

MASTER

Towards a reference architecture for context-aware recommender systems

Keijers, B.M.

Award date:
2014

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

EINDHOVEN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

MASTER'S THESIS

Towards a reference architecture for context-aware recommender systems

January 28, 2014

Author: ing. B.M. Keijers
b.m.keijers@student.tue.nl

Supervisor: dr. M. Pechenizkiy
m.pechenizkiy@tue.nl

Tutor: Y. Kiseleva, MSc.
j.kiseleva@tue.nl

Recommender systems have become increasingly important in web systems as a tool to overcome the information overload problem. Research has and will be an important factor in the development of recommender systems, focusing on opportunities to provide more personalized recommendations to the user. Incorporating contextual information from the users in the recommendation process has been one of the recent research directions, resulting in [Context-Aware Recommender Systems \(CARS\)](#). In the future acquisition, representation and evaluation of contextual information will remain an active research direction, as shown in recent literature [[VMO⁺11](#)].

Recommender systems are often specifically built towards a domain, or serving web system. These systems often adhere to the identified requirements and threats towards the recommendation quality. Unforeseen problems in these recommender systems can be hard to solve when the implementation is too specific. Moreover, one of the requirements of [CARS](#) is having a general strategy enabling the availability of context throughout the system. This is required in order to be able to optimal use the available contextual information in all the processing tasks of the [CARS](#). In both cases *flexibility* and *adaptability* in the recommendation processes is required. In this thesis a [Reference Architecture \(RA\)](#) for [CARS](#) is described that provides a general system outline which adheres to these requirements. This is done by identifying the main components and providing separation between core processing and recommendation deriving tasks.

A literature analysis has been conducted to obtain main references and components so that a state of the art [RA](#) could be constructed. In this analysis requirements and components of recommender systems are inventoried. Furthermore active research in the recommender system field is referenced in order to be able to adhere to these requirements as well.

The literature analysis is supported by a system review of the Masters Portal web system from Study Portals. In this way practical information of an actual live web system is used as well. In the system review the opportunities of context application and [CARS](#) are inventoried, generalized and used in the [RA](#).

The evaluation of quality, correctness and effectiveness of the [RA](#) is done by implementing parts of the [RA](#). The evaluation resulted in feedback on the completeness of the objects and components in the [RA](#). Additionally, the implementations are used in practice in an exploratory data analysis with the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm. This algorithm composes a new ranking of a recommendation based on usage behavior. Contextual information is used to provide separation in the usage data and to capture similar preferences of users having corresponding

contextual information.

The results consist of the [RA](#) and the experiment with the [CARA](#) algorithm. The [RA](#) provides a general guideline to implement [CARS](#). However, in order to obtain a more detailed architecture and an improved evaluation, a complete implementation of the system is required. Additionally, a complete implementation enables the opportunity to test various use cases based on system requirements. Furthermore, scheduling and collaboration between various components would be enforced which will yield insight in the complexity of the required scheduling in the recommender system.

The experiment results consist of the implementations and evaluation of the [CARA](#) algorithm. It is shown that the majority of the produced rankings show expected improvement. Nevertheless, an improved data processing strategy is necessary since the evaluation metrics show sensitivities towards anomalies in the data. This sensitivity is also shown when there is too less interaction data available, which causes that the potential improvement is hard to evaluate for these cases. Additionally, experimentation on the live environment of a web system is required to evaluate how the user values the quality of the algorithm.

Acknowledgments

Special thanks to Mykola Pechenizkiy for his guidance during this master thesis work. Many thanks to Julia Kiseleva for her guidance and collaboration in the experiment part. I also would like to thank Study Portals and especially Danny Peeters and Thijs Putman for their assistance in collecting the required data.

Special thanks to Twan Vermeulen and Erik Kouters for support during the construction of this thesis. Furthermore, I would like to thank my fellow students from Fontys Eindhoven who chose to follow this Masters program as well. I enjoyed the collaboration and support during coursework and projects.

Finally, I would like to express my gratitude to my family: My parents, Jan and Mieke, and my brother and sister, Bas and Bregje. Life is harsh in this period of time, and I hope that I can contribute to solutions that bring back peace and tranquility in our lives.

- API** Application Programming Interface. 52
- CARA** Context-Aware Recommendation Adjustment. iii, iv, 2–5, 15–19, 25, 59, 60, 62–64, 67–69, 71–73, 75–79
- CARS** Context-Aware Recommender Systems. iii, iv, 1, 2, 4, 10, 25, 33, 35, 36, 42, 48, 55, 59
- CCTR** Contextual Click-Through Rate. 17, 59, 63, 67–69
- COV** Contextual Coverage. 17, 18, 59, 63, 68
- CTR** Click-through Rate. 16–18, 59, 63, 67, 68
- Kendall correlation** Kendall tau Rank Correlation Coefficient. 18, 59, 67, 69, 71, 72
- ML** Machine Learning. 31
- PAM** Personal Access Model. 79
- POC** Probability of Click. 18, 19
- RA** Reference Architecture. iii, iv, 1–5, 20, 21, 25, 31–33, 35–39, 41, 45–49, 52, 53, 55, 56, 59
- REST** Representational State Transfer. 53
- ROI** Ratio of Improvement. 18, 19, 67, 69, 71, 72

1	Introduction	1
1.1	Motivation	1
1.2	Objectives and methodologies	2
1.2.1	The reference architecture	2
1.2.2	The CARA algorithm experiment	3
1.3	Evaluation and construction challenges	4
1.3.1	The reference architecture	4
1.3.2	The CARA algorithm experiment	4
1.4	Results	5
1.5	Thesis structure	6
2	Preliminaries	7
2.1	General settings	7
2.1.1	Events of interest	7
2.2	Context awareness	8
2.2.1	Contextual categories	10
2.2.2	Contextual features and elements	10
2.3	Recommender Systems and CARS	10
2.3.1	Hybridization	12
2.3.2	User feedback	12
2.3.3	Ranking function	12
2.3.4	The long tail in item popularity	12
2.3.5	Anomaly detection	13
2.3.6	The cold-start problem	13
2.3.7	Concept drift	13
2.3.8	Monitoring	14
2.3.9	User session	15
2.3.10	Computation environments	15
2.4	Scalability and used frameworks	15
2.4.1	Map-Reduce	15
2.4.2	The HADOOP and PIG frameworks	16
2.4.3	MAHOUT	16
2.5	The algorithms	16

2.5.1	The visibility function	17
2.5.2	The Context-Aware Recommendation Adjustment algorithm	17
2.6	Evaluation metrics	19
2.6.1	The Click-Through Rate	20
2.6.2	Contextual coverage	21
2.6.3	The Kendall tau Rank Correlation Coefficient	21
2.6.4	The Probability of Click	21
2.6.5	The Ratio of Improvement	22
I	The reference architecture	23
3	Introduction	25
4	Background	27
4.1	Study Portals	27
4.2	The Masters Portal web system	27
4.2.1	Content objects	28
4.2.2	Log systems	28
4.2.3	Available recommendations	29
4.2.4	Data sources	29
4.2.5	The online environment	30
5	Literature analysis	31
5.1	Approach, scope and focus	31
5.1.1	Main focus	31
5.2	Related architectural work	32
5.2.1	Architectural work on recommender systems	32
5.2.2	Reference architecture work	33
5.3	Related work on application of context	33
5.4	Key architectures	34
5.5	Missing information	36
5.5.1	Architectural and implementation details	36
5.5.2	Design choices	36
5.5.3	Used artifacts	37
5.5.4	Recommender data storage	37
5.5.5	Contextual data sources	37
6	Masters Portal system review	39
6.1	Approach	39
6.2	Focus	40
6.3	Key findings	40
6.3.1	Context awareness	40
6.3.2	Raw data processing strategy	40
6.3.3	Discipline content object	40
6.3.4	Recommender system improvements	41
7	The reference architecture	43
7.1	Reference architecture components	44
7.1.1	The pool	44
7.1.2	Processors	44

7.1.3	Learners	44
7.1.4	The recommender	44
7.1.5	Monitoring	45
7.1.6	Logging and profiling	45
7.1.7	Context awareness	45
7.1.8	Data processing	45
7.2	Comparison with other recommender system architectures	45
8	Implementation	47
8.1	The offline environment	48
8.2	The Context library	48
8.2.1	Data sources	49
8.2.2	Device context resolving	49
8.2.3	Limitations	49
8.3	Learners	49
8.4	Processors	50
8.5	Session data processing	50
8.6	The web service	50
8.7	Implementation feedback	51
8.7.1	Interfaces and typing	51
8.7.2	The control mechanism	51
8.7.3	Component outline	51
9	Conclusions	53
9.1	Results and contributions	53
9.2	Limitations	54
9.2.1	Generalization trade-off	54
9.2.2	Evaluation of the reference architecture	54
9.3	Future work	54
9.3.1	Specialization of the reference architecture	54
9.3.2	Complete implementation of the system	55
II	The CARA algorithm experiment	57
10	Introduction	59
11	Implementation outline	61
11.1	Session data processing	62
11.2	Data processing	62
11.3	Data selection	63
11.4	The CARA algorithm	64
11.5	Visibility function	64
11.6	Ratio of improvement	64
11.7	Recommendation data joining	65
11.8	Changed recommendations	65
11.9	Kendall correlation	66
12	Results	67
12.1	Data volumes and CTR	67
12.2	Recommendation interaction	67

12.3	Improvement evaluation	68
12.3.1	The Kendall tau rank correlation	68
12.3.2	The number of changed recommendations	69
12.3.3	The ratio of improvement	69
13	Conclusions	73
13.1	Results and contributions	73
13.1.1	Contribution to the reference architecture evaluation	74
13.2	Limitations	74
13.2.1	Data volume	74
13.2.2	Data impurity	75
13.2.3	The ratio of improvement	75
13.2.4	Combining of data sources	75
13.2.5	The visibility function	76
13.3	Future work	76
13.3.1	Data volume	76
13.3.2	Evaluation of alternative contextual features	76
13.3.3	Evaluation of hybridizations	77
A	Masters Portal screen shots	79
B	Further experimentation	85
B.1	The algorithms	85
B.2	Experiment 1: Applying re-ranking on collected visit data	86
B.3	Experiment 2: Evaluating the re-ranked recommendations in the online environment	86
	Bibliography	89

1.1 Motivation

Recommender systems have been an active research area since the mid-90's [AT05]. This has led to increased adoption and usage of these systems. Therefore recommender systems will also remain an active research area in the future focused on improving the overall quality [BOHG13]. Recommender systems have and will become more important in web systems in order to solve the information overload problem; nowadays users are overwhelmed with information and hence there is need of a system which is able to provide them only with relevant objects.

One important aspect in improving the performance of recommender systems is incorporating of the contextual situation of users in the recommendation process, which is often referred to as **Context-Aware Recommender Systems (CARS)** [AT11, VMO⁺11]. Incorporating context complicates data- and recommendation processing tasks, because of the additional resolving of contextual information and the introduced complexity in deriving recommendations. Therefore, with the addition of **CARS**, recommender systems are becoming complex systems which are required to be maintainable and adaptable to new requirements or threats to recommendation quality.

Currently recommender systems are specifically built towards a domain, or system. These systems do not have the required *flexibility* and *adaptability* towards changing recommendation requirements, resource management, scheduling of processing tasks and threats against the overall quality of the recommendations.

Moreover, one of the requirements of **CARS** is having a general strategy enabling the availability of contextual information. Besides the recommendation process also user profiling and monitoring can benefit of access to contextual information, allowing for more insight in the preferences of users. Currently, research is focused on defining and proposing general contextual frameworks that are able to provide context to any demanding system [HLGZ12]. In this work the focus is on resolving the context available for web systems, opposed to the general notion of context often outlined in other frameworks.

In order to adhere to the described requirements a **Reference Architecture (RA)** for **CARS** is constructed. The goal of the **RA** is to be used as a guideline when constructing **CARS**. In order to construct the **RA** a literature analysis and a system review are conducted. In the literature analysis main requirements and components, as well as relevant latest research are referenced in order to adhere to latest requirements. Furthermore a system review of the Masters Portal web system is conducted to have practical information from an actual live web system.

Furthermore, evaluation is an important step in the **RA** design process [GA11]. In this step the quality, effectiveness and correctness of the **RA** are assessed. The **RA** has been partially implemented in order to evaluate the architecture. The goal is to evaluate how well parts of the architecture can be instantiated and whether the required *flexibility* and *adaptability* can be ensured. Additionally, the implementations are used in the experiment with the **Context-Aware Recommendation Adjustment (CARA)** algorithm. This algorithm composes a new ranking based on the interaction data of users and the assigned contextual information. This new ranking, or re-ranking, should be able to capture preferences of groups of users characterized by particular contextual information and therefore improve the quality of the recommendations.

1.2 Objectives and methodologies

1.2.1 The reference architecture

A **RA** captures the essence of the architecture of a collection of similar systems in a certain technology-, application- or problem domain [GA11]. In this case the **RA** captures the essence of **CARS** in the recommender system domain.

The **RA** serves the following objective:

To propose a complete reference architecture having minimal interference with the serving system, offers flexibility and adaptability in its core components, and functions as a guideline to implement CARS.

Minimal interference means that there is as less communication channels and shared objects between the web system and the recommender system as possible. Furthermore, it is required to have a separate physical environment for each system in order to not interfere with the operational processes of the web system. This requirement is necessary in order to avoid that the recommender system can disrupt the operational processes of the web system, and therefore compromises the availability of the system. An example of an interfering recommender system is data storage is shared or when various algorithms are performed on the operational environment of the web system.

Completeness means that the essential components of recommender systems are available in order to have a complete and implementable **RA**. This requirement is achieved by inventory and include the components of referenced recommender systems in the **RA**.

The *flexibility* and *adaptability* requirements impose that core functionality can be changed and scheduled in order to be adaptable to changes in the recommender system. These requirements are guaranteed by separating functionality of the various processes in the **RA**.

The **RA** part is organized as a case study containing a literature analysis, system review and partial implementation of the **RA**. The literature analysis is conducted in order to obtain documentation which can serve as input for **RA** design process. The obtained literature contains an overview of components and requirements of recommender systems. The literature analysis is supported by a system review of the Masters Portal web system as described in Chapter 4. In this system review the data sources, process flow and available recommendations are examined. This results in an overview how recommender systems can improve the system. These findings are generalized and applied in the **RA** design process.

Evaluation of the quality, correctness and effectiveness of the reference architecture is supported by partial implementation of the **RA**. The correctness is assessed by evaluating whether the implementation adheres to the **RA**. If this is not the case then there is no support for effective instantiation which

motivates changes in the [RA](#). Evaluation of the actual quality and effectiveness of the partial implementation is done in the experiment with the [CARA](#) algorithm. In this experiment a subset of the partial implementation is used in practice on data sources from the Masters Portal web system. The next section outlines experiment specific objectives as well as the contribution towards the evaluation of the [RA](#).

1.2.2 The CARA algorithm experiment

Context-aware re-ranking can also be applied in other ranked lists. An example is re-ranking a [Search Engine Results Page \(SERP\)](#) based on contextual information. The [SERP](#) can be more personalized towards user groups having the same contextual information and preferences. The starting point of the [CARA](#) algorithm is in fact the re-ranking of [SERP](#).

Furthermore, in recommender systems composing a new recommendation based on one or more existing ones is not new. Examples are tuning an recommendation based on user feedback, or blending multiple recommendation with for example weighting, adjusting the recommendation scores of recommended items, or mixing, combining multiple recommendations to obtain a single recommendation. These strategies cause problems like duplicate elimination that are required to be solved in order to obtain a useful new recommendation. In this experiment these problems are not relevant since the re-ranking is performed on a single recommendation. Instead, it must be determined whether the [CARA](#) algorithm actually can improve recommendations. The reason for this focus is that it is currently not known what the effect of the [CARA](#) algorithm is on recommendations. Hence, the algorithm is first tested in an offline setting on history data prior in order to evaluate expected improvement, prior to test in the live environment of an actual system. This requirement is contained in the following objective:

To show that the [CARA](#) algorithm has the potential to effectively re-rank and improve the quality of existing recommendations.

Effective re-ranking means that the algorithm is able to compose a different ranking of the existing recommendation. The degree of difference between the original and re-ranked recommendations is measured with the [Kendall tau Rank Correlation Coefficient \(Kendall correlation\)](#) as elaborated in Section 2.6.3. Quality improvement means that the re-ranked recommendation is expected to have a higher quality than the existing recommendation. Potential quality improvement is measured by the [Ratio of Improvement \(ROI\)](#) as described in Section 2.6.5.

The experiment is organized as an exploratory data analysis. The exploratory nature of the experiment is in the search to a suitable contextual feature that can provide separation between groups of users with matching preferences. The data sources of the Masters Portal web system as outlined in Section 4 are used to perform the experiment. The required computations are done on the offline environment as described in Section 8.1.

This experiment also contributes to the evaluation step of the [RA](#). This is defined as the following sub-objective:

To evaluate the quality and effectiveness of the partial implementation of components of the reference architecture.

The quality and effectiveness of the partial implementation is evaluated by the practical usage in this experiment. This usage provides feedback regarding the *adaptability* by evaluating how changes can be processed and clarity of the code can be maintained. The *flexibility* is evaluated by obtaining an insight in the resource allocation, scheduling and execution times for the various processing tasks.

1.3 Evaluation and construction challenges

1.3.1 The reference architecture

Generalization/specialization trade-off

When constructing a reference architecture one of the main challenges is that there is a trade-off between generalization and specialization. A RA that is too general will fit every system specification, which will make the architecture not useful for implementation. A RA that is too specialized will be limited on *flexibility* and *adaptability* of its components. Hence the challenge is finding a solution that is in the middle of the described cases. This challenge is addressed in the case study. The construction of the RA is defined as an iterative process that switches between obtaining referencing literature in the literature analysis regarding components or architectural views of the recommender systems and the actual design of the architecture. Additionally, the system review is expected to provide more practical information about the usage of recommender systems which can provide more detailed information beneficial to the specialization of the RA.

Reference architecture evaluation

An important aspect in the design and construction of a reference architecture is how to assess its quality. The implementation of the architecture is an important aspect in the evaluation process, especially when the architecture is built from scratch [GA11].

Given the limited amount of resources to do a full implementation the challenge is to still have a sound evaluation of the RA. This challenge is addressed by involving the architectural properties of the used HADOOP framework in the evaluation as well. The framework is evaluated as a candidate for implementation of the complete system, and therefore possibilities of implementing the remaining components are considered as well.

1.3.2 The CARA algorithm experiment

Flexibility in the implementations

The exploratory nature of the experiment causes that the implementations are subject to changes. The challenge is keeping the clarity in the code and processing scripts when applying modifications. In order to address this challenge the user defined functions of the used PIG framework as described in Section 2.4.2 are used. These functions can be used to develop custom processing functions in Java. In this way functionality can be separated from the scripts, which improves clarity of the code. Furthermore, this separation allows for modification and testing of processing independent parts of the implementation. Additionally, each processing task has a library containing these processing functions. This is done to prevent that two processing tasks need to utilize the same user defined function with a different amount of parameters.

Raw data transformations

The experiment implementations contain a series of intensive data processing tasks where datasets originating from various data sources are combined and transformed into new datasets. The quality and validity of these datasets is hard to evaluate due to the large volumes of the data. In order to

address this challenge intermediate datasets are generated that also give an insight in data which could not be combined, as well as intermediate results in case of transforming data.

Offline evaluation of recommendation quality

Evaluation of the quality of a recommendation in an offline environment, because the interaction with users is missing. Also it is necessary to obtain the most recent data sources else the captured user behavior might be outdated by system changes for example. In order to address this challenge the experiment the **ROI** metric as described in Section 2.6.5 is used to evaluate expected improvement based on the interaction data of users with from the recommendations.

1.4 Results

The results of this thesis consist of reference architecture and experiment specific parts. In the reference architecture part the main result is the constructed **RA** for **CARS**. The minimal interference requirement is addressed by only allowing communication when the serving system is requesting recommendations and in case of required logging and profiling. Furthermore, separation of data sources is obtained by including a data storage component of the **RA** and also minimizes interference between the two systems. In order to address the *flexibility* and *adaptability* requirements a separation between core processing other recommendation deriving is available in the **RA**, enabling scheduling and re-usage of the core functionality of the system.

The evaluation is done by partial implementation of the **RA**. This implementation is done with the HADOOP and PIG frameworks as described in Section 2.4.2. The correctness of the **RA** is shown by the ability to adhere to the architectural structure.

Furthermore, the partial implementations have contributed to other experiments in the research group. The session data processing as described in Section 8.5 has contributed to a data analysis experiment giving insight in the division of contextual information over the user base of the Masters Portal web system. The web service as described in Section 8.6 has been used in an experiment that evaluates the use of contextual information to modify search results the Masters Portal web system.

The experiment with the **CARA** algorithm results in the evaluation of the expected improvement of the re-ranked recommendations of Masters Portal. Three datasets have been composed of the recommendation interaction data: The top-100 dataset containing the 100 recommendations having the most interaction, the random-100 dataset containing 100 randomly selected recommendations and the whole dataset which consist of the complete interaction data. These dataset represent the optimal, average and complete cases when applying the **CARA** algorithm on recommendations. The contextual feature browser is used to re-rank the recommendations based on an evaluation of the distribution of its contextual elements over the data.

The results of the **Kendall correlation** shows that the **CARA** algorithm produces a significant degree of difference in the re-ranked recommendations. This degree of change is the most significant in the top-100 datasets, while the random-100 and whole datasets show a lesser degree of difference. This is as expected by the selection of these datasets, where the top-100 dataset represents the optimal case containing the recommendations having the most interaction, while the random-100 and whole datasets represent the average and overall cases. Hence, the results show that the **CARA** algorithm is able to effectively re-rank the input recommendations.

The results of the **ROI** show that the metric evaluates re-ranked recommendations from browsers having less interaction and a lower data volume as improvements. As a result these browsers show higher

expected improvement scores than browser having better interaction and data volumes. Therefore, it is expected that these browsers show a fair **ROI** score, which indicate potential improvement of the recommendations in the optimal case of the top-100. However, in order to have a better insight in the potential improvement the **ROI** must take into the number of items which are re-ranked by the **CARA** algorithm in order to normalize the scores and obtain a better insight in the potential improvement. Therefore, the expected improvement requirement cannot be completely evaluated.

Finally, the a subset of the partial implementation of the **RA** is used in this experiment in order to evaluate the quality and effectiveness of the **RA**. The PIG framework allows a separation between the processing function and actual executed scripts, which improves the *adaptability* of the implementation since changes towards these functions can be implemented and tested without being required to run the complete functionality on a data source. Furthermore, the HADOOP framework allows for flexible scheduling and resource allocation of processing tasks which also adheres to the required *flexibility* of these functions. Therefore, it is shown that the implementations have the required quality and effectiveness and that the HADOOP and PIG frameworks are suitable candidates for complete implementation.

1.5 Thesis structure

This thesis is composed of a **CARA** algorithm experiment part and a reference architecture part. The reason for this separation is the experiment part also serves objectives which are independent from the objectives of the reference architecture part.

The **RA** part is organized as follows: Chapter 4 contains background information about the company and web system of which the data is used in both the experiment and reference architecture parts. Chapter 5 contains a literature analysis consisting of related work on context application in **CARS** and architectural work on recommender systems. A system review is outlined in Chapter 6. The proposed reference architecture is elaborated in Chapter 7, and Chapter 8 describes the implementations and obtained feedback. Finally Chapter 9 contains the conclusions, limitations and future work on the **RA**.

The **CARA** algorithm experiment part is organized as follows: Chapter 11 contains the implementations used in the experiment. Chapter 12 describes the results of the experiment. Finally, Chapter 13 contains conclusions and limitations, and future work.

In this Chapter the used concepts, definitions and metrics are described. The general settings Section formally defines the main objects used in this thesis. The concepts applicable to both the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm experiment and the [Reference Architecture \(RA\)](#) are elaborated.

2.1 General settings

Let W denote the web system, which is instantiated as the Masters Portal web system. U is defined as the user space of W . Let $O = \{o_1, \dots, o_n\}$ be the collection of content objects defined in W . The collection is defined according to the available content object on Masters Portal as described in [Wis12], $O = \{program, university, country, discipline, scholarship, provider, search, banner, link\}$. $P = \{p_1, \dots, p_n\}$ denotes the pages of W where each page holds at least one object of O . Then p_i can be defined as a pair (o, R) , where $o \in O$ represents the content object and R the list of recommendations. Let R be defined as a collection of triples of a page, a rank and a recommendation score, $R = \{(p, r, s) : p \in P' \wedge r \in \mathbb{N} \wedge s \in \mathbb{R}\}$, with $1 \leq |R| \leq 10$. Let $V = \{(u, p) : u \in U, p \in P\}$ be the collection of visits on W . The definition the page, content object and the list of recommendations adheres to the setup on the Masters Portal page as shown in Figure 2.1.

2.1.1 Events of interest

The *click* and *view* events on a recommendation are considered in the evaluation metrics as shown in Section 2.6 and the [CARA](#) algorithm as described in Section 2.5.2. Formally these events are defined as follows: Let $click \in \{0, 1\}$ denote the click event on the list of recommended programs. The click on a recommendation is defined as follows:

$$click = \begin{cases} 1 & \text{If the user } clicks \text{ on a program in the list of recommendations} \\ 0 & \text{If the user } views \text{ the list and no further interaction takes place} \end{cases}$$

Let $V' = \{(u, p', click) : u \in U, p' \in P', (u, p') \in V\}$ be an extension of the visit collection such that the click event is recorded. Let V_{click} be defined as the subset of V' such that $click = 1$, and V_{view} as the subset of V' such that $click = 0$. In this thesis the former collection will be addressed as *clicks* and the latter as *views*.

Figure 2.1: An example of a master study page of Masters Portal. The considered recommendation list (A) and the content objects (B) are highlighted.

Note that clicks are defined as a visit to a program page that was in the recommendation of the previously visited page. Hence it cannot be guaranteed that this visit is initiated by clicking on the program in the list of recommendations or that the user generated this view otherwise (e.g. by manually navigating to the page). The reason for using this definition is that the actual click on a recommendation is not logged in Masters Portal web system outlined in Chapter 4, and hence the event has to be approximated.

2.2 Context awareness

Context awareness can be defined in various ways depending on the application. A common sense definition can be found in [SAT⁺99] where context awareness is defined as *knowledge about the users and IT devices state, including surroundings, situation, and, to a lesser extent, location*. Also more generalized definitions are used as in [ADB⁺99] where context awareness is defined as *the use of context to provide task-relevant information and/or services to a user with context defined as any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object*. The latter definitions of context and context awareness apply to this thesis.

The used contexts can be defined according to the requirements of the system. In [CK⁺00] and [SAW94], Schilit and Kotz define four categories¹ of context:

- *Computing context* which includes network and device information as contextual situation of a physical machine.
- *Physical context* which considers the external situation in an examined area such as lighting, noise and traffic conditions.

¹Despite the fact that these references are tuned towards mobile computing, they apply to web system and recommender systems settings as well. This is because generalization of context is required in all systems.

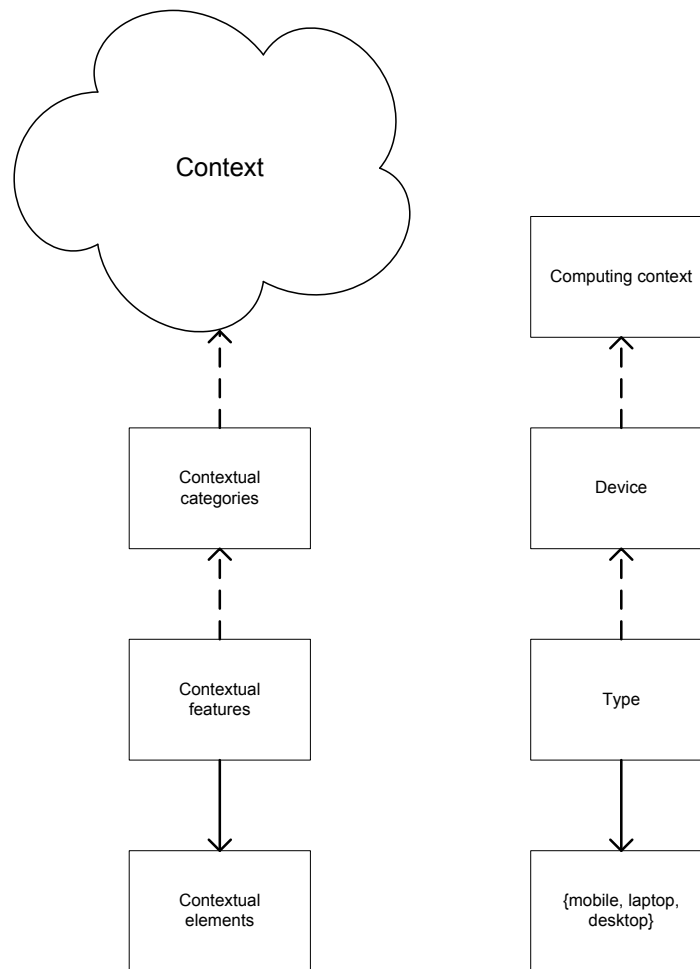


Figure 2.2: The way context is perceived in this thesis. Context is considered to be a cloud of available information. This information is categorized into the contextual categories and features. The contextual features provide the actual context in the form of contextual elements. The dotted arrows denote the inheritance of contextual information. On the right side an example is given of obtaining device type information from the computing context.

- *User context* as the umbrella term for the location, (social) situation and (possibly inferred) user profile state.
- *Time context* which is the usage of various time frames.

In order to enable the usage of context a hierarchy is proposed as shown in Figure 2.2. The contextual information is categorized in the contextual categories in order to provide structure and overview to the available contextual information. The contextual features are able to obtain the required information from the required contextual category in order to resolve contextual elements to the input data. This strategy enables freedom in defining, combining and applying context by enabling custom defining of objects on each level in the hierarchy. An example is given in the right side of the figure: From the computing context device related information is captured in the contextual category device. From the device category the contextual feature type is constructed which determines whether the user utilizes a mobile device, laptop or desktop. As an alternative the mobile and non-mobile can be defined as contextual elements for the type feature. Furthermore, the device category also can offer contextual features like operating system or browser. The computing context can also offer contextual information regarding the network for example. The contextual feature bandwidth for example can

be used to infer the quality of the connection of the user with the system. This information can be used to determine the level of compression necessary in the server response.

2.2.1 Contextual categories

The contextual categories inventory the context into objects such that contextual information belonging to the same entity is available in the same object. In this way the usage of contextual categories improves as well as the freedom to define the contextual category as required. The contextual category is the resource for the contextual features as described in the next section. The following three categories are used in this thesis:

- *Device*: Identification of device and software such as type of device (e.g. mobile or desktop) or the web browser.
- *Location*: Geographical location like country or city.
- *Time*: Temporal characteristics like the timezone of the user or server.

Note that this categorization does not imply completeness. For example, the contextual category *activity* could be used to infer what kind of tasks, appointments or other activities the user has scheduled. This can be done by inferring information from a planning or scheduling system.

2.2.2 Contextual features and elements

Contextual features are used to map the input data to the contextual elements. The contextual information provided by the contextual category is used to establish map the input data to a corresponding contextual element, which can be considered to be the actual contextual value. The separation between mapping contextual data and the contextual elements enables extension and overloading of contextual features. This allows the required *flexibility* in the contextual features. Another advantage of the current setup is that defining of the contextual elements takes place in the features. In this way the contextual elements can be adapted to the requirements of the system.

In the context library as described in Section 8.2 temporal and locational context is inferred by the user characteristics, while device context is inferred by the actual request of the page. In order to maintain simplicity it is assumed that the user space can provide the required information.

Formally contextual features and elements are defined as follows: Let $C_s = \{c_1, c_2, \dots, c_n\}$ denote the contextual space of contextual elements. Let F_x be a contextual feature. Then F_x is a mapping from the user space U to the contextual space:

$$F_x : U \rightarrow C_s$$

An example of a contextual feature is the continents feature. This contextual feature maps the IP address of a user to the continent the user resides. The contextual category location is used to do the mapping and the continents are defined as contextual elements.

2.3 Recommender Systems and CARS

In this thesis the recommender system is defined as *a system that produces individualized recommendations or guides the user in a personalized way to interesting or useful objects in a large space of*

possible options [Bur02]. A recommendation can be considered as a set of objects which are matched to the preferences of the user that are captured by the recommender system.

The word *system* denotes independence between the recommender system and the serving system. This is necessary because of the complexity of the various processing tasks that are required in order to obtain a recommendation. Furthermore, various software artifacts are often present which are not used by the serving system, which also infers independence.

From the user's perspective, the recommender system is a solution for the experienced information overload; while the amount of content on systems is expanding it becomes harder to find the items which meet the requirements of the user. From a business perspective, the recommender system can enhance e-commerce sales in three ways: persuade users to do a purchase by leading them to items similar to previously preferred once to enforce extra sales, suggesting additional items to accomplish a cross-sell and by attempting to have users returning to the system by learning the preferences and contextual parameters [SKR99]. Hence the recommender systems can be a valuable addition to a system.

Recommender systems attempt to incorporate the preferences of the user in the recommendation process in order to get a suitable and personalized recommendation. Initially, only users and content of the system were involved, however the context of the user has become an important aspect in recommender systems in the form of **Context-Aware Recommender Systems (CARS)**. In this type of recommender system contextual information is used to obtain a more personalized recommendation. Several ways of incorporating are proposed by Adomavicius and Alexander in [AT11]. Filtering the input or output of the recommendation process based on contextual information are the main options described. In this way context adds a third dimension to the recommendation process allowing for a more fine-grained personalized recommendation.

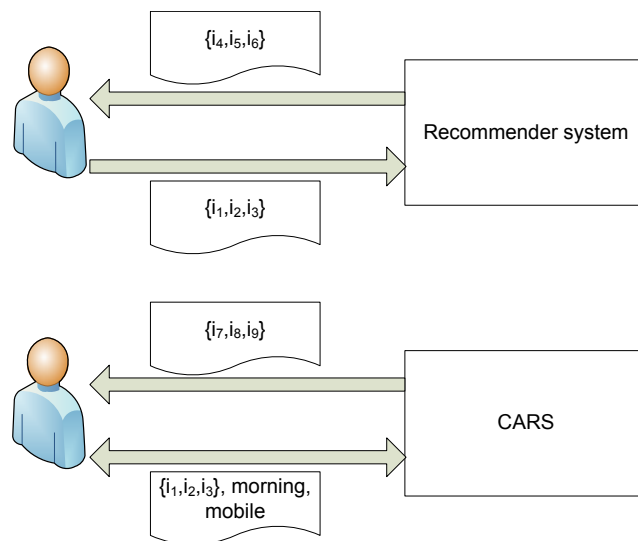


Figure 2.3: An example of a recommender system and **CARS**: The recommender system recommends items based on the session of the user. The **CARS** obtains the contextual *time of day* and *device* having morning and mobile as contextual category. Therefore, the **CARS** is able to derive a more specific recommendation.

Figure 2.3 shows examples of a recommender system and **CARS**. Suppose that the items represent articles on a news website. The recommender system provides a recommendation containing items that are similar towards the content of the items in the session of the user. The **CARS** also obtains the contextual features *time of day* and *device* from the user. Suppose that this user is a regular visitor, then the **CARS** can determine the preferences of the user belonging to the contextual categories.

For example, the user likes to read sports related items in the morning and financial items when using the mobile device. Then based on the contextual information along with the current session the **CARS** is able to recommend items that are similar to the visited items and are related to the sports or financial categories. Hence the recommendation of the **CARS** is more personalized than the recommender system.

2.3.1 Hybridization

Hybrid recommender systems combine collaborative filtering and content-based recommendations in order to improve the quality of the system. Additionally, knowledge-based strategies are often incorporated, where user's requirements are matched to items based on information how the items can satisfy these requirements [Bur02].

Hybridization is defined as the process that combines various recommendation strategies. This combining, also called blending, can be done with weighting, adjusting the recommendation scores of recommended items, or mixing, combining multiple recommendations into a single recommendation for example. Hence hybridization is an important part of the recommender systems since it is the last processing step of the recommendations.

2.3.2 User feedback

Feedback on recommendations is defined as information regarding the usefulness and quality as perceived by the user. Two types of user feedback are considered:

- *Explicit feedback*: When users have the opportunity to apply ratings to recommended items.
- *Implicit feedback*: When the interaction with recommended items is used to obtain information about the quality of the recommendation.

Feedback provides valuable information about the quality of the recommendations. This information can be used in the recommendation process for example to change the ranking of recommended items based on the average rating score.

2.3.3 Ranking function

Let M be defined as the ranking function for R . Then M is a mapping from web system objects to R :

$$M : W \rightarrow R$$

In this document two ranking models are instantiated: Collaborative filtering which utilizes visits data from the users in the ranking process, and the content-based recommendation which uses item similarity in the ranking process. Formally collaborative filtering is defined as a mapping from the visit collection V to R , and the content-based recommendation is defined as a mapping from the object collection O to R .

2.3.4 The long tail in item popularity

The popularity of items in web systems often follows a power law distribution: the popularity of items can vary a power of the number of visitors [And06]. Therefore, a common phenomena observed in the visit data of web systems is that there is a small subset of items are very popular and have high

visit rates, while the remaining set of items form a long tail having lower visit rates. Recommender systems have difficulties to derive recommendations for the items from the long tail because of this limited popularity. This problem occurs when collaborative filtering approaches are used in the recommendation process.

Popularity prediction can be used to identify a power law in the popularity of items. The information can be used as input in the recommendation process as well [MBN⁺13, SH10]. Popularity correction is an approach to reduce the long tail effect [Ste11]. In this correct the difference in popularity is dampened in order to reduce the popularity difference. In the viewpoint of recommender systems and especially collaborative filtering, popularity correction can be applied by normalizing the scores of the popular recommended items to reduce the difference with average popular items. The expected results would be that the the recommendations are not heavily dominated by popular items.

In the Masters Portal web system this long tail is also showing in the visit data from the system. This is shown in the results of the experiment as shown in Section 13.2.1.

2.3.5 Anomaly detection

In this thesis anomaly detection is defined as the problem of finding patterns in visit data that do not conform to expected behavior, as taken verbatim from [CBK09]. With expected behavior it is designated that average visit numbers are expected. Furthermore average user behavior is expected with respect to the rate pages are visited and clicking of links. So anomaly detection also focuses on finding patterns in the visit data which do not adhere to average behavior.

Anomalies can be caused by fraudulent behavior like automatic processes boosting visits numbers on particular pages. But also popularity outbursts can be considered to be anomalies. Hence anomaly detection takes also into account whether the anomaly is caused to harm the system or not. When collaborative approaches are used in the recommendation process, then bots and other automatic processes can be used to massively visit particular pages in the web system in an attempt to influence the ranking of these items. These anomalies can be removed from the recommendation data, however popularity correction can be used in the case of outbursts to dampen the effect and make sure fair recommendations are derived [Ste11].

2.3.6 The cold-start problem

The cold-start problem refers to the fact that recommender systems have difficulties to recommend items to a new user or to find similar items from a newly introduced item [SKR99]. The cold-start problem for newly introduced users in the system is considered to be a specific period of time where certain characteristics, preferences and contextual features of the user cannot be established. For new items it is the time before this item is included in recommender processes.

The scope of the cold-start problem is restricted to the introduction of new user in the system and unknown profiling information to recommender systems, monitoring and profiling. Obtaining and using contextual information can also help to reduce the cold-start problem. Hence the cold-start problem is intrinsically used throughout this thesis.

2.3.7 Concept drift

Concept drift is defined as the problem that user attributes change over the course of interaction with a system, as taken verbatim from [SWSY06]. With respect to recommender systems concept drift

causes problems when recommendation data differs from the actual preferences of the user. There are four types of drift: sudden, gradual, incremental and reoccurring [Koy00], which all could require different strategies to correct or overcome concept drift. Learner adaptivity, where the base learners, parameterization or training set selection can be made adaptive can be a solution of handling concept drift. Also model selection and evaluation can be involved to handle concept drift. In this case the focus lies on assumptions about the future data source rather than the type of concept drift detected [Zli09]. Concept drift is intrinsically used in the design and construction of the RA.

2.3.8 Monitoring

Monitoring of recommender systems consists of collecting and visualizing information of components of the recommender system with respect to determined periods of time. The goal of monitoring is to get insight in the performance and quality of the various components of the recommender system. In this thesis the following monitoring tasks are of interest:

- Resources
- Cold-start
- Concept drift
- Anomaly detection
- User intention

Resource monitoring consists of collecting various measures like running time and allocation of resources like main memory and CPU time. Resource monitoring is important because recommender systems can change over time; data volumes will become larger and more algorithms and processes will be necessary and be dependent on each other, and hence scheduling everything with respect to available resources will be harder. Therefore monitoring can be used to get insight in the performance of various components of the system which can help discovering and repairing emerging problems.

Cold-start monitoring concerns the introduction of a new user to the recommender systems, as described in the previous section. Having a new item in a recommender systems will be considered as a trigger for re-computation of recommendations. In cold-start monitoring it is examined how long users experience cold-start and how and when are the required parameters resolved. Having this monitoring of cold-start is important because it gives insight in the current situation, allows to validate new approaches of overcoming cold-start and can serve as a warning system when parameters are not resolved anymore.

Monitoring concept drift is measuring the difference between current user preferences and logging and profiling data. Furthermore, detection and identification of concept drift is monitored. Concept drift monitoring serves as input in decision process how to handle concept drift.

Anomaly detection monitoring should bring insight in the number of detected anomalies, whether they are caused by fraudulent behavior and whether they occur in particular parts of the system. This monitoring can support decision processes for correcting or removing anomalies, correcting for popularity or otherwise deal with anomalies in data.

Finally user intention monitoring is used to identify what the user's goal is in the system. When the user has as purpose to explore possibilities in the system or wants to find the one matching item it has influence on the way recommender systems should behave. Furthermore, it is important to know the statistics of these users so that recommender systems can be tuned towards the purpose of the visit.

2.3.9 User session

The session of a user is defined as the subset of visit data selected with respect to a temporal interval. The notion of a session is used in the data processing as described in Section 8.5, when the data is divided in session windows in order to allow individual user session processing. Furthermore, the notion of a session is intrinsically when describing various tasks and components of recommender systems in part I.

2.3.10 Computation environments

The notions online, semi-online and offline are used to denote computational environments where operational processes take place. These processes belong to the recommender systems or the web system. The online environment is defined as the operational environment of the web system. The processes necessary for the web system take place in the online environment. The offline environment is defined as a separate computation environment where operational processes cannot interfere in any way with the online environment. The offline environment can be a HADOOP cluster where the recommendation data is processed for example. The term semi-online denotes a process of which the tasks are divided between the online and offline environment. An example of a semi-online task is the recommendation process; the processing in order to obtain recommendations is done on the offline environment while the appropriate recommendations are selected and displayed in the online environment.

2.4 Scalability and used frameworks

This section contains the frameworks used in the various implementations.

2.4.1 Map-Reduce

Map-Reduce is a programming paradigm for distributed and parallel processing of large datasets proposed by Dean and Ghemawat in [DG08]. It is designed to process sets of key-value pairs. Map-Reduce consists of two functions: the Map function and the Reduce function. The Map function is used to transform the input into a pair. Internally pairs with the same key are grouped together in a set. The reduce function applies processing on each set of pairs having the same key. The result is again in key-value format, ranging from a single value to a complete dataset.

Formally the Map-Reduce function is defined as follows:

$$\begin{aligned} \text{map} \quad (k_1, v_1) &\rightarrow \text{list}(k_2, v_2) \\ \text{reduce} \quad (k_2, \text{list}(v_2)) &\rightarrow \text{list}(v_2) \end{aligned}$$

An example of a Map-Reduce program is given in Figure 2.4. The word count program counts the number of occurrences of each word in a document. First the input document is split into separate pieces. Then the map function **M** maps each word to a key and its count to the value. In the shuffle stage **S** the matching keys are grouped. In the reduce function **R** the words are counted and the resulting dataset is produced.

In this example it can be seen that the Map and Reduce functions operate independent from each other. Also in the shuffle stage each Map function just sends the output to the correct reducer. Therefore Map-Reduce allows to execute each Map or Reduce function on separate processor cores or

even physical machines. This allows for distributed and parallel processing of the data², which can result in shorter execution times with respect to single core or single machine processing.

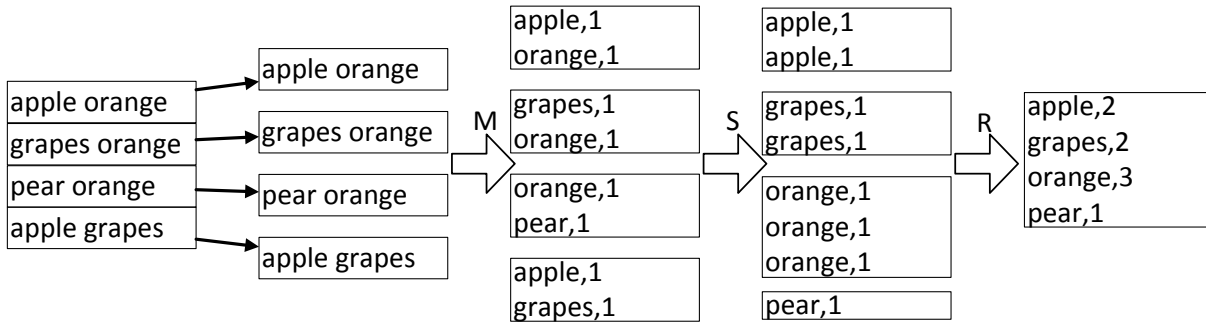


Figure 2.4: An word count example of a Map-Reduce program. The input dataset is first split into separate pieces. Then the Map function **M** maps the word to a key with value 1. In the shuffle stage **S** the matching keys are grouped. Finally the Reduce function **R** counts the number of words and outputs their occurrences.

2.4.2 The HADOOP and PIG frameworks

HADOOP is an open-source framework for scalable distributed computing. HADOOP allows for the distributed processing of large data sets across clusters of computers using the Map-Reduce programming paradigm. The framework is scalable in the number of used processing cores and machines. Furthermore HADOOP can detect and handle failures at the application layers, and hence there is less need to have high-availability, fault tolerant and therefore expensive fault hardware.

In order to use HADOOP tasks must be constructed which can do the preferred processing by utilizing the Map-Reduce paradigm. The PIG framework has been chosen to implement the required functionality in. PIG is a high-level framework that allows the user to construct Map-Reduce implementations by utilizing PIG latin as scripting language. PIG latin has similarities in implementation with commonly known relational languages like SQL. Therefore it requires less effort to get familiar with the framework. Furthermore the user is not required to construct the actual Map and Reduce functions; a library of built-in functionality is available along with the opportunity to construct custom functionality when required. Therefore the focus lies on implementing the processing function, rather than implementing the actual map and reduce functions.

2.4.3 MAHOUT

MAHOUT is a machine learning library that offers various algorithms focused on machine learning, statistics and mathematics. A part of these algorithms has been implemented in HADOOP as described in Section 2.4.2 allowing for distributed processing of large amounts of data. The recommendation algorithms of MAHOUT are used in the implementations as elaborated in Section 8.3.

2.5 The algorithms

This section contains the algorithms used in the experiment with the CARA algorithm as described in Part II.

²In the remainder of this thesis distributed computing is mentioned as the umbrella term denoting distributed and parallel processing.

UserID	\mathbf{F}_x	Clicked program
u_1	1	p_1
u_2	1	p_2
u_3	0	p_3
u_4	1	—
u_5	0	p_3
u_6	0	p_2
u_7	1	p_1
u_8	0	p_2
u_9	0	—
u_{10}	1	p_1

Table 2.1: A collection of visit data for 10 users (**UserID**) with a binary contextual feature (\mathbf{F}_x) on the pages p_1, p_2 and p_3 . The clicked recommended program (**Clicked program**) is recorded where p_x denotes the *click* event on master study program p_x and - the *view* event of the recommendation.

2.5.1 The visibility function

When a recommendation is shown a page, the user is more eager to interact with the top recommendations than the bottom ones. The reason for this behavior is that the top recommended items are the most similar to the current content object. Moreover, recommendations are often not completely visible and the user might be required to scroll down the page to examine the complete recommendation. Because of this behavior and the page setup there can be a difference in interaction between the top ranked recommended items and lower ranked ones. In the case of limited exposure of lower ranked recommendation items to the user this interaction difference can be unfair especially when the differences in similarity between the recommended items is very small.

To correct this interaction difference, the visibility function can be applied. This function corrects the score of recommended items based on their rank which results in a new ranking list. In this way the interaction difference based on visibility only is corrected.

In the case of Masters Portal the visibility function is not known. A simple function is proposed to approximate the visibility of the recommendations on the Masters Portal web system.

The visibility function of the Masters Portal recommendations is defined as follows: Let S denote the visibility of $p_i \in R$. Then S is a mapping from $r_i \in R$ to a scoring interval:

$$S_{r_i} = 1 - \frac{(i-1)}{10}, 1 \leq i \leq 10.$$

2.5.2 The Context-Aware Recommendation Adjustment algorithm

The user's interaction with a recommendation is of importance because it provides information about the perceived quality. It is expected that the majority of interaction will take place with the top ranked items of the recommendation. Interaction patterns showing other usage behavior can indicate that the recommended items valuable to the users are not ranked at the top.

Moreover, the recommendation can be too general to serve the general population of a web system. In the view of the Masters Portal web system an example can be that users located in countries having an active oil industry are more interested Oil Engineering and similar master studies, while users located in countries that do not have an oil industry could be more interested in Sustainable Energy. This example provides two types of master studies on the same Energy Engineering discipline. Hence

a recommendation which takes into account the contextual feature geographical location can be tuned towards more specific user groups than a general recommendation like a content-based one.

The **CARA** algorithm can overcome the two described problems. The algorithm ranks the items of a recommendation based on the interaction of the user with the recommendation. This is called re-ranking. This re-ranking is done based on the contextual feature of the processed usage information. In this way for each contextual element a re-ranked recommendation is constructed that is tuned towards the usage behavior of the belonging users.

The **CARA** algorithm generates a new score for each recommended item and the contextual elements belonging to the selected contextual feature. The usage data consists of the *click* and *view* events as described in Section 2.1.1 of the recommendation under processing. The new score is the the fraction of recorded *click* events on the recommended item divided by the total number of *view* and *click* events for events generated with a particular contextual element. Then the recommended items are ranked based on the re-ranking scores. The result is a re-ranked recommendation.

Consider a practical example: Assume that Table 2.1 is the collected usage data for pages p_1, p_2, p_3 data from 10 users, $u_1 \dots u_{10} \in U$. Suppose current ranking model M returns $R = \{(p_1, 1, s_1), (p_2, 2, s_2), (p_3, 3, s_3)\}$ with $s_i \in \mathbb{R}$ and $s_i > 0$. Let F_x be a contextual feature with $C_e = \{0, 1\}$ being the collection of contextual elements. Let the visibility of the recommended items be defined by the visibility function; $S_{p_1} = 0.9, S_{p_2} = 0.8, S_{p_3} = 0.7$. Then the probability that a $u \in U$ with $F_u = 1$ has clicked on a program in R can be calculated as follows:

$$\begin{aligned} Pr(F_x = 1, p_1, R) &= \frac{3}{5} = 0.6 \\ Pr(F_x = 1, p_2, R) &= \frac{1}{5} = 0.2 \\ Pr(F_x = 1, p_3, R) &= \frac{0}{5} = 0 \end{aligned}$$

The visibility function is applied to correct for the difference in interaction between top and bottom ranked items. Let $v = \{0, 1\}$ denote that the page has been visited. Then by definition of the *click* event $v = 1$. So then the visibility score is as follows:

$$\begin{aligned} Pr(F_x = 1, p_1, R|v = 1) &= \frac{Pr(F_x = 1, p_1, R)}{S_{p_1}} = \frac{0.6}{1.0} = 0.6 \\ Pr(F_x = 1, p_2, R|v = 1) &= \frac{Pr(F_x = 1, p_2, R)}{S_{p_2}} = \frac{0.2}{0.9} = 0.\bar{2} \\ Pr(F_x = 1, p_3, R|v = 1) &= \frac{Pr(F_x = 1, p_3, R)}{S_{p_3}} = \frac{0}{0.8} = 0 \end{aligned}$$

Now calculate the click probability and visibility correction for $F_x = 0$;

The probability that a user with $F_x = 0$ has clicked on a program in R can be calculated as follows:

$$\begin{aligned} Pr(F_x = 0, p_2, R) &= \frac{2}{5} = 0.4 \\ Pr(F_x = 0, p_3, R) &= \frac{2}{5} = 0.4 \\ Pr(F_x = 0, p_1, R) &= \frac{0}{5} = 0 \end{aligned}$$

When two items have the same probability the ordering of the items defines the rank. By inferring the visibility function, the following probabilities are obtained:

$$\begin{aligned} Pr(F_x = 0, p_3, R | s = 1) &= \frac{Pr(F_x = 1, p_3, R)}{S_{p_3}} = \frac{0.4}{0.9} = 0.\bar{4} \\ Pr(F_x = 0, p_2, R | s = 1) &= \frac{Pr(F_x = 1, p_2, R)}{S_{p_2}} = \frac{0.4}{1.0} = 0.4 \\ Pr(F_x = 0, p_1, R) &= \frac{Pr(F_x = 1, p_1, R)}{S_{p_1}} = \frac{0}{0.8} = 0 \end{aligned}$$

The [CARA](#) algorithm provides the following re-ranking of the recommendation:

$$\begin{aligned} R_1 &= \{(p_1, 1, 0.6), (p_2, 2, 0.\bar{2}), (p_3, 3, 0)\} \text{ for } \{u \in U : F_x = 1\} \\ R_2 &= \{(p_3, 1, 0.\bar{4}), (p_2, 2, 0.4), (p_1, 3, 0)\} \text{ for } \{u \in U : F_x = 0\} \end{aligned}$$

For users with $F_x = 1$ the ranking does not change. However, for users having $F_x = 1$, p_3 is ranked first, p_2 second and p_1 third. So the [CARA](#) algorithm can be considered as a special case of ranking model, which we define as the re-ranking model. Let M^* be defined as the re-ranking function. Let $R^* = \{R_1, \dots, R_n\}$ denote the collection of re-ranked recommendations. Then M^* is defined as a mapping from V and R to R^* :

$$M^* : (V, R) \rightarrow R^*$$

Recommendation data usage

Table 2.2 shows two recommendations with different lengths. The left recommendation contains 10 items which are all visible. The right recommendation contains at least 15 items of which ten are visible. For the left recommendation the users can interact with all recommended items. This does not hold for the right recommendation as a part is not visible. However, when visible items obtain low re-ranking scores because the interaction is low, items previously ranked in the invisible part can enter the visible part of the recommendation. Hence the application of the [CARA](#) algorithm enables to parsing options for the recommendations.

In the experiment as described in part II a set of recommendations is used which consists of 30 recommended items. The first 10 of the recommended items are visible to the user. It is chosen to only consider the visible part of the recommendation because the actual data is available for these items. When items appear in the visible part of the recommendation of which no data is available then it cannot be decided whether the recommendation is improved because the previous interaction from users with this item is missing. The recommendation the [CARA](#) algorithm is applied is described in Section 4.2.3.

2.6 Evaluation metrics

This section contains the used evaluation metrics of the experiment with the [CARA](#) algorithm.

		Rank	Item
		1	R_1
		2	R_2
		3	R_3
Rank	Item	4	R_4
1	R_1	5	R_5
2	R_2	6	R_6
3	R_3	7	R_7
4	R_4	8	R_8
5	R_5	9	R_9
6	R_6	<u>10</u>	R_{10}
7	R_7	<u>11</u>	R_{11}
8	R_8	<u>12</u>	R_{12}
9	R_9	<u>13</u>	R_{13}
10	R_{10}	<u>14</u>	R_{14}
		<u>15</u>	R_{15}
		.	.
		.	.
		.	.

Table 2.2: A representation of the two options for the recommendations in conjunction with the CARA algorithm. The left recommendation is completely visible for the user. From the right recommendation only the first ten items are visible, while the lower ranked and underlined items are not visible for the user.

2.6.1 The Click-Through Rate

The Click-through Rate (CTR) is used as a way to measure the effectiveness of an online advertising campaign for a particular web system. It gives the percentage of users which clicked on the advertisement. In this thesis the CTR is used to justify how representable the recommendation data is by showing the percentage of the user base which utilized the recommendations. The CTR is used in the results of the experiment as outlined in Chapter 12.

The CTR is defined using the events of interest as described in Section 2.1.1. The CTR is defined as follows:

Definition 1 $CTR(R, V_{click}, V_{view}) = \frac{|V_{click}|}{|V_{view}|} \times 100$

Contextual Click-through rate

The Contextual Click-Through Rate (CCTR) provides the CTR of a recommendation for each contextual element of a preferred contextual feature. It is used to obtain insight the proportion of the CTR per contextual element. The CCTR is applied in the experiment as outlined in Section 12.2.

Suppose F_1 is the contextual feature of interest, and let $c_1 \in C_s$ be the contextual category of interest. Let $V'_{click} = \{v \in V_{click} : F_1(u) = c_1\}$ be the subset of V_{click} containing visits of users which have the contextual category c_1 . $V'_{view} = \{v \in V_{view} : F_1(u) = c_1\}$ is defined as the subset of V_{view} containing visits of users which have the contextual category c_1 . The CCTR is defined as follows:

Definition 2 $CCTR(R, V'_{click}, V'_{view}) = \frac{|V'_{click}|}{|V'_{view}|} \times 100$

2.6.2 Contextual coverage

The **Contextual Coverage (COV)** is used to see the distribution of the contextual elements of a contextual feature among the **CTR**; It shows how much coverage a particular contextual category has in the **CTR**. The **COV** is used to get insight in how well contextual elements of a feature are distributed. The application of **COV** is described in Section 12.2.

Let F_1 be the contextual feature of interest, and let $c_1 \in C_s$ be the contextual category of interest. Let $V'_{click} = \{v \in V_{click} : F_1(u) = c_1\}$ be the subset of V_{click} containing visits of users which have the contextual category c_1 . The **COV** is defined as follows:

Definition 3 $COV(R, c_1, V'_{click}, V'_{view}) = \frac{|V'_{click}|}{|V'_{view}|} \times 100$

2.6.3 The Kendall tau Rank Correlation Coefficient

The **Kendall tau Rank Correlation Coefficient (Kendall correlation)** is used to measure the similarity between two different rankings [Ken38]. In this thesis this metric is used to measure the degree of difference between the input recommendation and re-ranked recommendation produces by the **CARA**. This is described in Section 12.3.

Let R be the input recommendation and let R^* be the recommendation re-ranked by the **CARA** algorithm. Then $|R| = |R^*|$ and both recommendations contain the same elements.

The **Kendall correlation** compares each pair of rankings, the input recommendation rank and the re-ranked recommendation rank, against each other. These pairs are defined as (x_i, y_i) and (x_j, y_j) , $x_i, x_j \in R$ and $x_j, y_j \in R^*$ and $i \neq j$. The set of concordant pairs P_{con} is defined as $P_{con} = \{((x_i, y_i), (x_j, y_j)) | (x_i > x_j \wedge y_i > y_j) \vee (x_i < x_j \wedge y_i < y_j)\}$. The set of dis-concordant pairs P_{dis} is defined as $P_{dis} = \{((x_i, y_i), (x_j, y_j)) | (x_i > x_j \wedge y_i < y_j) \vee (x_i < x_j \wedge y_i > y_j)\}$. Then the **Kendall correlation** is defined as follows:

Definition 4 $\tau = \frac{|P_{con}| - |P_{dis}|}{\frac{1}{2}|R|(|R|-1)}$

The range of the coefficient is $-1 \leq \tau \leq 1$. If R^* is exactly the same as R then $\tau = 1$. If R^* is the reversed order of R then $\tau = -1$. R^* and R are said to be independent when $\tau \approx 0$. When evaluating the degree of difference between the input and re-ranked recommendations the intervals $[-1, 0)$, $(0, 1]$ and $[0.5, 1]$ are examined. When the **Kendall correlation** score is in the interval $(0, 1]$ it is considered to show a degree of difference between input and re-ranked recommendation, while the interval $[0.5, 1]$ is considered to slight to no difference. When the **Kendall correlation** score is in the interval $[-1, 0)$ it is considered to show the most difference as the re-ranked recommendation then resembles the inverse of the input recommendation.

2.6.4 The Probability of Click

The **Probability of Click (POC)** is the expected probability that the user will click an item of a recommendation. The **POC** is used in the **Ratio of Improvement (ROI)** as described in the next section. The **POC** is used as a baseline to determine whether the **CARA** shows expected improvement.

UserID	R	Clicked program
u_1	1	p_1
u_2	1	p_2
u_3	1	p_3
u_4	1	—
u_5	1	p_3
u_6	1	p_2
u_7	1	p_1
u_8	1	p_2
u_9	1	—
u_{10}	1	p_1

Table 2.3: A collection of visit data for 10 users (**UserID**) on a recommendation (**R**) on pages p_1, p_2 and p_3 . The clicked recommended program (**Clicked program**) is recorded where p_x denotes the *click* events on master program page p_x and - denotes the *view* event of the recommendation.

Consider an example: Table 2.3 shows collected visit data of ten individual users. The probability of click is defined as the number of clicks per page divided by the total number of log entries:

$$\begin{aligned} Pr(p_1, R = 1) &= \frac{3}{10} = 0.3 \\ Pr(p_2, R = 1) &= \frac{3}{10} = 0.3 \\ Pr(p_3, R = 1) &= \frac{2}{10} = 0.2 \end{aligned}$$

In this example p_1 and p_2 have an equal chance to be clicked by the user, while p_3 has less chance to be clicked.

2.6.5 The Ratio of Improvement

The **ROI** is used to determine whether the re-ranked recommendation produced by the **CARA** algorithm show potential improvement. This potential improvement is measured by comparing the score of each item in the re-ranked recommendation with the **POC** as described in the previous section. The re-ranking scores are essentially the probability a user characterized by particular contextual information will click on a recommendation. The **POC** represents the probability that the user will click on an item of a recommendation. When the re-ranked items have a higher score than the **POC** of the recommendation then the re-ranking shows potential improvement. Formally the **ROI** is defined as follows:

Definition 5 $ROI(R, p_x, c_x) = \frac{pr(p_x, R)}{Pr(C_s, p_1, R)}$

The resulting score is interpreted as follows:

$$ROI = \begin{cases} < 1 & \text{The CARA algorithm shows expected regression} \\ 1 & \text{The CARA algorithm shows nor regression or improvement} \\ > 1 & \text{The CARA algorithm shows expected improvement} \end{cases}$$

The **ROI** is used in the result of the experiment as outlined in Section 12.3.

Part I

The reference architecture

This first part of the thesis contains the case study that is performed in order to construct a [Reference Architecture \(RA\)](#) for [Context-Aware Recommender Systems \(CARS\)](#). The goal is to provide a [RA](#) that offers independence from the online environment as outlined in [Section 2.3.10](#), completeness by incorporating main components identified in recommender systems and can function as a guideline when constructing [CARS](#). In order to inventory and fulfill these requirements, a literature analysis has been conducted. The analysis focuses on obtaining architectural views, requirements and implementation details of related recommender systems. Moreover, the literature analysis concentrates on identifying and incorporating context in a flexible way in [CARS](#).

Furthermore, a system review of the Masters Portal web system as described in [Chapter 4](#) has been conducted. The goal of this review is to obtain an insight in the way [CARS](#) can improve the quality of the system, and to generalize these findings in order to use them as input for construction of the [RA](#).

Additionally various parts of the [RA](#) are implemented in order to provide a partial validation of the architecture and to serve the experiment with the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm as outlined in [Chapter 11](#). The obtained feedback of the implementations is used to outline extensive work on the [RA](#).

The main result of the case study is the proposed [RA](#) as elaborated in [Chapter 7](#). Secondly, various implementations and collected feedback as described in [Chapter 8](#) are also part of the results. Currently the main limitation of the [RA](#) is the absence specialization of various components. Further limitations are described in [Section 9.2](#).

This Chapter contains additional information about the Masters Portal web system of which the data sources are used in both reference architecture and experiment parts. Study Portals is the company behind Masters Portal. In the remainder of this chapter company information as well as available data sources, recommendations and environments of Masters Portal are described.

4.1 Study Portals

Study Portals is a company that focuses on international study choice by providing web systems which offer information about various European study options. The company started in 2007 as Masters Portal, named after the first web system, and was renamed in 2009 to Study Portals. Currently, Study Portals holds various web systems containing study information about master studies, short courses, PhDs, bachelors and scholarships. STeXX, Student Experience Exchange, is the latest web system of Study Portals. In this system users can write reviews of their study experiences in European countries.

4.2 The Masters Portal web system

Masters Portal is the main web system of Study Portals. The goal of the system is to connect users interested in masters education in Europe to universities. Currently the system holds 21,000+ master studies, along with information about the country and university they are located. An extensive search engine allows the users of the system to find the most suitable master study with respect to geographical, financial, temporal and various other requirements.

The Masters Portal web system is the system of interest this thesis. The system is subject of a review in Chapter 6. The data sources available data sources are used in both implementations contributing to both reference architecture and experiment parts.

4.2.1 Content objects

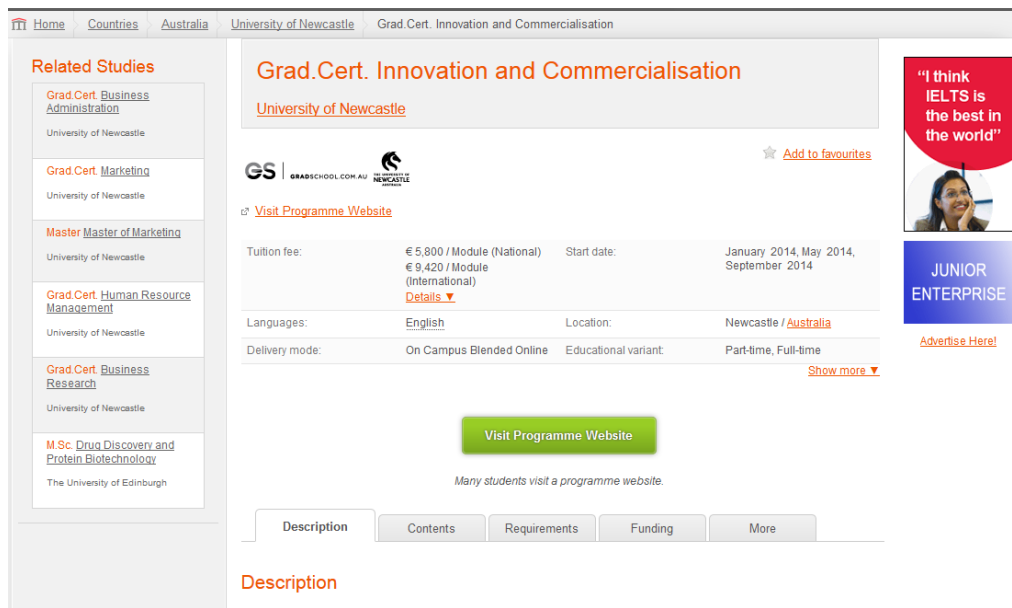


Figure 4.1: An example master study page available on the Masters Portal web system.

Currently Masters Portal holds the following content objects:

- *Study*: description about a particular master study.
- *Discipline*: the discipline a master study belongs to.
- *University*: practical information of an university.
- *Country*: information about studying and living in a particular country.

In addition to these content objects the *search* object is defined, which holds information about search queries from the user. In the session data processing as described in Section 8.5 this object instantiated.

These content objects are instantiated as web pages in the system. Figure 4.1 shows the master study page. Typically each page on Masters Portal contains the same separation between recommendation (if available) and content object. Appendix A contains screen shots of the remaining pages of Masters Portal.

4.2.2 Log systems

Log systems are able to capture usage information and system events. This information is stored for usage in various processes like validation of the performance of the system. Currently Masters Portal holds two main log systems which are of importance in this thesis; the Apache log system and the statistics log system.

Apache access log system

The Apache access log system logs the traffic processed by the web system. The system produces access log files which are highly configurable and therefore are able to log a variety of logging information.

A common log line consists of IP-address, the time and type of the request, the status code, the user agent, requested path and returned object size without response headers [Fou13]. The access log data of the Masters Portal web system is transformed to the session data processing outlined in Section 8.5.

Statistics log system

The statistics log system captures user and system events of the Masters Portal web system. Besides visits of pages, user events also contain clicks on promoted items and search actions. System events consist displaying recommended and promoted items on various pages visited by the user. An item is defined as any possible object on a portal of Study Portals which are master study-, university-, country- and discipline items [Wis12].

From the resulting log data the relevant recommendation interaction data is extracted and used as described in Section 4.2.4.

4.2.3 Available recommendations

Masters Portal holds a recommender system which derives recommendations for the master study, university and discipline pages. Furthermore the recommendation part of pages is often used to show factual lists like *Masters in United Kingdom* for example. The following recommendations are presented on Masters Portal:

- *Related studies*: A content based recommendation of master studies visible on the majority of master study pages.
- *Featured studies*: Although unclear whether promoted or recommended, this list holds an enumeration of master studies related to the visited university page.
- *Suggested studies*: A recommendation of master studies on the discipline page.

Each recommendation consists of 30 items. About 10 recommended items are shown on each page. The *related studies* recommendation is used in the experiment outlined in part II.

4.2.4 Data sources

Recommendation interaction data

A subset of logging data is used which are the impressions of recommendations on master study pages and the views of master study pages. From this data the click and view events on the recommendation as described in Section 2.1.1 are generated and selected, hence this data represents the interaction of the user with the recommendation. This data is used in the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm experiment as described in Part II.

Session data

The Apache access logs are filtered, cleaned, semi-structured and extended with contextual features in order to transform them into session data. This data processing is described in Section 8.5. The session data is used in practice in the [CARA](#) algorithm experiment to provide contextual features to the recommendation interaction data. This is outlined in Section 11.2.

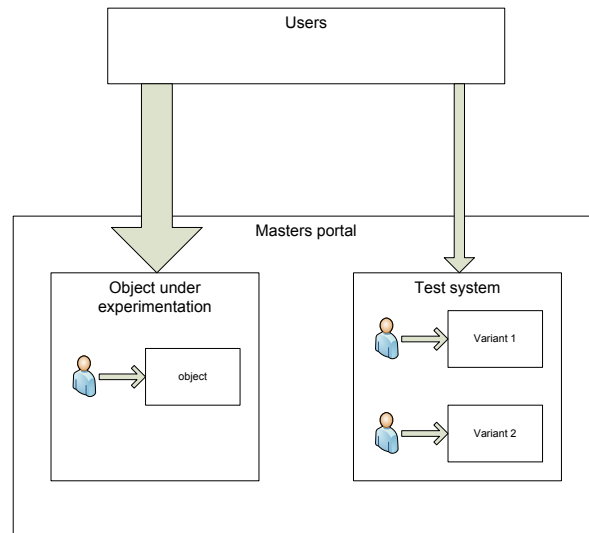


Figure 4.2: An overview of the online environment. The thickness of the horizontal arrows denotes the difference in traffic amount. The vertical arrows indicate which object is shown to the user.

4.2.5 The online environment

In the online environment is the operational environment of Masters Portal and is mainly considered to be the source of the logging data in this thesis. Moreover this environment contains an A/B testing environment that can be utilized to test variants of various objects available on the system like advertisement blocks, recommendations or search engines. In A/B testing the daily traffic of the system is divided between the object available on the web system and an experiment containing variations of this object. The user interaction on the object and variants is logged and compared to each other. In this way the quality of the variants can be evaluated. In case a variant has a better quality than the current used object in the web system, it can be chosen to replace the object of the web system with this variant. Figure 4.2 shows an overview of this A/B test system.

The literature analysis is described in this chapter. The analysis serves as input for the [Reference Architecture \(RA\)](#) construction process. The approach, scope and main references are elaborated in the remainder of this chapter.

5.1 Approach, scope and focus

The goal of the literature analysis is to obtain main requirements and architectural components of current and future recommender systems in order to use the findings as input in the [RA](#) construction process. First important components such as web usage mining and personalization [[MCS00](#)], user monitoring and profiling [[MSDR04](#), [MDG⁺09](#)] and popularity prediction and correction in web systems [[SH10](#), [Ste11](#)] are considered. Then analysis has been expanded towards context awareness [[ADB⁺99](#)], one of the main components in [Context-Aware Recommender Systems \(CARS\)](#). Architectural views and information of recommender systems are important in order to obtain insight in the way these systems are designed and constructed. An overview of [CARS](#) containing context and important recommender systems components has been explored in this literature analysis.

5.1.1 Main focus

The literature analysis is focused on multiple areas: First the focus lies on gaining insight in the active research areas in the recommender systems field in order to be able to adhere to additional requirements. Second application of contextual information in [CARS](#) is of importance. This information gives insight in how contextual information is used and available. Finally the focus lies on generalizing the components of the found architectural material and process flows for incorporation in the [RA](#). In this way the components adhere to the required *flexibility* and *adaptability* requirements.

Highlighted components are parts of the recommender systems that require more depth and insight because of their indispensability to the system or lack of detailed information. The highlighted components identified context awareness, monitoring, learners and processors.

There is extra focus on context awareness because a [CARS](#) requires that context can be used in all processes of the recommender systems. The goal is to obtain insight in how similar systems incorporate context. The acquired information is then incorporated in the context component of the [RA](#).

Monitoring is an important component of the recommender system. It allows insight in the quality and performance of the system. Therefore various aspects such as monitoring of the recommendation process, concept drift and anomaly detection are examined in order to outline the required connections to other components.

Finally a separation of functionality is enforced by the learner and processor components. Learners are instantiations of recommendation algorithms while the processors are the agents performing the remaining processing. This separation has been made since the pre-processing steps like raw data processing often require a different computation strategy and computation environment than recommendation algorithms. Additionally, the separation enables more *flexibility* and *adaptability* towards the scheduling and resource allocation of these components.

5.2 Related architectural work

The related work outlined in this section contains architectural views and information used in the construction of the RA. Unfortunately in most of the work on recommender systems, architectural information is usually described very briefly or is not mentioned at all. Furthermore, in the majority of cases only the high level views are given and implementation details are omitted. For example, in most of the references processing agents and data stores are provided and only the main components can be identified. Therefore important details that could be beneficial to the RA are often missing in literature.

Furthermore, a reference on empirical work on RA has been obtained as well. The reference is used to validate the envisioned process and to ensure that the proper steps were taken to construct the RA.

5.2.1 Architectural work on recommender systems

Cho, Kyeong Kim and Hie Kim utilize a recommender system in the field of e-commerce in [CKK02]. This system is required to incorporate possible associations between the purchase of products and customer preferences. The architectural overview of the system shows the required data stores, entry points of the system and the various processing agents.

Tran and Cohen propose a hybrid recommender system architecture in [TC00]. The system combines collaborative filtering with a knowledge based system; a system which recommends items based on explicit feedback the user provided. Interfaces, agents and data stores are the main components in the architecture. The architectural views show these general building blocks of components in the architecture for example in deriving the recommendations. Also the ability to combine data sources is displayed.

Middleton, Shadbolt et al. introduce an ontological approach to user profiling for recommender systems in [MSDR04]. Two systems are instantiated, of which the Quickstep recommender system is of interest. This is a hybrid recommender system in which content-based and collaborative approaches are used. Architectural views of the ontological approach and the Quickstep system are given. .

Mobasher, Colley and Srivastava also focus on user profiling, web usage- and association rule mining by proposing a general process flow in [MCS00]. The notions of online and offline are introduced to denote the used computation environment. Additionally a data pre-processing strategy is given allowing insight in the envisioned data processing for the recommender system. The fact that the recommendation process is actually continuous is important; the process should be adaptable and flexible in order to be able to quickly correct threats to the recommendation accuracy.

Abbar, Bouzeghoub and Lopez first show a global architecture of CARS in [ABL09]. The focus of their work lies in acquiring contextual information for user profiling. An architectural overview of a collaborative filtering based CARS is shown. The reference architecture shows the process of matching users to contextual profiles. This work, although not taken into account in the RA construction process, can serve as a starting point for detailing the user profiling part of the RA.

Berkovsky, Kuflik and Ricci show the notion of having remote systems accessible by a mediator server in their architecture in [BKR08]. This mediator is responsible for the user modeling functionality. Furthermore technical storage aspects of various used matrices for computations are outlined and elaborated. This work shows the importance of architectural aspects like system division, as well as the importance of data storage and handling in user modeling. Although not applied in the construction process of the RA, these concepts can support the physical view of the architecture.

Amatriain and Basilico elaborate the general system architecture of recommendations of Netflix in [AB13]. Netflix is a provider of on-demand media available in a variety of countries. In addition to the previously introduced notion of online and offline computation environments the architecture introduces the semi-online environment¹. The semi-online computation environment is a compromise between online and offline computations. Essentially offline computations are performed on the online environment and presented asynchronously to the user. An example is updating recommendations to reflect that a movie has been watched. This update computation then would start when the user starts to watch the movie in order to have updated recommendations when the user stops or finishes watching the movie. This architecture shows the incorporation of various computation environments and frameworks in the recommender system, as well as the need to schedule computations according to demand of the data and the availability of resources.

5.2.2 Reference architecture work

In [GA11] Galster and Avgeriou present an approach to empirically ground the validity and re-usability of a reference architecture (RA). Six steps are proposed to offer a guideline that can take RA construction from type selecting to design strategy to construction and evaluation. Having a RA construction guideline this is important in order to ensure of the completeness of the RA and to crosscheck with the envisioned design process.

5.3 Related work on application of context

In [AT05, AT11] Adomavicius and Tuzhillin outline three algorithmic paradigms to apply context in recommendations: *Pre-filtering*, *post-filtering* and *modeling*. Contextual *pre-filtering* means that the input data of the recommendation is selected based on the the relevant context. Contextual *post-filtering* uses context to filter irrelevant items or to adjust the ranking of items, which can be done by using heuristics or model-based techniques. The ranking adjustment strategy is similar to the *Context-Aware Recommendation Adjustment (CARA)* algorithm, especially the variant using weighting based on predicted relevance. The *modeling* strategy adds a third dimension to the recommendation space of users and items. Heuristics or model-based approaches can be applied. In case of contextual *modeling*, the context of the user should be available in order to retrieve the appropriate recommendation.

Furthermore, Abbar et al. [ABL09] outline a *Personal Access Model (PAM)*, which provides services ranging from context discovery to binding the contextual properties of the user to a set of profile

¹Netflix refers to this as the nearline environment.

preferences denoted as the operational profile. Together with the known concept of the user profile the operational profile is used in a contextual modeling strategy in the recommendation process to obtain contextual top- k similar users of which the item ratings are aggregated in order to derive a recommendation. This is a different approach to derive recommendations opposed to the [CARA](#) algorithm, yet the contextual profiling and the used data sources overlap.

Baltrunas and Ricci [[BR09](#)] experiment with incorporating context in collaborative filtering. The notion of item splitting is important: Items can be split up in multiple vectors according to various contextual conditions. This is also the case in the [CARA](#) algorithm: Recommended items are re-ranked differently according to the belonging context.

Finally approaches where users, items and context is represented as matrices and vectors are currently actively researched [[BKR08](#), [ASST05](#)]. The reasons behind this strategy is the combination of local machine learning methods, a variant on lazy learning where generalization of the query is delayed as long as possible and multi-dimensional data warehousing systems. The various approaches of finding similarities are elaborated when utilizing cross-user, -item, -context and -representation, where the latter one means incorporating user, item and context in the mediation are interesting with respect to matrix calculation. The strategies in both papers differ from this experiment in calculation perspective and explore a broader sense of similarity in recommendations.

5.4 Key architectures

Figure [5.1](#) shows the recommender systems architecture proposed by Cho, Kyeong Kim and Hie Kim in [[CKK02](#)]. This architecture outlines an e-commerce recommender systems tuned towards association rule mining and deriving recommendations based on customer preference models. The architecture contains a detailed process flow from raw data to the actual models.

In this architecture two main ingredients of recommender systems are visible: the usage of various data sources and a process flow from raw data to recommendation lists. Data processing is an indispensable activity in recommender systems and hence the [RA](#) must be able to incorporate these kind of process flows as well. Furthermore the data processing must *flexibility* and *adaptability* in order to be able to accommodate to changes.

Mobasher, Cooley and Srivastava also provide a recommender system architecture corresponding to data processing in [[MCS00](#)]. Figure [5.2](#) shows this architecture. Besides association rule mining usage mining is applied in order to obtain the required preference information from the user. The architectural view contains a detailed sequential outline of the required data processing, which can be used to identify and generalize components.

The data processing divided in an online recommendation deriving and offline processing. In recommender systems this separation, often extended with the notion of semi-online processes, is important because often different environments are used as well.

Moreover, it is shown that the recommendation process is in principle a continuous process. Providing the recommendations to the user does in principle not depend on the freshness of the data, and hence besides the separation in computation environment there is also a separation between providing recommendations and deriving them. This is an important notice which is taken into account in the recommendation part of the recommender systems.

Figure [5.3](#) shows the architectural approach Middleton, Shadbolt et al. used to instantiate the two systems as described in [[MSDR04](#)]. In this system usage data is classified towards a database of research papers in order to recommend similar research articles that have not been visited by the user. Hence in this approach clustering and classification algorithms and hence ML algorithms can

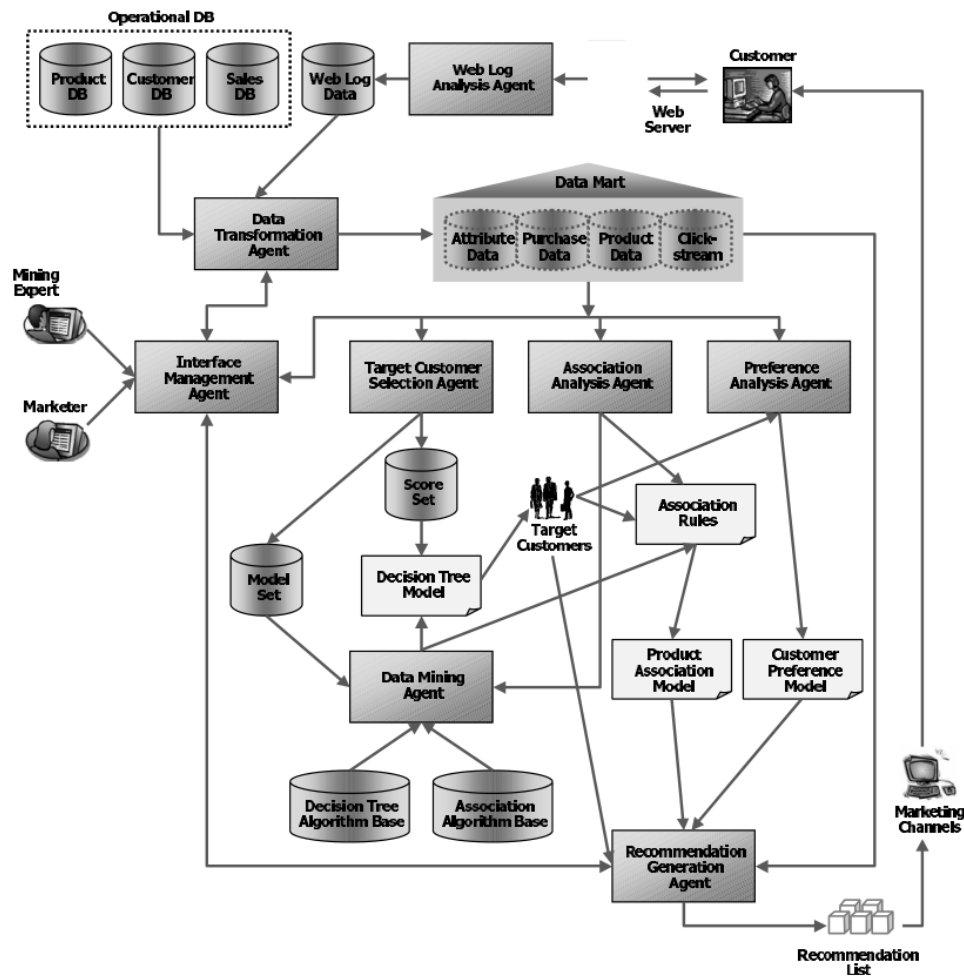


Figure 5.1: The recommender system architecture as proposed in [CKK02].

also be of importance in recommender systems.

The ontology object is a container holding relationships, entities, attributes and axioms supporting the recommendation process. This object is available throughout the recommender system and hence each component can contribute and benefit from the collected information. Hence the setup of this system shows the importance of having the required information available to all processes and components of the recommender system.

Figure 5.4 shows the general architecture used in Netflix as elaborated by Amatriain and Basilico in [AB13]. The architectural view focuses on showing the various used computation environments and data storage systems. These notions are important since it shows the requirements of being able to use various frameworks, environments and data storages. Furthermore, it shows the importance of being able to schedule the required processing based on demands in the system as well as availability of resources.

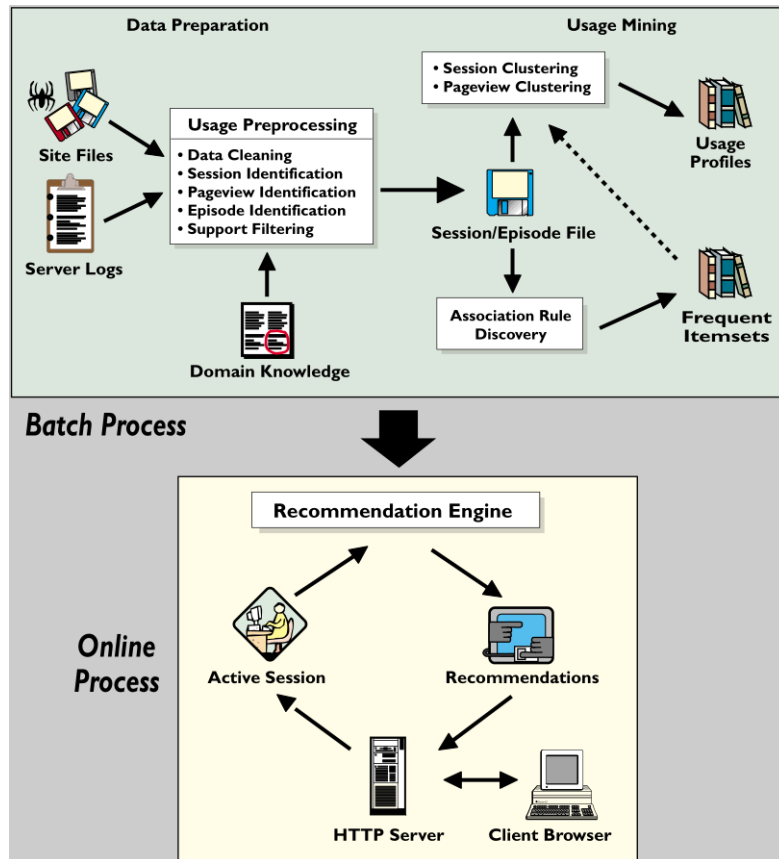


Figure 5.2: The recommender system architecture as used in [MCS00].

5.5 Missing information

5.5.1 Architectural and implementation details

Most of the architectural view found in the literature only contains the general high-level components and objects required in recommender systems. Detailed information such as specialization of components and types of relationships between objects are not shown. These details are important as input for a RA, because they can define the relationships amongst objects and components.

Furthermore, in case of an implementation based on the architecture the implementation details and feedback are often not mentioned. This information can be used to determine the quality of the architecture and offer an insight in the quality of the used strategies and approaches.

5.5.2 Design choices

In most of the referencing material the design choices of the shown architectural material are often not motivated. Examples of design choices are usage of architectural patterns, separation of functionality in various components and how relationships between components and objects are defined. Motivation of considerations in the design process of the architecture can offer insight in reason to design an architecture in particular ways, which in turn can be used in the construction process of the RA.

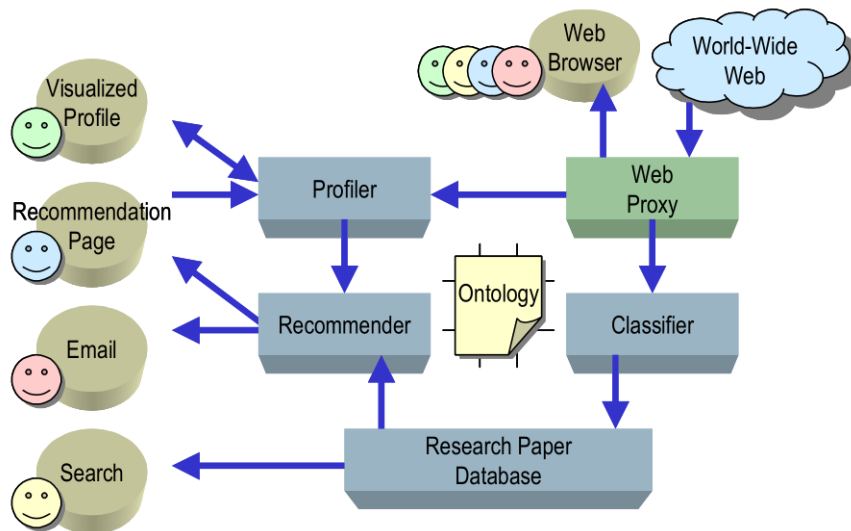


Figure 5.3: The recommender systems architecture as used in [MSDR04].

5.5.3 Used artifacts

The software artifacts used to construct a recommender systems. Incorporating external software artifacts can enforce the architecture of a recommender system to adhere to a particular programming paradigm or structure. This information can reveal part of the motivation why a particular recommender system architecture is designed.

5.5.4 Recommender data storage

In most recommender system architectures a simplification of the data storage mechanism in RA of recommender systems literature are given, which implies that the data is in the same format and database in each of the recommender system processes. This is not necessarily the case as frameworks like HADOOP often require semi-structured key-value data. Either a particular part of the data is in semi-structured for or the data sources produces the dataset when required. Hence detailed information about data storage strategies in recommender systems can be used to improve the data storage component of the RA.

5.5.5 Contextual data sources

In the referenced material about context, the approach of resolving contextual features is often not revealed. Also the data sources providing contextual information are often not described. The context library outlined in Section 8.2 shows that contextual data often has uncertain accuracy, especially when using data that is publicly available. Yet in most of the contextual work this is not mentioned. Analogous to this question the strategy used to process data in which the contextual features could not be resolved is often missing or not described. Insight in extensive processing of contextual features can yield additional detailed information applicable to the RA .

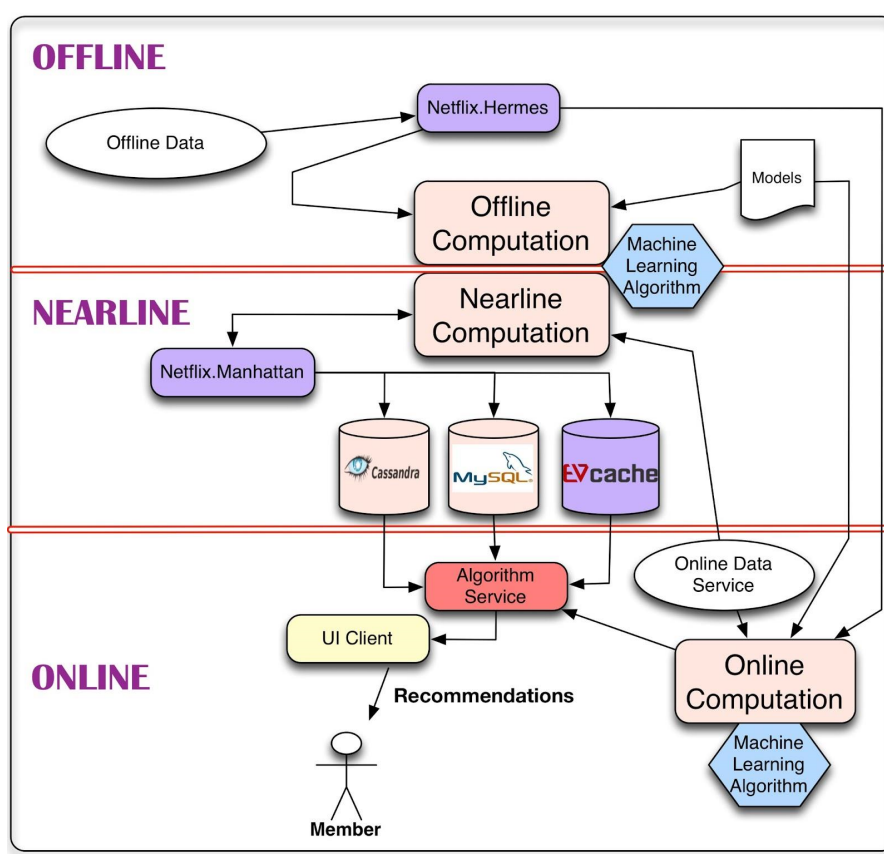


Figure 5.4: The recommender system architecture as proposed in [AB13].

This chapter outlines the system review conducted on the Masters Portal web system. In this system review the recommendation and context awareness opportunities of the Masters Portal web system are outlined by examining data sources and process flow. In the remainder of this chapter the approach, main focus and results are elaborated.

6.1 Approach

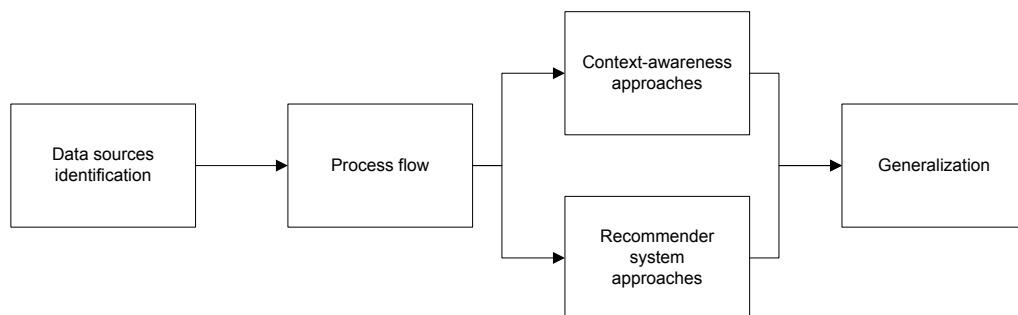


Figure 6.1: The outline of the approach of the system review.

The goal of the system review is to analyze the Masters Portal web system in order to inventory recommender system possibilities and interesting context applications. This goal is achieved by identifying the available data sources and process flow, analyzing the possible applications context awareness and recommendations and generalizing the findings in order to use them as input for the [Reference Architecture \(RA\)](#). The process flow denotes the navigational process of Masters Portal. Figure 6.1 shows the process outline for the system review. This setup of the system review enables the identification of data sources and possibilities, along with pitfalls in the process flow of the web system. Context awareness and recommender system approaches are proposed in order to improve the quality of the web system. The generalization step allows to obtain generalized components or strategies which could be applicable in other recommender systems as well.

6.2 Focus

This system review is focused on the data sources and the process flow of Masters Portal. The data sources are assessed in order to inventory the recommendation possibilities. The process flow is examined in order to find pitfalls in the process of the Masters Portal web system for instance when the user obtains an empty search result. A recommendation could help support the user and prevent the user from leaving. Also pitfalls caused by the objects on the system for example when the user lands on the *discipline* page it is desired to recommend appropriate pages to get the user back to searching a suitable master study. In this way it is attempted to cover the whole operational process of the system.

6.3 Key findings

6.3.1 Context awareness

Various contextual features are proposed to incorporate in various recommender system. Examples are the intention of the user (orienting or searching) or the degree of traveling the user is willing to make to study abroad. This application of context requires an outline of context which enables combining of contextual information. An example is combining of information about semesters of universities in various countries with the geographical location of the user. If the date of the visit in a particular interval in ending of semester of an university close to the user, then it is expected that the user visits the system to search for follow up education. Note that semester information would related to the activity context as outlined in Section 2.2.1.

6.3.2 Raw data processing strategy

An strategy to process the raw data from the identified data sources of the Masters Portal web system has been proposed. In this strategy data selection, extension with contextual information and anomaly detection and processing are outlined. This strategy has formed the base of the construction of the session data processing task as described in Section 8.5. Additionally, this strategy also shows that it is required to have *flexibility* and *adaptability* in the core processing of the raw data, depending on the application of the resulting dataset. For example, when the dataset is used to identify the resolved context for anomalies such as bots the data processing must not apply processing to remove anomalies, while a dataset used for recommendation bots are required to be removed.

6.3.3 Discipline content object

The main obtained finding is that the *discipline* can be considered as a weaker content object. There is no strong additional information to be found on the discipline pages. Hence it is more likely that users land on the discipline page out of curiosity or that a master study has been clicked which falls into the wrong discipline, rather than that actual information is necessary. Hence the goal on the discipline page is to guide the user to the search of master studies with the correct discipline. The latter requirement can be identified using a contextual feature that monitors the various discipline objects associated with the user and can be used in a recommendation towards the user. In this way a user can be encouraged to narrow the search towards a single discipline. These kind of contextual features can also be used in other systems as well, and are considered to be part of the activity context as considered in the context awareness paragraph in Section 6.3.1.

6.3.4 Recommender system improvements

Most of the recommendations generated on Masters Portal are content-based and hence do not include collaborative strategies, let alone the contextual situation of users. In the system review a variety of recommendation strategies is outlined, which can give Masters Portal an insight in the application of collaborative and contextual recommendation strategies.

The majority of the proposed improvements combine various recommendation algorithms and strategies. In order to achieve this the blending object is introduced in the [RA](#), as well as the tuning object which can be used for regulating the influence of a recommendation algorithm in the resulting recommendation. These objects are outlined in [Section 7.1.4](#).

The reference architecture

Figure 7.1 shows the proposed recommender systems reference architecture. This Reference Architecture (RA) is based on the literature analysis and system review as described in Chapters 5 and 6. The components are explained in the remainder of this chapter.

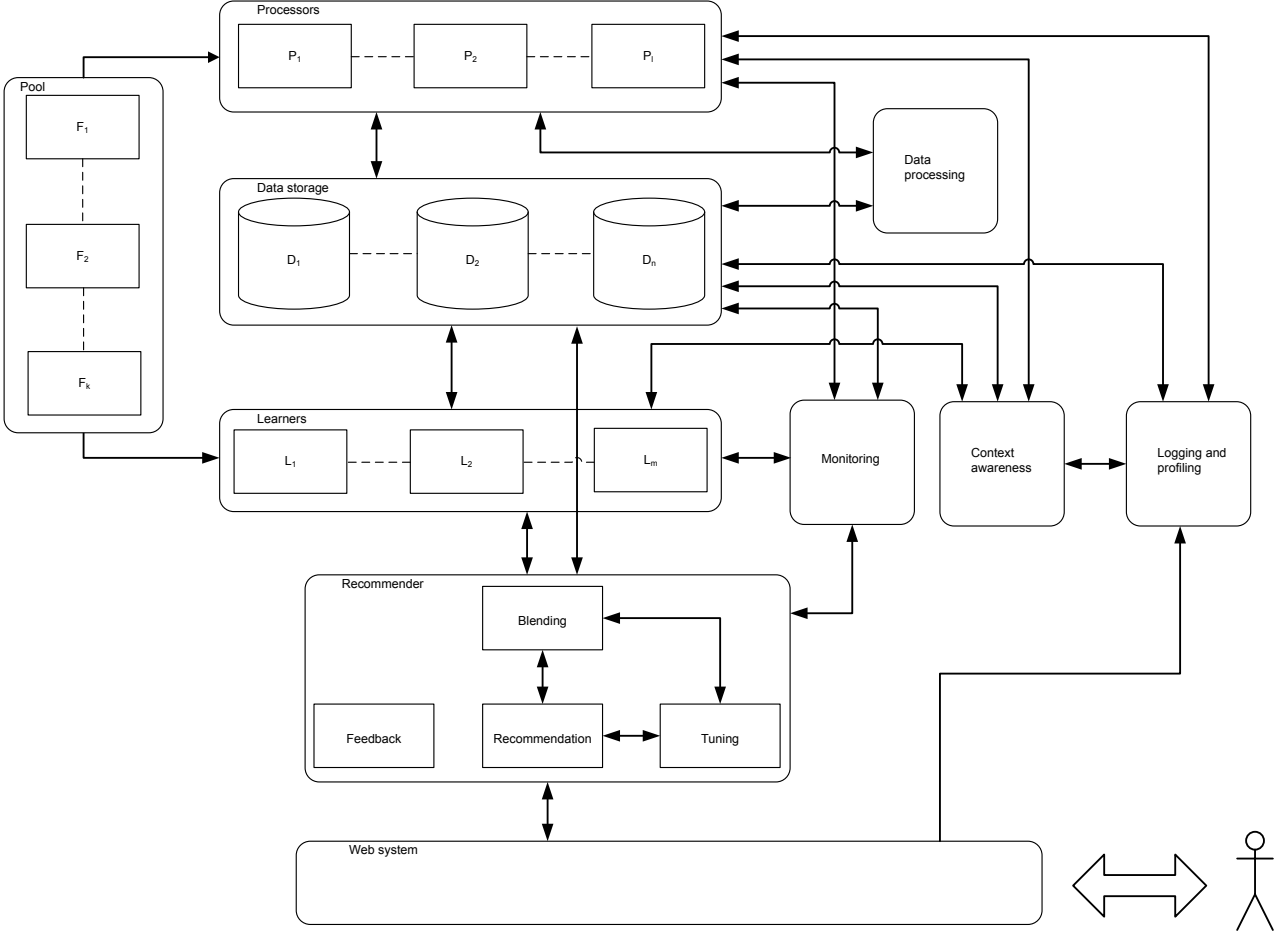


Figure 7.1: The proposed recommender systems reference architecture. The rounded boxes denote the main components, the rectangles denote objects and the cylinders represent data storage. The dotted lines denote that there can be multiple instantiations of objects, while the arrows denote relationships between objects and components. The actual type of the relations between objects and components is not defined.

7.1 Reference architecture components

7.1.1 The pool

Recommender systems are incorporate various software artifacts and frameworks. Therefore there is need for a central place for accessing libraries, frameworks and separate computation environments like HADOOP. The *pool* component is instantiated for this purpose.

7.1.2 Processors

The *processor* contains the actual processing agents; instantiations of software artifacts available in the *pool*. The purpose of the processor is to perform the algorithmic work except for the actual recommendation deriving. This component provides the required instantiations of software artifacts and implementations to perform the processing tasks in the recommender system. The reason to have this component is to allow for scheduling and the creation of process flows containing various processors utilized to complete a particular task in the system. Control flow, having a process scheduling based on conditional requirements, can be available allowing for setting up predicates between various processes. This enables the required *flexibility* and *adaptability* in the *RA*.

7.1.3 Learners

The *learner* component contains the instantiations of recommendation algorithms. The learners are used for the actual generation of recommendations. Recommendation algorithms from the *pool* are instantiated and the required data is available from the *data storage* component. The *blending* and *monitoring* components are connected to the *learner* component in order to obtain the required information for monitoring.

The *learner* component is used to have a separation between the offline data processing and online or semi-online recommendation parts of the general recommendation process. Because of the difference in computation strategy, *tuning* and incorporation of context awareness, it is required that the *learner* component is separate from the *processor* component. Moreover ensemble learning can be initiated as well. Hence *flexibility* and *adaptability* is ensured in the *learner* component.

7.1.4 The recommender

The *recommender* component engages the deriving of the recommendations along with the various comprehensive processes. A number of objects are available in this component. The *blending* object is used to combine various recommendations into one. The separation between blending and learners enables *flexibility* and *adaptability* in the scheduling of these processes. The *tuning* object is used to apply tuning on the recommendations. It is required to have *flexibility* in tuning for application in the various stages of recommendation processing. The *recommendation* object is an instantiation of the final recommendation which allows for easy querying and post-processing. Finally the *feedback* object is used to receive explicit feedback of the recommendations from the web system.

7.1.5 Monitoring

The *monitoring* component provides an insight in the operational processes of the recommender systems. Various monitoring possibilities as described in Section 2.3.8 can be applied on virtually any object in the system. Therefore a complete overview of the performance and quality of the system can be obtained.

The *monitoring* component offers a central place to obtain insight in the various monitored properties of the operational processes of the recommender systems. Comparison, blending and correlating of various monitor statistics can be performed in this component enabling a detailed insight in the performance and quality of the operational processes in the system.

7.1.6 Logging and profiling

The *logging and profiling components* component provides instruments to the web system to process usage logging and to construct the required user profiles. In this way the acquiring of usage data still required to be implemented in the web system, but the processing of this data is managed in the recommender systems. In this way the recommender system controls the data processing. Additionally the appropriate computation environment can be chosen by the system as well.

7.1.7 Context awareness

The *context awareness* component is used to provide context throughout each operational process of the system. It can be considered as a separate subsystem that has separate data sources and implementation. This enables the administering of the *context awareness* component without interfering in the operational process of the recommender system. In this way the user of the recommender systems can freely incorporate new contexts as well as constructing and combining contextual features which can improve the overall quality of the recommendations.

7.1.8 Data processing

The *data processing* component is used to administer the data processing tasks. Scheduling and routing of these tasks is the main goal of this component. The data processing tasks require a different strategy opposed to other operational recommender systems processes because data processing is time and resource consuming. Furthermore different computation environments like HADOOP can be used for the data processing and hence other paradigms might be in use which require a different application in scheduling.

7.2 Comparison with other recommender system architectures

In Section 5.4 three architectures used as main input for the RA are elaborated. The proposed RA is compared with these architectures in order to inventory the main differences.

The main difference between the RA and the described architectures is that the actual processing tasks or software agents are centralized in the *processor* component of the RA. This allows for centralized instantiation, scheduling and logging of the actual processing tasks. Therefore multiple tasks can

benefit from the same instantiation of a processing task, which allows for better scheduling and re-usage of processors.

Furthermore it is attempted to separate the various processing tasks of recommender systems. For instance there is a separation between recommendation algorithms and complementary processing tasks. This separation is important since data processing and recommendation deriving require different scheduling for example. Differences in complexity, computation environments and execution times require these tasks to be scheduled differently. Hence separation enables better scheduling and allocation to computation environments.

Complementary to the described architectures, the [RA](#) incorporates a subsystem to deal with context awareness enabling [Context-Aware Recommender Systems \(CARS\)](#). This allows the usage of contextual features throughout the whole system.

Furthermore the *monitoring* component is a complementary yet vital part of the [RA](#). Recommender systems are becoming more complex and insight in the performance of multiple components is required. Certainly, when using collaborative strategies which require usage data it is important to have insight in the quality of the data and the performance of the recommendation algorithms.

Figure 8.1 shows the proposed **Reference Architecture (RA)** as outlined in Chapter 7 and indicates the implemented components of the architecture. The data processing implementation utilizes the data sources as described in Section 4.2.4. The remaining implementations use separate data sources. The described offline HADOOP environment serves as the computational environment.

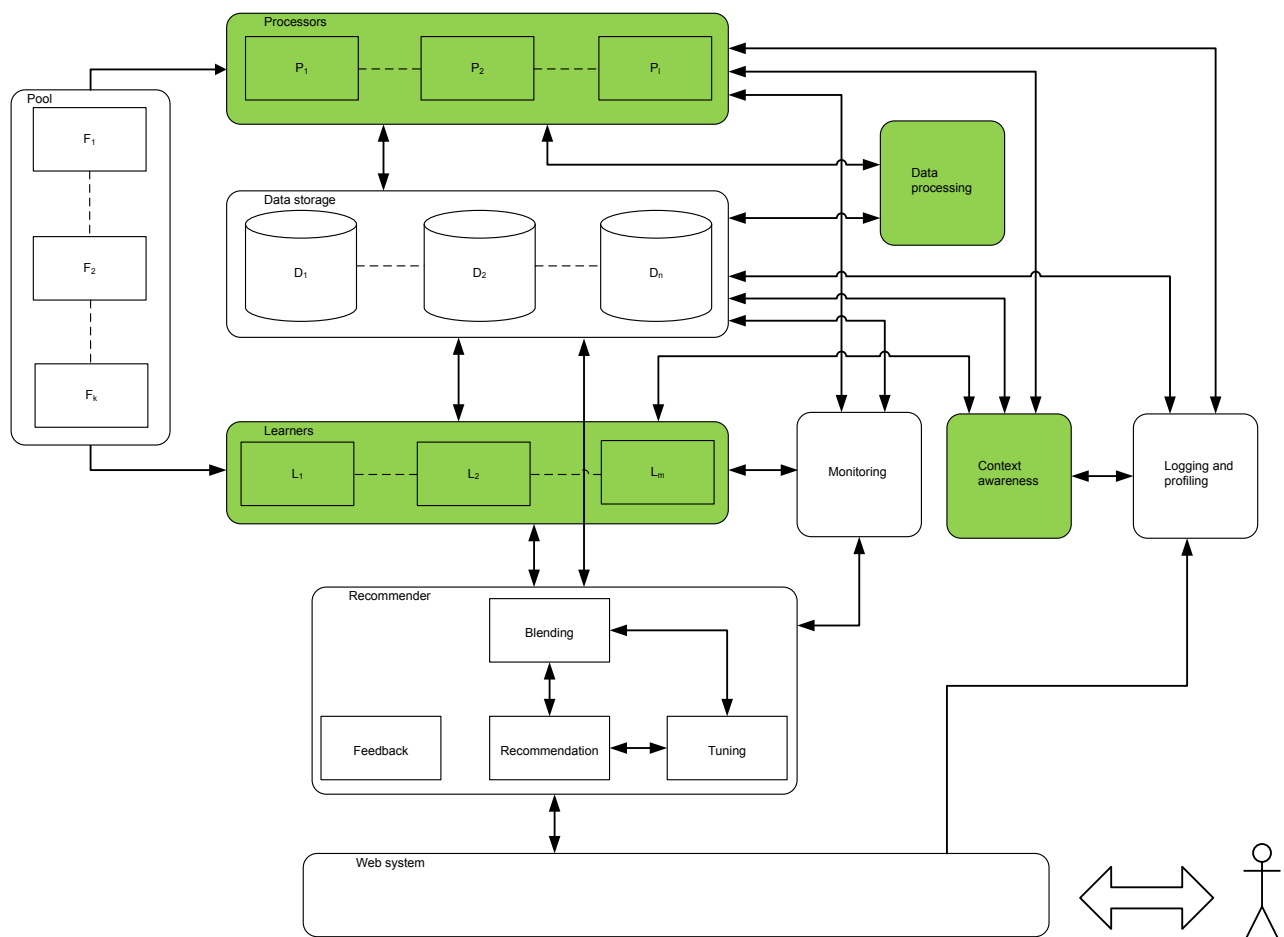


Figure 8.1: The proposed RA of Chapter 7. The colored blocks denote components which have been fully or partially implemented.

8.1 The offline environment

The offline environment is defined as a separate experimentation environment that does not interfere with operational environments of web systems. In this case the offline environment is a computational environment which does not interfere with the Masters Portal web system. The HADOOP framework is used in order to allow distributed computations enabling the processing of large amounts of data.

Two offline environments are used: the CAPA server, a shared server located on the campus of the university of technology of Eindhoven, and SURFSARA, is a shared server cluster located in Amsterdam. While the CAPA server is one physical machine with multiple cores, the SURFSARA cluster consists of 66 machines. The SURFSARA cluster is able process multiple tasks at once on large volumes of data. The drawback is that the SURFSARA cluster has a large user base as well and hence the execution times of the processing tasks are the same on both environments.

These computation environment is chosen based on availability; for example the CARA environment has not been available throughout the whole implementation phase. Being able to use any available computation environment¹ is one of the advantages of using HADOOP.

8.2 The Context library

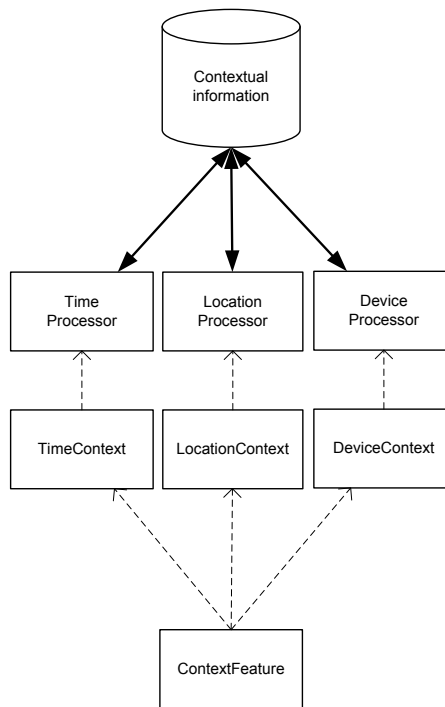


Figure 8.2: The general architectural outline of the context library. The double arrow lines indicate communication between the *processor* objects and the data source, while the dotted lines indicate usage and inheritance between various objects.

The context library resolves contextual features based on the characteristics of the user of the web system. The general specification as described in Section 2.2 is implemented in this library. The current available contexts are time, location and device. Figure 8.2 shows the general architectural outline. The *processor* objects are the implementation of contextual categories and are used to resolve

¹Given appropriate version of the HADOOP framework and the availability of used third party software artifacts.

the contextual information for the *context* classes, which provide the contextual information to the *ContextFeature* objects. The *ContextFeature* objects are the implementations of the actual contextual features, which provide define the set of contextual elements.

This architecture allows for dynamic construction of new context and contextual features. Moreover it allows contextual features to use combinations of different contextual features and and elements. Hence this architecture allows for *flexibility* and *adaptability* in construction of the required contextual features.

8.2.1 Data sources

In order to provide the required context the library uses a variety of data sources. The following data sources are used:

- *Timezones by country*: a dataset containing all countries and their timezones.
- *Coordinates to timezone*: a dataset containing coordinates of timezones. This dataset originates from the so-called Olson/timezone database which is commonly used in various software artifacts [Ols09].
- *IP address to country*: a dataset which can resolve IP ranges to a country.
- *IP address to city*: a dataset which can resolve IP ranges to a city constructed by MaxMind, a company focusing on IP to location resolving. The accuracy varies from 30% correct city resolving in Ukraine to 95% resolving in Cote D'Ivoire [Max13].
- *Country to continent*: a dataset which can resolve a country to a continent.

Only a few data sources have accuracy number available. Therefore it is difficult to reason about the quality of resolved contextual features.

8.2.2 Device context resolving

The third party library *UserAgentUtils* is used to resolve the device context. This library is used to resolve contextual information about the user's operating system, web browser and device. The library already offers a complete device context, which is implemented in the context library.

8.2.3 Limitations

The initial idea was to construct a context library that can be extended and enables inheritance between contextual features. Currently the library is lacking the required interfacing and typing, making extending of the code difficult. Also the context resolving part of the library requires more generalization work in order to be able to have better adaptation towards new contextual data sources. The context library is thus an effective library to use in combination with the data processing for example, but to use it as a general provider of context, more architectural work is necessary.

8.3 Learners

The *learner* objects as described in Section 7.1.3 can be instantiated the MAHOUT library. Two recommendation jobs are available: the *ItemSimilarityJob*, which computes a set of items similar

to the preferences of the user, and the *RecommenderJob*, which is a set of recommended items. The session data as elaborated in Section 4.2.4 is used to derive recommendations.

The output of these jobs can be directly used as a recommendation. Together with their high amount of configurable options, it is shown that the MAHOUT recommendation jobs are suitable for instantiating the *learner* objects.

8.4 Processors

The various PIG and Map-Reduce implementation opportunities can be considered to be instantiations of the *processor* objects as described in Section 7.1.2. The main advantage of this is that distributed processing can be performed, which enables the processing of large amounts of data. Also many options can be parameterized allowing for *flexibility* in the configuration of various *processor* objects.

8.5 Session data processing

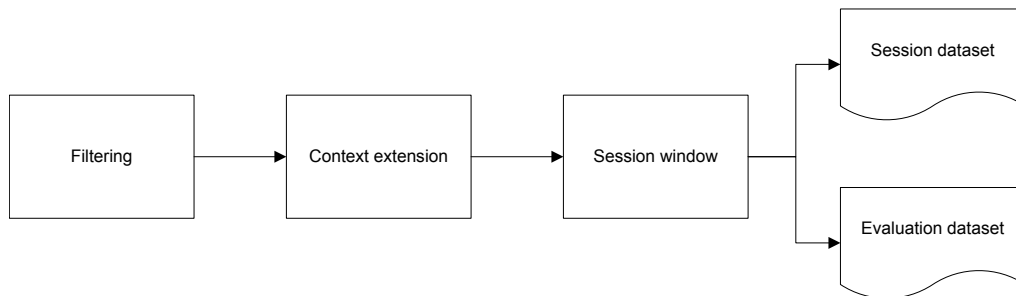


Figure 8.3: The session data processing subtasks and resulting datasets.

The data processing component consists of one processing task that processes the Apache access log data as described in Section 4.2.4. Figure 8.3 shows the subtasks of the data processing. The data filtering subtask requires that each log line contains a non-empty IP address, user agent and request fields. This filtering serves as a validity check of the access log under processing. These fields are required for the context extension task. In this task the data is extended with the available contextual features as elaborated in Section 8.2.

The result of the session data processing task is the session dataset. This dataset is used in the data processing task as outlined in Section 11.2. Furthermore, the evaluation dataset consists of input, output and intermediate data volumes. Moreover, the number of entries by IP address and IP address and category is counted. Also the total session time per IP address is calculated. The evaluation dataset offers an insight in the performance of the session data processing tasks, as well as an insight in various characteristics of the data.

8.6 The web service

The web service is an object that enables communication between various servers. This object can be used in the *RA* to enable communication between the recommender and web system. The web service is constructed using JAX-RS, which is a Java based *Application Programming Interface*

(API) that allows to bind web requests to code using annotations. JAX-RS uses the [Representational State Transfer \(REST\)](#) protocol to enable the required communication [[PGP13](#)].

The advantage of JAX-RS is that it is easy to implement in existing projects. Basically an object must be available which contains annotated methods that are able to send and receive the required information. Often the communication objects are different from the internal data objects. In this case a transformation is necessary as well.

The web service object has been designed in advance of online experimentation as outlined in [Appendix B](#) to serve as access point for the online environment to request recommendation. Furthermore, the web service object has been used in practice in another experiment on the Masters Portal web system where it was used to provide a calculation based on input data. This practical application of the web service shows that it is suitable to use in the [RA](#).

8.7 Implementation feedback

8.7.1 Interfaces and typing

There is need for specialization of the processors and learners. For example there is a clear difference between context awareness and data processing processors. This difference is in the type of object; data processors are HADOOP implementations while context awareness processors can be implemented as regular classes. For the learners there should be a clear separation in the actual recommendation algorithm as well as pre- and post- processing for recommendations.

8.7.2 The control mechanism

The scheduling of the various processors and learners is of importance for the functioning of the recommender system. Various different processes for data processing, recommendation algorithms and other processing tasks often depend on each other. Therefore a scheduling component needs to be incorporated in the [RA](#) which is able to schedule and control the various processing necessary in recommender systems.

8.7.3 Component outline

Each of the data processing, logging and profiling, context awareness and monitoring components needs more specialization. The various objects of these components, like administration entry points or monitoring tasks, must be outlined in the [RA](#). In this way the [RA](#) will contain a more complete and structured architectural view.

9.1 Results and contributions

A case study has been conducted in order to propose a reference architecture for [Context-Aware Recommender Systems \(CARS\)](#) consisting of a literature analysis and a system review. The literature analysis resulted in an overview of architectural work on recommender systems, the application of context in [CARS](#) and specific aspects of recommender systems such as monitoring or user profiling. The system review contributed to the [Reference Architecture \(RA\)](#) by showing that contextual information is required to be combinable. For example the experience with installing software can be estimated by determining whether the user is using the standard installed browser on his operating system. Furthermore, the system review shows that the core processing is required to be flexible in scheduling and requires control flow. The outlined process flow for the raw data processing shows that anomalies detected could require different processing strategies. For example, anomalies caused by bots are required to be filtered out of the data while a popularity outburst on an item would only require popularity correction. The data processing work flow has been the base of the session data processing implementation.

Evaluation of the [RA](#) has been performed by constructing a partial implementation. This implementation consists of the processor, data processing learner and context awareness components. Additionally, communication between the recommender system and web system has been simulated with the implementation of the web service. The HADOOP, PIG and MAHOUT frameworks are used to construct these implementations. The implementations show the correctness of the [RA](#) as they fit in the proposed architecture. Furthermore the separation of processing functions and processing task adheres to the required *flexibility* and *adaptability*, since subtasks can be changed and tested locally without using the HADOOP environment. Additionally, the HADOOP environment also adheres to the *flexibility* requirement as processing tasks can be scheduled and configured in a flexible way. The environment handles the required resource management to have fair scheduling in the execution of processing tasks.

Furthermore, the implementations have contributed to experiments conducted on the Masters Portal web system and data. The session data processing has been used in a data analysis experiment giving insight in the division of contextual information over the user base of the Masters Portal web system. The web service has been used as communication component between the Masters Portal web system and experiment base server in an experiment that uses contextual information to modify search results the web system.

The evaluation shows the required *flexibility* and *adaptability* in the core components of the [RA](#).

This is supported by the architectural properties of the HADOOP framework. The required minimal interference between recommender system and serving system is shown by only allowing communication when the web system requests recommendations and to enable the required logging and profiling. In the latter communication it is preferred to have the recommender system push the required information based on opportunities, for example when the system does not have high visitor rates. Additionally, the recommender system has a separate data storage component which also benefits the minimal interference requirement.

9.2 Limitations

9.2.1 Generalization trade-off

As stated before in Section 1.3 in each reference architecture there is a trade-off between generalization and specialization. In the RA this problem arises as well in the absence of incorporating core components in the monitoring and logging and profiling components such as schedulers that can schedule core processes or object used to visualize monitoring results. Furthermore, components also require access points that can be used to administer the system, modify scheduling of core processing and assess the performance and quality of various components in the system. Incorporating the described core components and access point in the RA can improve the quality of the system outline.

9.2.2 Evaluation of the reference architecture

Evaluation of the RA has been done by the partial implementation and the used frameworks. Implementation of the complete system will enable evaluation of the remaining components. Furthermore, in the current evaluation the required scheduling of the various required processing tasks is not included in the current evaluation, but is of importance of the overall effectiveness of the system. When too many processing tasks are executed at the same it can cause increased processing times which can delay depending tasks in the system. This evaluation can be achieved by simulation scheduling use cases with a complete implementation.

9.3 Future work

9.3.1 Specialization of the reference architecture

The components outlined in the RA require more detailing and specialization in order to obtain a complete system outline. Furthermore, access points in the system used for administer the system are currently not incorporated in the architecture. This also requires incorporation of scheduling objects usable to provide the required core function scheduling.

Furthermore, the advantages and disadvantages of frameworks and other useful software artifacts must be outlined much more in order to be able to recommend a preferred setup for the system. The RA could benefit by incorporating architectural properties of these frameworks. An example is including a distributed database system in order to support for distributed core processing.

9.3.2 Complete implementation of the system

A complete implementation of the RA can contribute to the quality and validity of the architecture. Useful feedback, such as more scoping with respect to frameworks and components, as well as architectural requirements and interfaces can be yielded by this prototyping. Test cases can be defined to evaluate requirements such as adapting the process flow of recommendation deriving based on the results of anomaly detection or automatic tuning of various recommendation based on monitoring of recommendation quality.

Part II

The CARA algorithm experiment

This part contains the experiment that is performed with the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm. The goal of this experiment is twofold: First we want to show that the [CARA](#) algorithm effectively re-ranks and improves the quality of existing recommendations. Secondly, the experiment is used as a practical case for testing partial implementations constructed to evaluate the [Reference Architecture \(RA\)](#) as described in part I. The context library and the session data processing task described in respectively Sections 8.2 and 8.5 are evaluated in order to assess the quality of the proposed [RA](#) for [Context-Aware Recommender Systems \(CARS\)](#).

The data used in this experiment originates from various log systems available on the Masters Portal web system. The recommendation interaction and session datasets as outlined in Section 4.2.4 are used. In the recommendation usage the *click* and *view* events are used as elaborated in Section 2.1.1. A delay of five minutes is allowed between the display the recommendation and the actual visit of a recommended item.

The experiment focuses on the data logged in July 2013. Three datasets are generated from this data:

- *Top-100 dataset*: the 100 recommendations having the most views and/or clicks
- *Random-100 dataset*: 100 random recommendations.
- *Complete dataset*: the whole dataset of available recommendation interaction data.

The top-100 dataset contains the recommendations having the most interaction. Hence this dataset provides an insight in the expected improvement in the optimal case for the [CARA](#) algorithm on the Masters Portal recommendations. The reason to use this subset is the distribution of the visitor data as described in Section 13.2.1, which shows the presence of a long tail in the data. Using the 100 recommendations having the most interaction data ensures that the optimal case is selected. The random-100 set provides an insight in the average expected improvement when applying the [CARA](#) algorithm on random selected recommendations. Finally, the whole dataset is used to determine the overall expected improvement shown when the [CARA](#) algorithm would be used on all available recommendations.

The experiment is performed on the offline environment as described in Section 4.2. The implement consists of 9 main tasks, constructed using the HADOOP and PIG frameworks.

Exploratory data analysis approach is used to perform the experiment. The available contextual features have been explored to obtain a suitable candidate for application in the [CARA](#) algorithm. This exploration is performed by evaluating the distribution of contextual features over the recommendation

interaction data. This resulted in choosing the contextual feature browser in this experiment. Note that mobile browsers are grouped together in order to have sufficient data and an improved distribution over the data. The same strategy is applied on browsers that are not actively used as described in Section 11.3. The resulting re-ranked recommendations are evaluated using various metrics. The difference between the original ranking and the re-ranking is measured with the **Kendall tau Rank Correlation Coefficient (Kendall correlation)** described in Section 2.6.3. The expected quality improvement is measured with the ratio of improvement explained in Section 2.6.5. Additionally the **Click-through Rate (CTR)**, **Contextual Click-Through Rate (CCTR)** (2.6.1) and **Contextual Coverage (COV)** (2.6.1) metrics provide the distribution of the contextual categories among the recommendation interaction data.

The experimental results are an indication of the potential improvement the re-ranked recommendations produced by the **CARA** algorithm. The main limitations of this experiment is that the **Ratio of Improvement (ROI)** evaluation is hard because of low interaction with the recommendations in the majority of cases. This causes that recommendation of which only a few items are re-ranked are expected to be a huge improvement. Furthermore, offline evaluation of the potential improvement and quality of recommendations is hard due to the missing interaction with the user. Therefore, further experimentation on the online environment of Masters Portal is required to have an improved evaluation of the potential improvement of the re-ranked recommendations produces by the **CARA** algorithm.

Finally, the session data processing task and context library of the partial implementation constructed are used in this experiment in order to evaluate the quality and effectiveness. The implementations have shown that these requirements are met by having separation between processing function and task which improves the changing and testing of the code. Furthermore, the resource scheduling and parameterization possible in HADOOP contributes to the *flexibility* and *adaptability* of the implementation.

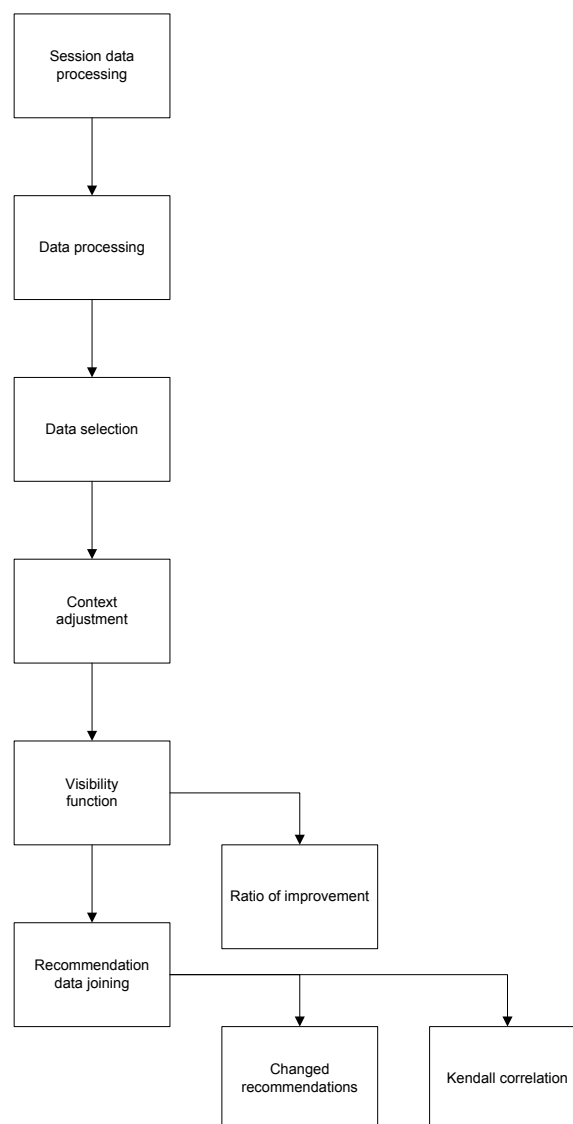


Figure 11.1: The implementation outline of the experiment.

The implementation outline of the experiment consists of 9 processing tasks as shown in Figure 11.1, which are sequentially performed. The implementation is focused on data processing and deriving of the various metrics. Furthermore, evaluation datasets containing intermediate results are used in various tasks to be able to evaluate and monitor the quality and results of the processing tasks.

The implementation uses the HADOOP and PIG frameworks as described in Section 2.4.2. These frameworks enable distributed computing that is expected to lower the execution times for data intensive processing tasks.

In the remainder of this section various images are shown outlining the input and output datasets for each processing task.

11.1 Session data processing

The session data processing task processes the raw input data, as described in Section 4.2.4, to semi-structured session data extended with contextual features. This data is used to provide the required contextual features used in the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm. The session data processing is elaborated in Section 8.5. The resulting session data is used in the data processing to extend the generated click and views data with contextual features.

11.2 Data processing

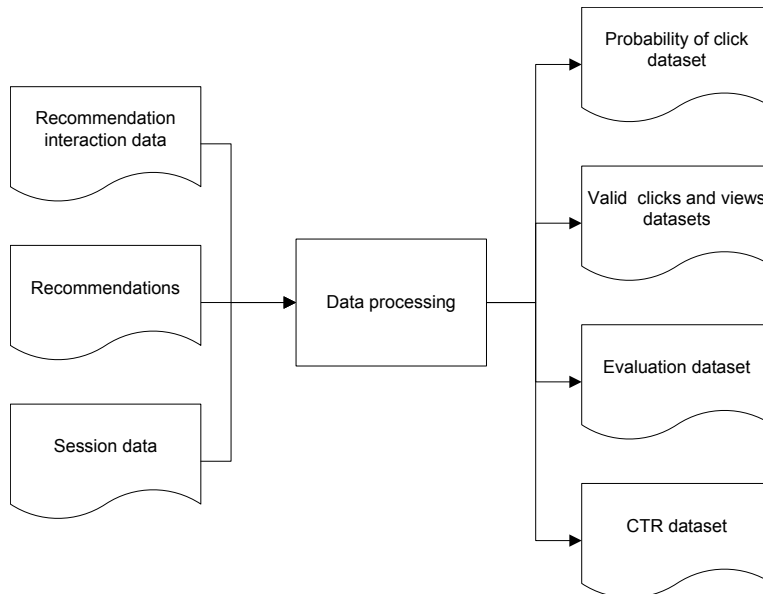


Figure 11.2: The input and output datasets of the data processing task.

The data processing task is used to combine the recommendation interaction data with the session dataset. Figure 11.2 shows the outline of this task. The valid click and views data is generated from the recommendation interaction data as described in Section 2.1. Furthermore, this dataset is extended with the session data in order to have the contextual features available. Based on this data the [Probability of Click \(POC\)](#) as described in Section 2.6.4 is calculated. This dataset is used in the ratio of improvement processing task as described in Section 11.6. Also the [Click-through Rate \(CTR\)](#) as elaborated in Section 2.6.1 is calculated based on the filtered clicks and views dataset.

The evaluation dataset contains data used to evaluate the performance of the processing task and the quality of the data. The difference data consists of the clicks and views are not included in the output dataset, for example when extension with the session data did not succeed. This dataset is used to check whether there are problems in the process of generating click and views data and extending with session data. Additionally, the data volumes of the input and output datasets are calculated contributing to the evaluation of produced output datasets.

11.3 Data selection

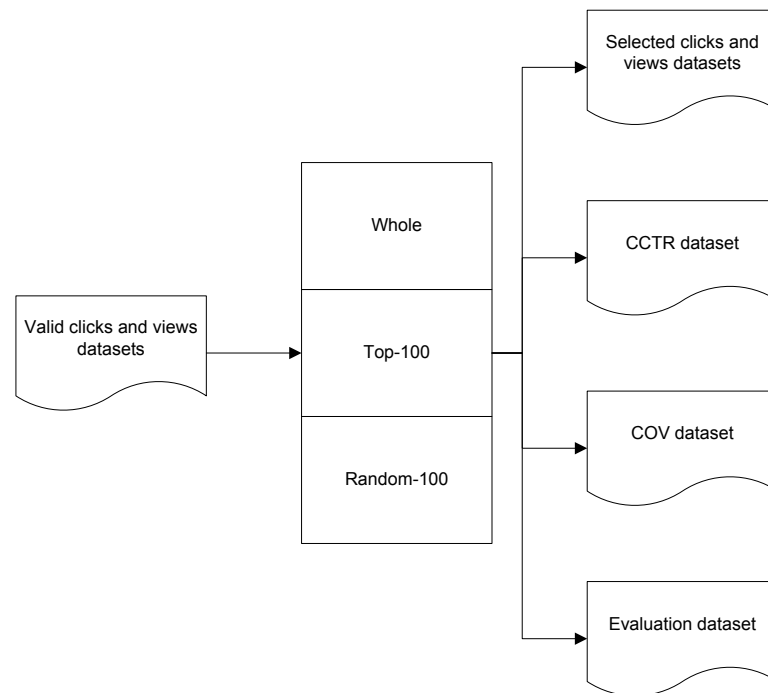


Figure 11.3: The input and output datasets of the data selection tasks.

The selection processing task consists of the clicks and views data selection for the top-100, random-100 and complete datasets as described in Section 10. Furthermore, the [Contextual Click-Through Rate \(CCTR\)](#) and [Contextual Coverage \(COV\)](#) metrics as described in respectively Sections 2.6.1 and 2.6.2 are calculated in this task since these metrics are depending on the selected data. Finally, the evaluation dataset consists of the distributions of the available contextual features over the data. This dataset has been used in order to obtain an insight in the suitability of features to be used in the [CARA](#) algorithm.

Moreover, the mapping of contextual elements is can also be performed in this task. In the contextual feature browser used in this experiment all mobile browsers are represented by the contextual element Mobile. Browsers having too less interaction data are represented by the contextual element Other. An example is Konqueror which is the default browser of Linux. This mapping can be provided to the available contextual features in the data by applying a processing function in which the mapping is specified. In this way the contextual elements of the browser feature contain sufficient data in order to have decent results in the adjustment recommendation generated by the [CARA](#) algorithm.

11.4 The CARA algorithm

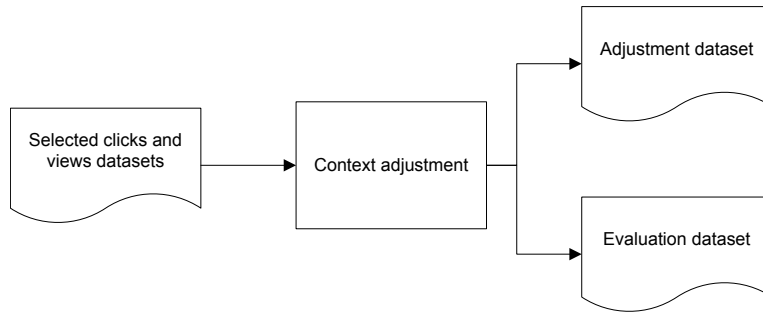


Figure 11.4: The input and output datasets of the context adjustment processing task.

Figure 11.4 shows the outline of the CARA processing task. In this task the CARA algorithm as elaborated in Section 2.5.2 is applied selected clicks and views datasets from the previous Section. This results in the context adjustment dataset. The evaluation dataset contains data about the data volumes of the input and output datasets used in order to obtain information about the number of actual re-ranked recommendations.

11.5 Visibility function

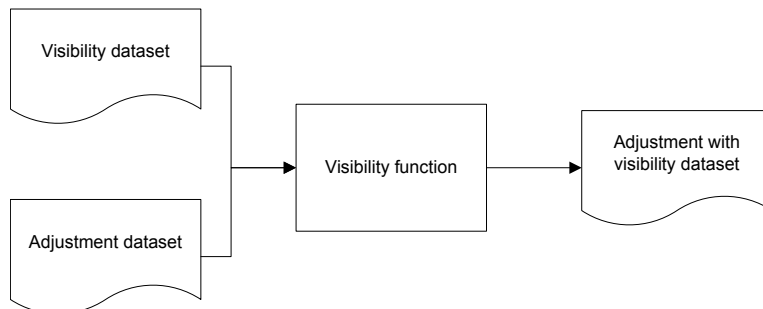


Figure 11.5: The context adjustment processing task and resulting datasets.

Figure 11.5 shows the outline of the visibility function processing task. This processing task applies the visibility function as outlined in Section 2.5.1 on the adjustment dataset. This is done based on visibility data that represents the preferred visibility function.

11.6 Ratio of improvement

The ratio of improvement processing task as shown in Figure 11.6 is an evaluation metric of the potential expected improvement as described in Section 2.6.5. This measure is applied to the adjustment with visibility dataset in order to only examine the actual re-ranked recommendations. The statistics dataset contains the mean, variance and standard deviation of the improvement scores. These statistics datasets are part of the results as described in Section 12. The evaluation data consist of intermediate datasets used to check whether the calculations are applied correctly.

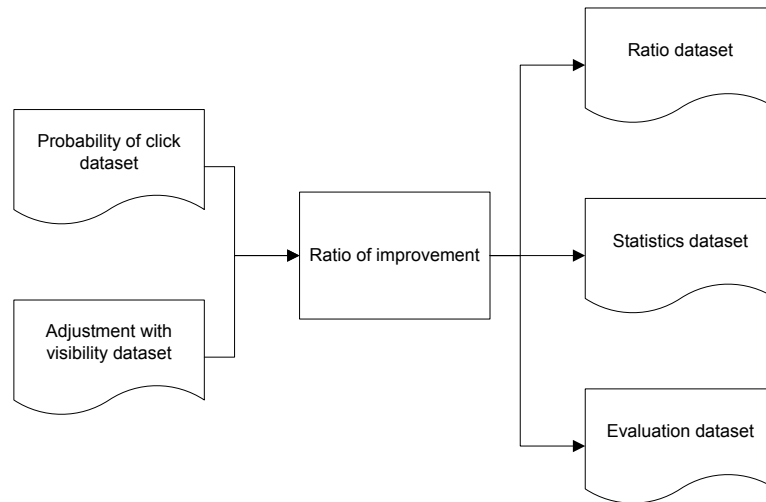


Figure 11.6: The context adjustment processing task and resulting datasets.

11.7 Recommendation data joining

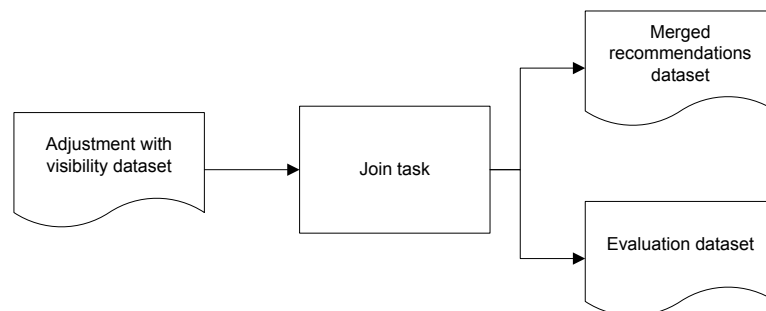


Figure 11.7: The input and output datasets of the recommendation data joining processing task.

Figure 11.7 shows the resulting datasets for the recommendation data joining processing task. The resulting dataset of [CARA](#) algorithm only contains the recommended items for which clicks and views are generated and selected. When an input recommendation has too less clicks and views then the resulting re-ranking produced by the [CARA](#) algorithm does not contain all recommended items from the input recommendation anymore. Therefore the re-ranking is completed with the history recommendation data in order to obtain the merged recommendations dataset which is used in the changed recommendations processing task as described in the next section and the Kendall correlation processing task as described in Section 11.9.

The evaluation dataset contains the ordered output of the merged recommendations used to evaluate whether the ranking of items is still consistent.

11.8 Changed recommendations

The changed recommendations task is used to get an insight in how many cases the [CARA](#) algorithm actually changes a recommendation. A recommendation is changed if there is a recommended item which has obtained a different rank in the re-ranking produced by the [CARA](#) algorithm. The degree of change, the amount of difference between the original recommendation and the re-ranked

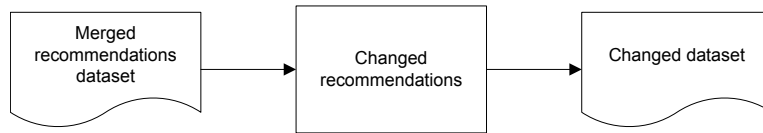


Figure 11.8: The input and output datasets of the changed recommendations processing task.

recommendation, is measured with the Kendall correlation in the next section. The outline of the changed recommendation processing task is displayed in Figure 11.8

11.9 Kendall correlation

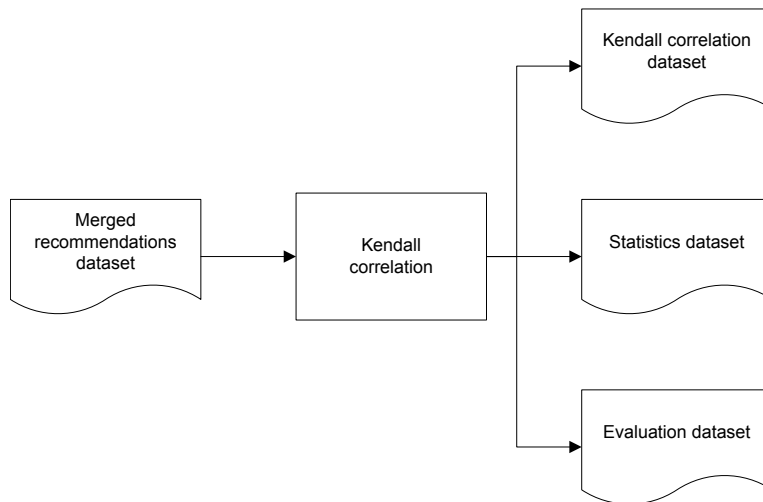


Figure 11.9: The input and output dataset of the Kendall correlation processing task.

The Kendall correlation processing task applies the Kendall tau rank correlation as outlined in Section 2.6.3 to the merged recommendation dataset. Furthermore, the statistics dataset consists of the mean, variance and standard deviation of the Kendall scores for each contextual element. These datasets are part of the results as described in Section 12. The evaluation dataset provides an overview of the [Kendall tau Rank Correlation Coefficient \(Kendall correlation\)](#) by showing the distribution of the scores in the proposed evaluation intervals. This dataset is assess in order to get an insight in the degree of change of the re-ranked recommendations of the [CARA](#) algorithm.

This chapter contains the results of the experiment with the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm. The experiment is conducted with the settings outlined in Chapter 10 and the implementation as described in the previous chapter.

12.1 Data volumes and CTR

From the history recommendation data 163,681 views of Masters Portal study pages and 902,051 recommendation impressions are extracted. From this data 120,892 clicks and 428,750 views are generated and extended with the contextual features of the session data in the data processing task as outlined in Section 11.2. The [Click-through Rate \(CTR\)](#) is 18.1%. Note that the [CTR](#) is based on the history recommendation data in order to have an insight in the exact usage of the recommendations on the Masters Portal web system.

12.2 Recommendation interaction

Figure 12.1 shows the distribution of the contextual elements over the browser feature for the generated click and views data as described in Section 11.2. The data is selected according to the selection task as outlined in Section 11.3, and used in the [CARA](#) algorithm to re-rank the input recommendations. It is shown that Chrome, Firefox and Internet Explorer have the most interaction by the distribution of the whole dataset. Also the Other browsers category is significant in the distribution of the whole data, however the distribution of the top-100 dataset shows that the interaction is not among the most used recommendations. In the random-100 dataset the Opera browser is more present in the distribution than in other datasets. Hence it is expected that in general the browsers Chrome, Firefox and Internet explorer will show the most interaction in all datasets. The [Contextual Click-Through Rate \(CCTR\)](#) and [Contextual Coverage \(COV\)](#) as outlined in Sections 2.6.1 and 2.6.2 provide an insight in the interaction with the recommendation per contextual element. The second column of the tables 12.4, 12.5 and 12.6 shows the measured [CCTR](#) for the three generated datasets. In the top-100 dataset the browsers Mobile, IE, Chrome and Firefox have highest [CCTR](#) score. This indicates that apparently Chrome users have less with the recommendations, considering the distribution of the browser in all datasets. Also Mobile browser users are actively using the recommendations considering its share in the distribution over the whole dataset. Furthermore, in the random-100 dataset the Safari browser shows high interaction. This is due to the selection of the data.

The third column of the tables 12.4, 12.5 and 12.6 shows the measured **COV**. The **COV** can be used to see how much the clicks of a contextual element are part of the total **CTR**. The results show that Chrome, Internet Explorer and Firefox have the largest share in the **CTR**. So despite the lower **CCTR** for the Chrome browser, it shows a lot of coverage. Also the Mobile browsers show high coverage confirming the findings of the **CCTR**.

The distribution of contextual elements over the datasets together with the **CCTR** and **COV** metrics indicate that the Chrome, Internet Explore, Firefox and Mobile browsers are expected to show the most change in the re-ranked recommendations and expected improvement for the **CARA** algorithm. The evaluation of this expected improvement is outlined in the next section.

12.3 Improvement evaluation

In order to evaluate the potential quality improvement of the re-ranked recommendations produced by the **CARA** algorithm the degree of change between input and re-ranked recommendation, as well as the expected improvement are evaluated. The degree of the difference is evaluated with the **Kendall tau Rank Correlation Coefficient (Kendall correlation)** as described in Section 2.6.3. The **Ratio of Improvement (ROI)** as outlined in Section 2.6.5 evaluates the potential improvement of the re-ranked recommendations.

12.3.1 The Kendall tau rank correlation

The fourth column of the tables 12.4, 12.5 and 12.6 shows the mean, variance and standard deviation for the **Kendall correlation**. First, the overall results are assessed. Then, dataset specific results are elaborated. Furthermore, the **Kendall correlation** scores are evaluated towards $[-1, 0)$, $(0, 1]$ and $[0.5, 1]$ intervals in order to obtain an insight in the degree of difference in the general case.

In general the averages range from 0.01 to 0.62 indicating that on average there is difference between the re-ranked and input recommendations. The standard deviation shows that the values are spread out from significant to slight improvement.

The top-100 show the most degree of difference between the re-ranked and input recommendations by having averages close to zero. Furthermore, in 38% of the cases the **Kendall correlation** score is lower than 0 indicating a significant difference between re-ranked and input recommendation. Moreover, in 62% of the cases the **Kendall correlation** score is higher than zero, and in 27% of the cases the score is between 0.5 and 1, indicating a small to no difference between re-ranked and input recommendation. Hence 72% of the re-ranked recommendations are indicating a significant degree of change. These results were expected since this dataset consists of the recommendations having the most interaction. In the random-100 dataset 23% of the cases have a **Kendall correlation** score lower than 0, and 72% has a score higher than 0. In 45% of the cases the **Kendall correlation** score is between 0.5 and 1. Therefore, the degree of change in this dataset is not as significant as in the top-100 dataset. Therefore, the results random-100 show that on average it is expected that the re-ranked recommendation will have a significant degree of difference with respect to the input recommendation in half of the cases. In the whole dataset 14% of the cases have a **Kendall correlation** score lower than 0, and 85% higher than 0. Moreover, in 60% of the cases the score is between 0.5 and 1. These results shows a slight overall improvement of the re-ranked recommendations in the general case.

Concluding, the re-ranked recommendations based on the Chrome, Internet Explore, Firefox and Mobile browsers show the highest degree of difference with respect to the input recommendations

in all datasets. The top-100 dataset shows the highest degree of difference, as expected since this dataset represents the optimal case for the **CARA** algorithm. The random-100 and whole datasets show lower, but still significant enough degree of change.

12.3.2 The number of changed recommendations

The number of changed recommendations is calculated as described in Section 11.8. Tables 12.1, 12.2 and 12.3 show the number of total, changed and unchanged recommendations for the three datasets. The changed recommendation results show that the majority of cases the **CARA** algorithm produces a re-ranking which is different from the input recommendation, meaning that in the majority of cases at least one valid click on a recommended item which is not ranked first is recorded. In addition to the **Kendall correlation** as discussed in the previous section the results in the degree of difference show that the top-100 dataset changes the most recommendations while the random-100 and whole datasets show more not changed recommendations. Additionally, it is remarkable that the ratio of changed recommendations of the Opera and Safari browsers having a lower share in the distribution resemble browsers with a high share such as Chrome. This can indicate that for recommendations having less interaction data equally changes are recorded in the data. In the next section this problem is elaborated further.

Feature	Total	Changed	Not changed
Opera	26	25	1
Other	31	31	0
Chrome	94	93	1
Mobile	86	85	1
Safari	62	61	1
Firefox	92	92	0
Internet Explorer	83	82	1

Table 12.1: Number of recommendations and changed recommendations per feature for the top-100 dataset.

Feature	Total	Changed	Not changed
Opera	4	4	0
Other	16	15	1
Chrome	66	65	1
Mobile	43	41	2
Safari	19	15	4
Firefox	57	53	4
Internet Explorer	37	37	0

Table 12.2: Number of recommendations and changed recommendations per feature for the random-100.

12.3.3 The ratio of improvement

The fifth column of the tables 12.4, 12.5 and 12.6 shows the mean, variance and standard deviation for the **ROI**. The **ROI** scores smaller than 10 are used in this analysis; The **ROI** scores are considered to be caused by anomalies which cause extreme skewness in the data. For example, suppose that a recommendation has 99 views from users having the browser Internet Explorer and 1 view and click from a user having Chrome as browser. Then the **Probability of Click (POC)** is $\frac{1}{100}$ and the **CARA**

Feature	Total	Changed	Not changed
Opera	310	294	16
Other	3215	3010	205
Chrome	9234	8868	366
Mobile	3724	3569	155
Safari	1509	1413	96
Firefox	6985	6721	264
Internet Explorer	4767	4546	221

Table 12.3: Number of recommendations and changed recommendations per feature for the whole dataset.

algorithm score is $\frac{1}{2}$. So then the ROI score is $\frac{0.5}{0.01} = 50$. Hence anomalies caused by automatic processes which massively visit the same set of study pages and consequently create a high number of view events.

Based on the recommendation interaction data and the result of the Kendall correlation it is expected to have the most potential improvement for the Chrome, Internet Explore, Firefox and Mobile browsers, especially in the top-100 dataset. However, the Safari, Opera and Other browsers are among the highest average ROI scores in all datasets. This is caused by the low data volumes as shown in the distribution of the data and the low interaction by the CCTR and COV. Therefore the sparse re-ranked items are boosted as potential improvements by the ROI metric. The POC is low in this recommendations, which results in boosting the improvement of the re-ranked recommendation. This is essentially the same problem as the skewness described in the previous paragraph, except that the cause is absence of click data in stead off anomalies. These results show that the ROI requires interaction data with the complete recommendation per contextual feature in order to obtain a fair insight in the actual potential improvement. Therefore, the average ROI scores of the Chrome, Internet Explorer, Firefox and Mobile browsers in the top-100 are considered to be the most fair results since the most interaction and data volumes are available. These results indicate potential improvement in the re-ranked recommendations. The standard deviation suggests that not in all cases a recommendation is improvement. For the Chrome, Internet Explorer, Firefox and Mobile browsers respectively 5.9%, 1.6%, 4.8% and 1.1% of the cases show a ROI score lower than 1 and therefore show expected regression. Hence it can be concluded that these browsers show a fair expectation of the potential improvement the re-ranked recommendation produced by the CARA algorithm can offer.

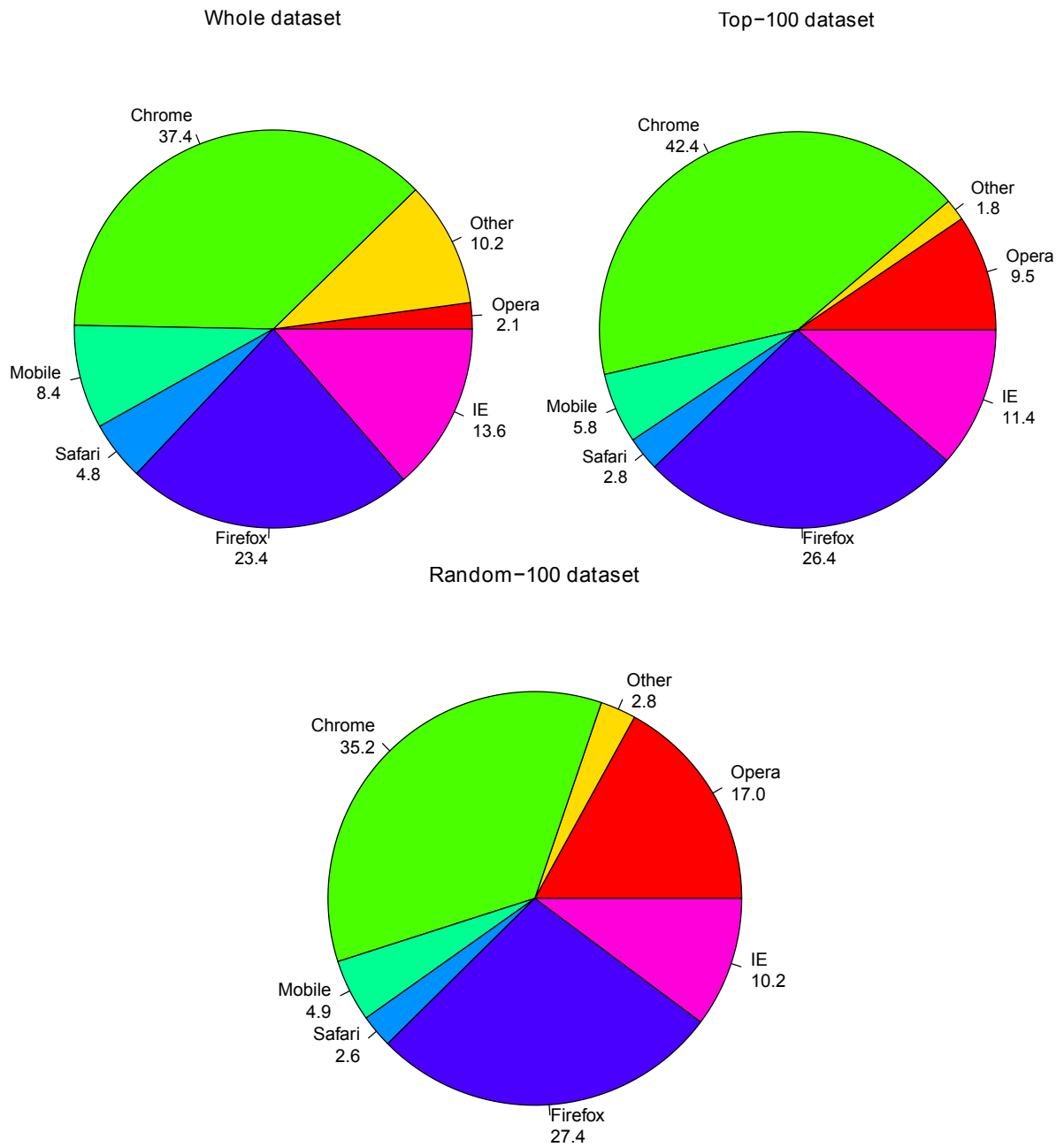


Figure 12.1: The distribution of the generated click and views data over the contextual elements for the whole, top-100 and random-100 datasets.

Feature	Contextual CTR	Contextual coverage	Kendall rank correlation			Ratio of improvement		
			$E(X)$	$V(X)$	σ	$E(X)$	$V(X)$	σ
Opera	4.0%	0.4%	0.49	0.09	0.30	4.29	5.77	2.40
Other	17.9%	0.3%	0.51	0.09	0.30	3.49	3.52	1.88
Chrome	23.1%	8.0%	0.01	0.12	0.34	1.49	0.54	0.73
Mobile	36.6%	1.8%	0.13	0.14	0.38	2.62	2.30	1.52
Safari	21.4%	0.5%	0.39	0.12	0.35	3.30	3.89	1.97
Firefox	17.7%	4.7%	0.03	0.11	0.33	1.54	0.53	0.73
Internet Explorer	30.2%	3.4%	0.11	0.14	0.37	2.13	1.43	1.20

Table 12.4: Contextual CTR and Coverage, the Kendall correlation and the Ratio of improvement for the top-100 dataset.

Feature	Contextual CTR	Contextual coverage	Kendall rank correlation			Ratio of improvement		
			$E(X)$	$V(X)$	σ	$E(X)$	$V(X)$	σ
Opera	4, 7%	0.9%	0.62	0.01	0.11	4, 06	3.60	1.90
Other	20.0%	0.5%	0.58	0.05	0.23	3.36	4.88	2.20
Chrome	14.5%	5.1%	0.24	0.14	0.37	2.19	2.37	1.54
Mobile	41.1%	1.6%	0.34	0.10	0.32	3.84	4.45	2.11
Safari	21.0%	0.5%	0.52	0.19	0.43	4.46	6.84	2.61
Firefox	15.9%	4.3%	0.34	0.17	0.41	2.57	2.56	1.60
Internet Explorer	24.2%	2.3%	0.31	0.14	0.38	2.85	2.92	1.70

Table 12.5: Contextual CTR and Coverage, the Kendall correlation and the Ratio of improvement for the random-100 dataset.

Feature	Contextual CTR	Contextual coverage	Kendall rank correlation			Ratio of improvement		
			$E(X)$	$V(X)$	σ	$E(X)$	$V(X)$	σ
Opera	13.8%	0.3%	0.59	0.08	0.28	4.83	4.86	2.20
Other	22.1%	2.4%	0.55	0.10	0.31	3.49	4.15	2.04
Chrome	22.8%	8.9%	0.41	0.14	0.37	2.70	2.40	1.54
Mobile	35.2%	2.8%	0.48	0.11	0.33	4.14	4.80	2.19
Safari	20.9%	1.1%	0.56	0.08	0.29	4.21	4.90	2.21
Firefox	36.0%	7.9%	0.41	0.14	0.37	3.20	3.73	1.93
Internet Explorer	37.6%	4.8%	0.46	0.13	0.36	3.76	4.40	2.10

Table 12.6: Contextual CTR and Coverage, the Kendall correlation and the Ratio of improvement for the whole dataset.

13.1 Results and contributions

An exploratory data analysis has been conducted in order to evaluate whether the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm has the potential to effectively re-ranks and improve the quality of existing recommendations. Furthermore, this experiment also contributes to the evaluation of the [Reference Architecture \(RA\)](#) as described in part I by enabling practical usage the session data processing task.

The effective re-ranking is evaluated with the [Kendall tau Rank Correlation Coefficient \(Kendall correlation\)](#) which measures the degree of difference between the input and re-ranked recommendation produces by the [CARA](#) algorithm. The results of the [Kendall correlation](#) shows that the algorithm is able to effectively re-rank the recommendations. It is shown that the re-ranked recommendation in the top-100 dataset show the highest degree of difference between the input and re-ranked recommendations, while the random-100 and whole datasets show lesser degrees of difference. This is as expected since the top-100 dataset represents the optimal case, while the random-100 and whole datasets represent average and overall improvement cases.

The potential quality improvement is evaluated with the [Ratio of Improvement \(ROI\)](#). The results show that browsers having less interaction show a higher expected improvement than browsers having high interaction data. This is caused by the low [Probability of Click \(POC\)](#) value of these browsers, which boost the improvement score. Hence, the most fair evaluation of the [ROI](#) is done with the top-100 dataset on the Chrome, Internet Explorer, Firefox and Mobile browsers since these browser show the most interaction. It is shown that the recommendations re-ranked based on these browsers show expected improvement. However, in order to evaluate the expected quality improvement objective the [ROI](#) metric must be improved in order to avoid boosting of recommendation having low interaction data.

Finally, evaluation of the potential improvement and quality of recommendations in an offline environment is hard since the interaction with the user is missing. Therefore, further experimentation with the [CARA](#) algorithm in the online environment is necessary in order to evaluate whether the re-ranked recommendations are perceived as an improvement by the user. Eventually, the users of the Masters Portal web system decide whether the displayed recommendation is helpful or not.

13.1.1 Contribution to the reference architecture evaluation

The session data processing task as described in Section 8.5 is used to extend the recommendation data with contextual features in this experiment. The practical usage of this processing task shows the required *adaptability* of the implementation, by the separation of processing functions and scripts. This allows implementation and testing of required changes in the processing task without using the HADOOP framework to execute the complete processing task. Furthermore, the practical usage in shared HADOOP environments shows the required *flexibility* in scheduling and resource allocation of processing tasks, showing that the required quality and effectiveness of the RA is achieved.

13.2 Limitations

13.2.1 Data volume

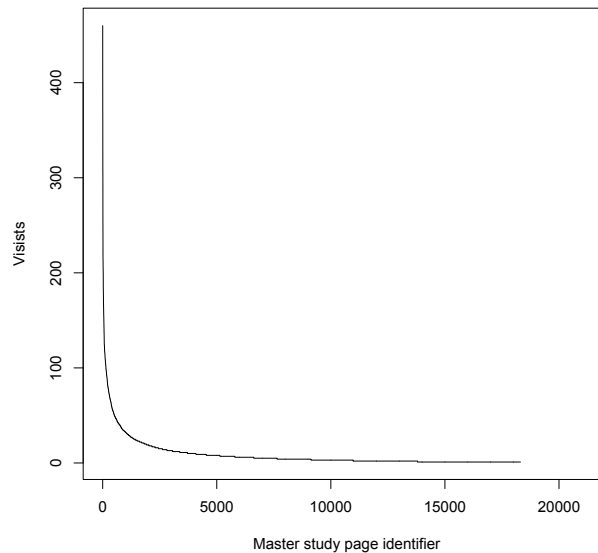


Figure 13.1: The number of visits on master study pages on the Masters Portal web system.

Figure 13.1 shows a plot of the visit numbers on master study pages on Masters Portal. The data shows that a small set of master study pages have high visitor rates while the remaining pages have average to low visitor rates. Consequently, the input data for this experiment shows the same long tail since this data is based on the visits of pages. As a result, 60% of the recommendations in the input data contain less than 10 clicks, which means that in these cases CARA algorithm cannot apply re-ranking to items of the input recommendation. Hence in most of the cases the re-ranking of the recommendations depends on the availability of a click of the recommended item rather than differences in click behavior. This is not the intention of the CARA algorithm and influences the evaluation of the results.

In order to address this issue the experiments can be performed using a higher data volume, for example using a quarter or a half year of log data. Although the data will still show a long tail, it is expected that the items in the tail will also have more recommendation click data which will improve the results obtained with the CARA algorithm.

Nevertheless, it must be investigated whether the data volume actually fetches more clicks on the recommendations as well. Currently each master study page has its own recommendation and not

every master study page generates a sufficient amount of traffic. Hence an alternative is to only apply the re-ranking on recommendations that have sufficient clicks recorded, for example require that every recommended item has at least one recorded click.

Another alternative is to aggregate click data. Recorded clicks can be aggregated per discipline in order to obtain the interaction behavior of users visiting master study pages in a particular discipline for example. A disadvantage is that the click data then represents a general interaction with the recommendations, which could miss specific preferences of user groups in particular recommendations.

13.2.2 Data impurity

The evaluation metrics in the experiment are sensitive to impurities in the data. Anomalies in the data can boost the [ROI](#). The [Kendall correlation](#) is affected as well, however the effects is less visible because this metrics considers the re-ranking of the recommended items and does not take rating scores into account. Improved anomaly detection process can be used to map out the amount of fraudulent visit data. This data needs to be excluded in the input data sources for the [CARA](#) algorithm.

Furthermore, Masters Portal derives a set of around thirty recommendations for each study program. Only the top then of these recommendations are visible. In the current setup it can be the case that a non-visible recommendation is re-ranked and suddenly is visible in the ranking. This is an issue depending whether the focus remains on re-ranking the visible part of the recommendation.

Finally the current bot filtering process only filters the top-28 known bots. It can be observed in the data that there are still entries of bots, spiders and other automatic processes in the data sources. Contextual parameters can often not be resolved completely which causes erroneous results. A better bot filtering process can exclude more bots and eventually improve the resulting re-ranked recommendations of the [CARA](#) algorithm. This process can also be considered as an implementation a bot filtering strategy in the [RA](#) and support the evaluation of addressing bot filtering.

13.2.3 The ratio of improvement

The [ROI](#) measure should incorporate the number of re-ranked items and the rank improvement of these items in order to normalize the boosting of recommendation having low interaction data and have a better estimate of the actual improvement, currently visible in all datasets. For example, when only one item of a recommendation is re-ranked the [ROI](#) for this item is high because this item is automatically ranked first. However the actual improvement of the recommendation is small because the remaining recommended items are not re-ranked at all. Hence incorporation of the number of re-ranked items along with rank improvement will provide a better insight in the actual expected quality the [CARA](#) algorithm can provide per recommendation.

13.2.4 Combining of data sources

In some cases the synchronization of the timestamps between the session and the clicks and impressions datasets is not synchronized due to delays in the processing of the logging of these events. This causes that a small part of the actual input data is not joined in the processing task as described in [Section 11.7](#) and hence not taken into account in the remaining processing. The expectation is that the addition of the missing data does not cause dramatic changes in the resulting re-ranking and metrics of the [CARA](#) algorithm.

13.2.5 The visibility function

The used visibility function approximates the visibility of the recommendations on the master study page of Masters Portal much worse than it actually is. If the master study page is visited having a 1024×768 screen resolution then the first four recommendations are fully visible. When the user wants to see the first paragraphs of content of the page the remaining recommendations will become visible while the first recommendations become not visible anymore. When this screen resolution is used as a lower bound, the actual visibility function of the masters study pages of Masters Portal should take into account the probability that the user wants to see the content of the page, the probability that the user immediately clicks on one of the top-4 recommended items and the probability that the user is scrolling up and down the page to see the content.

The visibility function has a high influence on the resulting re-ranking in case of multiple re-ranked items. Then the visibility function can change the rank of the items produced by the [CARA](#) algorithm. Therefore, when the experiment is performed with a visibility function which is more tuned towards the actual visibility of the recommendations, it is expected that the visibility function will not boost the scores of the re-ranked items as in the current situation. It is expected that there will be less improvement than displayed in the current results.

13.3 Future work

This chapter contains future work on the experiment. Additionally, [Appendix B](#) outlines further experimentation on the online environment of Masters Portal.

13.3.1 Data volume

As described in [Section 13.2.1](#) in some cases the data is too sparse to obtain a usable re-ranking of a recommendation, let alone obtain a complete re-ranking. The [CARA](#) algorithm is most effective when the data contains at least one click per recommended item and when a difference exists in clicks between preferred and less popular item. A data volume spanning multiple months can help to accommodate this sparsity. Nevertheless, it can be the case that seasonal popularity of recommended items either is dampened. For example, when students having the same geographical location only visit master study pages having discipline A in one month and visit pages from other disciplines in the following two months the preferences for discipline A is less clear in the complete dataset. So the choice data volume depends on experience with this experiment, information about the usage of Masters Portal and preferences to incorporate popularity information.

13.3.2 Evaluation of alternative contextual features

Contextual category	Available features
Locational	Continent , country, city
Device	Device, device version, OS , OS version, browser, browser version
Temporal	Date, time, timezone, time of day

Table 13.1: The available contextual features. The bold features are proposed to be used in experiments with the [CARA](#) algorithm.

In the [CARA](#) algorithm any desired contextual feature can be used. In the case of Masters Portal there are also other contextual features like operating system (OS), continent and time of day which have equal distribution over users and are able to capture preferences of particular user groups. These proposed features could obtain better results in the re-ranking of the recommendations. Other potential contextual features are displayed in [table 13.1](#).

It is also possible to obtain multidimensional contextual features by combining contextual features. For example combining the contextual features time of the day and country to be able to catch preference differences on temporal and locational level. A multidimensional feature will contain more contextual categories and therefore provide more separation of users. However, the data volume should be high enough to be able to derive usable recommendations.

13.3.3 Evaluation of hybridizations

In order to be able to reach the objective of showing that the [CARA](#) algorithm can improve the quality of recommender systems, it is required to do apply the algorithm in several recommendation deriving cases. Hybridizations of recommendation algorithms in combination with the [CARA](#) algorithm have not been examined in this experiment. While hybridization of recommender systems often only consider the hybridization between content-based and collaborative algorithms [[AT05](#)], nowadays hybridizations include every possible recommender system for example rule-based [[AT11](#)], demographic, utility-based and knowledge-based [[Bur02](#)]. Further experiments with hybridizations can adhere to the experimentation outline as described in [Appendix B](#).

Masters Portal screen shots

This Appendix contains the screen shots of the pages available on the Masters Portal web system.

Browse by country ▶ About us ▶

Start your search for Masters here!

Search the largest database in Europe for Masters!

Any discipline and/or Try: Business administration

Any subdiscipline

Find & compare 22,052 Masters in Europe!

[Home](#)

GET PERSONAL UPDATES

Help us improve StudyPortals
Give us your feedback and win an Amazon voucher (€50 or \$50)
[Click here](#)

Most Popular Studies

Business & Economics

- ▶ [Business Administration](#)
- ▶ [Entrepreneurship](#)
- ▶ [Finance](#)

Programmes in business and economics provide students with theoretical and practical knowledge of various business, economics, finance, management, accounting and marketing subjects.

[All Masters in Business & Economics ▶](#)

Law

Universities in the spotlight

Bilkent University
Ankara, [Turkey](#)
Bilkent University is recognized and ranked worldwide as the premier institution of higher education in Turkey. Located in the countrys...
[More information ▶](#)

Kingston University - Distance Learning
London, [United Kingdom](#)
Kingston University London stands up for its reputation as a leading centre for postgraduate centre by providing a stimulating and...
[More information ▶](#)

LSE THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE
London, [United Kingdom](#)
The London School of Economics and Political

Figure A.1: The index page of the Masters Portal web system. The search bar is located on the top of the page, while recommended studies and universities are displayed on the center of the page. Not visible in this picture is the footer of the page containing additional information such as contact and support information.

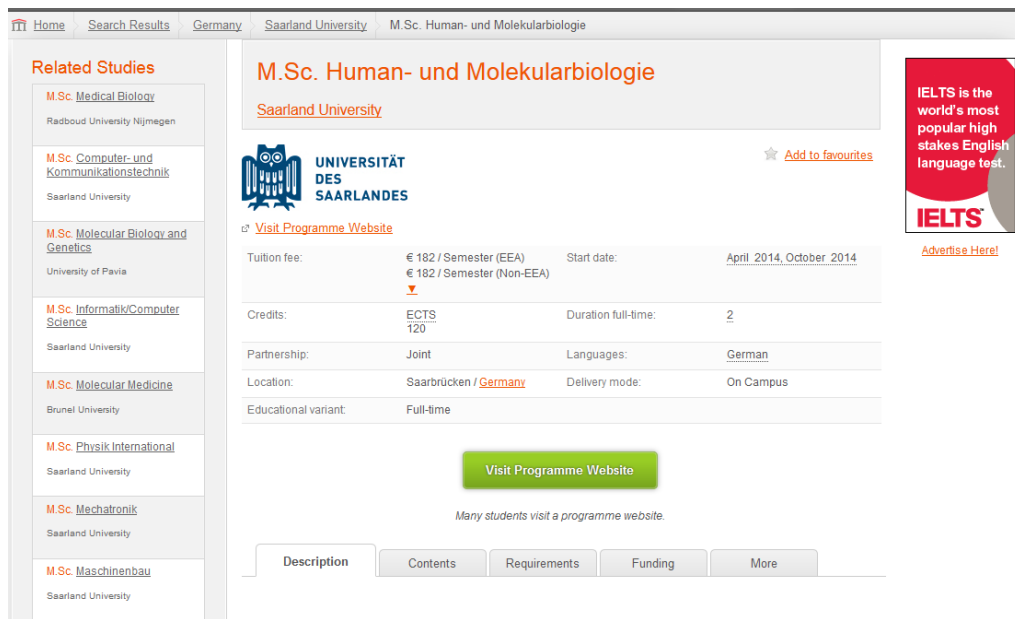


Figure A.2: An example of the master study page. The related studies recommendation recommends master studies related to the university or discipline.

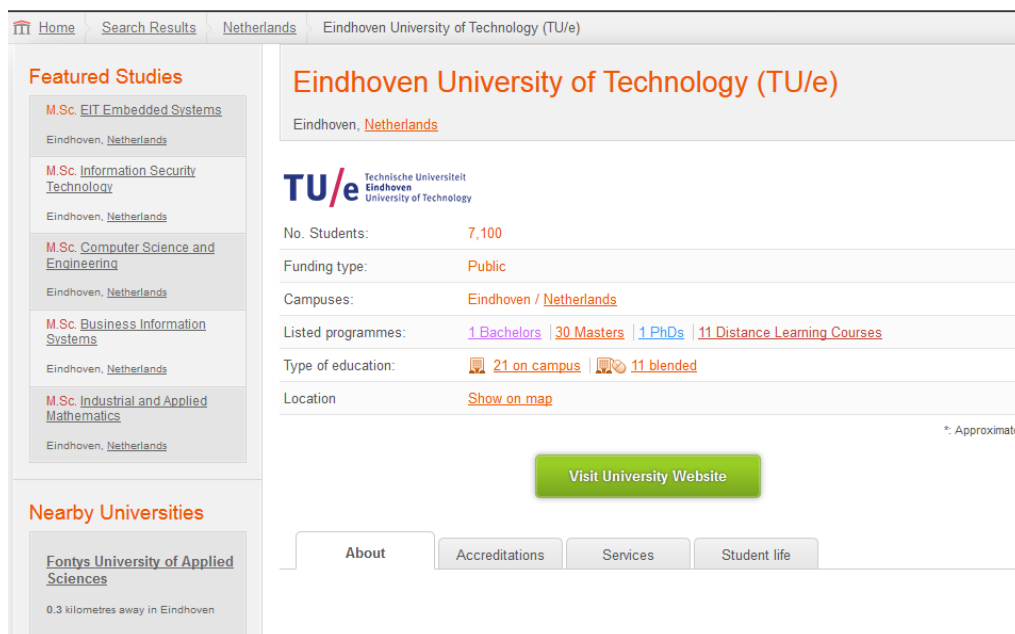



Figure A.3: An example of the university page. Two recommendations are available: The featured studies promotes master studies available on the universities, while the nearby universities nearby located universities.

Home Greater Europe Netherlands

Masters in Netherlands

[More countries](#)

[Master of Science \(M.Sc.\): 717 studies](#)
[Master of Arts \(M.A.\): 238 studies](#)
[Master of Business Administration \(MBA\): 35 studies](#)
[Master of Law \(LL.M.\): 47 studies](#)



Population:	16,730,000	# of Students:	656,000*
# of Int. Students:	82,000*	# of Institutes:	77
Education Expenditure:	59% of GDP	Academic Year:	Runs from September to July

* Approx. total

[Advertise Here!](#)

[Visit Website](#)
[View all Masters](#)

[Study in Netherlands](#)
[Research & Career](#)
[Living](#)
[About](#)
[Institutes](#)






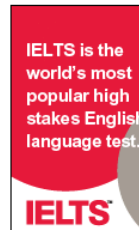
Figure A.4: An example of the country page. The recommendation shows the available master studies per degree.

View all Masters in:

- [Aerospace, Aeronautical & Marine Engineering](#)
- [Applied Mathematics](#)
- [Bio & Biomedical Engineering](#)
- [Chemical Engineering](#)
- [Civil Engineering, Architecture & Construction](#)
- [Computer Science & IT](#)
- [Electrical Engineering](#)
- [Electronics & Embedded Technology](#)
- [Energy Engineering](#)
- [Engineering & Business](#)
- [Engineering Physics](#)
- [Environmental Engineering](#)
- [Industrial Design](#)
- [Materials Engineering](#)
- [Mechanical Engineering](#)

[View all Masters in Computer Science & IT](#)

Scholarships
Erasmus
Mundus



More information about the Computer Science & IT discipline

Computer science deals information about computation, software, hardware and computer systems. The field consists of theoretical and practical approaches. Both of them contribute to research of the main important issues that computer science investigates: IT, which deals with computer and other systems, its development, artificial intelligence and mathematics, software engineering, systems and network engineering. Graduates can work in fields related to: manufacturing of computer and information systems, maintenance and sales of them as well as software and hardware developers, e-business and telecommunication networks experts.

Suggested Studies




<p>Master in Business Information Systems</p>  <p>Information systems play an important role in the design, control, support and improvement of business processes. These systems...</p> <p>Detailed Description ▶</p>	<p>Master in Information Technology...</p>  <p>With Walden University's M.S. in Information Technology, you can expand your knowledge of technology while gaining the skills to...</p> <p>Detailed Description ▶</p>	<p>Master in Computer Science and...</p>  <p>There is a huge demand for efficient and reliable software systems. The complexity of many of these systems is an important...</p> <p>Detailed Description ▶</p>
--	--	--

Figure A.5: A partial example of the discipline page. Besides links to other disciplines and the discipline description, master studies are recommended in the lower part of the page.

Home Search Results

Refine your search

Remember this search

Keywords

computer science Filter

Discipline

Any Discipline Filter

Any subdiscipline Filter

Language of instruction

Any language Filter

Location

Any country Filter

Show all countries Sort by most studies first Sort alphabetically

Europe

United Kingdom (355) Switzerland (64) Ireland (35) Poland (24)
 Germany (90) Sweden (59) Finland (31)
 Netherlands (77) France (54) Austria (25)

World

Australia (4) United States (2)

1,007 Masters. Showing 1 to 10

Show all studies from:

- Department of Computer Science, University of Liverpool
- EPITA Graduate School of Computer Science
- Warsaw School of Computer Science

Remember this search

Sort by: Relevance 1 2 3 4 Next ... Last

M.Sc. Computer Science ★

University of Twente (UT) Enschede, Netherlands EN ON CAMPUS

Start date: Aug 2014 Tuition Fee: EEA € 1,835 per year Duration: 2 Years (120 ECTS)

UNIVERSITY OF TWENTE.

Information and communication technology is omnipresent in modern society. The latest hardware and software developments are crucial in domains such as telecom, health, energy and information systems.

Detailed Description ▶

"I think IELTS is the best in the world"

Advertise Here!

Figure A.6: A partial example of the search page. The search refine options are located on the left side. The top of the figure shows country level refinement and sorting options. A search result is visible at the bottom of the figure.

Further experimentation

An outline for further experimentations has been part of the initial experiment proposal. The described experiments are performed on the online environment of Masters Portal. These experiments are necessary in order to evaluate how the users value the re-ranked recommendations. Together with proposed algorithms and baselines the experiments outline continues and completes the [Context-Aware Recommendation Adjustment \(CARA\)](#) algorithm experiment.

B.1 The algorithms

The algorithms used are displayed in Table B.1. Algorithms 1 and 2 are common recommendation algorithms as described in Section 2.3.3. Apache Mahout offers out of the box processing tasks for these algorithms. Algorithm 4 is the [CARA](#) algorithm applied on Algorithms 2 and 3, which is described in Section 2.5.2.

Algorithm 3 is a random recommendation generator. A random permutation of recommended items is generated to serve as a recommendation. This algorithm is used as a baseline for all recommender systems in this experiment.

Algorithm 5 is a random re-ranking algorithm. It takes an existing recommendation and produces a random permutation. This algorithm is used as a baseline for the recommender systems with context adjustment to validate that the [CARA](#) algorithm improves quality.

#	Type	Algorithm
1	Ranking model	Content-based recommendation
2	Ranking model	Collaborative filtering
3	Ranking model	Random recommendation
4	Re-ranking model	Context-aware recommendation adjustment
5	Re-ranking model	Random re-ranking

Table B.1: The various algorithms and combinations with their type.

B.2 Experiment 1: Applying re-ranking on collected visit data

In Experiment 1 the **CARA** algorithm is applied on the online collected usage data from all recommender systems, having the same setup as Experiment 1. As input for this experiment the previously collected click data is used. The contextual feature used in the **CARA** algorithm can be chosen based on an investigation of the distribution of contextual features among the data. It is expected that in half of the cases a different ranking is produced.

When the context adjustment shows a significant amount of re-ranking then the recommendation data is prepared for the next online data collecting phase. The random re-ranking of Algorithm 4 is then applied on the recommendation data.

B.3 Experiment 2: Evaluating the re-ranked recommendations in the online environment

In Experiment 2 the quality of all recommender systems is evaluated. This is done to get insight which recommender systems have the best performance. The goal is to validate whether the recommender systems with context adjustment perform better than the random recommender, random re-ranking and the recommender systems without adjustment.

Hypothesis 1 *Applying adjustment on the content-based recommender system improves the quality of the recommendation.*

H_0 : The **CARA** algorithm applied on the content-based recommender yields a higher CTR value than the content-based recommender without adjustment.

H_1 : The **CARA** algorithm applied on the content-based recommender does not yield a higher CTR value than the content-based recommender without adjustment.

Hypothesis 2 *Applying adjustment on the collaborative filtering recommender system improves the quality of the recommendation.*

H_0 : The **CARA** algorithm applied on the collaborative filtering recommender yields a higher CTR value than the collaborative filtering recommender without adjustment.

H_1 : The **CARA** algorithm applied on the collaborative filtering recommender does not yield a higher CTR value than the collaborative filtering recommender without adjustment.

Hypothesis 3 *Applying adjustment on the history recommender system improves the quality of the recommendation.*

H_0 : The **CARA** algorithm applied on the history recommender yields a higher CTR value than the history recommender without adjustment.

H_1 : The **CARA** algorithm applied on the history recommender does not yield a higher CTR value than the history recommender without adjustment.

Hypothesis 4 *The random recommendation has the worst performance.*

H_0 : The random recommendation with and without the application of the **CARA** algorithm yields lower CTR values than the other recommender systems.

H_1 : The random recommendation without application of the **CARA** algorithm yields lower CTR values than the other recommender systems, while the random recommendation with adjustment yields a higher CTR than at least one other recommender system.

H_2 : The random recommendation with and without application of the **CARA** algorithm yields higher CTR values than at least one other recommender system.

Hypothesis 5 *Random re-ranking does not improve the performance of a recommender.*

H_0 : A recommender system with random re-ranking applied yields a higher CTR value than the recommender without random re-ranking

H_1 : A recommender system without random re-ranking applied yields a higher CTR value than the recommender with random re-ranking

It is expected that the recommender systems where the **CARA** algorithm is applied will have a higher CTR value than the recommender without adjustment, the random recommendation and the random re-ranking.

Bibliography

- [AB13] Xavie Amatriain and Justin Basilico. System architectures for personalization and recommendation. <http://techblog.netflix.com/2013/03/system-architectures-for.html>, March 2013.
- [ABL09] Sofiane Abbar, Mokrane Bouzeghoub, and Stéphane Lopez. Context-aware recommender systems: A service-oriented approach. In *VLDB PersDB workshop*, pages 1–6, 2009.
- [ADB⁺99] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.
- [And06] Chris Anderson. *The long tail: Why the future of business is selling more for less*. Hyperion, 2006.
- [ASST05] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems (TOIS)*, 23(1):103–145, 2005.
- [AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.
- [AT11] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.
- [BKR08] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction*, 18(3):245–286, 2008.
- [BOHG13] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 2013.
- [BR09] Linas Baltrunas and Francesco Ricci. Context-dependent items generation in collaborative filtering. In *Proceedings of the 2009 Workshop on Context-Aware Recommender Systems*, pages 22–25, 2009.

- [Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.
- [CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.
- [CK⁺00] Guanling Chen, David Kotz, et al. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.
- [CKK02] Yoon Ho Cho, Jae Kyeong Kim, and Soung Hie Kim. A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, 23(3):329–342, 2002.
- [DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [Fou13] The Apache Software Foundation. Log files documentation v2.2. <http://httpd.apache.org/docs/2.4/logs.html>, july 2013.
- [GA11] Matthias Galster and Paris Avgeriou. Empirically-grounded reference architectures: a proposal. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pages 153–158. ACM, 2011.
- [HLGZ12] Tim Hussein, Timm Linder, Werner Gaulke, and Jürgen Ziegler. Hybreed: A software framework for developing context-aware hybrid recommender systems. *User Modeling and User-Adapted Interaction*, pages 1–54, 2012.
- [JRMG06] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396. ACM, 2006.
- [Ken38] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.
- [Koy00] Ivan Koychev. Gradual forgetting for adaptation to concept drift. Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning,, 2000.
- [Max13] MaxMind. Geopip databases and web services. http://www.maxmind.com/en/geolocation_landing, December 2013.
- [MBN⁺13] Luís Marujo, Miguel Bugalho, João Paulo da Silva Neto, Anatole Gershman, and Jaime Carbonell. Hourly traffic prediction of news stories. *arXiv preprint arXiv:1306.4608*, 2013.
- [MCS00] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43(8):142–151, 2000.
- [MDG⁺09] Mark Meiss, John Duncan, Bruno Gonçalves, José J Ramasco, and Filippo Menczer. What’s in a session: tracking individual behavior on the web. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 173–182. ACM, 2009.
- [MSDR04] Stuart E Middleton, Nigel R Shadbolt, and David C De Roure. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–88, 2004.
- [Ols09] Arthur Datvid Olson. Sources for time zone and daylight saving time data. <http://cs.ucla.edu/~eggert/tz/tz-link.htm>, may 2009.

- [PGP13] Santiago Pericas-Geertsen and Marek Potociar. Jax-rs: Java api for restful web services. http://www.maxmind.com/en/geolocation_landing, May 2013.
- [SAT⁺99] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *Handheld and ubiquitous computing*, pages 89–101. Springer, 1999.
- [SAW94] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.
- [SH10] Gabor Szabo and Bernardo A Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.
- [SKR99] J Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.
- [Ste11] Harald Steck. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 125–132. ACM, 2011.
- [SWSY06] Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The Knowledge Engineering Review*, 21(2):97–126, 2006.
- [TC00] Thomas Tran and Robin Cohen. Hybrid recommender systems for electronic commerce. In *Proc. Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, Technical Report WS-00-04, AAAI Press*, 2000.
- [VMO⁺11] Katrien Verbert, Nikos Manouselis, Xavier Ochoa, Martin Wolpers, Hendrik Drachsler, Ivana Bosnic, and Erik Duval. Context-aware recommender systems for learning: a survey and future challenges. 2011.
- [Wis12] Daan Wissing. Design and implementation of a business intelligence system for website visitors statistics, 2012.
- [Zli09] Indre Zliobaite. Learning under concept drift: an overview. Technical report, Overview, Technical report, Vilnius University, 2009 techniques, related areas, applications Subjects: Artificial Intelligence, 2009.