Mobile application usage prediction through context-based learning

Author(s): Leroux, P (Leroux, Philip)[1] ; Roobroeck, K (Roobroeck, Klaas)[1] ; Dhoedt, B (Dhoedt, Bart)[1] ; Demeester, P (Demeester, Piet)[1] ; De Turck, F (De Turck, Filip)[1]

Source: JOURNAL OF AMBIENT INTELLIGENCE AND SMART ENVIRONMENTS **Volume:** 5 **Issue:** 2 **Pages:** 213-235 **DOI:** 10.3233/AIS-130199 **Published:** 2013

Abstract: The purchase and download of new applications on all types of smartphones and tablet computers has become increasingly popular. On each mobile device, many applications are installed, often resulting in crowded icon-based interfaces. In this paper, we present a framework for the prediction of a user's future mobile application usage behavior. On the mobile device, the framework continuously monitors the user's previous use of applications together with several context parameters such as speed and location. Based on the retrieved information, the framework automatically deduces application usage patterns. These patterns define a correlation between a used application and the monitored context information or between different applications. Furthermore, by combining several context parameters, context profiles are automatically generated. These profiles typically match with real life situations such as 'at home' or 'on the train' and are used to delimit the number of possible patterns, increasing both the positive prediction rate and the scalability of the system. A concept demonstrator for Android OS was developed and the implemented algorithms were evaluated in a detailed simulation setup. It is shown that the developed algorithms perform very well with a true positive rate of up to 90% for the considered evaluation scenarios.

Accession Number: WOS:000317100800005

Document Type: Article

Language: English

Author Keywords: Context modeling; context prediction; mobile application development

KeyWords Plus: ACTIVITY RECOGNITION; RECOMMENDATIONS; ALGORITHM; SEMANTICS

Reprint Address: Leroux, P (reprint author)
Univ Ghent, Dept Informat Technol, IMinds, Gaston Crommenlaan 8-201, B-9050 Ghent, Belgium.

Addresses:
[1] Univ Ghent, Dept Informat Technol, IMinds, B-9050 Ghent, Belgium

E-mail Addresses: Philip.Leroux@intec.ugent.be; Klaas.Roobroeck@intec.ugent.be;

Bart.Dhoedt@intec.ugent.be; Piet.Demeester@intec.ugent.be; Filip.DeTurck@intec.ugent.be

Publisher: IOS PRESS, NIEUWE HEMWEG 6B, 1013 BG AMSTERDAM, NETHERLANDS

Web of Science Categories: Computer Science, Artificial Intelligence; Computer Science, Information Systems; Telecommunications

Research Areas: Computer Science; Telecommunications

IDS Number: 119MF

ISSN: 1876-1364

Journal of Ambient Intelligence and Smart Environments 0 (0) 1
IOS Press

Mobile Application Usage Prediction through Context-based Learning

Philip Leroux [a,*] Klaas Roobroeck [a] Bart Dhoedt [a] Piet Demeester [a] and Filip De Turck [a]
[a] *Ghent University - IBBT - Department of Information Technology, Gaston Crommenlaan 8/201, 9050 Ghent, Belgium*
E-mail: Philip.Leroux, Klaas.Roobroeck, Bart.Dhoedt, Piet.Demeester, Filip.DeTurck@intec.ugent.be

Abstract. The purchase and download of new applications on all types of smartphones and tablet computers has become increasingly popular. On each mobile device, many applications are installed, often resulting in crowded icon-based interfaces. In this paper, we present a framework for the prediction of a user's future mobile application usage behavior. On the mobile device, the framework continuously monitors the user's previous use of applications together with several context parameters such as speed and location. Based on the retrieved information, the framework automatically deduces application usage patterns. These patterns define a correlation between a used application and the monitored context information or between different applications. Furthermore, by combining several context parameters, context profiles are automatically generated. These profiles typically match with real life situations such as 'at home' or 'on the train' and are used to delimit the number of possible patterns, increasing both the positive prediction rate and the scalability of the system. A concept demonstrator for Android OS was developed and the implemented algorithms were evaluated in a detailed simulation setup. It is shown that the developed algorithms perform very well with a true positive rate of up to 90% for the considered evaluation scenarios.

Keywords: context modeling, context prediction, mobile application development

1. Introduction

1.1. Mobile Services: Situation

On many currently popular smartphone and tablet computer platforms, the continuous selling of mobile applications has shown to be a major (commercial) success. Mobile application markets are flooded with hundreds of thousands of mobile applications, often with limited free and more advanced commercial versions in all types of categories. With users installing all kinds of free and paid applications on their mobile phone, the icon-based interfaces, currently applied on many popular mobile operating systems, are often composed of several window containers loaded with application icons. However, the usage of many of these applications may be specifically tied to specific context parameters. For example, an employee may use more work-related applications (e.g. Email, Agenda or QuickOffice) during working hours while in the evening he (or she) might use more *fun*-related applications such as Youtube, Music or Facebook. While driving, a user might often consult the GPS application and for instance just before going to bed he might set his alarm clock. All these patterns are related to context parameters such as traveling speed, location or time. There may also exist patterns between applications such as frequently consulting the agenda after a phone conversation.

With the static user interfaces that are currently implemented on many mobile platforms, the most advanced type of implemented personalization with respect to mobile application usage is the integration of one or more home screens containing the user's favorite applications and widgets. These home screens are configured by the user and remain static irrespective of the user's current context or situational environment. A more efficient way of home screen interaction

*Corresponding author. E-mail: Philip.Leroux@intec.ugent.be

would be to dynamically adapt some elements of the home screen based on the user's current context situation. In this paper, the design of algorithms for the automatic detection of only those applications that are most likely to be used, based on the user's current context, is studied and evaluated in detail.

1.2. Contributions

In this paper, a framework is proposed for the automatic detection and prediction of mobile application usage behavior patterns by taking different context parameters into account. The detection of the usage patterns is performed automatically, i.e. without the requirement of explicit user input. Moreover, the algorithms can be applied for different types of users while no predefined types of context situations or fixed rules are defined. We will show that a 3 phase approach is most suited, this approach consists of the following steps: (i) Expectation-Maximization clustering on (a combination of) the different context parameters that define a user situation: time, location, speed and day of the week together with used applications to learn from previous user behavior and to generate different context situation clusters or context profiles; (ii) The detection of application usage patterns supporting context profile dependent patterns through graph analysis, popular application patterns by using a short time window and intra-cluster correlation patterns by using association rules. Intra-cluster patterns are usage patterns that may exist between different applications or between applications and very specific (individual) context parameters, and not to context profiles which are defined by the total context situation; (iii) Generation of a combined recommendation by bundling the information of all pattern types. A prototype application has been implemented running on an Android mobile phone. This application is composed of a data gathering service and a mobile widget application showing the top six recommended applications on the home screen of the user. For performance reasons, the implementation of the algorithm-specific modules was offloaded to a server-side component. This prototype and the generated recommendations are also extensively evaluated by means of two different scenarios A and B, respectively containing 17 and 26 different mobile applications. For scenario A, the framework generated a true positive rate of up to 90% when the top six ranked applications were taken into account. For scenario B a maximum true positive rate of 78.89% was found for similar conditions.

1.3. Related Work

A lot of research has already been focusing on the detection and representation of and the reasoning about the context or activity of users in a mobile or smart (home) environment. For instance, within the domain of activity detection, solutions have been proposing the usage of for example video data [21], different types of sensor data, such as door sensors and light switches [7], wireless sensors [25], RFID tags [20], accelerometer data [10], etc. In addition, several mobile context-aware service platforms have been designed in order to simplify the development of context-aware mobile applications [19,5,14,6]. For instance, the Reconfigurable context-sensitive middleware (RCSM) [26] is a middleware framework that facilitates the development of mobile applications that require context-sensitive ad hoc communication. Based on a context-sensitive interface description, compilers build application-specific adaptive object containers that allow for runtime context data acquisition, monitoring, and detection. Our research differs from these mobile context service architectures as the goal is not to build a service platform able to provide context-awareness to mobile applications, but to predict the use of such applications based on the available context. Although a feasible option would be to offload the context gathering and interpretation algorithms to one of the presented service platforms, no platforms were found that monitor the actual use of mobile applications.

For the representation of and the reasoning about context data, many solutions [4] propose the application of ontology languages as they are well-suited to represent a formal context model that can be shared, reused and extended for specific use case scenarios or domains. In addition, semantic reasoners allow for the reasoning about the different context models and user preferences described by these ontology languages. For example, the European FP7 research project m:Ciudad demonstrates [9] an approach to integrate contextual information with other search attributes to enable efficient service retrieval and recommendation in mobile user and application scenarios. Ontologies are used to describe services, users, situations and their domain knowledge. The ontology-based data representations are then used as the basis for search and proactive recommendations. Several other semantics-based frameworks for both context capturing and reasoning are evaluated in [4].

The work described in this paper differs from these semantic approaches as its goal is to automatically detect or predict patterns in the user's behavior based on the user's raw context data without the need of semantic interpretation of or reasoning about specific context situations. Such an approach has the advantage that it does not require the in advance semantic definition of context situations and rules and thus can be automatically applied in any situation and for all types of users, even without the need of user interaction during the learning phase. A similar approach [11] combines K-Means clustering [16] and Self-Organizing Maps [17] for unsupervised clustering of mobile context data. Naive Bayesian networks have been applied to classify the contexts of a mobile device user in his normal daily activities [18]. In this paper, the context information was derived mainly from audio features. In order to support media recommendation, adaptation and delivery for smartphones a context-aware recommendation platform for mobile devices is described [28] which uses an N x M-dimensional model and a hybrid processing approach. Many of the techniques that were proposed in these papers have been evaluated for our application domain. As will be stated in Section 2 a combination of other techniques and new algorithms was required for our use case scenario.

It should be noted that both the semantic and the machine learning approaches should not always be seen as an if/if scenario, as they may also be combined with each other within a general context framework. Such a framework should apply ontologies and semantic reasoning as a basis for the overall context interpretation. Specific techniques and algorithms are then used for automatic pattern deduction and pattern matching for very specific use case scenarios. An example of a framework encompassing both the interpretation of raw sensor data and the translation to a semantic model has already been presented [27].

Within the domain of context-based application usage prediction, a context-aware user modeling framework is described [23] that retrieves user-related information from a desktop environment to learn a user's preferences for different application actions. In order to generate predictions, a combination of a learning classifier system and a decision tree learner were used. For each new supported application a new decision tree was implemented. As one of the main functional requirements of our system is not to make presumptions about certain user behavior, context situations or used application such an approach was not feasible for our framework.

1.4. Paper Structure

This paper describes the implementation of a framework that is able to predict a user's future use of mobile applications on a mobile device. In order to generate such predictions, certain patterns should be found in the user's application usage. These patterns can be based on two types of information: (i) (physical) environment parameters that define a certain context situation or profile during which an application is used; (ii) actions that the user performs on the mobile device. In this paper, both types of patterns are integrated and combined. Therefore, a three-step sequential process flow was designed:

1. Processing of the monitored data in order to provide more structure to the data and the discovery of user context situations.
2. The parallel detection of different types of patterns that model the user's application use.
3. Evaluation of the different pattern prediction values in order to generate a final prediction.

The first step is described in Section 2. Section 3 provides a more detailed overview of the process flow and describes the different patterns that are supported in step 2 and the classification process of step 3. Implementation details of our framework are provided in Section 4. This section also describes several performance measurements. Section 5 describes our evaluation setup and methodology and Section 6 presents the obtained evaluation results. Finally, Section 7 states our conclusions.

2. Data Interpretation and Context Profile Creation

In this section, the first step of our three-step process flow, as introduced in Section 1.4 and described in more detail in Section 3, is explained. The goal of this first step is twofold:

- To discover structure in the monitored data (i.e. context data and application usage data). The output of this process can be used as input for the application pattern algorithms of step 2.
- To detect user context situations based on the monitored context data. The detected context situations, in this paper labeled as 'Profiles', are then used to limit the scope of certain application patterns.

For both objectives, the concept of context is introduced. A definition of context often cited in literature comes from Dey [8], a pioneer in the research on context-aware services and architectures: 'Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.' In this paper when referring to context we specifically target the context of the user, measured by a predefined set of four parameters: time, location, day of week and speed. Contrary to the definition of Dey, the user's application usage was deliberately not included in the set of context parameters from which a context profile may be deduced. This approach enables to limit the scope of certain application patterns to the currently active context profile. This is described in Section 3. For both objectives, the handling of context data is described in Section 2.2. Note that in the first objective, also monitored application usage data are processed. This is defined as 'Application Data Clustering' and is detailed in Section 2.3.

2.1. Expectation-Maximization (EM) Clustering

In order to find sets of relating context parameter values or a profile defined by a combination of value sets taken from each context parameter, a technique should be applied that is able to detect structure in the unstructured context data that were gathered. One of the main challenges is the continuous nature of several context parameters such as time and location, as the location is indicated by its geographic coordinates. This requires a technique that is able to match or group *similar* but not equal values while the exact or numerical definition of *similar* is not known in advance. As there is no previous knowledge about the data, an unsupervised learning technique is required. Based on the related work, as described in the Related Work section, several possible techniques were selected: Competitive Networks [22], Adaptive Resonance Theory (ART) models [13,24] and several clustering algorithms [15] such as hierarchical, k-means and Expectation Maximization (EM) clustering.

An extra criterion that was imposed was that all types of users and scenarios should be supported without making any presumptions about the context data. In addition, neither predefined rules nor training of the system based on generic user profiles should be allowed. Due to this restriction only EM clustering and ART models remained, as only those two techniques (out of the ones selected) do not require to set a fixed number of output nodes or clusters. Finally EM was chosen as no suitable conversion method was found to convert the different input data to a one-dimensional vector of binary values, as required by the ART base network (ART1). Furthermore, E.M. supports a Gaussian function as underlying distribution function which allows to detect very different shapes of clusters, including more stretched out clusters that represent actions like driving, traveling, etc. Therefore, the Expectation Maximization (EM) algorithm [1] was selected for the first step of our process flow and was used for both the context and the application data processing.

The EM algorithm is a method for finding the maximum likelihood for the model parameter(s) for which the observed data are the most likely. In an iterative process, the algorithm alternates between performing an expectation (E) step and a maximization (M) step. During the E-step, using the conditional expectation, the missing data are estimated given the observed data and a current estimate of the model parameters. In the M-step, the expected likelihood function found in the E-step is maximized under the assumption that the missing data are known. These new parameter estimates are then used to determine the distribution of the latent variables in the next E-step. The algorithm ends when the iterated solutions converge or when a given number of iterations has been reached.

With respect to the structure discovery use case scenario, mixtures of Gaussian functions were chosen to model the clusters of points. A Gaussian function is assigned to each cluster, with its mean in the middle of the cluster, and with a standard deviation that measures the spread of that cluster. The EM algorithm iteratively estimates the maximum likelihood of the mean and standard deviation of the Gaussian distributions and the probability for each point being drawn by a specific Gaussian.

This iterative process allows for the discovery of different clusters without the need to define in advance the number of clusters that ought to be found. With the application of Gaussian mixtures, these clusters may have very different shapes (in the n dimensional space) allowing to detect both compact and stretched out clusters. In our use case scenario, the latter type of clusters are found when users are e.g. driving or walking.

In order to evaluate the performance of EM clustering on the different types of data, a data set containing

Table 1
Monitoring sample of user data containing the user's context data and currently used applications. A question mark denotes that the context parameter could not be monitored, a back-slash denotes that no application was used.

latitude	longitude	day of week	time	speed (km/h)	applications
51.03419	4.21742	Monday	7:40	0.027	AlarmClock
51.03418	4.21739	Monday	7:50	0.025	Agenda
51.03950	4.21611	Monday	8:00	71.255	Music, Email
51.04300	4.21559	Monday	8:10	81.010	Music, Facebook
51.04877	4.21504	Monday	8:20	69.877	Music, Email
51.05122	4.21401	Monday	8:30	0.020	Quickoffice
51.05122	4.214	Monday	11:00	0.015	\
51.05122	4.21415	Monday	15:20	0.028	Phone
51.04328	4.21558	Monday	17:00	70.444	Music, Facebook
?	?	Monday	17:10	?	Music, Email
51.03417	4.21742	Monday	22:30	0.027	\
51.03418	4.21738	Tuesday	0:05	0.025	Youtube

four weeks of user data was used. A sample of this user data set, containing the four monitored context parameters (with location indicated by a combination of longitude and latitude variables) and the at that moment used application(s) is shown in Table 1. Based on this data sample, the process of context clustering and application clustering will be detailed in the following sections.

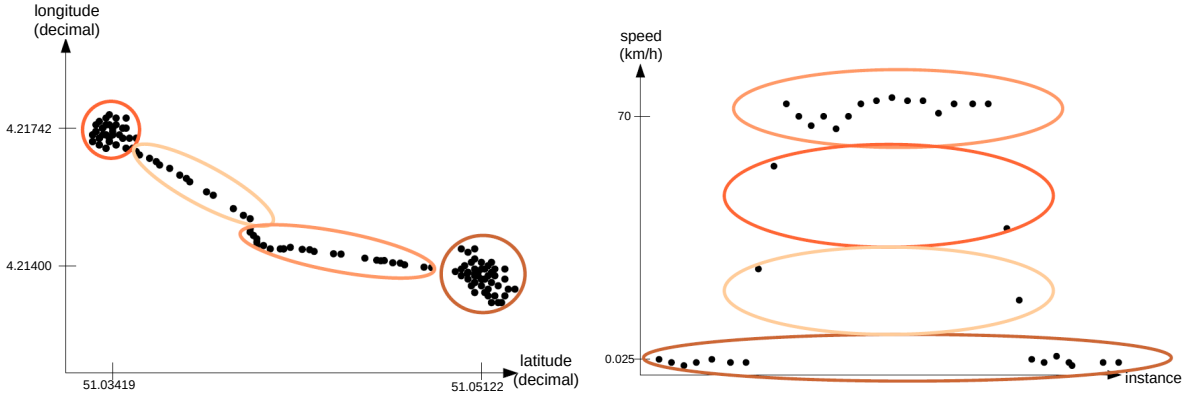
While the rows in this example are chronologically listed, the actual data are more fine-granularly monitored, containing extra rows between each listed row of data in this sample. Additionally, larger data samples typically contain more repetitive patterns allowing to cluster more accurately. Note that some parameters may not be available during a monitoring session. For instance, GPS functionality may not be available resulting in the omission of the latitude and longitude parameters in some monitoring samples. In the listed fragment in Table 1, a user first finds himself at the same location, then moves at a high speed to another location from which he returns at the end of the fragment. The next section details the clustering process of these parameters based on the sample of Table 1.

2.2. Context Data Clustering

A first series of data clustering is performed on the level of the context parameters. First, EM clustering is performed based on each separate context parameter. For instance, to find clusters based on the different locations where a user has been, *longitude* and *latitude* are taken as the input parameters for the EM clustering algorithm. Similar to this two-dimensional clustering, one-dimensional clustering is performed on the context parameters *time* and *speed*. With respect to the context parameter *day of the week*, no additional clustering is required as a week can be considered as a composition of 7 days or clusters. The resulting clusters, based on the example of Table 1, are shown in Table 2.

For the context parameters *location*, *speed* and *day of the week* all monitored data points are clustered, even when no applications were used during that monitoring point. The more data are known about e.g. the location, the more accurate location clusters can be found. However, for the context parameter *time*, taking every monitoring point into account would only create a uniform time axis. Therefore, for time-based clustering, only those monitoring points are taken into account where the user was actually using an application. The adjective *time* clusters are also shown in Table 2.

When clustering is performed based on each separate context-parameter, too many clusters may be found in some scenarios. This is shown in Figure 1 where two-dimensional *location* (Figure 1a) and one-dimensional *speed* (Figure 1b) monitoring values of the same data set are first separately clustered. In this example a person took the train from one location to another. For the person this feels as being on one location, i.e. on the train, but through EM clustering the train journey results in multiple clusters for each of the context parameters. This is due to the fact that a train does not move at a constant speed (e.g. due to intermediate stops or acceleration) and that several location



(a) (b)

Fig. 1. EM clustering on two-dimensional location data points (a) and one-dimensional speed data points (b).

Table 2
Data sample after clustering on individual context parameters.

lat.	long.	day of week	time	speed (km/h)	applications
Location 1		Day 1	Time 1	Speed 1	AlarmClock
					Agenda
Location 2			Time 2	Speed 2	Music, Email
					Music, Facebook
					Music, Email
Location 3			11:00	Speed 1	Quickoffice
					\
			Time 3		Phone
Location 2			Time 4	Speed 2	Music, Facebook
					Music, Email
Location 1			22:30	Speed 1	\
		Day 2	Time 5		Youtube

Table 3
User data sample after context profile clustering.

lat.	long.	day of	time	speed	applications
Profile 1					AlarmClock
					Agenda
Profile 2					Music, Email
					Music, Facebook
					Music, Email
Profile 3					Quickoffice
					\
					Phone
Profile 2					Music, Facebook
					Music, Email
Profile 1					\
					Youtube

points on the train's path are also monitored resulting in additional and different location clusters.

In order to cope with this problem, an additional clustering is performed on the monitored data, combining all context parameters. This results in a five-dimensional clustering. Through EM clustering, this complex clustering was found to result in clusters which very much resemble typical real-life profiles such as on the train, at work or sleeping at home. The result of this combined clustering on the data sample of Table 1 is shown in Table 3.

It is important to note that these five-dimensional clusters or profiles are not replacing the previously found clusters. In our process flow, the profile clusters are used in the second step to find profile-specific patterns. These are patterns that only take data into account that were gathered during the occurrences of the same profile. The clusters found by clustering on the individual parameters are used as input data for certain application patterns of step 2.

2.3. *Application Data Clustering*

A second type of clustering is performed on the level of the monitored applications. When a monitoring session is initiated, the application that is currently running in the foreground (i.e. which is visible to the user) is registered. As only one application can run in the foreground at one given point in time, only one application can be registered. An exception is made for the *music* application. Although the concepts of (real) multitasking and background services vary depending on the mobile platform that is used, on all popular mobile platforms the platform's native music application

is able to perform its main task (i.e. playing music) in parallel with other active applications. As a result, each monitoring session may contain up to two active *foreground* applications. When the phone is in standby mode or when the home screen of the mobile platform is shown, no active application is registered (with the exception of the music application when active). For the clustering on application level, the standby mode of a device is taken as the reference point. All applications that are used between two standby modes of a device are taken as one cluster. Typically, such a cluster contains only a small number of applications. In Table 4, an example of combining context data and application data clustering is shown.

2.4. EM Performance and Configuration

Time and memory complexity of EM clustering are harder to predict due to the unknown number of iterations. It is shown [2] that EM clustering generally performs well for huge data sets when compared to other clustering algorithms such as for instance hierarchical clustering.

The exact configuration of parameters is a complex process as it is hard to define values that perfectly match all types of users and user scenarios. Furthermore, this process requires manual involvement and often a trade-off between performance and accuracy should be made. Therefore, the impact of the several configuration parameters was evaluated based on a data sample encompassing four weeks of user data containing different types of realistic user scenarios. The following configuration parameters were evaluated:

- *maxIterations*: the number of maximum iterations, has an impact on accuracy and performance.
- *seed*: determines the initial number of random clusters.
- *minStdDev*: the minimum standard deviation of each Gaussian distribution.

maxIterations and *seed* were found to perform weakly for lower values, while too high values had a major impact on the time performance with little accuracy improvement. During all further experiments both parameters were set to 1000 which was found to produce accurate results within a reasonable time frame. For the parameter *minStdDev* evaluation values ranged from 10^{-8} up to 100. An optimal value for *minStdDev* was found to be subject to the proper interpretation of the data. For example as opposed to the application of lower values, the application of higher values resulted for instance in the distinction between *evening at home during the working week* and *weekend at home* profile clusters while this configuration was not able to distinguish between *working at home* and *weekend at home* data clusters. Overall, when taking the proposed context parameters into account, several appropriate profile clusters were found for both high and low values for *minStdDev*.

3. Application Usage Pattern Recognition

EM clustering on both context and application level, as described in the previous section, is performed to obtain more structure in the unstructured data. After this initial data preparation phase, the algorithm starts with the discovery of patterns in the user's application behavior.

3.1. Application Usage Pattern Overview

Several user application behavior patterns may exist. The following patterns are taken into account in our recommendation framework:

1. Profile-specific applications. E.g. in a *working* profile the applications Quickoffice, Agenda and Email are typically used.
2. Sequential patterns between applications. E.g. after the email application, the mobile agenda is often consulted. The sequence of application usage within a profile, e.g. an application that is always used at the beginning of a profile, is also included in this pattern search.
3. Applications that are used together. E.g. the music application is often used together with the Facebook application.
4. Applications that are typical for specific context parameters. E.g. a user may have certain application routines when waking up, while driving, at the super market or during the weekend.
5. Applications that have recently been used. When a user has recently used an application several times, there is a certain chance that he or she will use that application again. E.g. when a new game has been installed on the mobile phone.

Figure 2 provides a schematic overview of the clustering process on the different instances of input data together with the subsequent pattern deduction meth-

Table 4
Monitoring sample of user data after application usage clustering.

lat.	long.	day of week	time	speed (km/h)	app.
Location 2		Day 1	Time 4	Speed 2	(Music,Email,Facebook)
Location 3			Time 2	Speed 1	(Quickoffice),(Phone)
Location 1		Day 2	Time 1		(AlarmClock,Agenda),(Youtube)

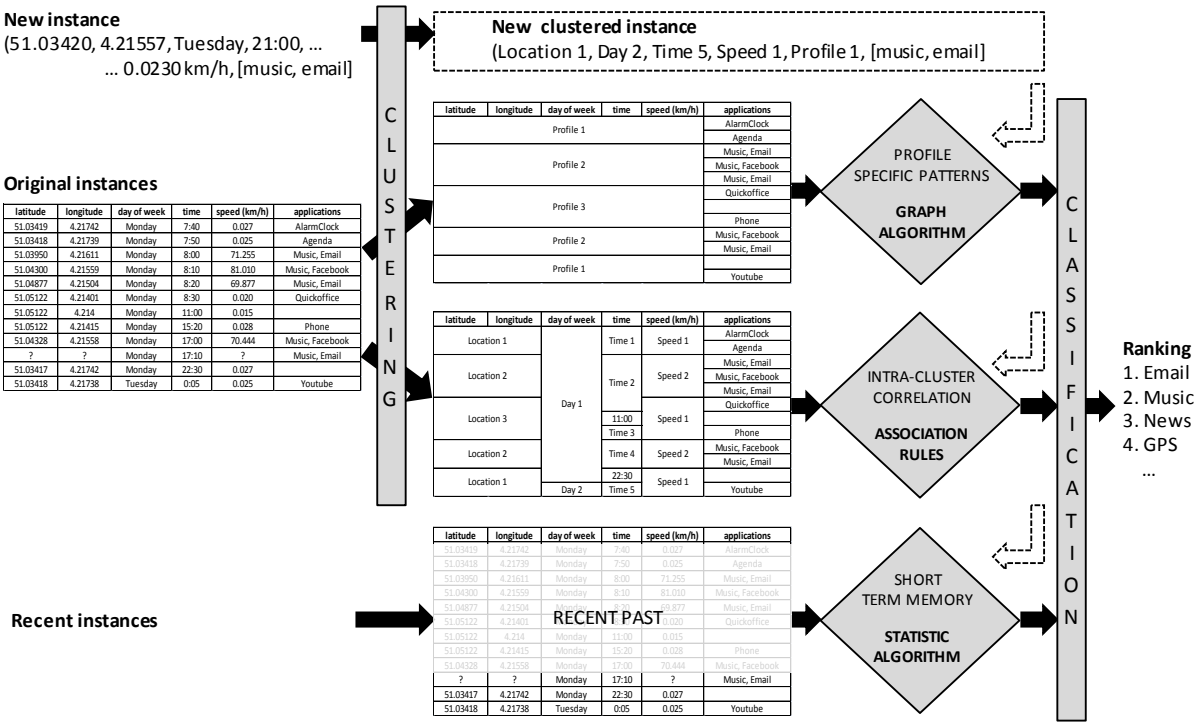


Fig. 2. Schematic overview of the data input, clustering process and subsequent pattern recognition processes.

ods. For the first two types of patterns, a graph algorithm is applied. These are the profile-specific patterns. For pattern types 3 and 4 association rule learning is used to find the intra-cluster correlations. Finally for the recently used applications a statistical algorithm is applied.

As shown in Figure 2, a distinction is made between three types of input data. The original or historic data instances are clustered on both application and context level. For the context level clustering both separate and combined clustering of the context parameters are taken into account. Combined context or profile clustering is taken as the input for the profile-dependent patterns (types 1 and 2) while individual context parameter clustering is used to find the intra-cluster patterns (types 3 and 4). The most recent monitoring instances are used as the input for the short term functionality that is encompassed by pattern type 5. Every new data instance that is monitored is clustered on both application and context level and taken as input for all three algorithms.

3.2. Graph Algorithm for Profile-dependent Patterns

The usage of applications is often very closely related with the different user profiles as found through EM clustering of the combined context parameters. E.g. at home, more leisure-related applications such as the mobile browser, facebook or the mobile TV guide, are used while at work more business-related applications such as the agenda, office, etc. may be consulted. Once the profiles have been detected, as detailed in the previous section, a weighted directed graph is composed for the currently active profile. An example of such a graph is shown in Figure 3.

In this graph, the nodes represent the applications that have been used most during a specific profile, in this case a working profile. Each of these nodes has a value representing the number of times that the application has been used during that specific profile. Additionally, in order to represent the sequential patterns between applications, directed and weighted edges are used between the nodes with each edge containing two weights: the number of times an application has been used after another application and the average time between the usage of those two applications. E.g. in the working profile represented by the graph of Figure 3, the agenda has been consulted 27 times of which 8 times occurred almost immediately (i.e. within an average time span of 1 minute) after a phone conversation.

This graph allows the system to take into account both the current active profile and the current time period since the last opened application. This last opened application is used as a starting point in the graph to determine which applications should be taken into consideration. For each of these applications, the ratio of the time since the last application has been opened with the average time of each application, is multiplied by the number of times each application was started after the last opened application. The resulting scores of all applications are then compared to calculate the probabilities of the applications.

For example, if the Phone application was last used during the working profile as shown by the graph in Figure 3 only the Email, QuickOffice and Agenda applications should be taken into consideration. If half a minute passed since last opening the Phone application, the corresponding probabilities for the applications are about 20% for the Email application, 2% for the QuickOffice application and 78% for the Agenda application. Note that when the last opening time exceeds the average time, the inverse time ratio is taken into account. As such, after 10 minutes of not opening any application the corresponding probabilities are respectively 64%, 26% and 10%.

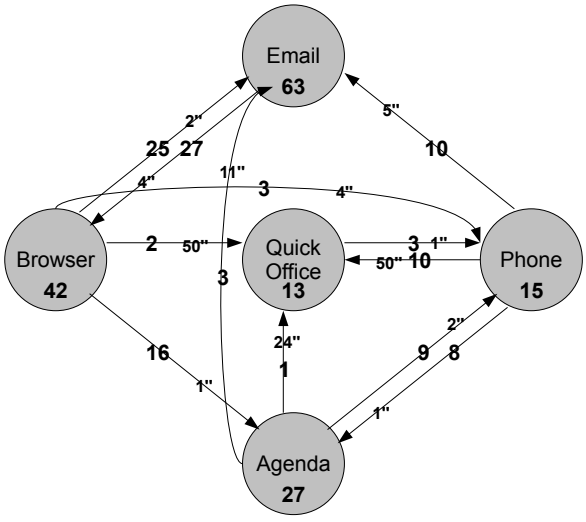


Fig. 3. A weighted directed graph for a working profile, taking into account the time differences between different applications.

3.3. Association Rules for Intra-cluster Correlations

Application usage is not necessarily profile-related. A user may have certain routines which are linked to a specific point in time, a day, a place, etc. For example, a student may set his alarm clock every evening regardless of his location (e.g. at home or at his student flat). This is defined as a more time-specific action. In order to retrieve this type of patterns, association rule learning is used. This is a method for discovering interesting relations between variables in large databases. More precisely, in our system the Apriori algorithm [3] was applied to the context parameter cluster information, found by clustering each context parameter separately, and to the application level clustering information. This algorithm is designed to operate on databases containing transactions, for example collections of items bought by customers. Given a series of itemsets, the algorithm attempts to find subsets which are common to at least a minimum number of the itemsets. In our scenario, the combined context and application clusters are used as transactions and the different attributes *day of week, time, location, applications,* etc. are considered as the items of the transactions. An example set of association rules that may be found is e.g.:

```
[88%] Location 1,Time 5==>AlarmClock
[97%] Location 1,Time 5,Speed 1==>AlarmClock
[1%] Location 1, Time 5,Speed 2==>AlarmClock
[74%] Location 2,Email==>GoogleMaps
[77%] Facebook,Email==>Youtube
```

The premise of such association rules (e.g. *Location 1, Time 5*) is the condition that needs to be fulfilled to have, with a certain confidence (88%), the consequence *AlarmClock* as a result. For the recommendation of applications at a certain moment, the contextual information of that moment is clustered and the resulting clusters are compared with the premises of the as-

sociation rules. If a premise of an association rule is fulfilled, the application of the consequence receives the association rule's accompanying confidence. This value is set as the probability of that application. When multiple association rules correspond to certain context parameters, only those with the highest number of corresponding parameters are taken into account. For instance, in the above set of association rules, if the current context is indicated by the clusters *Location 1, Time 5, Speed 1*, a probability of 97% is awarded to the *AlarmClock* application, neglecting the 88% that is stated in the first association rule. Not only context parameters can be part of the premise, also applications can be taken into account. For instance, the last rule of the above set of association rules indicates that the recent use of the *Facebook* and *Email* applications may lead to the usage of the Youtube application. A premise containing a combination of context and application parameters is also supported.

3.4. Statistic Algorithm for Short-Time Window Analysis

When a user has recently used an application multiple times, there is a certain chance that he or she will use the same application again in the near future. To take recently opened applications into account a short-term window is mapped on to the user's application usage. This window determines which applications should be taken into account. All applications that are not included in the window, receive for this pattern a probability of zero per cent. While the composition of the short-term window may be changed, in our system the window encompasses a fixed number of the most recently opened application instances. Each application instance receives an index based on the position in the window. For each application in the window, the index values of its instances are averaged and compared to those of the other applications that are encompassed by the window. The mutual index ratios are then taken as the probability values for this pattern. Note that for this algorithm, no input from the data preprocessing phase is required.

3.5. Combined Recommendation

When the probabilities of all patterns have been calculated, these probabilities are combined via a function *f()* to achieve a combined recommendation value for each application. These different probabilities can be combined in several ways. During the experiments, a weighted sum and a maximum function have been evaluated. The weighted sum function weights the predictions of all three algorithms while the maximum value only takes the relative highest prediction value for each application into account.

4. Implementation Details

This section provides details of our framework architecture, describing how the required information was gathered on a mobile device and how the algorithms were implemented. The implementation of the prototype widget application is also described. Due to performance considerations, a client-server oriented approach was chosen, where all computational intensive tasks are offloaded to the server component. The Android platform was chosen as the target OS for the mobile device, Java was chosen for server side development.

4.1. Client-Server Communication

Communication between mobile client and server is performed by means of the REST (REpresentational State Transfer) protocol. Server side an Apache Tomcat v7.0 web server was used together with the Jersey library, an open source JAX-RS implementation for the creation of RESTful Web services. On the Android client, an Android version of the Spring framework was used to handle REST communication in combination with the Jackson JSON processor for the serialization of Java objects.

Three interface classes were defined for communication between client and server. One class, labeled *Fix*, bundled all context parameter values from one monitoring session, such as time, speed, whether or not music was playing and the active application. For the applications only the package name of the application was required and saved as a *String*. For the mobile Facebook application for instance, the string *com.facebook.home* is used. A second class bundled extra information about the currently active context profile, such as the average speed or the typical location in coordinates. Finally, a third communication object contained a sorted list of the predicted applications, in order of prediction rank, together with the identification of the relevant profile.

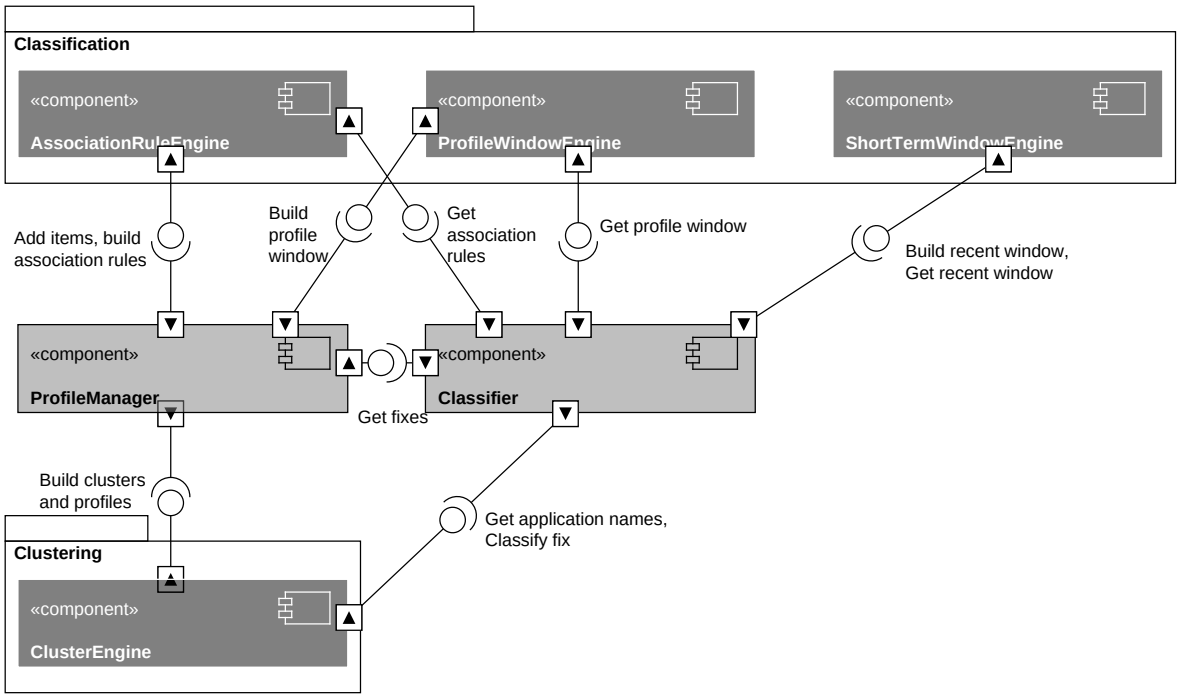


Fig. 4. Component diagram of the server components that handle the core functionality.

4.2. Server Component

The server can be split up in two main parts: components that handle all web or communication functionalities and components that handle the core functionality. Figure 4 shows the main components that handle the core framework. The *Engine* classes are responsible for the intelligence of the framework and implement the different algorithms that have been described in this paper. All engines have a *build* method to execute the data mining algorithms and a *get* method to retrieve the results of that operation. This distinction is made as the detection of patterns is not always directly linked with the prediction of applications. For instance, association rules are built after every cluster generation or update, while the predictions are only required when new application recommendations or predictions are actually required.

The *ClusterEngine* clusters all received *Fix* objects, as described in Section 2. For the Expectation-Maximization clustering, the Weka framework was used from within our Java code. Weka is a collection of machine-learning algorithms for data mining tasks. An introduction to the Weka workbench can be found in [12]. During clustering, Weka builds up a cluster model, which allows to cluster new data to one of the previously found clusters. This is required to classify new contextual data to the clusters that are common for that user.

The *AssociationRuleEngine* bundles all methods that are required to build up or request info from the association rules. The A Priori algorithm was used, which is also found in the Weka framework. After each new generation of association rules, all association rules that contain no application information in the consequence of the rule are filtered out of the result set, as they are not relevant for our use case scenario. This greatly improves the performance and efficiency of the predictions.

During the build method of the *ProfileWindowEngine* new graphs are generated, based on previous user behavior. When a prediction is requested, the engine runs trough the active graph of the current context profile, starting from the last opened application, to predict the usage probability for each application. The *ShortTermWindow* continuously keeps track of the most recent applications to calculate their probability rating.

ProfileManager and *Classifier* are the main interface points between the web-oriented modules and the core functionality. The former is responsible to man-

Table 5
Time overhead for the creation of context clusters and association rules.

	Duration [s]
Clustering: Location	9.08
Day	0.905
Time	45.691
Speed	6.174
Profile	16.146
Association Rules	3.886
Overhead	13.234
Total	**95.116**

age all tasks that are related to the context profile creation while the latter manages the creation of the predictions.

In order to situate the time overhead of each algorithm some time benchmarks with respect to the performance of the server components and algorithms were executed. The server components were running on an Intel Core i5-520M (2.4 GHz) dual core processor with 5.8 GB of internal RAM storage and a data set containing 2,046 data samples was used. Table 5 shows the time that is required for the creation of context clusters and association rules.

As opposed to clustering based on the day of the week, Table 5 shows that location-based clustering (two dimensional) and Profile clustering (five dimensional) require much more time. A very high time overhead was found for clustering on the parameter time. This is due to the fact that it is very hard to detect structure or coherence in the time-related data. The creation of association rules only requires around 4 seconds while the extra time overhead that is introduced by the framework is more than 13 seconds. This is mainly due to the preparation and filtering of the data. In total all of these operations take up around 95 seconds. While this is relatively long, it should be noted that these operations are not continuously required. It should be sufficient to regenerate the clusters of rules only once a day. If a considerable amount of data from one user have been collected, it should be sufficient to execute these steps even less frequently.

Table 6 shows the results of the time overhead that is imposed for the generation of new predictions. This shows that most time is taken up by the classification of a new *Fix* with the active association rules. This is due to the fact that thousands of rules have to be compared with the *Fix* to determine if they are valid or not. For each comparison, the clustered attribute values have to be compared with the attribute values of the *Fix*, which induces wrapping and unwrapping of objects.

Table 6
Time overhead of the different prediction algorithms.

	Duration [s]
Profile Graph	0.021
Association Rules	4.864
Short Time Window	0.04
Total	4.889

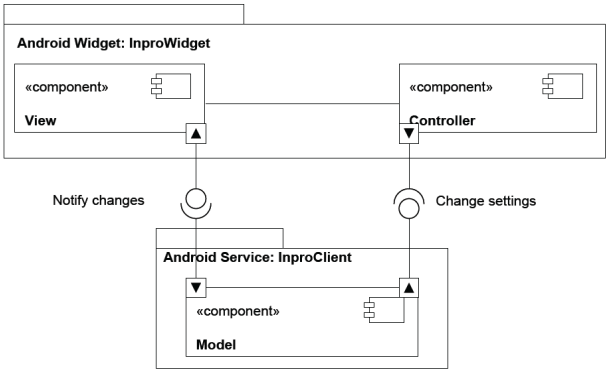


Fig. 5. Component diagram of the client service and widget components.

In this evaluation scenario, the total required time for the generation of predictions for a given context situation was almost 5 seconds. If we want to recalculate the predictions every minute for one user, this 5 second overhead time causes no problems. However, if the service has to be offered to many users at the same time, additional techniques or better hardware is required. A possible optimization would be the introduction of a classifier, for instance decision trees, in the filtering step of the association rule creation phase in order to filter the created association rules and only retain the most relevant rules.

4.3. Client Component and Mobile Prototype Application

For the implementation of the mobile client, the Android OS was chosen. As shown in Figure 5, the client architecture was built upon the Model-View-Controller paradigm and was composed of two separate components: a widget component and a service component. The service component should be seen as a substantial part of the overall architecture, while the widget component was only implemented as a prototype in order to visualize the predictions.

The service component provides the following functionalities:

– Monitoring of the context data and the application usage data on the mobile device.
– Temporary storage of the monitored data on the mobile phone.
– Sending data to the server component.
– Sending predictions to the prototype widget application.

All the monitored context data are bundled in a *Fix* object. Two types of monitoring requests can be made. A first request monitors the current context situation, containing the currently active application (if open) together with other disposable context data. A second request retrieves all applications that have been opened since the last standby modes of the device. These applications are also bundled into one *Fix* object. In order to detect the standby mode of an Android mobile phone, a class was created that inherits from *BroadcastReceiver* in order to intercept the intents ACTION_SCREEN_OF and ACTION_SCREEN_ON that are sent via *sendBroadcast*.

Fix objects are not directly sent to the server upon creation, but will first be internally stored until multiple *Fix* objects have been monitored. Both the monitoring rate of new fixes and the required number of fixes in order to send them to the server can be dynamically changed depending on the battery consumption, internal storage capacity and speed and network characteristics. The average size of one *Fix* object was found to be 577 bytes.

The prototype application that was developed is an Android widget that visualizes relevant information with respect to the current context situation together with a pyramid construction containing the icons of the top six predicted applications for the current context state of the mobile user. A screenshot of the prototype widget is shown in Figure 6.

In addition to the display of current context information such as the typical average speed or average location, the contact person that was last called during that context profile, or a previous active state of that context profile, is shown. The widget also keeps track of the mobile phone's sound mode (normal, silent, etc.) for each context situation and automatically switches to the last active sound mode of a context profile when an active context profile is detected.

5. Evaluation Setup

A profound evaluation of the generated predictions requires an extensive data set containing detailed monitoring samples of different types of user profiles over a relatively long period. A proper evaluation methodology is also required to determine whether the predictions are considered to be good or bad. This section first details how evaluation data were obtained by implementing a modeling tool. Second, a proper evaluation metric is presented.

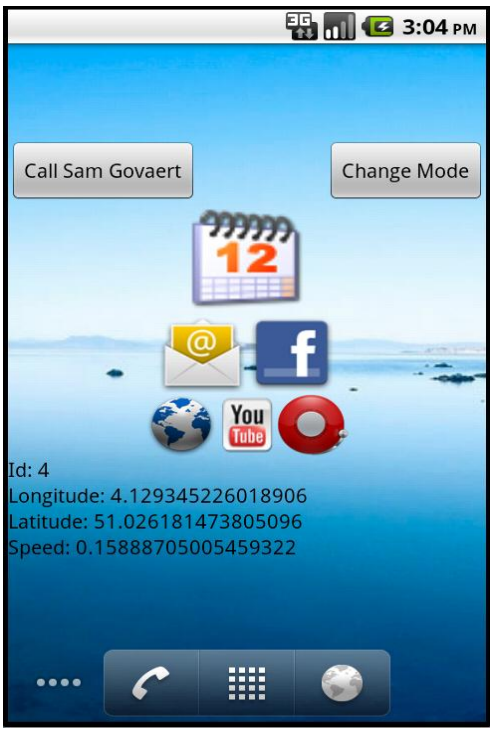


Fig. 6. Screenshot of the prototype widget application running on the home screen of an Android OS mobile device.

5.1. Modeling Tool

In order to evaluate the predictions that are made by the presented algorithms, realistic user data from different user profiles and over a longer period of time should be obtained. Instead of continuously monitoring real users, we chose to implement an advanced modeling tool. This modeling tool emulates users' behavior in terms of context parameters and application usage. The advantages of using the modeling tool over real user data consists in the fast evaluation of different user profiles over a longer period of time together with the easy integration of 'exceptional situations' such as only working half a day at work or a significant train delay. It should be noted that the modeled user scenarios are based on realistic use cases, encompassing real-life context situations, based on actual context data

(e.g. coordinates from the authors' home and work address, actual train route and speed).

Figure 7 shows the profile generation work flow of the modeling tool. This flow encompasses the following consecutive steps:

1. First, different context profile building blocks are generated. These profile blocks correspond to the profile clusters that were found in Section 2.2. Examples of such building blocks are *Wake Up, Work, Train,* etc. All of these profile blocks are defined by one or more context parameters. A certain random factor is automatically added to the context information. The addition of time information is compulsory and can be variably configured for each profile block. E.g. a *Working* block may in some cases take up the whole day, while in other cases or on certain days, it may only take up half a day.
2. To each profile block, certain applications are connected, each application having a certain probability that it may be used during that specific profile block.
3. If necessary, additional patterns are further described. These additional patterns can be coupled to one specific context profile block or can be defined in general, applying to all context situations. The supported patterns are:
   - Start profile patterns, defining which application is typically first used when a new profile starts.
   - Application patterns, to couple application usage to one specific context parameter. For instance using the AlarmClock application before going to bed.
   - Cluster patterns: define relations between applications that are often used together.
   - Repetitive patterns: encompass applications that may be frequently used during a given period of time.
4. Once all building blocks and patterns are defined, the different context profile blocks are coupled to generate a context time line composed of (variable) day and weekend schedules. In addition, special conditions may apply to each context profile block such as a delay during a train journey.
5. Finally, output data are automatically generated for the defined context time line, based on the defined patterns and on the context profile blocks' characteristics. The output data contain a continuous set of realistically changing context data together with the used applications.

5.2. Evaluation Methodology

In order to measure the performance of our recommendation system, two different user scenarios have been evaluated based on the data that were generated by the modeling tool. Both scenarios were defined in and generated by the modeling tool. In the first scenario A a moderate mobile application user was defined, consuming up to 17 different applications per week with each application being used several times a week. In the second scenario B a more active user was created, consuming around 25 different applications per week with some applications being used only very occasionally. Both user scenarios are composed of several types of working, home, weekend and transportation profile blocks. Scenario A encompasses two weeks of data, for user scenario B a time line for three varying weeks was created.

In order to evaluate the recommendations, both historical user data and new evaluation data are required. Therefore at least two sets of data should be generated for each user scenario and compared to each other. For each user scenario five different data sets were created by the modeling tool. Just like in real-life situations these data sets should not contain exactly the same data as context parameters and application usage often vary. However, when evaluating over a longer period, similar profiles and patterns should be detected in both the training and the evaluation scenarios. By means of the modeling tool, enough variation was created between the different sets for each scenario while similar profiles and patterns were generated. This is shown in the diagrams in Figure 8a and Figure 8b, which compare the five different data sets of respectively user scenario A, encompassing 17 applications (N=17) frequently spread over a period of two weeks, and user scenario B, encompassing 26 applications in total spread over a period of three weeks. In order to obtain valid evaluation results the number of disjunct applications was set large enough. In addition, the chance that an application may be opened was limited so that no extreme scenarios are evaluated as this could lead to extreme positive or negative evaluation results.

For both user scenarios, each generated data set is once used as training data set, while the other four data sets are used as evaluation data, resulting in 20 similar evaluation experiments per scenario to measure the impact of one recommendation configuration. For

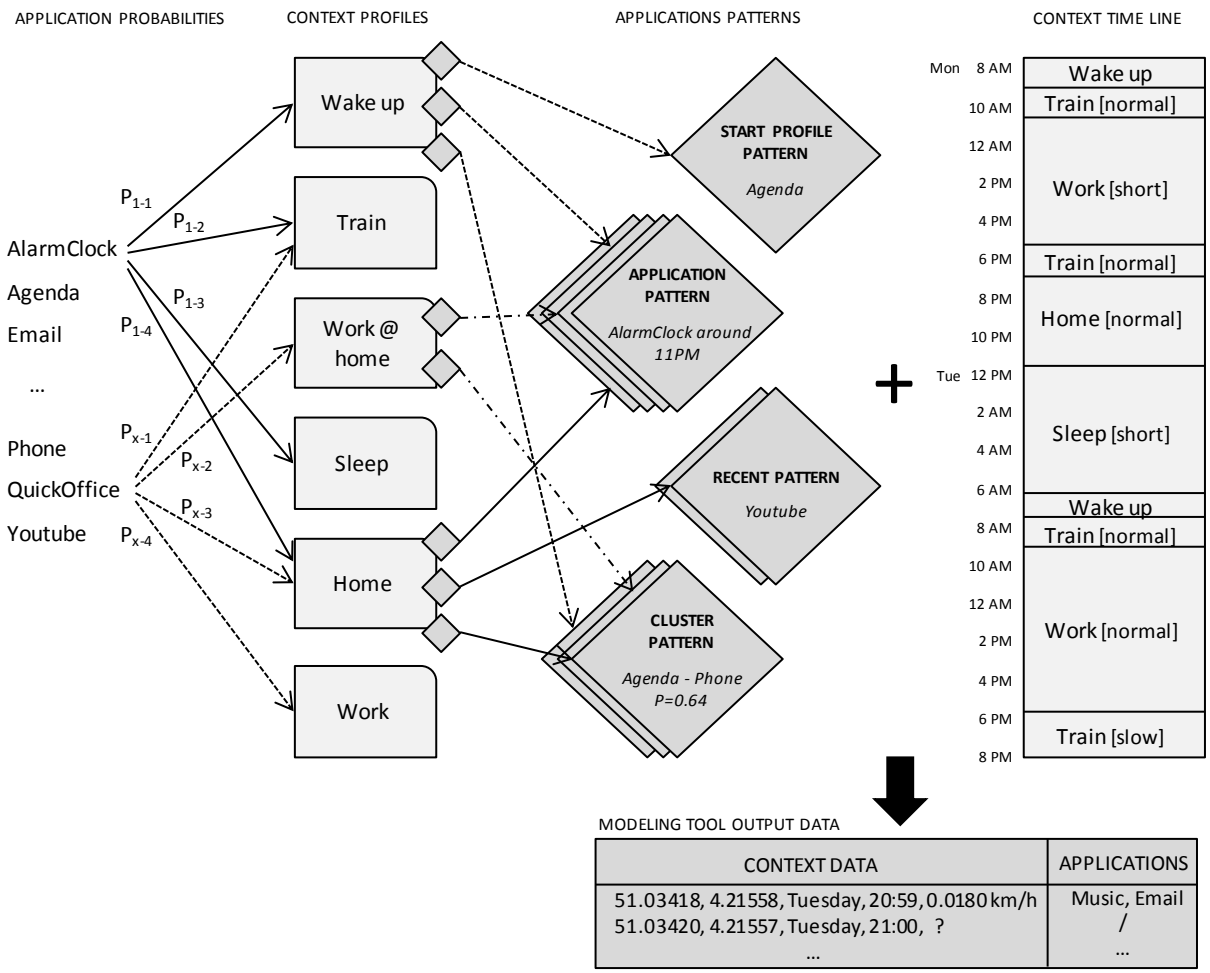
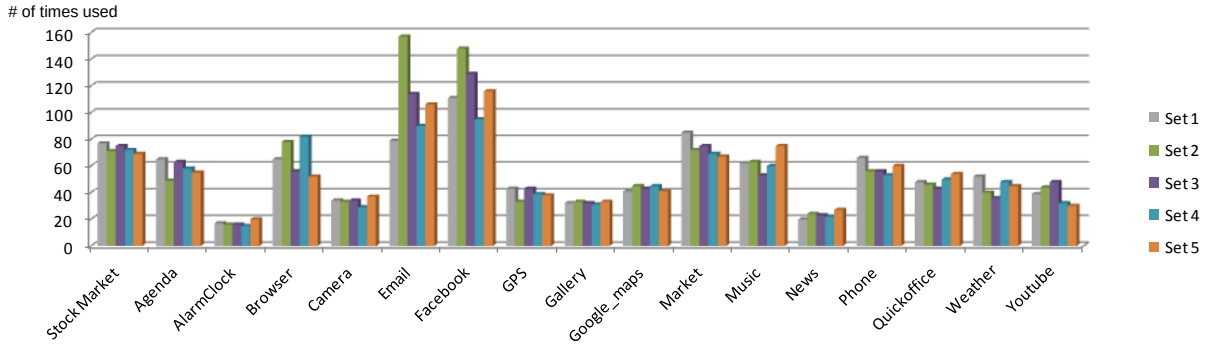


Fig. 7. Profile generation work flow of the modeling tool.

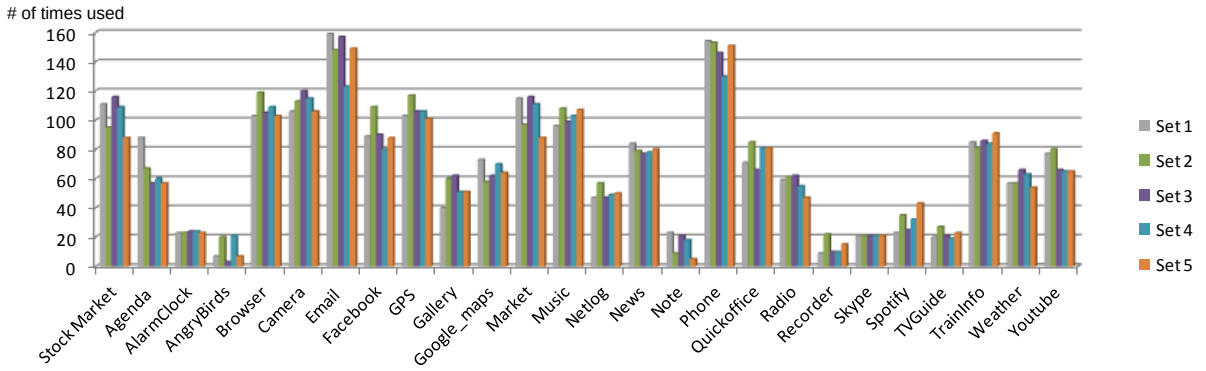
the training data, both the context data and application data are clustered and used as input for the detection of the different patterns, described in Section 3. Concerning the evaluation scenarios, the context data are used as input for the recommendation algorithms. Based on the training data and the context data of the evaluation scenario, the recommendation engine then proposes a list of applications. During our evaluation, different list sizes are taken into account, with a maximum of up to six applications as this corresponded with our mobile client's widget interface. In order to evaluate the performance of the recommendation engine, the recommended applications are compared with the used applications of the evaluation data by means of an evaluation metric, which is described in the following section.

5.3. Evaluation Metrics

Within the field of recommendation systems, several evaluation metrics have been defined. One of the most commonly used metrics is the Mean Absolute Error (MAE), a quantity used to measure how close predictions are to the evaluation data. This metric is less applicable for our evaluation setup as both the predictions and the evaluation data are not exactly quantified. Another approach is to compare the recommendation set with the evaluation set. Two often applied metrics are Kendall's τ and Spearman's ρ. Both metrics focus on the similarity of the orderings of the two data sets when both are ranked by a specific function. A requirement for this type of metrics is that both data sets should contain the same number of elements. As the evaluation data set is only composed of one appli-



(a)



(b)

Fig. 8. Distribution of the mobile applications in the different generated data sets for scenario A (a) and scenario B (b).

cation while the recommendation set contains six applications, this type of metrics is not applicable.

Finally, a third type of evaluation metrics focuses on the fact how well a recommendation engine is able to distinguish correct from incorrect predictions or recommendations. Receiver Operating Characteristic (ROC) curve analysis is a well-known example of this type of evaluation metric. It is created by plotting the fraction of true positives out of the positives versus the fraction of false positives out of the negatives at various threshold settings. The set of positives bundles the items that an engine predicts as good or relevant items. Negatives are the set of items that are predicted by an engine to be unrelevant or unsuitable items (for a user). The concepts of 'true' and 'false' indicate whether the engine was right or wrong for that prediction. For instance, true positives are all items that were predicted to be relevant and after evaluation indeed proved to be relevant to the user.

For our evaluation, the concepts of *true negatives* and *false positives* are harder to define accurately and therefore a proper metric was applied, only taking into account the concepts of *true positives* and *false negatives*. For each application that was used in the evaluation sets, it was evaluated whether that application was included in the recommendation list at the moment the user started to use that specific application. Subsequently, the ratio of the number of used and recommended applications on the total number of used applications is taken. This metric is labeled the Information Parameter (IP) and its formula is shown in Equation (1).

$$IP(M,N,T) = 100 \times \frac{\sum_{t=0}^{T} R(r(a_t))}{T} \qquad (1)$$

$$with\ R(r(a_t)) = \begin{cases} 1 & \text{if } r(a_t) \leq M \\ 0 & \text{if } M < r(a_t) \leq N \end{cases}$$

In this formula, M is the size of the recommendation set (varying between 1, 3 and 6 in our experiments) and N is the number of disjunct applications

that can be opened on the user's phone. *T* is the number of data instances in the evaluation data set (Used B) where the actual usage of an application a_t was monitored. A high value is required in order to guarantee a certain accuracy. In our experiments *T* was varied between 890 and 1008 for user scenario A and between 1758 and 1901 for user scenario B. $r(a_t)$ is the position (between 1 and N) of the application a_t in the list of applications, sorted based on the calculated probability for each application at the moment that application a_t was opened. Finally, $R(r(a_t))$ translates the position to a value. If the application was included in the recommendation list, $R(r(a_t))$ is set to 1, otherwise it would be zero.

In this formula, normalization should be applied to take into account that even a random recommender has a certain chance $\frac{M}{N}$ that the used application is included in the recommendation set. This Normalized Information Parameter (NIP) is defined as follows:

$$NIP(M,N,T) = 100 \times \frac{\frac{\sum_{t=0}^{T} R(r(a_t))}{T} - \frac{M}{N}}{1 - \frac{M}{N}} \quad (2)$$

6. Evaluation Results

During the evaluation phase, each pattern deduction technique was first separately evaluated and then the combined recommendation based on different classification functions was evaluated.

6.1. Profile-dependent Patterns

For the evaluation of the profile-dependent pattern deduction techniques, the graph algorithm described in Section 3.2 was slightly simplified by not taking intra-application time changes into account as the modeling tool was not able to model such time changes. Table 7 shows the performance of the recommender system when only taking the profile-dependent patterns into account, by means of the graph algorithm as detailed in Section 3.2.

For the *true positives*, i.e. the applications that were used and recommended, a distinction is made between M (=6) different recommendation values, dependent on the relative position of the applications in the recommendation set, with the smallest value being those applications that were recommended most for a given context situation (and as such appear on top of the recommendation list). For the evaluation of multiple context situations, the term *bin* is used to encompass those applications that received the same recommendation position, with a bin for each position (up to 6). The objective of these experiments is to obtain a higher percentage of true positives in bin *i* than the number of true positives in bin *i*+1. IP-3 and NIP-3 are respectively the Information Parameter and Normalized Information Parameters in case the recommendation list was only composed of the top 3 recommendations (Bin 1 up to 3), while IP-6 and NIP-6 take all six bins into account. The results of both scenario A and B are listed in Table 7. As explained in the description of the evaluation methodology of Section 5.2, five different data sets were generated for each scenario and each of these data sets is once used as input for the training set. Table 7 shows the average percentages (Avg) obtained over 20 experiments, together with the standard deviation (StDev) and the minimum (Min) and maximum (Max) values that were found during the experiments. Note that the column with header Bin 1 corresponds to IP-1, i.e. the Information Parameter when only the highest recommended application is taken into account.

Table 7 shows that, despite the fact that during the evaluation experiments the intra-application time changes are not taken into account, the simplified graph algorithm is still able to perform relatively well with average IP-6 values of 69.73% and 59.62% for respectively scenarios A and B. This implies that for scenario A in almost 70% of the cases, the opened application was in the list of top 6 recommended applications at that moment. For one experiment (out of 20) for scenario A a maximum IP-6 value of 73.47% was obtained while the minimum IP-6 value for scenario B was found to be 59.62%. When taking only the top 3 recommended applications into account an average IP-3 of 48% was found for scenario A, for scenario B an average percentage of 39.06% was obtained. The rather low standard deviation values for all parameters also indicate that the listed average value is a good estimation of the overall performance. One of the main reasons the simplified graph algorithm was still able to result in these relatively high values was due to the precendence of the data preparation phase. During this preparation phase, each new context situation is first mapped to one of the already known context profiles. Because for each of these context profiles a separate graph was used, often several applications had already been filtered out as recommendation candidates when they are never used for such a context situation.

Table 7
Profile-dependent pattern evaluation: percentage true positives (TP) per bin together with different IP and NIP values for both user scenarios.

	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6	IP-3	NIP-3	IP-6	NIP-6
Scenario A										
Avg	22.57	13.66	11.78	9.15	7.30	5.26	48.01	36.87	69.73	53.21
StDev	2.33	1.63	1.73	1.82	1.79	1.39	2.02	2.45	2.30	3.55
Min	16.72	11.29	8.70	5.72	4.45	3.39	42.82	30.56	64.57	45.25
Max	25.66	16.86	14.57	12.83	10.58	8.11	50.59	40.01	73.47	58.99
Scenario B										
Avg	15.50	12.25	11.31	8.81	6.23	5.52	39.06	31.12	59.62	47.51
StDev	1.52	0.99	1.01	1.36	0.87	1.35	1.52	1.72	1.70	2.21
Min	12.91	10.69	9.94	6.53	4.82	3.70	34.68	26.16	57.15	44.30
Max	18.57	13.73	13.93	10.85	8.24	8.46	41.86	34.27	62.54	51.30

Table 7 shows that the recommendation system clearly performs better for scenario A than for scenario B. This is due to the more complex character of user scenario B with the inclusion of more defined patterns (of all types), more applications that are used and a longer period of data in which more slightly diverse day time lines are introduced. In addition, some applications were defined to be used only very sporadically. However, when filtering out the chance that a random recommended application would be a true positive, by analyzing the Normalized IP values, it is shown that the real difference in performance is around 5.7% for both the NIP-3 and NIP-6 values. After normalization, the NIP-6 minimum values for both scenarios were found to be almost equally low.

A final observation is that for both scenarios, the highest percentage of true positives was found in the first bin while the number of true positives decreases with increasing bin number from 22.57% in Bin 1 down to 5.26% in Bin 6 for scenario A and from 15.50% down to 5.52% for scenario B. This implies that the highest impact on system performance is made by the upper bins, which contain the most recommended applications.

6.2. Intra-Cluster Correlations

As described in Section 3.3, association rules were used for finding the intra-cluster correlation patterns. Table 8 shows the evaluation results when only the results of these association rules are taken into account for both scenario A and B.

With an average IP-6 of 84.12% and an average IP-3 of 61.89%, the association rules perform very well for scenario A. Similar to the graph algorithm, a lower but still very good performance is measured for all parameters of scenario B. After normalization a performance difference of around 6% for NIP-6 and of 4.3% for NIP-3 was found between both scenarios. In the worst experiment of this series still 72.80% of the used applications was included in a recommendation list with size 6. The standard deviation values indicate that again, the obtained results did not vary a lot. Similar to the graph algorithm, it should be noted that clustering of the current context situation is first performed. As such, these relatively high performance results not only indicate that the association rule mechanism is able to define and distinguish correct association rules but that the data preprocessing and clustering is able to deliver very relevant input information to the pattern recognition engines. A second reason that these association rules perform so well is that they are searching for patterns between the different context clusters, and some of these clusters may coincide with profile clusters. As such, profile-related associations may also be found by applying association rules. However, sequential structures between different applications or within a specific context profile can not be found with this technique. Overall, it can be concluded that association rules in combination with EM clustering perform very well for the tested scenarios.

6.3. Short Time Window Analysis

Short time window analysis was used to recommend recently often used applications. The evaluation of this technique is shown in Table 9. Different window sizes (*W_Size*) were evaluated for both scenarios, taking into account the last 5, 10 and 20 recently opened applications. The average values are shown in Table 9.

Table 8
Intra-cluster pattern evaluation: percentage true positives per bin together with different IP and NIP values for both user scenarios.

	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6	IP-3	NIP-3	IP-6	NIP-6
Scenario A										
Avg	28.68	18.88	14.33	9.37	8.23	4.62	61.89	53.73	84.12	75.46
StDev	1.94	1.60	1.65	1.97	1.31	0.72	2.39	2.90	2.27	3.51
Min	24.83	15.69	10.70	6.13	5.67	3.37	56.49	47.16	77.55	65.30
Max	31.96	21.51	16.86	13.93	10.60	6.38	66.41	59.21	88.28	81.89
Scenario B										
Avg	26.78	16.02	12.44	9.37	6.52	5.34	55.24	49.40	76.47	69.42
StDev	0.91	1.20	0.97	1.12	1.06	0.92	1.60	1.81	2.24	2.91
Min	24.92	14.09	10.34	7.74	4.42	3.34	52.26	46.03	72.80	64.64
Max	27.94	18.31	14.38	11.66	8.67	7.17	58.34	52.91	79.68	73.58

Table 9
Short time window evaluation: average percentage true positives per bin together with different IP and NIP values for both user scenarios and different window sizes.

	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6	IP-3	NIP-3	IP-6	NIP-6
Scenario A										
5	11.87	18.07	10.28	7.33	7.84	7.78	40.22	27.41	63.17	43.08
W_Size 10	12.64	15.21	12.70	8.43	6.43	7.27	40.56	27.82	62.69	42.33
20	12.08	13.84	11.09	10.54	7.95	7.28	37.01	23.51	62.79	42.49
Scenario B										
5	6.69	12.49	7.04	4.40	5.55	5.94	26.22	16.60	42.11	24.74
W_Size 10	6.34	11.42	7.60	5.72	4.27	4.25	25.36	15.62	39.60	21.48
20	6.90	9.58	5.80	6.52	6.41	4.96	22.28	12.15	40.18	22.23

With an average IP-6 of approximately 63% for scenario A and around 40% for scenario B, this algorithm yielded relatively bad results in comparison to the other two algorithms. In addition, the number of true positives is not monotonously decreasing with increasing bin number with the highest percentage of true positives found in the second bin. It should be noted that the performance of the time window algorithm is strictly bound to the extreme popularity of certain applications during a given time frame. In addition, this pattern does not gain from the data preprocessing phase as it does not need to take into account the information from the context (profile) clusters. The advantage is that this algorithm allows the overall recommendation system to be partially independent of the clustering process while still providing reasonable recommendations.

With respect to the size of the short time windows, a window size of length 20 was found to perform worse than the other two window sizes when taking the top 3 recommendations into account. Overall, a window size of 5 was found to perform best for the calculated IP and NIP values. It should be noted that the smallest window size was subject to the highest deviation between the obtained results with a standard deviation of up to 4.49% and extreme IP-6 minimum and maximum ratings of respectively 53.53% and 70.71%. Similar to the previous results, the standard deviation values for scenario B were smaller than those of scenario A.

6.4. Combined Recommendation

In this section the overall recommendation of our framework is evaluated. As described in section 3.5, different functions may be used to combine the predictions of all three techniques. Two different types of functions have been evaluated: a weighted sum and a maximum function. As a reference, a simple recommender system was developed that issues predictions based on the overall usage of applications: the more an application was used in the past, the higher it ranks in the recommendation list. Figure 9 compares this simple recommender, referred to as *Popular* due to its nature of recommending only popular applications, with the results of the weighted sum and maximum classification functions of our framework. As both classification functions also take the results of the short time

window algorithm into account, their performance is also subject to the size of the short time window. In Figure 9, a window size of length 10 was chosen for both classification functions. As the association rules already performed exceptionally well, this algorithm was also added to the comparison chart. All values are average IP values over 20 experiments, while minimum and maximum values are indicated by the error bars.

It is shown that our framework, irrespective of the combined prediction function or evaluation scenario, clearly outperforms the popular recommender classifier. When comparing the weighted sum with the maximum function, the latter performs slightly better than the former for scenario B. More specifically, for IP-3 the maximum function has an average gain of 4.3% over the weighted function. For scenario A, the weighted function performs overall slightly better. The impact of the association rules is also clearly shown in this Figure. More specifically, the maximum function has a rather similar performance than the association rule algorithm. The main difference between both approaches is that the maximum function reduces the more extreme performance results of the association rules. This results in higher minimum performance values, while the maximum values still approximate those of the association rules.

Figure 10 shows the bin level performance of all four algorithms for scenario A (left) and scenario B (right) and clearly acknowledges the direct correlation between the association rules and the maximum classification function. Only for scenario A, from bin 3 up to 6, the maximum function is slightly performing better than the association rules approach. Furthermore, it is shown that both algorithms have a much higher percentage of true positives in Bin 1 in comparison to the weighted function, but their performance is more rapidly decreasing with increasing bin number while the weighted function has a slower gradual decreasing curve.

While the difference between the two classification functions and the association rules approach is relatively small, it should be noted that for our evaluation the implementation of the profile-dependent patterns was slightly simplified by not taking time changes into account as these could not be modeled by the modeling tool. This implies that the training and evaluation data were missing a part of one common application usage pattern. A pattern that cannot typically be detected by association rules. When this missing pattern part would also have been integrated in the evaluation data, the obtained scores of the association rules would have been (slightly) lower while the impact on the scores of the combined classification methods, both weighted sum and maximum, should at least be less negative, or even positive.

Table 10 provides an overview of the performance of both the weighted and maximum classification functions for different short time window sizes. With respect to the IP-6 performance, the window size clearly has a very small impact, both on average performance and on minimum and maximum values.

However, for Bin 1 and IP-3 performance, the application of a shorter window size clearly produced worse results.While the opposite results were found when only taking the short time window algorithm into account, as described in Section 6.3, this can be explained by the added value of the short time window in comparison to the two other algorithms. For short window sizes, the short time window's knowledge is very fast only related to applications that were used during the currently active profile. Within such a profile, the two other pattern algorithms also take all this information into account. As a result, the shorter the window size, the larger the percentage of overlap and the less added value that is produced by the short time window algorithm. For longer window sizes, the knowledge is longer (partially) based on application usage from the previous active profile cluster. This can be a main advantage over the other algorithms, especially when applications are suddenly very actively used, but are not yet much known to a context profile that has just been activated.

As shown in Table 10, the highest obtained result of all experiments was an IP-6 of 90.76%, produced by a weighted classification function with a short time window size of length 10. When only taking the top 3 recommendations into account, a maximum classification function with window size 10 was able to recommend the correct application in 67.94% of the cases during one evaluation run. Maximum classification with window size 20 scored the highest Bin 1 percentage with 32.70%. All maximum values were found for scenario A. The minimum percentages were found during the application of the weighted classification function for scenario B, with an IP-6 of 71.73% for window size 20 and and an IP-3 of 45.80% for window size 5. The standard deviation was relatively low for all experiments, while the smallest margin after normalization was found for the minimum IP-3 values, still showing a 3% margin.

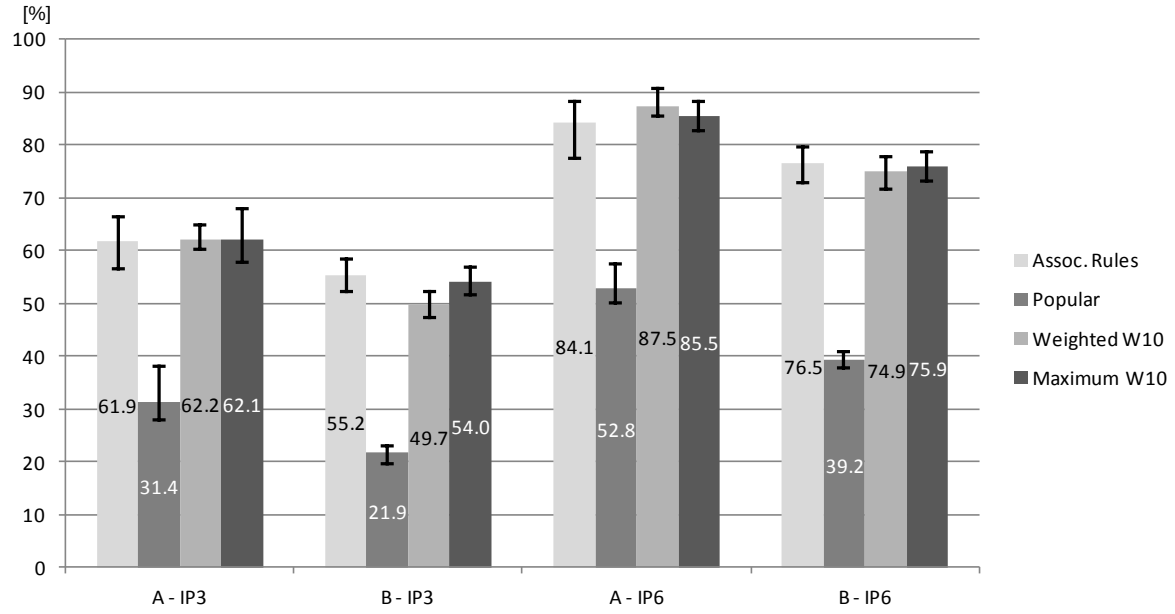


Fig. 9. Combined recommendation algorithm evaluation: IP-3 and IP-6 performance for scenarios A and B and short time window size 10.

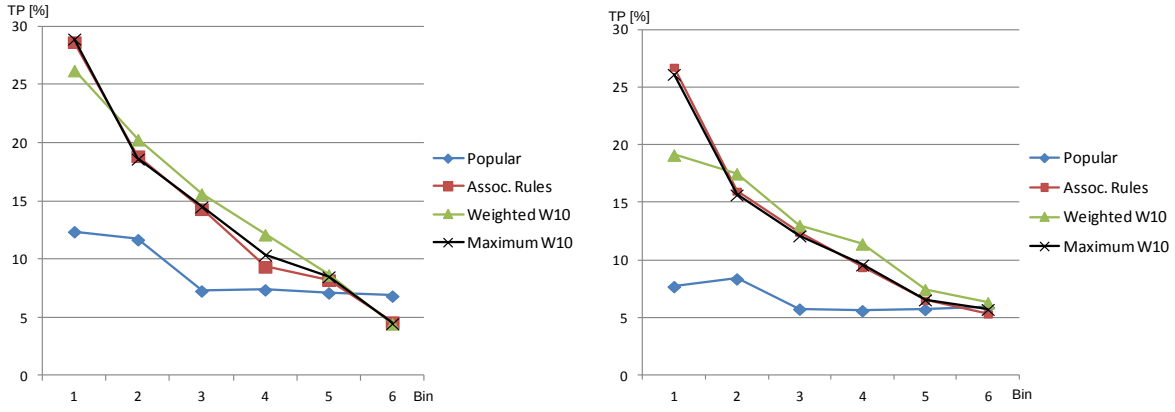


Fig. 10. Combined recommendation algorithm evaluation: average percentage of true positives per bin for scenarios A (a) and B (b) and short time window size 10.

6.5. Evaluation conclusions

To conclude the analysis of the experiment results, it can be stated that the combination of first EM (Expectation-Maximization) clustering followed by the application of association rules, produces a very high rate of true positives for different scenarios. The overall recommender system further improves this performance, especially when higher short time window sizes are used. When only taking the top 1 or top 3 recommendations into account, the maximum function produces the best results, while the weighted function performs better for longer recommendation lists in combination with user profiles that were not too complex. For more complex profiles, both classification functions perform equally well for longer recommendation lists.

7. Conclusions

In this paper, a framework was presented for the automatic prediction of a user's future mobile application

Table 10
Combined recommendation algorithm evaluation: Bin 1, IP-3 and IP-6 performance of the weighted and maximum classification functions for different short time window sizes.

	Bin 1			IP-3			IP-6		
	Avg.	**Min.**	**Max.**	**Avg.**	**Min.**	**Max.**	**Avg.**	**Min.**	**Max.**
Scenario A									
W5	24.90	22.14	29.29	60.78	58.85	64.24	87.66	85.77	90.66
Weighted W10	26.24	23.89	29.45	62.18	60.11	64.84	87.47	85.40	90.76
W20	26.73	24.05	29.29	62.46	60.53	65.04	87.30	84.80	89.76
W5	28.13	24.97	32.36	60.87	57.18	63.86	86.15	83.78	88.87
Maximum W10	28.95	25.38	32.40	62.10	57.74	67.94	85.51	82.79	88.43
W20	29.04	25.66	32.70	62.20	58.16	67.64	85.45	82.43	88.58
Scenario B									
W5	16.90	14.07	18.97	49.03	45.80	52.89	75.19	72.24	77.89
Weighted W10	19.16	15.86	21.62	49.71	47.36	52.25	74.94	71.54	77.89
W20	20.41	17.57	23.23	50.71	48.37	52.89	75.04	71.73	78.05
W5	25.10	22.88	27.23	53.04	50.37	55.79	75.93	72.55	78.39
Maximum W10	26.16	24.57	28.51	53.98	51.53	56.86	75.89	73.09	78.78
W20	26.52	24.88	28.51	54.54	52.49	56.86	75.91	73.09	78.47

usage. The described framework works fully transparently to the user as no additional user interaction is required and all types of users are supported as no predefined scenarios or assumptions are made. For this purpose, the framework learns from the user's previous mobile application usage and his previous context situations which are both continuously monitored by the framework's client application. The Expectation Maximization clustering algorithm is then applied on different combinations of context data, resulting in an undefined number of clusters that clearly approaches users' real-life perception of different context situations. After this first phase, three different algorithms are performed, each specifically designed to find or support its own specific application usage patterns and to predict which application shall be used based on the current context situation. Association rules are used to find correlations between contextual parameters and used applications. Profile-related graphs are developed to find profile-dependent patterns and a short time window is applied for analyzing the most recently used applications. Finally, the predictions that are made by all three algorithms are bundled into one final rating for each application and for each new context situation. For the evaluation of these three separate algorithms and the final combined ratings a proper modeling tool and evaluation metric were used. After a profound analysis based on two different user scenarios, it is shown that the application of association rules resulted in a very high ratio of true positives, while the combined classification functions were able to further optimize these scores. When only taking the top 1 or top 3 recommendations into account, the maximum function produced the best results. The weighted function performed best for longer recommendation lists in combination with user profiles that are not too complex while for more complex profiles, both classification functions performed equally well for longer recommendation lists. Finally, to further demonstrate the applicability of the framework, an Android-based mobile prototype application was developed, consisting of a dynamic home screen widget showing the application icons of the top six predicted applications, together with the most recently called contact person and the previous phone settings (e.g. ringing mode) for the currently active context situation.

References

[1] A. P. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[2] O. Abu Abbas. Comparisons between data clustering algorithms. *International Arabian Journal of Information Technology*, 5(3):320–325, 2008.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] A. Bikakis, T. Patkos, G. Antoniou, and D. Plexousakis. A survey of semantics-based approaches for context reasoning in ambient intelligence. In M. Mühlhäuser, A. Ferscha, and E. Aitenbichler, editors, *Constructing Ambient Intelligence*, volume 11 of *Communications in Computer and Information Science*, pages 14–23. Springer Berlin Heidelberg, 2008.

[5] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, Sept. 2003.

[6] P. D. Costa, L. F. Pires, M. V. Sinderen, J. Gonçalves, and P. Filho. Towards a service platform for mobile context-aware applications. In *Ubiquitous Computing - IWUC*, pages 48–61, 2004.

[7] A. S. Crandall and D. J. Cook. Using a hidden markov model for resident identification. In *Proceedings of the 2010 Sixth International Conference on Intelligent Environments*, IE '10, pages 74–79, Washington, DC, USA, 2010. IEEE Computer Society.

[8] A. K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, Jan. 2001.

[9] A. Emrich, A. Chapko, and D. Werth. Context-aware recommendations on mobile services: the m:ciudad approach. In *Proceedings of the 4th European conference on Smart sensing and context*, EuroSSC'09, pages 107–120, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] D. Figo, P. C. Diniz, D. R. Ferreira, and J. M. Cardoso. Preprocessing techniques for context recognition from accelerometer data. *Personal Ubiquitous Computing*, 14:645–662, October 2010.

[11] J. A. Flanagan. Clustering of context data using k-means with an integrate and fire type neuron model. In *Proceedings of Workshop on SelfOrganizing Maps*, pages 17–24, 2005.

[12] M. Hall, F. Eibe, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[13] L. G. Heins and D. R. Tauritz. Adaptive Resonance Theory (ART): An Introduction, 1995.

[14] Y. Her, S.-K. Kim, and Y. Jin. A context-aware framework using ontology for smart phone platform. *JDCTA*, 4(5):159–167, 2010.

[15] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[16] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.

[17] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464 –1480, sep 1990.

[18] P. Korpipää, M. Koskinen, J. Peltola, S.-M. Mäkelä, and T. Seppänen. Bayesian approach to sensor-based context awareness. *Personal Ubiquitous Comput.*, 7:113–124, July 2003.

[19] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E. J. Malm. Managing context information in mobile devices. *Pervasive Computing, IEEE*, 2(3):42–51, 2003.

[20] U. Naeem and J. Bigham. Activity recognition in the home using a hierarchal framework with object usage data. *Journal of Ambient Intelligence and Smart Environments (JAISE)*, 1(4):335–350, 2009.

[21] M. Roininen, E. Guldogan, and M. Gabbouj. Audiovisual video context recognition using svm and genetic algorithm fusion rule weighting. In *Content-Based Multimedia Indexing (CBMI), 2011 9th International Workshop on*, pages 175 –180, june 2011.

[22] D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9(1):75 – 112, 1985.

[23] A. Shankar and S. J. Louis. Xcs for personalizing desktop interfaces. *Trans. Evol. Comp*, 14(4):547–560, Aug. 2010.

[24] A.-H. Tan and H.-S. V. Soon. Predictive adaptive resonance theory and knowledge discovery in databases. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, PADKK '00, pages 173–176, London, UK, UK, 2000. Springer-Verlag.

[25] D. Wyatt, M. Philipose, and T. Choudhury. Unsupervised activity recognition using automatically mined common sense. In *In AAAI*, pages 21–27, 2005.

[26] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, July 2002.

[27] J. Ye and S. Dobson. Exploring semantics in activity recognition using context lattices. *J. Ambient Intell. Smart Environ.*, 2:389–407, Dec. 2010.

[28] Z. Yu, X. Zhou, D. Zhang, C.-Y. Chin, X. Wang, and J. Men. Supporting context-aware media recommendations for smart phones. *IEEE Pervasive Computing*, 5:68–75, July 2006.