Inferring the Everyday Task Capabilities of Locations

Patricia Shanahan, William G. Griswold

University of California, San Diego

Abstract. People rapidly learn the capabilities of a new location, without observing every service and product. Instead they map a few observations to familiar clusters of capabilities. This paper proposes a similar approach to computer discovery of routine location capabilities, applying machine learning to predict unobserved capabilities based on a combination of a small body of local observations and a larger body of data that is not specific to the location. We propose using the time and place of deleting items from a to-do list application to provide the local data. For reminder purposes, an area within easy walking distance is a single location, but may contain many different shops and services, collectively offering its own combination of capabilities. Truncated singular value decomposition maps the observations to combinations of features, rather than to a single cluster. Simulations, using distributions derived from real world data, demonstrate the feasibility of this approach.

1 Introduction

Everyday life includes many tasks that cannot be performed at arbitrary times and places. Some tasks require a specific place, such as the conference hotel for a given conference. Others require places with a specific relationship to the user, such as “my home” or “my office”. Some tasks depend only on the context having a specific capability, such as offering a given service or selling a given product. Any location with similar capabilities is a possible location for the task. Shopping is a typical example of this kind of task.

The following scenario illustrates how a future reminder system might support context-capability dependent tasks. Joe is a college professor and a user of a reminder system, a few years in the future.

While doing his laundry, Joe notices there will not be enough detergent for next week. He runs his phone’s barcode scanner over the detergent box’s product code. The phone adds detergent to its shopping list. Postage stamps are already on the list. Joe has also made a note that he needs access to a copy of the Oxford English Dictionary (OED), to check the history of a word.

The next day, Joe happens to walk past his college’s reference library. His phone reminds him of the word he wanted to look up in the OED.

A couple of days later, Joe’s wife, Alice, phones him to say that she is running late at work, and asks him to buy some food items. Because of where he is, and the time of day, he drives to a different supermarket than he normally shops at.

Nevertheless, his phone vibrates as he nears the supermarket. He looks at it, and sees text reminding him to buy detergent and postage stamps.

Without the reminders, Joe could have easily forgotten some of these tasks. The literature on prospective memory — on remembering to do things that cannot be done here and now — assumes that people do sometimes forget such tasks. For example, Farrimond et. al. mention the task of buying bread on the way home from work: “It is, however, a common experience to arrive home without the bread.” [1].

This matches our personal experience. Indeed, one of us, wishing to consult a paper-only reference on prospective memory, printed out the bibliographic information and library call number, got distracted on the way to the printer by the arrival of an elevator, and did not remember to pick up the printout until outside the computer science building on the way to the library.

Most quantitative research on prospective memory uses artificial laboratory tasks, but a few do attempt to model real-world tasks. Farramond et. al. use a laboratory model of shopping tasks. Their younger control group fail to remember about 4% of the time [1]. However, comparison of similar tasks in a laboratory model and in the real-world can get different results [2].

The library reminder is relatively easy. Joe has previously carried out similar tasks at the same place. A smart reminder application that notes time and location of to-do list deletions would know where he can do that task.

Joe faces a harder problem than remembering to buy the detergent and stamps at times and places he had planned. To take advantage of the unplanned trip, he not only has to know that his location offers the items, he has to remember that fact, and his intention of buying them, while thinking about another errand.

Joe may never have bought either detergent or stamps at that location, so his phone cannot depend only on the information it can collect. It has to function as part of a larger system, as shown in Figure 1. A server can be dedicated to a single user or small group of users, or be public and widely shared. Each client device is associated with one user. Joe’s deletion of “detergent” from his shopping list itself supplies data; confirmation that the location has detergent-supplying capability.

The scenario is set in the near future, not today, because it requires several advances in technology:

– Joe’s phone generally knows where it is, with sufficient accuracy for Joe to be able and willing to take actions based on his phone’s estimate of its location.
– The phone can decide whether Joe should be interrupted now, given an opportunity to complete a task.
– Joe’s phone can interpret bar codes with high accuracy.
– The phone has access to a server that maps from a required capability to a list of nearby locations with the capability.

The last would allow Joe’s phone access to a wider body of information. Given informed consent, that might include pooled to-do list deletions from many users. However, even a small shopping center may offer thousands of products and

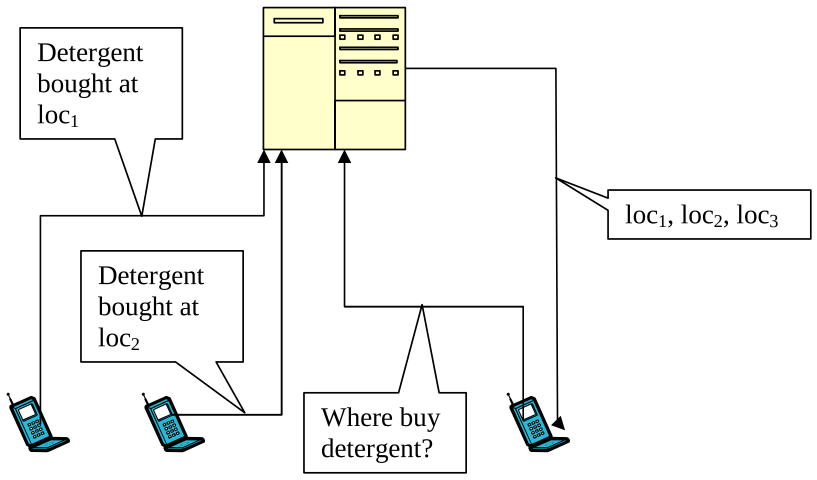
Detergent bought at loc_1
Detergent bought at loc_2
Where buy detergent?
loc_1, loc_2, loc_3

Fig. 1. A server receives reports of detergent having been bought at locations loc_1 and loc_2. loc_3 is a third location that has been found to be similar to the other two. When Joe's phone asks where detergent can be bought the server responds with all three locations.

services. A system that depended exclusively on to-do list data, even if pooled from multiple users, would require an impractically long time to accumulate a useful body of data.

This problem is the focus of this paper. Fortunately, businesses do not make completely arbitrary choices of which products to stock and services to offer. There are clusters of products that are commonly sold together. People often use those patterns to predict which shop sells what without memorizing a list for each shop, albeit imperfectly. If experience indicates that a location is a source for dishwasher detergent and fabric softener, it is probably a valid location for Joe's laundry detergent purchase, even if there is no direct evidence.

This paper proposes applying machine learning methods to extract a mathematical representation of those patterns from a body of capability data. Ideally, if the data includes even a relatively small amount of information about a specific location, the method will use those patterns to infer its other capabilities. After experimenting with a number of approaches, we selected truncated singular value decomposition (T-SVD).

The capability prediction problem is complicated by the fact that a single location, such as a shopping center, may offer a wide range of products and services because it contains multiple businesses. Even if location detection were accurate enough to distinguish them, a reminder should be issued if a required capability is available within easy walking distance. Similarly, a shop like a supermarket combines the capabilities of a butcher, a bakery, a pharmacy, etc. T-SVD can theoretically cope with this complexity because it maps purchases at locations to combinations of capability clusters, rather than assigning the location to a unique cluster. This factoring also enables bootstrapping the training by conveniently supporting the infusion of summary capability clustering data.

Since different capabilities have different frequencies of use, and those frequencies may affect the accuracy of the algorithm, we assessed our approach using actual shopping basket data obtained from a major supermarket chain. The algorithm was effective if either the locations had simple behavior or we supplied summary capability clustering data to supplement data about the locations. In particular, with summary capability data and given 1000 observations of 135 simulated locations, equivalent to an average of 7.4 observations per location, the algorithm achieved 100% precision and 90% recall.

The next section surveys data sources for generating capability-based reminders. Section 3 develops our proposed approach. Section 4 describes our experiments, and the final section concludes.

2 Limitations of Location Capability Data Sources

There are several data sources that can be used for location capability-based reminders. Although each data source on its own has limitations for providing reminders for everyday activities, they can be used to train a machine learning algorithm for inferring place capabilities.

2.1 Direct Input of Individual Information

The simplest method of selecting a location for a reminder is to have the user name a location or point to it on a map. In effect, this method requires the user to do the mapping from capability requirements to locations having those capabilities.

Joe could have used a direct location-based reminder system, such as Place-Its, to remind himself about the detergent next time he is at his usual supermarket [3]. However, Place-Its would only remind him at the planned location, not at other locations with similar capabilities.

CybreMinder has a situation description language with power similar to a database query language [4]. Locations can be referred to by symbolic names such as “Bob’s front door”. This is likely to be most effective if each user has a limited number of known reminder locations or groups of locations.

Although CybreMinder’s situation language may be more complicated than is appropriate for non-expert use, it would be useful to apply time restrictions to a reminder. For example, a user whose shopping list includes perishable foods would probably not wish to be reminded of it while driving past a supermarket on the way to work. The reminder would be useful, at the same location, when the user is returning home.

The main limitation of direct input is the sheer number of places that have some capabilities. A reminder system is not useful if the work of entering the data exceeds the benefit of the reminders. Yet it may be the only solution for a user in a new area, if the system does not have any local information.

2.2 Automated Entry of Individual Information

The use of a to-do list for tracking activity, as proposed in this paper, is an example of automated entry of individual information. Every task completion is a potentially observable event. With the user's consent and suitable privacy protections, the application on the phone could collect the time and location of a task check-off, and supply the data to its server. The privacy protections could include lower bounds on elapsed time and change in location between the event and its transmission to the server, so that the server would never be told the user's current or very recent location. This data collection method presents minimal cost to the user, and is applicable to non-commercial tasks.

Mankoff et. al. use cash register receipt scanning as a convenient way to help people keep track of the nutritional characteristics of the mix of foods they buy, and presumably eat [5]. Their technique could be adapted to capture the shopping data from the receipt.

Such approaches have the major advantage of capturing the information that most affects the individual user, with minimal user effort. The user's favorite shops and services will definitely be covered. However, as the sole data source, information accumulates slowly, and cannot supply any data the user did not already know or discover by other means.

2.3 Localized Web Searches

Location-limited web searching has considerable promise for finding locations by capability, but does not yet work well for many common tasks.

In practice, the user will often know the types of facilities, such as shops, that are appropriate for completing a task. A "Find businesses" search on Google Maps (maps.google.com) for "supermarket" near zip code 92126, on March 14th. 2007 got the following top results:

- A sponsored link to the home page for a supermarket chain
- A link for "Seafood City", which is a supermarket with an emphasis on seafood.
- A link to a supermarket chain pharmacy that does not exist at the indicated location.
- A link for a specialized Vietnamese market
- A link for "Slimmer Body Mall"

The nearby stores of three major supermarket chains were not shown.

The problem is harder if the user does not already know the appropriate types of facilities for completing a task, or if there are so many that entering them takes too long. Querying "Find businesses" on Google Maps for "instant coffee" near zip code 92126 was converted to "Categories: Motels & Hotels, Health & Diet Food Products", and again failed to find the local supermarkets and convenience stores at which most people would buy instant coffee.

Other web searches for common items have produced similar results. Exotic, rare, and expensive items are well represented. Large facilities such as major chain stores are well represented. Trivial items that everybody knows about such as instant coffee are not. Small facilities, such as mini-marts may be represented, depending on owner initiative and whether they are part of a chain.

2.4 Web-Accessible Database Searches

Many organizations provide web pages that in essence can query a database. For example, the US Postal Service web site (www.usps.gov) supports a search for places that sell postage stamps near a given address or zip code. The problem for a reminder system is navigating the web pages to get to the stamp-buying search. A program can be written to navigate any specific set of web pages to obtain specific data, but it would have to be done for each web site, and could stop working at any time if the web site were modified.

This problem could be ameliorated by having standard interfaces for machine-based search of location-based information. For example, the Impulse Location-based Agent Assistance project currently uses location-limited URL searches and the Wherehoo server (wherehoo.org), to identify nearby locations that might satisfy the user's "wants" [6, 7]. The main difficulty is the bootstrapping problem of accumulating enough initial data to attract enough users to motivate database providers to supply data for low value tasks.

Nonetheless, databases are an attractive source for bootstrapping a machine learning algorithm. As our experiments presented later show, even a small set of accessible databases can provide useful clustering information.

3 Proposed Inference System

We propose applying the machine learning process of truncated singular value decomposition (T-SVD) to raw location-based capability data, including some general background data, to establish patterns of association between capabilities and at least a little data about each location of interest.

This proposed process is analogous to one way that many people reason about a location: Suppose Alice needs some laser printer paper. She sees a new store, with a name she does not recognize, but with posters in the windows advertising special offers on ball point pens, highlighters, ink jet printer cartridges, photocopying, and file folders. Alice uses her general experience of places offering those products and services to deduce the probable presence of an office supply store and expects to be able to buy her paper there.

Now instead suppose Alice has laser printer paper on her *digital* to-do list. Betty, another user of the system, previously visited the new store. While there, she checked off ball point pens, a highlighter, an ink jet printer cartridge, a photocopying task, and file folders. If Alice's phone depends only on directly collected data from previous shoppers like Betty, there is insufficient information for the phone to issue a reminder.

How much data would be enough, if Alice's phone had access to inferences from both the local data and a larger body of data from many other locations?

To answer that question, we built a model of location capabilities and used it to test a number of inference algorithms, eventually settling on truncated singular value decomposition.

The next two subsections describes our data model and algorithm. The final subsection outlines a possible architecture for a scalable system using this approach.

3.1 Data Model

The data model initially assigns a weight, representing the number of observations, to each combination of location and capability. The weights form a matrix, with one row for each capability and one column for each location. For example:

Table 1. Training Data

	Pete's Pets	Happy Paws	Hal's Hardware	Mike's	Stuff
Dog Food	40	153	0	0	4
Cat Food	26	95	0	0	0
Bolt	0	0	203	3	0
Nail	0	0	100	4	0
Screwdriver	0	0	23	2	0
Hammer	0	0	45	0	0

Here, each value represents a numbers of observations. For example, the data contains 153 observations of dog food selling at Happy Paws.

The frequencies of use of different capabilities at a location are useful data for predicting other capabilities. Generally, heavily used capabilities represent core characteristics, and the location is likely to continue to have those capabilities, as well other capabilities normally associated with them. On the other hand, a single observation of a capability may be an anomaly or a data entry error.

The location and product names are for reader convenience. The locations will typically only be known by GPS or similar coordinate. The product names may be product codes. Of course, a real set of training data would be much larger, with possibly tens of thousands of products and, even in small test cases, dozens to hundreds of locations. Normally, there are many more products and services than locations.

The data in this example suggests that cat food and dog food are normally sold at the same location, so it is reasonable to expect Stuff to sell cat food, despite the lack of any direct observations. A system that only uses actual history, without any generalization, would not trigger a reminder for cat food at Stuff.

This data structure can also represent seed data deduced from other data sources. A row represents a relationship among locations. A column represents a relationship among capabilities. Suppose a data source associated the label “pet store” with Pete's Pets, Happy Paws, and Stuff, and that we assigned that data source a weight of 3, given its general reliability. We would add a row containing

3 for each of those locations, and 0 in all other entries. Similarly, if we assigned weight 5 to an externally supplied list of hardware items, we would add a column with 5 for those items, and 0 for the remaining rows:

Table 2. Extended Training Data

	hardware	Pete's Pets	Happy Paws	Hal's Hardware	Mike's	Stuff
pet store	0	3	3	0	0	3
Dog Food	0	40	153	0	0	4
Cat Food	0	26	95	0	0	0
Bolt	5	0	0	203	3	0
Nail	5	0	0	100	4	0
Screwdriver	5	0	0	23	2	0
Hammer	5	0	0	45	0	0

3.2 T-SVD Algorithm

The objective of the algorithm is to infer a list of capabilities for each location that could have caused the observations, under the assumption that most of the variation between locations can be explained by a relatively small number of factors, such as the presence or absence of a bakery, as either a shop or a department in a larger shop. We map from our specialized ubiquitous computing problem into a common problem in linear algebra, find a low-rank approximation to the matrix, and then use truncated singular value decomposition, T-SVD, to solve it.

T-SVD has been applied successfully to similar linear algebra problems from other domains. For example, Deerwester et. al. applied it to text analysis [8]. Cernekova et. al. use it for detecting video shot transitions [9]. Chu notes that, if each column of X represents an unpredictable sample of a certain unknown distribution, the truncated singular value decomposition "not only is the best approximation to X in the sense of norm, but also is the closest approximation to X in the sense of statistics." [10].

T-SVD, as well as compressing data, may reduce various forms of noise that a full SVD would preserve. Our input data may have a zero for an available product because nobody has happened to report a purchase of that product at that location. It may also include false positives, because someone deleted an item from their to-do list for a reason other than completion of that task at the current location. T-SVD, by producing a lower rank approximation, removes detail that is more likely to be due to sampling or errors.

The first step in the algorithm is to build a matrix representing the data. Each location is represented by a column of the matrix containing the observed capability weights. For instance, using our original example data without the added "hardware" and "pet store" data, the column vector for "Pete's Pets" is

(40, 26, 0, 0, 0, 0). Similarly, each capability is represented by a row. The matrix for the example data is:

$$X = \begin{pmatrix} 40 & 153 & 0 & 0 & 4 \\ 26 & 95 & 0 & 0 & 0 \\ 0 & 0 & 203 & 3 & 0 \\ 0 & 0 & 100 & 4 & 0 \\ 0 & 0 & 23 & 2 & 0 \\ 0 & 0 & 45 & 0 & 0 \end{pmatrix} . \tag{1}$$

Despite having six rows and five columns, X only has rank four. Each of the last four rows can be generated by a linear sum of any two of them. However, if the basic hypothesis of product clustering is correct, X is a noisy, sampled approximation to a rank 2 matrix, where each column is a linear sum of a column representing an idealized pet food store and a column representing an idealized hardware store.

The general strategy is to use T-SVD to calculate a lower rank matrix Y, rank k, that is the least squares closest rank k matrix to X, and then use $Y_{i,j}$ as a score for predicting whether location j has capability i. Full singular value decomposition factorizes a matrix into USV^T such that each of U, S, and V has some useful properties, conveying information about the underlying structure of the original matrix. Nicholas and Dahlberg's work on finding topics in documents is particularly relevant [11]. Their problem is similar to ours, if we map "capability" to "term", "location" to "document", and treat a cluster of capabilities that tend to appear together, such as pet food in the example, as a "topic".

S is a diagonal matrix containing the "singular values", a list of weights related to the importance of different factors in forming the original matrix. After performing full singular value decomposition using Matlab "[U S V] = svd(X)", S is a diagonal matrix with values (231.9131, 186.3347, 2.8800, 2.3201, 0). There are only four non-zero singular values because X is rank 4, despite having 5 columns.

The full SVD result retains all noise and sampling effects in the data. The next step is to truncate, keeping only the largest k singular values. In our current algorithm, k is supplied externally. For this example, we will use $k = 2$. Note that the first two singular values, 231.9131 and 186.3347, are relatively close, but the third singular value, 2.88, is much smaller. Future work will examine selecting the truncation rank based on the singular values.

The truncation to k singular values results in a rank k approximation. The truncation could be done by running the full SVD and extracting the data relating to the first two singular values, but for large matrices it is more efficient to only obtain the required values, for example by running Matlab "[U S V] = svds(X,2)". The actual results contain several values of absolute magnitude less than 10^{-14}, due to floating point rounding error. Those numbers have been replaced by zero for clarity:

$$\overset{U}{\begin{pmatrix} 0 & -0.84895 \\ 0 & -0.52848 \\ 0.87541 & 0 \\ 0.43145 & 0 \\ 0.099325 & 0 \\ 0.194 & 0 \end{pmatrix}} \quad \overset{S}{\begin{pmatrix} 231.91 & 0 \\ 0 & 186.33 \end{pmatrix}} \quad \overset{V}{\begin{pmatrix} 0 & -0.25598 \\ 0 & -0.96651 \\ 0.99981 & 0 \\ 0.019622 & 0 \\ 0 & -0.018224 \end{pmatrix}} \qquad (2)$$

We next calculate Y, the rank 2 matrix approximation to X, by multiplying the factors:

$$Y = USV^T = \begin{pmatrix} 40.493 & 152.89 & 0 & 0 & 2.8828 \\ 25.208 & 95.176 & 0 & 0 & 1.7946 \\ 0 & 0 & 202.98 & 3.9837 & 0 \\ 0 & 0 & 100.04 & 1.9634 & 0 \\ 0 & 0 & 23.03 & 0.452 & 0 \\ 0 & 0 & 44.983 & 0.88284 & 0 \end{pmatrix}. \qquad (3)$$

Finally, we map the results back to our original problem domain, and use them as scores for estimating location capabilities:

Table 3. Scores

	Pete's Pets	Happy Paws	Hal's Hardware	Mike's	Stuff
Dog Food	40.493	152.89	0	0	2.8828
Cat Food	25.208	95.176	0	0	1.7946
Bolt	0	0	202.98	3.9837	0
Nail	0	0	100.04	1.9634	0
Screwdriver	0	0	23.03	0.452	0
Hammer	0	0	44.983	0.88284	0

All elements that were non-zero in Table 1 are non-zero in Table 3. Two additional elements are non-zero, representing cat food at Stuff and hammers at Mike's. The decision to issue a reminder will be made by comparing a cutoff value to the value in Table 3 for the combination of location and capability. The cutoff has to be somewhat greater than zero to avoid false positives due to rounding errors. For any cutoff below 0.452, the system would predict cat food at Stuff and hammers at Mike's, despite the lack of direct observations.

The approach can be extended to make predictions about a location that was not in the original training data, given some data about it, without relearning the result matrix. First we observe that any column of Y can be reproduced by multiplying together U, U^T, and the corresponding column of X:

$$UU^T \begin{pmatrix} 40 \\ 26 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 40.4933 \\ 25.2076 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4}$$

This operation, applied to a new or updated location column, produces an estimate of the location's capabilities. Suppose we have seen one cat food purchase and one hammer purchase at Totally Square, which could happen if Totally Square is a shopping center containing both a pet store and a hardware store. The corresponding location vector is (0, 1, 0, 0, 0, 1). Then:

$$UU^T \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.4487 \\ 0.2793 \\ 0.1698 \\ 0.0837 \\ 0.0193 \\ 0.0376 \end{pmatrix}. \tag{5}$$

Equation 5 predicts both hardware and pet food at Totally Square, but less strongly than for the locations with more data.

If the Totally Square data had been in the original training set, the algorithm would also have replaced some zero elements in the original result with slightly positive scores, reflecting the possibility that each location has both hardware and pet food, or that hammers are a form of pet food. This behavior creates some anomalies for the example data — Pete's Pets is unequivocally a pet food store and does not sell hammers — but may be a good reflection of the uncertainties and overlaps of the real world.

The algorithm has two parameters that are currently specified externally, the cutoff score and the truncation rank. It may be useful to adjust the cutoff according to a user selected trade-off between being notified of all possibilities, at the risk of false alarms, or being notified only when there is near certainty of a required capability. The correct choice may depend, for example, on how urgently the user wants to complete a particular task, and on the cost of an alert, given the user's context and activities.

The truncation rank is an estimate of the number of distinct factors, such as presence or absence of a facility such as a bakery, that are required to explain the data. Too low a truncation rank destroys data about real differences between locations. Too high a truncation rank preserves too much detail. At the extreme, keeping all non-zero singular values results in Y being identical, within rounding error, to X. Further work will aim to automate the truncation rank determination, and to normalize the scoring so that a given cutoff value has similar effects regardless of the volume of training data.

3.3 Scalability and Architecture

We have described the proposed system in terms of how it would be used, and have described the key algorithm. This subsection discusses how the architecture can be adjusted to scale to large problem sizes.

The configuration in Figure 1, with a single server, is sufficient for small- to medium- scale operations. Part of the workload, responding to queries and collecting data, is embarrassingly parallel. The volume of data required by each server could be limited by splitting transactions geographically.

However, the T-SVD calculation should be done with as much data as possible, so it must be centralized. Figure 2 illustrates this architecture.

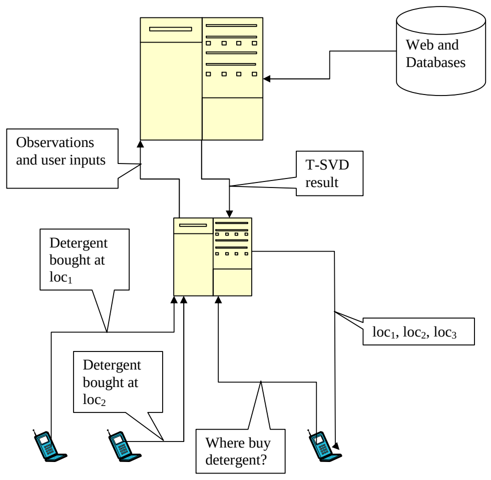
Web and Databases
Observations and user inputs
T-SVD result
Detergent bought at loc_1
loc_1, loc_2, loc_3
Detergent bought at loc_2
Where buy detergent?

Fig. 2. The front-end server that communicates with Joe's phone also communicates with a back end inference engine server. The inference engine collects data from front-end servers and other sources. Periodically, it distributes updates to the front-end servers.

Suppose there are c capabilities and l locations. There may be thousands of each. The input matrix X is has cl elements, but is sparse, having a non-zero entry only for observed combinations of location and capability.

The output matrix is the same size, and may be less sparse, but does not need to be stored. As noted in the description of the algorithm, elements of it can be calculated from X and U.

Each front-end server needs a copy of the U matrix. It has ck entries, where k is the truncation rank. For 25,000 capabilities and $k = 40$, the U matrix would have one million entries and could be stored in 8 MB. A front-end server also needs the observed capability vector for each location it serves. The number of observed capability vectors could be limited by dividing work geographically, and the vectors are sparse.

Although runtime cost has not been a significant factor in our initial testing, the cost of the T-SVD calculation will tend to increase with data volume. However, the wide use of SVD has resulted in extensive work on efficient solution techniques. For example, Lin has developed an out-of-core method. [12]. In a large version of the system, the inference engine server could be effectively scaled by giving it additional memory and compute power.

The combination of geographic division of the front-end work, the gains in SVD performance from compute and memory size gains, and the potential for more sophisticated SVD methods if needed ensure that the system can scale.

4 Evaluation

We conducted simulation experiments to test the hypothesis that machine learning could fill in gaps in the observations, as well as merge different types of data.

We considered two types of input data, small numbers of random observations, such as might result from to-do list deletions, and seed data derived from bulk lists of products that are commonly stocked together, such as might be extracted from on-line databases. We tested two simulated environments, “Simple” and “Compound”. In the Simple environment, each location has the capabilities of a single cluster. The Compound environment, some locations have multiple clusters of capabilities, including some that exhibit all eight clusters. In each case, we compared results to a “Null” learner that reports availability of a capability at a location if, and only if, the training data contained that combination. The traditional measures of precision and recall were used to evaluate the results.

Two important questions remain for future studies:

Group structure These experiments are all based on eight disjoint product groups. Future experiments will use more groups, and include groups that overlap, with some products in more than one group.

False data The training data does not include any false reports of purchases at locations that do not stock the product.

A US national supermarket chain (who asked to remain anonymous) supplied shopping basket data. The data includes classification of products into major groups. These groups provide a model of related products, and were used to model different types of shops, with unique capabilities. For our experiment, we selected the eight groups with the highest total purchase counts. We generated a table of group number and number of purchases for each product in those groups. The table contains over 40,000,000 purchases. Each group represents a set of related products, so we used each group as a model of a specialized shop type.

The training data for each location was sampled from the groups for its shop types, with the probability of each product proportionate to its frequency in the real data. We used the set of products in each group, with no frequencies, as seed data, because lists of groups of related products and services are often publicly available, but frequencies of sales are normally trade secrets.

The “Simple” environment used 8 locations per shop type, 64 locations total, where each location corresponded to a single shop. The “Compound” experiment used a mix of different types of locations. Each location has one or more shop types, as shown in Table 4. There are a total of 135 locations, 64 similar to the “Simple” experiment, the remainder modeling different types of locations, where each type has capabilities from multiple clusters.

Table 4. Compound Location Types

Instances	Shop Types
8	1
8	2
8	3
8	4
8	5
8	6
8	7
8	8
10	1,2
8	1,2,3
15	1,2,4
10	4,5
10	4,5,6
6	1,2,3,4,5,6,7,8
12	1,8

We ran each experiment in two training data modes:

– Randomly generated samples. Each random sample is produced picking a location with equal probability, and then picking one of its capabilities with probability proportionate to the frequency of that capability in the shopping basket data.
– Randomly generated samples, as above, plus seed data derived from bulk lists of the products in each group.

For T-SVD’s two external parameters, we chose 8 for the rank and 10^{-6} for the cutoff. The rank was based on the expectation that at least eight singular values would be needed and an observed lack of improvement for using more. The cutoff was selected to suppress rounding error on values that should be zero.

We also ran a “Null” learner to establish a baseline. The Null learner reports a capability at a location if, and only if, that combination of capability and location appeared in the training data. It does no extrapolation. A learning method must do better than the Null learner to be worth using.

There are two ways a result can be wrong – with different consequences for a reminder system – leading to two quality measures. “Precision” is the proportion of reminder triggers that would be correct, issued at a location that affords the required capability. "Recall" is the proportion of locations affording the capability that would trigger the reminder. As an example, the simple Null learner always achieves perfect precision, at the expense of recall.

In reporting our results, we use unweighted precision and recall, a conservative approach. Some capabilities have frequencies so low that they have practically no chance of appearing in a few thousand field samples, but an error related to one of those rarities (not unlikely with our approach) carries the same error penalty as an error related to a high probability product (unlikely with our approach).

In each of the charts, the "SVD-N" line is the performance of our algorithm without the use of seed data. The "SVD-Y" line is with seed data. "Null" represents the Null learner.

Figures 3 and 4 show precision and recall for the "Simple" data, measured as a function of the number of training data samples. "Seen" in Figure 4 is the proportion of capabilities that have appeared in at least one training data sample. It is an upper bound on the recall that can be achieved without seed data. As can be seen from the chart, SVD-N achieves that bound given at least 2,000 samples, an average of 31.25 samples per location. The main benefit of the seed data in the "Simple" experiment is allowing the system to make projections about capabilities that have never been observed in the training data, based on knowledge that the unseen capability is related to ones that have appeared.

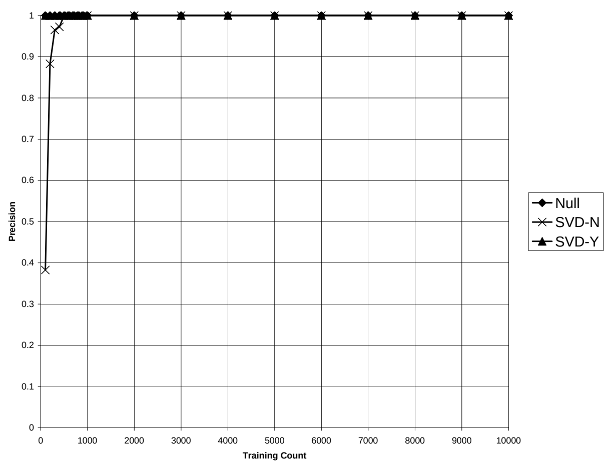

Fig. 3. Precision for simple locations

Figures 5 and 6 show precision and recall for the "Compound" data. For this workload, the seed data is needed to achieve acceptable precision. Without it, the precision is about 40%. With seed data, T-SVD has 100% precision, and exceeds 90% recall given at least 1000 training data samples.

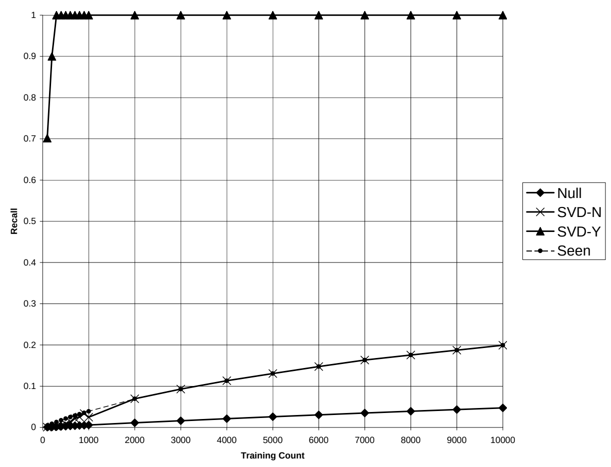

Fig. 4. Recall for simple locations

The low precision without seed data may be explained by an effect discussed in the description of the algorithm. A location with multiple shop types may cause a low but positive score at any location with just one of its constituent shop types. In our training data, sampled from a realistic distribution, some observed capabilities have extremely low frequencies, leading to an overlap in the scores for capabilities that do not exist and for capabilities that do exist, but observed at very low frequencies. The seed data cures this by raising the scores for existing but rarely observed capabilities.

The current T-SVD algorithm is very effective if either the locations are simple or there is seed data to merge with the local data. As hypothesized, seed data is needed in some cases.

5 Conclusions and Future Work

A location-based reminder system could be much more useful if the user only needed to provide the content of the reminder, without having to provide all the locations at which the reminder should be delivered. One way to achieve this is for the reminder system to learn where similar previous reminders have resulted in success in the past. Unfortunately, impractically large numbers of observations would be needed to learn anything about infrequently used capabilities from observations alone. Truncated singular value decomposition can generalize from a small number of direct observations to build high-fidelity models of locations. T-SVD's structure supports the addition of summary capability-clustering data, enabling rapid generalization in real-world situations such as the existence of hybrid store types. In our experiments based on real shopping basket data, 1000 samples randomly distributed across 135 locations gave 100% precision and 90% recall.

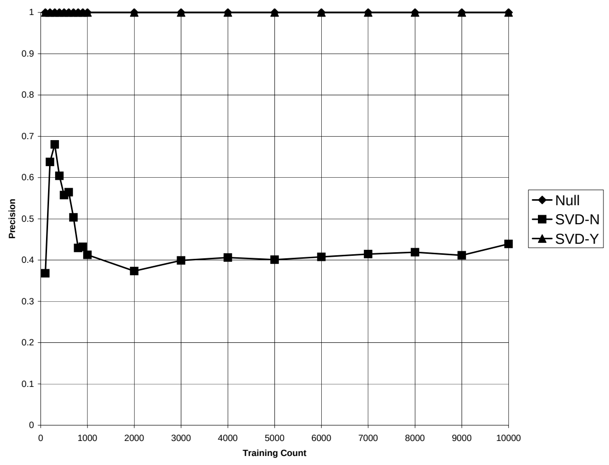

Fig. 5. Precision for compound locations

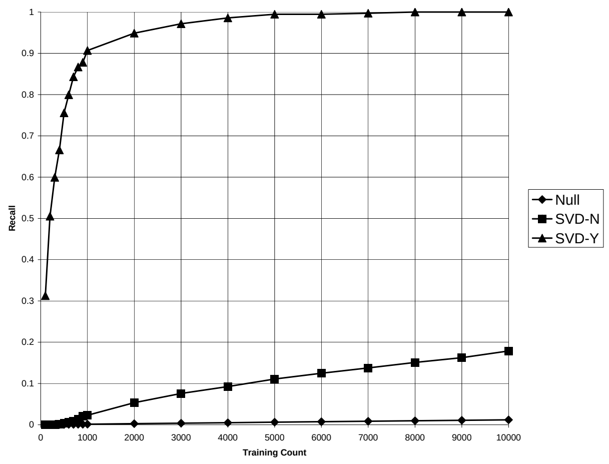

Fig. 6. Recall for compound locations

Future work will explore enhancements to the algorithm, such as automatic selection of the truncation rank, normalization of the cutoff score, and improved cluster detection for compound locations without seed data. Likewise, the workload will be made more realistic, for example by increasing the number of product groups. Finally, we plan to field test a small-scale reminder system based on this approach.

Acknowledgments

This research was supported in part by the UCSD FWGrid Project, NSF Research Infrastructure Grant Number EIA-0303622. It was also supported in part by a gift from Microsoft Research. We are grateful to Sanjoy Dasgupta for his consultations on machine learning. We thank the anonymous US national supermarket chain for sharing their shopping basket data, and Kevin Patrick for helping with the arrangements.

References

1. Farrimond, S., Knight, R.G., Titov, N.: The effects of aging on remembering intentions: Performance on a simulated shopping task. In: Applied Cognitive Psychology. Volume 20. (April 2006) 533–555
2. Rendell, P.G., Craik, F.I.M.: Virtual week and actual week: Age-related differences in prospective memory. In: Applied Cognitive Psychology. Volume 14. (2000) S43–S62
3. Sohn, T., Li, K.A., Lee, G., Smith, I.E., Scott, J., Griswold, W.G.: Place-its: A study of location-based reminders on mobile phones. In: Ubicomp. (2005) 232–250
4. Dey, A.K., Abowd, G.D.: Cybreminder: A context-aware system for supporting reminders. In: HUC. (2000) 172–186
5. Mankoff, J., Hsieh, G., Hung, H.C., Lee, S., Nitao, E.: Using low-cost sensing to support nutritional awareness. In: Proceedings of the 4th international conference on Ubiquitous Computing, Springer-Verlag (2002) 371–376
6. Youll, J., Morris, J., Krikorian, R., Maes, P.: Impulse: Location-based agent assistance (2000)
7. Youll, J.: Wherehoo and periscope: a time & place server and tangible browser for the real world. In: CHI '01 extended abstracts on Human factors in computing systems, ACM Press (2001) 109–110
8. Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by latent semantic analysis. Journal of the American Society of Information Science **41**(6) (1990) 391–407
9. Cernekova, Z., Kotropoulos, C., Pitas, I.: Video shot segmentation using singular value decomposition. In: ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo, Washington, DC, USA, IEEE Computer Society (2003) 301–304
10. Chu, M.T.: On the statistical meaning of truncated singular value decomposition
11. Nicholas, C., Dahlberg, R.: Spotting topics with the singular value decomposition. Lecture Notes in Computer Science **1481** (1998) 82–??
12. Lin, M.H.: Out-of-core singular value decomposition. Technical report