection of products and use that as a signal that all those products were perceived as relevant.

3.2.2 Product selecting and deselecting

When users browse products on the website they can select a product by clicking on the product's thumbnail image which is displayed together with other products in a grid system. When a product has been selected a container containing a larger product image and all available details about that product is displayed. By selecting a product the user is showing some kind of interest in it. This expression of interest could be used as a form of positive implicit feedback. A user can also deselect a product by closing the container that was displayed when the user selected it. This could be interpreted as a negative form of implicit feedback since the user was not fully satisfied with the selected product and decided to close the container so other products would be shown again.

The behavior of selecting a product from a grid of related products on a web page is very similar to that of selecting a document from a list of search results on a search engine, which is what has been studied previously as described in chapter 2. The user makes a judgment of what prod-

uct seems most relevant, or what document contains the answer to the user's query, and selects it to found out more. Implicit feedback, such as clicks on search results, and its use in search engines has been studied several times in the past as explained previously in this chapter.

3.2.3 Lead generation

The commercial purpose of the website used in this project is to generate potential customers to the online stores that sell the products that are displayed on the website. If a user is interested in a particular product the user can click on a link with the text "Go to store" and be redirected to an online seller where the product can be bought. A commission based on the value of generated sale is then paid to the website's owner through a third party. A click on this link can be interpreted as a strong indicating that the product displayed was relevant to the user since it made the user consider a purchase of the product.

3.2.4 Product favoring

If a user finds an interesting product on the website the user can add that product to a list of personal favorites. This is a feature that lets users save the products they liked while continuing to browse other products on the website. It also allows users to leave the website and come back at a later date to look at their favorite products again. To add a product to the favorites list can be interpreted as an indication that the product was relevant to the user, since the user is showing an interest in the product and want to save it. This action is performed by clicking on a button and is therefore a form of click data.

3.3 **Chosen algorithm**

To be able to understand how relevant an item is according to measured user actions, an algorithm to evaluate the data is needed. A research for an algorithm to use was done.

3.3.1 Wilson Score

The algorithm chosen to evaluate data in the system was the lower bound of the Wilson score interval for Bernoulli parameters. The Wilson score is used to get a value between 0 and 1 that shows the relevance in a percent-

age. Using simple mathematical functions makes the algorithm easy to implement in most programming languages. The Wilson score interval uses a confidence interval to make sure that small sample sizes give lower relevance scores than higher sample sizes. [4]

The Wilson score interval calculates a score from two numbers, one being the number of positive votes and the other being the number of votes overall. A full positive vote increments the number of votes overall with 1 and since it has the weight of 1 also increments the number of positive votes with 1. A full negative vote also increments the number of votes overall with 1, but since it has the weight of 0 does not increment the number of positive votes. A half negative vote increments the number of votes overall with 0.5 but since it has a weight of 0 this time as well does not increment the number of positive votes. A half positive vote increments the number of votes overall with 0.5 and increments the number of positive votes with 0.5 as well. This is why different vote weights are incorporated in the calculation of the relevance score.

3.3.2 Usage

In this project, the Wilson score interval algorithm is used together with weighted data from different user input methods to get a relevance score between 0 and 1 for a specific item, usable to represent a percentage of relevance. This is then taken into consideration when selecting items from the database.

3.3.3 Alternatives

To evaluate user actions that are interpreted as either positive or negative votes, other algorithms could be used. One such alternative is a simple algorithm where the relevance score is calculated through subtracting the positive votes from the negative votes. This was not chosen because the result it generates isn't a percentage and because the result can be wrongfully interpreted. For example:

An item has 10 563 positive votes and 10 000 negative votes. The relevance score according to this algorithm would be 563, but the positive votes are 51 % of the total votes. Whereas another item with 427 positive

votes and 300 negative votes would have a relevance score of 127, much lower than the earlier item but having a higher percentage of positive votes at about 59%.

Another method that could be used to create rankings from implicit feedback measurements is to use a *Support Vector Machine* (SVM). A modified version of this model has been used previously to optimize search engine results using implicit feedback [26]. The SVM is a form of machine learning system. The SVM can recognize patterns in large sets of measurements and analyze the collected data to do find relationships between different variables. This is a far more advanced and mathematical intense approach that would be challenging to implement in a simple web application.

3.4 Simulations

To test if the chosen algorithm behaves in a predictable and useful way a series of simulations were conducted. The input data and configuration of the vote weights were estimated to simulate real-life scenarios, but were used to the test the basic functions of the algorithm rather than optimizing it for maximum precision. A positive weight is indicated by a positive number between 0 and 1, and a negative weight is indicated by a negative number from -1 to 0. The *relevance scores* that are mentioned in these test refers to a theoretical product's level of relevance to a theoretical attribute.

The simulation tool developed to conduct these tests was a simple JavaScript application operated from a regular *HyperText Markup Language* (HTML) web page. On this web page the number of different user actions that should be included in the test and the weight of those user actions was be specified from a web form. The votes representing those user actions were then put through a JavaScript implementation of the lower bound Wilson score interval with a with a standard 95 % probability. The JavaScript implementation of the algorithm was based on a function from Honza Pokorny [27].

The weights of user actions used in these simulations were as follows:

- Weight of select action vote: +0.5
- Weight of view action vote: +0.25
- Weight of go-to-store action vote: +1.0
- Weight of favorite action vote: +0.75
- Weight of deselect action vote: -0.5

3.4.1 Simulation of confidence ranking

This simulation was conducted to test the function of the Wilson score interval. In theory this algorithm takes the number of votes into account when calculating the relevance score. If the number of votes is higher than the score should be higher as well. This will ensure that established products that have proved their relevance by receiving many positive votes will not be outranked by new products with only a few votes. In this test one product was given 100 select actions and 10 deselect actions, while another product was given 10 000 select actions and 1 000 deselect actions. Since the first product has fewer votes its relevance score should be lower than the second product's, despite the fact that the proportion between these positive and negative votes are the same. As the diagram in *figure 5* shows, the first product had a relevance score that was 10.8% lower than the second product. This proved that the Wilson score interval algorithm was working as expected.

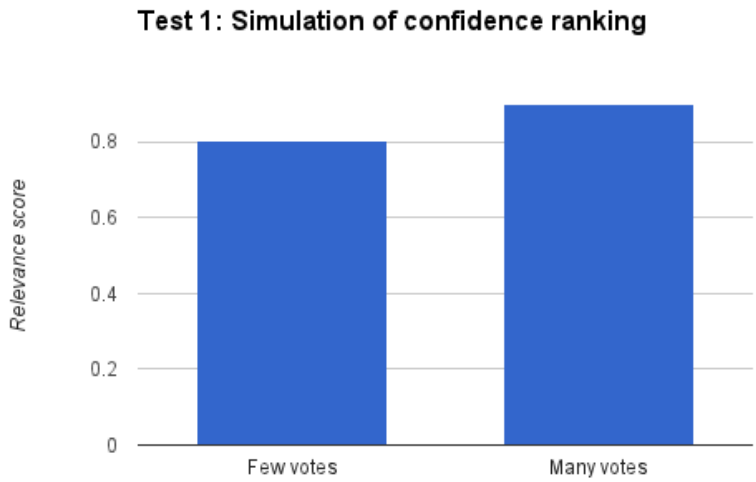

Figure 5: Diagram of how the relevance score will differ between a product with few votes and a product with many votes.

3.4.2 Simulation of use cases

This simulation tests how the relevance score of a relevant product will differ from that of a non-relevant product. The input data in this simulation is an estimation of the user actions that would be generated from 10 000 page visits. The relevant product is assumed to trigger more positive user feedback since it receives more positive user actions. The two products were given the following user actions:

Non relevant product:

- 90 select actions (0.9 % of 10 000 page visits)
- 10 view actions (0.1 % of 10 000 page visits)
- 10 go-to-store actions (0.1 % of 10 000 page visits)
- 10 favorite actions (0.1 % of 10 000 page visits)
- 90 deselect actions (0.9 % of 10 000 page visits)

Relevant product:

- 9000 select actions (90 % of 10 000 page visits)
- 7500 view actions (75 % of 10 000 page visits)
- 2500 go-to-store actions (25 % of 10 000 page visits)
- 2500 favorite actions (25 % of 10 000 page visits)
- 2500 deselect actions (25 % of 10 000 page visits)

In this simulation the non-relevant product was expected to have a significantly lower relevance value than the relevant product. As the diagram in *figure 6* shows, the non-relevant product had a 44.1% lower value than the relevant product. This proved that the algorithm was working as expected in this test as well.

Since the non-relevant product still received a significant amount of positive feedback, its relevance value is quite far from 0.0 which could be considered completely irrelevant. If more negative user actions could be measured and included in the calculation the value would probably be closer to 0.0.

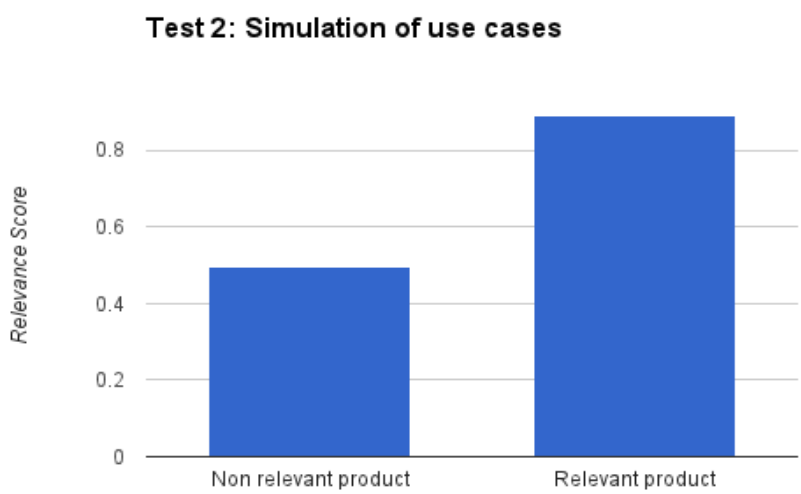

Figure 6: Diagram of what the relevance score of a non-relevant product and a relevant product will look like.

3.4.3 Simulation of a malicious individual user's effect on rankings

This simulation tests how an individual user who acts irrationally can affect an established relevance score. By taking the previously used input data from *test 2* and adding a limited number of user actions that either indicates that the relevant product is non-relevant, or that the non-relevant product is relevant, we can compare the relevance score before and after an irrational user has affected the system.

In this simulation 10 user actions (go-to-store), which increased the relevance score were added to the non-relevant product. 10 user actions (de-select) which decreased the relevance score were also added the relevant product. This in turn gave a 0.0427% decrease for the relevant products relevance and a 7.6% increase for the non-relevant products relevance. *Figure 7* shows that when a product has received thousands of relevance votes, a few new votes from an irrational user who generates implicit feedback that contradicts the consensus will not affect the ranking in any noticeable way. A non-relevant product that has not received as many votes is, however, more vulnerable to contradicting votes.

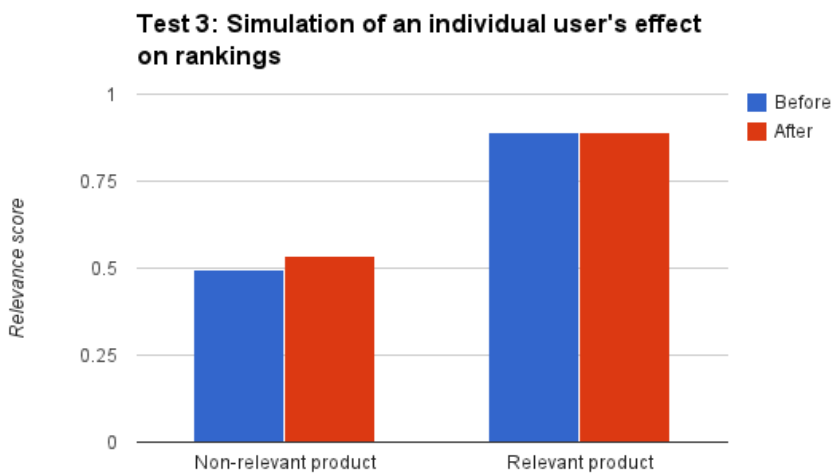

Figure 7: Diagram of a malicious individual user's effect on rankings, before and after the user's implicit feedback was included in the calculation.

3.5 Technologies

3.5.1 Ruby on Rails on the server side

Ruby on Rails (RoR) is a web development framework developed for the programming language Ruby [29]. The framework itself, which is open source, is also referred to as only *Rails*. Rails is built as a library on top of Ruby and runs on the Ruby web server *WEBrick*. The framework follows a number of design-patterns such as *model-view-controller* and *convention over configuration* that dictates how directory structures, classes, variable names and other elements should be designed in a Rails project. Rails also follows the *Representational state transfer* (RESTful) architecture as it applies to web services. These design and naming conventions have the purpose of making the development process faster, but gives the developer little room for customization and configuration.

3.5.1.1 Usage

Ruby on Rails is used on the server side in this project to store the measurements of user actions in a database and to generate the rankings of products that are presented to the users. Measurements are added when the server receives a *Hypertext Transfer Protocol* (HTTP) POST-requests to the appropriate *Uniform Resource Locator* (URL), as the RESTful architecture dictates. Rankings are generated in the same way when the server receives a HTTP GET-request to the appropriate URL.

Measurements, products and other data objects are managed using *ActiveRecord* which is Rails' way of mapping objects in the MVC framework to relational tables in the database system. Among other things, ActiveRecord manages the foreign keys of objects and allows changes to the database to be rollbacked at any time [31].

3.5.1.2 Alternatives

There are a wide variety of frameworks and programming languages that could be used as an alternative to Ruby on Rails in this project. Microsoft's .NET [32] and JavaServer Faces [33] are two software frameworks that have an extensive feature set for developing advanced web

applications. There are also scripting languages like PHP [34] and Python [35] that have the necessary functionality to be used in this project.

Since this project did not require any special features, other than to read and write data to a database and handle POST- and GET-requests from web browsers, and performance was not a key issue, the choice of framework on the server side was not an important consideration. Ruby on Rails was chosen because the servers were already set up and was ready to be used. No new software was needed to be purchased or installed.

3.5.2 PostgreSQL as the database server

PostgreSQL is a relational database [28] used in the system created to save user measurements, different measurement types and all the different products and attributes. Most of the database was already designed and filled with data for the website when this project started, although the tables for feedback types and feedback measurements needed to be created.

3.5.2.1 Usage

PostgreSQL is used in the system to persistently store the different types of feedback measured as user interactions.

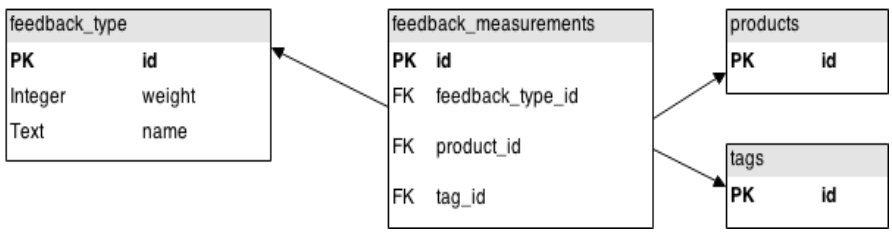

Figure 8: An Entity-Relationship model of the database tables used in the system.

Each type of measured user interaction is abstracted as a row in the table feedback_type, containing a name and a weight. The products table contains products with associated data. The tags table contains the different categories or attributes that products can be sorted in. The measured user interactions are abstracted in the database as a row in the table feed-

back_measurements, containing the different primary keys for feedback_type, products and tags.

3.5.2.2 Alternatives

Alternative database systems could have been used, for example Microsoft SQL Server [30]. The reason for not using another database system to store the implicit feedback measurements was that it made the relations between the measured feedback and the existing products easier to establish. Multiple databases hosted on multiple servers would make the queries more complicated and less efficient.

3.5.3 Heroku in the cloud

Heroku [39] is a cloud platform that hosts web application developed with a variety of programming languages and frameworks. Among the supported systems are Ruby on Rails and the database PostgreSQL which are used in this project. Heroku offers a complete platform as a service that can scale according to the website's needs. To provide this scalability, Heroku uses Amazon Web Services as the infrastructure behind the scenes.

3.5.3.1 Usage

Heroku has been used to host the Rails application and the PostgreSQL database in this project. The version control system Git, which is supported by Heroku, was used to upload and download code from the Heroku repository.

3.5.3.2 Alternatives

An alternative to Heroku would be to create a private server to host the website and database. This would not be viable in the long term since it would require continuous server maintenance, which the company does not have the resources or will to provide. Another alternative would be to use a new cloud or web hosting service. This was, however, not feasible due to budget constraints.

Both of these alternatives were not as time and resource efficient as using Heroku, since the original website was already deployed to that cloud service.

3.5.4 JavaScript on the client side

JavaScript is a lightweight scripting language developed to make web pages more dynamic and interactive [36]. The scripts written in JavaScript are executed in the web browser and are therefore separate from the server software. This also means that different browsers may support different functions within JavaScript. According to the *Mozilla Developer Network* [36], all "modern" browsers did, however, support the latest JavaScript standard called *ECMAScript 5.1* as of 2012.

3.5.4.1 jQuery

JQuery [37] is a JavaScript library that makes HTML document traversal, element manipulation and event handling easier to work with as a developer. Less code has to be written by the developer and many of the advanced features of JavaScript can be accessed through a simple API. JQuery also has an important advantage since it handles the cross browser incompatibility issues that arise when different web browsers support different APIs. With jQuery, the developer only has to use a single API supplied by jQuery and not multiple APIs for various different web browser. The drawback of using jQuery, and similar JavaScript libraries, is that they add an overhead to the web page which can make loading and rendering times slower.

3.5.4.2 Usage

JavaScript is used in this project to capture the clicks and viewing times that constitute the measured user actions on the web page. The scripts both detect when these actions are performed and send the measurements to the server so that they can be stored in the database. The *Document Object Model* (DOM) API is used to detect when a user has clicked on a specific element on the page and the *setTimeout* function is used to detect when an element has been viewed for a specified period of time. To send measurements to the server, the *XMLHttpRequest* API is used. This

functionality is used to generate asynchronous HTTP requests to the server in the background without interrupting the user experience.

To simplify the JavaScript code and make it compatible with all common web browsers used by users, the jQuery library was used. Since the website our solution was developed for already used jQuery this didn't burden the loading times more than they already were.

The JavaScript code was written in a dynamic way, using HTML classes detected via the DOM, so that more clicks and viewing times can be measured on the web page in the future. This will be useful if new functions and buttons are added to the page at a later stage.

3.5.4.3 Alternatives

One alternative to using JavaScript to measure clicks and viewing times would be to scan the logs from the web server. This method was discussed in chapter 2 and would have the drawback of not being able to measure user actions that do not result in a request to the sever. It also would require the system to be developed for a single web server and its logging system, which would make the system less dynamic. Another alternative would be to use a middleware like *Adobe Flash* [37] which is enabled in the web browser through a plugin. This solution, although theoretically possible, would be inefficient since it requires all users to have the plugin installed and activated. Adobe Flash is also considered an outdated technology that is currently being replaced by HTML5 [38].

3.6 **Prototype**

A prototype using the chosen algorithm was developed and tested with real users during a limited period of time. An implementation of both the ranking algorithm and the measurement system which collected measurements of user actions was used live on the website for seven days. During this time normal users visited the website and used it as they normally would. The prototype created rankings for three different attributes using a set of 50 different products.

3.6.1 Prototype algorithm

The ranking algorithm implemented in the prototype had the following parameter weights. These weights were only estimations of what could generate efficient rankings. After analyzing the results from the prototype more efficient weights were found.

- Weight of select action: +0.5
- Weight of deselect action: -0.5
- Weight of view action: +0.25
- Weight of go-to-store action: +1.0
- Weight of favorite action: +0.75

The ranking algorithm used in the prototype generated a result set of 25 products for each attribute. In principle, the result set should contain the 25 products that are most relevant to that attribute. Since the algorithm used in the prototype incorporates random products this is, however, not the case. Only 20 of the 25 products in the result set are chosen by their relevance. The reasons for adding random products and randomizing the order of the products are described in section 2.2.8 of this report.

This prototype algorithm could be described as follows, using pseudo code:

Prototype algorithm:

Let X be an integer with the maximum value of 20.

Let R be a result set of products.

Extract the X most relevant products that have a relevance score higher than 0.3 and append these to R.

Extract 25-X random products, regardless of relevance, and append these to R.

Randomize the order of products in R.

4 Results

Before the results from the test of the prototype were collected, all of the 50 products tested were manually categorized in each of three attributes that were used in the prototype. The rankings that were created from the prototype algorithm, and all other algorithms examined in this section, were compared against this manual categorization. Each algorithm created a ranking for each of the three attribute used during the test. To reduce the number of bars in the diagrams the evaluation measurements for each of the three attributes tested were not included. The numbers shown in the diagrams are an average of the evaluation measurements from the three attributes.

Precision, Recall and *F1-measure* were used as to evaluate the relevance of the result sets, as described in chapter 2.4. Precision is the fraction of retrieved products that were relevant according to the manual categorization, recall is the fraction of relevant products that were retrieved by the algorithm and F1-measure is a balanced combination of these two measurements that gives a single number that represents the algorithm's performance. The higher these three numbers are, the better the algorithm is at matching the manual categorization. These measurements do not take the order of the retrieved products into account. The order of the products was considered irrelevant in this first basic evaluation of the algorithms.

One caveat to keep in mind is that only 284 feedback measurements were collected from users during the seven days the prototype was tested. This was due to a lack of visitors to the website. If a larger number of users had visited the website more measurements could have been made and the results in this chapter could have been given with more certainty.

4.1 Performance of prototype algorithm and different cutoffs

In the following diagrams the algorithm that was used in the prototype is evaluated. This algorithm included at least five random products in the result set, which reduces the performance of the algorithm but allows

more products be exposed to users. The prototype algorithm used a cutoff value of 0.3 which means that only products that had a relevance score higher than 0.3 were retrieved by their relevance. Two smaller cutoff values were also tested. No cutoff values higher than 0.3 was tested since these did not give any results at all due to the small number of measurements collected for each product and attribute. These smaller cutoffs were tested without incorporating random products, which could skew the results, in the result set. *Figure 9* shows a comparison between the precision of the various algorithms, *figure 10* shows the differences in recall and *figure 11* shows the differences in F1-measure.

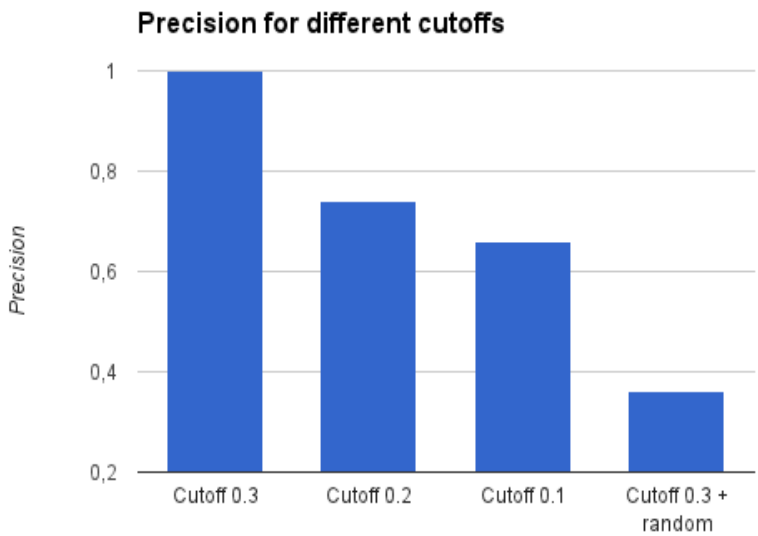

Figure 9: Precision for different cutoffs and the algorithm used in the prototype that incorporated random products in the result set.

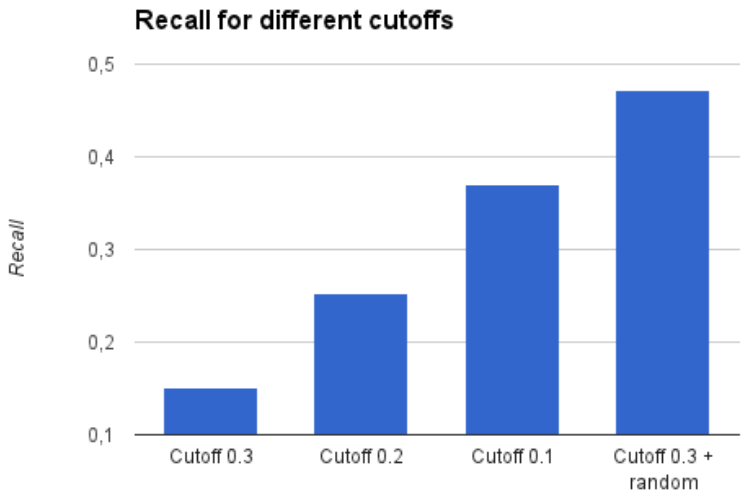

Figure 10: Recall for different cutoffs and the algorithm used in the prototype that incorporated random products in the result set.

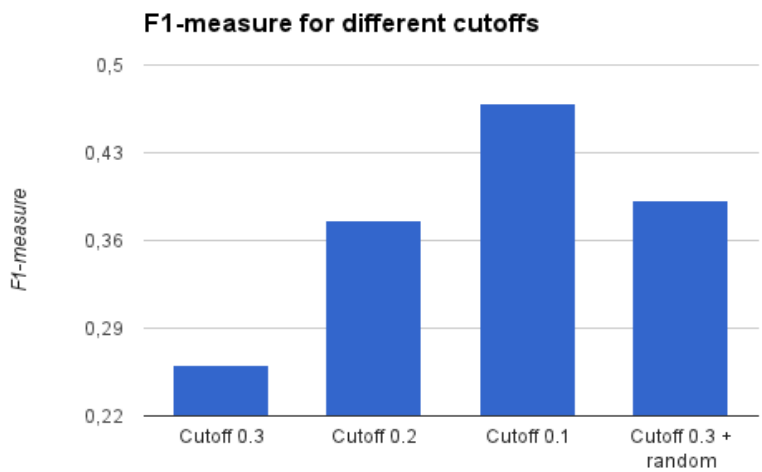

Figure 11: F1-measure for different cutoffs and the algorithm used in the prototype that incorporated random products in the result set.

4.2 Performance of individual user actions

To find out what user actions had the best correlation with the manual categorization, each measured user action was evaluated individually. This means that products had their relevance scores calculated based on a single user action only rather than all five actions combined, which was used in the prototype and the evaluations in section 3.6.2.1. The cutoff values used in this evaluation were 0.3 for all user actions except for *favorite* and *go-to-store* which only generated any results when the cutoff point was lowered to 0.2. *Figure 12* shows a comparison between the precision of the various algorithms, *figure 13* shows the differences in recall and *figure 14* shows the differences in F1-measure.

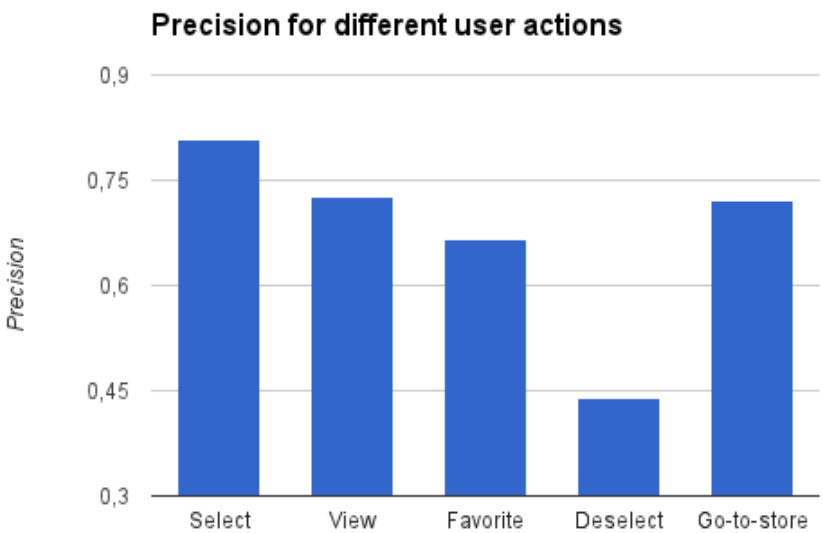

Figure 12: Precision for different user actions.

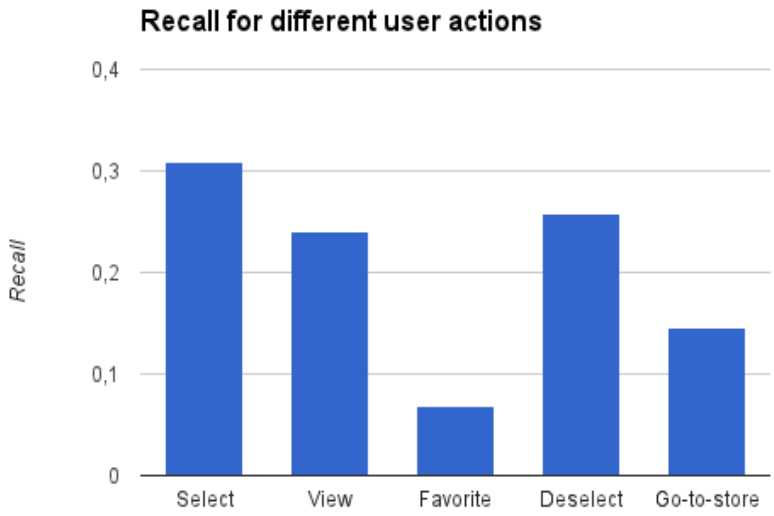

Figure 13: Recall for different user actions.

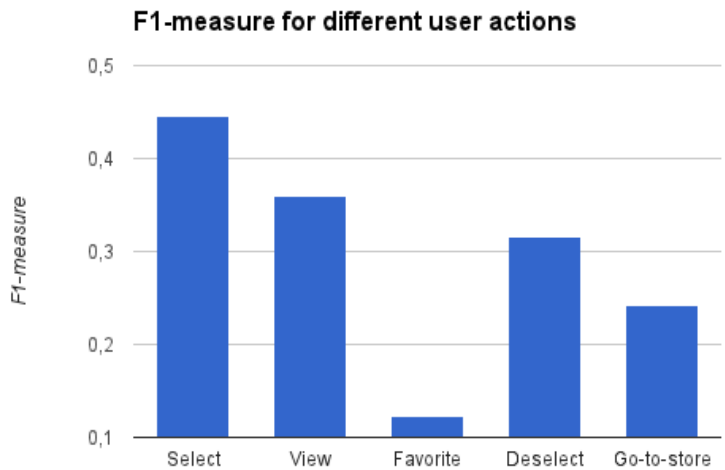

Figure 14: F1-measure for different user actions.

4.3 Performance of different sets of weights

In this section three new sets of weights are tested. These new weights were calculated by finding the user action with the highest value of precision, recall or F1-measure, then giving it a weight of 1. The other weights were then weighted relative to that user action. This was done to find a more efficient set of weights that could generate more relevant results. *Figure 15, figure 16* and *figure 17* contain a comparison with the original prototype algorithm so that possible improvements can be visualized.

The weights used in the algorithm based on precision were as follows:

- Select: 1
- View: 0,898379971
- Favorite: 0,824742268
- Deselect: −0,544918999
- Go-to-store: 0,89347079

The weights used in the algorithm based on recall were as follows:

- Select: 1
- View: 0,780260708
- Favorite: 0,219739292
- Deselect: −0,834264432
- Go-to-store: 0,472998138

The weights used in the algorithm based on F1-measure were as follows:

- Select: 1
- View: 0,804225566
- Favorite: 0,275851884
- Deselect: −0,708993857
- Go-to-store: 0,544186047

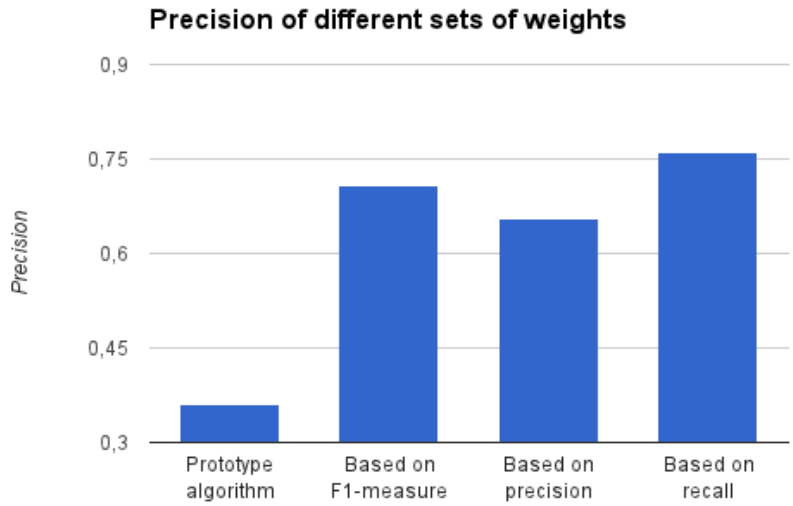

Figure 15: Precision for different sets of weights, compared with the original prototype algorithm incorporating random products.

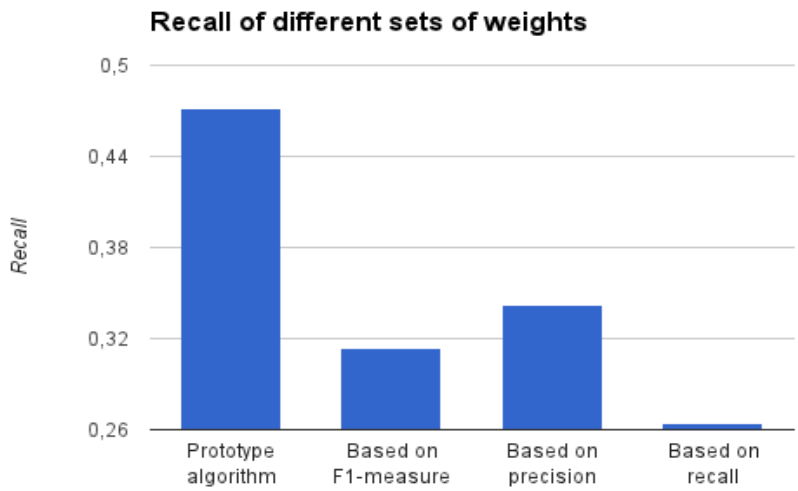

Figure 16: Recall for different sets of weights, compared with the original prototype algorithm incorporating random products.

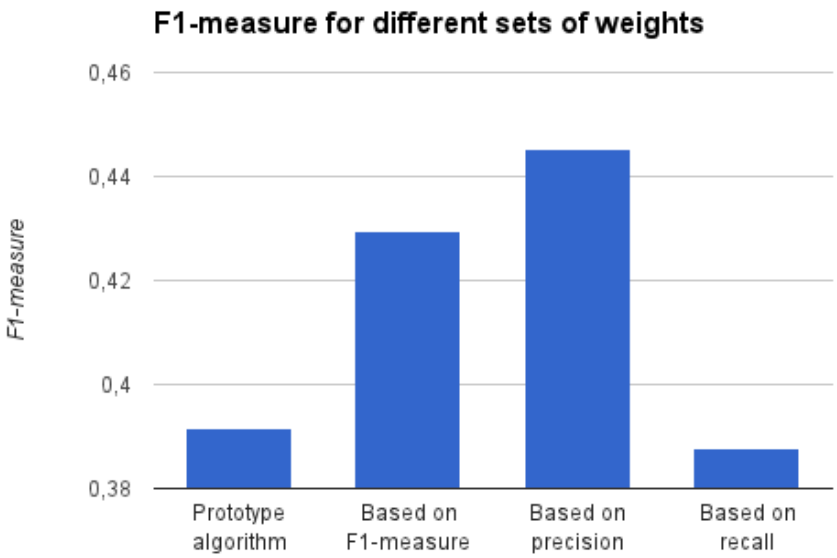

Figure 17: F1-measure for different sets of weights, compared with the original prototype algorithm incorporating random products.

5 Discussion

In this chapter the results from the prototype and the project as a whole are analyzed. Some aspect of the degree objectives for this program, such as those involving environmental science and occupational safety and health, are not discussed in this thesis but have been studied in previous courses.

By examining previous studies done on the use of implicit feedback in ranking systems we described a reliable way of interpreting user actions on a websites as relevance votes. These votes were then put through the Wilson score interval algorithm, as used by a large social media website, to generate a fair score of how relevant a specified product is to a specified attribute.

The most flexible way of capturing user actions was shown to be JavaScript. By using JavaScript on the website, the capturing of user actions could be completely separate from the server which makes it possible to implement it on any website regardless server software. To store the measurements of user actions and generate rankings, the web application framework Ruby on Rails and the database manager PostgreSQL were used. These systems had an adequate set of features and had the benefit of already being setup and available to use on a server from the Heroku cloud service.

5.1 System performance

In this section the performance of the algorithms tested in section 3.6.2 are analyzed and discussed. As the graph in *figure 9* shows, a higher cutoff value leads to greater precision. This means that more of the products included in the result sets are relevant, but also that the numbers of retrieved results are lower as well. When a cutoff of 0.3 where used, the number of retrieved results were unusably low, which is visible in the low recall value.

However, a higher cutoff value also lowers the recall as shown in *figure 10*, which means that a larger number of relevant products fail to be included in the result sets. This is to be expected and means that system is working as it should. According to the F1-measures in *figure 11,* having a cutoff value of 0.1 gives the best balance between precision and recall. Having a cutoff value of 0.1 is, however, not desirable due to the relatively low precision value it generates, compared to higher cutoffs.

The graph in *figure 14* shows that the action of selecting a product from the product grid on the web page was the user action that correlated best with the manual categorization, according to the F1-measure. This suggests that the select action should have a higher weight than the other measured actions. Only a handful of measurements of the favoring and go-to-store actions were collected during the prototype test period. This might have skewed the results for those two actions.

To find a set of weights that was more efficient than the original set that was used in the prototype algorithm, three new sets were created based on the findings shown in *figure 12, figure 13* and *figure 14*. These three new sets of weights were also tested.

As *figure 15* shows, all of the new weights gave result sets with higher precision. Due to the original algorithm incorporating up to 25 random products in the result set, and therefore had a larger result set that by chance included more of the relevant products, it still had a higher recall than the new algorithms with the new weights. This is shown in *figure 16*.

According to the F1-measures shown in *figure 17*, the most efficient sets of new weights were those created by comparing the precision values generated from testing individual user actions. This set of weights was shown to have a 14.8 % higher F1-measure than the original algorithm, which is an improvement. The precision of that set of weights were, however, a more impressive 82.1 % higher. The values generated by all of these algorithms are still far off the perfect score of 1.0, which is what a ranking system would get if all of the relevant products, and none of the non-relevant products, would be included in the result set.

One significant difference between the simulations and the results from the prototype test was how low the relevance scores were overall in the prototype. Only a handful of products had a relevance score higher than 0.4, despite being relevant according to the manual categorization. This can easily be explained by the lack of measured user actions during the test period. The simulation used hundreds and thousands of user actions in the calculations, while the prototype only received tens of user action per product and attribute.

If the system was to run for a longer period of time or be used by a larger number of users, a cutoff value of 0.3 would probably not be effective since the relevance score for both relevant and non-relevant products would increase. In that case, a higher cutoff value would need to be implemented.

To prevent non relevant products from getting a high relevance score closer to 1.0, more negative feedback types would need to be implemented to balance out the positive feedback types. In the algorithms tested in this thesis, the only negative feedback type was the “deselect” action.

5.2 Sustainable socioeconomic development

This system makes the categorization process somewhat more democratic since it is the users themselves who are, indirectly, placing products in different categories. The categorization is not performed by a lone editor who decides where products should be placed, instead it is the users who are in control. This could be especially beneficial if the editor is not a part of the website’s target audience and hence would not categorize products in the same way as the users would.

If this automated system was to be implemented successfully, the workload on the company’s employees would be lessened. There would be no need for employees to perform the repetitive and dreary task of categorizing products. This could allow them to focus on more stimulating and creative tasks instead.

Since this system, at least in theory, will scale automatically according the number of products, categories and users, there would also be no need to hire new editors if the website were to expand in the future. The website could expand faster, cheaper and with more flexibility than if new editors would have to be brought in.

5.3 Privacy

This system was designed to measure user actions completely anonymously since there is no need to endanger user privacy by storing any personal information. The measurements do not contain any names, identification numbers or contact information about the users that have visited the website. The only sensitive information that is collected by the system is the users' IP addresses, but these are scrambled using the cryptographic hash function SHA-1 which turns the address into an unreadable text string of random characters. The reason IP addresses are stored was that there needed to be a way of limiting the number of collected measurements from each individual user so that a single user couldn't flood the system with measurements that could damage the relevance rankings.

There is, however, always a risk that privacy issues would arise if someone with malicious intents were to gain access to the collected measurements. If someone had access to both the collected measurements and a user's browser history, they could probably find out what actions that user performed by looking at the timestamps. The same is probably true if someone also gained access to a log from the web server. This is why security is important on all parts of the system, even if some part seems harmless on its own.

6 Conclusion

This thesis describes a method of using implicit feedback to categorize products on website. Both the mathematics behind the ranking system created for this project and technologies used to build it are described in this report. An overview of the benefits and limitations of implicit feedback, along with a description of how those limitations can be mitigated, is also provided.

6.1 Recommendations

The concept of replacing manual categorization with an automated system that leverages user behavior to create relevance rankings is an exciting one. To be able to evaluate this concept further we would recommend the product owner to implement this system on a website with a larger number of visitors so that more measurements of user behavior can be made. Before implementing this system on a large scale, it would also be appropriate to test it with a larger number of products and attributes. The system might behave differently when it is forced to categorize 50 000 products in 3 000 categories, instead of only 50 products in three categories which was tested here.

As described in section 2.2.8, behavioral bias involving one-dimensional lists has been studied previously, but to the best of our knowledge no studies on the type of two-dimensional grids that are used in this project have been published. This could be one avenue for continued work in this field.

The measurement of user interactions on a website could also be studied from a purely technical perspective. A study comparing different methods of measuring clicks and other kinds of user behavior using cookies, JavaScript, server logs or other more unexpected techniques could be made in the future. This study could perhaps result in a general tool or framework that could be used as a platform to build a ranking system on top of.

References

1. P. Melville and V. Sindhwani, “Recommender Systems”, *Encyclopedia of Machine Learning*, 2010.
2. D.W. Oard and J. Kim, “Implicit Feedback for Recommender Systems”, *Proceedings of the AAAI Workshop on Recommender Systems*, 1998.
3. A. Salihefendic, “How Reddit ranking algorithms work”, http://amix.dk/blog/post/19588. Published 2010-10-23. Retrieved 2014-03-28.
4. E. Miller, “How Not to Sort by Average Rating”, http://www.evanmiller.org/how-not-to-sort-by-average-rating.html,
Retrieved 2014-03-31.
5. T. Joachims, L. Granka, B. Pang, H. Hembrooke and G. Gay, “Accurately Interpreting Clickthrough Data as Implicit Feedback”, *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2005.
6. F. Radlinski and T. Joachims, “Query chains: Learning to rank from implicit feedback”, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2005.
7. E. Agichtein, E. Brill and S. Dumais, “Improving Web Search Ranking by Incorporating User Behavior Information”, *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006.
8. Z. Dou, R. Song, X. Yuan and J.R. Wen, “Are Click-through Data Adequate for Learning Web Search Rankings?”, *Conference on Information and Knowledge Management (CIKM)*, 2008.
9. D. Kelly and N.J. Belkin, “Reading Time, Scrolling and Interaction: Exploring Implicit Sources of User Preferences for Relevance Feedback during Interactive Information Retrieval”, *Proceedings of the 24th Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, 2001.
10. E. Agichtein, E. Brill, S. Dumais and R. Ragno, “Learning User Interaction Models for Predicting Web Search Result Preferences”, *Proceedings of the ACM Conference on Research and Development on Information Retrieval (SIGIR)*, 2006.
11. D.W. Oard and J. Kim, “Modeling Information Content Using Observable Behavior”, *Proceedings of the 64th Annual Meeting of the American Society for Information Science and Technology*, 2001.

12. M. Morita and Y. Shinoda, “Information Filtering Based on User Behavior Analysis and Best Match Text Retrieval”, *Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, 1994.
13. G. Smith and H. Ashman, “Evaluating implicit judgements from Image search interactions”, *Journal of the American Society for Information Science and Technology*, 2012.
14. S. Turner, “Analog 6.0: How the web work”, http://www.analog.cx/docs/webworks.html. Published 2004-12-19. Retrieved 2014-04-17.
15. Ruby on Rails, “Overview”, http://rubyonrails.org/. Retrieved 2014-04-17.
16. Ruby on Rails Guides, “Action Controller Overview”, http://guides.rubyonrails.org/action_controller_overview.html#cookies. Retrieved 2014-04-17.
17. Mozilla Developer Network, “JavaScript timers”, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Timers. Published 2014-03-07. Retrieved 2014-04-13.
18. Mozilla Developer Network, “XMLHttpRequest”, https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. Published 2014-04-01. Retrieved 2014-04-17.
19. Mozilla Developer Network, “HTTP cookies”, https://developer.mozilla.org/en-US/docs/Web_Development/HTTP_cookies. Published 2012-01-26. Retrieved 2014-04-22.
20. P. Eckersley, “How Unique Is Your Web Browser?”, 2010, https://panopticlick.eff.org/browser-uniqueness.pdf. Retrieved 2014-04-22.
21. I. Zeifman, “Report: Bot traffic is up to 61.5% of all website traffic”, http://www.incapsula.com/blog/bot-traffic-report-2013.html. Published 2013-12-09. Retrieved 2014-04-22.
22. Google Webmaster Tools, “Googlebot”, https://support.google.com/webmasters/answer/182072. Retrieved 2014-04-22.
23. C.D. Manning, P. Raghavan and H. Schütze, “Evaluation of unranked retrieval sets”, *Introduction to Information Retrieval*, 2008.
24. S.M. Beitzel, E.C. Jensen and O. Frieder, “MAP”, *SpringerReference,* http://www.springerreference.com/docs/html/chapterdbid/63603.html. Retrieved 2014-04-23.

25. E. Zhang and Y. Zhang, “Average Precision”, *SpringerReference,* http://www.springerreference.com/docs/html/chapterdbid/63593.html. Retrieved 2014-04-23.
26. T. Joachims, “Optimizing Search Engines using Clickthrough Data”, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
27. H. Pokorny, “Ranking algorithm - Lower bound of Wilson score confidence interval for a Bernoulli parameter”, GitHub, https://gist.github.com/honza/5050540. Retrieved 2014-05-14.
28. PostgreSQL, http://www.postgresql.org/about/, Retrieved 2014-05-14.
29. Daniel Kehoe, “What is Ruby on Rails?”, RailsApps, http://railsapps.github.io/what-is-ruby-rails.html. Published 2013-09-11. Retrieved 2014-05-14.
30. Microsoft, http://www.microsoft.com/en-us/server-cloud/products/sql-server/, Retrieved 2014-05-14
31. Ruby on Rails Guides, “Active Record Basics”, http://guides.rubyonrails.org/active_record_basics.html. Retrieved 2014-05-15.
32. Microsoft, “Microsoft .NET Framework””, http://www.microsoft.com/net. Retrieved 2014-05-19.
33. Java.net, “Mojarra JavaServer Faces”, https://javaserverfaces.java.net/. Retrieved 2014-05-19.
34. PHP.net, “PHP: Hypertext Preprocessor”, http://www.php.net/. Retrieved 2014-05-19.
35. Python.org, “Welcome to Python.org”, https://www.python.org/. Retrieved 2014-05-19.
36. jQuery.com, “jQuery”, http://jquery.com/. Retrieved 2014-05-19.
37. Adobe.com, “Adobe Flash runtimes”, http://www.adobe.com/products/flashruntimes.html Retrieved 2014-05-20.
38. D. Goldman, CNN Money, “The beginning of the end for Adobe's Flash”, http://money.cnn.com/2011/11/10/technology/adobe_flash/. Published 2011-11-10. Retrieved 2014-05-20.