Design and Implementation of an Auto Complete Algorithm for E-Commerce

Jonathan Persson (jonathan.persson.85 **at** student **dot** lth **dot** se)

Department of Electrical and Information Technology
Lund University

Advisor: Mikael Hammar

Examiner: Anders Ardö

November 30, 2010

Printed in Sweden
E-huset, Lund, 2010

Preface

This report is the result of a Master's Thesis at the Faculty of Engineering at Lund University (LTH), which was done at Apptus Technologies in Lund, Sweden. In this thesis I have had the opportunity to experiment with a rather large data set containing searches resulting in purchases, a data set full of possibilities. This data contains information about which queries that have led to the purchase of which items and at what time. The suspicion was that this data would be a very good base on which to build an auto complete upon.

I would like to thank Mikael Hammar at Apptus Technologies for many hours of valuable discussions and advice. I would also like to thank Anders Ardö at LTH for his valuable input and advice.

Abstract

In this paper two main ways of ranking are designed and presented for auto complete drop-down lists on e-commerce sites: Probabilistic Ranking, and Popularity Measurement. These rankings are based on data of search events that have led to the purchase of one or more products. The goal was to design a ranking algorithm that maximizes both usability and the commercial effectiveness of the auto complete.

This thesis shows that using this data as the set of possible auto complete suggestions gives very good results. If the number of purchases ever followed by the query is used as the query's rank, both usability and commercial effectiveness is maximized. This is how the first way of ranking works. The second way of ranking also takes time into account and ranks based on the latest purchase frequency.

The different rankings in this paper and variations of them are tested and compared. Testing has been possible due to the available data, which in fact contains use cases of what users have searched for and then purchased. During the testing the goal has been to primarily maximize the rate of successful auto completions, and secondly to minimize the average required input length before the correct completion is found. Increased commercial effectiveness of the auto complete can unfortunately not be tested without real users.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The World Wide Web is an immense network of websites and was estimated at 11.5 billion pages in 2005 [1]. Finding relevant information in this vast network of billions of pages has always been a challenge for the web search engines, and sometimes for the users utilizing them.

Electronic commerce (e-commerce) is today a very common way to sell things, and a very convenient way for the customers. But e-commerce faces a similar challenge as the web search engines: large retailers might have to help tens of millions of customers to find the right items in a catalog with millions of items [2]. Therefore every tool available is used to help bring the products to the customers. Among these are for example recommenders, which have become very common on e-commerce sites. They recommend items to customers in different ways, for example with a panel on a product page labeled: “Others who bought this item also bought...,” or a panel on the search result page labeled: “Customers who searched for ... bought these items.”

Recommenders help a great deal, but one still needs to type the whole query in the search box, which might be long, which in turn makes misspellings more likely, and if the search engine is not forgiving enough, one might end up without hits.

This is where the auto complete comes in. The auto complete is a drop-down list populated with suggestions of what one can write in the search box (see Figure 1.1). When a customer starts to write a few letters of the beginning of a query the list appears below the search box and is populated with query suggestions that start with the same letters as he has already typed in, or in another way resembles the input. The customer can then navigate in this list and select the right suggestion. This spares him of having to formulate the complete query, which saves time and frustration and helps customers when they did not remember the complete product name.

The auto complete has become a very popular function on the web and on e-commerce sites, simplifying search a great deal for the end-user. But on e-commerce sites it is not only a convenience for the customers, but also an opportunity for the site to show customers a selection of products they might be interested in, like a recommender, or a form of directed advertising. So the question is: how can one create an auto complete that is both useful and profitable?

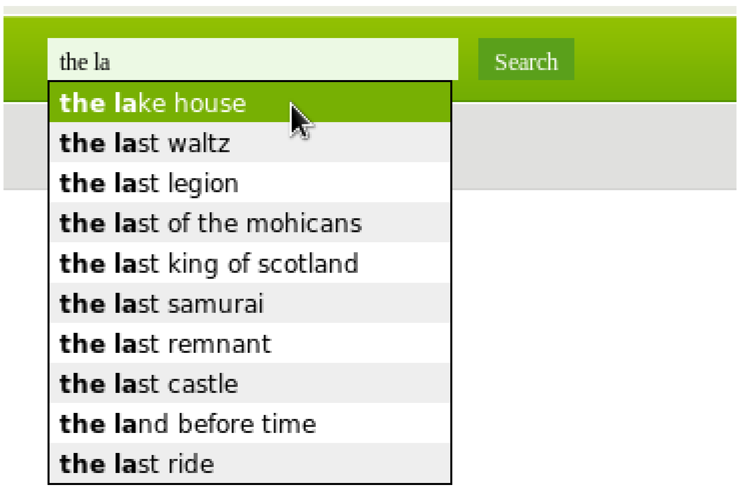

Figure 1.1: The graphical auto complete web function implemented using jQuery.

1.2 Related Work

Not much related work was found during the studying phase of this thesis. One document was found about an auto complete for the address field in a web browser, called “Using Supervised Learning to Improve Url Ordering in Autocomplete” by Ranjan et al. [3]. The report is a study using a softmax like probabilistic model to rank URL completions in the Mozilla browser. Therefore it only seeks to maximize usability, where this thesis also seeks to maximize profit. Furthermore, the available data in this thesis has fewer parameters than the data used in the report. Therefore the approaches in this thesis were considered to be more promising for an e-commerce site with the data available, and in lack of time an approach based on neural networks is left out in this thesis.

“Type less, find more: fast autocompletion search with a succinct index” by Bast et al. [4], and “Output-sensitive autocompletion search” by Bast et al.[5], are two articles that discuss autocompletion of the last word of the input for web search. This might be usable on e-commerce sites, using product data with text from for example descriptions, titles and artists. But completing the last word of the input by looking at product data is just what the existing auto complete developed by Apptus Technologies does, which does not work satisfactory for e-commerce sites. This is both due to the fact that just suggesting the continuation of the last word instead of a complete query requires much more work from the user, and due to the fact that using product data has the limitation that it does not show what is popular. Therefore the theories from these articles are not pursued.

There is a lot of literature about recommenders, which normally comes to calculating similarity between users and users or products and products, such as “Item-Based Collaborative Filtering Recommendation Algorithms” by Sarwar et al. [6] but no promising applications of these theories were found for the problem in this thesis.

1.3 Overview

The thesis begins with a formulation of the problems and goals in Chapter 2. Chapter 3 goes through the different rankings that are later tested. The testing procedure, criteria and other issues are described in Chapter 4, and the test results are found in Appendix A and B. These results are interpreted in Chapter 5, and the findings are summarized in Chapter 6. In Chapter 7 a few other web functions where the algorithms could be applied are mentioned.

Chapter 2

Problem

2.1 Goals

The goal is to maximize the long-term profit from the auto complete, and this is of course a hard problem. This will be handled by setting up more specific goals that bring functionality to customers at the same time as pushing profitable products. Here are the goals for the auto complete.

1. The auto complete should rank queries that are thought to be more profitable, or commercially effective, higher without noticable loss in usability.
2. If the user has a specific query in mind, the auto complete should after as few characters as possible suggest the wanted query.
3. If the user has a specific query in mind, the auto complete should as often as possible find the wanted query.
4. The auto complete should as often as possible be able to offer suggestions.

In short the goal is to create an auto complete that maximizes the commercial effectiveness of the suggestions, as fast as possible guesses what the customer wants and in as many cases as possible has something to suggest.

2.2 Problem Formulation

The auto complete is a list of suggestions of what one can write in the search box to reach different products or categories. These suggestions will also be referred to as *query suggestions* or *completions*. After one has written a few letters of the beginning of the query and the list is populated with query suggestions that in some way match the input. In the normal case matching means that the suggestion starts with the input.

One assumes that one has the following situation: a random customer on an e-commerce site is typing into the search box, and one is given the n first letters of the intended query q. This is the input or the “query prefix” to be completed. Two main problems are now defined, where the second is the focus of the thesis:

1. Which suggestions are considered to match the input? This is what will be referred to as matching (Section 3.1).

2. How can one find the x matching suggestions that best fulfill the goals? This is discussed in Section 3.4.

To solve these problems the data described below is available. One can also assume that the system can be continuously updated once every 15 minutes, and at these occations receive new data.

2.3 Data

The data studied in this thesis comes from a leading Scandinavian online store for mainly music, movies and books, which did not have an auto complete at the time the data was collected. The data available is the search-to-purchase data, and the product data.

2.3.1 Search-to-Purchase Data

The search-to-purchase data contains search queries that have led to purchase. This is given in tuples of:

transaction id, **query**, **product key of purchased product**, **time stamp**

The data is taken from the time period February to September 2009 inclusive, contains 1,921,652 tuples, and is about 300 MB in XML format. Only a small assessment of the data quality has been done, which shows that the data quality is quite good (see Section 3.2.1).

2.3.2 Product Data

The product data contains a set of about 120,000 products from the product catalog with product key and various attributes such as artists, authors, description, directors and title.

2.4 Use Cases

In this section a few possible scenarios are described, when a user is writing in the search box and has different intentions. These are provided to give better insight into the problem. A product will be denoted by enclosing a text in brackets, and a query with quotation marks.

Case 1:

- User types: “the ” and intends to write “the lord of the rings: the fellowship of the ring”.
- User selects: “the lord of the rings: the fellowship of the ring” in the suggestion list.
- User buys: [The Lord of the Rings: The Fellowship of the Ring].

Case 2:

- User types: “the ” and intends to write “the lord of the rings”.
- User selects: “the lord of the rings: the two towers” in the suggestion list.
- User buys: [The Lord of the Rings: The Two Towers].

Case 3:

- User types: “the ” and intends to write “the lord of the rings: the two towers”.
- User sees in the suggestion list that there is a new title he did not know about.
- User selects: “the lord of the rings: the return of the king” in the suggestion list.
- User buys: [The Lord of the Rings: The Return of the King] and [The Lord of the Rings: The Two Towers].

Case 1 and 2 represent normal simple cases when a user, as in Case 1 is looking for a specific product and then buys that product, or in Case 2 when the user could be looking for a more general topic and buys a product under that topic. In Case 3 the user is looking for a product but when he sees another item in the list he gets interested and buys that too. Especially in Case 2 and Case 3 the auto complete has the possibility to affect the customer’s choice into one that would be more beneficial for the retailer, such as a choice that would make him more satisfied and return as a customer. Unfortunately, cases where the users change their minds will not be tested since that would require tests on real users. The weaknesses in the testing will be discussed further later on.

Chapter 3

Analysis

3.1 Matching

The first problem that needs to be solved is finding the possible suggestions that should be considered showing, that is: which suggestions are considered as *matching* the input?

On the web this is done in at least two ways. If one for example considers a list of products and wants to auto complete based on the title, the probably most common way to select the possible titles to show is to take all titles that start with the user input. Another way is to also consider titles that don't have the right word order but otherwise match. These two ways have been tested in this thesis and are described in more detail, as implemented in this thesis, below.

First a few terms have to be defined. What needs to be done is to match the user's *query prefix* with existing *queries* from some kind of database. Query prefixes are here viewed as a number of *words* where the last word is viewed as a *word prefix* if and only if there is no white-space following it, e.g. "the last of the mohi" contains four words and a word prefix whereas "star wars " contains two words and no word prefix.

Below the two ways of matching that have been tested in this thesis are described.

3.1.1 Order-unspecific matching

This way of matching was done by saying that a query prefix matches a query if the query contains every word from the query prefix. If there is a word prefix in the query prefix, there has to be a word in the query that starts with the word prefix.

This means that for instance "michael jackson ", which has no word prefix, matches "michael jackson", "jackson michael" and "michael jackson thriller", but not "michael jacksonn".

As another example "the last of the mohi" (where "mohi" is a word prefix) matches "the last of the mohicans", "the mohicans of the last", but not "the last of the mo".

3.1.2 Order-specific matching

This way of matching was done by saying that a query prefix matches a query if and only if the query prefix is a prefix to the query (or identical to the query).

Therefore with this matching “michael jackson ” matches “michael jackson thriller” but not “jackson michael thriller”

3.2 Data

There is a big limitation with using product data to to rank suggestions. For example, the most common continuations of the input in the product data can be suggested. In that case suggestions of what occurs most in the data are shown in the auto complete, such as artists that have produced many albums, but it does not suggest what is popular or how people choose to refer to a product, which does not have to be by the title.

Search-to-purchase data on the other hand shows how people usually refer to a product when they make a purchase, and how often they do so. Therefore one can rank based on what is popular to write, but only when the query has led to a purchase, so it will not be the true popularity of a query. It is later shown that this has a very interesting effect.

3.2.1 Correctness of the Data

The data is collected by for every purchase recording what the last query in the same session was. This data cannot be completely accurate since users can search for one thing and then browse for something else and purchase an unrelated product. Hence, a small manual assessment of the data quality has been made for 100 consecutive data tuples on a random location in the data. Although such a small assessment cannot be completely trusted, the results show that the data is actually surprisingly correct.

Out of 100 cases, in 95 cases the product key was obviously related to the query (for instance the query was the name of an artist, a song track, a title or part of the description of the product). Out of the remaining five, three were completely unrelated, and two did not exist in the available product data.

Another thing to point out is that the search-to-purchase data (and the product data) contains information in various languages mixed up. This should give worse tests results than when single language queries are used and the data has a single language, since there is a bigger set of queries to choose from, but it does not change the correctness of the data.

3.3 Conversion Rate

Conversion rate on an e-commerce site is an important measure of Web-based advertising effectiveness and is defined by the percentage of visits that convert to orders [2]. The conversion rate could also be calculated for a specific product as a way of measuring how commercially effective a product is, or even for a specific suggestion as a way of measuring how commercially effective a suggestion is. Maximizing the conversion rate for a suggestion is thus to maximize the commercial effectiveness for a suggestion, which was one of the goals. The conversion rate of a suggestion or query will in this thesis more precisely be defined as: the percentage of times the query is searched for that lead to purchase.

3.4 Ranking of Suggestions

3.4.1 Ranking 1: Probabilistic Ranking

In this section the fundamental discovery of this paper is described. It is here shown that the search-to-purchase data theoretically is a very good source to use for the auto complete ranking. This ranking will be referred to as the Probabilistic Ranking (PR).

Recalling the goals of the auto complete, these could be achieved by maximizing the *probability of that a suggestion is first selected and then a product is bought.* In the calculations below the following situation is assumed: a query prefix to be completed and the set of matching suggestions are provided, and one can assume that one out of these suggestion will be selected by the user.

Let P be the event of a purchase of an item, and S the event of a search on the query $Q_a \in Q$, where Q is the set of all suggestions in the data set. One can then calculate the probability of a purchase given that the query Q_a was selected:

$$P(P|S) = \frac{p_{Q_a}}{s_{Q_a}} \tag{3.1}$$

where p_{Q_a} is the number of purchases after searching on Q_a, and s_{Q_a} the number of searches on Q_a.

The definition of conditional probability [7] states that

$$P(P|S) = \frac{P(P \cap S)}{P(S)} \tag{3.2}$$

therefore $P(S \cap P)$, the sought probability of selection and purchase, is defined by

$$P(S \cap P) = P(P \cap S) = P(P|S) \cdot P(S) \tag{3.3}$$

$P(S)$, the probability of selecting Q_a, can be calculated by

$$P(S) = \frac{s_{Q_a}}{\sum_{Q_i \in Q} s_{Q_i}} \tag{3.4}$$

Inserting Equation 3.1 and 3.4 into 3.3 gives

$$P(S \cap P) = \frac{p_{Q_a}}{s_{Q_a}} \cdot \frac{s_{Q_a}}{\sum_{Q_i \in Q} s_{Q_i}} = \frac{p_{Q_a}}{\sum_{Q_i \in Q} s_{Q_i}} \tag{3.5}$$

This is the probability that a suggestion will be selected and result in the purchase of an item. If one uses Equation 3.5 in ranking suggestions, one can skip the division by $\sum_{Q_i \in Q} s_{Q_i}$ and rank solely on p_{Q_a}, since $\sum_{Q_i \in Q} s_{Q_i}$ is constant for every matching suggestion set.

This is a very impressive result; it's very simple, powerful and the only needed data is the number of purchases followed by every query, which can be found in the search-to-purchase data. This also means that no run-time calculations and no pre-calculations are needed, since a commercial system can store a counter for every query that is incremented every time a new search-to-purchase event is detected.

As very nice bonus with this ranking model, it is found from Equation 3.3 that the rank $P(P \cap S)$ contains the factor $P(P|S)$, and as seen in Equation 3.1 this is exactly the conversion rate for a query as defined in Section 3.3: the percentage of times the query is searched for that lead to purchase.

One can of course not rely only on conversion rate because that could show queries to very unpopular products that always are bought by the few who like them. However with the probability of selection included in the rank, the queries to these unpopular products get a much lower rank.

Another good quality about the number of purchases is that it is intuitively a good indicator: products that sell much are probably liked by many, and it seems clever to make selling products more accessible. Moreover, customers who are satisfied with an item might recommend it to their friends, increasing the number of purchases, so queries to items that satisfy customers to some extent get a higher rank, which in turn hopefully leads to more satisfactory purchases. Also, if the auto complete shows what people buy rather than what they type one could say that they get what they want, not what they asks for.

A problem with the probabilistic approach is that it assumes that no matter how old a query is it is equally likely that it will be searched for again. This is obviously not true as, for example, products often are more popular shortly after they are released, or after a commercial campaign. This problem is of course worse the longer the time span of the data is, and will be addressed in the next ranking approach.

Machine Learning

The Probabilistic Ranking is meant to be used in a system with continuous updates. What the users select, and thus what is added to the database, will depend on the output of the auto complete itself. Therefore the auto complete is self-learning: for example if users always select the tenth suggestion after a specific prefix, it will climb up the suggestion list to eventually be first. This is a very good thing which also, for example, implies that if the majority of the users select the correctly spelled item in the list when many alternate spellings are presented, the auto complete will learn the correct spelling of items and place them first.

On the other hand, if a query is misspelled the first time and the user still finds the product and purchases it, the misspelled query will be auto completed in the future, which could lead to more users selecting the misspelled query and might prevent the correct spelling from being added, and if it is eventually added the question is if the majority of the users will choose the misspelling on the first place or the correct spelling on the second place.

This kind of phenomena could lead to unexpected results, both good and bad, and although it is very interesting, to study these would require studying how the auto complete changes with time on an e-commerce site with real users, which is outside the scope of this paper.

Properties of the Probabilistic Ranking

To sum up the good properties, and the bad properties of the Probabilistic Ranking will be listed. The good properties are the following:

- The PR ranks on the probability of selection followed by purchase, a simple and intuitive rank that aims to fulfill the goals.
- The PR has a simple rank (p_{Q_a}) requiring neither run-time calculations nor pre-calculations.
- Queries with high conversion rate get higher rank.
- The PR is self-learning in a continuously updated system (might also be bad).

The only noted bad properties are the following:

- The PR does not take time into consideration, so queries that have been popular but no longer are popular will still have a high rank.
- The PR is self-learning in a continuously updated system. This might have bad effects, but this cannot be studied in this thesis.

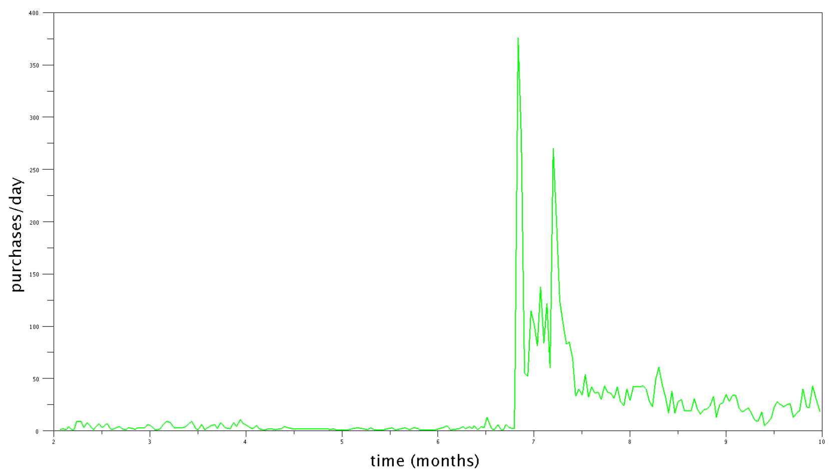

Figure 3.1: Purchases per day after searching for "michael jackson", February to October 2009

3.4.2 Ranking 2: Popularity Measurement

In Figure 3.1 one can see one of the reasons why measuring popularity is so important. The plot shows the number of purchases per day after searching for "michael jackson" from February to October 2009. In the end of June, when Michael Jackson died, there is a huge spike, followed by a higher popularity level. In this section this kind of curves, and which properties that can be changed to get the best ranking, is studied.

Since the previous approach did not take aging and trends into account, an attempt is made to rank suggestions based on how popular a product is right now, which is possible since the system is updated every 15 minutes. By doing this one could even have different reoccurring auto completion results at different hours of the day, depending on what people tend to buy at these hours. Below different ways of measuring popularity are presented, which all view popularity as the frequency: purchases per time unit. These ranking methods will be referred to as Popularity Measurement (PM) ranking methods.

Like before, the total count of purchases connected to a given query is denoted by p_{Q_a}.

Approach 1 - The Simple Way

The simplest way of making sure the Probabilistic Ranking (PR) does not base its rank on outdated data would probably be to cut off every purchase that is older than a certain limit, say for instance a week, i.e. measuring p_{Q_a} the last x time units. This method has the benefit that the PR approach applies just like before without modification since the only difference is that a data set from a shorter time period is used. But all older data is then lost, which especially for rare queries can be a significant loss. Some rare but reoccurring queries, like classical music, might perhaps not even have been entered during the last week and would be lost.

The parameter that can be experimented with here is the look-back time x. If x is small, data is lost but the results are more recent, and if it is large more data is kept but it is less recent on average.

Approach 2

This approach tries to cope with the weakness of the previous one by always keeping a high enough number of time stamps for every query, and rank based on the latest frequency of purchases.

The purchase count during a time period t_c is here denoted by c. The basic idea is to measure c/t_c for the last n time stamps, where n is a fix number. This means that the actual rank can be calculated by

$$c = min(p_{Q_a}, n) \tag{3.6}$$

$$rank = \frac{c}{t_c} \tag{3.7}$$

If there are fewer than n time stamps, c is set to the number of available time stamps, p_{Q_a}, and t_c is set accordingly, i.e. to the time elapsed since the first purchase. Thus one always ranks by the frequency: queries/time unit. Figure 3.2 illustrates the selection of c and t_c on a time axis.

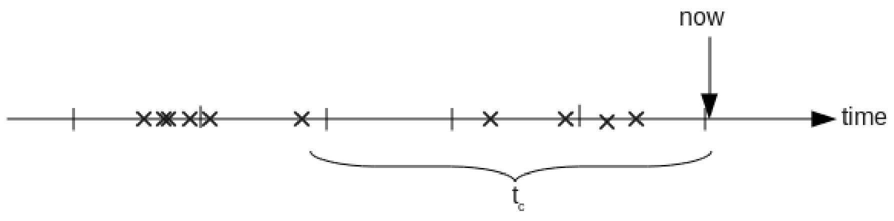

Figure 3.2: Purchases in time. In this example n is set to 5, which gives $c = 5$ since $p_{Q_a} > n$, and t_c becomes the time elapsed since 5 purchases ago.

This approach makes sure that one ranks by as recent data as possible while keeping enough time stamps to properly determine the frequency. It also has the benefit that rare queries are never left out and get a fair rank. Also note that if n is set to 1, one gets Approach 1 or the "The Simple Way" from above, which means that Approach 1 is a special case of this approach.

Insecurity Punishment
However, there is a problem here. If for example, an unusual query that has never been entered before happens to be entered twice, it can get a very high rank if it's recent enough. Therefore one might want to add something to adjust for the insecurity in determining the popularity of a query with few purchases.

One way of doing this is to say that the number n is the limit above which it is assumed that there is enough evidence about the popularity of the query. Then unsure queries are punished using a rank (as a function of c)

$$rank = \begin{cases} f(c)/t_c, & \text{if } 0 < c < n, \\ c/t_c, & \text{otherwise.} \end{cases} \tag{3.8}$$

$$f(c) < c, \quad \text{for } 0 < c < n \tag{3.9}$$

$$f(n) = n \tag{3.10}$$

The condition in Equation 3.10 given to connect $f(c)$ with the line $g(c) = c$ in the point $c = n$ so that there is no jump in rank from queries with $n - 1$ time stamps to queries with n time stamps. One possible function is $f(c) = \frac{1}{n^2}c^3$, and another is $f(c) = \frac{1}{n}c^2$ which will be referred to as Cubic and Quadratic Insecurity Punishment.

Another concern for popular queries is that the rank could vary too much from one minute to another. Without testing, there were no imaginable means to know how quickly a rank should be allowed to change to best reflect the current popularity value. Therefore, two ways of evening out the rank over time are described below, and will later be tested.

Introducing Look-Back Time in Approach 2
To prevent the rank from changing too fast a minimum look-back time (in the future termed as LB) is introduced, making the algorithm in its simplest form as follows:

Algorithm 1.
Input: p_{Q_a}, n, LB
Output: $rank$

```
if $p_{Q_a} > n$
        set $c = n$
else
        set $c = p_{Q_a}$
end
set $t_c = time(c)$
if $t_c < LB$
        set $t_c = LB$
        set $c = count(LB)$
end
set $rank = \frac{c}{t_c}$
```

The function $time(c)$ returns the time difference between the current point in time and the point at c purchases backwards, and the function $count(t_c)$ returns the number of purchases that have occurred since t_c time units back in time.

Adding a Constant in the Denominator
Another way of preventing the curve from varying too fast is to add a constant in the denominator, making the formula for the rank: $rank = \frac{c}{t_c+const}$, with the same rules for selecting c and t_c. This formula gives an overall smoother rank curve, not only in intense regions as was the case with the minimum look-back time approach (this will be confirmed in the plots).

The smoothening comes from that a curve on the form $\frac{1}{t+const}$ moves to the left if $const$ increases, so that t, which is greater or equal to zero, cannot reach the steep slope where $1/t$ goes to infinity near $t = 0$. Consequently, with a higher $const$ the rank will not explode for short time differences.

Plotting
Now experiments by altering n and LB can be done. When n is increased, the mean value is taken for more points and thus the variance in the curve decreases, as shown in Figure 3.3.

When LB is increased the mean value is taken over a longer time, and thus the variance in the curve here too decreases. But, as seen in Algorithm 1, LB only influences the curve where $t_c < LB$, and for that reason using LB has a much stronger effect with low n-values, because low n-values make the average t_c shorter. This is shown in Figure 3.4 and 3.5. One can see that in Figure 3.4 where n is set to 5, the lines with different LB values vary more than in Figure 3.5, where n is set to 10.

Figure 3.6 shows a comparison between the rank curves for three different queries. “wow” was more popular than “forrest gump” and “korn” during the month in question, but for almost five days “forrest gump” had a higher popularity. In Chapter 5 test results are presented of best combination of n and LB.

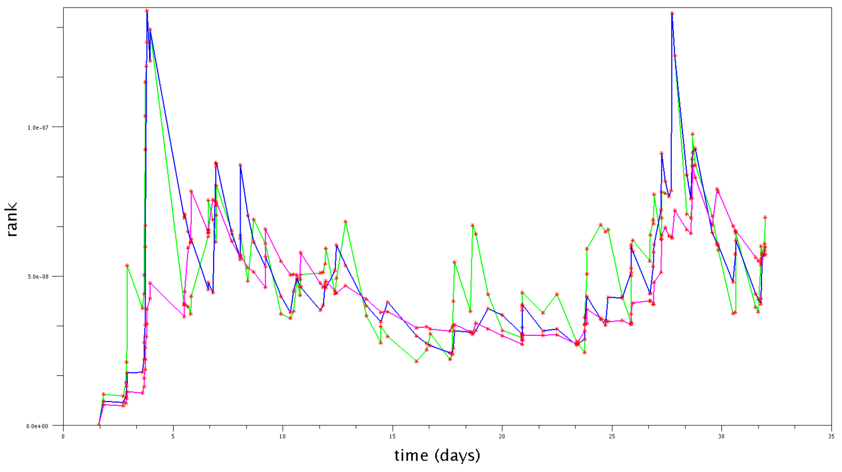

Figure 3.3: Approach 2 rank at different time instances over a month's time for the query "wow". The following constants are set: $LB = 1$ day, $n = 5$ (green), 10 (blue) and 20 (magenta).

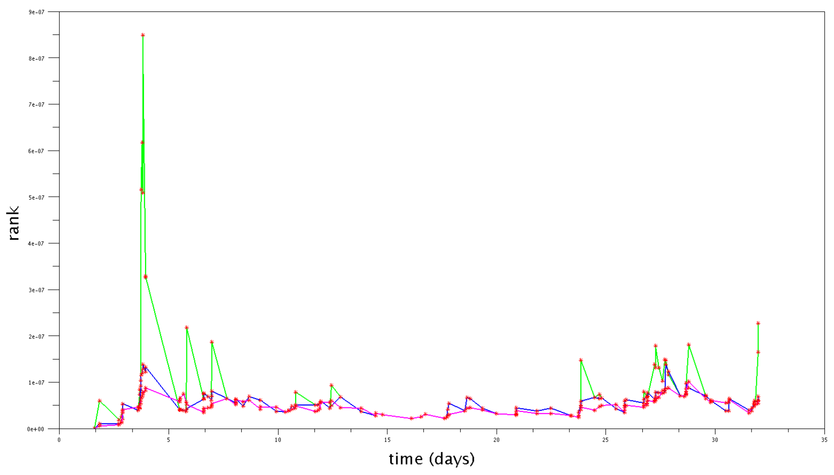

Figure 3.4: Approach 2 rank at different time instances over a month's time for the query "wow". The following constants are set: $n = 5$, $LB = 1$ ms (green), 1 day (blue) and 2 days (magenta).

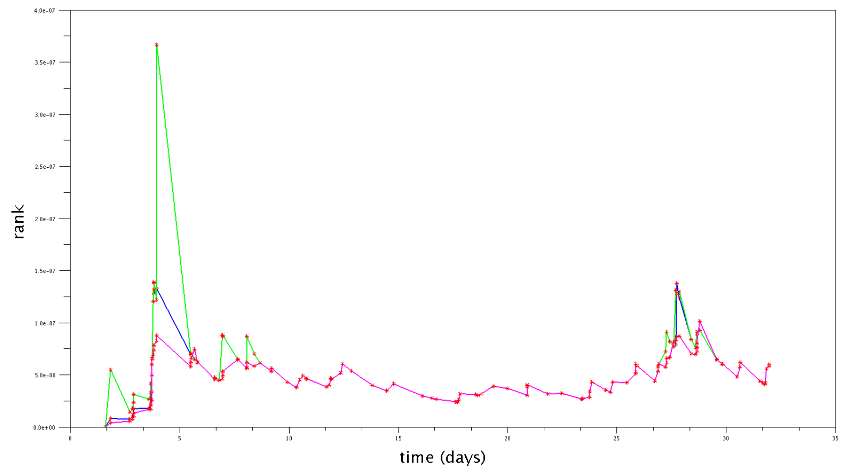

Figure 3.5: Approach 2 rank at different time instances over a month's time for the query "wow". The following constants are set: $n = 10$, $LB = 1$ ms (green), 1 day (blue) and 2 days (magenta).

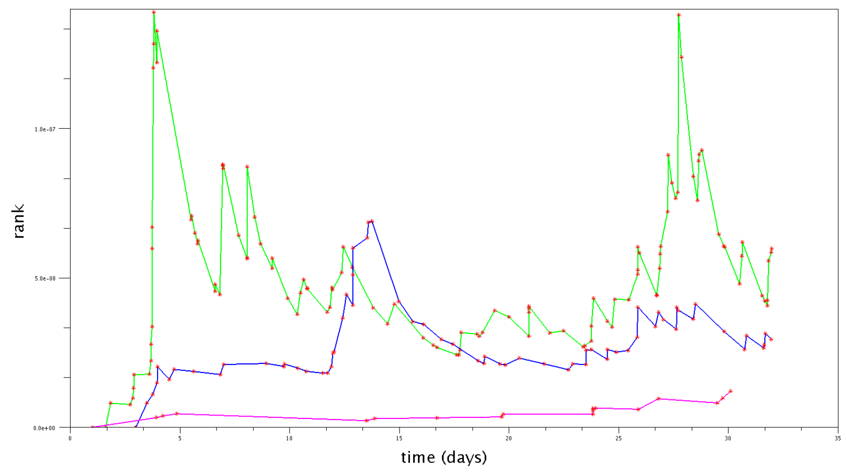

Figure 3.6: Approach 2 rank at different time instances over a month's time for the queries "wow" (green), "forrest gump" (blue) and "korn" (magenta). The following constants are set: $n = 10$, $LB = 1$ day. "wow" was obviously more popular than "forrest gump" and "korn" during that month, but for almost five days "forrest gump" had a higher popularity.

Approach 3

To better explore the options an attempt is made to calculate the recent mean purchase frequency in a different way. This is here done by taking the mean of $\frac{1}{t_i - t_{i-1}}$ for the last n time stamps. This is expressed with the following mathematical expression:

$$\frac{1}{n} \sum_{i=m-n+1}^{m} \frac{1}{t_i - t_{i-1}} \tag{3.11}$$

where m is the index of the last time stamp. This can, like in the previous approach, be extended to use a minimum look-back time, a constant in the denominator and insecurity punishment. A constant in the denominator is rather necessary in this case as seen in Figure 3.7, since the rank can vary very much from one moment to another.

Equation 3.11 is now plotted for some different queries connected to purchase time series. A function plot point corresponding to the rank is calculated for every time instance in the time series, as if it were a moment when an auto complete ranking had to be done.

Figure 3.7 shows how the rank changes during a month for the query "wow", which has 119 purchases, and the query "korn", which has 18 purchases. First one can see that the values change very much, which is a bad sign. Second, the query "korn" gets a very high rank in the end, that is even close to the max of "wow". That is not acceptable, since one can see by the number of red plot points that "wow" is much more popular all over and also has a much higher total number of purchases.

The reason for this is that the division in Equation 3.11 is done for every pair of points and then the mean value is taken, which makes it more probable to land on the steep part of the curve $1/x$. In Figure 3.7 one can see that there are two very close pairs in the end of the curve for "korn", and starting there the points have a very high rank since they include the close pairs in their mean value calculation.

In Figure 3.8 the result after introducing a constant equal to 2 hours in milliseconds is shown. This plot looks much better, but "korn" is still in the end in level with "wow", which is not acceptable, and unfortunately this doesn't change with introducing look-back time or changing parameters in other ways, so this method clearly has problems. One can also punish insecurity in this method, and will do so in the testing.

Relation to the Probabilistic Ranking

This section explains how Popularity Measurement relates to the PR, and it is important that the good properties from the PR remain even here. The rank c/t_c from Approach 2 is compared first. As mentioned before, using look-back time and an n set to 1 gives Approach 1, which is the same as the PR with data from a shorter time period and thus it must have the properties of the PR. But what happens when n is higher, which allows c to be higher? If t_c is smaller than LB, one still looks back the look-back time so the Probabilistic Ranking still holds, but when t_c is greater than LB the situation is different.

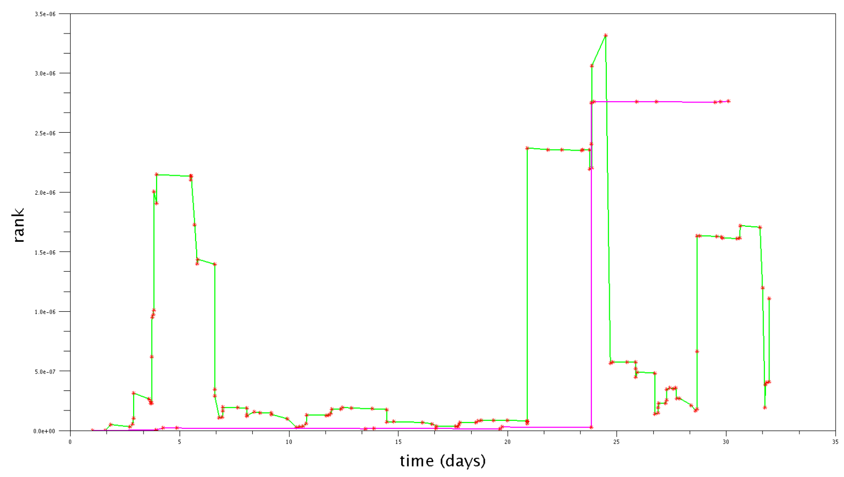

Figure 3.7: Approach 3 rank at different time instances over a month's time. The query "wow" in green and "korn" in magenta. The following constants are set: $const = 1$ ms, $n = 10$.

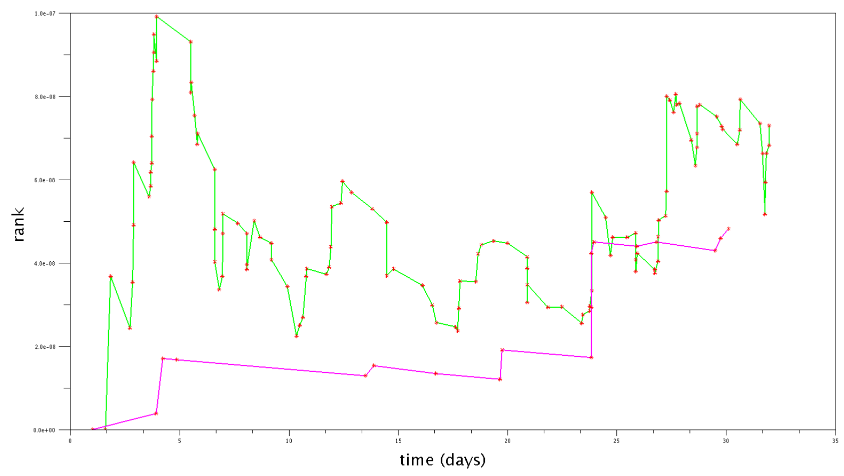

Figure 3.8: Approach 3 rank at different time instances over a month's time. The query "wow" in green and "korn" in magenta. The following constants are set: $const = 2$ hours, $n = 10$.

In the Probabilistic Ranking the rank of a query Q_a is p_{Q_a}, the number of purchases after searching for Q_a over the total time that purchases have been recorded LB. As nothing is said about the popularity trends in the PR and all that is required is that the sum of the purchases during LB is p_{Q_a}, one can assume that the popularity during the whole period LB on average is the same, recalling later that it is not generally so.

Mathematically this can be described as that a purchase occurs after T time units, where T is a random variable with a certain probability density function (pdf) and the expected value $E[T]$. $E[T]$ can be calculated by taking the total time divided by the number of purchases during this time:

$$E[T] = \frac{LB}{p_{Q_a}} \tag{3.12}$$

If the Approach 2 ranking is used there is a c fulfilling $1 \leq c \leq p_{Q_a}$ and a t_c corresponding to the time elapsed since c purchases ago. t_c can be approximated with

$$\hat{t_c} = cE[T] = c\frac{LB}{p_{Q_a}} \tag{3.13}$$

which gives that the rank c/t_c can be estimated by

$$c/\hat{t_c} = \frac{p_{Q_a}}{LB} \tag{3.14}$$

which is the same ranking as the PR, but divided by the constant look-back time. So when the pdf doesn't change over time, the Probabilistic Ranking and the Approach 2 ranking are approximately the same, and thus Approach 2 still ranks based on what is stated in Section 3.4.1 about the Probabilistic Ranking.

In reality the expected value of the pdf of T changes over time, which actually is the whole point with the PM ranking, so what this method does is that it ranks based on the most recent estimate of T. In other words it ranks based on the most recent PR value that can be found for each individual query.

Approach 3, using the mean of $\frac{1}{t_i - t_{i-1}}$ values, is just another way of calculating the mean frequency than Approach 2. Therefore this approach is based on PR ranking too even though it is sensitive to small divisors, which could result in a poor estimate of T.

Adding a constant in the denominator in Approach 2 or 3 smoothens the rank curve, making the ranking a bit different, but it is still based on the PR ranking.

3.5 Merging Similar Suggestions

After trying the Probabilistic Ranking auto complete one can see that sometimes many alternate forms of the same suggestion litter the suggestion list. This can be misspellings, the suggestion with an extra mistakenly entered symbol, titles with and without “the” in the beginning, two words that are written as one or one word written as two. *Merging Similar Suggestions* here means locating all suggestions that look too similar and removing all but the best one, which is determined by choosing the one with the highest rank.

It is also risky to normalize some things like various spellings because if one would initially have more searches on the incorrectly spelled suggestion, this would remove the correct spelling from the list and users would be helped to spell incorrectly, which could prevent the correct spelling from ever appearing in the list, so only very careful normalization will be done.

Similar suggestions are assumed to become less of a problem in a live e-commerce system since the data used here is taken from a system without an auto complete. The statistics collected from a system using an auto complete is assumed to cause fewer misspelled queries to be added to the data as users will be able to choose the correct spelling among multiple ones. This was also discussed in Section 3.4.1, Machine Learning.

Still this effect might not be enough so some degree of normalization could give better results. This is tested in the next chapter.

Chapter 4

Testing

4.1 Procedure

In the testing the search-to-purchase data is used, which in fact is real data of when users have searched for a text string and then bought a product. It is therefore possible to do the testing automatically by using this data.

The test data used is referred to as Test Data Set 1 (TDS 1), and Test Data Set 2 (TDS 2), which are part of the search-to-purchase data. These contain the queries that are tested: TDS 1 consists of the approximately 4000 first search-to-purchase tuples of September 2009, and TDS 2 consists of the following approximately 4000 tuples. The reason for using two test data sets is that these data sets have different qualities, and it is also practically more managable due to the long test execution time.

The testing was done by going through every query in the test data set by asking for an auto completion given the first character as input, then the two first, then the three first, and so on until a good completion is found in the completion list or the end of the query was reached without finding a good completion. What a good completion is will be defined in the Test Criteria below. During the testing of a test data set, all the earlier search-to-purchase data to the point of the beginning of the test data set is loaded in the auto complete.

For the tests on Popularity Measurement (described in Section 3.4.2) it is important that the test updates the search-to-purchase database continuously during the test, but it is less important for the other tests. The updates are done every 15 minutes, in time stamp time (not test execution time), which is often enough to quickly capture new trends and at the same time allow the tests to run in manageable time.

The tests were performed on a virtual server with 8 GB available memory and each test case took between 3 and 12 hours each to perform depending on server load and look-back time.

4.2 Test Criteria

The testing is performed in the manner described above and measure the *Average Required Input Length* (henceforth termed as ARIL), which more specifically is the average number of characters needed to be typed in before a good completion is found. The *Successful Rate* (henceforth termed as SR) is also measured, which is the percentage of times the correct completion was found. The testing seeks to optimize these two values as these values directly correspond to point two and three of the goals in Section 2.1.

Now it has to be defined what a good completion is. In this thesis the following definition is used:

A good completion is defined as a completion that, when searched for, returns the product that the user was going to buy among:

a) the first 10 search results, or
b) the first 5 recommendation results.

This is a guess of what a user might see on a result page after a search and if the product to be bought exists on this page, it was a good completion (the recommendation results are of the type “Customers who searched for ... bought these items.”). This definition is motivated because many queries are misspelled or have multiple synonyms, like “jennifer lopez” and “jlo”, which usually refer to the same artist. It is also motivated by the fact that if the definition simply would have required the auto complete to return a completion that matched the query by exact string match, then for example if a misspelled query would have given a correctly spelled completion, it would not count as a match in the test. This gives misleading test results, and tests of merging of similar suggestions with different spellings give worse test results.

This definition also gives the easily interpreted and useful results: the average number of characters one needs to type in to reach the product one wanted to buy, and the percentage of the times one will find this product from a completion in the auto complete.

4.2.1 ARIL and SR

The Average Required Input Length (ARIL) and the Successful Rate (SR) have a strong relationship to one another. The ARIL depends on the SR: if the SR increases or decreases, so does the ARIL (note that a high SR is good, but a high ARIL is bad). For this reason primarily SR will be maximized, and if there are many cases where the SR is equal, the ARIL is minimized.

A logic explanation for this would be that when the SR increases it means that additional queries are found. Since these additional queries were not found before they must be rarer queries that are competing to be included in the list before the end of the input (which in general means a long input length). When they were not included they were ignored by the ARIL, but when they became included long required input length values are added to the ARIL calculation, and thus the ARIL increases.

A 100 % SR will not be achievable since this would mean that there are no new queries that start in a new way or lead to a new product.

4.2.2 Testing Increased Revenue

One of the goals was to maximize the revenue for the seller, but this is unfortunately impossible to test. This is because in the tests it has to be assumed that every user already has set their mind on which product to buy, as the search-to-purchase data contains information of decisions that users already have made. This kind of testing would require real customers that test every ranking and make purchases. Because the users in the data have already made their decisions it does not matter which other products the auto complete algorithm shows; the “users” will still not change their mind. In Section 3.4.1 about the Probabilistic Ranking, however, it was shown that the auto complete also includes this in the rank. Moreover, the SR and ARIL values with the criteria defined above gives a measure of how often and how fast the users would reach their desired product.

4.3 Errors in the Test Results

Unfortunately the search-to-purchase data is not completely accurate since users very well can search for one thing and then browse for something else and buy an unrelated product to the previously entered search text as was discussed in Section 3.2.1 Correctness of the Data. However, since the auto complete has a limited output (10 completions) it is unlikely to get false positives, which happens if a tested query has a random unrelated product and the auto complete suggests a completion with top search/recommendation results containing this random product. There would then in most cases instead not be any matches for this test query. Therefore the test results are worse than the reality, but should be accurate in comparing different algorithms.

An important thing to note is that the tests depend on the performance of the search function and the recommender. It is assumed that the test cases would perform in a similar way for most search functions are recommenders.

Another important thing to note is that the test results might be inaccurate since an auto complete provided with data from an input that is affected by itself could have unpredictable effects. Moreover, when users are given an auto complete they would often not find the exact same thing in the auto complete list as they would have entered without one.

As mentioned in Section 2.4, it is assumed in the tests that the users always already have set their mind on what to buy. This is another error source, because the auto complete could change the user behaviour.

4.4 The Algorithm Implementation

The existing product matching auto complete, the new algorithms from this thesis, and the combination of the new algorithms and the product matching auto complete will be compared.

The existing product matching auto complete, developed by Apptus Technologies, is quite complex but basically ranks on the most common continuation of the input in the product data, which means that strings that often occur in the product data get a higher rank. This, however, is not the focus of this thesis and this existing mechanism will simply be used like a black box.

The combination of one of the new algorithms and the product matching algorithm is done by first using the new algorithm, and in the case it outputs fewer suggestions than what fills the list, the product matching algorithm is used for the rest of the places in the list. The auto complete is set to show a maximum of 10 suggestions in the list, and the product matching auto complete is set to use product data from the product attributes: title, artists and authors. When the search-to-purchase data is loaded queries are normalized to lowercase.

The new auto complete algorithms were implemented as functions in a system developed by Apptus Technologies, which let the auto complete answer to HTTP requests with XML documents, and gave it access to a database management system that could be loaded with search-to-purchase and product data.

Chapter 5
Results

In the appendixes, tables with the test results are presented. In this chapter the results are discussed referring to these tables. As explained before, the tests were done for Test Data Set 1 and 2, and one can see that the results are different for the two test data sets. More about this in the discussion below (Section 5.7).

5.1 Comparing the Product Matching and the Query Matching Auto Complete

As seen in Table A.1 for Test Data Set 1 and Table B.1 for Test Data Set 2, the query matching auto complete (`ac_new_only`) using PR ranking performs better than the product matching auto complete (`ac_old`). PR ranking alone compared to the product matching auto complete gave an improvement of 4.4 % in successful rate (SR) and 1.1 characters in average required input length (ARIL). These values are averages of both test data sets.

Evidently, there is potential in using search-to-purchase data; it gives a high successful rate and quickly finds the right completions. Combining PR ranking with the product matching auto complete gave an even higher SR, or more specifically an improvement of 2.5 % in SR compared to just using PR ranking. Moreover, as seen in Section 3.4.1, the query matching auto complete also takes conversion rate into account and the product matching auto complete does not.

5.2 Merging Similar Suggestions

In this section the results on merging similar suggestions will be discussed, which was described in Section 3.5. Comparing `ac_merge` and `ac_merge_with_the` with `ac_new_old` in Table A.1 for TDS 1 and Table B.1 for TDS 2, one can see that `ac_merge` and `ac_merge_with_the` had very little difference in SR, so merging by removing some special characters, or removing an initial “the” does not make a big difference, but in general the results are somewhat worse except for the fact that `ac_merge` for TDS 1 got a somewhat higher SR. This means that it’s quite safe merging suggestions that look the same except for some special characters for the suggestions to look cleaner, but it cannot promise a better auto complete and might in fact even make it perform worse.

5.3 Removing Rare Queries

The reason that removing rare queries is considered is that the product matching and query matching algorithms are combined by just giving the query matching auto complete results precedence. If queries with few purchases are removed more space is allowed for the top suggestions of the product matching auto complete.

Comparing `ac_new_old` with `ac_size_gt_1`, `ac_size_gt_2` and `ac_size_gt_3` from Table A.1 for TDS 1 and Table B.1 for TDS 2, one can see that they all have lower SR and therefore no queries should be removed.

An e-commerce site might still choose to do so for the sake of the visual appearance of the completion list (rare queries are often a particular misspelling of another query or typos), or for the sake of increased performance (the number of different queries with just a few purchases is very high compared to the number of common queries).

5.4 Matching

A first thing to note is that the order-specific matching queries are a subset of the order-unspecific matching queries. To maintain high SR one can therefore in the order-specific matching tests first collect the data by matching using order-unspecific matching, and afterwards prioritize the order-specifically matching subset. As seen in the test results, there was a big difference between order-specific matching and order-unspecific matching.

Order-specific matching, meaning that suggestions that start with the entered prefix were prioritized, had much smaller ARIL (an improvement of 0.26 characters on average) and slightly higher SR (an improvement of 0.26 %). Compare the entries `ac_new_old` and `ac_no_prefix` from Table A.1 and Table A.2 for TDS 1, and Table B.1 and Table B.2 for TDS 2.

This result is not that strange. The data consists of a list of queries that all have been typed in from left to right. If one only keeps the ones that start with the given prefix, what remains is the list of queries where users started with writing a certain prefix and continued in different ways. So, taking the most frequent queries in this list will give the most common ways to continue the query after the prefix.

5.5 Using Data from a Shorter Time Period

Comparing the entry `ac_new_old` from Table A.1 or B.1 to the entries in Table A.3 or B.3 respectively, one can see that using the data from a shorter time period results in a shorter ARIL and a lower SR. This shows that a rank based on a shorter and more recent time period gives a faster auto completion, but gives a higher number of failed queries, just as suspected in Section 3.4.2.

Unfortunately it is not possible to test what happens for a longer data period. The SR might increase even more, or decrease, and if it increases it might start to decrease when the data period is long enough. This is a concern that the PM ranking does not have, but as long as time stamps are saved one can update the count of purchases for every query very given time interval.

5.6 Popularity Measurement

With Popularity Measurement the best features are taken from using short and long time periods; this gives both recentness and no data loss, which combined gives even higher SR than the PR ranking with a long time period.

5.6.1 Approach 2

Numerous tests were done to find the best combination of constants. The tables with test results are presented in Section A.5 and Section B.5 in the appendix for Test Data Set 1 and 2. As seen when comparing the results from both test data sets these values are data dependent, so the values will look different for other data sets. These values will therefore serve to give a hint of how to initially find a decent setup of the parameters n, LB and $const$, and will also show if it is desirable to use a look-back time and a constant in the denominator. Since most test cases here have very similar ARIL and SR values it is not crucial that the parameters are absolutely optimal.

First Table A.5 for TDS 1 and Table B.5 for TDS 2 are studied, to find the best n and LB.

For TDS 1 one can see see that the entries with the highest SR values are the entries with $n = 9$. The entry `ac_updating_n_9_LB_0.5d`, with LB set to 0.5 days is the winning entry of this table using the criteria to find the cases with highest SR first and then select the one of these with the lowest ARIL. This entry has an ARIL of 2.839 and an SR of 85.56 %, while the other has an ARIL of 2.843 and an SR of 85.56 %.

For TDS 2 however, the entries with $n = 20$ have the highest SR, and among these `ac_updating_n_20_LB_1d` is the winning entry with an ARIL of 2.837 and an SR of 87.13 %.

Table A.6 and B.6 explore the effect of adding different constants in the denominator for TDS 1 and 2 for some LB and n combinations. The best entry for TDS 1 is `ac_updating_n_9_LB_0.5d_const_4d` which has the same n and LB as the best entry of Table A.5 but has a constant corresponding to 4 days in milliseconds in the denominator. This gave an ARIL of 2.831 and an SR of 85.56 %, which is an improvement over the not using a constant. Although it is a small improvement, it might be a good idea to use a constant since it costs nothing in performance.

The best entry for TDS 2 is found to be `ac_updating_n_20_LB_1d_const_4d`, which has the same n and LB as the winning entry of Table B.5 but has a constant corresponding to 4 days in milliseconds in the denominator.

In Table A.7 and B.7 Quadratic Insecurity Punishment is tested to see if it gives better results. However, the tests performed gave worse results than their corresponding tests using Cubic Insecurity Punishment. One can also see that using no Insecurity Punishment gives worse results.

As seen above better results have been achieved in both ARIL and SR compared to `ac_updating_no_pop` in Table A.4 and B.4. `ac_updating_no_pop` is the updating version of `ac_new_old` in Table A.1 and B.1, which simply is a combination of the product matching auto complete and PR ranking auto complete. The improvement is however perhaps not as big as one would have hoped for: 0.3 % in SR and 0.07 characters in ARIL on average. More on this in the discussion in Section 5.8.

The best configurations, although combined with the product matching auto complete, on average gave an improvement of 7.3 % in SR and 1.1 characters in ARIL compared to only using the product matching auto complete.

5.6.2 Approach 3

In Table A.8 for TDS 1 and Table B.8 for TDS 2 one can see that a higher n value needs to be used to get the best SR and ARIL. The best settings found for TDS 1 were $n = 40$ and $const = 2$ days which gave an ARIL of 2.847 characters and an SR of 85.3 %, and the best settings for TDS 2 were $n = 40$ and $const = 2$ hours which gave an ARIL of 2.91 characters and an SR of 86.92 %. The results are better than expected but still none of the settings that were tried gave an SR value very close to the better Approach 2 test cases, as suspected in Section 3.4.2.

5.7 Relation to Data Density

The tests were done for two test data sets, and there was a quite big difference in the best PM ranking settings. A guess is that this has to do with the data/time density, which in Test Data Set 1 is 3981 purchases/14.5 hours (≈ 275 purchases/h) and in Test Data Set 2 is 3976 purchases/6.25 hours (≈ 636 purchases/h). There might be a simple relationship between this number and for example the best n, but to completely investigate this is not included in this thesis.

However, a quick check indicates that there might be such a relationship: take $\frac{best\ n}{purchases/h}$ for Test Data Set 1 and call this value C.

$$C = 9/275 \tag{5.1}$$

Now note that also for the second test data set $C \cdot purchases/hour$ approximately gives the best n, which was 21.

$$C \cdot 275 = 9 \tag{5.2}$$

$$C \cdot 636 \approx 20.8 \tag{5.3}$$

5.8 Summary and Discussion of Test Results

A big improvement was found in using Probabilistic Ranking (PR) and Popularity Measurement (PM) ranking compared to product matching. There was a smaller improvement going from PR to PM ranking, and only a small difference between most PM ranking configurations. The best results were found using PM ranking Approach 2 (combined with the product matching auto complete) with $n = 9$, $LB = 0.5$ days and a constant in the denominator set to 4 days for Test Data Set 1, and PM ranking Approach 2 with $n = 20$, $LB = 1$ days and a constant in the denominator set to 4 days for Test Data Set 2.

The effect of removing rare queries was explored, and merging similarly looking suggestions and none of these methods gave any improvement.

Some of the improvements are listed below, found by taking the average from both test data sets. These improvements can also be seen in Figure 5.1 and 5.2.

- PR ranking alone compared to the product matching auto complete gave an improvement of 4.4 % in successful rate (SR) and 1.1 characters in average required input length (ARIL).
- Combining PR ranking with the product matching auto complete gave an improvement of 2.5 % in SR compared to just using PR ranking.
- The best PM ranking configurations (combined with the product matching auto complete) gave an improvement of 7.3 % in SR and 1.1 characters in ARIL compared to only using the product matching auto complete.
- The best PM ranking configurations (combined with the product matching auto complete) gave an improvement of 0.3 % in SR and 0.07 characters in ARIL on average compared to basic PR ranking combined with product matching.
- A big advantage (especially in ARIL) was found in strictly prioritizing queries beginning with the user input, which made a difference of 0.26 % in SR and 0.26 characters in ARIL.

Even though PR ranking has worse results than PM ranking, it is much easier to implement and maintain and also requires less computational power than PM ranking since the sum of all purchases connected to a query can be stored and updated periodically. This means that no calculations have to be done to retrieve a rank for a suggestion. PM ranking is however generally very fast as well. The worst case, when there are many purchases within the look-back time, requires about as many operations as the number of purchases connected to the query during LB to retrieve a rank, and as seen earlier in this chapter, LB should in any case be set quite low. One should therefore consider if implementing PM ranking is worth the effort for the advantage it gives.

An important thing to note is that the PR ranking here has data from 8 months, and a longer data period will change the ARIL and the SR. It is likely that the difference between PR and PM ranking will be bigger when the data period is longer. PR ranking will remember more old hits that have been popular but are not anymore, and will always consider them popular. This is something one does not have to worry about with the PM ranking. PM ranking will also always quickly follow trends, while the PR ranking will be slower in noticing that a product has become popular again.

There seems to be a relationship between the best n for PM ranking and the data density, which if it is true might allow one to adapt the algorithm by changing parameters as the purchase intensity changes.

As an interesting note, a test with different criteria was performed using the best configuration of PM ranking for TDS 2. The new criteria was just like the previous one but, in addition, a completion was said to be the correct completion of it was exactly equal to the string that the user was going to enter. This gave a SR value of 96.86 % and an ARIL of 2.68 characters. The interpretation of this is that, in the cases that a customer is going to buy something, in 96.86 % of the cases the customer *on average* only has to type 2.68 characters to get a completion of either the exact string he was going to type or a completion with top search results including the product one was going to buy. In 3.14 % of the cases the customer will not find the product through the auto complete.

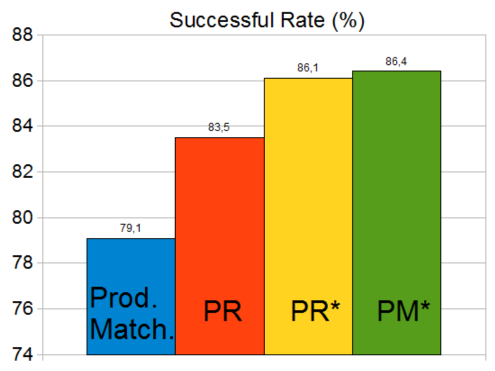

Figure 5.1: Comparison between different rankings in SR. An asterisk (*) means that the ranking is combined with product matching.

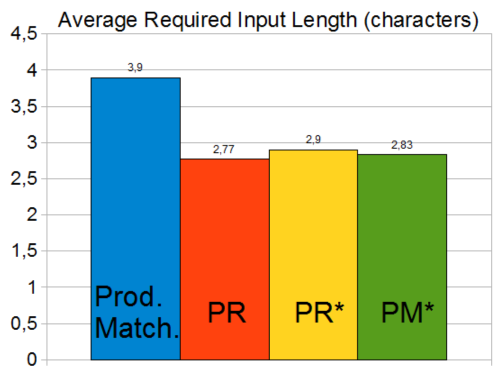

Figure 5.2: Comparison between different rankings in ARIL. An asterisk (*) means that the ranking is combined with product matching.

Chapter 6

Conclusion

It was concluded that the search-to-purchase data contains very valuable information for constructing an auto complete, namely the total number of purchases following every query (see Section 3.4.1). If these queries are used possible suggestions and ranked based on the number of associated purchases, one can show that the suggestions that are most likely to be selected and most commercially effective are prioritized. This ranking is called the Probabilistic Ranking (PR). The negative side with this approach is that it does not take into account how long ago the purchases were made, which is bad since product popularity changes over time.

Therefore a new ranking is created, still based on the number of purchases connected to a query in order to keep the good properties mentioned above, but using as recent data as possible for every query. This is done by calculating the mean purchase frequency for the last up to n purchases. Additional adjustments are also made which include forcing the algorithm to at least look back a specific look-back time and punishing the rank when there are too few purchases (see Section 3.4.2). This ranking is called Popularity Measurement (PM).

The parameters involved in the PM ranking approach are optimized through tests for two test data sets and achieve a higher rate of successful auto completions (denoted successful rate or SR) and a smaller average required input length to find the right completion (ARIL). To get higher rates of successful auto completions a product matching auto complete is used when the PR or PM ranking does not find a complete list of suggestions.

A big improvement was found in using Probabilistic Ranking (PR) and Popularity Measurement (PM) ranking compared to product matching. There was a smaller improvement going from PR to PM ranking, and only a small difference between most PM ranking configurations. The test results are summarized in Section 5.8, which also lists improvements achieved with different techniques given in a measure of SR and ARIL.

Chapter 7
Applications

A very nice property with the solution of the problem in this thesis is that it can be applied for many other functions as well. One can for example use the same algorithms for a "Did you mean?" function, the "Customers who searched for ... bought these items" recommender, and search ranking. How these three functions can be implemented will be described briefly below.

7.1 Customers who Searched for ... Bought These Items

A recommender of this kind takes an input search query and presents what people purchased after searching for that query. There are two differences from the auto complete. The first is that the input is a complete query instead of a query prefix, so the matching in the search-to-purchase data base should be done either by exact match with minor normalization or by order-unspecific matching (Section 3.1), in the latter case perhaps without the requirement that all the words are found. The second difference is that the PR or PM rank for each product connected to the matching query (or queries) has to be calculated rather than the PR or PM rank for each query.

7.2 Did You Mean?

A "Did you mean?" function is basically a function that suggests a correct query if one misspells or makes a typo. The search-to-purchase data contains many various spellings and variations of queries, and the first step one needs to solve is to find a search-to-purchase entry that matches the entered query either exactly or well enough by some measure of string similarity. One can then create groups of similar queries by looking at which queries lead to the same product purchases. Then the query with highest PR or PM rank in this group is suggested.

7.3 Improved Search Ranking

Products in the search result can be ranked just as described about the "Customers who searched for ... bought these items" recommender, and in some way the rank can be combined that with a document ranking from the information retrieval field, like for example tf-idf (see [8]).

Bibliography

[1] A. Gulli and A. Signorini. "The Indexable Web is More than 11.5 Billion Pages," In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 902 - 903, May 10 - 14, 2005. Chiba, Japan.

[2] G. Linden, B. Smith, and J. York. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, Jan./Feb., pages 76 - 80

[3] Nisheeth Ranjan, Andrew Ng. "Using Supervised Learning to Improve Url Ordering in Autocomplete," `http://www.mozilla.org/projects/ml/autocomplete/ac-report.html`, retrieved: November 28, 2010

[4] Bast, H., Weber, I. "Type less, find more: fast autocompletion search with a succinct index," In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 364 - 371. ACM, 2006

[5] H. Bast, C. W. Mortensen, and I. Weber. "Output-sensitive autocompletion search," In *Proceedings of the 13th International Conference on String Processing and Information Retrieval (SPIRE 2006)*, pages 150 - 162, 2006.

[6] B. Sarwar, G. Karypis, J. Konstan and J. Riedl. "Item-Based Collaborative Filtering Recommendation Algorithms," In *Proceedings of the World Wide Web Conference (WWWC10)*, pages 285 - 295, 2001.

[7] Gunnar Blom, Jan Enger, Gunnar Englund, Jan Grandell and Lars Holst. (2005). *Sannolikhetsteori och statistikteori med tillämpningar*, page 27. Lund: Studentlitteratur. ISBN-13: 9789144024424

[8] Ricardo Baeza-Yates, Berthier Ribeiro-Neto. (1999). *Modern Information Retrieval*, pages 29 - 30. New York: ACM Press. ISBN-13: 9780201398298

Appendix A

Appendix: Test Results from Test Data Set 1

This appendix presents the test results for Test Data Set 1. Below is a list of explanations of the notation used in the appendixes.

Training Data Period: the period from which search-to-purchase data is loaded to the auto complete and recommender.
Test Data: the test cases tested.
Matching: if order-specific or order-unspecific matching is used.
Entry Name: a name used to refer to a specific test in the table.
AC Comb.: the combination of auto complete algorithms used. Possible values are: P (referring to the product matching auto complete), Q (referring to the query matching auto complete), and Q & P.
Merging: if merge of similar suggestions in the suggestion list is done (see Section 3.5). Possible values are: None, Basic (normalizing by stripping the symbols separated by a comma: white space, -, :, ', ', ¨), BasicThe (like the previous but occurrences of "the" are also removed).
Size g.t.: keep only suggestions with a number of purchases greater than a given number.
Insecurity Punishment: punish when there are fewer than n purchases. Possible values: Cubic (using the function $f(c) = \frac{1}{n^2}c^3$), Quadratic (using the function $f(c) = \frac{1}{n}c^2$), or None.
Const.: the constant added in the denominator in either Approach 2 or Approach 3.
Size g.t. and Matching are not applicable for the product matching auto complete.

A.1 Non-updating tests

In Table A.1 the following settings are used:
Training Data Period: February - August
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific

Table A.1: Non-updating tests.

Entry Name	**AC Comb.**	**Merging**	**Size g.t.**	**ARIL**	**SR**
ac_old	P	None	-	3.97	78.07 %
ac_new_only	Q	None	0	2.77	82.67 %
ac_new_old	Q & P	None	0	2.90	85.23 %
ac_merge	Q & P	Basic	0	2.91	85.31 %
ac_merge_with_the	Q & P	BasicThe	0	2.92	85.20 %
ac_size_gt_1	Q & P	None	1	2.87	84.35 %
ac_size_gt_2	Q & P	None	2	2.85	83.90 %
ac_size_gt_3	Q & P	None	3	2.84	83.55 %

A.2 Order-unspecific tests

In Table A.2 the following settings are used:
Training Data Period: February - August
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-unspecific

Table A.2: Non-updating, order-unspecific.

Entry Name	**AC Comb.**	**Merging**	**Size g.t.**	**ARIL**	**SR**
ac_no_prefix	Q & P	None	0	3.15	84.98 %

A.3 Shorter time period tests

In Table A.3 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0

Table A.3: Non-updating, shorter time period tests

Entry Name	**Training Data Period**	**ARIL**	**SR**
ac_aug	Aug	2.78	84.23 %
ac_july_aug	July - Aug	2.82	84.75 %

A.4 Updating Tests - Probabilistic Ranking

In Table A.4 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: None

Table A.4: Updating Tests - Probabilistic Ranking

Entry Name	**ARIL**	**SR**
ac_updating_no_pop	2.90	85.28 %

A.5 Updating Tests - Approach 2

Cubic Insecurity Punishment
In Table A.5 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Cubic

Table A.5: Updating Tests - Approach 2

Entry Name	n	LB	**ARIL**	**SR**
ac_updating_n_1_LB_1d	1	1 day	2.94	85.36 %
ac_updating_n_5_LB_2h	5	2 h	2.86	85.51 %
ac_updating_n_5_LB_8h	5	8 h	2.85	85.48 %
ac_updating_n_5_LB_1d	5	1 day	2.83	85.46 %
ac_updating_n_5_LB_2d	5	2 days	2.83	85.48 %
ac_updating_n_5_LB_3d	5	3 days	2.81	85.48 %
ac_updating_n_6_LB_0.5d	6	0.5 days	2.84	85.41 %
ac_updating_n_6_LB_3d	6	3 days	2.81	85.38 %
ac_updating_n_7_LB_0.5d	7	0.5 days	2.84	85.51 %
ac_updating_n_7_LB_3d	7	3 days	2.82	85.46 %
ac_updating_n_8_LB_0.5d	8	0.5 days	2.85	85.51 %
ac_updating_n_9_LB_15min	9	15 min	2.85	85.56 %
ac_updating_n_9_LB_2h	9	2 h	2.85	85.56 %
ac_updating_n_9_LB_6h	9	6 h	2.84	85.56 %
ac_updating_n_9_LB_0.5d	9	0.5 days	2.84	85.56 %
ac_updating_n_9_LB_1d	9	1 day	2.84	85.53 %
ac_updating_n_9_LB_2d	9	2 days	2.83	85.53 %
ac_updating_n_9_LB_3d	9	3 days	2.83	85.53 %
ac_updating_n_9_LB_6d	9	6 days	2.83	85.51 %
ac_updating_n_9_LB_12d	9	12 days	2.83	85.46 %
ac_updating_n_10_LB_8h	10	8 h	2.84	85.48 %
ac_updating_n_10_LB_0.5d	10	0.5 days	2.84	85.48 %
ac_updating_n_10_LB_1d	10	1 day	2.83	85.48 %
ac_updating_n_10_LB_2d	10	2 days	2.83	85.48 %
ac_updating_n_10_LB_3d	10	3 days	2.83	85.48 %
ac_updating_n_10_LB_6d	10	6 days	2.82	85.46 %
ac_updating_n_11_LB_0.5d	11	0.5 days	2.85	85.43 %
ac_updating_n_20	20	1 ms	2.86	85.38 %
ac_updating_n_20_LB_0.5d	20	0.5 days	2.84	85.38 %
ac_updating_n_20_LB_1d	20	1 day	2.84	85.38 %
ac_updating_n_20_LB_2d	20	2 days	2.84	85.38 %
ac_updating_n_40	40	1 ms	2.88	85.13 %
ac_updating_n_40_LB_0.5d	40	0.5 days	2.86	85.13 %
ac_updating_n_40_LB_1d	40	1 day	2.86	85.13 %

Adding a Constant in the Denominator

In Table A.6 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Cubic

Table A.6: Updating Tests - Approach 2 with Constant

Entry Name	***n***	***LB***	**Const.**	**ARIL**	**SR**
ac_updating_n_8_LB_0.5d_const_2d	8	0.5 days	2 days	2.85	85.53 %
ac_updating_n_9_LB_15min_const_1d	9	15 min	1 day	2.84	85.56 %
ac_updating_n_9_LB_15min_const_2d	9	15 min	2 days	2.84	85.56 %
ac_updating_n_9_LB_0.5d_const_2d	9	0.5 days	2 days	2.84	85.56 %
ac_updating_n_9_LB_0.5d_const_4d	9	0.5 days	4 days	2.83	85.56 %
ac_updating_n_9_LB_0.5d_const_6d	9	0.5 days	6 days	2.83	85.53 %
ac_updating_n_9_LB_1d_const_4d	9	1 day	4 days	2.83	85.53 %
ac_updating_n_10_LB_0.5d_const_2d	10	0.5 days	2 days	2.83	85.48 %
ac_updating_n_20_LB_0.5d_const_2d	20	0.5 days	2 days	2.84	85.38 %

Other Insecurity Punishment

In Table A.7 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Quadratic
LB: 0.5 days

Table A.7: Updating Tests - Approach 2: Other Insecurity Punishment

Entry Name	***n***	**Const.**	**IP**	**ARIL**	**SR**
x2_n_9_LB_0.5d_const_4d	9	4 days	Quadratic	2.84	85.56 %
x2_n_20_LB_0.5d_const_4d	20	4 days	Quadratic	2.83	85.41 %
ac_updating_n_9_LB_0.5d_nip	9	None	None	2.87	85.46 %

A.6 Updating Tests - Approach 3

In Table A.8 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Cubic

Table A.8: Updating Tests - Approach 3

Entry Name	n	**Const.**	**ARIL**	**SR**
ac_updating_t2-t1_n_10_const_2h	10	2 hours	3.30	85.05 %
ac_updating_t2-t1_n_10_const_1d	10	1 day	2.98	85.23 %
ac_updating_t2-t1_n_10_const_2d	10	2 days	2.96	85.18 %
ac_updating_t2-t1_n_20_const_2d	20	2 days	2.86	85.25 %
ac_updating_t2-t1_n_20_const_7d	20	7 days	2.86	85.23 %
ac_updating_t2-t1_n_40_const_2h	40	2 hours	2.93	85.20 %
ac_updating_t2-t1_n_40_const_2d	40	2 days	2.85	85.25 %
ac_updating_t2-t1_n_40_const_7d	40	7 days	2.84	85.23 %

Appendix B

Appendix: Test Results from Test Data Set 2

This appendix presents the test results for Test Data Set 2.

B.1 Non-updating tests

In Table B.1 the following settings are used:
Training Data Period: February - August
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific

Table B.1: Non-updating tests.

Entry Name	**AC Comb.**	**Merging**	**Size g.t.**	**ARIL**	**SR**
ac_old	P	None	-	3.80	80.03 %
ac_new_only	Q	None	0	2.77	84.26 %
ac_new_old	Q & P	None	0	2.88	86.75 %
ac_merge	Q & P	Basic	0	2.87	86.67 %
ac_merge_with_the	Q & P	BasicThe	0	2.87	86.64 %
ac_size_gt_1	Q & P	None	1	2.86	86.32 %
ac_size_gt_2	Q & P	None	2	2.85	85.76 %
ac_size_gt_3	Q & P	None	3	2.84	85.26 %

B.2 Order-unspecific tests

In Table B.2 the following settings are used:
Training Data Period: February - August
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-unspecific

Table B.2: Non-updating, order-unspecific.

Entry Name	**AC Comb.**	**Merging**	**Size g.t.**	**ARIL**	**SR**
ac_no_prefix	Q & P	None	0	3.14	86.49 %

B.3 Shorter time period tests

In Table B.3 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0

Table B.3: Non-updating, shorter time period tests

Entry Name	**Training Data Period**	**ARIL**	**SR**
ac_aug	Aug	2.78	85.49 %
ac_july_aug	July - Aug	2.83	86.44 %

B.4 Updating Tests - Probabilistic Ranking

In Table B.4 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: None

Table B.4: Updating Tests - Probabilistic Ranking

Entry Name	**ARIL**	**SR**
ac_updating_no_pop	2.89	86.85 %

B.5 Updating Tests - Approach 2

Cubic Insecurity Punishment
In Table B.5 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Cubic

Table B.5: Updating Tests - Approach 2

Entry Name	***n***	***LB***	**ARIL**	**SR**
ac_updating_n_1_LB_1d	1	1 day	2.93	86.57 %
ac_updating_n_5_LB_2h	5	2 h	2.84	87.03 %
ac_updating_n_5_LB_8h	5	8 h	2.83	87.03 %
ac_updating_n_5_LB_1d	5	1 day	2.82	87.00 %
ac_updating_n_5_LB_2d	5	2 days	2.81	87.00 %
ac_updating_n_5_LB_3d	5	3 days	2.81	87.03 %
ac_updating_n_6_LB_0.5d	6	0.5 days	2.83	87.08 %
ac_updating_n_6_LB_3d	6	3 days	2.81	87.05 %
ac_updating_n_7_LB_0.5d	7	0.5 days	2.83	87.05 %
ac_updating_n_7_LB_3d	7	3 days	2.81	87.05 %
ac_updating_n_8_LB_0.5d	8	0.5 days	2.82	87.03 %
ac_updating_n_9_LB_15min	9	15 min	2.85	87.05 %
ac_updating_n_9_LB_2h	9	2 h	2.83	87.05 %
ac_updating_n_9_LB_6h	9	6 h	2.83	87.00 %
ac_updating_n_9_LB_0.5d	9	0.5 days	2.83	87.03 %
ac_updating_n_9_LB_1d	9	1 day	2.82	87.00 %
ac_updating_n_9_LB_2d	9	2 days	2.82	87.00 %
ac_updating_n_9_LB_3d	9	3 days	2.81	87.05 %
ac_updating_n_9_LB_6d	9	6 days	2.80	87.03 %
ac_updating_n_9_LB_12d	9	12 days	2.80	86.98 %
ac_updating_n_10_LB_8h	10	8 h	2.83	87.05 %
ac_updating_n_10_LB_0.5d	10	0.5 days	2.83	87.08 %
ac_updating_n_10_LB_1d	10	1 day	2.83	87.08 %
ac_updating_n_10_LB_2d	10	2 days	2.82	87.05 %
ac_updating_n_10_LB_3d	10	3 days	2.82	87.05 %
ac_updating_n_10_LB_6d	10	6 days	2.82	87.08 %
ac_updating_n_11_LB_0.5d	11	0.5 days	2.82	86.95 %
ac_updating_n_20	20	1 ms	2.86	87.10 %
ac_updating_n_20_LB_0.5d	20	0.5 days	2.84	87.13 %
ac_updating_n_20_LB_1d	20	1 day	2.84	87.13 %
ac_updating_n_20_LB_2d	20	2 days	2.84	87.10 %
ac_updating_n_21_LB_1d	21	1 days	2.84	87.10 %
ac_updating_n_40	40	1 ms	2.88	87.00 %
ac_updating_n_40_LB_0.5d	40	0.5 days	2.86	87.00 %
ac_updating_n_40_LB_1d	40	1 day	2.85	86.95 %

Adding a Constant in the Denominator
In Table B.6 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Cubic

Table B.6: Updating Tests - Approach 2 with Constant

Entry Name	***n***	***LB***	**Const.**	**ARIL**	**SR**
ac_updating_n_8_LB_0.5d_const_2d	8	0.5 days	2 days	2.81	87.00 %
ac_updating_n_9_LB_15min_const_1d	9	15 min	1 day	2.83	87.03 %
ac_updating_n_9_LB_15min_const_2d	9	15 min	2 days	2.82	87.00 %
ac_updating_n_9_LB_0.5d_const_2d	9	0.5 days	2 days	2.82	87.03 %
ac_updating_n_9_LB_0.5d_const_4d	9	0.5 days	4 days	2.82	87.05 %
ac_updating_n_9_LB_0.5d_const_6d	9	0.5 days	6 days	2.82	87.08 %
ac_updating_n_9_LB_1d_const_4d	9	1 day	4 days	2.82	87.03 %
ac_updating_n_10_LB_0.5d_const_2d	10	0.5 days	2 days	2.83	87.05 %
ac_updating_n_20_LB_0.5d_const_2d	20	0.5 days	2 days	2.84	87.13 %
ac_updating_n_20_LB_1d_const_2d	20	1 days	2 days	2.83	87.10 %
ac_updating_n_20_LB_1d_const_4d	20	1 days	4 days	2.83	87.15 %
ac_updating_n_20_LB_1d_const_6d	20	1 days	6 days	2.82	87.03 %
ac_updating_n_21_LB_1d_const_4d	21	1 days	4 days	2.84	87.15 %

Other Insecurity Punishment
In Table B.7 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Quadratic
LB: 0.5 days

Table B.7: Updating Tests - Approach 2: Other Insecurity Punishment

Entry Name	***n***	**Const.**	**IP**	**ARIL**	**SR**
x2_n_9_LB_0.5d_const_4d	9	4 days	Quadratic	2.82	86.95 %
x2_n_20_LB_0.5d_const_4d	20	4 days	Quadratic	2.81	87.08 %
ac_updating_n_9_LB_0.5d_nip	9	None	None	2.87	86.95 %

B.6 Updating Tests - Approach 3

In Table B.8 the following settings are used:
Test Data: 3981 first entries of September, corresponding to 14.5 hours
Matching: Order-specific
AC Comb.: Q & P
Merging: None
Size g.t.: 0
Insecurity Punishment: Cubic

Table B.8: Updating Tests - Approach 3

Entry Name	***n***	**Const.**	**ARIL**	**SR**
ac_updating_t2-t1_n_10_const_2h	10	2 hours	3.23	86.60 %
ac_updating_t2-t1_n_10_const_1d	10	1 day	2.95	86.90 %
ac_updating_t2-t1_n_10_const_2d	10	2 days	2.92	86.87 %
ac_updating_t2-t1_n_20_const_2d	20	2 days	2.84	86.90 %
ac_updating_t2-t1_n_20_const_7d	20	7 days	2.84	86.85 %
ac_updating_t2-t1_n_40_const_2h	40	2 hours	2.91	86.92 %
ac_updating_t2-t1_n_40_const_2d	40	2 days	2.83	86.80 %
ac_updating_t2-t1_n_40_const_7d	40	7 days	2.82	86.82 %