UNIVERSITY OF CALIFORNIA, SAN DIEGO

Machine Learning for Context-Aware Reminders and Suggestions

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Patricia Shanahan

Committee in charge:

Professor William Griswold, Chair
Professor Sanjoy Dasgupta
Professor Adriene Jenik
Professor Keith Marzullo
Professor Kevin Patrick

2009

The dissertation of Patricia Shanahan is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2009

DEDICATION

To my parents, Richard and Mary Shanahan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to first thank Bill Griswold, for his general support as my adviser, and especially for asking me how we could automatically issue reminders for tasks such as buying stamps. This dissertation is my attempt at an answer.

Similarly, I wish to thank Sanjoy Dasgupta, for several informative conversions about machine learning, and specifically for suggesting testing weighted SVD.

More generally, I want to thank all the wonderful people at UCSD for an intellectually stimulating and challenging experience, and all my friends for understanding when I was busy doing homework, or writing a paper, or writing this dissertation.

I wish to thank NCR, Celerity Computing, Floating Point Systems, Cray Research, and Sun Microsystems, both for all that I learned working for them, and for leaving me in a position to take a few years for study and research.

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.

An an anonymous US national supermarket chain supplied data. Thanks are also due to Kevin Patrick for suggesting contacting them, and for help with arrangements to get the data.

Portions of Chapters 1, 2, and 9 are based on material that appeared in "Inferring the Everyday Task Capabilities of Locations", Patricia Shanahan and William G. Griswold, LoCA 2007: 157-174, 2007. The dissertation author was the primary investigator and author of this material.

VITA

1970	Bachelor of Science in Mathematics, Imperial College London.
1975	Master of Science in Computer Science, Birkbeck College, University of London
1970 - 1975	NCR Ltd., London, England.
1975 - 1983	NCR Inc., San Diego, CA
1983 - 1988	Celerity Computing, San Diego, CA
1988 - 1991	FPS Computing, San Diego, CA
1991 - 1996	Cray Research Inc., San Diego, CA
1996 - 2002	Sun Microsystems, San Diego, CA
2002 - 2009	Graduate Student, University of California, San Diego.
2009	Doctor of Philosophy in Computer Science, University of California, San Diego.

PUBLICATIONS

William G. Griswold, Patricia Shanahan, Steven W. Brown, Robert T. Boyer, Matt Ratto, R. Benjamin Shapiro, Tan Minh Truong, "ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing", *IEEE Computer*, 37(10): 73-81, 2004.

Patricia Shanahan, William G. Griswold:, "Inferring the Everyday Task Capabilities of Locations" LoCA 2007: 157-174, 2007.

ABSTRACT OF THE DISSERTATION

Machine Learning for Context-Aware Reminders and Suggestions

by

Patricia Shanahan

Doctor of Philosophy in Computer Science

University of California, San Diego, 2009

Professor William Griswold, Chair

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, and assume the availability of other capabilities in the cluster. This dissertation proposes a similar approach to computer-based discovery of routine location capabilities, applying singular value decomposition 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. I propose using the time and place of deleting items from a to-do list application to provide the local data. I also examined the effect of feedback on false positive errors, combined with a weighted singular value decomposition.

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. A simple clustering algorithm would treat each combination as an independent cluster. 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.

The robustness of the technique was further tested by adding two difficulties, convenience stores and false training data. The convenience-store workload included some locations that provided only the thousand most frequently used capabilities, regardless of other cluster data. False positive feedback and feature weighting both allowed use of a larger truncation rank, improving convenience store results, and reduced errors due to false training data.

The technique extends to estimate whether a capability is available at a given time. Data for short time intervals was "folded-in" to the singular value decomposition to obtain projections for those time intervals. The projections, interpreted as Poisson distribution arrival rates, were used to compare posterior probabilities for various time intervals given the observed data. The time extension was tested with workloads that included 24 hour supermarkets and early opening for a subset of capabilities at one location.

1 Introduction

1.1 Why Location Capabilities?

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." [FKT06].

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 [FKT06]. However, comparison of similar tasks in a laboratory model and in the real-world can get different results [RC00].

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 1.1. 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 requires several phone features:

- 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. This decision may involve a combination of information about the task, information about Joe's activities, and explicit inputs.

- Joe's phone can interpret bar codes with high accuracy.

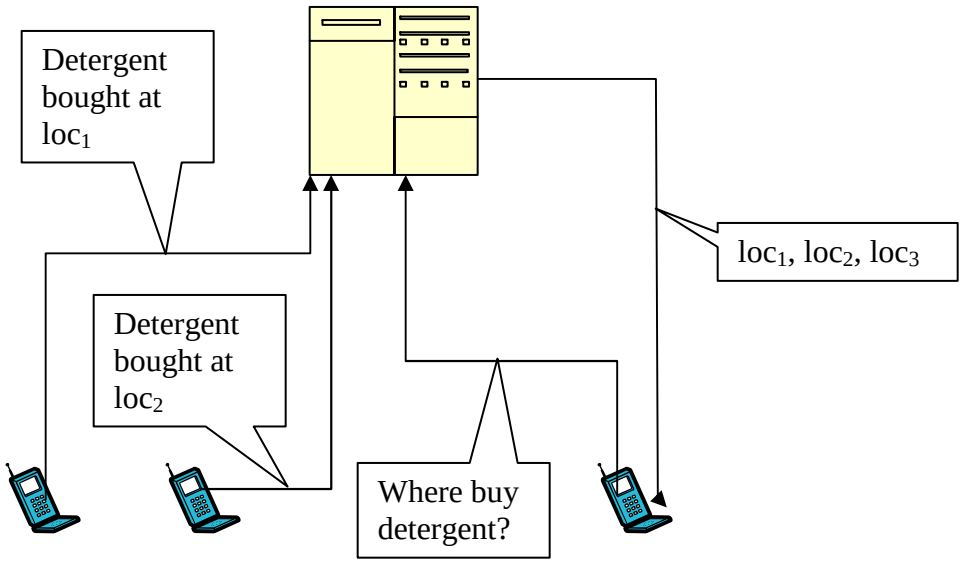

Figure 1.1: A server receives reports of detergent having been bought at locations loc_1 and loc_2. loc_3 is a third location that has been found to be similar to the other two. When Joe's phone asks where detergent can be bought the server responds with all three locations.

- 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 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.

A capability-based reminder system would need many components to work together. This dissertation discusses just one of them, the characterization of locations in terms of the capabilities they offer. As discussed in Chapter 2, none of the direct methods covers all capabilities. In particular, detailed but basic capabilities, such as availability of postage stamps or instant coffee, are not often reflected in accessible web data. The next section discusses the specific problem, and its possible solutions, in more detail.

1.2 The Everyday Shopping Problem

Everyday shopping presents a location characterization problem whose routine nature makes it both important and difficult. Many locations offer some, but not all, of the capabilities needed for everyday shopping. Because of their routine nature, everyday shopping tasks are eminently forgettable, increasing the value of reminders. They can often be deferred for days, until the user happens to be at location that offers the appropriate capabilities. On the other hand, the set of distinct capabilities involved in everyday shopping is immense. A shopping center may stock thousands of items, any one of which may be on a given user's shopping list. It would take a long time to accumulate enough user-supplied data to full characterize it, and by the time each capability had been reported at least once some of the older data would be out of date. This dissertation uses everyday shopping as the example problem both because of its usefulness and because any solution that works for it is likely to also handler cases in which locations have fewer relevant capabilities.

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 dissertation 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.

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.

Moreover, the capabilities of a location usually change with time, both cyclically

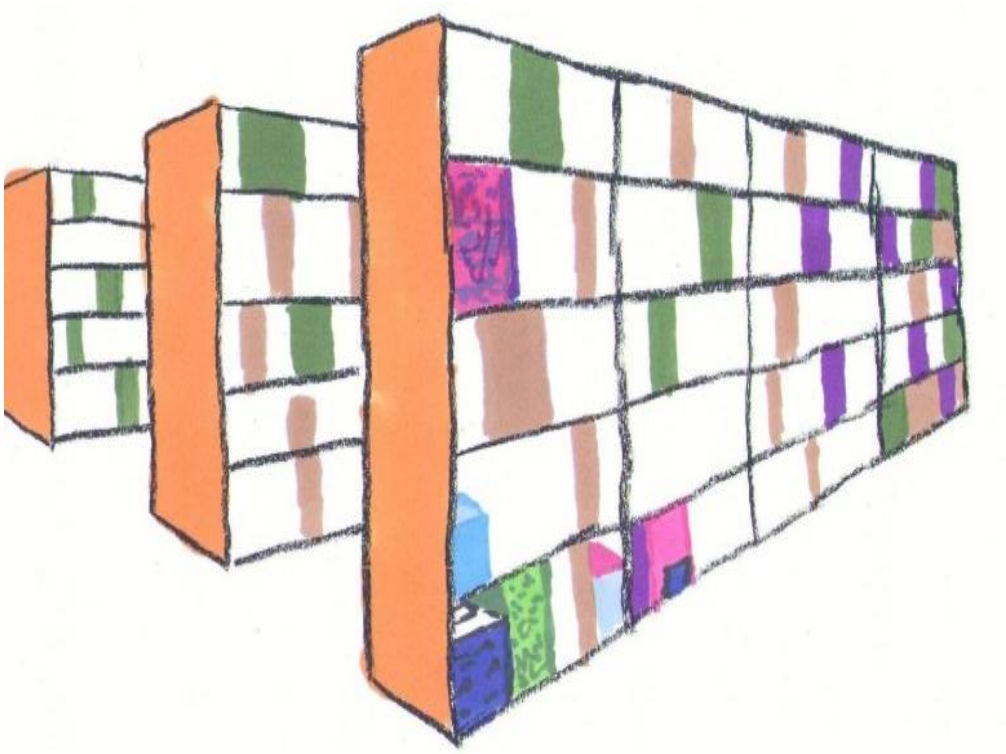

Figure 1.2: Supermarket Shelves

A normal shop may have many rows of shelves, each containing dozens of different items.

with various periods and permanently. The most important periods are one day, seven days, and one year. Most locations have some daily variation in their behavior. Even 24 hour stores often offer some capabilities only during limited hours. For example, a 24 hour supermarket may have a pharmacist on duty, and be able to fill prescriptions, during daytime and evening shifts but not at night. Different behavior at weekends leads to a seven day period. Seasonal changes, including special merchandise for holidays, operate with a one year period. Old stores go out of business, or move to a different location. New stores replace them. Managers make permanent changes in what they stock.

A location-aware reminder system for everyday tasks would not be very effective unless backed by a system for identifying suitable locations for the tasks. The research described in this dissertation used simulation studies to select, develop, and evaluate a system for inferring the capabilities of locations for everyday tasks.

There are numerous combinations of capabilities that can be combined in fairly arbitrary ways. A simple clustering algorithm would treat a strip mall with shoe shop as a different cluster from a strip mall that is similar except for lack of a shoe shop, and need to learn about them separately.

The problem calls for an algorithm that can learn about sums of clusters of capabilities. After experimenting with a number of approaches, I selected truncated singular value decomposition (T-SVD). Some experiments also used a weighted variation on truncated singular value decomposition. T-SVD also enables bootstrapping the training by conveniently supporting the infusion of summary capability clustering data.

Different capabilities have different usage frequencies, and their distribution may affect the accuracy of the algorithm. The experiments use data derived from actual shopping basket data obtained from a major supermarket chain. The algorithm was effective if either the locations had simple behavior or I supplied summary capability clustering data to supplement data about the locations. In particular, for the basic workload, with seed 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 almost 90% recall.

1.3 Scope

This dissertation proposes an algorithm, based on singular value decomposition, for predicting the capabilities of locations from seed data and a random sampled observations.

In order to develop and test algorithms, I wrote a simulation environment, described in Chapter 4. All tests use artificial data based on real world distributions. These tests, by their nature, cannot measure how well the simulated world matches the real world, only how well the inference algorithm operates relative to the simulated.

Many issues add to the difficulty of generalization. The test cases include several issues:

- Locations with cross-cutting behavior. The inference system is based on an assumption that capabilities appear in clusters. Cross-cutting locations exhibit capabilities from several clusters without exhibiting the whole of each cluster.
- Incorrect training data. In the real world, some inputs will be erroneous. For example, a user might record completion of a to-do list item late, at a different location, giving a false impression that the new location has the associated capability.
- Variations in capabilities based on time of day. Shops and departments open and close. The available capabilities at a location can change depending on the time of day.

Simulation testing motivated several enhancements to the algorithm.

1.4 Outline

The next chapter discusses some related work on other approaches to location-aware reminders. Chapter 3 gives background on the singular value decomposition algorithm. Chapter 4 describes the simulation environment I built to test the algorithm. Chapter 5 describes a test workload and discusses results of applying the basic algorithm to it. Following chapters describing some enhancements to the algorithm in the

context of more difficult workloads. Chapter 9 discusses how the algorithm would be used in a reminder system. The final chapter presents some conclusions and ideas for future work. The very large number of simulation experiment results would interfere with the flow of the text if placed in-line, but are useful background for understanding the discussion. An appendix contains the full results, with only the most relevant data presented in-line.

1.5 Acknowledgment

Portions of this chapter are based on material that appeared in “Inferring the Everyday Task Capabilities of Locations”, Patricia Shanahan and William G. Griswold, LoCA 2007: 157-174, 2007. The dissertation author was the primary investigator and author of this material.

2 Related Work

This chapter discusses related work on location based reminders. Although a full survey of singular value decomposition is beyond the scope of this dissertation, the next chapter contains some background and motivation for using it.

Several researchers have proposed location based reminders. There are two aspects to research in this area, only one of which is relevant to this dissertation, the problem of deciding whether a known location is an appropriate reminder location. The other aspect is determining the device's actual location.

There are several data sources that can be used for selecting locations for 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 location 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 [SLL+05]. 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 [DA00]. 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 dissertation, 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 [MHH+02]. 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 is a practical solution for a person needing an unusual capability. The lack of automation makes it impractical for identifying large numbers of common capabilities for many locations.

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" [YMKM00, You01]. 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.

The most general step in this direction is the Semantic Web project [W3C]. If successful, it will make the deep web, the parts of the web that are currently only accessible to humans through reading and understanding web pages, accessible to computer programs. The ultimate solution would be a web query asking e.g. for addresses of shops in a neighborhood that stock anything on the user's shopping list.

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

2.5 Acknowledgment

Portions of this chapter are based on material that appeared in "Inferring the Everyday Task Capabilities of Locations", Patricia Shanahan and William G. Griswold,

LoCA 2007: 157-174, 2007. The dissertation author was the primary investigator and author of this material.

3 Location Capability Inference Algorithm

This chapter gives some background on the most promising approach this research project identified, truncated singular value decomposition.

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:

Here, each value represents a number 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

Table 3.1: 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

Table 3.2: 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

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, that data source carried a weight of 3, given its general reliability. An added row containing 3 for each of those locations, and 0 in all other entries, represents the data. Similarly, an externally supplied list of hardware items with weight 5 would be represented by an added column with 5 for those items, and 0 for the remaining rows:

3.2 Algorithm Concept

The obvious and simplest algorithm, the "Null" algorithm, is to project availability of a capability at a location if, and only if, the data includes an observation of exactly that capability at that location.

The Null algorithm is not very effective. The 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.

To outperform the Null algorithm, an algorithm must effectively change some of the incorrect entries without changing too many of the correct ones. The assumption underlying this work is that the real capability-location matrix is, in some sense, simpler than the available data. It is based on a few big decisions such as "I'm going to start a bakery in this mall.", that drive clusters of product assortment decisions, such as "I'm going to sell bread.".

We need to construct a table that is simpler than the training data table, but also a close approximation to it. 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:

$$\mathbf{X} = \begin{pmatrix} 40 & 153 & 0 & 0 & 4 \\ 26 & 95 & 0 & 0 & 0 \\ 0 & 0 & 203 & 3 & 0 \\ 0 & 0 & 100 & 4 & 0 \\ 0 & 0 & 23 & 2 & 0 \\ 0 & 0 & 45 & 0 & 0 \end{pmatrix}. \tag{3.1}$$

Now our problem is to find a matrix that is as close as possible to $\mathbf{X}$ but simpler than it. We need to replace the intuitive ideas "simpler" and "close as possible" with mathematical concepts.

Rank is the most relevant measure of matrix simplicity. The rank of a matrix is the size of the largest set of rows such that no row is linearly dependent on the others, no row can be constructed from a weighted sum of the other rows. The rank of a matrix cannot be greater than the smaller of the number of rows and the number of columns, but can be less. Despite having six rows and five columns, **X** only has rank four. Each of the last four rows can be generated as a linear sum of any two of them. A "simpler" matrix than **X** would have even lower rank.

If the basic hypothesis of product clustering is correct, **X** is a noisy, sampled approximation to a rank two 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 most obvious measure of distance between two matrices, **A** and **B**, is their Euclidean distance $\sqrt{\sum_{i,j}(\mathbf{A}_{i,j}-\mathbf{B}_{i,j})^2}$, the Frobenius norm of their difference.

Thus the intuitive idea of finding a simpler capability-location table, similar to the one based on the observations, reduces to finding, given a matrix **X** and a positive integer k, k less than the rank of **X**, a rank k matrix $\widetilde{\mathbf{X}}$, that minimizes the Frobenius norm of the difference $\widetilde{\mathbf{X}}-\mathbf{X}$. This is a job for truncated singular value decomposition (T-SVD).

3.3 Singular Value Decomposition

Singular value decomposition is a matrix factorization, with a wide range of uses. David Austin's "We Recommend a Singular Value Decomposition" is a short introduction [Aus09]. It is also described in section 6.3 of Gilbert Strang's "Linear Algebra and its Applications" [Str05].

Given a matrix **X**, it computes matrices **U**, **S**, and **V** such that $\mathbf{X}=\mathbf{USV}^T$, where **X** is diagonal, and **U** and **V** are unitary matrices. For real matrices, the relevant case, a unitary matrix is orthogonal. That is, the transpose of the matrix is its inverse.

The singular values, the non-zero elements of the singular value matrix, **S**, appear on the main diagonal in descending order. Intuitively, each is the weight of some abstract feature as an explanation of the values in the original matrix.

The singular value matrix from the decomposition of the training data matrix in Equation 3.1 is

$$\mathbf{S} = \begin{pmatrix} 231.9131 & 0 & 0 & 0 & 0 \\ 0 & 186.3347 & 0 & 0 & 0 \\ 0 & 0 & 2.8800 & 0 & 0 \\ 0 & 0 & 0 & 2.3201 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.2}$$

The number of non-zero values in $\mathbf{X}$ equals the rank of the input matrix, four. The singular values are in descending magnitude order. Intuitively, each represents the importance of an abstract feature to explaining the values in the input matrix. Note that there are two relatively large singular values and two smaller ones.

Truncated singular value decomposition (T-SVD) drops all except the largest k singular values, for some positive integer k, treating the remaining ones as though they were zero. All except the first k columns of each of $\mathbf{U}$ and $\mathbf{V}$ can be dropped, because they would be multiplied by zeros in $\mathbf{S}$. Similarly, $\mathbf{S}$ is only k by k.

In our current algorithm, k is supplied externally. This example uses $k = 2$. 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.

The truncation to k singular values results in a rank k approximation. The truncation could be done by running the full decomposition 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:

$$\mathbf{U} = \begin{pmatrix} 0 & -0.84895 \\ 0 & -0.52848 \\ 0.87541 & 0 \\ 0.43145 & 0 \\ 0.099325 & 0 \\ 0.194 & 0 \end{pmatrix} \tag{3.3}$$

$$\mathbf{S} = \begin{pmatrix} 231.91 & 0 \\ 0 & 186.33 \end{pmatrix} \tag{3.4}$$

$$\mathbf{V} = \begin{pmatrix} 0 & -0.25598 \\ 0 & -0.96651 \\ 0.99981 & 0 \\ 0.019622 & 0 \\ 0 & -0.018224 \end{pmatrix} \tag{3.5}$$

Next calculate **Y**, the rank 2 matrix approximation to **X**, by multiplying the factors:

$$\mathbf{Y} = \mathbf{USV}^T = \begin{pmatrix} 40.493 & 152.89 & 0 & 0 & 2.8828 \\ 25.208 & 95.176 & 0 & 0 & 1.7946 \\ 0 & 0 & 202.98 & 3.9837 & 0 \\ 0 & 0 & 100.04 & 1.9634 & 0 \\ 0 & 0 & 23.03 & 0.452 & 0 \\ 0 & 0 & 44.983 & 0.88284 & 0 \end{pmatrix}. \tag{3.6}$$

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." [Chu]. **Y** is the required simpler approximation to **X**.

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

All elements that were non-zero in Table 3.1 are non-zero in Table 3.3. 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 3.3: 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

Table 3.3 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.

3.4 T-SVD Applications

The full range of uses of truncated singular value decomposition is beyond the scope of this dissertation. This section contains a few examples.

T-SVD, as well as providing a low-rank approximation to its input matrix as described above, also provides a very compact representation. Most uses of T-SVD derive some benefit from compact representation. For image compression, compactness is the primary motivation for its use. Singular value decomposition for image compression was proposed in 1976 [AP76].

Deerwester et. al. applied T-SVD to latent semantic indexing [DDL$^+$90]. It is related to the location capability problem, with documents replacing locations and terms replacing capabilities. It differs in starting from complete documents, not just samples. Latent semantic indexing deals with very large problems. One of the test data sets used in a study of rank selection contains 5 million documents and 1,809,597 unique terms [Bra08].

The recommender system problem is even closer to the location capability problem. The location capability problem could be rephrased as "Locations that offer capabilities similar to ones this location offers also offer . . . ". Sawar et. al. proposed singular value decomposition for recommender systems [SKKR00]. The main difference is in

the nature of the values in the matrix. In a recommender system, the matrix value for user i and item j measures how strongly user i likes or approves of item j. The value in the location capability matrix is the number of reports of the location exhibiting the capability.

3.5 Acknowledgment

Portions of this chapter are based on material that appeared in “Inferring the Everyday Task Capabilities of Locations”, Patricia Shanahan and William G. Griswold, LoCA 2007: 157-174, 2007. The dissertation author was the primary investigator and author of this material.

4 Evaluation Environment

This dissertation examines inference algorithms for a particularly demanding sub-case of the location capability problem, shopping. It is an important sub-case, because simple shopping tasks are both easily forgotten, and can be flexible as to both time and place, provided the location has the associated capability. A small shopping center may stock thousands of products, any one of which may be required by a reminder user, requiring fine granularity of capability inference. The key hypothesis is that machine inference can leverage limited local information, aided by seed data, to project capabilities that have not been reported for the location.

A simulation environment for testing this hypothesis presents both challenges and simplifications compared to an actual reminder system. Its main function is to compare results for different approaches, so it contains multiple implementations of the inference engine, and accepts many parameters for controlling the workload, the inference engine, and report generation. It runs many times, to compare, for example, effects of different amounts of training data. On the other hand, it is a simple batch program, reading one input file, and generally producing one output file.

Most of the program is in Java, with Matlab code for the singular value decomposition. Although the Java Colt linear algebra library implements singular value decomposition, it only does a full decomposition. The simulator needs truncated singular value decomposition. For all except the smallest problems, using Matlab's svds was faster than using Colt's cern.colt.matrix.linalg.SingularValueDecomposition and subsequently truncating.

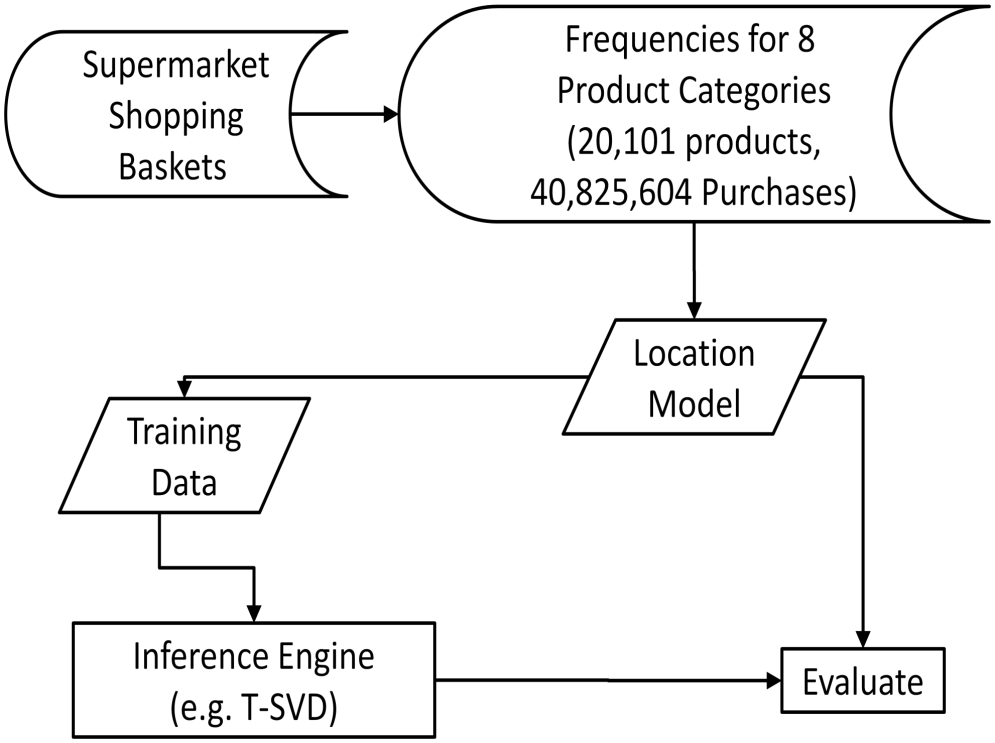

Figure 4.1: Simulation Process Outline Diagram

4.1 Process Outline

Figure 4.1 shows an outline of the test process. An anonymous US supermarket chain supplied the shopping basket data. Each test uses a location model of the world derived from the frequencies for eight product categories. The training data is a set of random samples from the training data. The final results are produced by comparing the inference engine's predictions, based on the training data, to the location model.

4.2 Modules

This section describes the highest level division of the simulator into modules. The main modules are control, generator, and inference, and control. They are supplemented by utilities, freestanding, and database output modules. Figure 4.2 shows a high

level view of the most important packages and classes.

4.2.1 Generator

The generator module represents ground truth for purposes of the simulation. It builds a model of simulated locations and their capabilities. It supplies randomly sampled training data. During report generation, it supplies the "actual" results for calculation of precision and recall.

4.2.2 Inference

The inference module contains multiple inference engine implementations. Each inference implementation provides a class implementing the InferenceEngine interface. An InferenceEngine can learn from a set of training data, and return an InferenceResultSet representing its inferences.

There are four basic InferenceEngine implementations, Relational, Cluster, Eigen, and SVD. Relational implements a simple probabilistic relational model [GFKT01]. Cluster implements an equally simple clustering algorithm. Neither Relational nor Cluster was a good fit for the problem. Both require randomly generated starting state: a prior distribution for Relational and an initial cluster set-up for Cluster. In each case, that lead to a need for multiple runs, and some of which were long due to slow convergence. Eigen calculates the eigenvectors of the correlation matrix. The size of the matrix was such that it used too much memory for all except the smallest problems.

SVD implements a truncated singular value decomposition. It produced good results on the preliminary tests, and is the basis for the rest of the investigation. The implementation has three variations. The first, a pure Java solution using Colt, is practical only for small problems because it calculates the full singular value decomposition and then truncates it. The more practical solution passes the data to Matlab and runs a script using the svds function, which directly computes the truncated SVD of a sparse matrix, without first computing the full SVD.

Some experiments also used a weighted SVD, implemented by an iterative algorithm in a Matlab script. Because of the number of SVD iterations, it uses a faster imple-

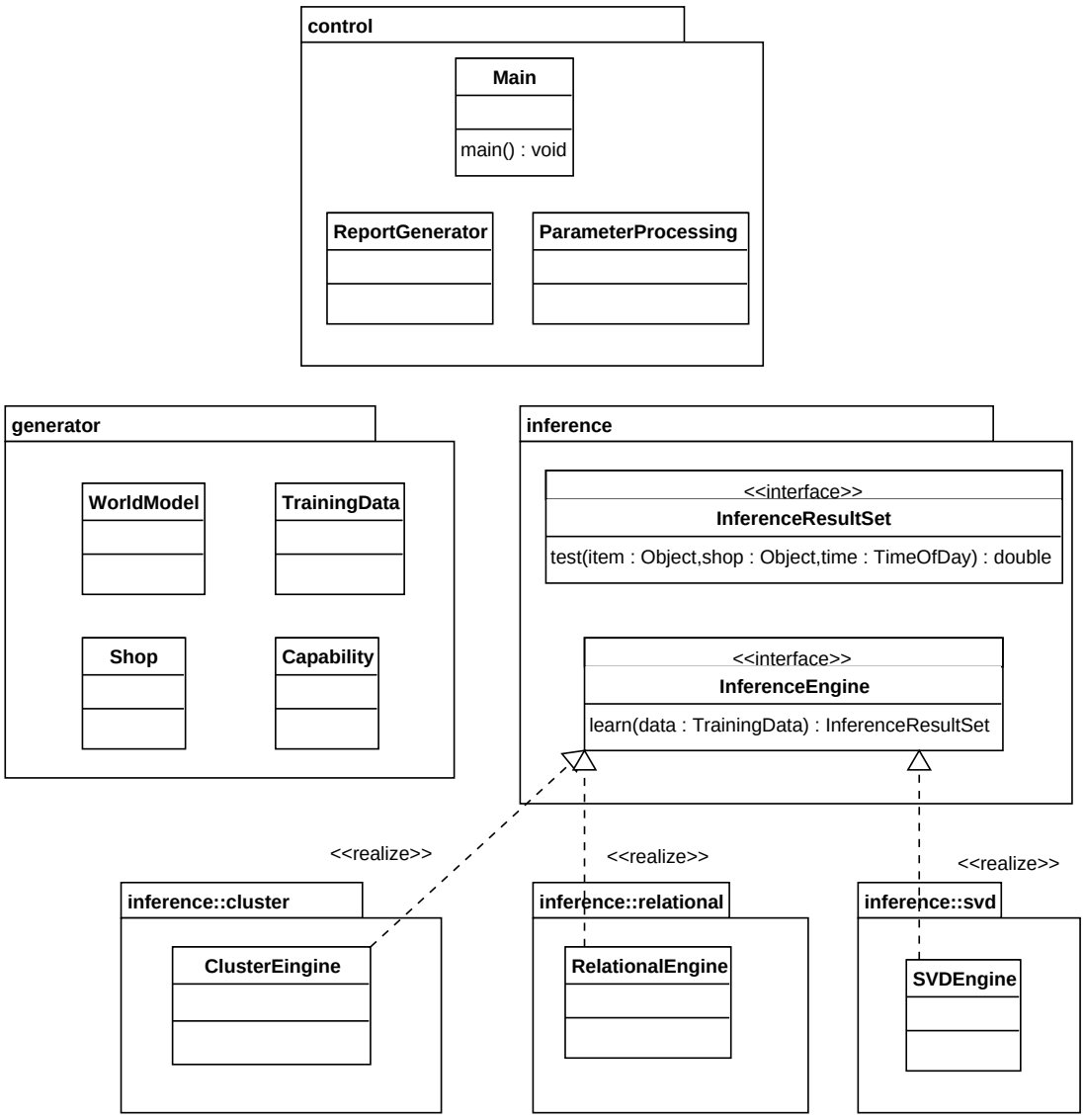

Figure 4.2: UML Class Diagram of Main Simulator Packages and Classes

mentation, based on the PROPACK library, http://soi.stanford.edu/~rmunk/PROPACK/.

4.2.3 Control

The control module contains the main program, parameter processing, and report generation. It sequences and coordinates each run.

The control module parses and validates a parameter file. It uses the generator to build a world model based on the parameters. It then obtains an InferenceEngine, and passes it training data supplied by the world model. Finally, it prepares reports combining estimated data from the InferenceResultSet with actual data from the world model.

4.2.4 Freestanding

The freestanding package contains a set of applications designed to run outside the simulation environment. They perform tasks such as post-processing of multiple result files.

4.2.5 Utilities

The utilities are generally useful classes, such controlled substitutes for random number sequences to aid in unit testing random number dependent functions such as training data generation.

4.2.6 Database Output

Almost always, the reports give sufficient information for evaluating a test, but there have been a few cases in which much more information was needed to understand what happened. Within the simulator, the generator and inference modules are isolated from each other to ensure the inference module uses only the training data. Coordinated assertion checking across the modules inside the program would break that isolation. When database output is enabled, the generator, inference, and control modules all supply information to the database output module, which places it in a relational database.

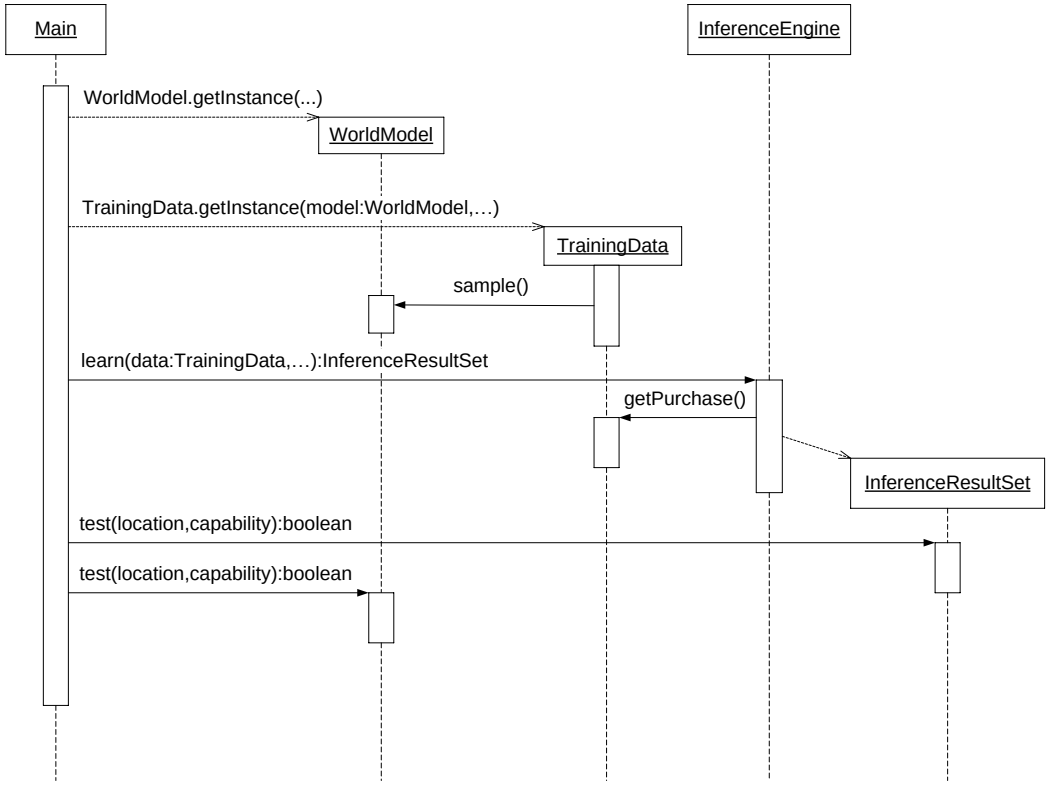

Figure 4.3: UML Sequence Diagram of Main Operations

Each relational table captures only local information from one part of the simulator. Subsequent database selects and joins bring out details about the behavior of the whole system, and verify system-wide assertions.

4.3 Inference Sequence

The sequence of operations is illustrated in Figure 4.3.

The control module reads three groups of parameters:

- World model parameters define a set of locations and their capabilities.
- Inference parameters select the algorithm to use and sets its controls.

- Report parameters select the reports. The results can be reported with different levels of detail.

The control module obtains a WorldModel from the generator, based on the world model parameters. It creates a TrainingData object, containing random samples of locations and capabilities from the WorldModel.

The inference parameters control the creation of an InferenceEngine. It is passed the TrainingData, and uses it to calculate an InferenceResultSet.

The most important form of report is a Precision Recall Report. It contains precision, recall, true positive, false positive, true negative, and false negative numbers. For each combination of location and capability, the results reflect the comparison of the result of asking the WorldModel whether the location has the capability to the result of asking the InferenceResultSet the same question.

5 Basic Inference Algorithm Test Workload

This chapter presents the basic algorithm, with two very simple examples, and introduces the result presentation. Subsequent chapters present more difficult cases, with related extensions and modifications to the algorithm.

5.1 Proposed Inference System

I 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, I built a model of location capabilities and used it to test a number of inference algorithms, eventually settling on truncated singular value decomposition.

5.2 Evaluation

I 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.

I 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. I 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, I 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.

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, I selected the eight groups with the highest total purchase counts. I 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 I 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. I 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.

Table 5.1: 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

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 5.1. 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.

I 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, I 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. Runs with seed data treated

each combination of artificial location and associated capability as though it had 1000 purchases. These parameters changed later, in the light of more difficult workloads.

I 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, 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. If TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives then

$$Precision = \frac{TP}{TP+FN}. \tag{5.1}$$

$$Recall = \frac{TP}{TP+FP}. \tag{5.2}$$

The reports show precision and recall separately, rather than combining them in a single figure of merit, because their values have different consequences for a reminder system. High precision indicates that all, or almost all, alerts are for locations that really have the required capability. High recall indicates that an alert is issued at all, or almost all, locations that do have the capability.

A useful learner achieves good precision and recall. A learner that claims every capability is available at every location has perfect recall but zero precision. A learner that never asserts availability of a capability has perfect precision but zero recall.

Ideally, precision and recall should be measured relative to the rate at which users will want to be reminded of each capability. That data is not available. We can measure them two ways, by count and by probability. The "by count" measures count all capabilities equally, regardless of their probabilities. An inference error for a very rarely used capability has the same effect on these measures as an error in the most frequently used capabilities. In practice, users can use other means to find locations for really infrequent capabilities. The "by probability" measures weight precision and

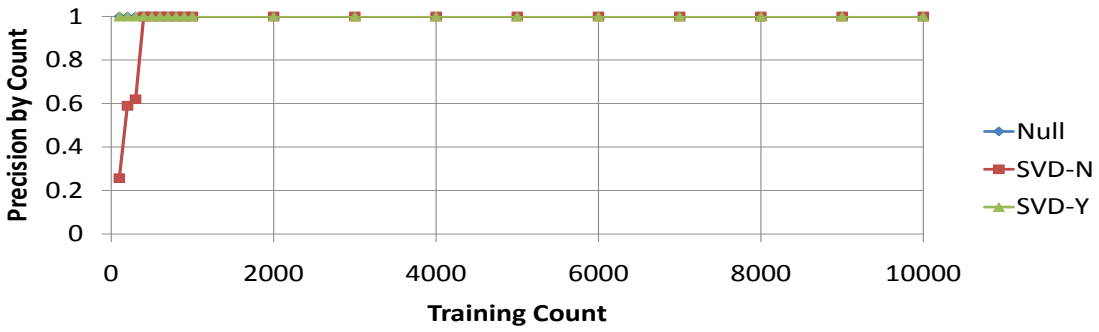

Figure 5.1: Precision by count for Simple Locations

recall according to the probability of each capability, reducing the weight given to low frequency capabilities.

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 5.1 and 5.1 show precision and recall by count for the "Simple" data, measured as a function of the number of training data samples. 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. Figures 5.3 and 5.3 show precision and recall by probability. Both Null and SVD-N have significantly worse recall for low probability capabilities because most of them do not appear in the training data.

Although the four charts together give a complete picture of the results for a series of tests, they are not all needed in-line, and can be difficult to compare. A summary, such as Figure 5.2 gives an outline of the results in a more compact form. The full charts are all in Appendix A. Individual charts are interleaved with the text as needed to illustrate specific points.

The full discussion of the summary format is at Appendix A.1. "Max" is the maximum and "Mean" the mean of the function formed by linear interpolation between the results. "N(0.9)" is the number of training samples needed to achieve 90%.

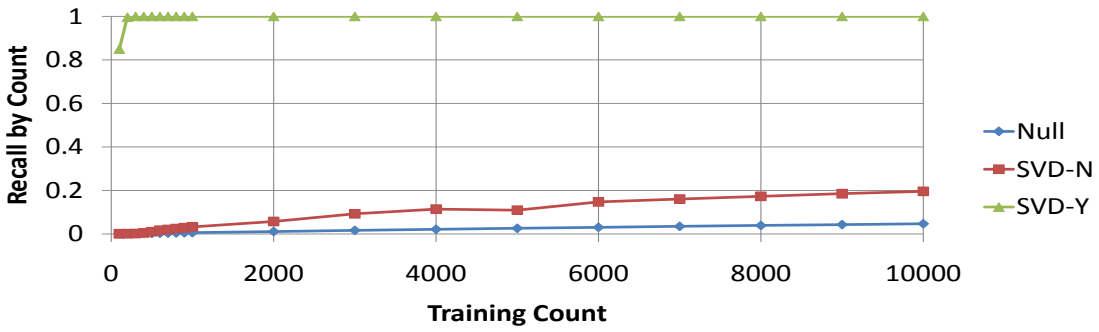

Figure 5.2: Recall by count for Simple Locations

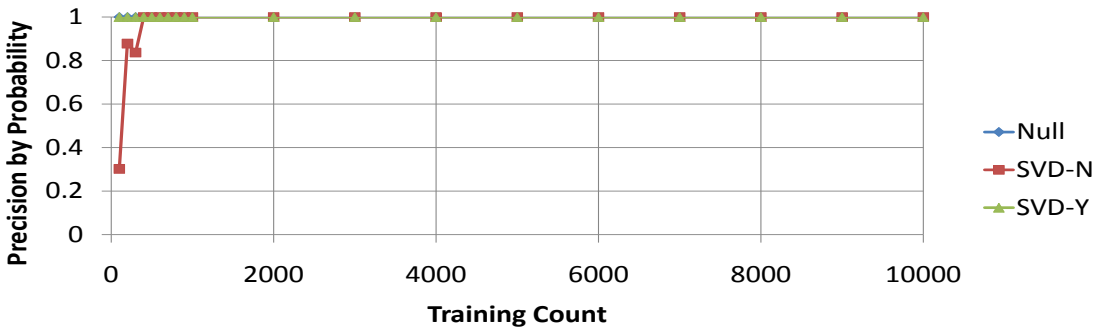

Figure 5.3: Precision by probability for Simple Locations

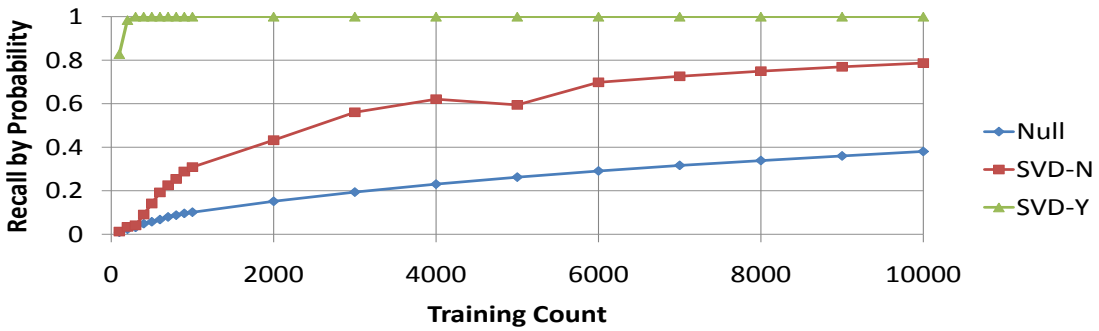

Figure 5.4: Recall by probability for Simple Locations

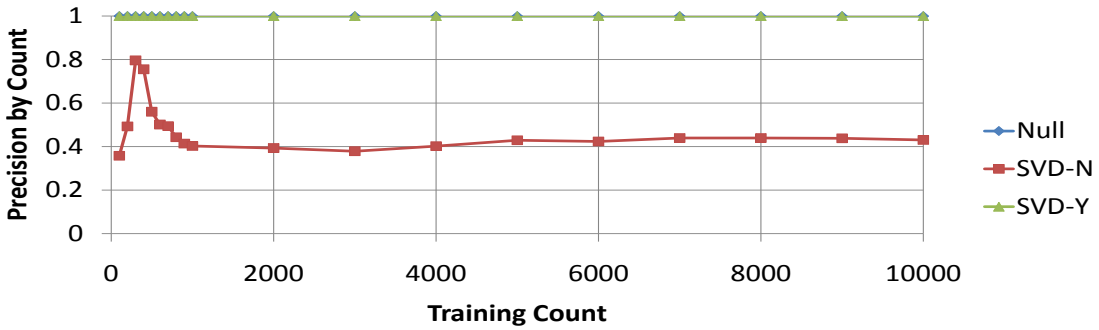

Figure 5.5: Precision by count for Compound Locations

"N(0.9M)" is the number of training samples needed to achieve 90% of "Max", a measure of consistency. In each case the results reflect the worse of the "by count" and "by precision" results.

Table 5.2: Summary of Basic Tests

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Simple (Figure A.3)								
Null	1.00	1.00	100	100	0.03	0.05		9,000
SVD-N	0.99	1.00	400	400	0.12	0.20		9,000
SVD-Y	1.00	1.00	100	100	1.00	1.00	200	200
Compound (Figure A.4)								
Null	1.00	1.00	100	100	0.01	0.01		9,000
SVD-N	0.43	0.80			0.10	0.17		9,000
SVD-Y	1.00	1.00	100	100	0.96	1.00	2,000	2,000

"Null" and "SVD-N" both perform significantly worse for the compound location case than the simple location case, as shown in Figures A.3 and A.4. In particular, "SVD-N" has significantly worse precision. The relatively good precision for low training counts for "SVD-N" is explained by the low number of capabilities it predicts. The low precision for higher training counts, 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. The training data, sampled from a realistic distribution, contains 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.

With seed data, "SVD-Y" achieves 100% precision throughout, and also at least 90% recall given at least 2,000 samples.

5.3 Implementation Issues

This section discusses some system issues that affect the algorithms. The use of these algorithms in a reminder system is discussed in more detail in Chapter 9.

The configuration in Figure 1.1, 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 should be centralized. Figure 5.6 illustrates this architecture.

Suppose there are c capabilities and l locations. There may be thousands of each. The input matrix $\mathbf{X}$ is has cl elements, but is sparse, having non-zero entries only for observed combinations of location and capability and for seed data. The output matrix is the same size, and may be less sparse, but does not need to be stored.

If observations were not changing, elements could be calculated as needed from the results of the decomposition:

$$\mathbf{U}\mathbf{S}\mathbf{V}^T = (\mathbf{U}\sqrt{\mathbf{S}})(\sqrt{\mathbf{S}}\mathbf{V}^T) = (\mathbf{U}\sqrt{\mathbf{S}})(\mathbf{V}\sqrt{\mathbf{S}})^T. \tag{5.3}$$

A row of $\mathbf{U}\sqrt{\mathbf{S}}$ corresponds to a capability. A row of $\mathbf{V}\sqrt{\mathbf{S}}$ corresponds to a location. Their dot product is the score for the capability at the location. The required storage for this calculation method is $k(c+l)$ where k is the truncation rank.

This method is efficient and compact for non-changing data. However, a location capability inference system may need to handle very rapid change, with three causes:

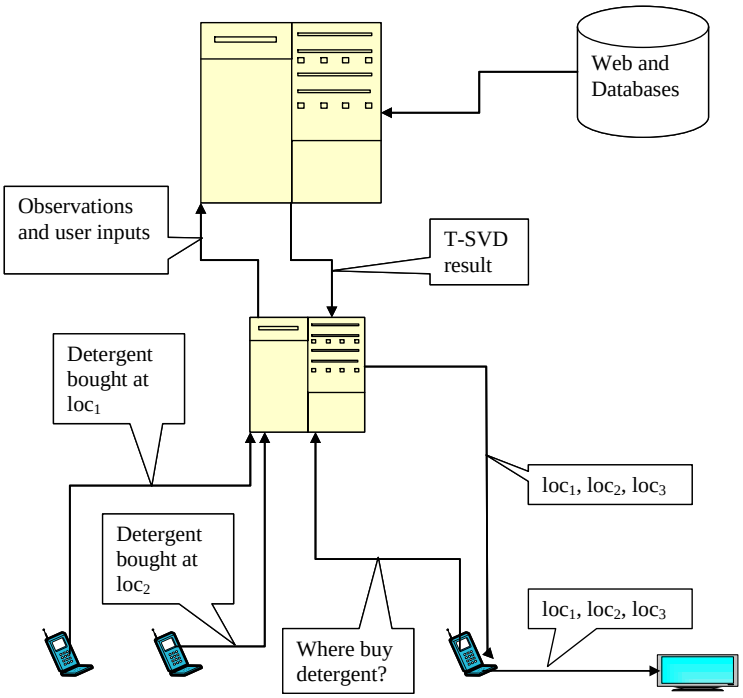

Figure 5.6: The front-end server that communicates with Joe's phone also communicates with a back end inference engine server. The inference engine collects data from front-end servers and other sources. Periodically, it distributes updates to the front-end servers. In his car, Joe's phone passes reminder data to his navigation system.

- Additional data about existing, unchanged, locations.
- Additional data about locations that are changing. New facilities open and old ones close. It will be necessary to have some form of forgetting to reduce the influence of old, possibly out-of-date observations, compared to new ones.
- New locations.

Folding-in is a technique for handling new documents and terms in latent semantic indexing [DDL$^+$90, TS07]. It adds rows to **U** and **V**, without rebuilding the relationships. However, even folding-in may be too static for the capability inference problem.

First observe that any column of **Y** can be reproduced by multiplying together **U**, $\mathbf{U}^T$, and the corresponding column of **X**:

$$\mathbf{U}\mathbf{U}^T \begin{pmatrix} 40 \\ 26 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 40.4933 \\ 25.2076 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{5.4}$$

This operation can be applied to any vector of capability observations, giving a corresponding vector of scores. Suppose the observations include 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 vector of capability observations is (0, 1, 0, 0, 0, 1). Then:

$$\mathbf{U}\mathbf{U}^T \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.4487 \\ 0.2793 \\ 0.1698 \\ 0.0837 \\ 0.0193 \\ 0.0376 \end{pmatrix}. \tag{5.5}$$

The cost of this technique is that it requires the observation vector for each location. That may be quite acceptable in a geographically distributed system with sparse, but frequently changing, observation vectors.

Equation 5.5 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. These issues are explored further in Chapter 6

To use this, 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. [Lin00]. Work in latent semantic indexing uses particularly large matrices, representing large collections of documents. For example, one of the test data sets used in a study of rank selection contains 5 million documents and 1,809,597 unique terms [Bra08]. That is equivalent to having 5 million locations and 1,809,597 capabilities that have each been observed at two different locations. Work in this area includes algorithms for updating a decomposition to reflect new data [ZS99, TS08]

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.

5.4 Parameter Selection

The algorithm has three key parameters, the truncation rank, the weight assigned to each combination of pseudo-location and capability in the seed data, and the lower bound on scores that indicate the location has the capability.

5.4.1 Truncation Rank

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. The problem of rank selection far from solved even in the field of latent semantic indexing, despite extensive research. A 2008 paper on the topic concludes "The experimental results reported here provide further evidence that the question of representational fidelity of LSI spaces as a function of dimension is a complex issue." [Bra08].

The current implementation requires the truncation rank as a parameter. Table 5.3 shows the first 15 singular values for the basic compound workload with seed data and 2,000 training data samples. There is a significant drop in singular values between rank 8 and rank 9, so 8 is the natural truncation rank for this workload.

5.4.2 Score Bound

The output from the learning process is a real number for each combination of location and capability, the score for that combination. High scores indicate the location is likely to have the capability. Scores close to zero do not.

The truncated singular value decomposition produces, in effect, a floating point function $f(l,c)$ where l is a location and c is a capability. A reminder program must commit to a definite choice as to whether to treat the current location as having a given capability or not. It needs a Boolean function $b(l,c)$ that is true if the algorithm indicates the location has the capability, false if it does not.

The original algorithm treats all results greater than 10^{-6} as true, all smaller results as false. That is, $b(l,c) = (f(l,c) > 10^{-6})$. This threshold does exclude some

Table 5.3: Singular Values for 2000 Training Samples

Rank	Value
1	69527
2	64203
3	55507
4	45706
5	45431
6	44023
7	36878
8	24759
9	13
10	8
11	7
12	6
13	6
14	6
15	6

non-zero values that are due to floating point rounding, but it is very arbitrary and does not adjust with volume of training data. The new approach defines the threshold as a function of the lowest score assigned to any positively trained capability for the shop.

Subsequent testing showed that this fixed value was too inflexible. I reran the basic tests with a new system for calculating the Boolean. For each location, there is a set of capabilities it exhibited in the training data. Let x be the score at location l for the capability in that set with the lowest score. Then the algorithm infers a capability is available at l if, and only if, its score for l is greater than $0.99x$.

It may be useful to adjust the bound 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.

5.4.3 Seed Data Weight

The seed data contains a pseudo-location for each cluster of capabilities. The pseudo-location appears in the training data with each of its capabilities at a fixed weight, the seed data weight. The initial tests used 10,000, but this may be too big relative to the amount of actual training data. The tests where rerun with a weight of 1,000.

5.4.4 Parameter Selection Results

Table 5.4 shows results for four combinations:

SVD-Y The SVD-Y algorithm with the original parameters.

DynamicBound SVD-Y with the score bound changed to $0.99x$, where x is the lowest score for any training data item for the current location.

Seed1000 The SVD-Y algorithm with the seed data weight changed from 10,000 to 1,000.

NewBase Seed1000 with the score bound changed to $0.99x$, where x is the lowest score for any training data item for the current location.

Table 5.4: Summary of Basic Compound Parameter Selection

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Parameter Selection (Figure A.5)								
SVD-Y	1.00	1.00	100	100	0.96	1.00	2,000	2,000
DBound	1.00	1.00	100	100	0.96	1.00	2,000	2,000
Seed1000	1.00	1.00	100	100	0.96	1.00	2,000	2,000
NewBase	1.00	1.00	100	100	0.96	1.00	2,000	2,000

As suggested by the summary results, and confirmed by the charts in Figure A.5, all four methods performed equally well. NewBase continued to perform well on more complex workloads. Accordingly, NewBase was chosen as the base method for the remaining experiments, and will be referred to as “Base”.

5.5 Algorithm Extension for New and Changed Locations

A reminder system should be capable of making predictions about locations for which it has some data, even if that data was not available the last time it ran the singular value decomposition. For optimal learning, the decomposition should be done by a central server with as much data as possible. Front end servers should use the results, and recently collected local data, to answer questions from users' phones.

The front end server would have the results of a previous singular value decomposition and a vector representing a sampling of the capabilities at the new location. It needs to produce a new vector representing the estimates of availability of capabilities at the new location. Each of those vectors represents a point in a vector space with a dimension for each capability. Each coordinate in the observation vector represents the number of time the location has been observed to provide the capability. Each coordinate in the result vector is an estimate of how strongly the location shows the capability.

Each matrix represents a linear transformation between two vector spaces, as shown in Figurefig:simpSVDVectorSpaces. The capability space has one dimension for each capability. Multiplication by the left singular vector matrix $\mathbf{U}$ maps a capability space vector to the feature space, which has one dimension for each abstract feature detected by the singular value decomposition. Multiplication by the singular values matrix $\mathbf{S}$ rescales the dimension of the feature space. The final vector space, location space, has a dimension for each location in the training data. Multiplication by the right singular vector matrix $\mathbf{V}$ maps a location to the feature space.

In particular, multiplication by the left singular vector matrix, $\mathbf{U}$, maps from the capability vector space to a feature vector space with lower dimensions. Multiplication by its transpose, $\mathbf{U}^T$, maps the feature vector back to a vector in the capability space.

Multiplication by $\mathbf{U}\mathbf{U}^T$ maps a vector of observed capability weights to a vector of estimated capability scores. Front-end servers will be able to map capability observations from new locations, and updated observations from existing locations, to estimates using only the matrix $\mathbf{U}$. The algorithm for for time varying behavior, discussed in Chapter 8, uses it to estimate capability scores for a time window at a location.

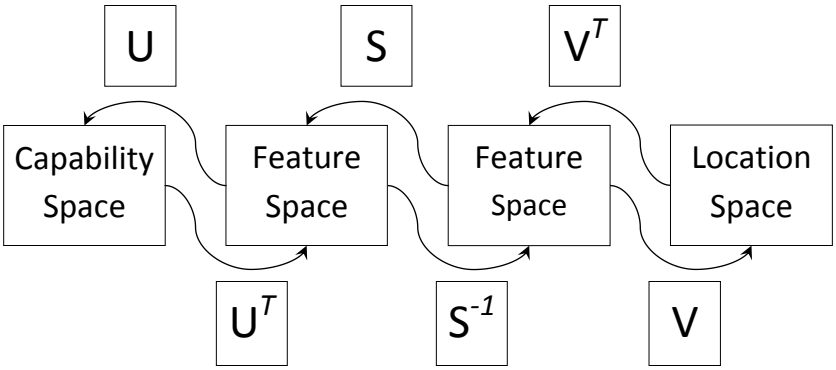

Figure 5.7: SVD Result Matrices as Vector Space Transformations

This method has serious limitations. It cannot use, or make predictions about, new capabilities. The SVD minimum error guarantees do not apply to this procedure - the new data could be of such a nature that the SVD result would have been significantly different if the new data had been used in the decomposition.

5.6 Conclusion

The basic workload results show that truncated singular value decomposition can handle some very simple cases well. The "Base" algorithm for future experiments uses the following values for the free parameters: seed weight 1000, and score bound $0.99x$, where x is the lowest score for any trained capability at the current location. The default truncation rank is 8.

The next three chapters describe tests with more difficult problems, and adjustments to the algorithm to improve handling of those cases.

5.7 Acknowledgment

Portions of this chapter are based on material that appeared in "Inferring the Everyday Task Capabilities of Locations", Patricia Shanahan and William G. Griswold,

LoCA 2007: 157-174, 2007. The dissertation author was the primary investigator and author of this material.

6 Cross-cutting Locations

Two aspects of the basic workload aid the inference algorithm. Each location stocks all or none of each cluster indicated by the seed data, and the relationships between capabilities can be fully represented by symmetrical correlation. In the real world, some locations will have behavior that contradicts the seed data, and some capabilities are related by asymmetrical implication. For example, a chain store may stock both its own store brands and corresponding national brands. Presence of the store brand implies the national brand, but there are may locations that stock the national brand without also stocking that particular store brand. This chapter introduces a harder workload, the "convenience store" workload with locations whose capabilities cut across the seed data clusters.

The convenience store workload adds 8 locations that each roughly models a convenience store by offering the 1000 most frequently used capabilities from the original data. The objective is to stress the algorithm, not to accurately model convenience stores. It does not reflect their actual biases, for example towards snack food.

There are three main objectives, in roughly descending order of priority.

- Handle the convenience stores without degrading results for other locations.
- Get reasonable overall results in the presence of convenience stores.
- Learn the convenience store behavior.

6.1 Initial Results

Table 6.1 shows results for all locations, for just the added convenience stores, and for just the locations from the Basic Compound workload, because results were

often very different for convenience stores from the others. The Base algorithm is unchanged from the previous chapter. It achieved the same results as before for the other locations. Its convenience store results show very poor precision, but good recall. Detailed examination of the results showed that it had treated the convenience stores as similar to supermarkets, stocking almost everything.

Table 6.1: Summary of Convenience Store Initial Rank 8

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
All Locations (Figure A.6)								
Base	0.85	0.92			0.95	1.00	2,000	2,000
Feedback-U	1.00	1.00	100	100	0.85	0.89		2,000
Feedback-W	0.87	1.00			0.94	1.00	2,000	2,000
Convenience Stores (Figure A.7)								
Base	0.05	0.06		200	0.88	0.99	4,000	4,000
Feedback-U	0.98	1.00	700	700	0.00	0.03		
Feedback-W	0.06	0.62			0.80	0.95	5,000	4,000
Other Locations (Figure A.8)								
Base	1.00	1.00	100	100	0.95	1.00	2,000	2,000
Feedback-U	1.00	1.00	100	100	0.95	1.00	2,000	2,000
Feedback-W	1.00	1.00	100	100	0.95	1.00	2,000	2,000

In a reminder system, this behavior would cause false alerts near convenience stores. The training data for a supermarket is dominated by the top 1000 items, making it difficult to learn the difference between the two types of locations from the positive data only. On the other hand, a reminder application could provide an option for the user to respond to a false alert.

A false prositive error is definitely a problem, but it also presents an opportunity. The user will be aware of the error when the user's phone issues an alert at a location that does not offer the required capability. The reminder application could accept feedback input from the user reporting the error. The next two algorithms, Feedback-U and Feedback-W, both model learning from such false positive feedback data.

6.2 Learning from False Positive Feedback

Each of the feedback algorithms, Feedback-U and Feedback-W, runs the decomposition twice, the first time without any feedback data, the second time with feedback data based on the results of the first decomposition. The number of feedback samples is 20% of the primary training data count.

The feedback data divides the zeros in the matrix representing the training data into two classes, those that are zero because there is no training data for the combination of location and capability, and those representing an assertion that the location lacks the capability.

Feedback-U uses a standard singular value decomposition, and represents a negative feedback sample by subtracting 10 from the training data matrix.

Feedback-W uses a weighted singular value decomposition, with two different weights. Seed data and feedback samples have weight 1.0. All other weights are a small positive number, 0.01.

Trucated singular value decomposition, for rank k, computes, from an a matrix M, the rank k matrix X that minimizes $\sum (M_{i,j} - X_{i,j})^2$. Weighted truncated singular value decomposition instead attempts to minimize $\sum W_{i,j}(M_{i,j} - X_{i,j})^2$ where W is a matrix of weights. $W_{i,j}$ is large for a false positive feedback zero, low for a zero that represents an absense of data.

The Feedback-W weighted SVD algorithm is based on the EM algorithm described by Nathan Srebro and Tommi Jaakkola in "Weighted Low-Rank Approximations" [NJ03]. The starting data for the iteration is the result of an unweighted singular value decomposition for the same input. Although Srebro and Jaakkola recommend starting with a full singular value decomposition and working down to the required rank, Feedback-W converged reasonably fast starting from rank 30.

False positive feedback data also aids selection of the score bound for determining whether to treat a score as positive. The Base algorithm uses $0.99x$, where x is the lowest score for any of the original training data at the current location. The Feedback-U and Feedback-W algorithms pick the bound to minimize the weighted count of mispredicted training data at the location. The weighting normalizes for different quantity of normal and feedback data. If there are n normal training data samples at the location, and

f feedback samples, mispredicting a feedback sample has d/f the cost of mispredicting a normal sample. If there are a pair of samples such that putting the bound between their scores would miminize the weighted cost, the bound is set at the mean of their scores. If there is no such pair, the feedback algorithms fall back to the same bound as the Base algorithm.

The results for Feedback-W in Table 6.1 show the same problem as the Base algorithm - a tendency to overestimate the convenience store capabilities, leading to low precision. Feedback-U severely underestimated the convenience store capabilities, giving better precision but much worse recall.

All three algorithms gave good results for the other locations, but none gave any indication of learning for the convenience stores. The obvious explanation is that truncation rank 8 is too low for this environment. The next experiment tests increasing the trunction rank to 9 with no other changes.

6.3 Truncation Rank Increase

Table 6.2 shows the effect of increasing the rank to 9, with no other changes in workload or algorithms. All three algorithms had significantly lower recall for the other locations, failing the first objective.

6.4 Feature Scaling

The previous results suggest that truncation rank 8 is too low, but 9 is too high, permitting over-fitting.

As described in Section 5.5, given a column vector x representing the observations at a location, and the left singular vectors matrix $\mathbf{U}$, $\mathbf{U}\mathbf{U}^T x$ estimates the scores for that location. Suppose $\mathbf{U}$ is the left singular vectors for truncation rank 9, and $\mathbf{R}$ is a diagonal matrix with main diagonal elements $[1,1,1,1,1,1,1,1,0]$. Then $\mathbf{U}\mathbf{R}\mathbf{U}^T x$ is the corresponding scores for the rank 8 decomposition.

Replacing $\mathbf{R}$ with a diagonal matrix that has a lower, but positive, value for the last main diagonal element than for the others would have the effect of reducing the

Table 6.2: Summary of Convenience Store Initial Rank 9

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
All Locations (Figure A.9)								
Base	0.81	0.95			0.45	0.50		10,000
Feedback-U	0.79	1.00			0.78	0.86		5,000
Feedback-W	0.90	1.00	8,000	8,000	0.40	0.50		
Convenience Stores (Figure A.10)								
Base	0.47	0.68			0.39	0.57		9,000
Feedback-U	0.04	0.18			0.51	0.64		
Feedback-W	0.49	0.79		10,000	0.32	0.43		
Other Locations (Figure A.11)								
Base	0.80	1.00			0.45	0.50		10,000
Feedback-U	0.81	1.00			0.79	0.86		5,000
Feedback-W	0.90	1.00			0.40	0.50		

influence of the factor corresponding to the ninth significant value, without eliminating it. The next question is how to choose the diagonal matrix $\mathbf{R}$. Directly picking numbers for the main diagonal would be unlikely to generalize to other problems. Instead, the main diagonal of $\mathbf{R}$ for truncation rank k is defined by

$$\mathbf{R}_{i,i} = \frac{k\mathbf{S}_{i,i}{}^{\alpha}}{\sum_{j=1}^{k} \mathbf{S}_{j,j}{}^{\alpha}} \tag{6.1}$$

.

The parameter α is a non-negative number that controls the relative weight to assign to features corresponding to high or low singular values. Setting $\alpha = 0$ makes $\mathbf{R}$ the identity matrix, and is equivalent to the normal rank 9 singular value decomposition. Setting α to higher values shortens dimensions with low singular values, especially the last dimension, the one that would be removed completely in a rank 8 decomposition.

This technique is related to a "tapered SVD" method that has been used in signal processing [KC94, KIC94].

Table 6.3 is a summary of the results for Feedback-W, rank 9, with α values from 0.5 to 3.0. Values 1.5 through 3 give similar results.

Table 6.3: Summary of Convenience Store SVD Feature Scaling

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
All Locations (Figure A.12)								
0.00	0.90	1.00	8,000	8,000	0.40	0.50		
3.00	0.93	1.00	6,000	6,000	0.92	0.97	2,000	2,000
2.00	0.93	1.00	6,000	6,000	0.91	0.97	2,000	2,000
1.50	0.94	1.00	6,000	6,000	0.91	0.97	2,000	2,000
1.00	0.94	1.00	6,000	6,000	0.91	0.96	2,000	2,000
0.50	0.96	1.00	6,000	6,000	0.89	0.94	3,000	2,000

Table 6.4 shows all three methods, with rank 9 and $\alpha = 2.0$. All three methods performed well for other locations, so the scaling did remove the harmful effects of increasing the rank. The high precision maximum for Feedback-U was due ignoring the convenience stores, giving them zero recall, for training data count 200. Otherwise, Feedback-W had better convenience store precision than Feedback-U, though worse recall. Feedback-W had the best combination of results for all locations, as shown in Figure A.13.

Table 6.4: Summary of Convenience Store Rank 9, $\alpha = 2$

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
All Locations (Figure A.13)								
Base	0.85	0.92			0.95	1.00	2,000	2,000
Feedback-U	0.89	1.00			0.92	0.97	2,000	2,000
Feedback-W	0.93	1.00	6,000	6,000	0.91	0.97	2,000	2,000
Convenience Stores (Figure A.14)								
Base	0.05	0.06		200	0.88	0.99	4,000	4,000
Feedback-U	0.06	1.00			0.58	0.89		
Feedback-W	0.13	0.68			0.43	0.80		
Other Locations (Figure A.15)								
Base	1.00	1.00	100	100	0.95	1.00	2,000	2,000
Feedback-U	1.00	1.00	100	100	0.95	1.00	2,000	2,000
Feedback-W	1.00	1.00	100	100	0.95	1.00	2,000	2,000

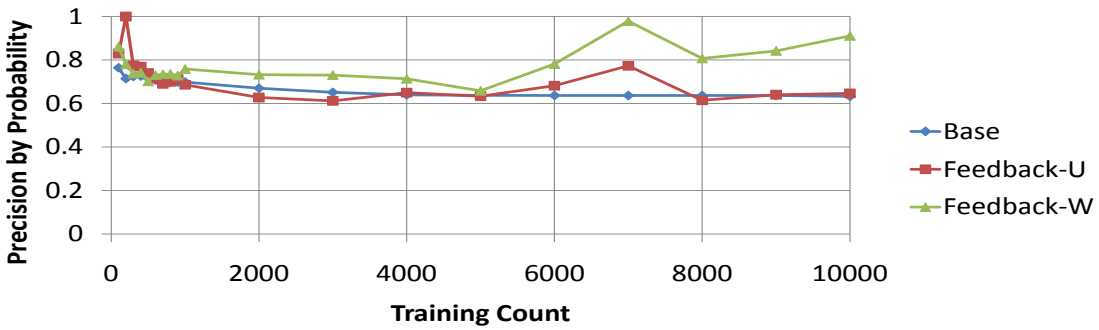

Figure 6.1: Convenience Store Rank 9, $\alpha = 2$ Precision by Probability

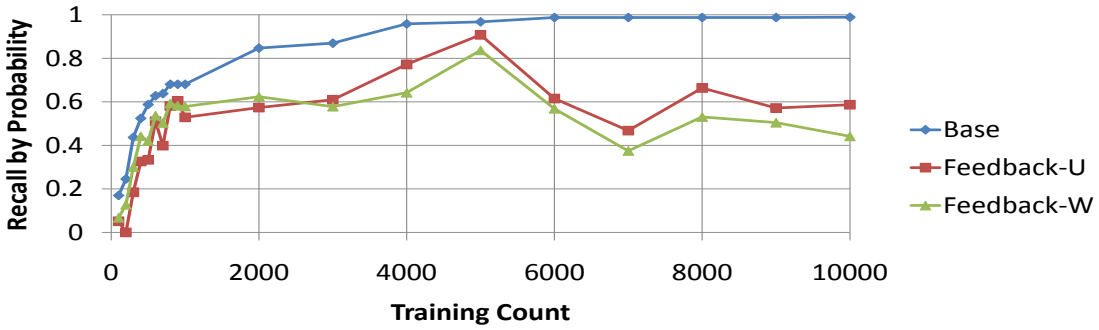

Figure 6.2: Convenience Store Rank 9, $\alpha = 2$ Recall by Probability

Feedback-U and Feedback-W with rank 9 and $\alpha = 2.0$ definitely met the first two objectives, no degradation on other locations and good overall results. As shown in Figure 6.1, Feedback-W has good precision by probability, about 90% or higher for all training counts above 6000. Figure 6.2 shows it has about 40% or higher recall by probability for all training counts above 6000. Those results are better than can be achieve by either ignoring the convenience stores or treating them as supermarkets. Success at the third objective, learning about the convenience stores, is somewhat less clear, because of the difficulty of determining whether apparent correlation between ground truth and inference merely reflects a tendency to project availability of high frequency items, regardless of whether they have been observed at the convenience stores or not.

6.5 Basic Compound Workload Revisited

Table 6.5 compares rank 8 or 9 and α 0 or 2 for the original Basic Compound workload. With α=0, equivalent to the Base algorithm, the increasing the rank to 9 produces significantly worse results, especially for recall. With α=2, both rank 8 and rank 9 work as well as the Base algorithm. Feature scaling makes the rank selection less critical.

Table 6.5: Summary of Revisited Basic Compound Workload

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Basic Comparison (Figure A.16)								
k=8, α=0	1.00	1.00	100	100	0.96	1.00	2,000	2,000
k=8, α=2	1.00	1.00	100	100	0.96	1.00	2,000	2,000
k=9, α=0	0.83	1.00			0.42	0.52		
k=9, α=2	1.00	1.00	100	100	0.96	1.00	2,000	2,000

7 False Training Data

This chapter introduces another difficult workload, intended to challenge the algorithm. Some of the data used by a real-world reminder application would be effectively random combinations of location and capability. Random combinations can arise through actual errors, such as a miss-read when scanning a cash register receipt. A random combination could also be produced by to-do list deletion. For example, a user might decide an unpurchased item is no longer needed, and delete it from their shopping list, at some location that does not stock the item. Even if the reminder application provides distinct actions for "Task completed here" and "Task canceled", the user will not necessarily apply them as intended.

The false training data experiments test the robustness of the algorithms in the presence of various quantities of completely random training data samples. The probability of picking a capability is its overall probability across all locations. The locations have equal probability. Each of the random samples picks a location and capability independently, regardless of whether the location actually has the capability.

7.1 Test Results

Table 7.1 shows results for all three algorithms from the previous section. As the proportion of bad data increases, the Base algorithm shows reduced precision. The Feedback-U and Feedback-W algorithms maintain their precision even with 10% of additional random data. The feedback mechanisms that improved the convenience store results also solve the problem of some random input data, increasing confidence that these enhancements are of general usefulness, not just a fix fr one workload.

Table 7.1: Summary of False Input

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
No Bad Data (Figure A.17)								
Base	1.00	1.00	100	100	0.96	1.00	2,000	2,000
Feedback-U	1.00	1.00	100	100	0.96	1.00	2,000	2,000
Feedback-W	1.00	1.00	100	100	0.96	1.00	2,000	2,000
0.001 Bad Data (Figure A.18)								
Base	0.99	1.00	100	100	0.96	1.00	2,000	2,000
Feedback-U	1.00	1.00	100	100	0.96	1.00	2,000	2,000
Feedback-W	1.00	1.00	100	100	0.96	1.00	2,000	2,000
0.01 Bad Data (Figure A.19)								
Base	0.89	0.99			0.96	1.00	2,000	2,000
Feedback-U	1.00	1.00	100	100	0.96	1.00	2,000	2,000
Feedback-W	1.00	1.00	100	100	0.96	1.00	2,000	2,000
0.1 Bad Data (Figure A.20)								
Base	0.55	0.91			0.96	1.00	2,000	2,000
Feedback-U	0.97	1.00	100	100	0.96	1.00	2,000	2,000
Feedback-W	0.95	0.99	100	100	0.95	1.00	2,000	2,000

8 Time Varying Behavior

The previous workloads modeled locations with fixed capabilities, regardless of time. In practice, capabilities vary on several timescales. Most shops close for at least part of each 24 hour period. Many have different opening hours on specific days of the week. Some change their stock in trade on a seasonal basis. Opening hours may vary around major holidays. Distinguishing annual changes from secular changes would require multiple years of data. Ad hoc changes from year to year further complication any attempt to learn annual behavior. Probably, the best that can be done for long-term changes is to give greater weight to recent data, so that the model adjusts reasonably fast. This chapter explores the simplest and most basic of the time inference problems, learning daily changes in capabilities.

8.1 Algorithms

The tests compare five time inference algorithms for each combination of world model and time window, NONE, SIMPLE, AUTO_1, AUTO_2, and AUTO_3.

NONE is the Base algorithm with no time inference additions. It ignores all time information, effectively treating all locations as being open 24 hours a day.

The remaining methods need training data with the time of day attached. For purposes of this test, all locations have the same capabilities every day, so a purchase of milk at 3 p.m. one day implies availability of milk at that location at 3 p.m. on every day.

SIMPLE is the most direct extension. It applies the method described in Section 5.5 to the training data samples for the time slice at the location, as though the time slice were a new location.

Different opening times for different capabilities can arise because of either different departments or different shops at the same location opening and closing at their own times. This method reflects the idea that locations not only tend to have common clusters of capabilities, they tend to open and close clusters as units. Suppose, for example, a location is known to sell butter at both noon and midnight, and has also sold a chain saw at noon. It would be reasonable to expect to be able buy milk at midnight, but not an electric drill. SIMPLE projects availability of capabilities associated with those that have been observed during the time interval.

Specifying a wider interval would produce more training data, and make SIMPLE more effective, unless the interval were so wide it included times at which the location has different behavior from during the target interval. The test cases make this a real risk, because two of the intervals end at an opening time.

8.1.1 Automatic Interval Widening

The three AUTO_* methods all automate interval widening, following the same pattern, with different parameters. Each uses a step size s. The process depends on comparing the posterior probability of the observations for a time interval t, under three different assumptions:

- The location has the same capabilities throughout a time interval starting s before the start of t and ending at the end of t.
- Time interval t stands alone.
- The location has the same capabilities throughout a time interval starting at the start of t and ending s after the end of t.

The process starts with the target interval as t. If time interval t standing alone gives the best results, the widening process is complete and that interval will be used. If either of the other options gives a better posterior probability, the new interval replaces t. The process stops when either the next widening step would make the interval longer than 24 hours, or it fails to widen the interval.

AUTO_1 uses a fixed step size s, 30 minutes. AUTO_2 and AUTO_3 both increase the interval by 30% of the current length at each step.

The posterior probability comparison is based on Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{8.1}$$

where B is the set of training data for interval t, and A is the vector of scores derived from the comparison interval. The training data B is identical for the three posteriors that need to be compared, so the $P(B)$ term will be equal and can be ignored. The calculation uses logarithms to avoid floating point underflow:

$$\log P(A|B) \propto \log P(B|A) + \log P(A). \tag{8.2}$$

Table 8.1 is a worked example of the algorithm in action, for the interval 9 a.m. to 9:30 a.m. for a shop that opens at 9 a.m. and closes at 9 p.m. At each step, $\log P(B|A)$ is the log of the probability of the observations for the candidate range, given the scores for the current base range. Extending the interval by including the 8:30 a.m. to 9 a.m. time slice would be undesirable, because the shop's behavior changes at 9 a.m. At each step, the total $\log P(B|A) + \log P(A)$ for that change is less than at least one of the total for not changing or the total for making the end of the range later. Table 8.2 is a partial worked example of the same algorithm applied to the 8:30 a.m. to 9 a.m. interval. Some repetitive steps are omitted in the middle. In this case, the time interval has to be extended earlier, but must end at 9 a.m. when the shop opens. The algorithm extends the interval by make the start earlier, until it is stopped by reaching 21:00, 9 p.m., when the shop closed the previous evening.

As described in Section 3.3, the scores are a smoothed approximation to the training data. In reality, training data events do not happen independently. A user shopping trip will result in groups of purchases, often of items that are related through their intended uses. For purposes of interval widening, I treat the training data events as being independent samples from a Poisson distribution associated with the combination of location, capability, and time of day. In this model, the score for a combination of location, capability, and interval is an estimate of the 24 hour arrival rate of the associated Poisson distribution.

Table 8.1: Time example - 9 a.m. to 9:30 a.m.

Range	$\log P(B\|A)$	$\log P(A)$	Total
Base := [9:00,9:30)			
[9:00,9:30)	-6.90641	-3.00000	-9.90641
[8:30,9:30)	-7.07417	-2.00000	-9.07417
[9:00,10:00)	-6.98852	-2.00000	-8.98852
Base := [9:00,10:00)			
[9:00,10:00)	-17.8565	-2.00000	-19.8565
[8:30,10:00)	-18.0444	-1.00000	-19.0444
[9:00,10:30)	-17.8739	-1.00000	-18.8739
Base := [9:00,10:30)			
[9:00,10:30)	-22.8091	-1.00000	-23.8091
[8:30,10:30)	-22.9401	-0.522879	-23.4629
[9:00,11:00)	-22.8108	-0.522879	-23.3337
Base := [9:00,11:00)			
[9:00,11:00)	-30.1423	-0.522879	-30.6652
[8:30,11:00)	-30.2528	-0.0969100	-30.3497
[9:00,11:30)	-30.1609	-0.0969100	-30.2578
Base := [9:00,11:30)			
[9:00,11:30)	-38.9732	-0.0969100	-39.0701
[8:30,11:30)	-39.0751	-0.0969100	-39.1720
[9:00,12:00)	-39.0547	-0.0969100	-39.1516

Table 8.2: Time example - 8:30 a.m. to 9 a.m.

Range	$\log P(B\|A)$	$\log P(A)$	Total
Base := [8:30,9:00)			
[8:30,9:00)	0.00000	-3.00000	-3.00000
[8:00,9:00)	0.00000	-2.00000	-2.00000
[8:30,9:30)	-0.434295	-2.00000	-2.43429
Base := [8:00,9:00)			
[8:00,9:00)	0.00000	-2.00000	-2.00000
[7:30,9:00)	0.00000	-1.00000	-1.00000
[8:00,9:30)	-0.579059	-1.00000	-1.57906
Base := [7:30,9:00)			
[7:30,9:00)	0.00000	-1.00000	-1.00000
[7:00,9:00)	0.00000	-0.522879	-0.522879
[7:30,9:30)	-0.651442	-0.522879	-1.17432
Base := [7:00,9:00)			
[7:00,9:00)	0.00000	-0.522879	-0.522879
[6:30,9:00)	0.00000	-0.0969100	-0.0969100
[7:00,9:30)	-0.694871	-0.0969100	-0.791781
...			
Base := [21:30,9:00)			
[21:30,9:00)	0.00000	0.00000	0.00000
[21:00,9:00)	0.00000	0.00000	0.00000
[21:30,9:30)	-0.832398	0.00000	-0.832398
Base := [21:00,9:00)			
[21:00,9:00)	0.00000	0.00000	0.00000
[20:30,9:00)	-1.66769	0.00000	-1.66769
[21:00,9:30)	-0.833845	0.00000	-0.833845

$$\log P(A|B) \propto \sum_{c \in C} \log f(k_c; \lambda_c) + \log P(A). \tag{8.3}$$

Where C is the set of capabilities, k_c is the number of observations of c at the location during interval t, λ_c is the score for capability c in the results for the interval being tested, normalized to the length of t, and $f(k;\lambda)$ is the probability of exactly k arrivals for a Poisson distribution with expected number of arrivals λ.

The remaining term, $P(A)$, is the prior probability that the interval over which A was calculated is a freestanding, cohesive interval for purposes of the behavior of the location. It is a reflection of cultural expectations. In a culture in which it is normal for some shops to close for lunch, the probability that noon through 1 p.m. is a distinct interval is higher than it would be in a culture where closing for lunch is abnormal. The algorithms that were tested all use prior distributions that depend only on the length of the interval. AUTO_1 and AUTO_2 only use an externally supplied distribution shown in Table 8.3. Only the ratios between the weights affect the comparisons.

Table 8.3: Duration Prior Weights

Duration	Prior Weight
length $<$ 1 hour	0.001
1 hour $\leq$ length $<$ 1.5 hours	0.01
1.5 hours $\leq$ length $<$ 2 hours	0.1
2 hours $\leq$ length $<$ 2.5 hours	0.3
2.5 hours $\leq$ length $<$ 3 hours	0.8
3 hours $<$ length	1.0

AUTO_3 uses a more complicated algorithm. It also considers, across all locations, the probability of the observations in interval t given the scores for the interval under evaluation. In theory, this should enable learning the general rules about how locations behave from a larger set of data.

8.2 Workloads

The workloads use four world models:

Time 0 The Basic Compound location model, but with all locations closing at 9 p.m. and opening at 9 a.m.

Time 1 Based on Basic Compound. Change the locations have all capabilities to also open early, from 7 a.m. to 9 a.m., for clusters 1, 2, and 3 only.

Time 2 Based on Basic Compound. Add one location that has all capabilities and is open 24 hours a day.

Time 3 Combines the two complications from Time 1 and Time 2.

The tests measure results for four different half hour windows:

Night 12:30 a.m to 1 a.m. All locations except except the 24 hour all-capabilities locations are closed.

Early Morning 6:30 a.m. to 7 a.m. The 24 hour location is open, but all other locations are closed. The early opening locations are about to open.

Morning 8:30 a.m. to 9 a.m. The 24 hour location is open, the early opening locations have their limited capabilities, all other locations are closed.

Day Noon to 12:30 p.m. All shops are fully open.

8.3 Results

8.3.1 Time 0

All locations are closed except during the day. The recall is always 1.0 when no capabilities are available, because there is no opportunity to fail. NONE has zero precision when the locations are closed, because it reports availability of some capabilities, and those reports are false. On the other hand, NONE has good results for the Day. The other methods all lose some recall, AUTO_1 getting the best results.

Table 8.4: Summary of Time 0 Tests

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Night (Figure A.21)								
NONE	0.00	0.00		100	1.00	1.00	100	100
SIMPLE	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_1	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_2	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_3	1.00	1.00	100	100	1.00	1.00	100	100
Early Morning (Figure A.22)								
NONE	0.00	0.00		100	1.00	1.00	100	100
SIMPLE	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_1	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_2	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_3	1.00	1.00	100	100	1.00	1.00	100	100
Morning (Figure A.23)								
NONE	0.00	0.00		100	1.00	1.00	100	100
SIMPLE	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_1	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_2	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_3	1.00	1.00	100	100	1.00	1.00	100	100
Day (Figure A.24)								
NONE	1.00	1.00	100	100	0.96	1.00	2,000	2,000
SIMPLE	1.00	1.00	100	100	0.49	0.69		7,000
AUTO_1	1.00	1.00	100	100	0.73	0.90	10,000	5,000
AUTO_2	1.00	1.00	100	100	0.51	0.69		7,000
AUTO_3	1.00	1.00	100	100	0.49	0.69		7,000

8.3.2 Time 1

Time 1 has a location that opens early for some capabilities, differing from Time 0 only in its morning behavior. NONE predicted too many capabilities during that time. The remaining methods all had high precision and moderate recall. AUTO_1 did best for recall, but had a slight reduction in precision.

Table 8.5: Summary of Time 1 Tests

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Night (Figure A.25)								
NONE	0.00	0.00		100	1.00	1.00	100	100
SIMPLE	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_1	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_2	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_3	1.00	1.00	100	100	1.00	1.00	100	100
Early Morning (Figure A.26)								
NONE	0.00	0.00		100	1.00	1.00	100	100
SIMPLE	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_1	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_2	1.00	1.00	100	100	1.00	1.00	100	100
AUTO_3	1.00	1.00	100	100	1.00	1.00	100	100
Morning (Figure A.27)								
NONE	0.07	0.07		2,000	0.93	1.00	2,000	2,000
SIMPLE	1.00	1.00	100	100	0.35	0.59		10,000
AUTO_1	0.94	1.00	10,000	10,000	0.55	0.77		8,000
AUTO_2	1.00	1.00	100	100	0.47	0.74		8,000
AUTO_3	1.00	1.00	100	100	0.36	0.59		10,000
Day (Figure A.28)								
NONE	1.00	1.00	100	100	0.96	1.00	2,000	2,000
SIMPLE	1.00	1.00	100	100	0.48	0.69		8,000
AUTO_1	1.00	1.00	100	100	0.72	0.90		5,000
AUTO_2	1.00	1.00	100	100	0.50	0.69		8,000
AUTO_3	1.00	1.00	100	100	0.48	0.69		8,000

8.3.3 Time 2

Time 2 has all six of the locations that have all capabilities open 24 hours. NONE again has unacceptable precision when more locations are closed. SIMPLE had very poor recall at night. None of the methods had high recall at night, but AUTO_3 was the best.

Table 8.6: Summary of Time 2 Tests

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Night (Figure A.29)								
NONE	0.02	0.02		3,000	0.86	1.00	3,000	3,000
SIMPLE	1.00	1.00	100	100	0.00	0.00		100
AUTO_1	1.00	1.00	100	100	0.34	0.61		
AUTO_2	1.00	1.00	100	100	0.10	0.20		10,000
AUTO_3	1.00	1.00	100	100	0.00	0.03		
Early Morning (Figure A.30)								
NONE	0.02	0.02		3,000	0.86	1.00	3,000	3,000
SIMPLE	1.00	1.00	100	100	0.21	0.31		10,000
AUTO_1	1.00	1.00	100	100	0.38	0.51		5,000
AUTO_2	1.00	1.00	100	100	0.33	0.51		10,000
AUTO_3	1.00	1.00	100	100	0.21	0.31		10,000
Morning (Figure A.31)								
NONE	0.02	0.02		3,000	0.86	1.00	3,000	3,000
SIMPLE	1.00	1.00	100	100	0.00	0.00		100
AUTO_1	1.00	1.00	100	100	0.33	0.59		
AUTO_2	1.00	1.00	100	100	0.04	0.24		
AUTO_3	1.00	1.00	100	100	0.16	0.24		4,000
Day (Figure A.32)								
NONE	1.00	1.00	100	100	0.96	1.00	2,000	2,000
SIMPLE	1.00	1.00	100	100	0.47	0.71		8,000
AUTO_1	1.00	1.00	100	100	0.71	0.89		5,000
AUTO_2	1.00	1.00	100	100	0.50	0.71		8,000
AUTO_3	1.00	1.00	100	100	0.47	0.71		8,000

8.3.4 Time 3

Time 3 has both the complications of Time 1 and Time 2, and had similar results to those tests.

Table 8.7: Summary of Time 3 Tests

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Night (Figure A.33)								
NONE	0.02	0.02		3,000	0.86	1.00	3,000	3,000
SIMPLE	1.00	1.00	100	100	0.00	0.00		100
AUTO_1	1.00	1.00	100	100	0.34	0.61		
AUTO_2	1.00	1.00	100	100	0.10	0.20		10,000
AUTO_3	1.00	1.00	100	100	0.00	0.03		
Early Morning (Figure A.34)								
NONE	0.02	0.02		3,000	0.86	1.00	3,000	3,000
SIMPLE	1.00	1.00	100	100	0.21	0.31		10,000
AUTO_1	1.00	1.00	100	100	0.38	0.51		5,000
AUTO_2	1.00	1.00	100	100	0.32	0.51		10,000
AUTO_3	1.00	1.00	100	100	0.21	0.31		10,000
Morning (Figure A.35)								
NONE	0.09	0.09		3,000	0.88	1.00	5,000	5,000
SIMPLE	1.00	1.00	100	100	0.27	0.57		9,000
AUTO_1	0.94	1.00	4,000	4,000	0.53	0.81		9,000
AUTO_2	1.00	1.00	100	100	0.42	0.65		9,000
AUTO_3	1.00	1.00	100	100	0.27	0.57		9,000
Day (Figure A.36)								
NONE	1.00	1.00	100	100	0.96	1.00	2,000	2,000
SIMPLE	1.00	1.00	100	100	0.47	0.71		8,000
AUTO_1	1.00	1.00	100	100	0.70	0.89		5,000
AUTO_2	1.00	1.00	100	100	0.50	0.71		8,000
AUTO_3	1.00	1.00	100	100	0.47	0.71		8,000

8.3.5 Conclusion

Automatic interval widening produced better results when some locations were closed than either NONE, which ignores time, or SIMPLE. NONE outperformed all the other methods when all locations are open. Among the automatic interval widening methods, AUTO_1 is the simplest and was not definitely outperformed by any of the others.

The results might be improved by a more sophisticated prior distribution function that takes into account tendencies to open and close at specific times, rather than just using the interval length.

9 Proposed Use in a Reminder System

The bulk of this dissertation discusses and evaluates a proposed inference algorithm for projecting the capabilities of locations. This chapter describes how it could be used in a location-aware reminder system. The reminder system is first described from the point of view of a user, then from the point of view of an administrator, and finally in terms of deployment of data and programs.

9.1 End-User View

The public face of the reminder system would be an application running on one or more devices for each user. The primary interface would be through an application running on a location-aware cell-phone with Internet access. That device should be so closely associated with the user that its location is a good surrogate for the user's location. The main end user interactions are reminder creation and deletion, receiving a reminder, and option selection. End users will also be concerned about privacy.

9.1.1 Reminder Creation and Deletion

The cell phone application could have a conventional GUI interface for entering and deleting reminders. Deletion needs to distinguish completion of the action that would be triggered by the reminder at the current time and location from a decision that the action no longer needs doing, or has been done elsewhere. Completion at the current time and location is input for the machine learning process.

Reminders can also be created on another computer and communicated to the cell phone. For example, a user might find it convenient to have a computer with bar

code scanner in or near the kitchen, to use the bar codes from empty, or nearly empty, boxes to create shopping reminders. A computer with a full keyboard and screen could support a more convenient GUI than would be possible on a phone, and also provide more assistance in terms of suggesting previously purchased items and interpreting inputs.

9.1.2 Receiving a Reminder

The primary delivery method for a reminder may be similar to a calendar alert on the cell phone. However, continuous location awareness has high power consumption [SKK05]. In addition, a cell phone is not a good interface for delivering reminders to the driver of a moving vehicle, yet passing near a shopping center may be an important event.

Car navigation systems may become the targets of convergence for human-computer interfacing in cars. Already, many navigation units act as Bluetooth hands free remotes for cell phones, including copying and displaying the phone address book. Similar collaboration between phone and navigation unit would enhance the reminder application. The navigation unit must maintain continuous location-awareness for its primary function. Access to the car electrical system gives it plenty of power. It has both screen and audible communication with the driver.

The phone would remain responsible for maintaining the list of reminders in terms of capabilities, and accessing servers to get the data to translate that list into a list of locations near the phone. At the start of a car trip, the phone would copy the location list to the navigation unit, which would take over detecting arrival at or near a location, and issuing reminders. The phone would continue to check location at a low frequency, in order to recalculate the locations if the car moves outside the area for which it had previously obtained data.

Car travel is the primary mode that allows both rapid movement and the flexibility to change plans based on reminders. Slower travel, such as on foot, does not cause as rapid change in location and so does not require very frequent location checking.

The user will have three possible responses to a reminder. The first is to accept it, indicating that the associated task can be done at the current location, and the reminder

is no longer needed. A false positive in the inference algorithm will cause a reminder to be issued at a location which lacks the required capability. The user will be able to indicate that, keeping the reminder active and supplying data to the inference system. The third option is to ignore the reminder, leaving it active and supplying no new data.

9.1.3 Options

The most important end-user option is level of participation. The client program can operate in either of two modes, “freeloader” and “contributor”.

In either mode, the device can make queries such as requesting the list of known locations in a geographical area. The most frequent and important form of query transmits a list of locations, a list of capabilities, and a time. The response contains one bit for each location-capability combination, indicating whether the inference system expects the capability to be available at the location at the indicated time. The response also contains the time at which the inference will change, for example because a shop opens or closes. The client device can request data covering future times, without waiting for the end of the period of validity of its current data.

These queries are sufficient for the basic reminder system, but the user’s experience will not affect the inference engine. In particular, the system will continue to be ignorant of any shop that no contributor has used, no matter how many purchases a freeloader makes at it. The client device program may remember specific capabilities at those locations, but there will be no generalization to related capabilities.

A contributor’s actions can be transmitted to servers, and affect the inference system. Deleting a reminder marking it “done here” or accepting it when issued causes the phone to report that the capability is available at the current location. Similarly, when a contributor indicates that a reminder was a false positive, the lack of the capability will be reported to a server. Depending on the contributor’s history, those reports may have a significant effect on inference results.

9.2 Privacy

The requests and reports that the client device sends to the servers necessarily reveal information about the user's location, plans, and actions.

Even the "freeloader" mode presents some privacy issues beyond those inherent in web searching. The client device requests are an indication of the user's planned itinerary and activities for the near future. The "contributor" mode presents additional issues of trust and privacy. Trust is important because of the risk of lies, either random vandalism or directed sabotage. For example, an unscrupulous business owner might falsely assert that a competitor had closed. In order to evaluate the trustworthiness of a given user's assertions, the system needs to maintain data that is tied to a user identity. There is no need to track the user's name or similar information. However, a history of locations and purchases may be enough to allow identification by someone who knows the user.

Solving these issues is beyond the scope of this dissertation, and should be based on privacy best practices at the time of implementation.

9.3 Administrator Functions

The reminder system will required administrator decisions in two areas, seed data and inference parameter setting.

9.3.1 Seed Data

The experiments show that adding seed data grouping related capabilities significantly improves recall, especially for infrequently used capabilities, and during start-up when there will be limited observations. Seed data must be supplied by the administrator. It can come from various ontologies, and from web site scraping.

9.3.2 Parameters

The most important parameter is the truncation rank. It is closely associated with the exponent used in feature scaling. The two parameters together define a trade-off be-

tween retaining detail and eliminating noise. There are several approaches to truncation rank selection, but experiments may be simple and effective for this domain. The administrator could track a number of index locations for which the actual capabilities are known, and compare results obtained with different parameter settings.

The decomposition only needs to be done for the largest rank under consideration. It can be truncated to any lower rank.

The time-based inference requires prior distributions that distinguish usual and unusual behavior. The prior distributions depend on cultural conventions. For example, the distributions will be different for cultures in which many shops close for lunch or a siesta period from one in which most shops stay open throughout the day. The prior distributions should be based on observations.

9.4 Deployment

The reminder application can be divided into a number of functions, many of which can be grouped together or spread across several physical computers, as show in Figure 5.6.

A singular value decomposition server collects data from web searches and from the location servers. It represents the data as a sparse matrix, performs truncated singular value decomposition, and supplies the results to location servers.

The singular value decomposition will generally produce more accurate results given more training data, so training data from a wide area should be pooled. On the other hand, the some of the conventions about which capabilities belong together are cultural. The same food may be stocked at every supermarket in one country, but only at specialist imported food shops in another. The singular value decomposition should be done for a region that has uniform product assortment conventions.

Given this scope limit, the matrix size is unlikely to exceed those used in latent semantic indexing, so the singular value decomposition will be feasible. Indeed, the decomposition could use the same algorithms as latent semantic indexing servers.

9.4.1 Location-Specific Services

Location-specific services can be distributed among several servers, each serving some locations. It receives and stores a copy of the relevant subset of the decomposition, the **U** columns for its locations, **S**, and the raw data for its locations. It passes a copy of the current raw data values to the singular value decomposition server as needed.

9.4.2 Web Front End

Client devices use the Internet to send requests and reports, and receive responses. The web front end, not shown in the configuration diagram, forwards each request or report to the location-specific server for the specified location. If a request covers multiple locations that are served by two or more different location-specific servers the front end will divide up the requests, forward to the appropriate servers, and combine the responses.

9.4.3 Client Device

The client device will usually be a smartphone or similar portable, location-aware device with Internet access. It will communicate with the location servers, either directly or through web front end servers.

9.5 Acknowledgment

Portions of this chapter are based on material that appeared in “Inferring the Everyday Task Capabilities of Locations”, Patricia Shanahan and William G. Griswold, LoCA 2007: 157-174, 2007. The dissertation author was the primary investigator and author of this material.

10 Conclusion

10.1 Contributions

This dissertation examined the issue of identifying the capabilities of locations, especially for the very difficult everyday shopping task. Capability identification is important for applications that issue reminders or suggestions based on availability of the capabilities needed for a task, rather than specified times or locations.

We proposed the use of truncated singular value decomposition for capability inference. We demonstrated its effectiveness in a test bed using artificial data derived from real world distributions. The algorithm blended location observations and bulk seed data about capability clusters to predict availability of capabilities that had not been directly observed.

The next workload added "convenience stores", locations whose capabilities cut across all categories. The original algorithm failed to predict their capabilities, but the results were improved by adding false-positive feedback data using weighted singular value decomposition, and increasing the truncation rank with feature scaling to prevent over-fitting. The same improvements also handled a workload including some false data.

We next examined location capabilities that depend on the time of day. Treating each time interval at each location as though it were a separate location had good precision, but low recall due to the small amount of training data for any one interval. We added automatic widening of the interval to obtain more training data, using a Poisson distribution model. We interpreted the singular value decomposition score for a combination of location and capability as an estimate of the mean rate of reported use of the capability at that location.

10.2 Advantages and Limitations of Truncated Singular Value Decomposition

Truncated singular value decomposition can effectively represent the idea of locations as exhibiting combinations of clusters of capabilities.

Its main limitation is that it deals with correlation, not one-way implications. For example, a supermarket chain might sell both its own store brands and corresponding national brands. Presence of the house brand implies presence of the national brand, but not the other way round. The "convenience store" workload illustrated this problem by having a class of locations that had only the highest frequency items from each seed cluster.

As demonstrated in that workload, false positive feedback data can mitigate the effect.

10.3 Future Work

10.3.1 Deployment

Deployment as part of a location-aware reminder system, as described in the previous chapter, would be the best test of the practical effectiveness of the inference algorithm.

10.3.2 Additional Simulation Studies

The next step within the simulation studies is to obtain and represent more realistic data. Seed data could be produced by web scraping. Some or all of the simulated locations should be replaced by observations of real locations.

Overlapping Seed Data

In the simulation tests, the seed data comprises non-overlapping clusters. The cross-cutting "convenience store" workload was tested with only the standard seed data.

Adding seed data for it would have resulted in overlaps between seed data clusters. There are two basic approaches, both of which should be tested:

- Allow overlaps.
- Avoid overlaps. If two seed clusters A and B overlap, replace them with three seed clusters, $A - B$, $B - A$, and $A \cap B$.

Avoiding overlaps would separate out, for example, supermarket store brand items from national brand items.

Observation-based Overrides

In the current simulation, inference results are reported even if they contract an observation. There are two levels of overriding:

- Global: Because of the possibility of bad input data, alleged observations cannot simply override the inference results. There must be some filtering based on user trustworthiness, or consistent reports from multiple users.
- Individual: A user's assertions about which capabilities are, or are not, available at each location should control the behavior of that user's phone, regardless of support or contradiction from other users.

Overriding could also be applied to opening time data. A positive assertion from a trusted source that a shop offering a set of capabilities is open at specified times is better evidence about the shop's location than inference from time stamped observations. Moreover, it could also be used in setting the prior probabilities for similar locations.

10.3.3 Learning about the User

Location-based reminders involve three main types of real world entities, users, locations, and capabilities. This dissertation applies machine learning to inference to the relationship between users and capabilities. There may also be opportunities to infer useful facts about the relationships between users and capabilities, and between users and locations.

For example, “cheese” on a shopping list could mean different things, and require different capabilities, depending on the user and situation. A convenience store might meet the need for some users. A supermarket would meet most users’ cheese requirements. A user planning an elaborate wine and cheese party might require a specialized gourmet cheese supplier, or at least a supermarket with a particularly extensive cheese selection. Even entering the shopping list by scanning bar codes on empty, or soon to be empty, packages does not entirely solve the problem. Joe only buys one store brand of breakfast cereal, in one size. When he scans the barcode, he means that barcode, and it can only be bought at a supermarket in a particular chain. Mary has also just finished a box of Joe’s favorite cereal, but she will replace it with any of a wide range of similar cereals.

The same word or barcode on the shopping list maps to different capabilities depending on the user. This problem could be solved in part by explicit user input. For example, a barcode scanning input system could allow the user to specify a required match precision in both size and type. A shopping list GUI could suggest clarifying options. Learning normal cases for a given user could improve usability by requiring clarification only when the user has a requirement that differs from whatever is normal for that user. Similarly, learning relationships between capabilities could support making useful suggestions for alternatives.

Similarly, a user may like or dislike particular locations, in some cases for specific tasks. For example, a user who is willing to buy packaged goods at some location may dislike the fresh produce at the same location.

Learning about the user requires tracking of user-specific data. That should be done on the user’s phone, for privacy reasons. However, some users may be willing to contribute the results, which would be useful prior distribution data for inferring other users’ preferences. For example, a large proportion of users avoiding buying fresh produce at a given location that offers it may indicate high prices or poor quality, reducing the probability that other users will want to buy produce there.

A Result Details

This appendix contains full results for all the tests discussed in the body of the dissertation. Each figure corresponds to a group of lines in a summary table in the body of the dissertation, and its caption contains a reference to the table.

A.1 Result presentation

Each experiment has a model of multiple locations, expressed in terms of the groups of items. The model generates the training data and represents ground truth for evaluation of precision and recall. The reports show the effects, for a range of training data sizes, of applying several different inference methods to each set of training data. Each experiment produces four charts, "Precision by Count", "Precision by Probability", "Recall by Count" and "Recall by Probability", with a data series on each chart for each inference method. The "by Count" charts weight every combination of location and capability equally. The "by Probability" charts weight each combination of location and capability according to the overall probability of the capability in the model.

Summary tables have one line for each series. Each summary table line shows four measures for each of precision and recall. Although a summary necessarily shows much less complete information than the actual chart, representing four charts with a few lines in a table enables convenient comparison across multiple sets of charts.

The Recall numbers in the example summary table A.1 illustrate the measures. They summarize the Recall by Count chart in Figure A.1 and the Recall by Probability chart in Figure A.2.

The first of the four measures is "Mean". It is the area under the data series in the chart, divided by the difference between the largest and smallest training data count.

Table A.1: Summary of Basic Simple Tests - Example

	Precision				Recall			
	Mean	Max	N1	N2	Mean	Max	N(0.9)	N(0.9M)
Simple								
Null	1.00	1.00	100	100	0.03	0.05		9,000
SVD-N	0.99	1.00	400	400	0.12	0.20		9,000
SVD-Y	1.00	1.00	100	100	1.00	1.00	200	200

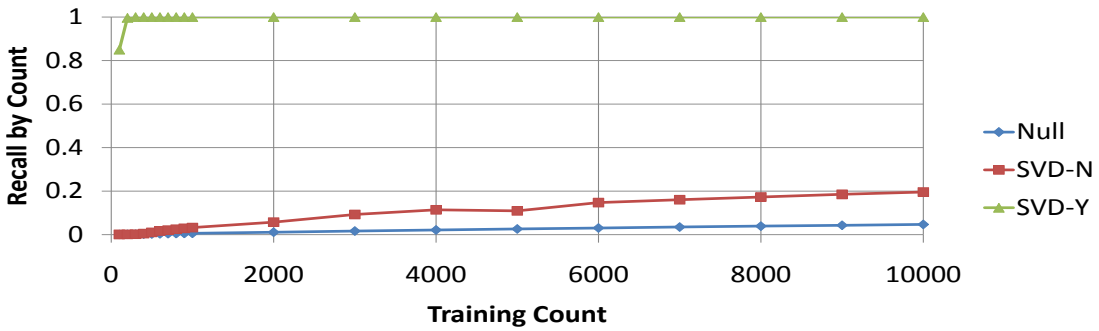

Figure A.1: Summary Example - Recall by Count

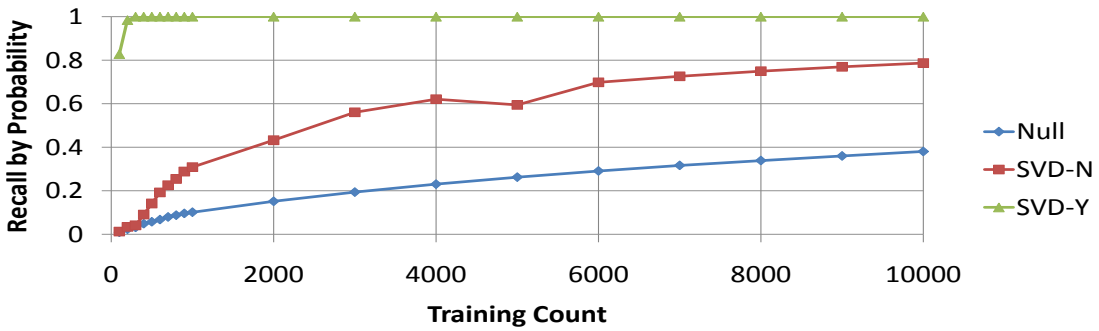

Figure A.2: Summary Example - Recall by Probability

The sum of the observed values divided by the number of observations would not be as meaningful a representation, because doing so would over-represent low training data results, separated by 100 samples, compared to large training data counts which are separated by 1000 samples. The summary shows the smaller of the means calculated from the "by count" and "by probability" charts.

Let $T = \{t_1, t_2, ..., t_N\}$ be the set of training counts. Let $f_c : T \rightarrow [0,1]$ be one of the series in either a "Precision by Count" chart or a "Recall by Count" chart. Let $f_p : T \rightarrow [0,1]$ be the series for the same method in the corresponding "by Probability" chart.

$$Mean = \min \left(\frac{\sum_{i=1}^{n-1} \left(f_c\left(t_i\right) + f_c\left(t_{i+1}\right) \right)}{2\left(t_n - t_i\right)}, \frac{\sum_{i=1}^{n-1} \left(f_p\left(t_i\right) + f_p\left(t_{i+1}\right) \right)}{2\left(t_n - t_i\right)} \right). \tag{A.1}$$

Thus the recall mean shown for "Null" is 0.03, the "by count" mean, reflecting the fact that the Null series in the Recall by Count chart never exceeds 0.05, and is lower for many training data counts. The Null series in the Recall by Probability chart has a significantly higher mean.

The mean is a reasonably good general measure of consistent effectiveness, but very much affected by the arbitrary choice of the exact range of training data counts to test. In general, removing some of the low counts and adding some larger counts would tend to increase the mean.

The next measure is "Max". It is the largest value x such that there is a training count t_i with both $f_c(t_i) \geq x$ and $f_p(t_i) \geq x$. That is, it is the best result that can be achieved with the same training data count for both "by count" and "by probability".

$$Max = \max\left(\min\left(f_c(t_i), f_p(t_i)\right)\right). \tag{A.2}$$

Again, the Recall Max value shown for the null learner is 0.05, the maximum from the Null series in the Recall by Count chart. The maximum from the Recall by Probability chart is almost 0.4.

Max may be misleading if a method did particularly well for a small set of training counts, and gives no indication of consistent effectiveness.

The next two measures, $N(0.9)$ and $N(0.9M)$, are based on the same function, $N(x)$, and do indicate consistent effectiveness. The function $N(x)$ the smallest training data count such that its result, and all results for greater training data counts, are at least x. A blank for one of the N columns in the summary table indicates that there was no value training data count meeting that condition. In other words, the set whose minimum would be displayed is empty.

$$N(x) = \min\{t_i | t_i \in T \wedge \big(\big(t_j \in T \wedge t_j \geq t_i\big) \Rightarrow \big(f_c(t_j) \geq x \wedge f_p(t_j) \geq x\big)\big)\}. \qquad \text{(A.3)}$$

The $N(0.9)$ column shows the values for $x = 0.9$. It represents the training data count required to get both "by count" and "by probability" to at least 0.9. A low value indicates very consistent effectiveness. In some cases, 0.9 is not achievable. It is also useful to ask whether the actual maximum represents a single exceptionally good result or sustained effectiveness, even if at a lower level than 0.9. The $N(0.9)$ column is based on 90% of the Max measure.

In the example summary, the $N(0.9)$ recall column is blank for the Null learner, because it never achieved recall 0.9. The "9,000" in the $N(0.9M)$ column means that both the "by count" and "by probability" recalls were at least 0.9×0.05 for training data count 9,000 or 10,000. This shows less consistency than SVD-Y, which achieved recall at least 90% of its maximum for all training counts from 200 through 10,000.

A.2 Result Charts

Each chart caption contains a reference to the corresponding summary table. The summary tables are interleaved in the text discussing the results.

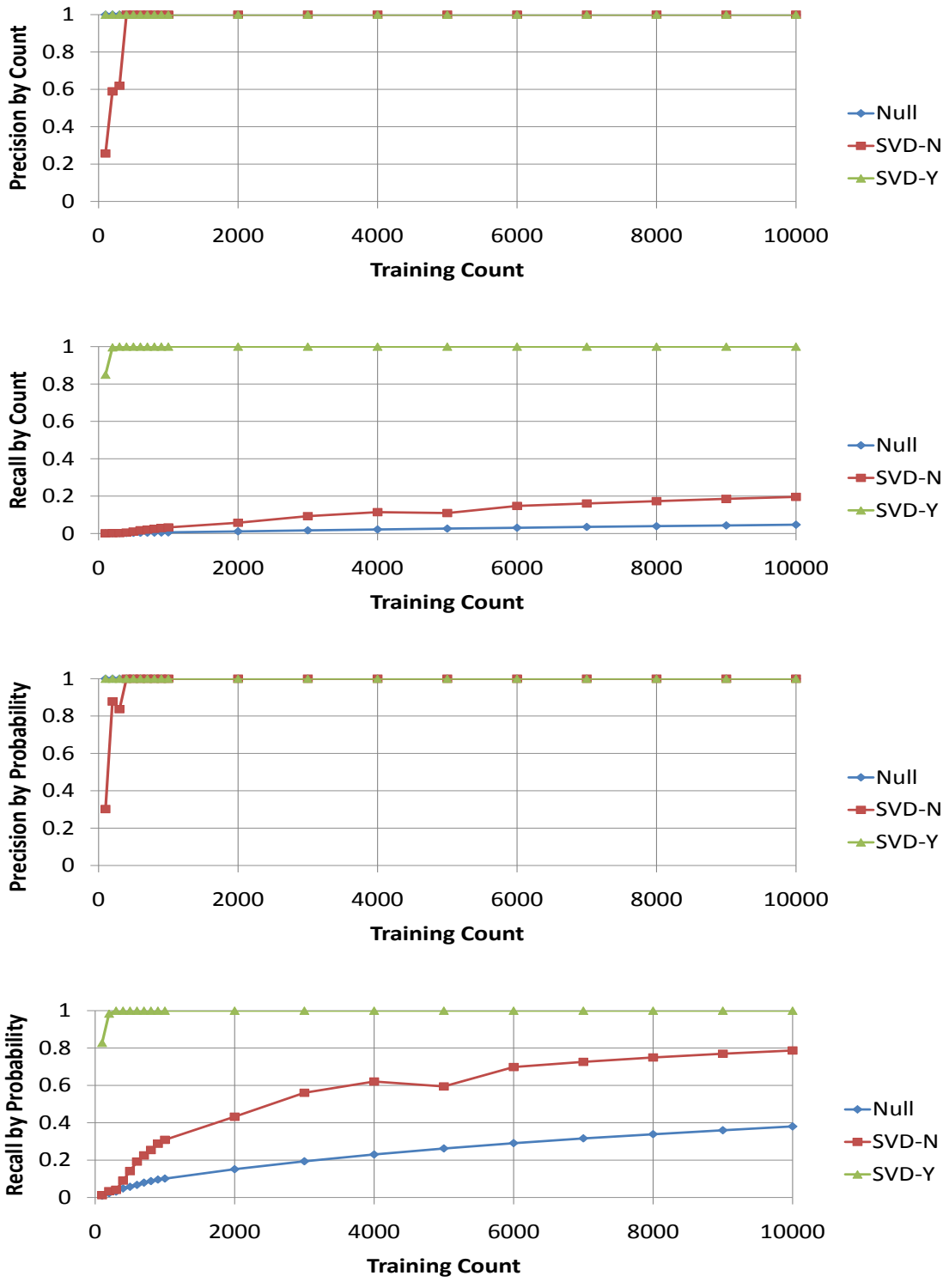

Figure A.3: Basic Tests - Simple (Table 5.2)

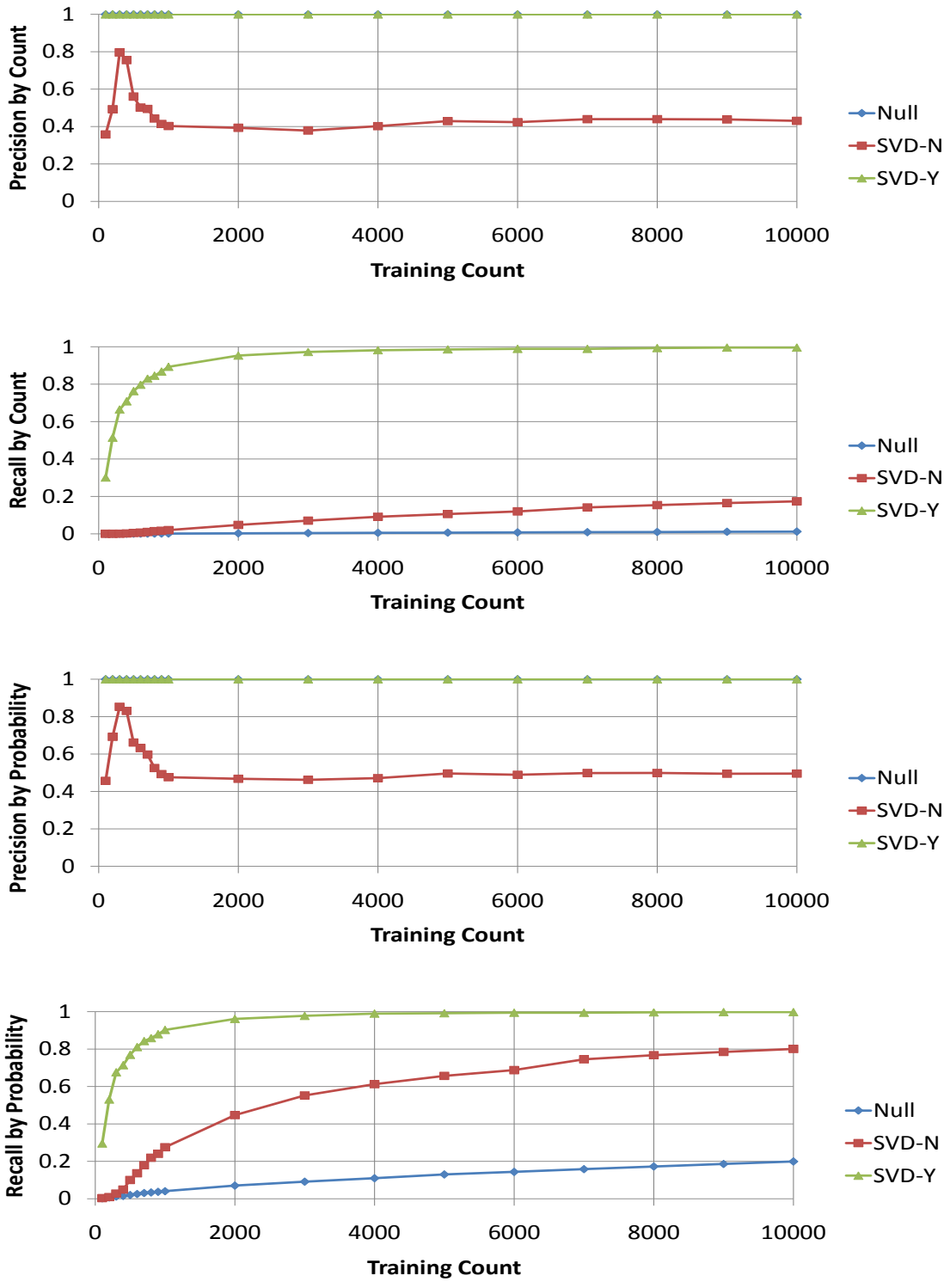

Figure A.4: Basic Tests - Compound (Table 5.2)

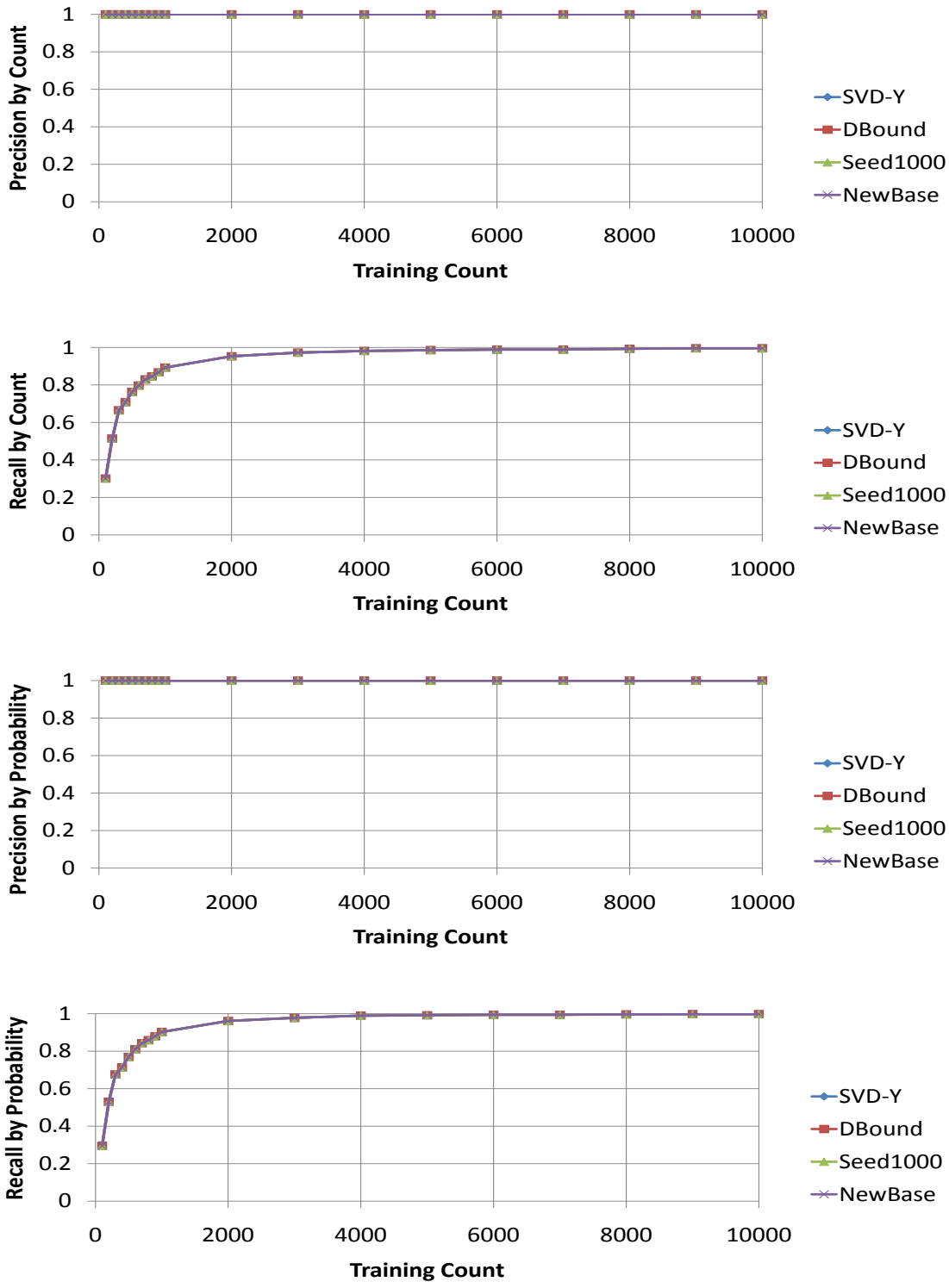

Figure A.5: Basic Compound Parameter Selection - Parameter Selection (Table 5.4)

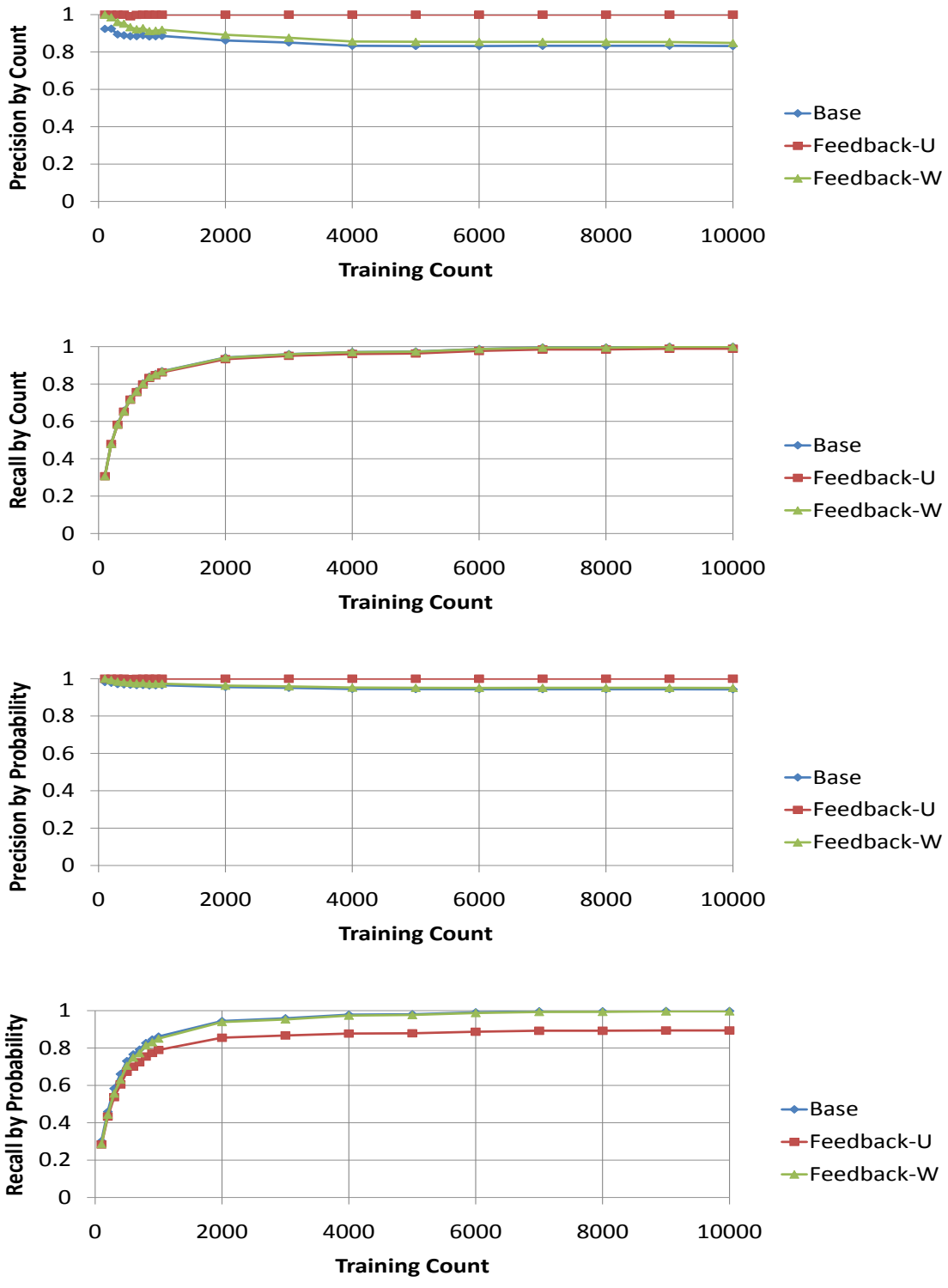

Figure A.6: Convenience Store Initial Rank 8 - All Locations (Table 6.1)

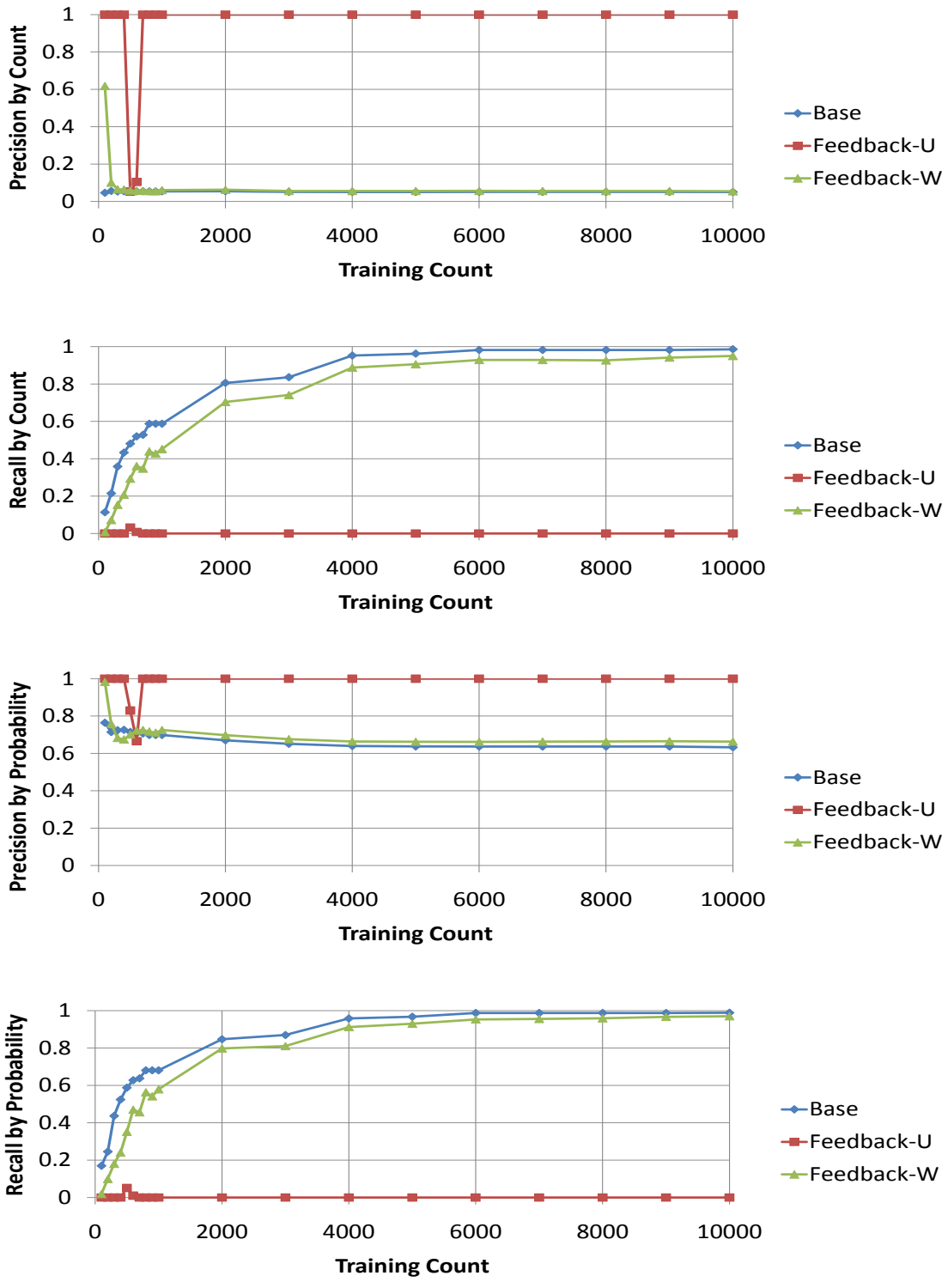

Figure A.7: Convenience Store Initial Rank 8 - Convenience Stores (Table 6.1)

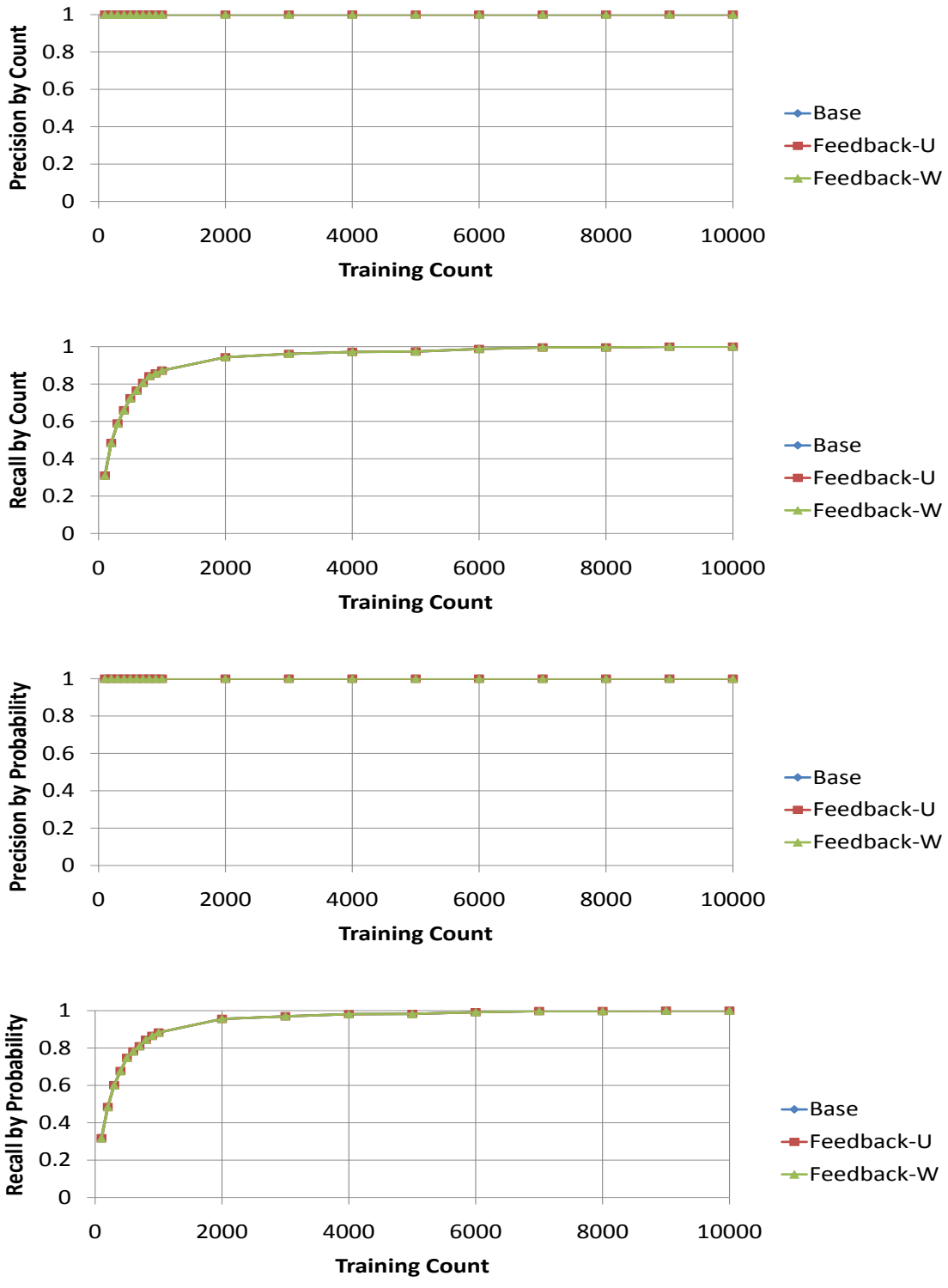

Figure A.8: Convenience Store Initial Rank 8 - Other Locations (Table 6.1)

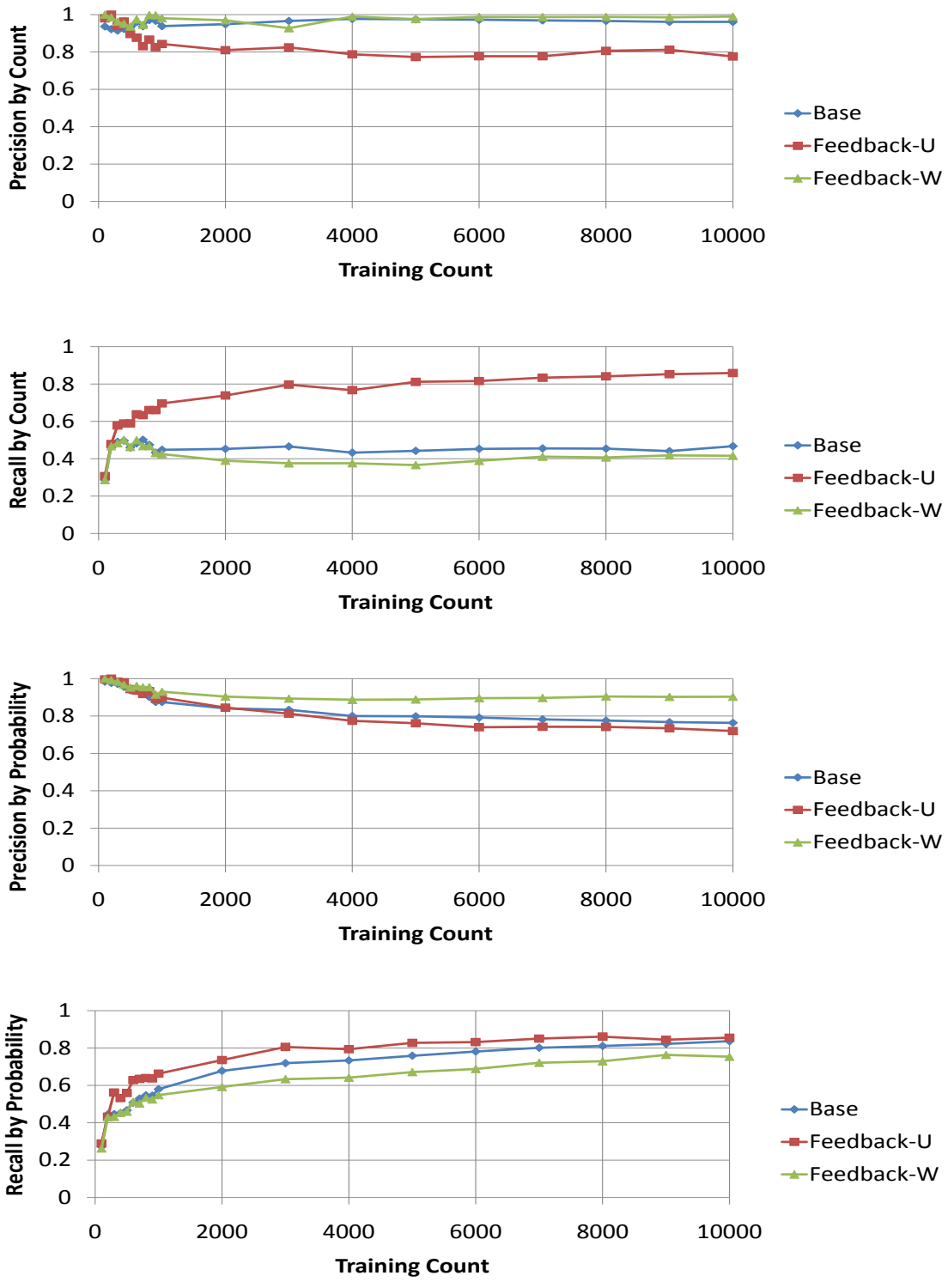

Figure A.9: Convenience Store Initial Rank 9 - All Locations (Table 6.2)

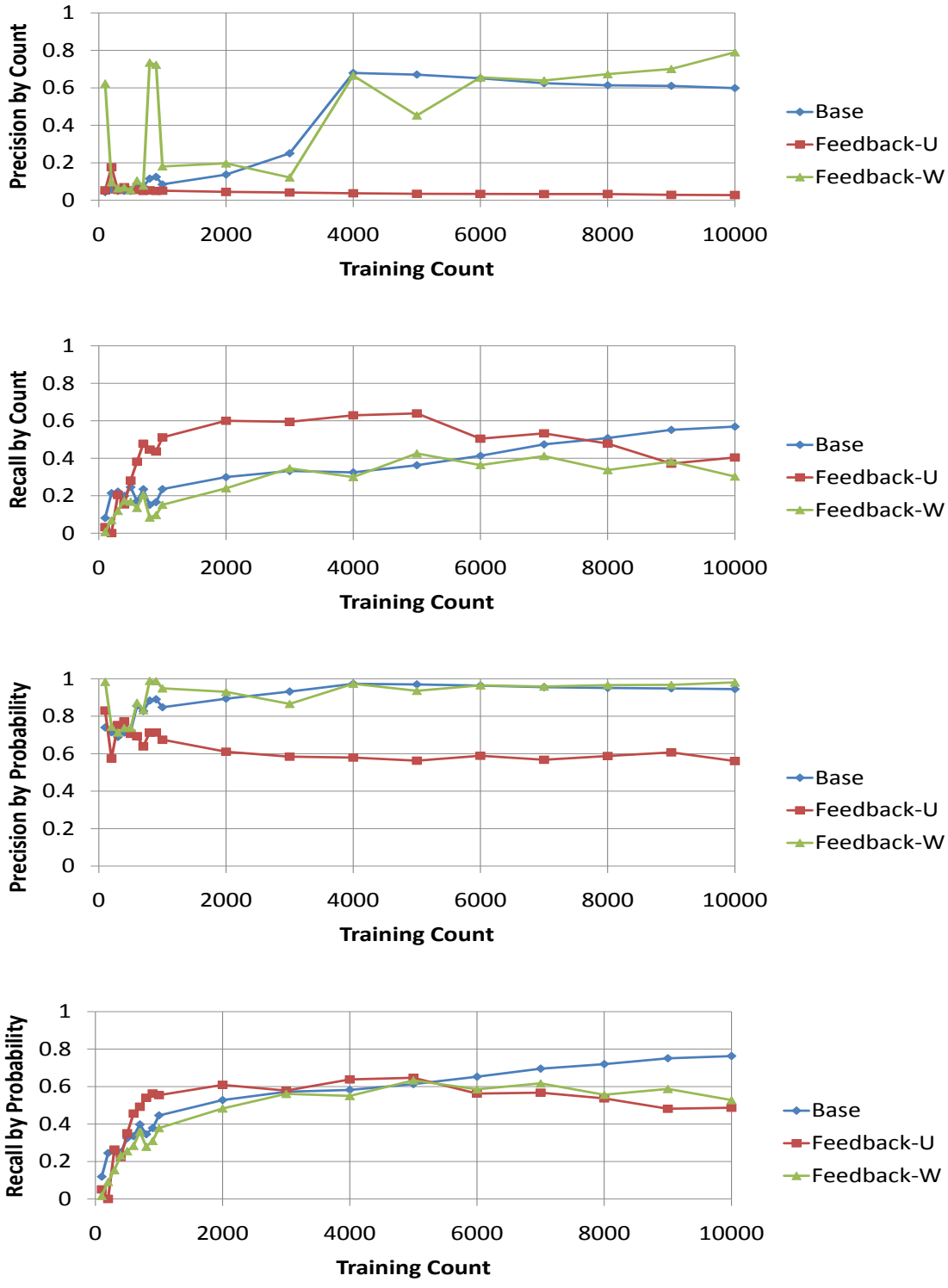

Figure A.10: Convenience Store Initial Rank 9 - Convenience Stores (Table 6.2)

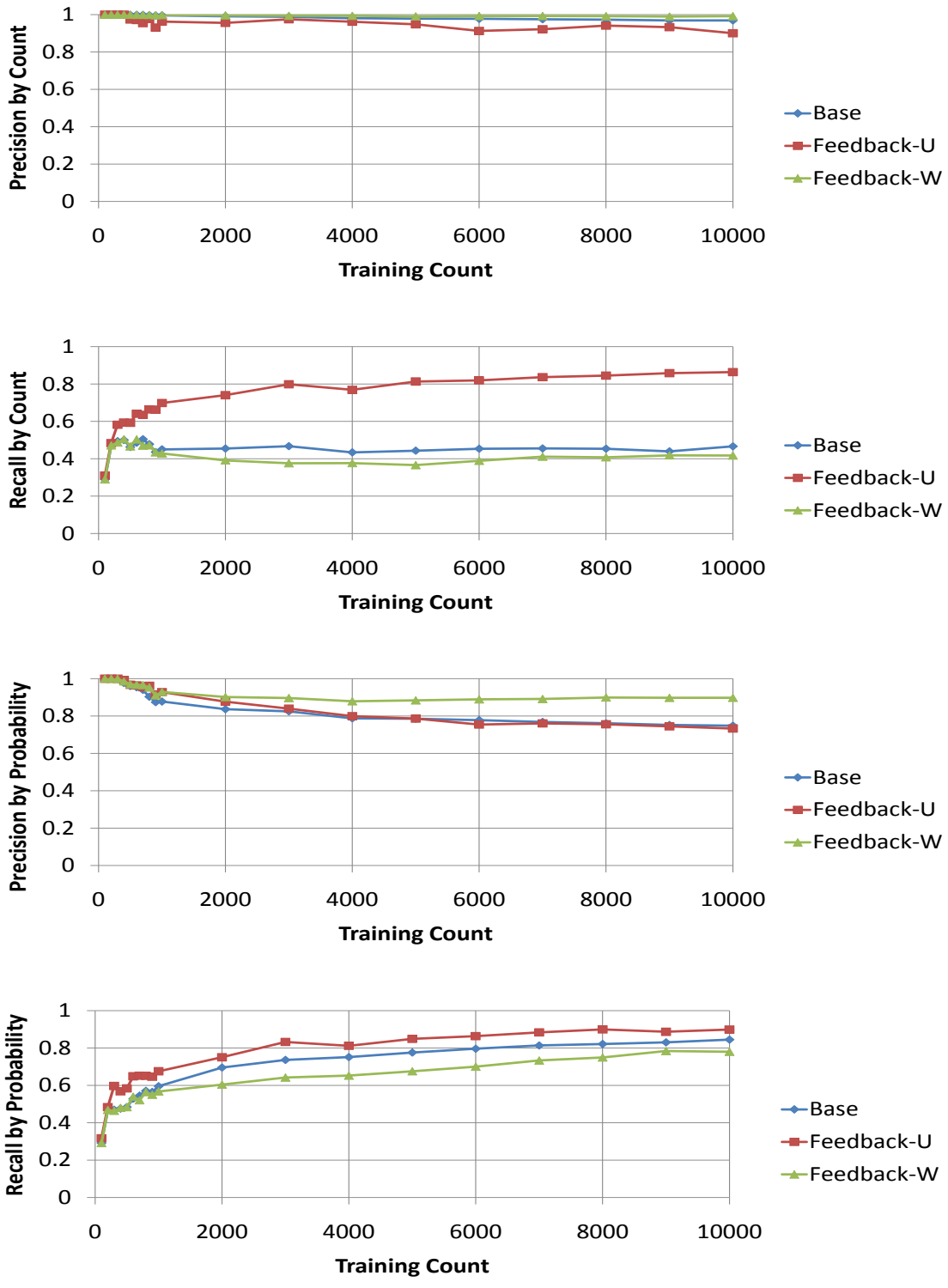

Figure A.11: Convenience Store Initial Rank 9 - Other Locations (Table 6.2)

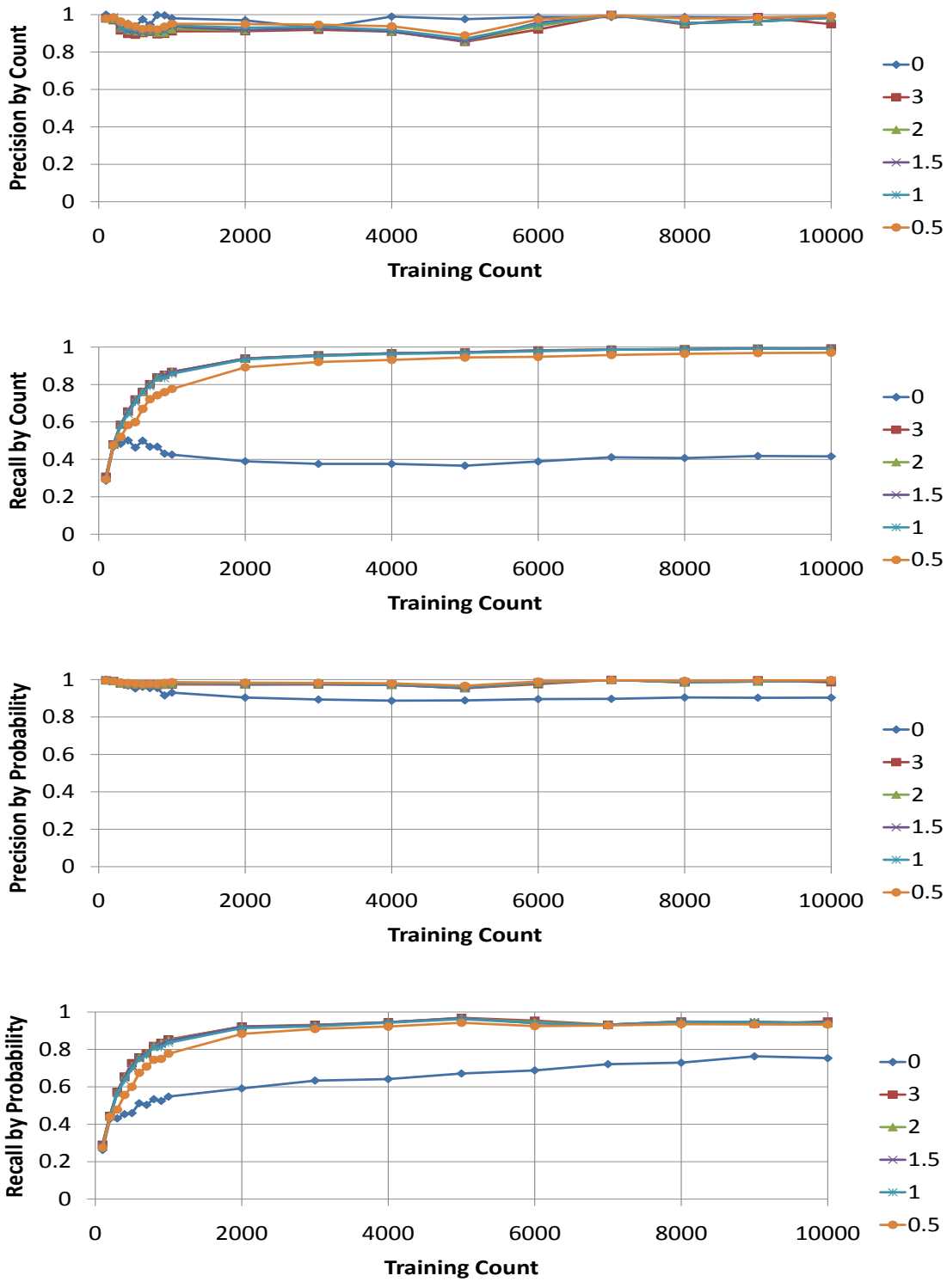

Figure A.12: Convenience Store SVD Feature Scaling - All Locations (Table 6.3)

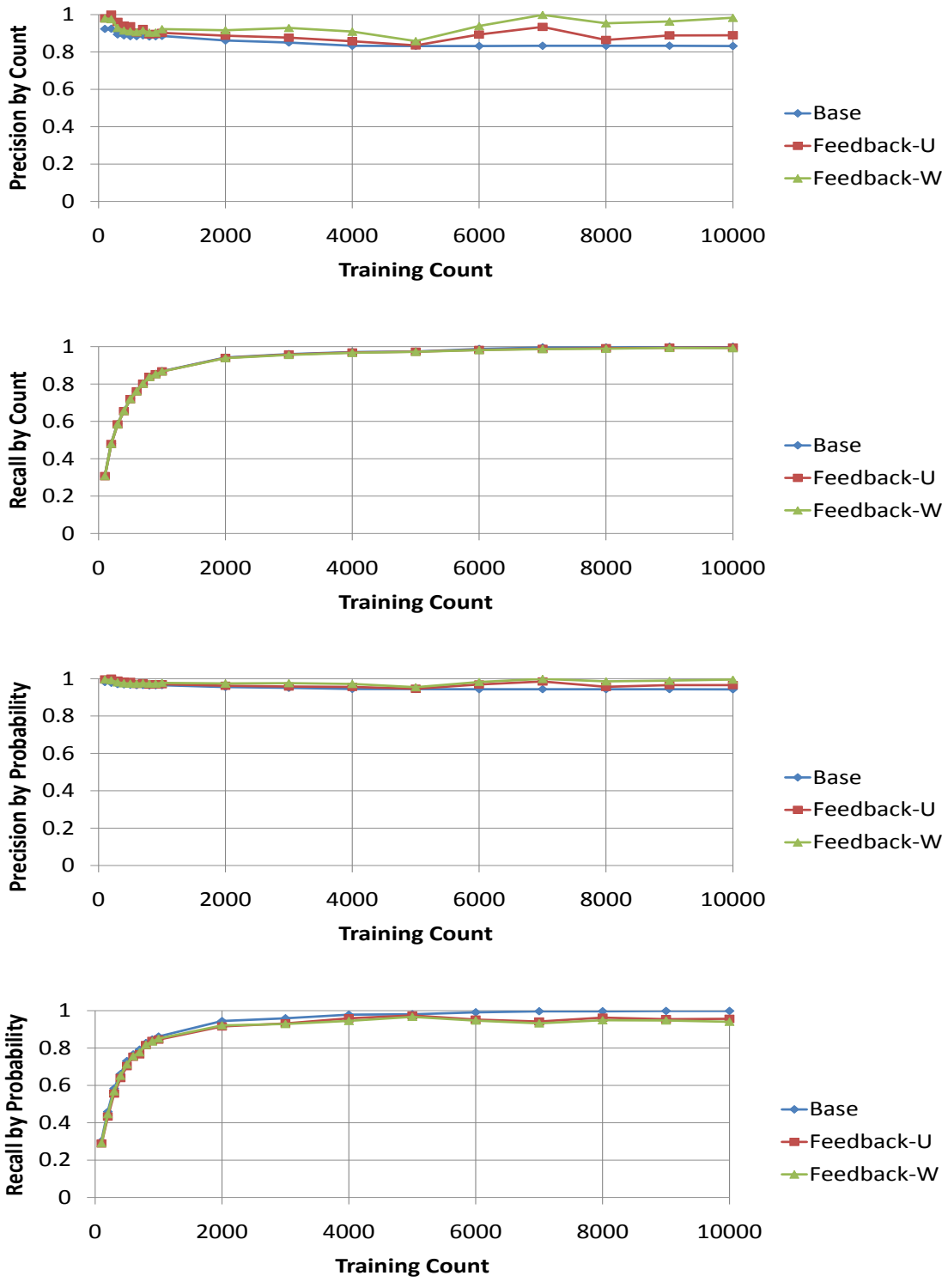

Figure A.13: Convenience Store Rank 9, $\alpha = 2$ - All Locations (Table 6.4)

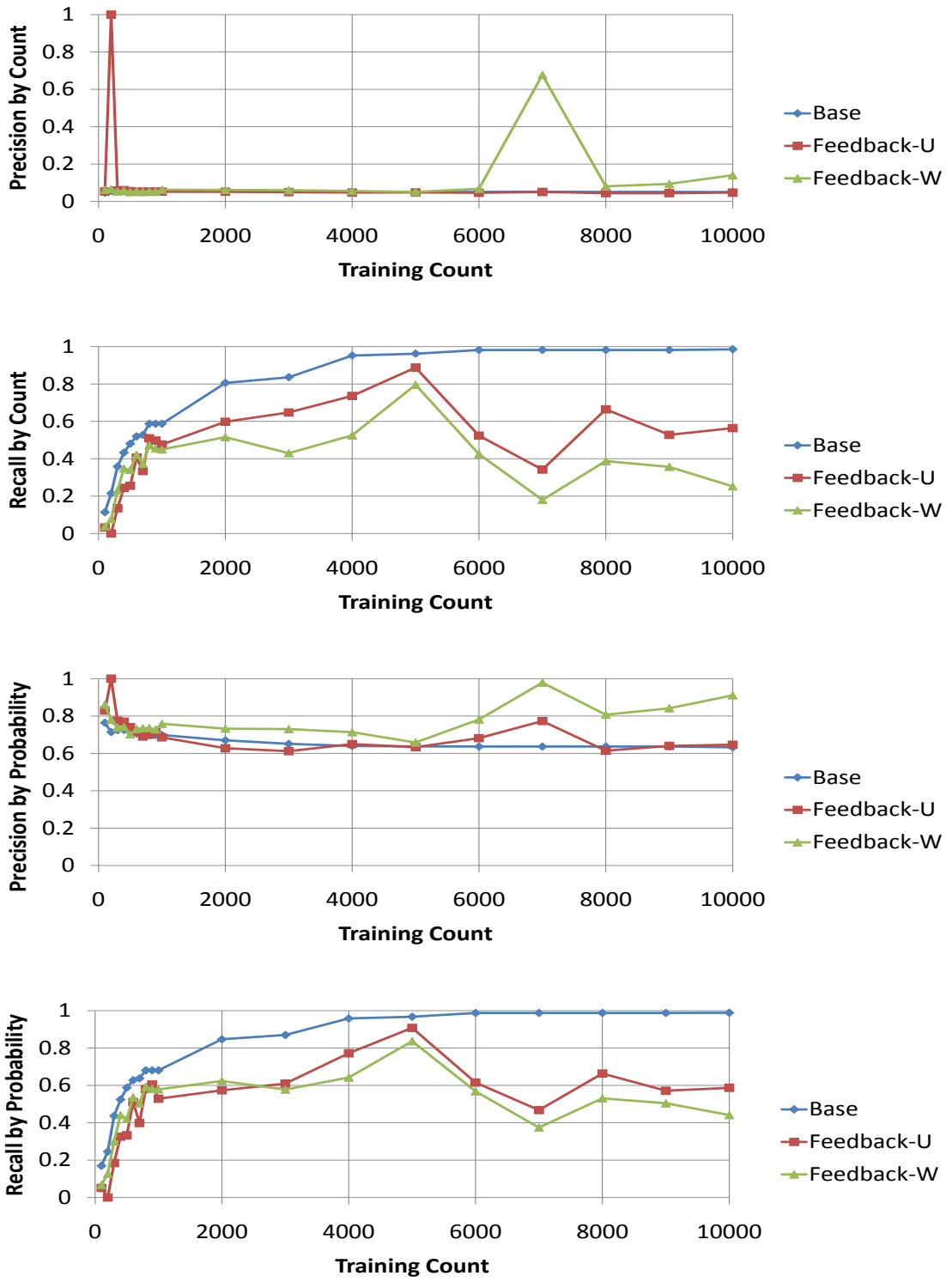

Figure A.14: Convenience Store Rank 9, $\alpha = 2$ - Convenience Stores (Table 6.4)

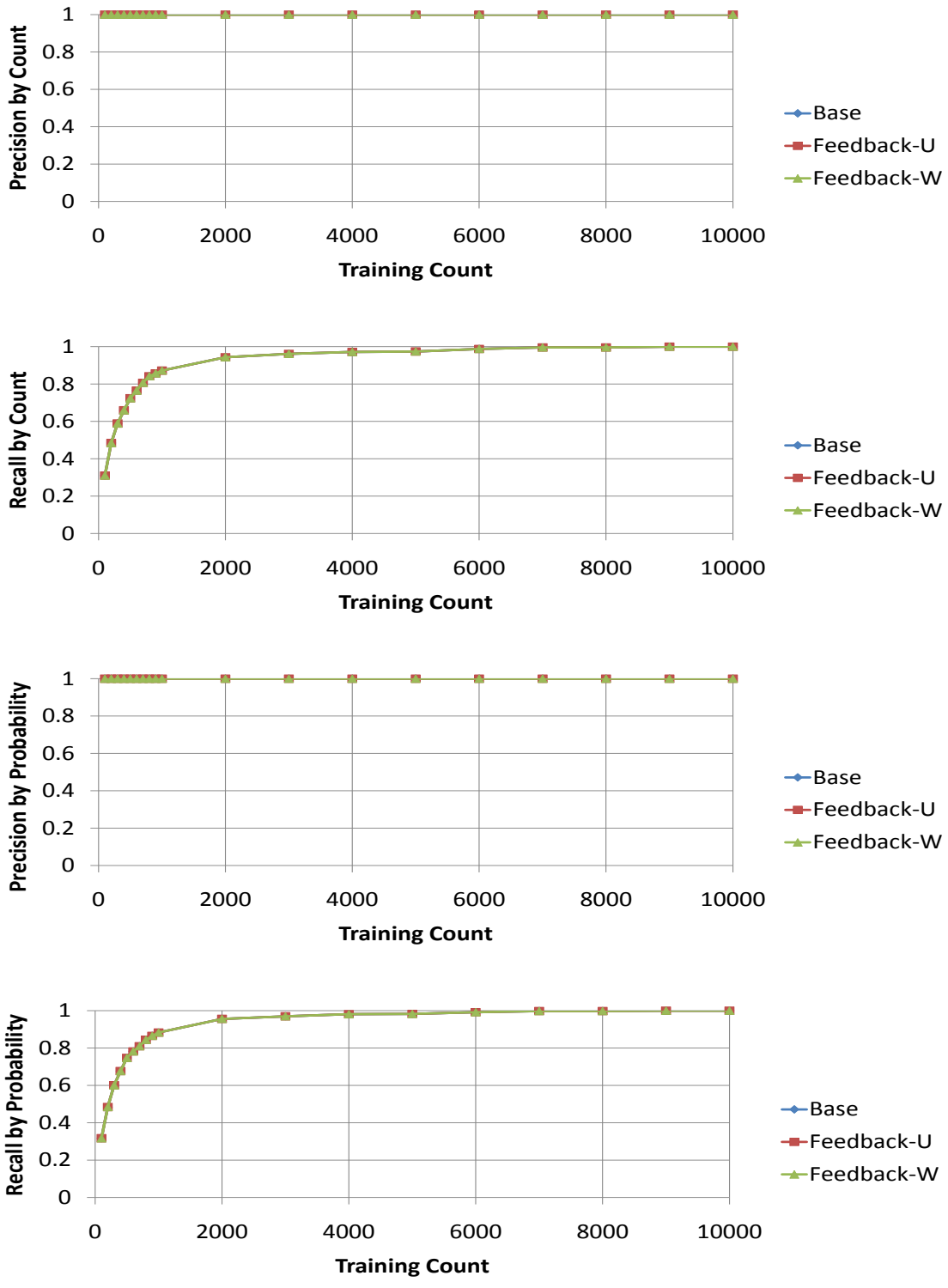

Figure A.15: Convenience Store Rank 9, $\alpha = 2$ - Other Locations (Table 6.4)

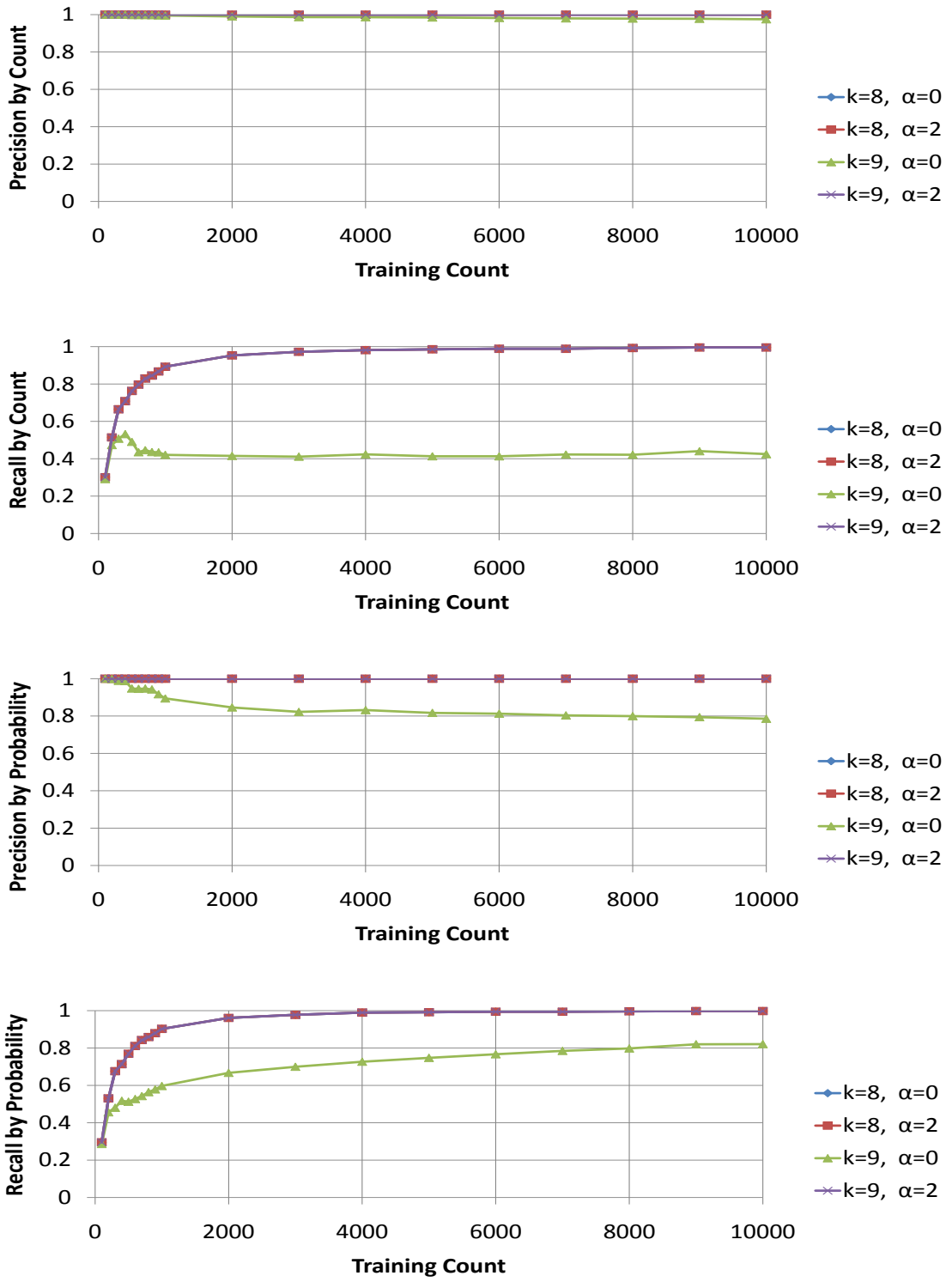

Figure A.16: Revisited Basic Compound Workload - Basic Comparison (Table 6.5)

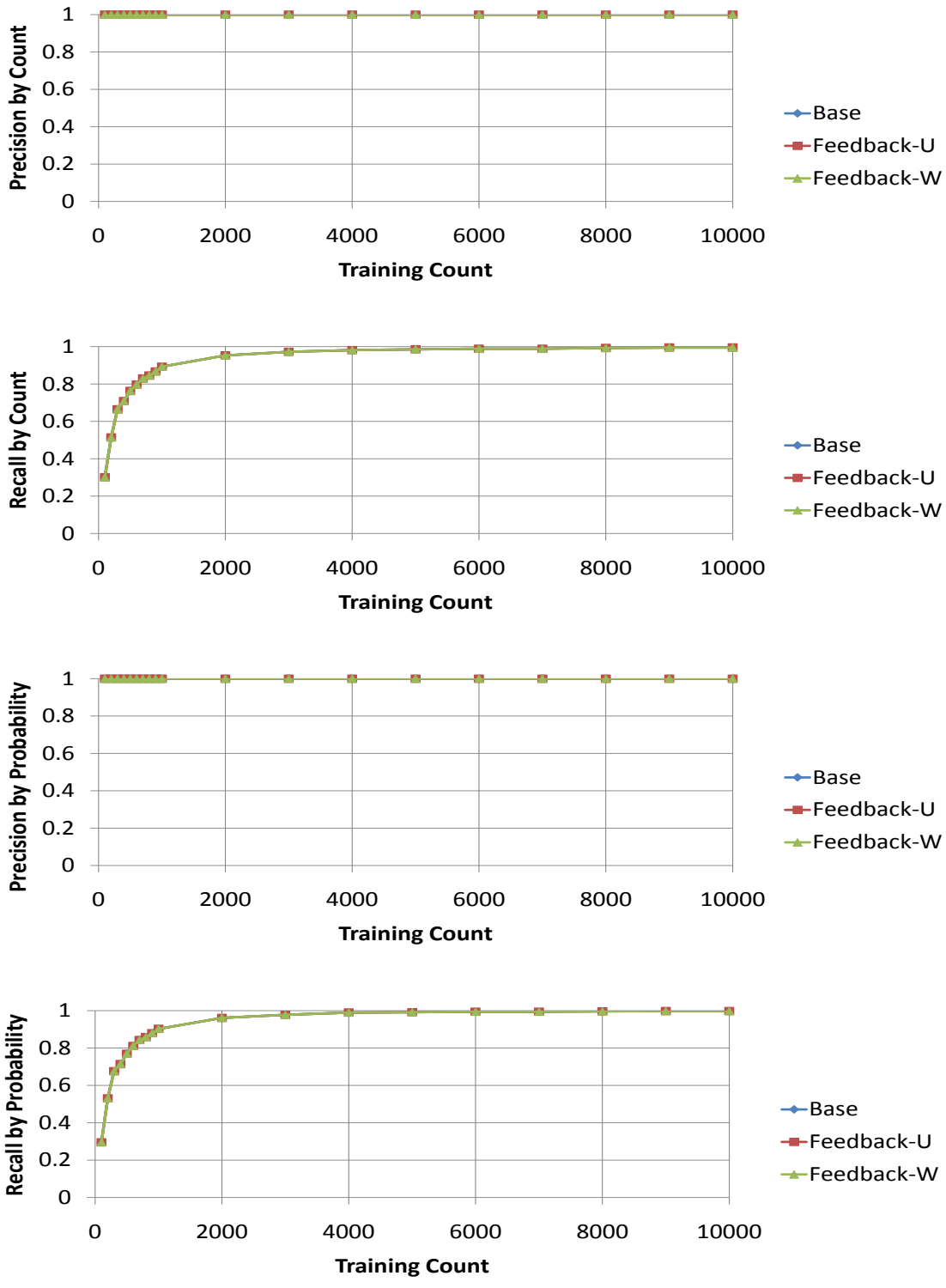

Figure A.17: False Input - No Bad Data (Table 7.1)

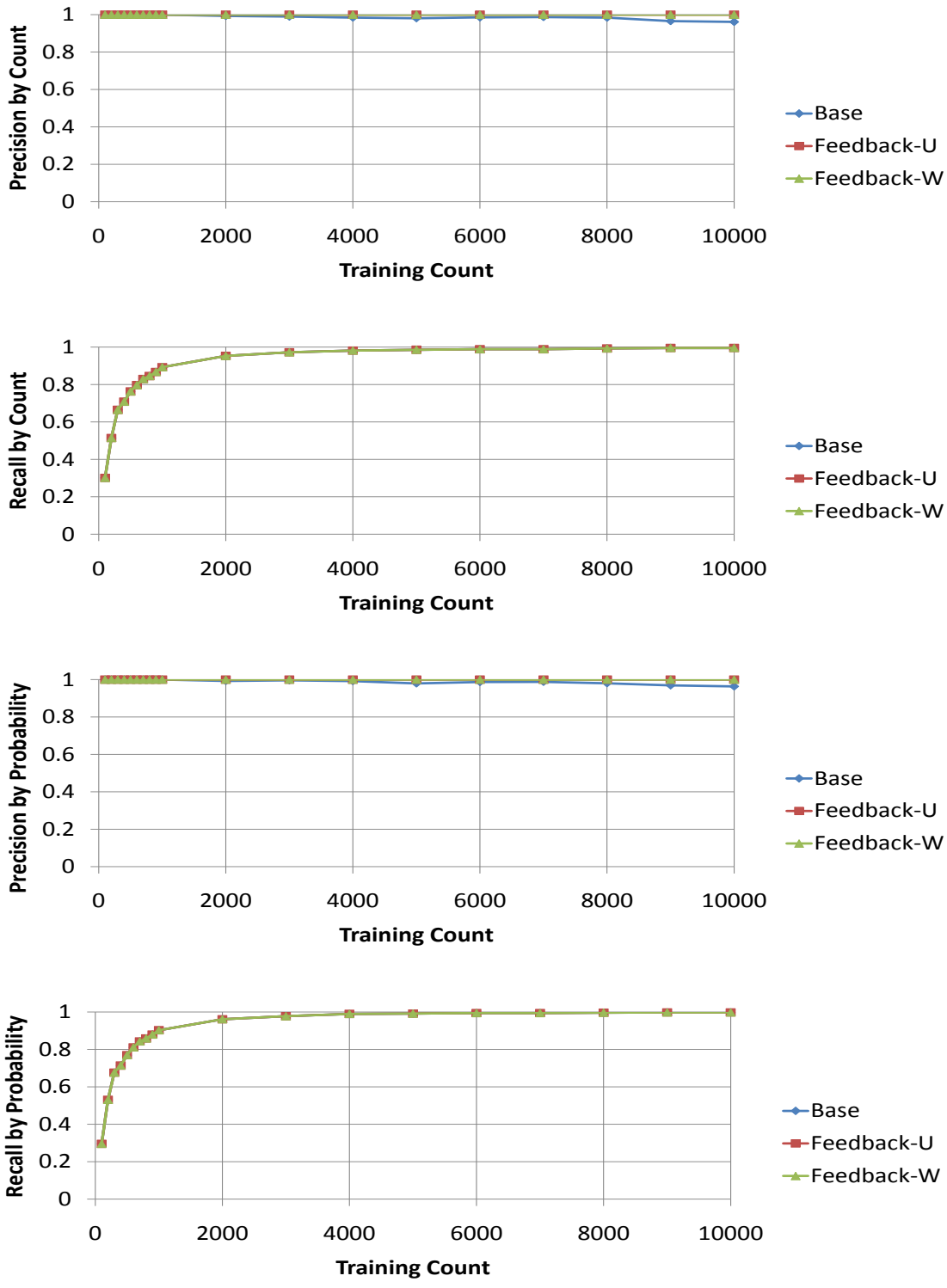

Figure A.18: False Input - 0.001 Bad Data (Table 7.1)

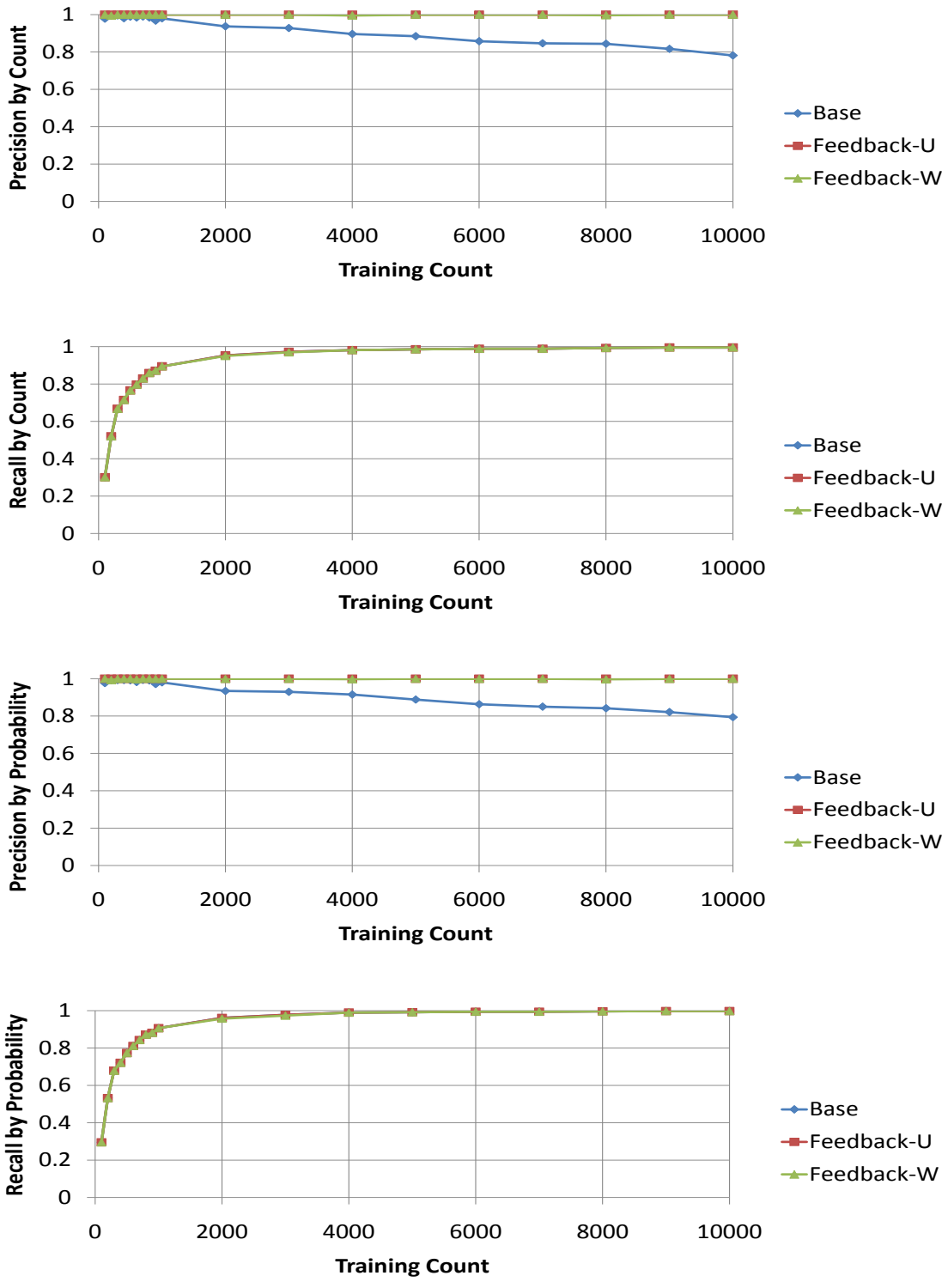

Figure A.19: False Input - 0.01 Bad Data (Table 7.1)

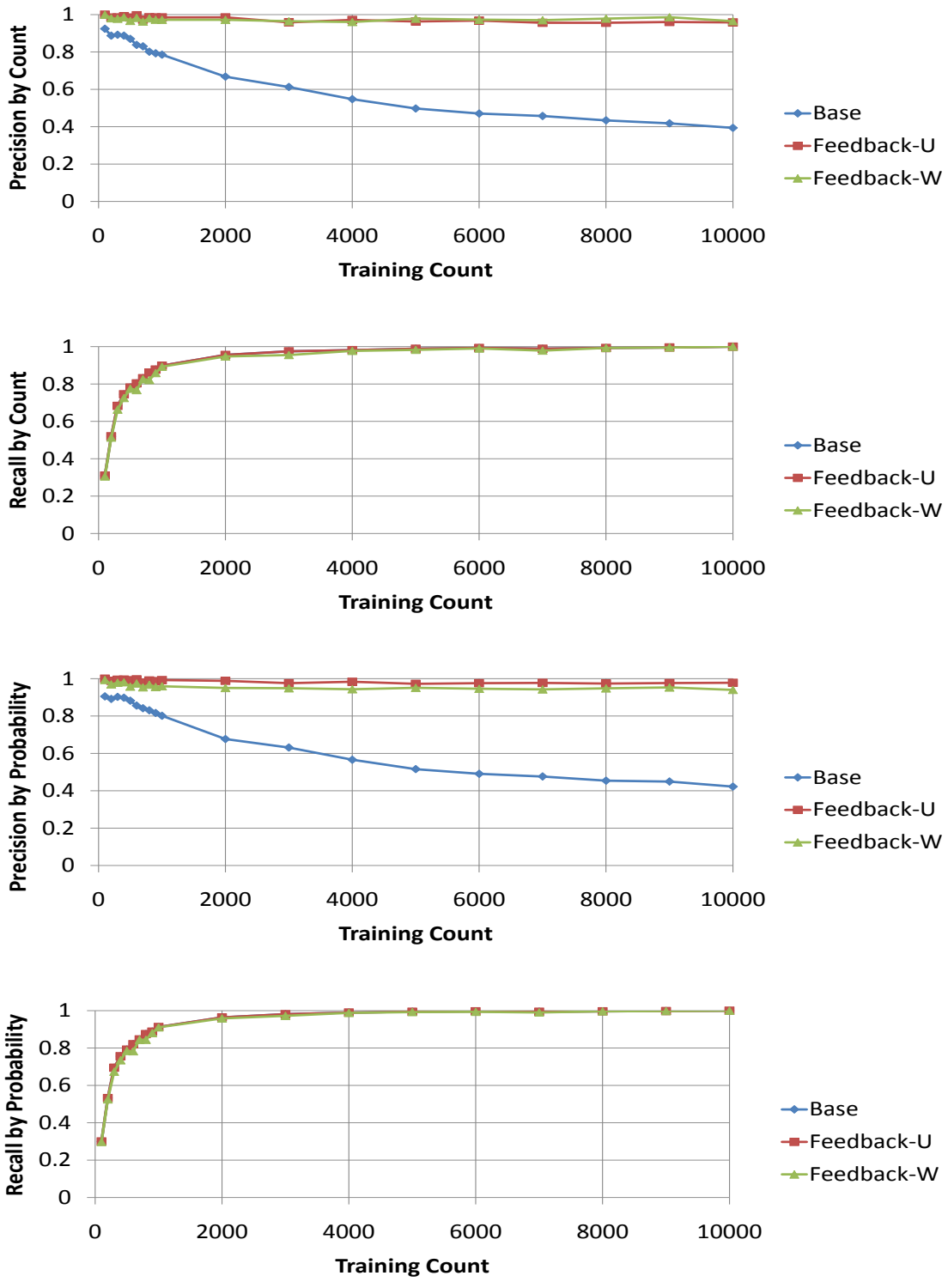

Figure A.20: False Input - 0.1 Bad Data (Table 7.1)

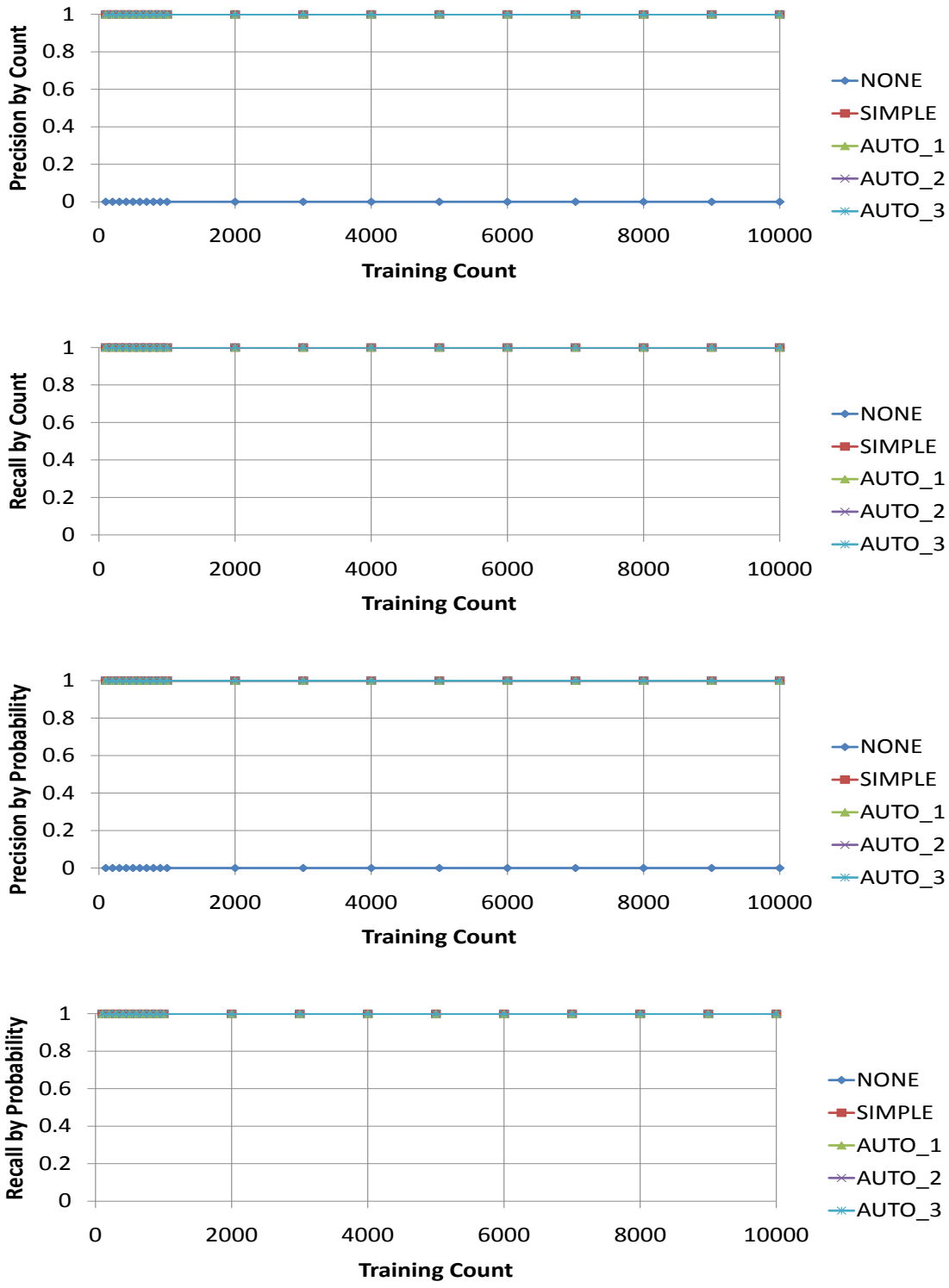

Figure A.21: Time 0 Tests - Night (Table 8.4)

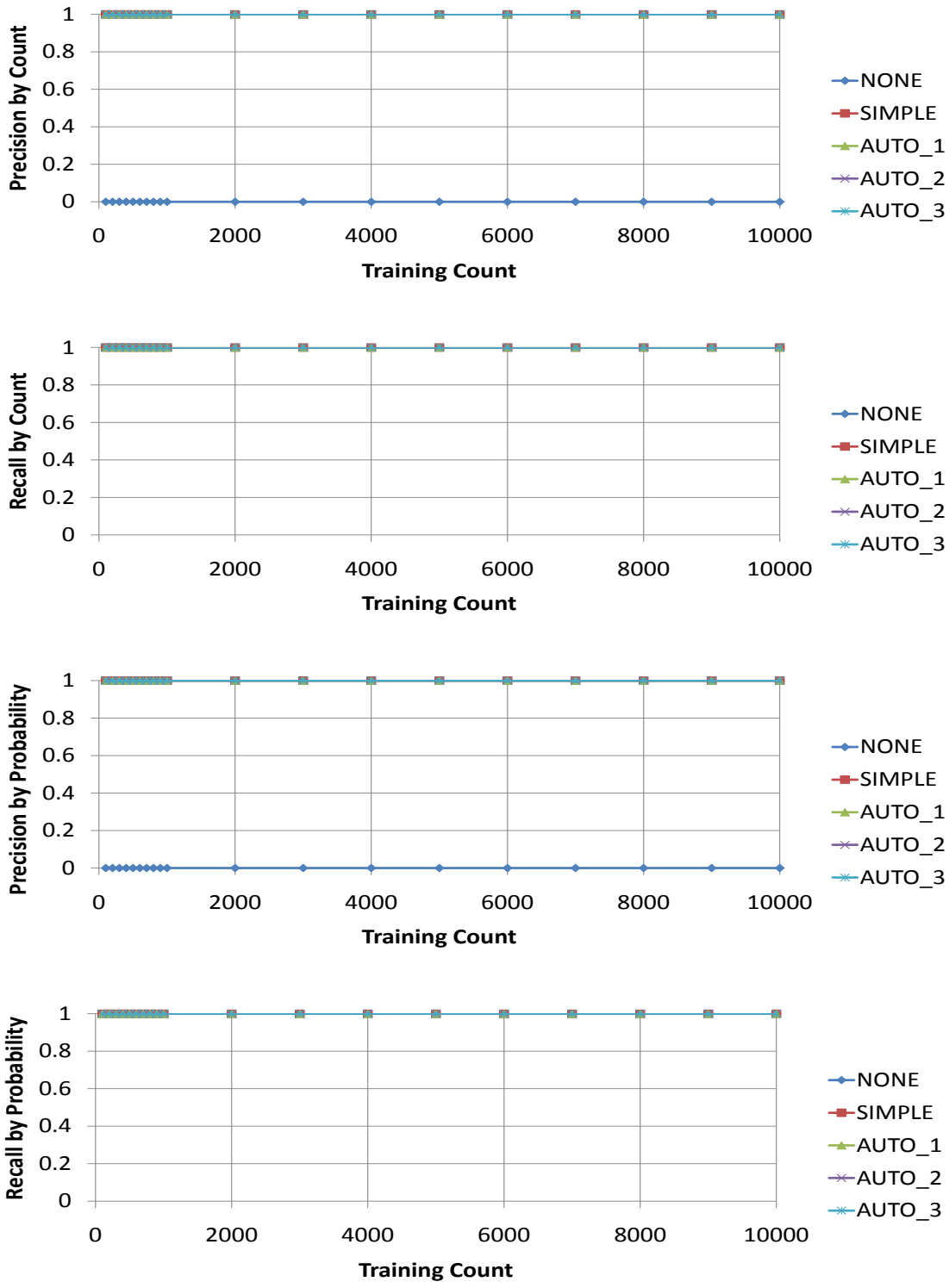

Figure A.22: Time 0 Tests - Early Morning (Table 8.4)

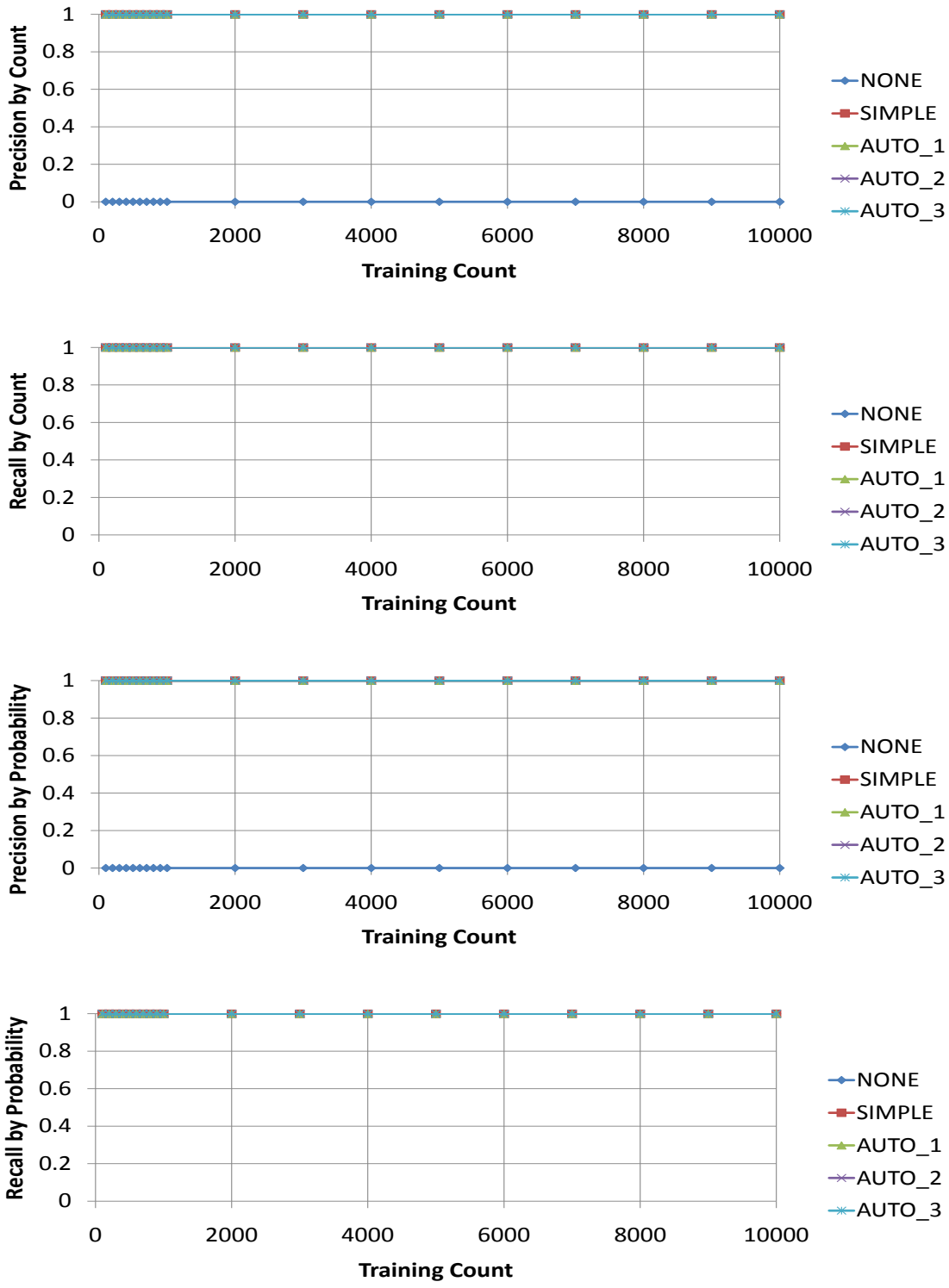

Figure A.23: Time 0 Tests - Morning (Table 8.4)

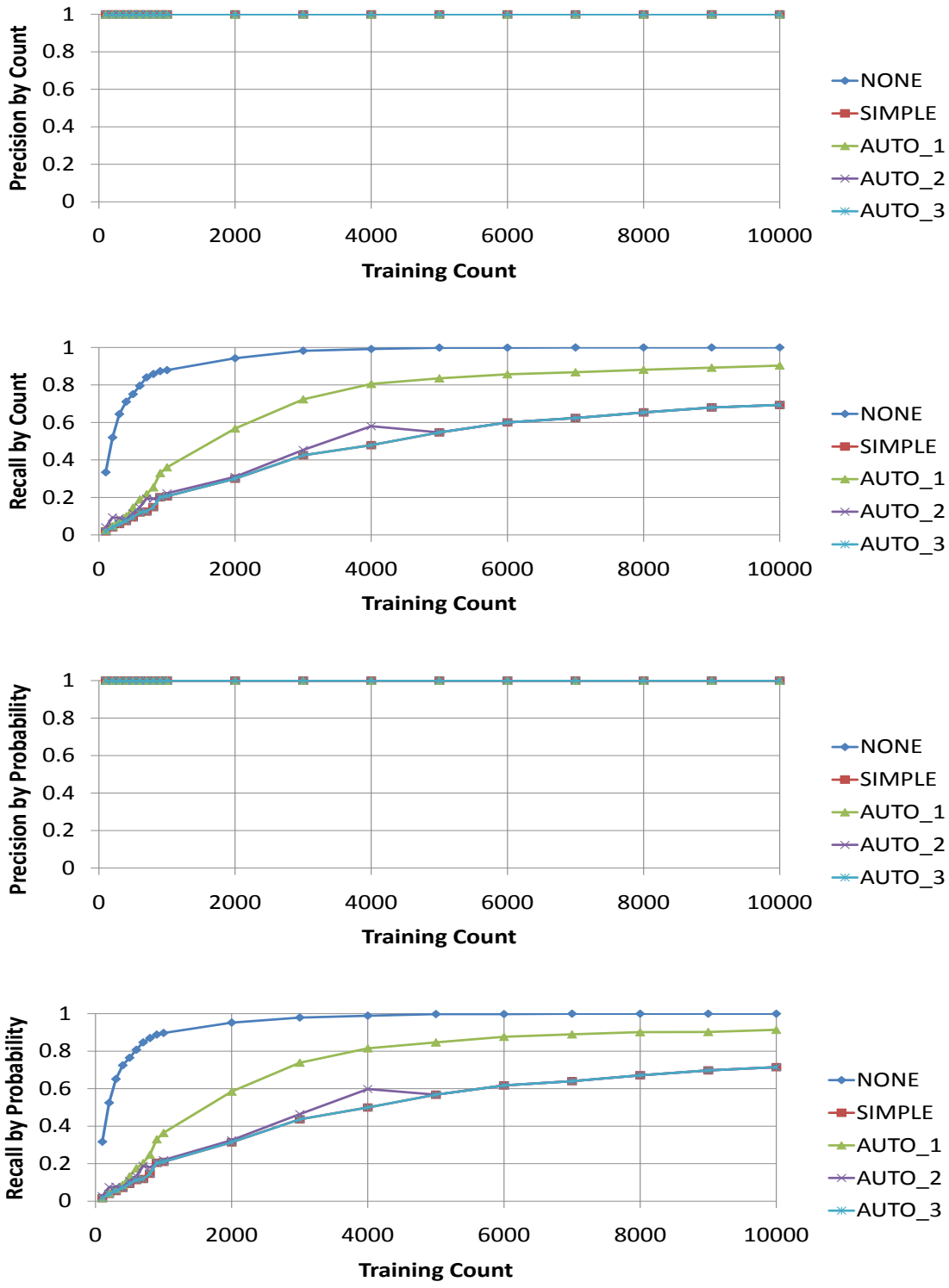

Figure A.24: Time 0 Tests - Day (Table 8.4)

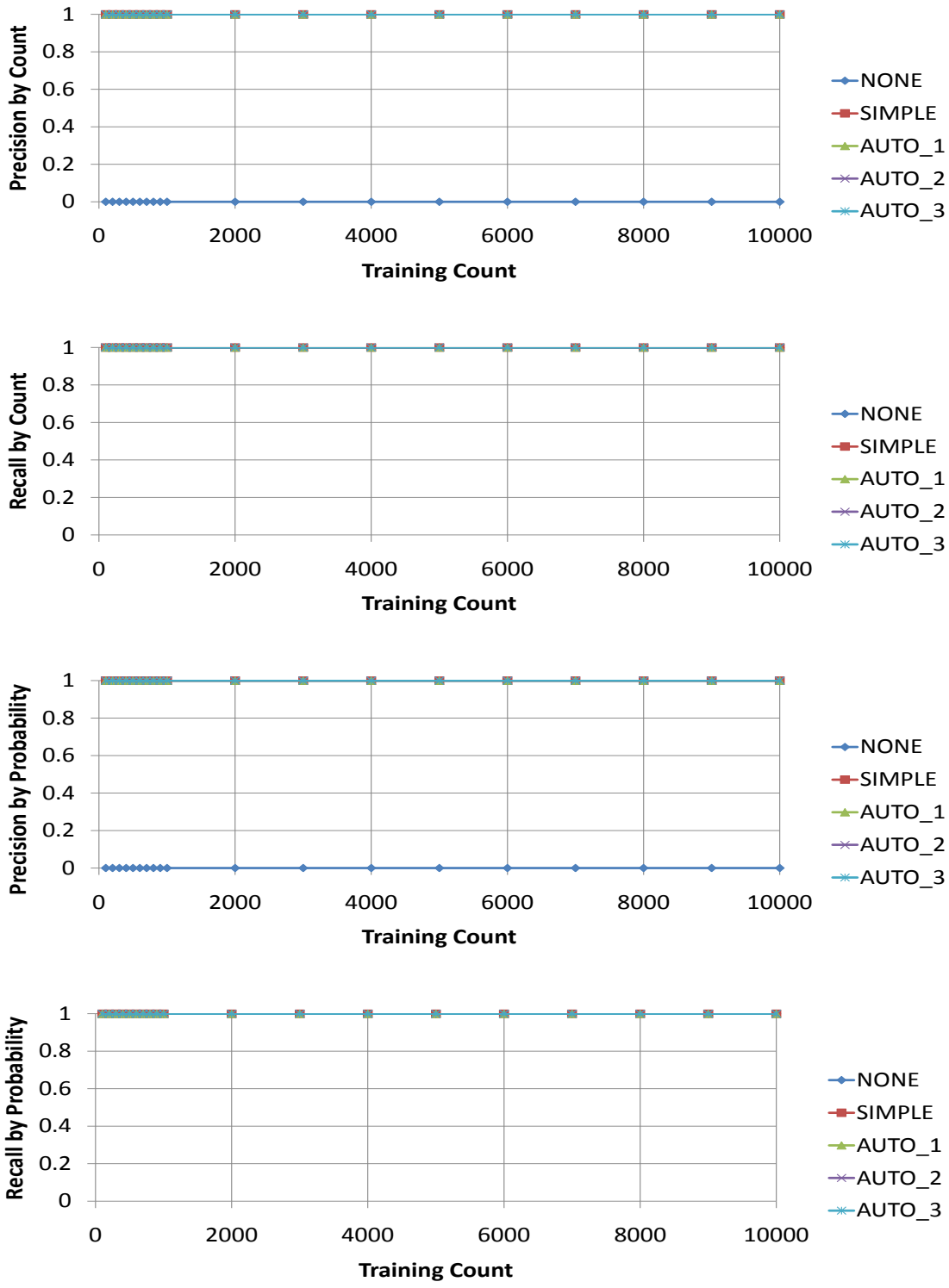

Figure A.25: Time 1 Tests - Night (Table 8.5)

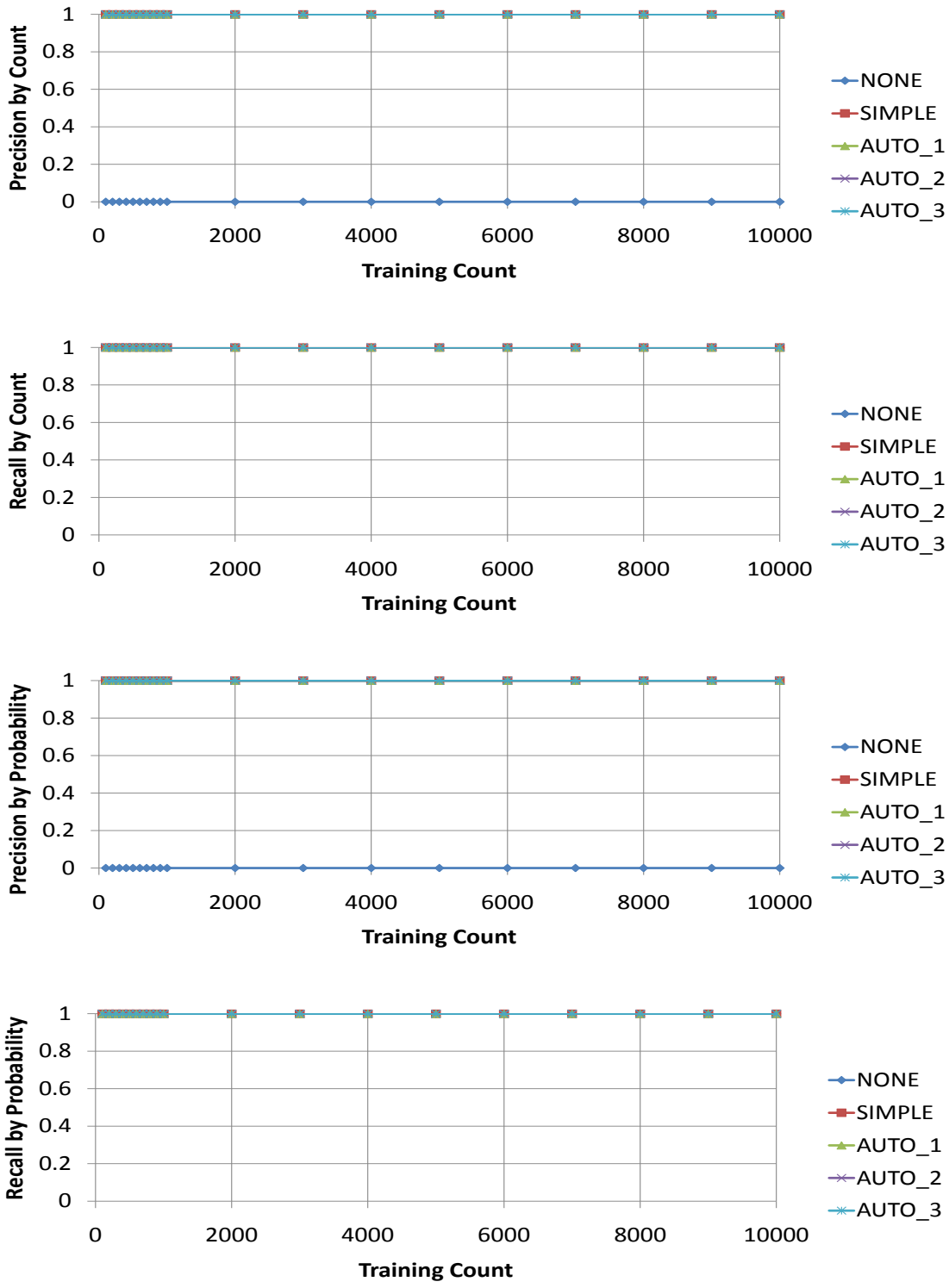

Figure A.26: Time 1 Tests - Early Morning (Table 8.5)

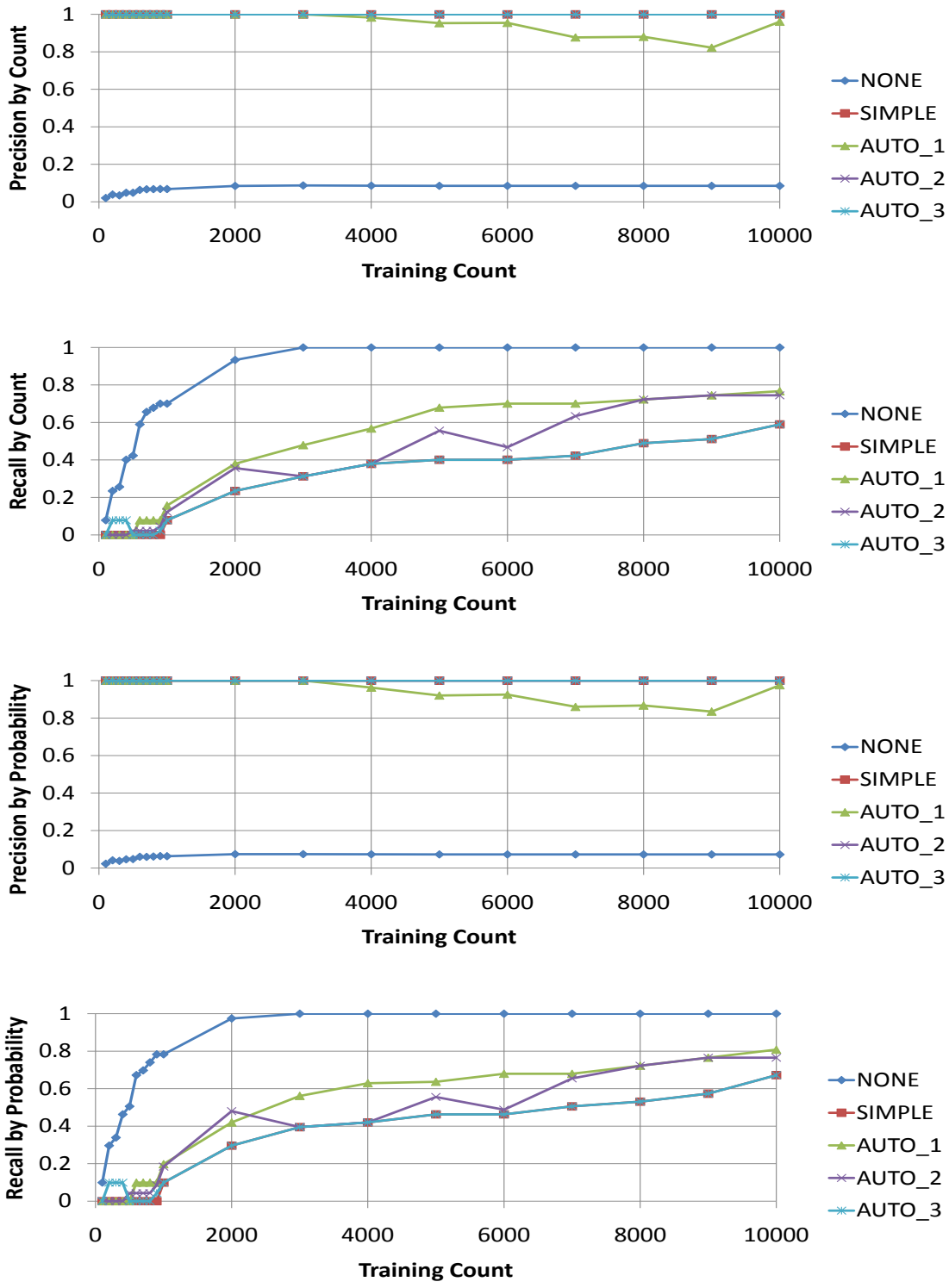

Figure A.27: Time 1 Tests - Morning (Table 8.5)

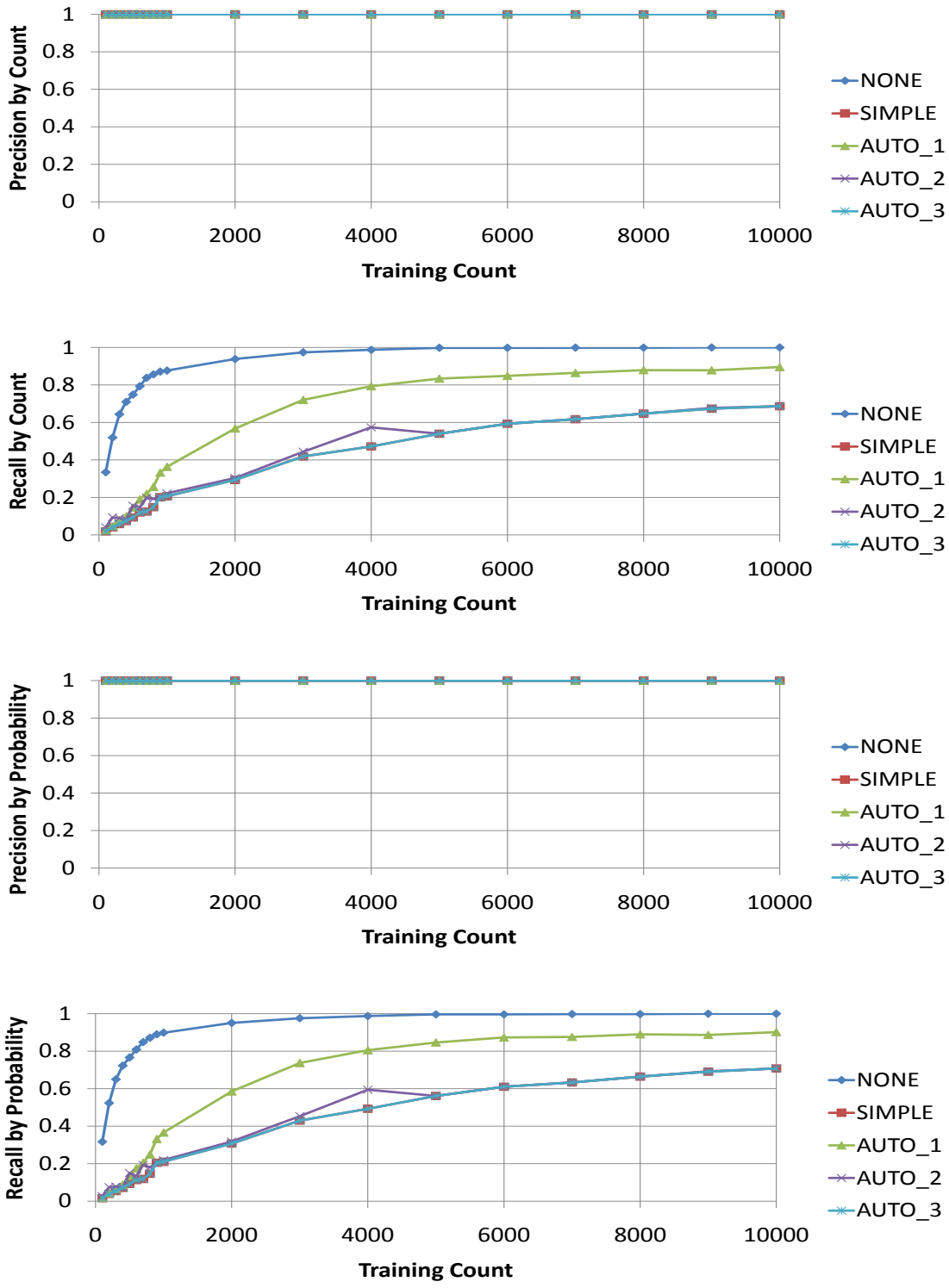

Figure A.28: Time 1 Tests - Day (Table 8.5)

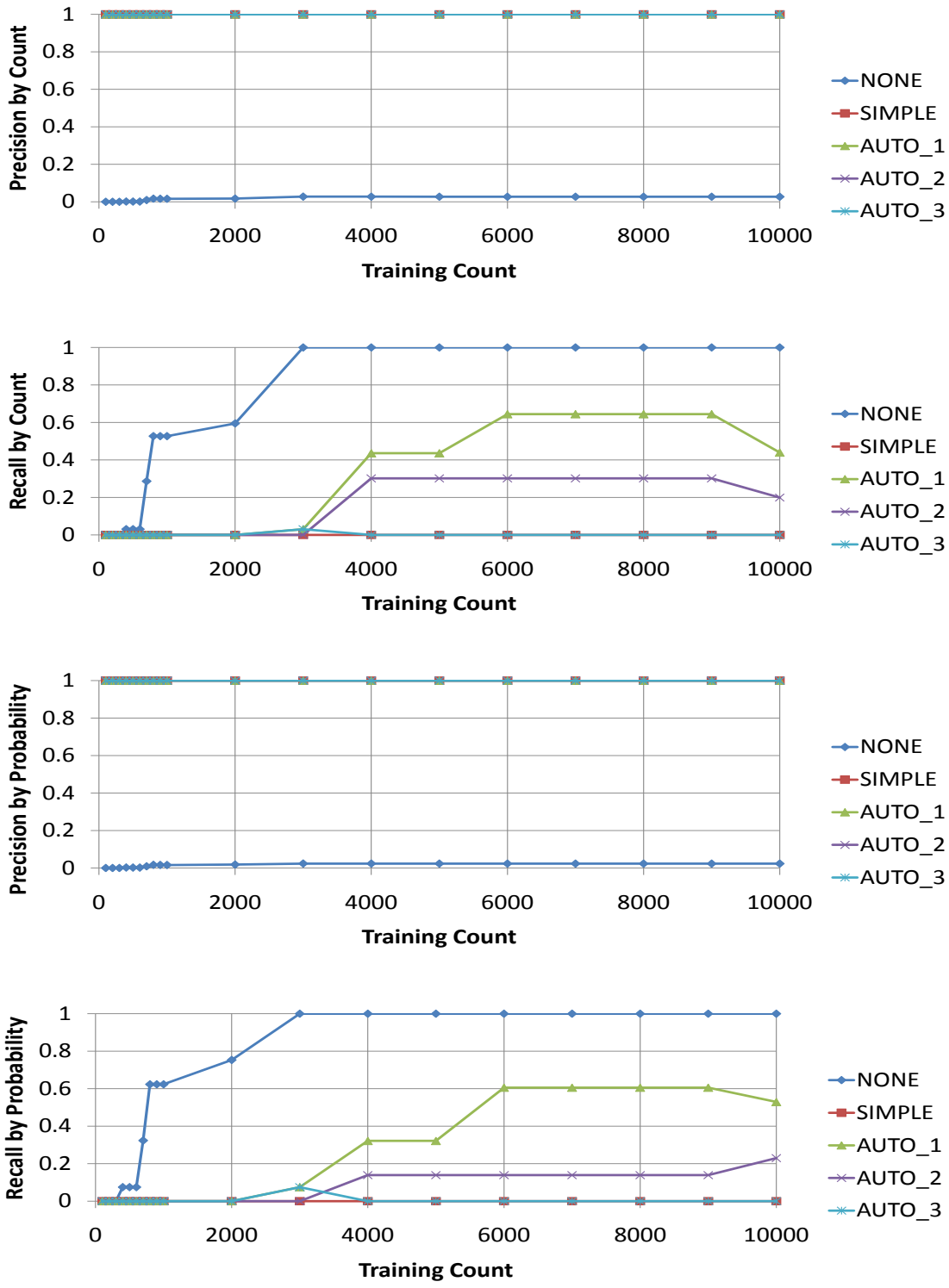

Figure A.29: Time 2 Tests - Night (Table 8.6)

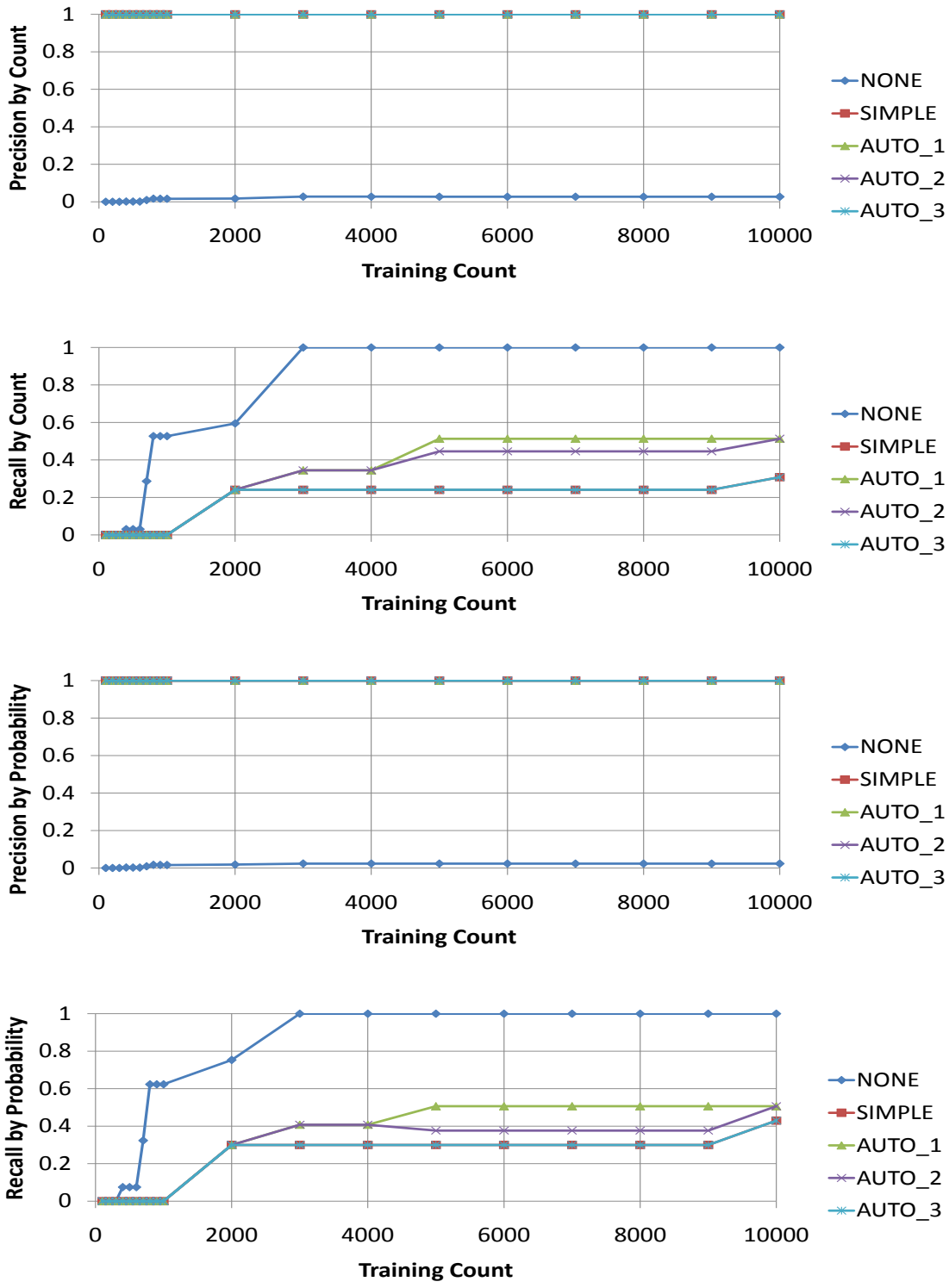

Figure A.30: Time 2 Tests - Early Morning (Table 8.6)

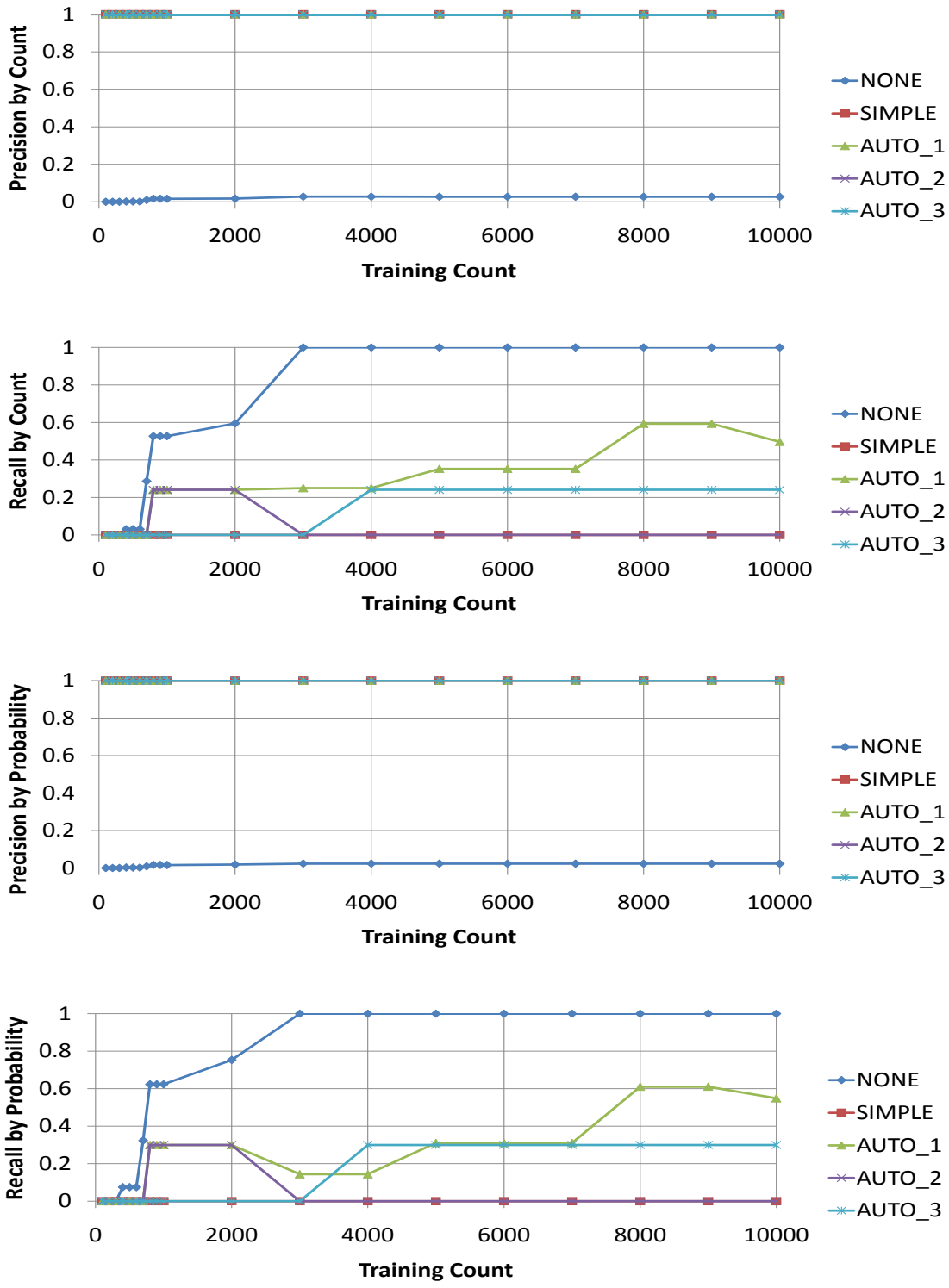

Figure A.31: Time 2 Tests - Morning (Table 8.6)

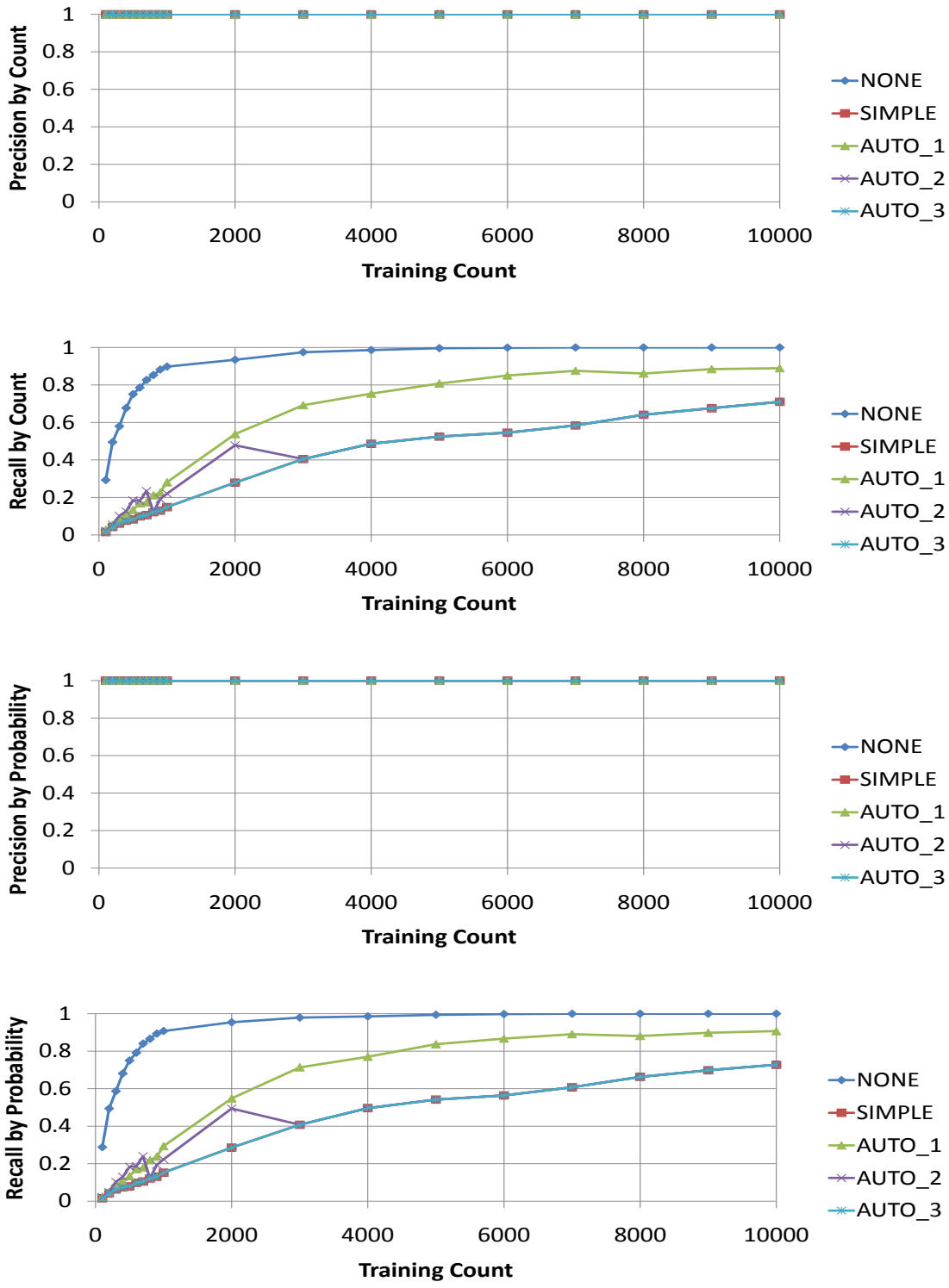

Figure A.32: Time 2 Tests - Day (Table 8.6)

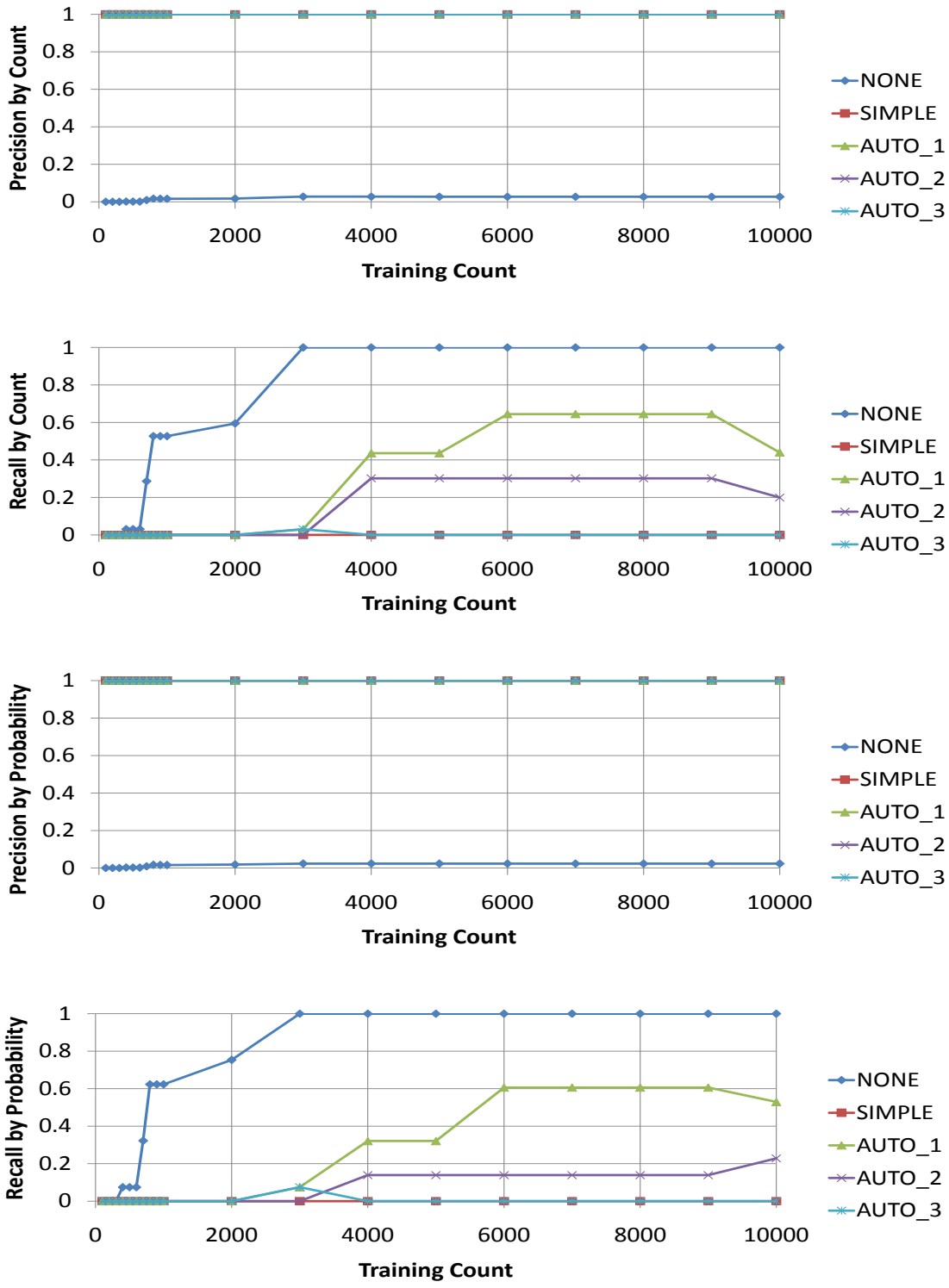

Figure A.33: Time 3 Tests - Night (Table 8.7)

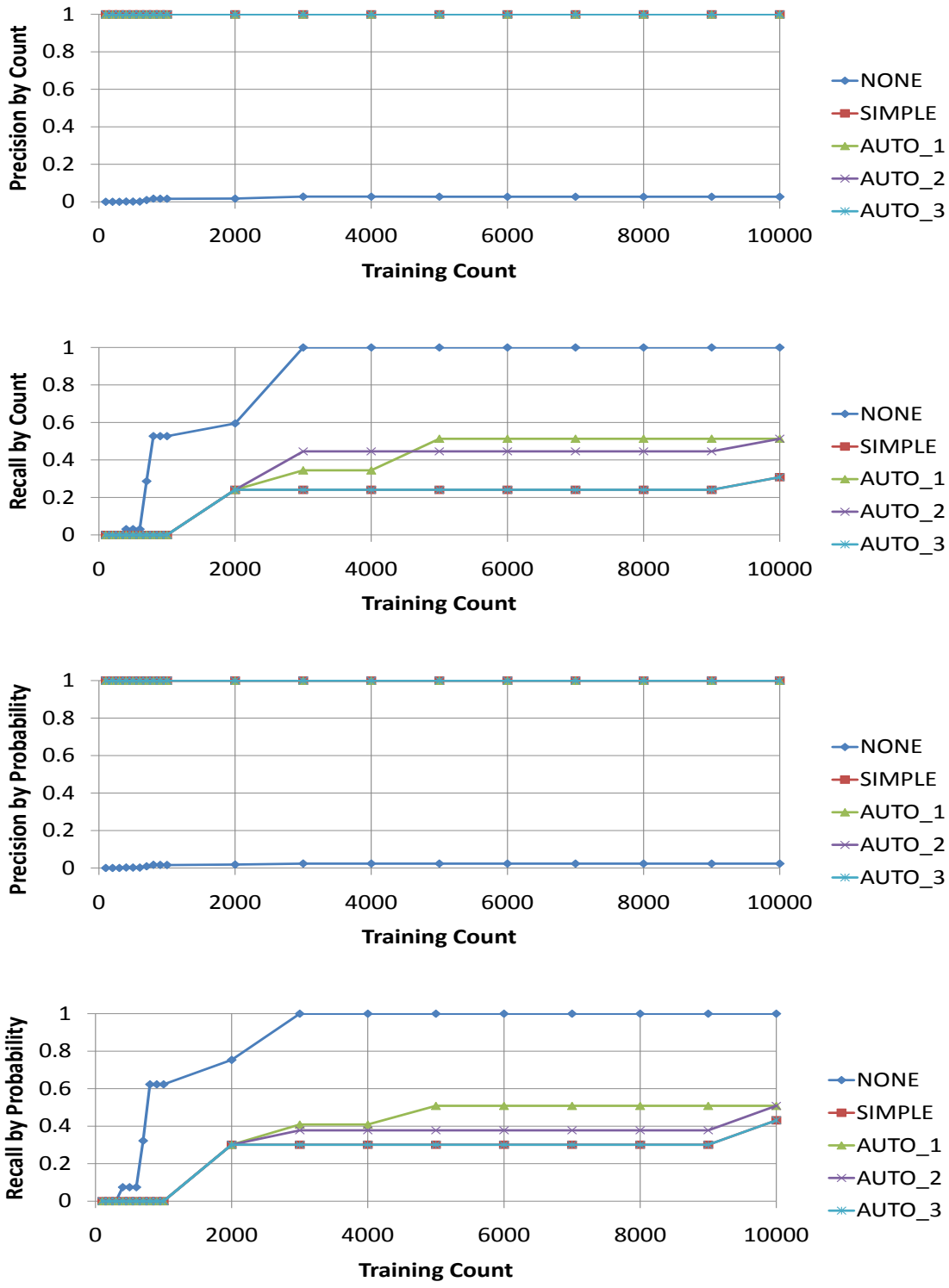

Figure A.34: Time 3 Tests - Early Morning (Table 8.7)

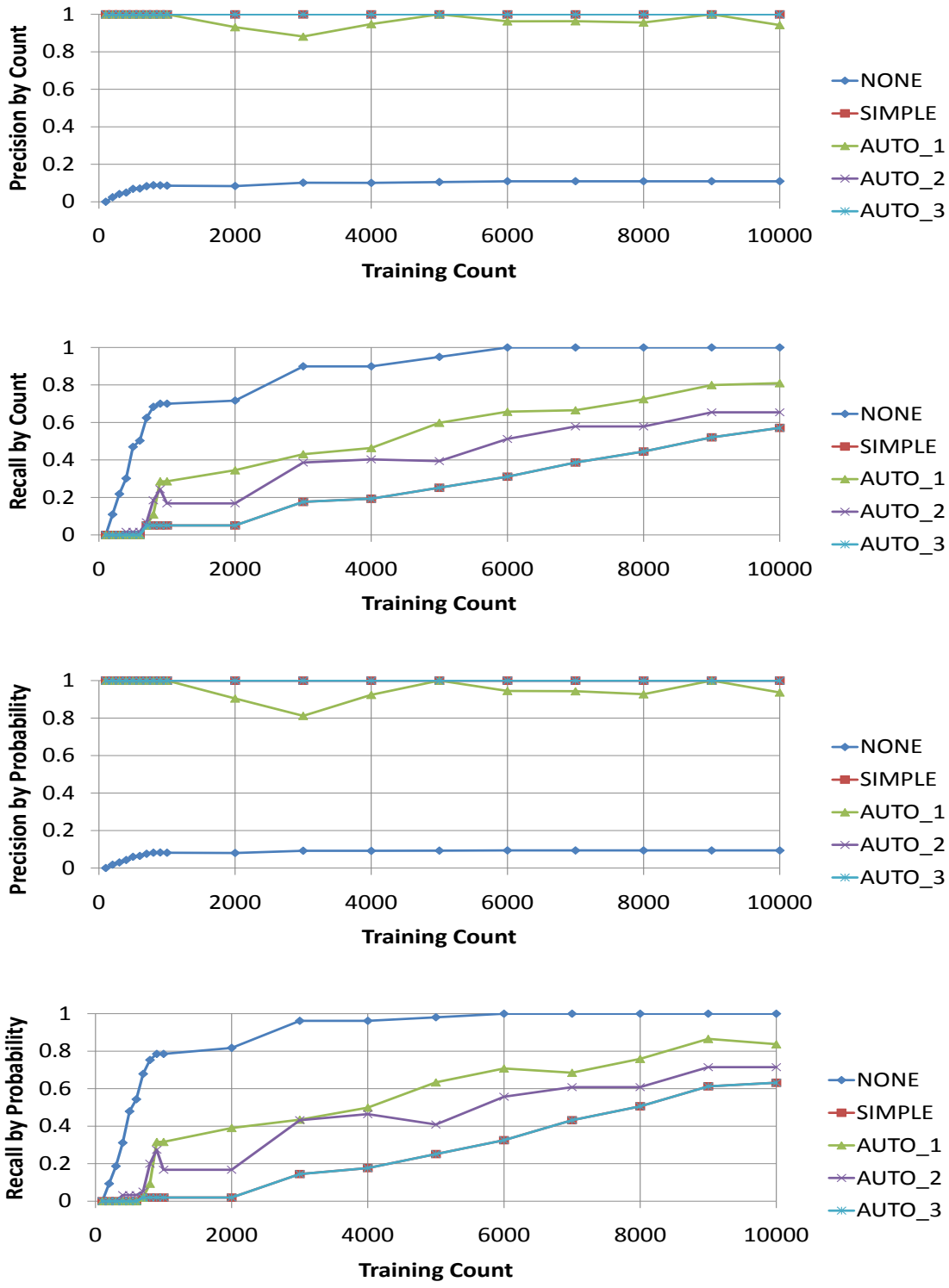

Figure A.35: Time 3 Tests - Morning (Table 8.7)

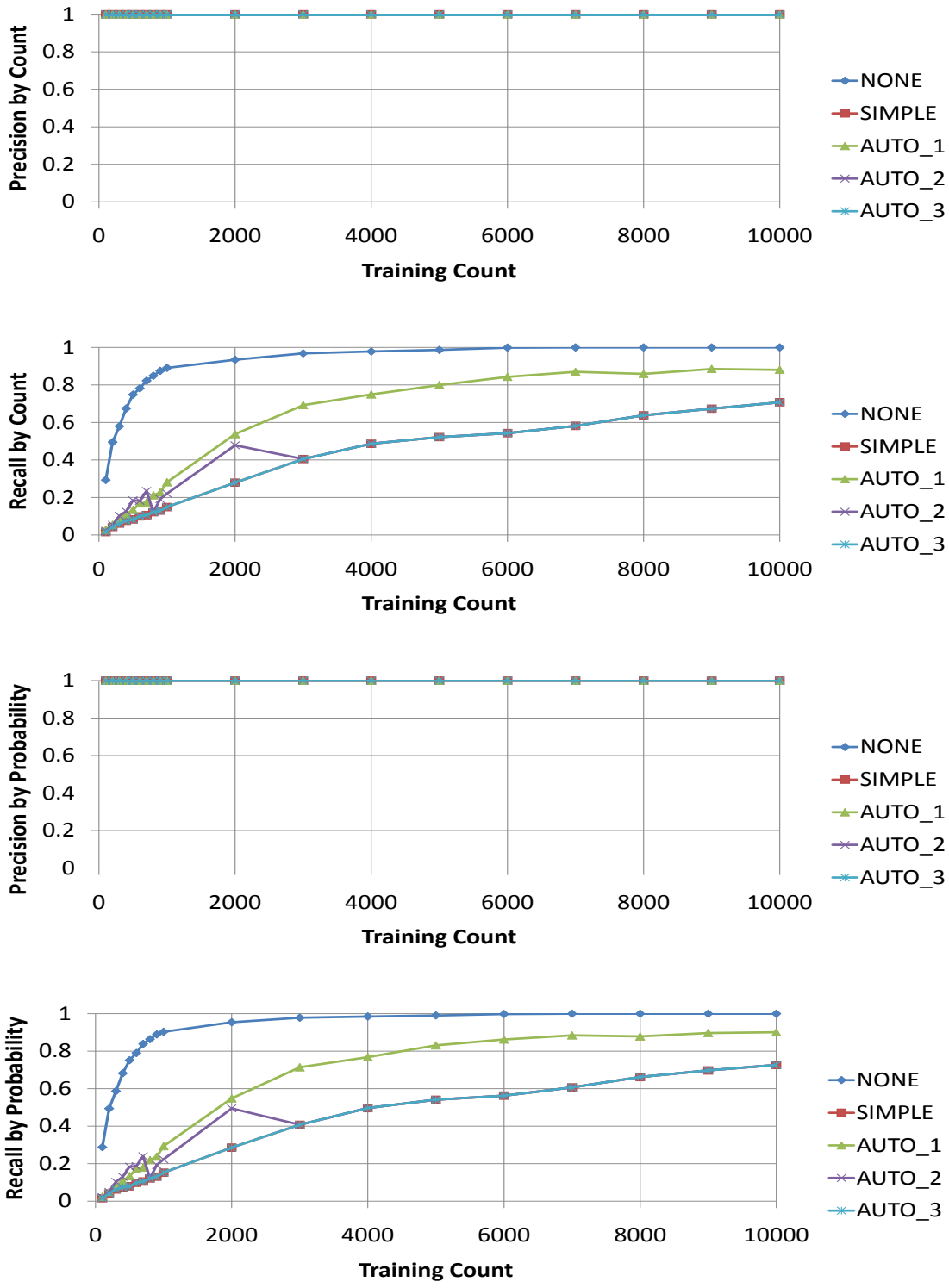

Figure A.36: Time 3 Tests - Day (Table 8.7)

Bibliography

[AP76] H. Andrews and III Patterson, C. Singular value decomposition (svd) image coding. *Communications, IEEE Transactions on*, 24(4):425–432, Apr 1976.

[Aus09] D. Austin. We recommend a singular value decomposition. American Mathematical Society Feature Column, http://www.ams.org/featurecolumn/archive/svd.html, August 2009.

[Bra08] Roger B. Bradford. An empirical study of required dimensionality for large-scale latent semantic indexing applications. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 153–162, New York, NY, USA, 2008. ACM.

[Chu] Moody T. Chu. On the statistical meaning of truncated singular value decomposition.

[DA00] Anind K. Dey and Gregory D. Abowd. Cybreminder: A context-aware system for supporting reminders. In *HUC*, pages 172–186, 2000.

[DDL+90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[FKT06] Samantha Farrimond, Robert G. Knight, and Nick Titov. The effects of aging on remembering intentions:performance on a simulated shopping task. In *Applied Cognitive Psychology*, volume 20, pages 533Ű–555, April 2006.

[GFKT01] Lise Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of relational structure. In *Proc. 18th International Conf. on Machine Learning*, pages 170–177. Morgan Kaufmann, San Francisco, CA, 2001.

[KC94] H. Kanai and N. Chubachi. Accurate estimation of ar model by tapered svd without rank determination. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 4:473–476, 1994.

[KIC94] H. Kanai, K. Ikikame, and N. Chubachi. A tapered svd without rank determination for estimation of multipulse input time series from noisy output. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 4:45–48, 1994.

[Lin00] Mei-Hui Lin. Out-of-core singular value decomposition. Technical Report TR-83, State University of New York, Stony Brook - Experimental Computer Systems Lab, may 2000.

[MHH+02] Jennifer Mankoff, Gary Hsieh, Ho Chak Hung, Sharon Lee, and Elizabeth Nitao. Using low-cost sensing to support nutritional awareness. In *Proceedings of the 4th international conference on Ubiquitous Computing*, pages 371–376. Springer-Verlag, 2002.

[NJ03] Nathan Srebro Nati and Tommi Jaakkola. Weighted low-rank approximations. In *In 20th International Conference on Machine Learning*, pages 720–727. AAAI Press, 2003.

[RC00] Peter G. Rendell and Fergus I. M. Craik. Virtual week and actual week: Age-related differences in prospective memory. In *Applied Cognitive Psychology*, volume 14, pages S43–S62, 2000.

[SG07] Patricia Shanahan and William G. Griswold. Inferring the everyday task capabilities of locations. In Jeffrey Hightower, Bernt Schiele, and Thomas Strang, editors, *LoCA*, volume 4718 of *Lecture Notes in Computer Science*, pages 157–174. Springer, 2007.

[SKK05] Hariharan Sankaran, Srinivas Katkoori, and Umadevi Kailasam. System level energy optimization for location aware computing. *Pervasive Computing and Communications, IEEE International Conference on*, 0:319–323, 2005.

[SKKR00] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system - a case study. In *In ACM WebKDD Workshop*, 2000.

[SLL+05] Timothy Sohn, Kevin A. Li, Gunny Lee, Ian E. Smith, James Scott, and William G. Griswold. Place-its: A study of location-based reminders on mobile phones. In *Ubicomp*, pages 232–250, 2005.

[Str05] G. Strang. *Linear Algebra and Its Applications*. Brooks Cole, 2005.

[TS07] Jane E. Tougas and Raymond J. Spiteri. Updating the partial singular value decomposition in latent semantic indexing. *Computational Statistics and Data Analysis*, 52(1):174 – 183, 2007.

[TS08] Jane E. Tougas and Raymond J. Spiteri. Two uses for updating the partial singular value decomposition in latent semantic indexing. *Appl. Numer. Math.*, 58(4):499–510, 2008.

[W3C] W3c semantic web frequently asked questions. http://www.w3.org/2001/sw/SW-FAQ.

[YMKM00] J. Youll, J. Morris, R. Krikorian, and P. Maes. Impulse: Location-based agent assistance, 2000.

[You01] Jim Youll. 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*, pages 109–110. ACM Press, 2001.

[ZS99] Hongyuan Zha and Horst D. Simon. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing*, 21(2):782–791, 1999.