

Predicting Future Locations and Arrival Times of Individuals

Ingrid E. Burbey

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in
partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In
Computer Engineering

Thomas L. Martin, Chair
Mark T. Jones
Scott F. Midkiff
Manuel A. Perez-Quinones
Joseph G. Tront

26 April, 2011
Blacksburg, Virginia

Keywords: Context Awareness, Location Awareness, Context Prediction,
Location Prediction, Time-of-Arrival Prediction

Predicting Future Locations and Arrival Times of Individuals

Ingrid E. Burbey

ABSTRACT

This work has two objectives: a) to predict people's future locations, and b) to predict when they will be at given locations. Current location-based applications react to the user's current location. The progression from location-awareness to location-prediction can enable the next generation of proactive, context-predicting applications.

Existing location-prediction algorithms predict someone's *next* location. In contrast, this dissertation predicts someone's *future* locations. Existing algorithms use a sequence of locations and predict the next location in the sequence. This dissertation incorporates temporal information as timestamps in order to predict someone's location at any time in the future. Sequence predictors based on Markov models have been shown to be effective predictors of someone's *next* location. This dissertation applies a Markov model to two-dimensional, timestamped location information to predict *future* locations.

This dissertation also predicts *when* someone will be at a given location. These predictions can support presence or understanding co-workers' routines. Predicting the times that someone is going to be at a given location is a very different and more difficult problem than predicting where someone will be at a given time. A location-prediction application may predict one or two key locations for a given time, while there could be hundreds of correct predictions for times of the day that someone will be in a given location. The approach used in this dissertation, a heuristic model loosely based on Market Basket Analysis, is the first to predict when someone will arrive at any given location.

The models are applied to sparse, WiFi mobility data collected on PDAs given to 275 college freshmen. The location-prediction model predicts future locations with 78-91% accuracy. The temporal-prediction model achieves 33-39% accuracy. If a tolerance of plus/minus twenty minutes is allowed, the prediction rates rise to 77%-91%.

This dissertation shows the characteristics of the timestamped, location data which lead to the highest number of correct predictions. The best data cover large portions of the day, with less than three locations for any given timestamp.

Dedication

This dissertation is dedicated to my family.

To my parents, for sacrificing for my education and demonstrating perseverance,

To my husband, for standing with me through my many (many!) ups-and-downs of the doctoral process, and

To my children, who continue to delight and inspire me with their creativity, their insight and their joy.

Grant Information

Portions of this research were supported by the U.S. National Science Foundation under Grant DGE-9987586. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

CHAPTER 1 INTRODUCTION.....	1
1.1 CONTRIBUTIONS	2
1.2 SUMMARY	3
CHAPTER 2 BACKGROUND	5
2.1 MOTIVATION.....	5
2.1.1 <i>Single-User Applications for Prediction</i>	6
2.1.2 <i>Privacy</i>	8
2.1.3 <i>Predicting Time</i>	9
2.2 OVERVIEW OF PREDICTION TECHNIQUES	11
2.2.1 <i>Supervised Learning</i>	12
2.2.2 <i>Unsupervised Learning</i>	14
2.2.3 <i>Number of Outputs</i>	14
2.3 TEMPORAL AND SPATIO-TEMPORAL DATA MINING.....	15
2.3.1 <i>Taxonomies of Temporal Data Mining</i>	17
2.3.2 <i>Enhancing a Sequence with Temporal Information</i>	19
2.4 LOCATION DETERMINATION	20
2.4.1 <i>Symbolic Location</i>	21
2.5 RELATED WORK IN LOCATION- AND TEMPORAL-PREDICTION.....	22
2.5.1 <i>MavHome</i>	22
2.5.2 <i>Using GPS to Determine Significant Locations</i>	24
2.5.3 <i>Dartmouth College Mobility Predictions</i>	25
2.5.4 <i>Smart Office Buildings</i>	26
2.5.5 <i>Reality Mining at MIT</i>	27
2.5.6 <i>Other Related Work in Location-Prediction</i>	27
2.5.7 <i>Summary</i>	30
2.6 REPRESENTATION	31
2.6.1 <i>Representing Music</i>	31
2.7 THE PREDICTION BY PARTIAL MATCH ALGORITHM.....	32
2.8 SUMMARY	34
CHAPTER 3 PREDICTING FUTURE LOCATIONS.....	35
3.1 LOCATION-PREDICTION PROBLEM STATEMENT	35
3.2 THE DATA—FRESHMEN AT UCSD	36
3.2.1 <i>Preprocessing</i>	40
3.2.2 <i>The MoveLoc (1-minute) Dataset</i>	41
3.2.3 <i>The SigLoc (10-minute) Dataset</i>	42
3.3 THE EXPERIMENT	44
3.3.1 <i>Determining the Amount of Training Data</i>	44
3.3.2 <i>Training and Testing the Model</i>	46
3.3.3 <i>Results</i>	49
3.3.4 <i>Entropy of Movement Patterns</i>	54
3.3.5 <i>Analysis of Predictable Data</i>	57
3.3.6 <i>Drawbacks of the Markov Model for Predicting Location</i>	67
3.4 SUMMARY	68

CHAPTER 4 PREDICTING ARRIVAL TIMES	70
4.1 TEMPORAL-PREDICTION USING THE MARKOV MODEL	70
4.2 THE “TRAVERSING THE SEQUENCE” MODEL	76
4.3 THE TRAVERSING-THE-SEQUENCE ALGORITHM	77
4.3.1 <i>An Example of the SEQ Model</i>	82
4.3.2 <i>Advantages and Disadvantages of the SEQ Model</i>	85
4.4 EXPERIMENT AND RESULTS	89
4.5 SCORING INDIVIDUAL PREDICTIONS	93
4.6 SUMMARY	98
CHAPTER 5 CONCLUSIONS AND FUTURE WORK	99
5.1 SPARSE DATA	99
5.2 CONCLUSIONS	100
5.3 CONTRIBUTIONS	101
5.4 FUTURE WORK	103
5.4.1 <i>Requirements for Prediction Applications</i>	103
5.4.2 <i>Measuring Changes in Predictability</i>	105
5.5 SUMMARY	105
APPENDIX A INVESTIGATING THE PING-PONG EFFECT	106
APPENDIX B THRESHOLDS APPLIED TO THE MARKOV MODEL FOR PREDICTING TIME	110
APPENDIX C ADDITIONAL GRAPHS OF SEQ MODEL RESULTS	114
SIGLOC (10-MINUTE) DATA	114
MOVELOC (1-MINUTE) DATA	116
APPENDIX D SOURCE CODE FOR THE MARKOV MODEL	118
PREDICT.C	120
MODEL-2.C	140
STRING16.C	159
CODER.H	161
MAPPING.H (EXAMPLE)	162
MODEL.H	173
PREDICT.H	176
STRING16.H	177
APPENDIX E UTILITIES	178
001PBOX.DAT (USED WITH 16-BIT MARKOV MODEL)	178
USER1.CSV	178
USERB1.CSV	178
DUR1.CSV	179
USERC1.CSV	179
BUILD_16BIT_BOXCOMBOS5.PY	179
COUNT_DATA_STATS.PY	185
ENTROPY.PY	187
FILE_STATS_XML.PY	192
PARSE_INPUT_LINE.PY	195

TEST_STATS_XML.PY	196
APPENDIX F SEQ MODEL SOURCE CODE	199
SEQ_MODEL.PY	199
REFERENCES.....	212

List of Figures

Figure 2-1 Overview of machine learning	12
Figure 2-2 Outputs of machine learning algorithms	16
Figure 2-3 Types of temporal data mining problems.....	18
Figure 2-4 A graph model of a smart home floor plan	23
Figure 2-5 Ashbrook & Starner's technique	25
Figure 2-6 PPM-C Model after training on the string abracadabra.	33
Figure 3-1 A portion of the UCSD dataset	37
Figure 3-2 The largest user data file (User 016)	39
Figure 3-3 The smallest user data file.....	39
Figure 3-4 Cumulative distribution of the number of unique locations.....	42
Figure 3-5 MoveLoc data (1-minute or less) for User 003	43
Figure 3-6 SigLoc data (10-minute) for User 003	43
Figure 3-7 Average log-loss for different training and testing schemes.....	46
Figure 3-8 Portion of the Markov model trained for location-prediction (User 003, weeks 1-5)	47
Figure 3-9 Example output from the Markov model location predictor.....	48
Figure 3-10 Aggregate results for predicting future locations using the Markov model.....	49
Figure 3-11 Normalized distribution of Markov model predictions for the 1-minute dataset.....	51
Figure 3-12 Normalized distribution of Markov model predictions for the 10-minute dataset....	51
Figure 3-13 Comparison of different prediction schemes on the MoveLoc 1-minute data.....	52
Figure 3-14 Comparison of different prediction schemes on the SigLoc 10-minute data.....	53
Figure 3-15 Distribution of the entropy S , the random entropy S^{rand} , and the uncorrelated entropy S^{unc} across 45,000 users	56
Figure 3-16 Frequency distribution of conditional entropy, uncoorelated entropy and random entropy as a function of the number of unique locations.	57
Figure 3-17 Normalized distribution of the number of unique locations	60
Figure 3-18 Normalized distribution of unique timestamps in the 10-minute data.....	61
Figure 3-19 Normalized distribution of the unique timestamps in the 1-minute data	62
Figure 3-20 Normalized distribution of table sizes in 5-week 10-minute training data	63
Figure 3-21 Normalized distribution of table sizes in 5-week, 1-minute training data	63
Figure 3-22 Normalized distribution of the number of unique locations in 1-minute training data	64
Figure 3-23 Normalized distribution of locations/timestamp (1-minute samples).....	65
Figure 3-24 User 016 locations at 1-minute polls.....	66
Figure 4-1 Predicting times at future locations using the Markov model.....	72
Figure 4-2 Data for User 013 (10-minute windows)	73
Figure 4-3 Predicting times at future locations using the Markov model including all predictions returned by the model.	75
Figure 4-4 A sample sequence.....	77
Figure 4-5 Using the SEQ model to predict when Alice will be at a meeting location (0th Order)	78
Figure 4-6 The SEQ model predicting times given previous context of a time or a location.	79
Figure 4-7 Flowchart: applying the SEQ model with both time and location as previous context.	80
Figure 4-8 SEQ model 2nd order: check that the quest location is in the sequence.....	81

Figure 4-9 SEQ model 2nd order: find context time in sequence.....	81
Figure 4-10 SEQ model 2nd order: find next occurrence of quest location	82
Figure 4-11 SEQ model 2nd order: time prediction	82
Figure 4-12 Two workweeks in Bob's life.....	83
Figure 4-13 Sequences truncated by the location in question.....	84
Figure 4-14 Sequences truncated or eliminated by additional context.	85
Figure 4-15 Portion of a training file used for the SEQ model.....	89
Figure 4-16 Aggregate results of the SEQ model on the 1-minute data.....	91
Figure 4-17 Aggregate results of the SEQ Model on the 10-minute data	91
Figure 4-18 Normalized distribution of SEQ model prediction results for the zeroth order (1-minute data)	92
Figure 4-19 Normalized distribution of SEQ model prediction results for the zeroth order (10-minute data)	92
Figure 4-20 Cumulative distribution of SEQ model prediction results for the zeroth order (1-minute data)	93
Figure 4-21 Cumulative distribution of results of SEQ model on 10-minute data.....	94
Figure 4-22 Average support and confidence metrics for correct predictions (1-minute data)....	97
Figure 4-23 Average support and confidence metrics for correct predictions (10-minute data)..	97
Figure 5-1 A hypothetical temporal prediction.....	104
Figure A-1 Example of ping-pong events.....	107
Figure A-2 Duration of the middle session during a ping-pong event	108
Figure A-3 Purging ping-pong records	109
Figure B-1 Thresholds for a hypothetical set of predictions.....	111
Figure B-2 Applying a probability threshold to time predictions for the 1-minute data	112
Figure B-3 Applying a probability threshold to time predictions for the 10-minute data	112
Figure C-1 Normalized distribution of prediction results of the SEQ model applied to the SigLoc data at 1loc order.....	114
Figure C-2 Normalized distribution of prediction results of the SEQ model applied to the SigLoc data at 1time order	115
Figure C-3 Normalized distribution of prediction results of the SEQ model applied to the SigLoc data at 2nd order.....	115
Figure C-4 Normalized distribution of prediction results of the SEQ model applied to the MoveLoc data at 1loc order	116
Figure C-5 Normalized distribution of prediction results of the SEQ model applied to the MoveLoc data at 1time order	116
Figure C-6 Normalized distribution of prediction results of the SEQ model applied to the MoveLoc data at 2nd order	117

List of Tables

Table 3-1 Structure of the PPM Markov model.....	58
Table 3-2 Test of statistical significance	65
Table 4-1 Average number of unique locations and times in training files.....	73
Table 4-2 Number of predictions returned for each test	74
Table 4-3 SEQ model counts for when Bob arrives at the office.	83
Table 4-4 Advantages and disadvantages of sequence approach	85
Table 4-5 Orders in the 'traversing the sequence' model	88
Table 4-6 Average number of sequences and pairs per sequence	89

Chapter 1 Introduction

Your location has much to say about what you are doing. You do not do the same activities in your office as you do in your kitchen. Knowing the location of your mobile device, and therefore your location, contributes to inferring your activity. Location-awareness is therefore a major component in context-awareness, which in turn, enables your devices to become your invisible assistants [1].

To be effective, these “invisible assistants” must be proactive instead of reactive. To be proactive, applications must not just sense the user’s current context, but also be able to predict the user’s future context. Applications must “infer the intent of the user” [2] and act accordingly. Existing location-based services report the current location of the user. In order to support context-prediction and proactive devices, location-based services must be able to *predict* location. Like the research goals at PARC [3], the goal of this dissertation is to progress from context-awareness to “contextual intelligence,” by transitioning from current location reporting to future location predicting.

The ability to predict the location of a user can support several types of context-aware applications such as assistive devices, reminder systems, social networking applications, and recommender systems. An application equipped with location-prediction can remind you how to prepare for your day and what to pack or let your friends know what you might be doing.

The ability to predict *when* someone will be at a given location can support collaborative applications by letting people know about their friends’ or associates’ routines. This knowledge

can help friends rendezvous or coworkers arrange impromptu meetings. Aides or social workers could use this application as a remote monitor, receiving an alert if discovering that someone did not arrive at a given location when they were expected.

The underlying assumption of this dissertation is that a person's movements are predictable: a person's history of movement is a good indicator of their future locations. Given that assumption, this dissertation has two objectives. First, the dissertation provides a method for predicting future *locations*, such as where someone will be at a given time sometime in the future. Second, it provides a novel method for predicting future *times*, such as when someone will arrive at a given location. These methods embed temporal information into location sequences and use these new *(time, location)* sequences to feed algorithms that can predict either someone's future location at a given time, or their time-of-arrival at a given location.

1.1 Contributions

This research makes four main contributions.

- **Future location prediction.** Existing research focuses on predicting next location, i.e. the next value in a sequence of locations. The research described in this dissertation uses a timestamp as context in order to predict not just someone's next location, but their location at any given time in the future, achieving a prediction accuracy of 78-92%.
- **The use of sparse data in location-prediction.** Current works use continuous location data, usually from GPS or cellular operators. This dissertation improves upon current methods by being able to make predictions without requiring continuous location data. This dissertation uses IEEE 802.11 wireless network data for location data. These data require less pre-processing and may include useful place names. However, these data are

sparse, with time gaps between wireless access point associations. The requirement to support sparse data inspired the development of a new representation to represent location data as *(timestamp, location)* pairs instead of ordered location sequences. This representation is successfully used to predict future locations and time-of-arrival at future locations.

- **Analysis of predictable data.** Historical data that result in high location-prediction rates can be characterized before training a model. This dissertation shows which of these characteristics can then be used to determine when a model needs to be retrained with more recent data or whether there is sufficient data of sufficient quality to warrant training the predictive model.
- **Time of arrival prediction.** This work is the first to predict *when* someone will be at a given location, in this case, by applying a heuristic solution. Temporal predictions can support rendezvous and collaboration. As a prediction problem, temporal-prediction is more difficult than location prediction because the number of possible temporal-predictions is much higher than the number of possible location-predictions. The heuristic model achieves prediction rates of 33 – 35%, which rise to 77 – 91% when a tolerance of 20 minutes is allowed.

1.2 Summary

In order to fulfill Mark Weiser’s vision of ubiquitous computing [1], mobile applications must evolve from being simply aware of their current context to being able to predict their future contexts. Location and time-of-day are key components of context. This dissertation contributes approaches to predicting both future locations and times at future locations. In other words, this

Introduction

dissertation answers two simple questions: how can we predict pedestrians' future locations and how can we predict when they will be at given locations?

Many prediction algorithms are based on predicting the next state in a series or sequence of states. These algorithms require historical data that are sampled regularly and are not missing any samples. The algorithms presented in this dissertation are not constrained by the requirement to have continuous data. A Markov model that is trained on timestamped location samples can predict future locations given a timestamp. An unsupervised heuristic model based on Market Basket Analysis can use similar timestamped location data to predict when someone will arrive at a given location. This heuristic model is also adaptable to different types of input or context in addition to simply timestamped location measures.

The models are applied to mobility data collected on PDAs given to 275 college freshmen. The location-prediction model predicts future locations with 78-91 % accuracy. The temporal-prediction model achieves 33-39% accuracy. If a tolerance of plus/minus twenty minutes is allowed, the prediction rates rise to 77%-91%.

The remainder of this dissertation is organized in the following manner. Chapter 2 begins with the motivation for this research, the theory of prediction and related work in location- and temporal-prediction. Chapter 3 focuses on predicting future locations and analyzing the characteristics of datasets that have high prediction rates. Chapter 4 focuses on predicting the times that someone will be at a given location. The dissertation concludes with Chapter 5.

Chapter 2 Background

This chapter begins with a list of the possible applications of location- and temporal-prediction. Some applications benefit the solitary user, such as smart reminders to oneself, while other applications show what can be done when predictions are shared with other people, such as friends or co-workers. Section 2.2 provides an overview of machine learning, data mining and different approaches to prediction. Section 2.3 covers temporal and spatio-temporal data mining. Section 2.4 reviews location determination and the issue of symbolic location. Related work in location prediction is in Section 2.5. The remainder of the chapter discusses the representation of time and location and explains the Prediction-by-Partial-Match algorithm used as the basis for the location-prediction model.

2.1 Motivation

Contextual intelligence is about combining current and past information in order to predict future situations [3]. Location is an important factor in one's "situation." Because of this, location-prediction can inform many types of proactive, context-aware applications. This section provides a survey of possible applications of both location-prediction and temporal-prediction.

Single user predictions can provide reminders and recommendations. When location- and temporal-predictions are shared with others, social networking can be enhanced, assistance can be given, resources can be reserved and advertising can be targeted. Section 2.1.1 considers applications for the solitary user.

2.1.1 Single-User Applications for Prediction

For a single user, location-prediction can be used for reminders and recommendations. A simple example is a reminder to pack up library books in the morning on a day you are expected to go past the library. The Magitti Recommendation System [4] remembers its owner's and other's past activities to recommend activities that its owners may enjoy at their current locations. With predictive capability, a recommendation system such as Magitti could additionally recommend activities for future locations and remind you what to bring.

A MOM for Independence

William Holbach is the Director of Assistive Technologies and Research Applications at Virginia Tech. Mr. Holbach works closely with students who need assistance to attend college and live in dorms. Mr. Holbach proposes a MOM, a *Micro-Omniscient Monitor*, and a NAG (*Not Achieving Goals*) [5]. The MOM would silently watch over the student's activities, and later, reward the student for being aware, accomplishing tasks correctly and finishing by deadlines. While the MOM silently observes and does not respond immediately, the NAG's function is immediate, proactively reminding the student of his surroundings, for example, an approaching bus stop, or his small, intermediate deadlines. The NAG would teach the user awareness of his surroundings, and help him complete large goals by helping him succeed with each small step towards the larger goal. These MOM and NAG functions provide "cognitive assistance" [6] to those who may need help due to mental disability, aging or simple distraction. Prediction is a necessary foundation of these applications.

Another example of cognitive assistance is when predictions are used to detect anomalies, such as when a user deviates from the expected path or routine [7]. The "Opportunity Knocks"

Background

system [6] tracks a user as he rides the bus home to his destination. If he accidentally boards the wrong bus, or gets off at the wrong stop, the system notices the discrepancy, politely “knocks” on the cell phone to alert the user, and directs him on where to go to catch the correct bus. The current implementation of the system asks the user to identify his destination before he boards the bus. With predictive ability, the application could either predict the user’s final destination or suggest a reduced set of likely destinations for the user to select.

Sharing Predictions with Others

As mentioned above, there are many situations in which prediction can assist the single user. Sharing predictions with others enables other diverse uses. Predictions can be shared with a caregiver to promote independence. Chang, Liu and Wang [8] developed a social networking system to assist people with mental illness. The system includes alarms for situations where users do not arrive at their expected destinations at expected times or if they deviate from their expected paths home. These predictions do not need to come only from mobile devices.

Mulligan [7] suggests that home appliances could also message a caretaker if their typical usage pattern was not being followed, indicating a possible problem.

An algorithm that can predict someone’s patterns can also determine when that person is being unpredictable and not following their usual behaviors. This *anomaly detection* has several uses. High-value-human-asset tracking services currently depend on people to watch tracking systems and learn people’s patterns of movements. An automated system that can learn patterns and alert the system to potential problems would be useful. Shklovski et al. [8] interviewed parole officers who tracked paroled sex offenders who had been fitted with GPS devices. Parole officers have the time-consuming task of learning parolee’s patterns, as well as determining when an anomaly has occurred and then judging the severity of the anomaly and the appropriate

Background

response. The addition of the GPS devices has made tracking easier, but has increased the officer's workloads as officers spend more time watching computer monitors. An automated application that flags potential anomalies would decrease their workloads.

Social networking can also be enhanced using predictions. When predictions are shared, a friend can determine when to "run-into" or serendipitously rendezvous with someone (or avoid them). Common destinations could be used for social matchmaking [9]. Merchants could use predictions to target advertising, perhaps offering a coupon to lure you to a different restaurant for lunch than your usual stop. In the summer of 2010, PepsiCo used location-based services to reward customers who frequented their retail outlets [10]. Not only is the location information used to target marketing materials, but the temporal information about when shoppers buy their products will also target their marketing. Currently, companies market to people who are at or near their retail locations. Marketing to users who are predicted to go near their retail locations is a logical next step.

Ashbrook and Starner [11] suggest several multiple-user applications of location information in addition to those already mentioned. One application is the exchange of favors, such as errands that someone else could possibly do in your place if they are closer to a location than you are. Another application could support coordinating a meeting of several people.

2.1.2 Privacy

A discussion on the topic of predicting someone's activities is not complete without the consideration of privacy. However, this dissertation is not going to attempt to find the perfect boundary between usefulness of this application versus invasion of the user's privacy. Some of the applications mentioned above would be considered intrusive by some and helpful by others.

Background

Any system that uses location- or temporal-prediction must be designed from the start to support the privacy of the user. All prediction algorithms are based on past history, which means that tracking will be done. The tracked information could be kept private by keeping the data on the user's local device or on a secure server. The data should be under the user's control to edit or delete. The user must be aware of the types of data that are collected, what is done with the data and who will have access to the data [12].

Privacy can be supported by location-prediction if it replaces tracking. For example, a corporation that tracks employees in order to find the closest technician in case of a problem can conceivably use location-prediction, instead, to produce a list of technicians who are likely to be nearby. In the article "Internet Predictions" [7], Estrin et al. propose creating *personal data vaults*: secure containers holding personal data streams. Only owners would have access to their personal data. The vault would then selectively share information by filtering the data or by processing computations on the personal data before sharing. Location- and temporal-predictions are instances of the types of information such vaults could share without revealing more personal information.

2.1.3 Predicting Time

As discussed previously, knowing someone's location is a major clue in inferring their context and activity. Temporal information does not replace location for context-awareness but can inform different kinds of context-aware applications.

One application of temporal knowledge is awareness of each other's presence. When coworkers share an office or a building, they learn each other's routines. For example, Ed does not come in until noon on Mondays, and Jane has class at 4:00 pm. Coworkers use this information to know

Background

when they can meet with someone or find them to talk. Kraut et al., when discussing informal corporate communication [13], argues that “informal communication, generally mediated by physical proximity, is crucial for coordination to occur.” Knowing *when* someone is going to be around can trigger those hallway meetings that can be surprisingly beneficial. A special case is *ambushing* [14], which “is the practice of waiting for a colleague to become present at a particular location in the interest of meeting with him. The location is known in advance, and the time at which the colleague will appear is typically not.”

This information is also useful for remote workers. Begole et al. [15] developed a temporal-awareness application which predicted when someone would be at the office. This application was developed to help remote coworkers to get a sense of the availability of their distant colleagues, so they would know when they could best reach each other.

Temporal information can be exploited for targeted marketing. While location-prediction can be used to target ads to the user—such as offering a coupon for a competing restaurant if the user is headed out to lunch—time-prediction could also be useful to further match promotions to potential customers. Perhaps a restaurant offers breakfast coupons instead of dinner coupons if Bob always goes to a particular restaurant for breakfast. Or his usual breakfast place may want to let him know that they have a dinner menu, too. Advertisers can deliver their targeted advertising at the right time—before the customer goes to a restaurant, not after [16].

Busy freeways have signs that tell the number of minutes it will take to get from the current location to the upcoming significant exits or downtown. A system that knows your patterns could also remind you that when you go somewhere at a given time or on a given day of the week, it may take longer. For example, drivers need to allot extra time if crossing campus

during class breaks, as the crosswalks are packed with students moving between classes and traffic is stopped.

2.2 Overview of Prediction Techniques

Machine learning, data mining, pattern recognition and statistical inference share common goals of finding patterns in data and interpreting those patterns for useful conclusions. While this dissertation uses the words ‘‘machine learning’’ to encompass all of these algorithms, keep in mind that many are based on pattern recognition and statistical inference. This section begins by taking an abstracted view of machine learning. Later sections will take a closer look to describe both the problem space of the future-location-prediction problem and the characteristics of the solution. Based upon this examination of machine learning, this dissertation will show that location-prediction is a sequence prediction problem while temporal-prediction is solved using Market Basket Analysis.

An overview of machine learning is shown in Figure 2-1. Machine learning techniques can be divided into two classes by the way that they learn: supervised and unsupervised learning. Supervised learning techniques use historical data that have been labeled with the correct answers. These data are used as training data to develop a model, which is then applied to new data to produce predictions or classifications. Unsupervised learning techniques do not use previous data to build a model and they may or may not have knowledge of the correct answers. Unsupervised learning techniques attempt to discover the patterns in the data.

2.2.1 Supervised Learning

Supervised learning can be divided into two types of techniques: regression and classification.

Regression involves fitting a function to numerical data and then interpolating or extrapolating to calculate missing or future values. Examples include linear regression or least squares regression. Classification techniques group the data into predefined output categories. An example of a binary classifier (one with only two output categories) is a disease risk model, which takes, as input, a set of physical symptoms or characteristics, and outputs a yes or no value as to whether the patient is at risk for a given disease.

Discrete Sequence Prediction

Sequence prediction, which predicts the next item in a sequence, can be considered a type of classifier. The possible outputs are the elements in the alphabet used to create the sequence. They are known in advance, and the model predicts which item is next in the sequence. This

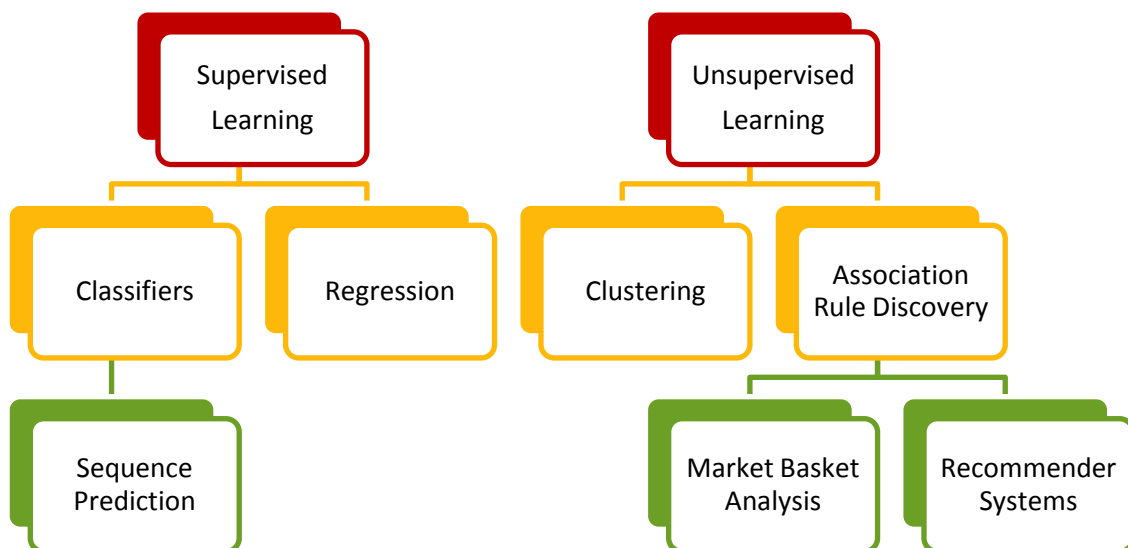


Figure 2-1 Overview of machine learning

section presents the formal definition of the sequence-prediction problem, as it forms the basis of the initial location-prediction model.

Let Σ signify the alphabet of possible symbols, s_1, s_2, \dots, s_n , where n is the number of symbols in the alphabet. The training sequence presented to the model consists of t symbols, $x_1, x_2, x_3, \dots, x_t$, where $x_i \in \Sigma$. The model calculates the conditional probability.

$$\Pr\{X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \dots\} \quad (2-1)$$

This model is called a *stationary Markov chain* [17]. The term *stationary* is included because the probabilities are not time-dependent and sub-sequences can repeat at different points in the sequence. For every shift, m , and for all $x_i \in \Sigma$,

$$\begin{aligned} \Pr\{X_1 = x_1, X_2 = x_2, \dots, X_n = x_n\} \\ = \Pr\{X_{1+m} = x_1, X_{2+m} = x_2, \dots, X_{n+m} = x_n\} \end{aligned} \quad (2-2)$$

The process is said to be Markov because the probability of a given symbol depends on the preceding variable(s). The number of preceding variables used, l , is the *length*, or *order*, of the model. The sub-sequence of preceding variables is called the *history* or the *context*.

If the length, l , of the context is set to a constant, the model is called a *fixed-length Markov chain*. In a *variable-length Markov chain*, the length, l , of the context used can vary up to some maximum number. A variable-length Markov chain was used in the author's preliminary work on location-prediction [18]. A first-order Markov model forms the basis of the location-prediction model presented in this dissertation.

2.2.2 Unsupervised Learning

The right half of Figure 2-1 shows the types of unsupervised learning. Unsupervised learning techniques do not build a model from historical training data. Instead, they attempt to discover relationships in the dataset. Clustering, also referred to as unsupervised classification, discovers groups of similar items. Unlike supervised classification, clustering techniques do not know the number or labels of the output categories ahead of time. An example of clustering which pertains to our subject is the work done by Ashbrook and Starner [11, 19] to cluster neighboring GPS readings into locations of significance. The resulting groups can either be labeled by the algorithm automatically with the center point of the cluster or manually by asking the user to enter a meaningful name.

Unsupervised learning techniques are also used to find relationships between items in a large dataset. One example is Market Basket Analysis, which results in a set of association rules. For example, a grocery store might use Market Basket Analysis to discover that when shoppers buy tortilla chips, they also buy salsa. (Hopefully, Market Basket Analysis can also produce results that would not be obvious to a human analyst, like the connection between beer and diapers, which is now considered a data-mining urban legend [20].) Recommendation systems, such as those used by Amazon to recommend books, can use unsupervised learning, searching through historical data to see what similar customers have purchased.

2.2.3 Number of Outputs

Another approach to categorizing these different algorithms is by the number of outputs they generate. Classification, regression and clustering tend to produce one or few outputs. A regression function used for prediction is given an input value and returns a single output.

Background

Classifiers have a predetermined number of output states, which is usually small. When a classifier is used as a predictor, the number of output states is equal to the size of the alphabet. Clustering algorithms are similar to classifiers, except that the number of output states is not predetermined.

On the other hand, unsupervised algorithms which discover associations may return a large number of association rules. In practice, a threshold is usually applied to limit the number of association rules or outputs that are considered to be potentially useful.

Consider both sides of the original problem: predicting future locations and future times at those locations. The expectation is that a query about a future location will receive one or at most a few results: e.g., Bob is either at the office or at home. A query about *when* Bob will be at the office, the expected result is either a list of times (8:00 am, 8:10 am, 8:20 am...) or ranges (8:00 am-10:00 am, 1:00 pm-5:00 pm), or perhaps even a probability density function. Because of this difference, a basic assumption of this dissertation is that the predictor for future locations will come from the left side of Figure 2-2 while the best approach for predicting future times will be found on the right side of Figure 2-2.

This contributes to this dissertation's hypothesis that sequence prediction can be used to predict future locations, but an approach such as Market Basket Analysis or a recommender system would be better for the temporal-prediction problem.

2.3 Temporal and Spatio-Temporal Data Mining

Until this point, this dissertation has introduced only general-purpose predictors. There is a field of data mining which is concerned with datasets that include temporal and/or spatial data. The prediction data used in this research contains both. This section discusses temporal and spatio-

Background

temporal data mining and their application to these problems of location- and temporal-prediction.

Data mining applies machine learning techniques to large databases. If the input data contain time information or time-dependent attributes, the techniques fall into the sub-category *temporal data mining*. For example, temporal data mining is used in engineering to extract information from sensor readings or to predict future Web pages accessed, in healthcare to analyze ECGs and in finance to determine trends or predict future prices [21]. Jakkula and Cook use temporal relationships between activities, such as *before*, *after* or *overlap*, to improve the activity predictions made in the MavHome [22], a testbed house filled with sensors at the University of Texas Arlington.

Problems that include moving objects, or data with both spatial and temporal attributes, may require *spatio-temporal data mining*. Spatio-temporal data mining is concerned with finding relationships between spatial and non-spatial data, such as topological (*intersects*, *overlaps*),

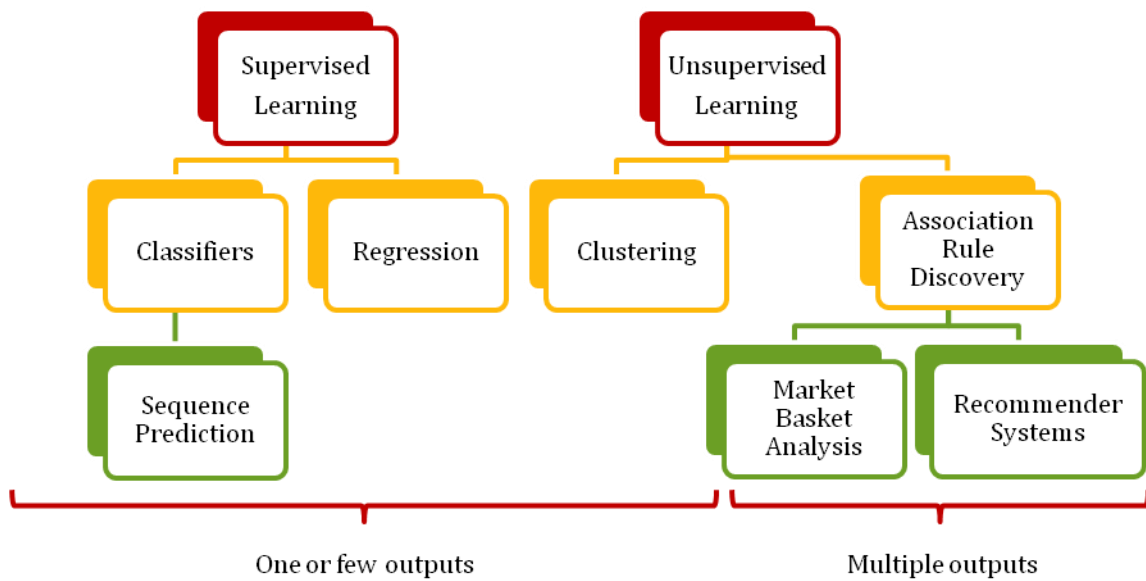


Figure 2-2 Outputs of machine learning algorithms

spatial orientation (*left of, on top of*) or distance information (*close to*).

Elnekave, Last and Maimon [23] use a spatio-temporal approach to predict future locations. Sections of a driven path are clustered into Minimum Bounding Boxes (MBB) which include time ranges. Prediction consists of finding the MBBs within the same (or close) time range as the query.

This dissertation's goal is to predict when people will be at their significant locations, not the paths they take between them. It is not concerned with relationships between locations, such as "close to" or "left of." Therefore, this dissertation assumes that the location-prediction problem is a temporal problem and not a spatio-temporal problem. The following section discusses the taxonomies of temporal data mining to see where the prediction problems that are the subject of this dissertation fall in the temporal problem-solving space.

2.3.1 Taxonomies of Temporal Data Mining

Hsu, Lee and Wang [24] classify temporal data mining problems by two aspects: first, the extent that time participates in the mining process, and second, the type of input data. These features are summarized in Figure 2-3. Roddick and Spiliopoulou [25] add a third criterion, the mining operation, to their taxonomy. Both taxonomies assume that the input data are continuous, real-valued, non-noisy data, such as daily temperatures or stock prices. Another taxonomy, by Antunes and Oliveira [21], classifies temporal data mining techniques by a) the representation and modeling of the data sequence, b) the definitions of similarity measures between sequences and c) the application of the model to actual problems. Moerchen [26] summarizes the various aspects of temporal data mining. Again, most of the discussion concerns continuous, real-valued

Background

data, often referred to as time-series. In contrast, the data in this dissertation are not time-series, but categorical, ordered discrete *sequence data*.

Sequence learning can also include problems where the current state may not only be dependent on the previous state, but also on a state that happened some time previously. Sun and Giles [27] note that “many existing models have difficulties handling such dependencies.” This dissertation’s data may have these dependencies. For example, stopping for groceries on the way home from work is still an instance of moving from the office to the home.

The left side of the taxonomy figure lists the possible uses of time information. A vast majority of, if not all, prediction algorithms fall into the left-most category. Temporal information is used at the preprocessing stage to put a series of measurements or events in order. Once a sequence is built, the time information is discarded. The resulting sequence shows that state x_t occurs after

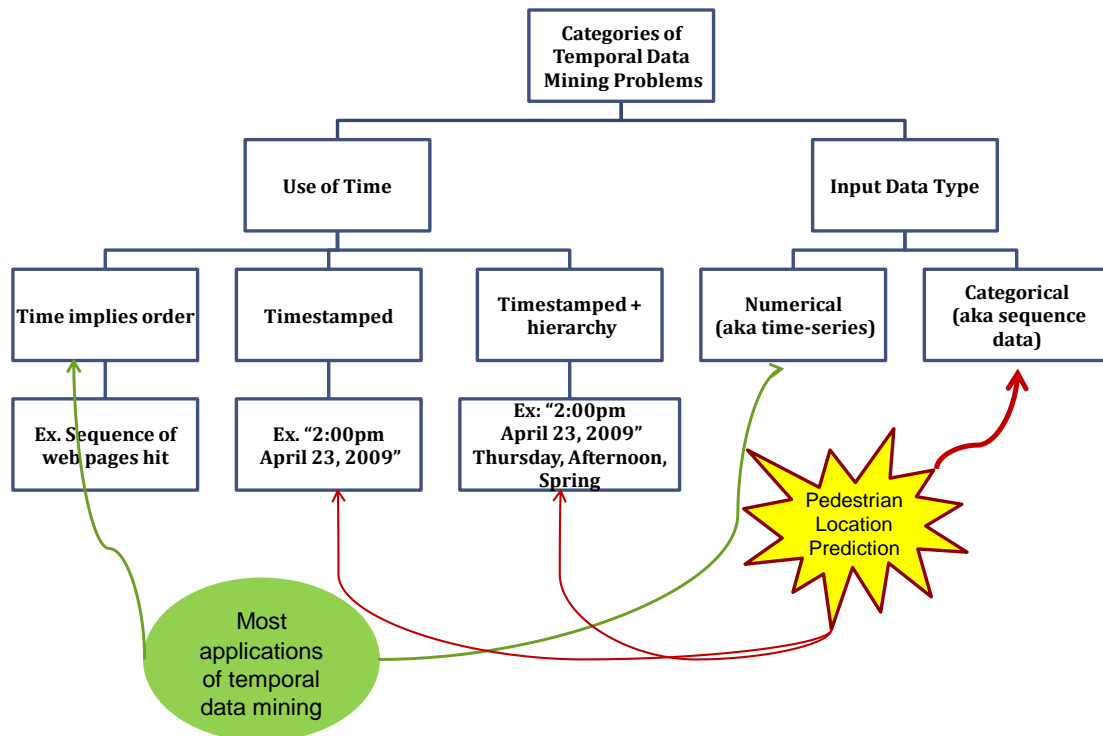


Figure 2-3 Types of temporal data mining problems

x_{t-1} and before x_{t+1} , but there is no absolute time information as to the value of t . For example, one cannot determine which state occurred at an absolute time such as 10:00 am on 1 May 2009.

In this study, the absolute time information is retained so that the predictive model can answer questions such as, "Where will our subject, Bob, be at 10:00 am?" This places this application in the middle category of time-stamped data points. Prediction using time-stamped data is one of the contributions of this work.

2.3.2 Enhancing a Sequence with Temporal Information

Some researchers have expanded location information by encoding temporal information within the spatial information. Lymberopoulos, Bamis and Savvides [28] recorded the activities of an elderly person living in a smart home. Triplets that contain (*room, time, duration*), such as {*Bed, 11:00pm, 300*}, are condensed into symbols and then abstracted into activities such as *Bed_Night_Sleep*. The symbols are then counted and the most frequent symbols are used for activity detection. This dissertation's approach is similar in that it must enhance a location sequence with temporal information. It does not combine the two into one symbol.

The previous sections provide a general overview of machine learning and data-mining with temporal information. They have shown that location-prediction is based on discrete sequence prediction while temporal-prediction can be done using an unsupervised learning technique.

The following section discusses location determination, specifically comparing GPS to IEEE 802.11 based location-determination systems.

2.4 Location Determination

Previous research projects relied on various methods to determine location, including manual records [29], environments with built-in location-monitoring infrastructure [30], GPS [11], and association with IEEE 802.11 wireless access points. These methods have their advantages and disadvantages. Manual records may be missing locations, but do emphasize key locations.

Environments with built-in infrastructure are likely to have precise measurements of location, but are expensive to implement and are not universally available. GPS does not work indoors or in urban canyons and requires extra processing to translate a range of GPS readings into a single significant place. In addition, GPS data are not currently readily available.

Other location determination systems include hybrid systems, such as Skyhook, which uses GPS, cellular tower triangulation and IEEE 802.11 access points [31]. Location systems based on cellular towers (cell ID) may be able to determine location within 5 meters [32].

This project uses IEEE 802.11 wireless access points as beacons for determining location [31, 33]. The advantages of this method are that, currently, IEEE 802.11 access points are ubiquitous on college campuses, providing a global location system. Many mobile devices include IEEE 802.11 radios, which make the location system inexpensive. If the location information is kept on the user's device and not broadcast, privacy is supported. Currently, location by wireless access point is the only globally available solution that works indoors, where most of us work and live.

There are a few disadvantages with using IEEE 802.11 access points to determine location. To use an IEEE 802.11 access point as a reference point, its exact location must be known.

Currently, most access points are installed without records of their exact locations. Corporate

users, such as universities, record access point locations for maintenance purposes. If the use of Voice-over-Internet-Protocol (VoIP) on mobile devices continues to grow, E-911 regulations will require that the device can be located in case of an emergency. It seems a logical conclusion that in the future, access points may be programmed to broadcast their locations.

Environmental factors can affect how strongly the access point signal is received by the mobile device. Because of these variations in received signal strength, location estimates using IEEE 802.11 wireless access points are not very precise, determining location within approximately 32 meters [34].

With the advent of location systems based on IEEE 802.11 wireless access points such as Place Lab [33], and public collections of wireless usage data such as CRAWDAD [35] and UCSD [36], it is now feasible to apply prediction algorithms to larger groups of people over indoor and outdoor locations.

2.4.1 Symbolic Location

For location-based applications to be useful, geographical coordinates, such as 37.2302323948768,-80.42062044809619 (Torgersen Hall) need to be translated into *symbolic locations* [37] or *places* [38, 39], such as “Torgersen Hall”, “work”, “office” or “lab.”

For more precise systems such as GPS, this translation requires clustering groups of GPS readings to find significant locations and then asking the users to label these significant locations [11]. (This research is described in more detail in section 2.7.2.) Sensors that are placed on walls or ceilings, such as Cricket [40] or Active Badge [41] can include this translation as part of the location system.

Background

Some IEEE 802.11 wireless access points broadcast their “place” because they have been programmed with location information as part of their Service Set Identifier (SSID). For example, the IBM access point data available as part of the CRAWDAD research database [42] includes a Building number as part of the access point ID. Residential access points are often programmed with a surname; the corresponding physical (postal) address is publicly available in the phone book.

The granularity of the location measurements returned by an IEEE 802.11 based positioning system can be an advantage when translating positions into places because it removes the need to first cluster the lat/long readings and then translate them into a place.

2.5 Related Work in Location- and Temporal-Prediction

There are two questions to be answered when attempting to predict: *what* we are trying to predict and *how* we are going to try to predict it. This dissertation predicts both future locations and the times that someone may be at those locations. The following sections describe existing projects that predict either location or time.

2.5.1 MavHome

The MavHome (“Managing an Adaptive Versatile Home”) at The University of Texas at Arlington was a smart home which sought to “maximize inhabitant comfort and minimize operation cost.” [30] The inside and surrounding area of the home was divided into zones (as shown in Figure 2-4) in order to track the inhabitants’ locations. The MavHome proposed to use location prediction in order to know which motion sensors to poll to find the inhabitant. The prediction serves two purposes, reducing the number of sensors which need to be polled and allowing longer time periods between location polls. In addition, the predictions can be used to

Background

allocate resources, such as adjusting the lights and temperature in rooms that are soon to be occupied [43]. The zones are modeled as a graph where the nodes are places (e.g. rooms) and the edges show connections between places, like hallways or doorways.

Prediction is done using the LeZi-update algorithm [44], an update scheme based on the dictionary-based LZ78 compression algorithm [45]. Movement history is stored as a string of zones, for example, *mamcmrkdkgoo*. The LZ78 compression algorithm encodes variable length string segments using fixed length dictionary indices, updating the dictionary as new ‘phrases’ are seen. For example, the string of zones above would be parsed into the unique phrases *m*, *a*, *mc*, *mr*, *k*, *d*, *kd*, *g*, *o*, *og*. Common phrases represent common paths through the house. The phrases and their frequencies are stored in a tree and are used to calculate the probabilities of each phrase, given the movement history. Results [46] report prediction success rates of ~94% for a retired person, ~90% for an office employee and ~85% for a graduate student. The current implementation of this model predicts the next location and path.

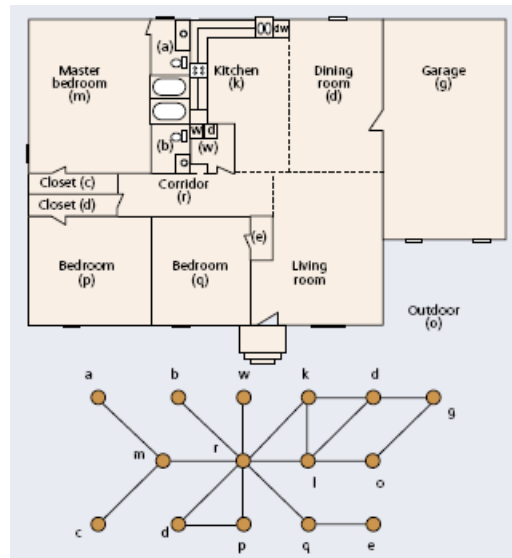


Figure 2-4 A graph model of a smart home floor plan
(S.K. Das, D.J. Cook,, A. Battacharya, E.O. Heierman III, Lin Tze-Yun, The role of prediction algorithms in the MavHome smart home architecture, Wireless Communications © 2003 IEEE)

Jakkula and Cook [22] used temporal relationships between activities (such as using a “cooker”, an oven and a lamp in a MavHome) to improve predictions of the inhabitant’s next activity. With a synthetic dataset, they found a 7.81% improvement in predictions of the inhabitant’s next activity. For their small dataset of “real” activities, the addition of temporal information improved activity predictions by 1.86%, showing that adding temporal information can improve prediction accuracy.

2.5.2 Using GPS to Determine Significant Locations

In [11, 19], Daniel Ashbrook and Thad Starner analyze a large collection of GPS data in order to predict users’ next ‘significant locations.’ Initially, GPS data are collected for a mobile user. The data are pared down into *places* by keeping only the data where the user stopped for more than 10 minutes or lost a GPS signal (entered a building). Because few GPS readings in the same significant location will exactly match, an iterative clustering algorithm is used to collect readings from the same general location to produce a set of *significant locations*.

An example of this process is shown in Figure 2-5. Of the five users studied, three had 11 significant locations, one had 9 and one had 6. People have a small number of significant locations.

Later, users were asked to label the locations tagged as ‘significant’ to see if the algorithm was truly capturing locations that were important to the user. At most, users had two locations which they were unsure about (which may be due to the fact that the users were asked for the location names several months after they had left Zurich, the city in the study.)

A second-order Markov model was used to predict the user's next location. Exact results were not reported, but the results achieved by the Markov model were significantly higher than those predicted by random chance using a Monte Carlo simulation.



Figure 2-5 Ashbrook & Starner's technique to produce significant locations from GPS data.
With kind permission from Springer Science+Business Media: Personal and Ubiquitous Computing, "Using GPS to learn significant locations and predict movement across multiple users", vol. 7, 2003, page 283, D. Ashbrook and T. Starner, Fig. 11.

The results of Ashbrook and Starner's work provide assurance that mobility data can be used to derive the places that are important to people (their 'significant locations') and that prediction can be successfully applied. Other approaches to determining 'places' from latitude and longitude are summarized in [38, 47].

2.5.3 Dartmouth College Mobility

Predictions

Researchers at Dartmouth College applied several prediction algorithms to extensive mobility traces of over 6,000 users collected over a two-year period [48]. As in the MavHome, movement history is stored as a string of location symbols. In this case, each letter in the movement string represents the IEEE 802.11 access point with which the mobile user is associated. The string includes location *changes* only and no time information is recorded. Several predictors were

considered, including Markov predictors and LZ-based predictors. They found that the simple low-order Markov predictors worked as well or better than the other predictors, including higher order Markov models, which confirms that recent history is shown to be a better predictor than the probabilities determined over long historical traces. When a predictor failed to make a prediction due to encountering a history it had never seen before, a fallback procedure was implemented which allowed the predictor to use shorter and shorter context strings until it could make a prediction. This approach is similar to the Prediction-by-Partial-Match (PPM) algorithm, which is described in section 2.8.2. This fallback procedure improved the accuracy, resulting in a total accuracy of 65 -72% for the second-order Markov predictor with fallback.

This project is the only large scale project using IEEE 802.11 positioning of a large number of users. The result of 65-72% prediction is this dissertation's baseline. This dissertation adds temporal information to improve the predictions and allow us to ask questions about the users' locations further out into the future.

2.5.4 Smart Office Buildings

Jan Petzold, in his PhD research [29, 49], applied several machine learning algorithms to predicting an office worker's location. Four office workers manually tracked their location for a summer. A Smart doorplate was developed and attached to their office doors and when a visitor came and found the office empty, the smart doorplate predicted the location of the absent office worker. Five machine learning algorithms were applied to the data: two neural networks (Elman Net and Multi-layer Perceptron), a Bayesian network [50], a State predictor [51] and a Markov predictor. Prediction accuracies ranged from 68% to 91%. However, none of the algorithms was optimal for all four subjects. In other words, there was no universal predictor.

Petzold also applied confidence estimation techniques (the strong state method, the threshold method and the confidence counter) [52] to the State Predictor method in order to decide if it is better to make no prediction rather than one that has a high probability of being incorrect.

2.5.5 Reality Mining at MIT

One of the project themes in the Reality Mining group at the Massachusetts Institute of Technology is modeling and predicting user behavior. Mobile devices log the time of day, nearby cell tower IDs and Bluetooth devices. The presence of other Bluetooth-enabled phones can indicate relationships that can be used to classify behavior: for example, business school students are likely to be in the same locations and exhibit similar movement behavior [53]. Bluetooth beacons were set up in subjects' homes and used to validate the results of three methods used to translate cell tower information into locations [54]. A Dynamic Bayesian Network was used to predict the subject's next location and reached very high accuracies of 93 to 99%.

2.5.6 Other Related Work in Location-Prediction

Zeibart et al. [55] predict driving destinations given a partially traveled route by calculating the probabilities of different possible trajectories. This dissertation's goal of pedestrian destination prediction is similar. While the authors can assume continuous location updates and create a trajectory of movement, this dissertation does not. Locative devices such as cell phones will occasionally be turned off, get left behind or enter areas where location systems do not have coverage, such as GPS "urban canyons." Applications such as Facebook [23] and Twitter [24] rely on user updates, meaning there will be significant time lapses between location updates. In

addition, to support privacy concerns, location-logging applications must allow users to delete records or stop recording desired [25-27].

Bauer and Deru [56] propose two methods for predicting future destinations given past histories. A Market Basket Analysis of many users could be used to find associations between locations, producing rules such as “if the user visits a shoe store, she will also visit a restaurant.” A clustering algorithm was also proposed to determine a user’s common paths. The authors proposed the algorithms but did not provide any results.

Lymberopoulos, Bamis and Savvides [28] applied our early results in using PPM-C [18] to predict the number of users at cellular system base stations. They used two algorithms, PPM-C and LAST, to predict which base station a mobile user will connect to next. PPM-C is a Markov Model described in detail in Section 2.7. LAST is a heuristic model which assumes that the user is not moving and the next base station will be the same as the previous base station. A Market Basket Analysis of the patterns of movement was used to determine which algorithm to apply for a given portion of the historical data. For example, Market Basket Analysis of the historical data showed that if the user stayed connected to the same base station for four time steps (“aaaa”), they were likely to still be connected to the same base station for the next timestep.

Subsequently, in cases where the historical data showed a pattern of *aaaa*, the LAST algorithm was applied and the most recent base station was predicted to also be the next base station. If the Market Basket Analysis had not found a common pattern, the PPM-C predictor was applied.

These algorithms return the single best location-prediction, which the authors refer to as the *hard* decision. The authors then use the vector of all the possible predictions returned, referred to as the *soft* decision, and show that aggregating the results of the soft decisions yield better results

Background

when tracking a group of users. While this dissertation's algorithms are used for single users and the results are not aggregated, later chapters will show that the use of multiple predictions, i.e., the soft decisions, can return useful results.

Vu, Do, and Nahrstedt recently published the results of their location, duration and contact predictor [57]. Their predictor uses a Naïve Bayesian classifier based on records that consist of a type-of-day (*weekday*, *weekend*) and a timeslot of 1 to 8 hours. Each type-of-day, timeslot category contains multiple records which consist of a timestamp, a location and a list of Bluetooth MAC addresses discovered. The Bluetooth MAC addresses correspond to people encountered by the user. Given an input of the type-of-day and a timeslot, the predictor returns the most likely location predictions (up to three), the mean and standard deviation of the duration measurements and the people (Bluetooth MACs) mostly likely encountered at that location. They achieved successful results for their tests on 50 users. When they used a 2-hour timeslot and considered the top two location predictions returned by the predictor, 80% of the test runs had more than 70% correct predictions. When they varied the timeslot size from 1 to 8 hours [58], the results did not change significantly, which implies that the predictor may be returning the most common location(s) (e.g., the office) and not predicting locations which were visited for a short period of time.

Calabrese et al. [59] combine collective movement patterns, time of day, land use and points of interests to build a probabilistic model which can predict an individual's next location within an hour correctly 60% of the time. This work may be one of the first to incorporate geography into a model and to apply the movement patterns of the group to individual location predictions.

Background

Monreale et al. [60] applied spatio-temporal data mining to trajectories in order to predict the next region and arrival time of a vehicle. The GPS location data is clustered into regions, and then the transitions between regions are encoded into a trajectory tree. Each child represents the transition from the parent region to the child region and is labeled with the support for that transition and a time range for moving from the parent region to the child region. Partial trajectories are then matched with pathways in the tree to predict future regions. When the model allows for some spatial and temporal tolerance, the prediction accuracy can be greater than 54%. This model applies collective vehicle behavior to individual predictions and also can predict arrival time.

The authors of [61] use time of arrival and duration information to predict future locations. Because their algorithm uses duration at the last location to determine the next location, it requires continuous data, which is not a requirement for the algorithms presented in this dissertation.

2.5.7 Summary

The beginning of this chapter described several research projects where a mobile users' next location was predicted. Other than the Dartmouth study, most of these projects use a small number of subjects and a small number of locations (~10) to predict only the users' next location. The study by Vu et al. uses more people but their large timeslots of 1-8 hours limit the number of locations. Begole does not predict location directly, but predicts the time someone will be online, which implies that they are in their office.

2.6 Representation

This chapter began with an overview of machine learning and data mining and then delved into previous research on location- and time-prediction. The previous research predicts either someone's next location or the time when they may be online, which are all problems in one dimension, either location or time. The prediction goal of this dissertation includes location *and* time, which are two separate dimensions. A corresponding representation is used in the related work described in the following section.

2.6.1 Representing Music

In their experiments comparing variable-order Markov models, Begleiter et al. [62] applied six algorithms to the prediction of musical selections. Musical notation has multiple variables: notes, their starting times and their durations. These variables were coded as character strings and prediction algorithms which are normally applied to a single dimension were able to recognize the patterns in the music. The multi-dimensional specification of music correlates with the specification of a user's path throughout the day, which consists of locations, starting times and durations.

In Begleiter's experiments, prediction performance was measured using the *average log-loss*.

Given an alphabet Σ , an algorithm is trained on a sequence $q_1^n = q_1 q_2 q_3 \cdots q_n$, where $q_i \in \Sigma$.

Given a test sequence $x_0^T = x_0 x_2 \cdots x_T$, the average log-loss $l(\hat{P}, x_0^T)$ is given by

$$l(\hat{P}, x_0^T) = -\frac{1}{T} \left[\sum_{i=1}^T \log_2 \hat{P}(x_i | x_0 x_1 \cdots x_{i-1}) + \log_2 \hat{P}(x_0) \right] \quad (2-3)$$

The log-loss equation relates to the number of bits required to compress a test string given that the algorithm was trained on another string. The average log-loss measures the average compression rate of the test string. In other words, the smaller the log-loss, the fewer bits are required to compress the string. Minimizing the average log-loss corresponds to maximizing the probability assignment for the test sequence. In other words, the smaller the average log-loss, the more predictable the test string. For example, if the model was trained on the string ‘*abracadabra*’, the average-log-loss for the test string ‘*abr*’ would be less than the average-log-loss for the test string ‘*car*,’ because the string ‘*abr*’ occurs as a pattern in the test string.

Begleiter et al. found that the Prediction by Partial Match (PPM) algorithm resulted in a low average log-loss when tested on MIDI (music) files. The following section describes the Prediction by Partial Match algorithm in detail, as it will be applied to location-prediction in Chapter 3.

2.7 The Prediction by Partial Match Algorithm

The Prediction by Partial Match (PPM) algorithm uses various lengths of previous contexts, or historical data, to build the predictive model [63]. As a training string is processed character-by-character, a table is built for each sub-string and the characters that follow it, including a count of the number of times that character has been seen occurring after that sub-string. For example, if the training string is *abracadabra*, the training begins by building an entry for *a* with count 1 in the order-0 table. It then adds an entry for *b* to the order-0 table with a count of 1, and begins the order-1 table, by creating a table for ‘characters which follow *a*’ with an entry labeled *b* with a count of 1. When the training is over, an ESCAPE character is appended to each table. This character is used during encoding to mark situations where novel characters are seen. In the

Background

‘Method C’ variation of PPM (called PPM-C), the ESCAPE character is given a count equal to the sum of the number of different symbols that have been seen in that context. Figure 2-6 below shows the PPM-C model after training on the string *abracadabra*, with a maximum order of 2. The values under the heading ‘c’ are the number of times that character has been seen following the given context. The ‘p’ column is the probability, which is the count for the given character divided by the sum of all the counts in that sub-table.

Order $k = 2$					Order $k = 1$					Order $k = 0$					Order $k = -1$				
Predictions			c	p	Predictions			c	p	Predictions			c	p	Predictions			c	p
ab	→	r	2	$\frac{2}{3}$	a	→	b	2	$\frac{2}{7}$	→	a	5	$\frac{5}{16}$	→	A	1	$1/ A $		
	→	E_{sc}	1	$\frac{1}{3}$		→	c	1	$\frac{1}{7}$		→	b	2		$\frac{2}{16}$				
ac	→	a	1	$\frac{1}{2}$	→	d	1	$\frac{1}{7}$	→	c	1	$\frac{1}{16}$	→	d	1	$\frac{1}{16}$			
	→	E_{sc}	1	$\frac{1}{2}$		→	E_{sc}	3		$\frac{3}{7}$	→	r		2	$\frac{2}{16}$				
ad	→	a	1	$\frac{1}{2}$	b	→	r	2	$\frac{2}{3}$	→	E_{sc}	5	$\frac{5}{16}$						
	→	E_{sc}	1	$\frac{1}{2}$		→	E_{sc}	1	$\frac{1}{3}$										
br	→	a	2	$\frac{2}{3}$	c	→	a	1	$\frac{1}{2}$	→									
	→	E_{sc}	1	$\frac{1}{3}$		→	E_{sc}	1	$\frac{1}{2}$										
ca	→	d	1	$\frac{1}{2}$	d	→	a	1	$\frac{1}{2}$	→									
	→	E_{sc}	1	$\frac{1}{2}$		→	E_{sc}	1	$\frac{1}{2}$										
da	→	b	1	$\frac{1}{2}$	r	→	a	2	$\frac{1}{3}$	→									
	→	E_{sc}	1	$\frac{1}{2}$		→	E_{sc}	1	$\frac{1}{3}$										
ra	→	c	1	$\frac{1}{2}$															
	→	E_{sc}	1	$\frac{1}{2}$															

Figure 2-6 PPM-C Model after training on the string abracadabra.

The column, c, is the count of the number of times that character (or set of characters) occurred in the training string. The column, p, is the probability of that character occurring in the given context. Escape characters are returned to tell the model to drop to a lower order. (J. Cleary, I. Witten, *Data Compression Using Adaptive Coding and Partial String Matching*, IEEE Transactions on Communications ©1984 IEEE)

To use the PPM-C model for prediction, the tables are traversed given the context. For example, if the context given is *ab*, the ‘Order $k=2$ ’ table is searched first to see if there is an entry for *ab*. Since there is an entry for *ab*, the prediction engine simply reports the character(s) with the highest probability, in this case *r*, which is reported to have a probability of $2/3$. If the given context is not found in the table, the model shortens the context until it finds an entry in the table. For example, if the given context is *ba*, the model first looks for a *ba* sub-table in the second-order ($k=2$) table. If it’s not found, it shortens the context to *a*, and looks for a table for *a* in the first-order ($k=1$) table. Finding that entry, it reports that the most likely next character is *b*, with a probability of $2/7$. If the context is not found in any of the higher order tables, the model falls back to the $k=0$ table, which simply reports the most commonly occurring character. The $k=-1$ table is used for characters that have not been seen during training, and applies to compression, not prediction, since a prediction algorithm will not predict states (or characters) it has never seen before. Arithmetic coding [64] is used to build the tables and to calculate the probabilities.

2.8 Summary

This chapter began with a survey of machine learning and temporal-data mining. It then reviewed the related work in location- and temporal-prediction. Mobility is usually represented as a sequence of locations and a discrete sequence prediction algorithm predicts the next location in the sequence. Begleiter et al. applied similar algorithms to music prediction, which uses a representation that includes notes, their starting times and durations. They found that the Prediction-by-Partial Match algorithm (PPM), a variant of a Markov model, could successfully predict music. This research supports this dissertation’s method of representing human mobility as a set of timestamps and locations and applying the PPM algorithm for prediction.

Chapter 3 Predicting Future Locations

This dissertation explores two problems: that of predicting future locations and that of predicting future times at a given location. This chapter focuses on the first problem, that of predicting where someone will be at a given time. The problem of predicting when someone will be at a given location is covered in Chapter 4.

This chapter begins with the problem statement. Section 3.2 then describes the experimental setup, beginning with the data and their preprocessing, and ending with a description of the prediction model. The chapter concludes with the results and analysis of the location-prediction experiments.

3.1 Location-Prediction Problem Statement

If one can assume continuous (or evenly sampled) location detection, someone's paths can be represented as a sequence of location symbols, e.g., *aabbbbccccddd*. Algorithms that predict the next symbol in a sequence can be applied to predict the next location symbol. These algorithms could potentially be applied to predict the next n symbols, thereby predicting n units of time into the future.

If the location tracking is not continuous, there are two choices for representing sparse location data. The first option is to continue to define a path as a sequence of location symbols as above and define an additional symbol to represent unknown locations. In the example above, a question mark could be used to represent times when the user's location was unknown:

aabbbbccc???ddd???aa. The potential problem with this approach is that, for *very* sparse data, the popularity of the unknown state outweighs the known locations in a probabilistic model.

The second option is to timestamp each change in location. The data are then represented as a series of (time, location) pairs: $\{(t_0, l_0), (t_1, l_1), (t_2, l_2), \dots (t_n, l_n)\}$. The location-prediction problem is then stated as follows: Given historical data of $\{(t_0, l_0), (t_1, l_1), (t_2, l_2), \dots (t_n, l_n)\}$, and given a future time, t_{n+m} , where $0 \leq m$, predict l_{n+m} , the location at time $n+m$ in the future.

3.2 The Data—freshmen at UCSD

As part of the Wireless Topology Discovery (WTD) project at UCSD, researchers issued PDAs to 300 freshmen and collected Wi-Fi access traces [36]. The data consist of two parts: one file containing local coordinates of access points with known locations and one file with trace data. The data were collected during an 11-week trace period from 22 September 2002 to 8 December 2002.

While a student's device was powered on, WTD sampled and recorded the following information every 20 seconds for all access points (APs) that it could sense across all frequencies—not just the access point the wireless card was associated with at the time.

USER_ID:	Unique identifier assigned to the user
SAMPLE_TIME:	The time the sample was taken by the WTD software
AP_ID:	Unique identifier assigned to the detected AP
SIG_STRENGTH:	Strength of AP signals received by device
AC_POWER:	Whether the device used AC power (1) or battery (0)
ASSOCIATED:	Whether the device was associated with this AP (1) or not (0)

Predicting Future Locations

Since WTD recorded the above information for all APs sensed during a sample, if a device sensed three APs in one sample, there were three entries recorded for that sample (which differ only in the AP detected, signal strength and associated flag).

The following figure shows a small portion of the original dataset. The following paragraphs discuss the meaning and use of each field.

```
USER_ID, SAMPLE_TIME, AP_ID, SIG_STRENGTH, AC_POWER, ASSOCIATED
123, 09-22, 00:00:00, 359, 8, 0, 0
123, 09-22, 00:00:00, 363, 5, 0, 0
123, 09-22, 00:00:00, 365, 11, 0, 1
191, 09-22, 00:00:00, 355, 31, 0, 1
101, 09-22, 00:00:00, 353, 8, 1, 0
101, 09-22, 00:00:00, 362, 30, 1, 1
129, 09-22, 00:00:00, 369, 31, 0, 1
156, 09-22, 00:00:00, 360, 19, 1, 1
184, 09-22, 00:00:02, 352, 29, 0, 1
```

Figure 3-1 A portion of the UCSD dataset

In this study, the ASSOCIATED field is used to extract the primary records that show the user's location. It is assumed that the user's device is located closest to the associated access point (ASSOCIATED=1). The access points that were sensed but not chosen for association (ASSOCIATED=0) are used to create a list of neighboring access points—that is, access points which are in the nearby vicinity of any given access point.

The AC_POWER field for each record could be used to decide if a user is mobile at any given time. The assumption is that if AC_POWER=1, the device is plugged into a wall outlet and the user is not mobile. This dissertation does not use this field in its experiments.

The SIG_STRENGTH field could be used in various location-determination algorithms. For example, if one locates the user by using trilateration between all of the access points sensed at a

Predicting Future Locations

given time, the signal strength (SIG_STRENGTH) can be used to weight the contribution of individual access points.

The AP_ID field is simply an assigned number used to label each access point in the dataset. This dissertation uses these values as location labels, and these values are outputted by the model when predicting future locations. In a real application, the AP_ID would be mapped back to a useful location name or coordinates on a map. (The UCSD dataset included coordinates for only a subset of the access points.)

The SAMPLE_TIME field includes the time and date. The devices record their state every 20 seconds over the 11-week period when the device is on. During preprocessing, the week fragments at the beginning and end of the 11-week period were discarded to provide 10 complete weeks of samples.

The USER_ID field maps each record to a specific user. This dissertation uses no user information, but simply uses the USER_ID field to partition the logs into logs for each individual user. Predictions are done on individual users, not the entire body of users.

The devices only log data when the device is turned on. For some users, this translates into almost continuous data, even if the device is simply sitting at a desk. For example, Figure 3-2 illustrates the logs for User 016, the user with the most recorded data. The time of day runs along the x-axis and the day numbers from the 10-week time frame form the y-axis. Each location is a different color. There is no correlation between the color and the actual location, i.e., "blue" locations may not be physically close to "green" locations. The location legend along the right side of the graph shows the colors used and the number of locations visited. In this case, User 016 visited 18 different locations (access points).

Predicting Future Locations

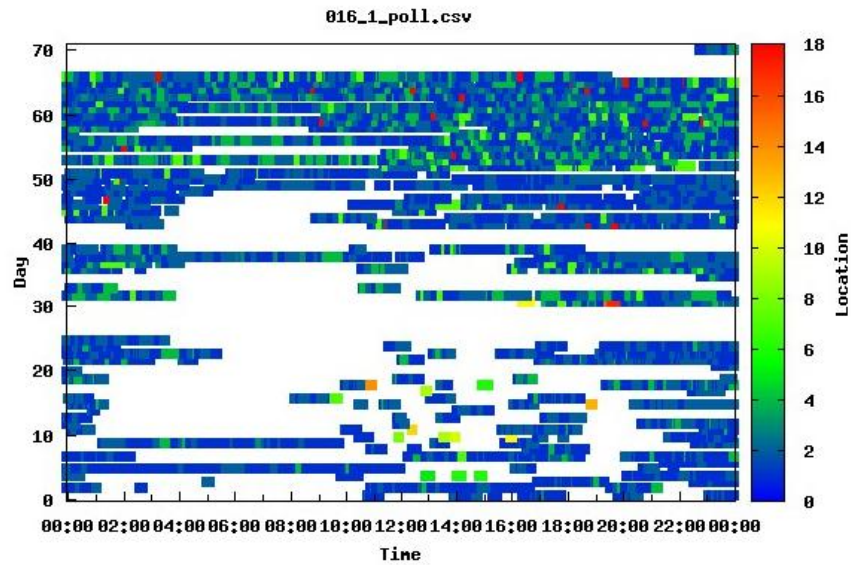


Figure 3-2 The largest user data file (User 016)

Other users barely used the device, such as the example shown in Figure 3-3, where User 130 used his device only once during the 11-week study.

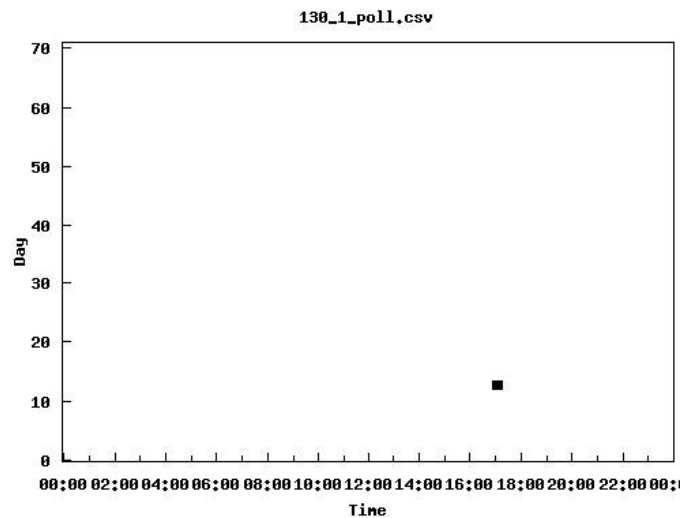


Figure 3-3 The smallest user data file.

The ping-pong effect occurs when a stationary device quickly switches its association between two different access points, which can be misinterpreted as movement. Initial preprocessing to

remove ping-pong effects is described in Appendix B and was later deemed unnecessary as ping-pong effects can be negated by increasing the time sample size and using the most probable access point in each time sample.

Much of the previously discussed related work in location-prediction assumes that the data consist of regular samples of someone's locations which form trajectories or paths, which are then extrapolated for prediction. The UCSD data are sparse and discontinuous, even for the users with the most data (e.g. Figure 3-2). If the end of a *path* is defined as a break of ten minutes or more between recorded locations, the average path length is only 1.47 locations. These data are too sparse for prediction by trajectories or paths, which confirms that this problem is a discrete sequence prediction problem.

3.2.1 Preprocessing

The raw data from the WTD experiment stored all logs from all users into one file. The raw data were divided into individual files for each user. Records that had the same time and date stamp were compared and the record with the associated access point was retained and any other records (usually of sensed, nonassociated access points) were discarded. Contiguous records were combined, where contiguous is defined as records with the same user number, the same access point and where the starting time was within one minute of the previous record's starting time. (Recall that polls are approximately 20 seconds apart.) The duration of each session was calculated and included in the output. These combined sessions were not allowed to run over a one-day boundary.

A list of neighboring access points was built. For each associated access point, a list was created of all of the access points that were sensed at the same time. Later, this list was used to see if

location-prediction was improved by expanding the predicted location to include neighboring access points.

Two different views of the data were created to represent different aspects of location. The first is movement data, which samples frequently to represent the information that would be recorded by an always-on mobile device. The second is destination data, which represents the locations that a user might disclose using a social networking application. These locations are the *significant locations* in someone's day. Significant locations are defined by Ashbrook and Starner as those places where someone spends at least 10 minutes [19]. People have few significant locations. The five users in Ashbrook and Starner's study [11] had six to eleven significant locations. Song et al. found that people spend 60% of their time in their two most common locations and 80% in the top ten [65]. Throughout the rest of this paper, the 1-minute movement data are labeled "MoveLoc" (for "Movement Locations") and the 10-minute destination data are labeled as "SigLoc" (for "Significant Locations") as a reminder of the meaning behind the selection of these sampling rates. Later, as explained in Section 3.4.1, the data are divided into 5-week training periods. Figure 3-4 shows the distribution of the number of unique locations for the 1-minute and 10-minute datasets. The number of locations drops significantly, from a maximum of 123 for the MoveLoc (1-minute) data to a maximum of 30 for the SigLoc (10-minute) data.

3.2.2 The MoveLoc (1-minute) Dataset

The timestamps for starting time in the MoveLoc dataset are rounded to the closest preceding minute. The MoveLoc dataset includes as much movement data as possible, so all sessions, even those with durations of less than 20 seconds, are included. If more than one session occurred in

Predicting Future Locations

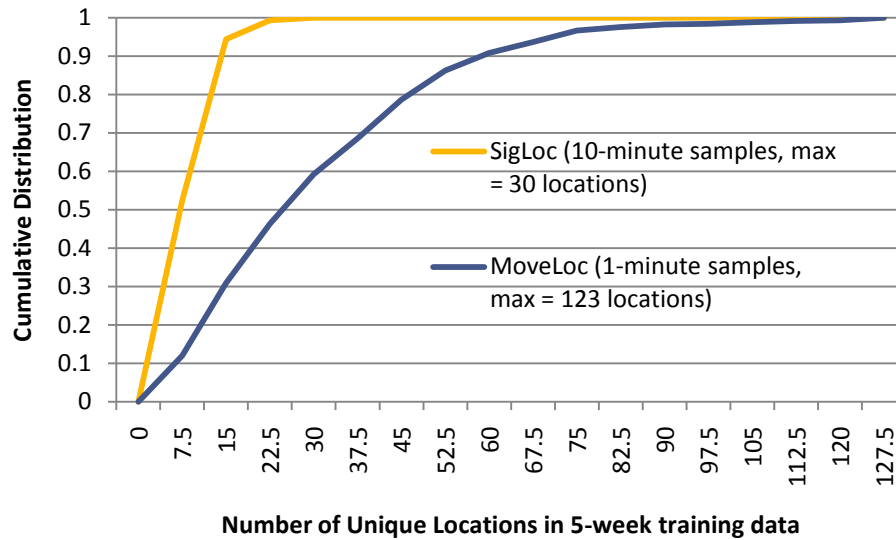


Figure 3-4 Cumulative distribution of the number of unique locations.

the same 1-minute window, then the session with the longest duration was used for that 1-minute timeslot and the others were discarded. This pre-processing also removes short ping-pong episodes, where a stationary device hops between two access points for only a few seconds.

3.2.3 The SigLoc (10-minute) Dataset

The SigLoc dataset is comprised of places where users spent at least 10 minutes, their significant locations. The starting times of the all sessions with durations over or equal to 10 minutes are rounded down to the closest 10 minutes. The SigLoc data for User 003 is shown in Figure 3-6. In comparison with Figure 3-5, one can observe that the number of locations dropped from 30 to 20 and the transitions between locations have been removed. (Please note that the colors assigned to various locations are not the same for the two figures.)

Predicting Future Locations

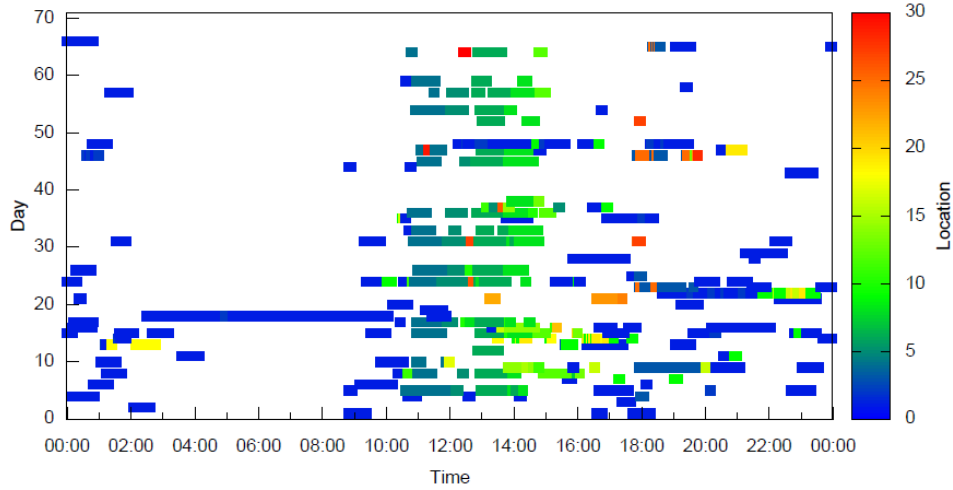


Figure 3-5 MoveLoc data (1-minute or less) for User 003

Due to the large number of possible timeslot states, times and locations are encoded as 16-bit symbols, with one range for times and another for locations. It is not necessary to restrict the symbols to a given range; however, it is useful for validating the results returned by the model. This 16-bit format was used for the Markov model. A second, human-readable, format was used for the SEQ temporal-prediction model which will be introduced in Chapter 4.

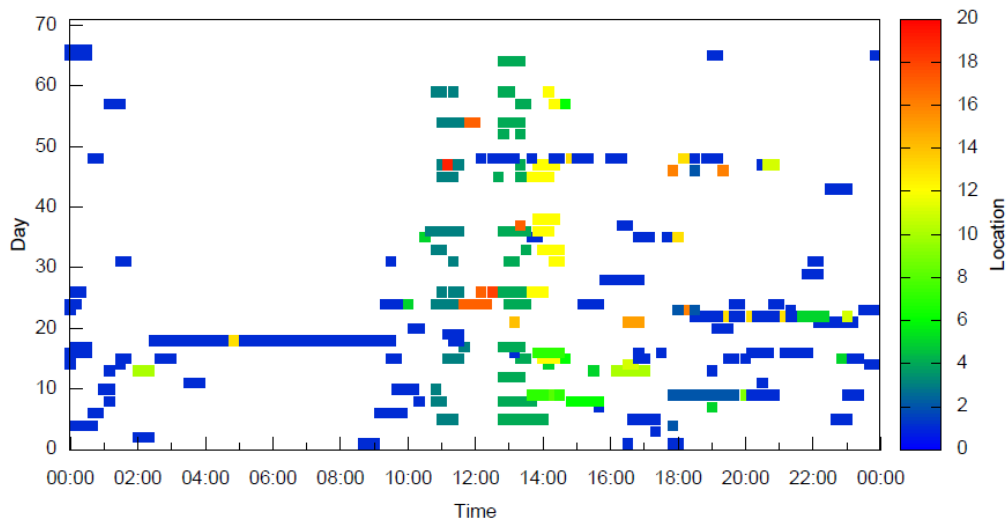


Figure 3-6 SigLoc data (10-minute) for User 003

3.3 The Experiment

A PPM implementation of a variable-order Markov model was used for the prediction model. Source code from [66] was used as the basis of this implementation. The purpose of the original code was to use the PPM-C variant of the PPM algorithm for data compression. Data compression is a three-step process. First, the model is trained on a body of historical data, and then it calculates the probabilities for each symbol or set of symbols. Finally, these probabilities are used by the encoder to use the least number of bits for the most common symbols to achieve optimal compression. Prediction models omit the final encoding step. Instead, the probabilities of the trained model are used to produce predictions. The output of the model can be either cumulative results expressed in XML or individual results in a verbose mode. Various metrics, such as a printout of the model after training, can also be produced.

3.3.1 Determining the Amount of Training Data

As mentioned previously, our selection of the PPM model was influenced by the previous use of the PPM model to predict music [62], where the authors use the log-loss function to measure the algorithm's success. This dissertation applied the log-loss function to calculate the amount of training data needed for future location-prediction. Given an alphabet, Σ , an algorithm is trained on a sequence $q_1^n = q_1 q_2 q_3 \cdots q_n$, where $q_i \in \Sigma$. Once the model is trained using the training sequence, q_1^n , the average log-loss of a test sequence, $x_0^T = x_0 x_1 x_2 \cdots x_T$, is given by

$$l(\hat{P}, x_0^T) = -\frac{1}{T} \left[\sum_{i=1}^T \log_2 \hat{P}(x_i | x_0 x_1 \cdots x_{i-1}) + \log_2 \hat{P}(x_0) \right] \quad (3-1)$$

Predicting Future Locations

The log-loss equation relates to the number of bits required to compress a string. Minimizing the average log-loss corresponds to maximizing the probability assignment for the test sequence. In other words, a lower value for average log-loss indicates a better predictor.

The UCSD data spans 11 weeks. The partial week fragments at the beginning and the end of the dataset were dropped to provide 10 full weeks of data. Six users were selected from the dataset.

The most-mobile users, those associated with the largest number of access points (72, 67 and 64), were selected, along with three fairly mobile users who each associated with 33 access points.

The most-mobile users were included because they are the worst case for predictability, as opposed to the users who rarely, if ever, visited more than a few access points, thus making it easier to predict their location. The fairly mobile users were selected on the assumption that they are representative of an average, yet mobile, user. A series of tests were run using various subsets of the data for training and testing. The model was trained on the subset of the data selected for training and the average log-loss was calculated for the week of test data (either Week 6 or Week 10). Lower values of log-loss indicate more compression, which correlates to better predictability.

Test results showed that training for five weeks, and then testing on the subsequent week, produced the best results. This is indicated by the dark bars in Figure 3-7. All of the subsequent experiments on the PPM model used five weeks of historical data for training and tested on the following week.

3.3.2 Training and Testing the Model

The data are represented as sequences of $(time, location)$ pairs. Each user's data are in separate files where each line has the format:

$$date; \{time_0, location_0, time_1, location_1, \dots, time_n, location_n\}$$

Ten weeks of data are partitioned into test sets, or runs, where each run consists of five weeks of training data and a corresponding week of testing data. For example, one run could consist of User 003's data from Weeks 2–6 for training and User 003's data from Week 7 for testing.

Initially, the model was trained and tested for first-order and third-order. The first-order test used each $time_x$ as input and counted how often the model produced the correct prediction of $location_x$ as output. The third-order test consisted of giving $time_{x-1}, location_{x-1}, time_x$ as

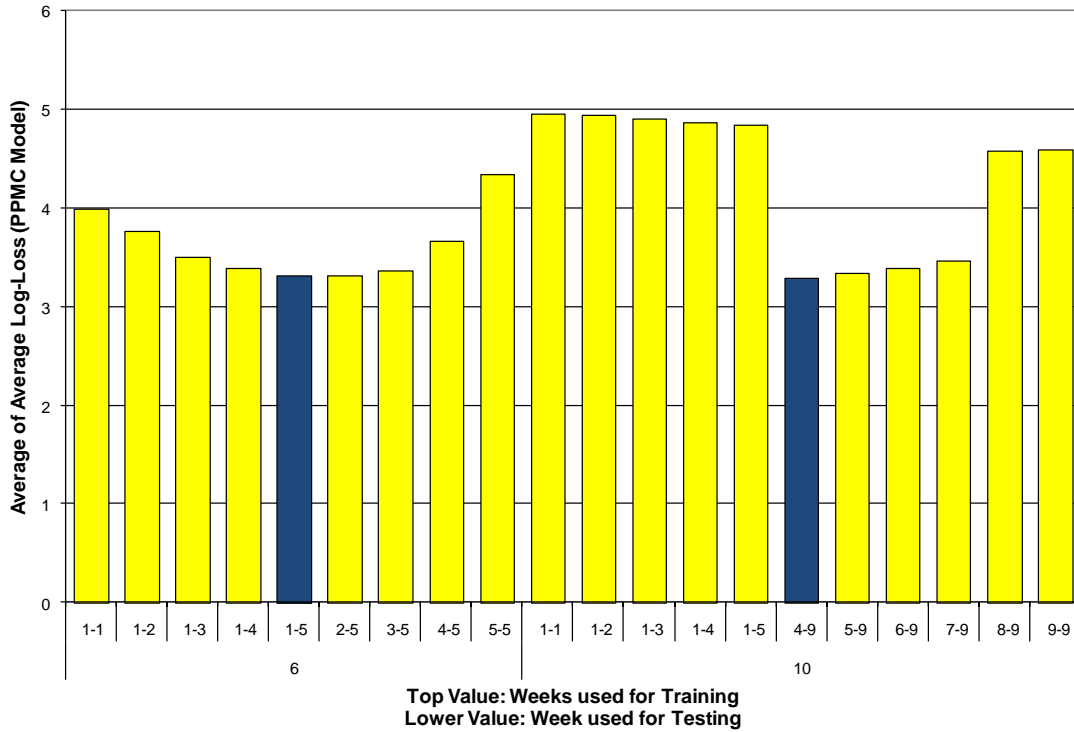


Figure 3-7 Average log-loss for different training and testing schemes (for sequences containing only locations) The dark bars indicate the best number of training weeks for the given testing week.

input and seeing how often the model produced the correct prediction of $location_x$. The first-order model asks the question, “Where is the user at $time_x$?” The third-order model asks the question, “Where is the user at $time_x$, if I also state that he was at $location_{x-1}$ at $time_{x-1}$? An example could be “Where is Bob at 8:04 am if I know that he is at the office at 8:03 am?” This does not seem to be a very useful question. The results of the third-order model are reported in [18]. The rest of this chapter will focus on the first-order model.

A portion of a PPM model trained for the first order is shown in Figure 3-8.

Context times	Locations seen at the context time	Number of times each location seen at given context time
Symbol: 09:30, counts: 5	Symbol: 0x24b4, counts: 5	
Symbol: 11:00, counts: 5	Symbol: 0x2835, counts: 5	
Symbol: 14:20, counts: 5	Symbol: 0x23d7, counts: 3	
	Symbol: 0x2561, counts: 1	
	Symbol: 0x2551, counts: 1	
Symbol: 14:30, counts: 5	Symbol: 0x2551, counts: 3	
	Symbol: 0x23d7, counts: 2	

Figure 3-8 Portion of the Markov model trained for location-prediction (User 003, weeks 1-5)

Once the model is trained on five weeks of data, the following week is used to test. The test data are parsed into individual $(time, location)$ pairs. For each pair, the time is fed into the trained model as the input (context), and the model returns a list of predicted locations with their probabilities, sorted with the highest probability predictions at the top of the list. The model then compares each of the returned predictions against the correct answer, the test location. The

Predicting Future Locations

predictions with the highest probabilities are checked first. The lower-probability predictions are checked next. An example is shown in Figure 3-9.

If none of the returned predictions are correct, the neighboring access points of the incorrect predictions are checked. Neighboring access points were determined during preprocessing when a list was created of all of the access points that were sensed at the same time as each associated access point. These access points are likely to be in the same building. For example, if, in our test, the correct answer was access point #72, and the Markov model returned a prediction of access point #63, the model would check to see if access point #63 was a neighbor of #72, and likely to be in the same proximity.

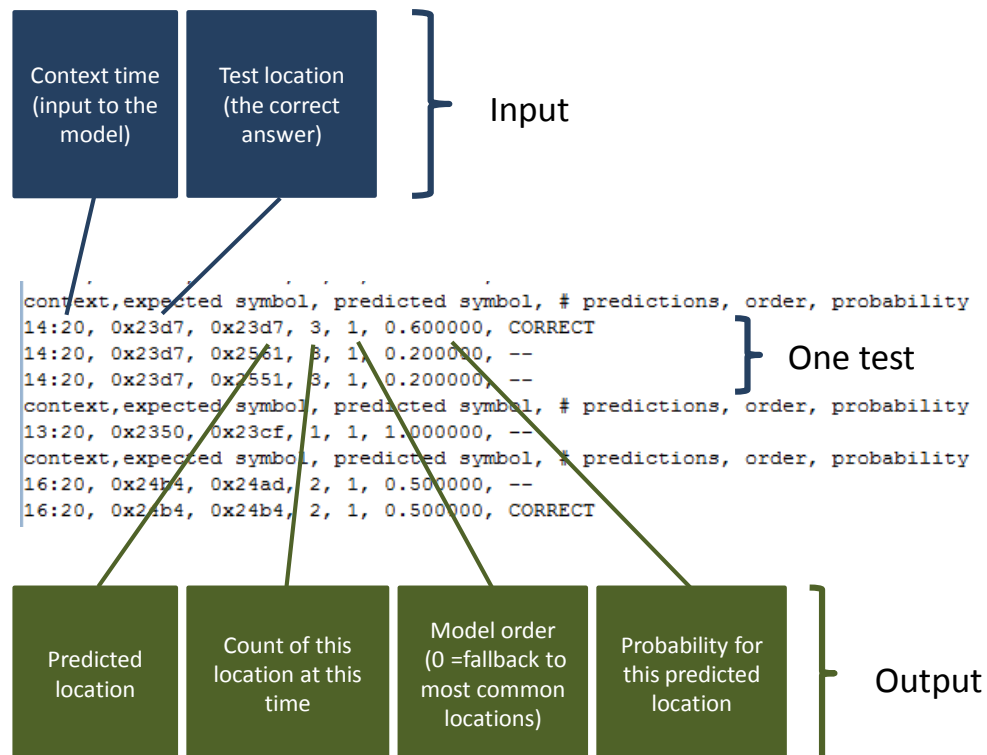


Figure 3-9 Example output from the Markov model location predictor (User 003, trained on weeks 1-5 and tested on week 6). This type of output is produced when the “verbose” flag is set (-v).

Predicting Future Locations

If the input time was not found at the first order, the PPM algorithm falls back to the zeroth order and returns a list of the most likely locations, regardless of the input time. This means that the model has no information about the time that was entered as the context, so it simply returns the most popular locations.

3.3.3 Results

The cumulative results of using the Markov model to predict future locations are shown in Figure 3-10. The lowest results are for the MoveLoc dataset of 1-minute timeslots. This is understandable because the Markov model does not account for any variability in time. For

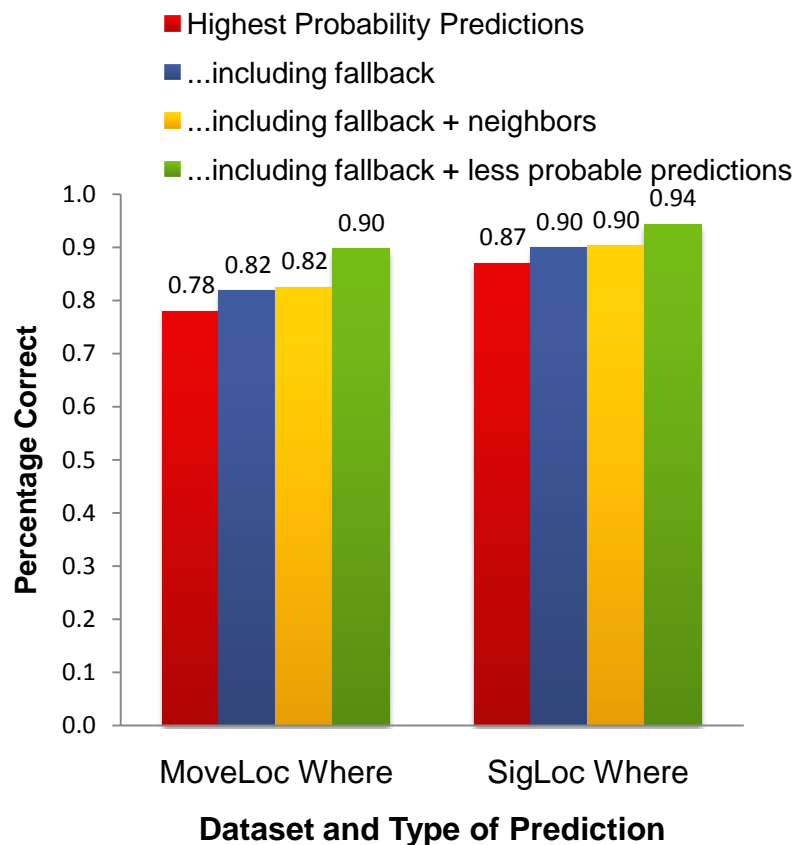


Figure 3-10 Aggregate results for predicting future locations using the Markov model

example, if the user arrives at location x at 8:00 am, and the next day arrives at 8:01 am, these two pairs are recorded as separate states in the Markov model, which reduces their individual probabilities. In the SigLoc dataset of 10-minute timeslots, these two records are counted as the same 8:00 am state, increasing its probability. The increased size of the timeslot compensates for variability in arrival times.

The best results, 94% for the SigLoc 10-minute data, are achieved when all of the returned predictions are checked against the correct answer, regardless of their probabilities. This complete list of predictions includes all of the locations seen at the given timeslot. The median number of possible predictions returned for each test is 2, with a maximum of 13 for the MoveLoc 1-minute data and 9 for the SigLoc 10-minute data. For both datasets, results improve when fallback predictions are included. Results improve further when neighboring access points are used as predictions.

Figure 3-11 and Figure 3-12 show the normalized distribution of the prediction results for each dataset. The following section will show the cumulative distribution of the results and compare them to random guesses and ideal results.

Comparing the Markov Model to Random Guessing

This section compares the results of the Markov Model for prediction location with other approaches in order to show that the Markov Model performs better than random guesses or educated guesses. Figure 3-13 is a normalized distribution of prediction rates for various prediction schemes for the 1-minute data. The corresponding graph for the 10-minute data is in Figure 3-14. The ‘ideal’ line is included as a comparison to the perfect predictor. The ideal

Predicting Future Locations

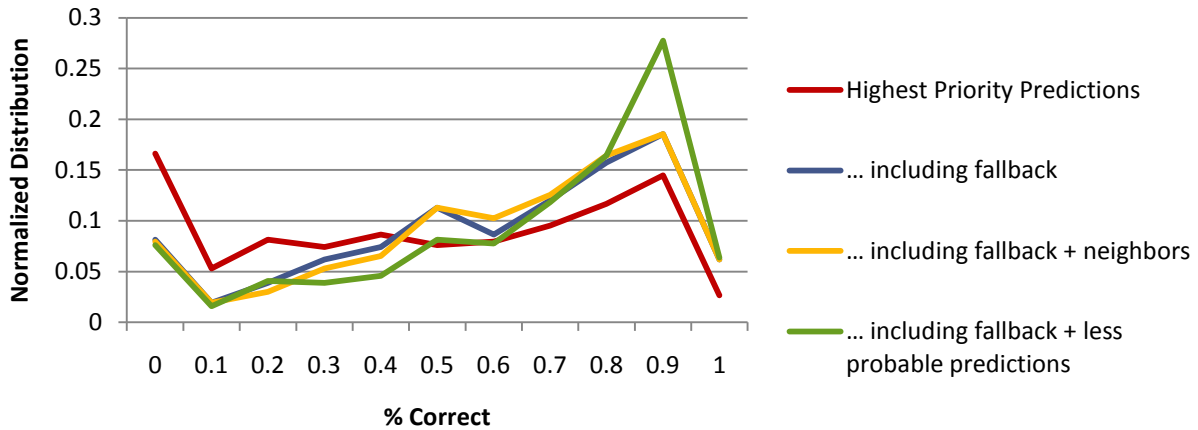


Figure 3-11 Normalized distribution of Markov model predictions for the 1-minute dataset

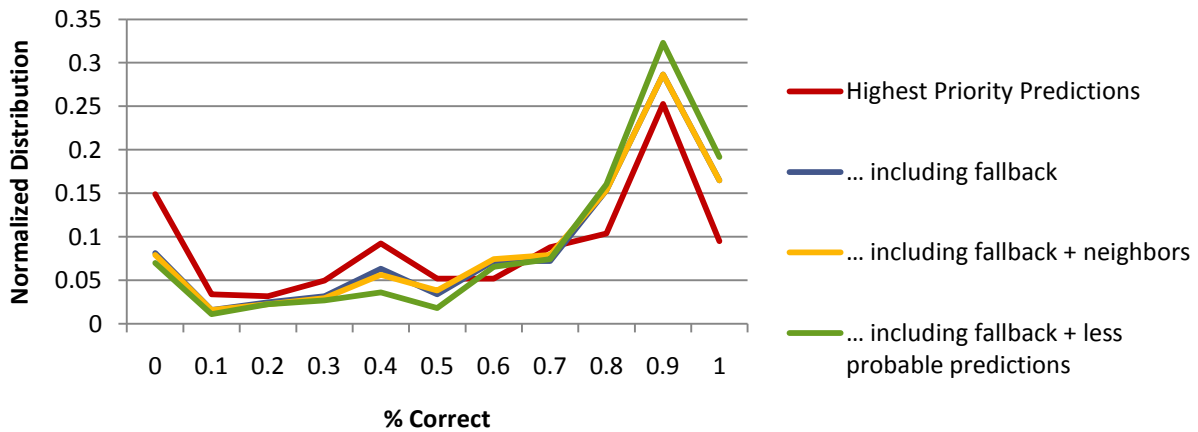


Figure 3-12 Normalized distribution of Markov model predictions for the 10-minute dataset

result stays at 0 until it steps to 1 at 100%. The closer the other curves are to the ideal, the better their prediction scheme.

Each test has a fixed number of possible locations for the predictor to choose from, the locations in the training data. A random predictor has a prediction rate of $1/n$, where n is the number of unique locations in the training dataset. The results of random guesses on the current datasets are shown on the top line.

Predicting Future Locations

Another prediction scheme is the fallback predictor which uses some knowledge about the training data to always predict the most common location. The results of the fallback predictor are shown in the figures. Even though the results are much better than random guessing and much closer to the results of the Markov Model, it is not a very useful application, since it will always answer every prediction question with the same answer, that of the most common location.

Two sets of results from the Markov model are shown. The upper line shows the prediction rate when only the highest probability prediction is considered. It includes the fallback cases, where there is no temporal context. The lower line is based on the results from the Markov model when all predictions are tested to see if they are correct. These predictions include the highest probability predictions, the less likely predictions and the fallback cases.

Figure 3-14 shows similar results for the SigLoc data.

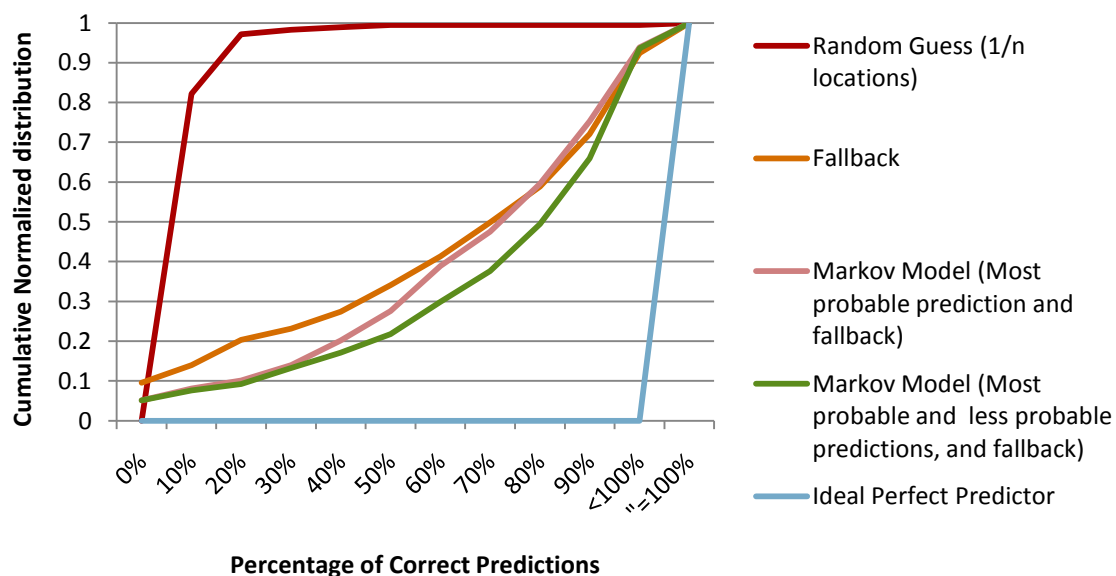


Figure 3-13 Comparison of different prediction schemes on the MoveLoc 1-minute data

Predicting Future Locations

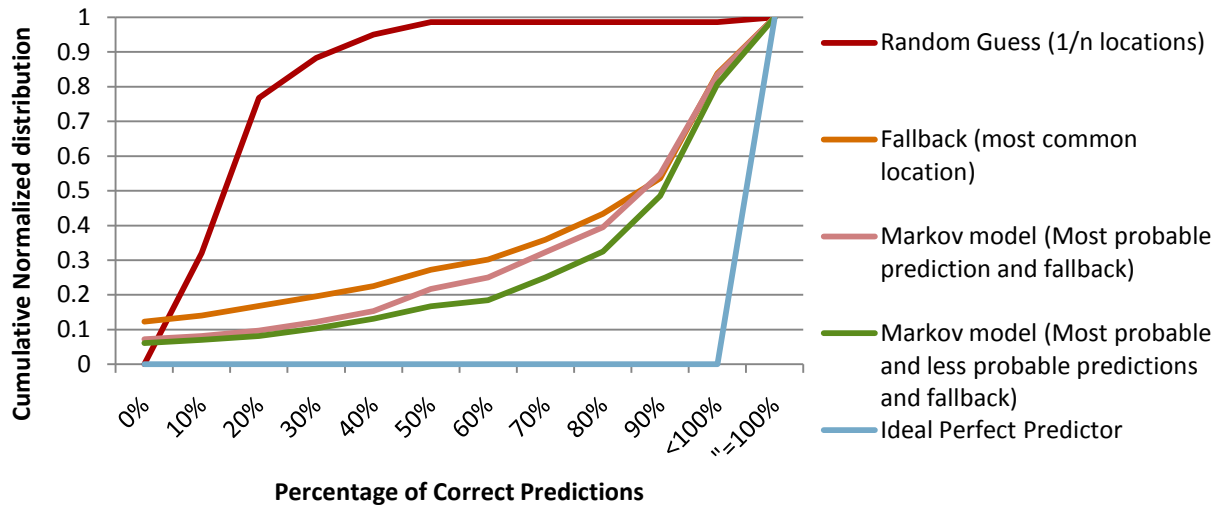


Figure 3-14 Comparison of different prediction schemes on the SigLoc 10-minute data

Figure 3-13 and Figure 3-14 show that the Markov model performs much better than random guessing. Fallback prediction based on guessing the most common location yields results that are close to the same as the Markov model, especially when the Markov model only considers the single highest-probability locations. When the Markov model considers all of the possible locations concerned, its results improve. This reveals the need for careful application design. A useful application needs to communicate not just the highest probability prediction, but also those that may be less probable. An example would be a result that says, “Bob is most likely to be in the office (60%), but he may be in the library (40%).” The fact that Bob spends a majority of his time in the office may not be the key piece of information that makes the application truly context-aware and useful. Imagine that you arrive at Bob’s office hoping to meet with him, but his office is empty. The answer that you need from the predictive model is not the most likely result that he should be in his office but the less likely prediction that he may be at the library.

3.3.4 Entropy of Movement Patterns

Song et al. [65] apply entropy to calculate the potential upper limit for the predictability of cell phone users' next base station (their location). They represent their data as a sequence of cell phone tower symbols and show that people's locations are potentially 93% predictable. This section reviews their results and shows that applying similar entropy calculations to data represented as *(timestamp, location)* pairs yields similar results. Their definitions of entropy are as follows.

First, they define random entropy,

$$S_i^{rand} \equiv \log_2 N_i \quad (3-2)$$

Where N_i is the number of distinct locations visited by user i . This value reflects the number of different places a user goes, without consideration of which places get visited the most often or the order in which they are visited.

Their second definition of entropy is the temporal-uncorrelated entropy,

$$S_i^{unc} \equiv - \sum_{j=1}^N p(j) \log_2 p(j) \quad (3-3)$$

where $p(j)$ is the probability of location j in the historical data,

$$p(j) = \frac{\# \text{ occurrences of location } j}{\# \text{ occurrences of all locations.}} \quad (3-4)$$

This value of entropy factors in the popularity of each location, but not the order.

Predicting Future Locations

The final definition of entropy is the actual entropy, which factors in not only the popularity of each location, but also considers the sequence in which the locations are visited.

$$S_i \equiv - \sum_{T' \subset T} P(T') \log_2 P(T') \quad (3-5)$$

where T' is a sub-sequence in the trajectory.

Their data representation consisted of a series of base-station symbols (e.g. ‘AABBCC’) with a place holder inserted (‘?’) when a user’s basestation was unknown. They sampled every hour and discarded users who did not have data for at least 80% of the time. A Lempel-Ziv estimator was used to calculate actual entropy, S .

If people’s trajectories were completely predictable, their actual entropy, S , would be 0. If they were completely random, their actual entropy, S , would be the same as the random entropy, S^{rand} , meaning that there is a $1/N$ chance of predicting their next location correctly. When the authors calculated the three definitions of entropy over all of their users, they found that actual entropy is quite low, meaning that people are very predictable. A portion of their results are shown in Figure 3-15. Their additional calculations showed that “a significant share of predictability is encoded in the temporal order of the visitation pattern.”

The previous definition of actual entropy does not apply to this dissertation’s representation because it does not encode the ‘temporal order’ as an ordered sequence, but as a timestamp in the *(timestamp, location)* pairs. Because of this dissertation’s encoding and prediction model, where a location is predicted given a time, the *conditional* entropy is the appropriate measure of the randomness in users’ movements. The conditional entropy is a function of locations at a given time. The conditional entropy of Y (location) given X (time) is:

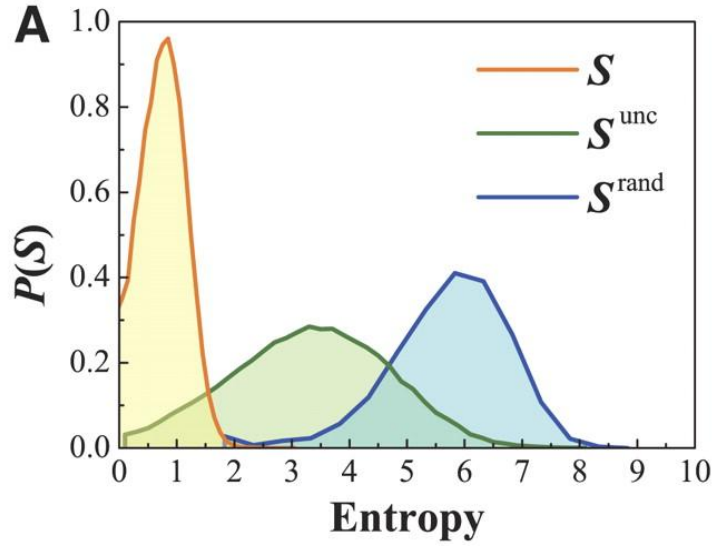


Figure 3-15 Distribution of the entropy S , the random entropy S^{rand} , and the uncorrelated entropy S^{unc} across 45,000 users
(From C. Song, Z. Qu, N. Blumm, A.-L. Barabási, Limits of Predictability in Human Mobility, Science © 2010 AAAS. Reprinted with permission from AAAS.)

$$H(Y|X) \equiv \sum_{x \in X} p(x) H(Y|X = x)$$

(3-6)

$$= - \sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log_2 p(y|x)$$

$p(x)$ is the probability of timestamp x :

$$p(x) = \frac{\# \text{ occurrences of timestamp } x}{\# \text{ occurrences of all timestamps}} \quad (3-7)$$

$p(y|x)$ is the probability of location y occurring at timestamp x :

$$p(y|x) = \frac{\# \text{ occurrences of pair (time } x, \text{ location } y)}{\# \text{ occurrences of pairs (time } x, \text{ any location)}} \quad (3-8)$$

Figure 3-16 shows the distribution of the conditional entropy and the distribution of the random entropy, as calculated for all 10 weeks for all of our users. The graph is similar to Figure 3-15.

Predicting Future Locations

The conditional entropy is lower, showing that there is less randomness, or more predictability, in our *(timeslot, location)* pairs than in randomly predicting a location from a list of the user's locations or by picking the most probable location. This validates the representation.

Experimental results shown previously in Figure 3-13 and Figure 3-14 confirm these findings.

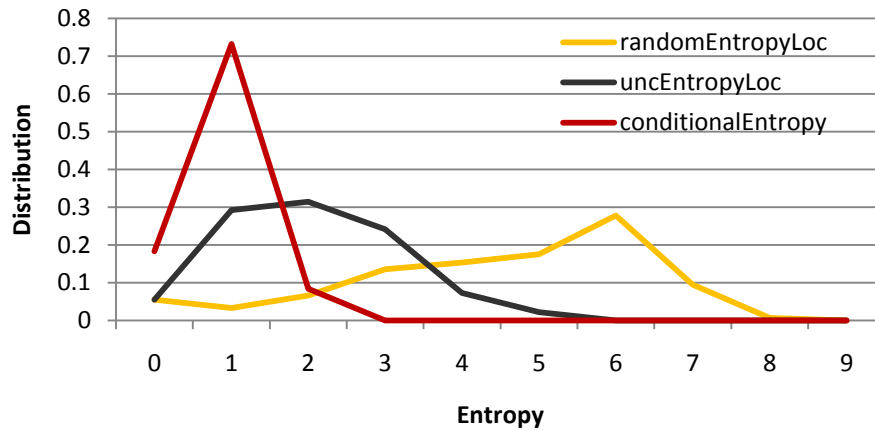


Figure 3-16 Frequency distribution of conditional entropy, uncoorelated entropy and random entropy as a function of the number of unique locations. The randomEntropyLoc is based on randomly predicting a future location. UncEntropyLoc is based on predicting the most common locations and the conditionalEntropy is based on predicting a location based on its timestamp.

3.3.5 Analysis of Predictable Data

The previous section showed that the Markov model can be used to predict future locations.

These results were collected from the entire training dataset, which consisted of 566 files for the 1-minute data which is used in this section. The purpose of this section is to determine if there are characteristics of the training data that correlate to predictability and if there is a relationship between the amount of data in the training data and the quality of the predictions. This section compares the training sets that result in prediction rates of 90% or more to the general population of training sets, in order to identify which characteristics of the historical training data result in

Predicting Future Locations

high prediction rates. There are 96 training files which resulted in prediction rates of greater than 90%.

The Markov model can be visualized as a nested table. The outer table has an entry for each unique timestamp in the training data. For each unique timestamp, there is an inner table with an entry for each unique location seen at that timestamp, along with a count of how many times it was seen. Table 3-1 shows the structure of the Markov model tables.

Table 3-1 Structure of the PPM Markov model

First Order			Zeroth Order (Fallback)	
Timestamp 1	Location 1	Count	Location 1	Count
	Location 2	Count	Location 2	Count
Timestamp 2	Location 1	Count	Location 3	Count
	Location 2	Count		
	Location 3	Count		

The zeroth order table is used for queries for which the context timestamp is not found in the first order table. As will be shown in Section 3.3.5, ideal datasets have 24-hour coverage, or at least complete coverage for the time period of interest to the application. Therefore, this analysis will focus on the first order portion of the Markov model and will not include fallback predictions in the prediction results. For the most part, only the highest probability predictions returned by the model, which corresponds to the lowest bar in the results histogram in Figure 3-10, are used in the analysis. When necessary, prediction results were expanded to include the neighbors of the highest probability location predictions.

As seen in Table 3-1, there are three dimensions that characterize the first order portion of the Markov model: the number of unique locations, the number of unique timestamps, and the size of the table. The size of the table is a function of the number of unique timestamps and the number of locations seen per timestamp. These characteristics are analyzed in the following paragraphs.

Number of Unique Locations

The graphs in Figure 3-17 show the normalized distribution of the number of unique locations in the 5-week training sets. Three distributions are shown: for the general population, for those training sets that resulted in a prediction rate greater than 90%, and for training sets that resulted in a prediction rate greater than 90% when neighboring locations were included in the predictions. Graphs are shown for both the MoveLoc 1- minute datasets and the SigLoc 10- minute datasets. The SigLoc 10-minute data are coarser and lead to smaller tables and smaller numbers of unique locations. The 10-minute threshold used to create the SigLoc data removes the transient locations that may be recorded as the user moves between significant locations. The following discussion will focus on the MoveLoc 1-minute datasets. The graph for the SigLoc 10-minute data shows similar results and is included here for completeness.

In Figure 3-17, the 90% group and the general population show the same correlation between the number of locations and the prediction rate, so the number of unique locations is not an indication of predictable data. A statistical p-test confirms that there is no statistical significance between the 90% group and the general population (p-value=0.49).

These graphs show that when the training data contain a large (over 100) number of locations, the prediction rate is 0. The large number of locations could be due to overlapping Wi-Fi access

Predicting Future Locations

points, which are each considered unique locations. A physical space that is covered by several access points could be recorded as several different unique locations in the dataset. To see if this is the case, the MoveLoc graph in Figure 3-17 includes the prediction results when the model checks if neighboring access points are the correct predictions. The third set of bars in the MoveLoc histogram graphs the number of unique locations for datasets that resulted in prediction rates of 90% or better when neighboring access points are also considered in the predictions. The rightmost three bars (which comprise 10 datasets) in the MoveLoc histogram are training data with a large number of unique locations. If these values were due to an abundance of overlapping access points, one would expect to see instances where including

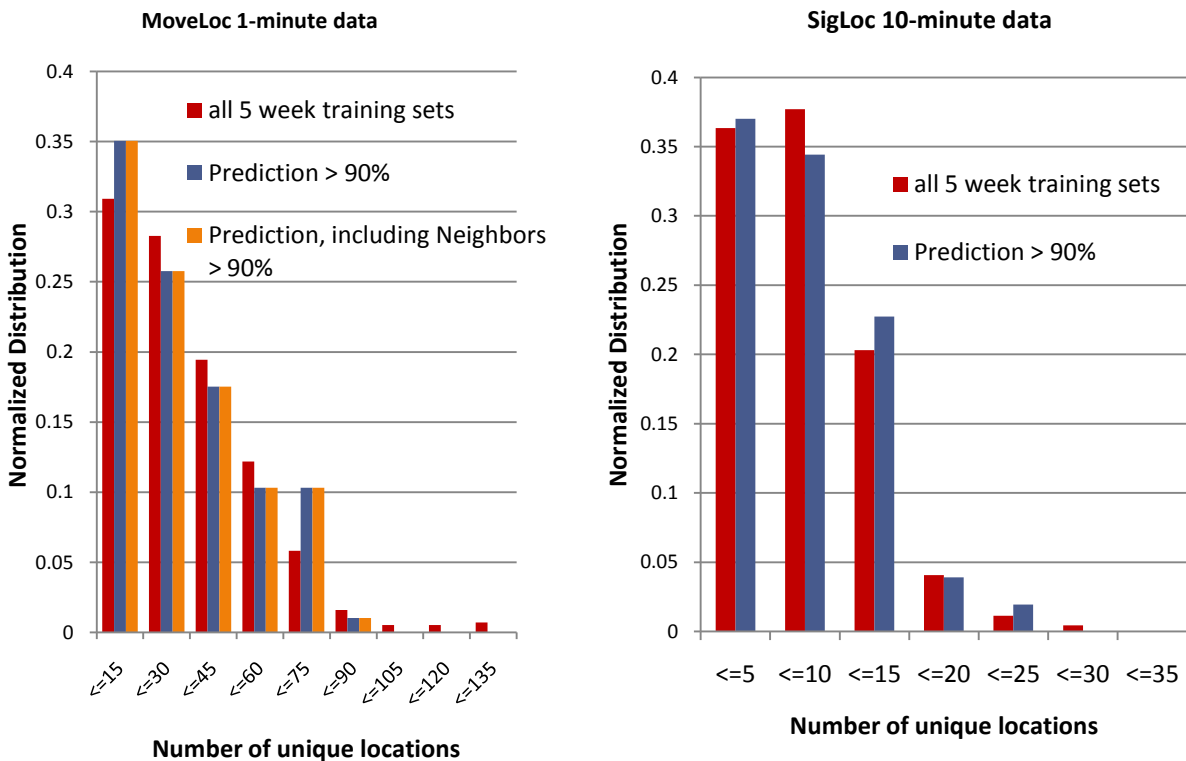


Figure 3-17 Normalized distribution of the number of unique locations in the training datasets for both the MoveLoc (1-minute) and SigLoc (10-minute) 5-week data.

neighbors significantly improves the prediction rate. Instead, the graph reveals that including neighbors increases the prediction rate only slightly. This shows that the unique locations in the training data truly are unique and not overlapping access points.

Number of Unique Timestamps

The next characteristic of the training data is the number of unique timestamps. In the nested table, the timestamps are the outer dimension, the context used to determine which of the inner nested tables to use for the prediction. The figure below compares the distribution for the entire population of training data versus the datasets that resulted in prediction rates greater than 90%. The x-axis is a count of the number of unique timestamps recorded. The maximum value for the 1-minute data is 1440, which covers every minute of a 24-hour day. The high bars on the right indicate that many of the devices were left on over a 24-hour period.

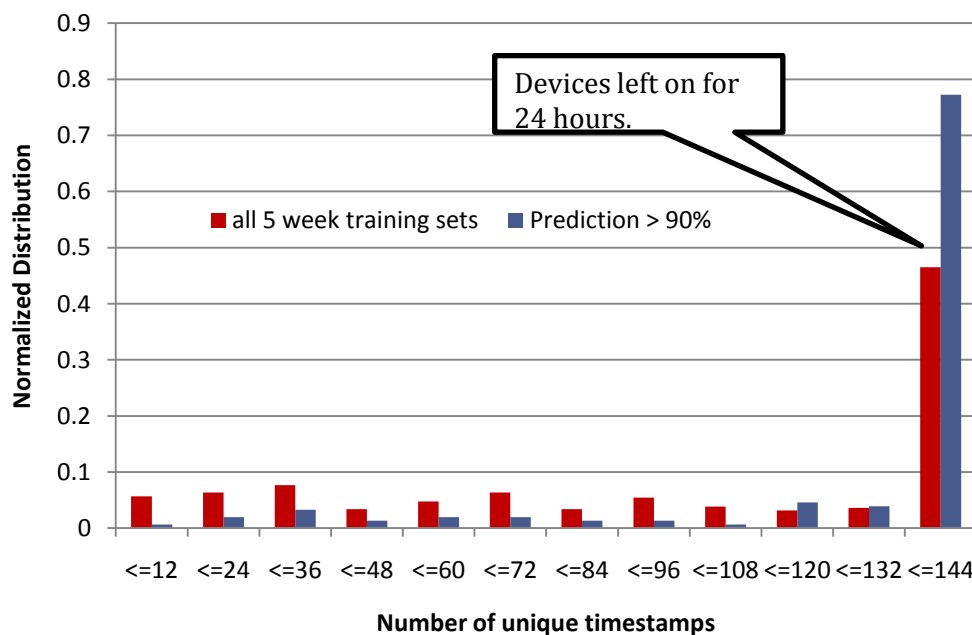


Figure 3-18 Normalized distribution of unique timestamps in the 10-minute data

Predicting Future Locations

The figure indicates that the more predictable datasets had more timestamps, or more coverage over the course of the day, but not to the level of statistical significance ($p\text{-value}=0.91$). Similar results are shown below for the SigLoc, 10-minute data.

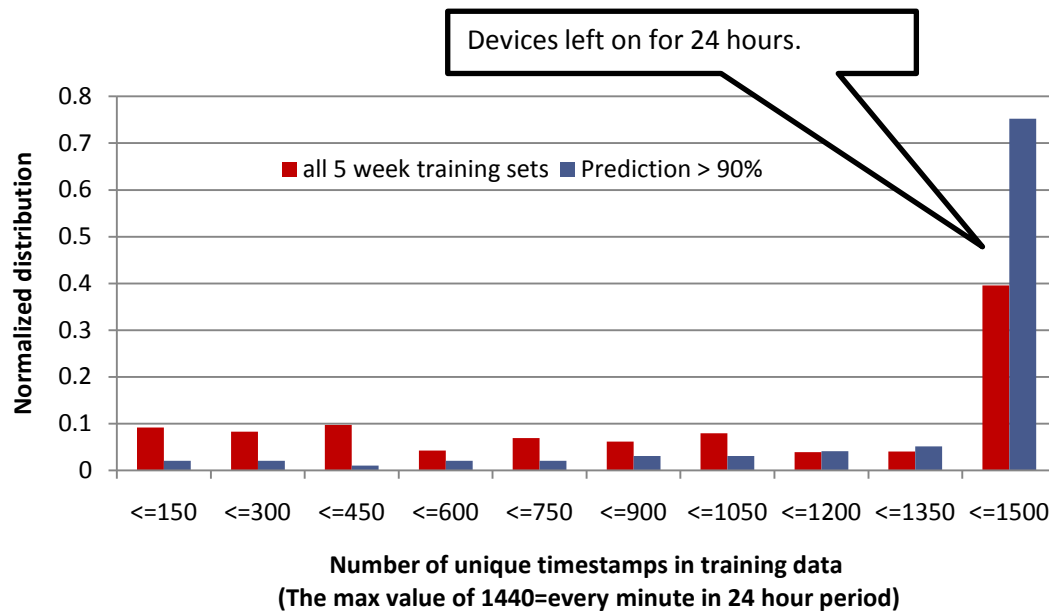


Figure 3-19 Normalized distribution of the unique timestamps in the 1-minute data

Table Size

The final characteristic of the training datasets is the table size. Table size is a function of the number of unique timestamps and the number of locations for each unique timestamp. These figures compares table size between the 90% group and the general population. The two differ at both ends of the x-axes in Figure 3-20 and Figure 3-21.

Predicting Future Locations

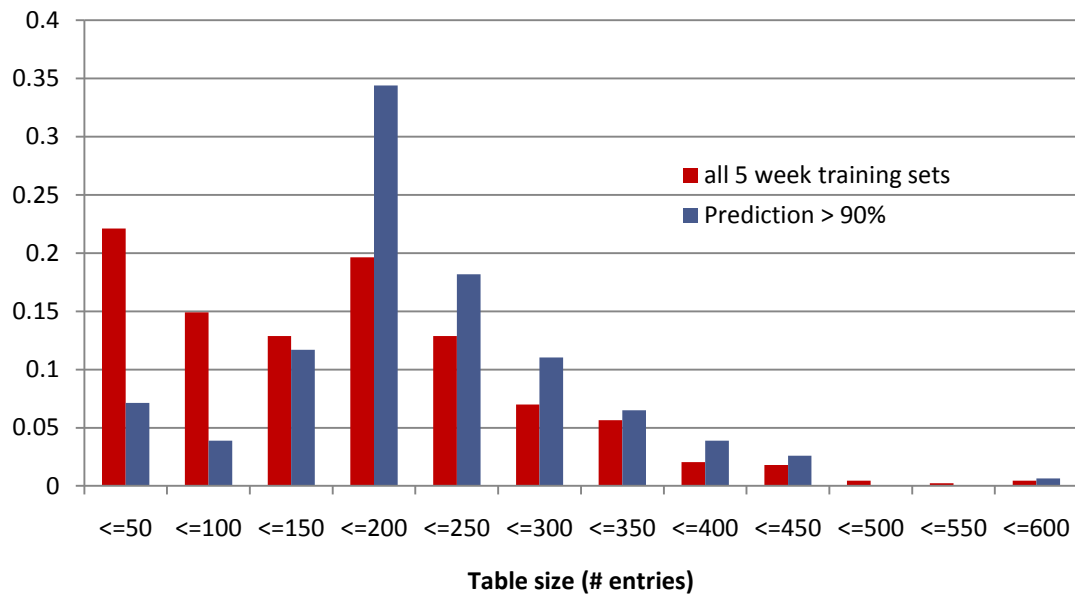


Figure 3-20 Normalized distribution of table sizes in 5-week 10-minute training data

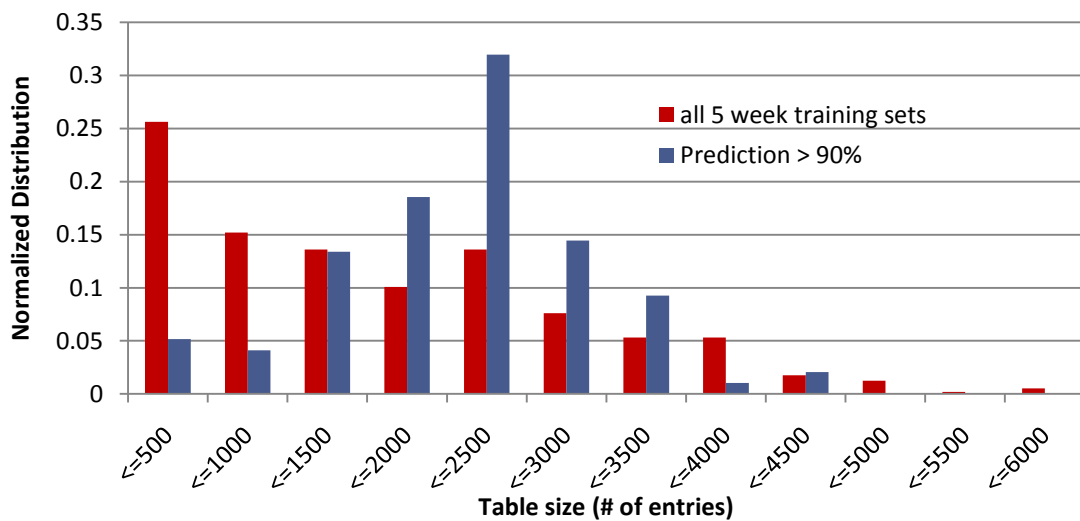


Figure 3-21 Normalized distribution of table sizes in 5-week, 1-minute training data

Predicting Future Locations

Small tables are represented at the left end of the x-axes. Small tables lack the timestamps to cover large portions of the day, so they will form poor predictors. The right side of the x-axis, which indicates large tables, also reveals a difference between the 90% group and the general population. The following section characterizes the datasets that create models with large tables with low prediction results to find the factors that make these tables large.

Large Tables

The datasets on the right side of Figure 3-21 have a table size of greater than 4200. The following figures compare these large datasets with the rest of the training data population. Out of the 567 5-week datasets, 15 of them have more than 4200 entries. All 15 had 1440 timestamps (24-hour data).

As shown below, there is no striking correlation between the number of unique locations in our ‘large table’ datasets and the rest of the population (p-value=0.48).

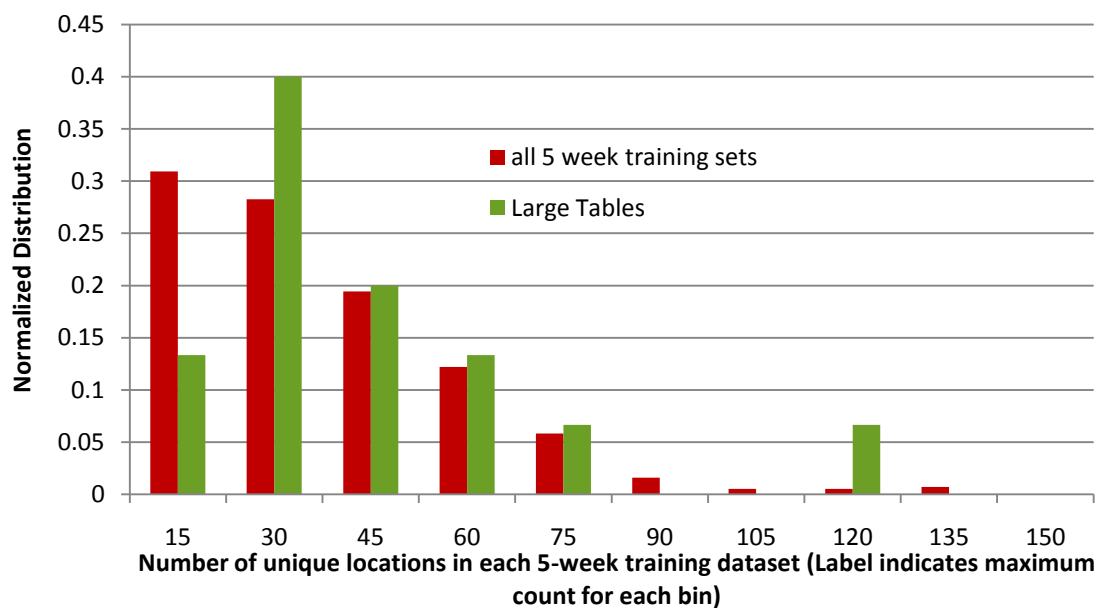


Figure 3-22 Normalized distribution of the number of unique locations in 1-minute training data

Predicting Future Locations

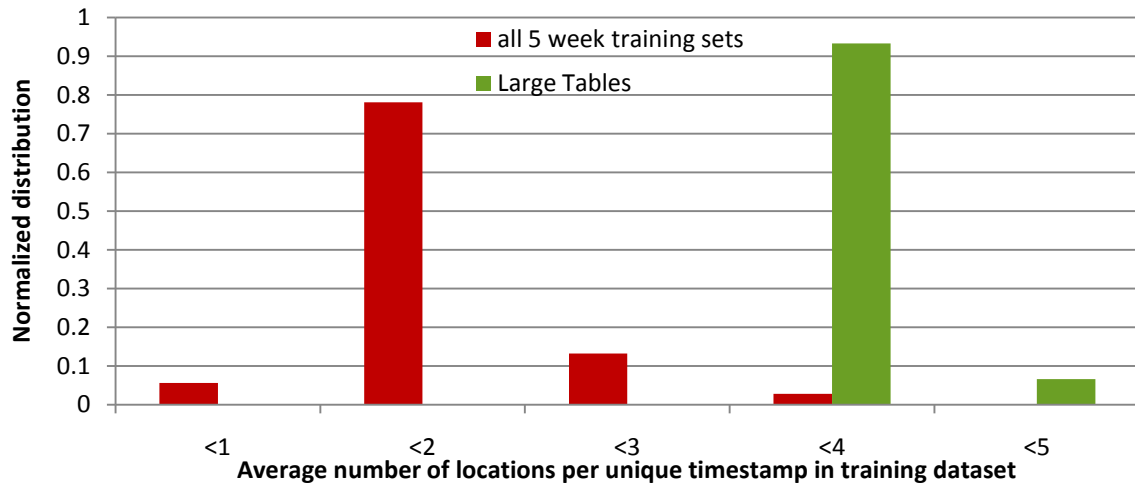


Figure 3-23 Normalized distribution of locations/timestamp (1-minute samples)

The difference between the ‘large-table’ group and the general population is evident when one compares the average number of locations per unique timestamp, as shown in Figure 3-23. The p-value of statistical comparison is 0, signifying that the mean of the large-table group is statistically different than the mean of the general population, as shown in Table 3-2. Our large-table group has a larger number of locations per timestamp. This increases the number of possible choices for the prediction and lowers the probabilities for each choice.

Table 3-2 Test of statistical significance

	All Tables	Large Tables
Sample Size	566	15
Mean (Average)	1.52	3.38
Std. Dev	.56	.33
t-test (p-test for small sample sizes)		21.8
p-value		0

A person who has been logged at many different locations at many different times is inherently less predictable, unless there is additional context that can narrow down the possible choices. In all but one instance, these datasets do not have a large (greater than 65) number of unique

locations. They are simply users who are logged at many different locations throughout the day. One user who fits this criterion is shown in Figure 3-24.

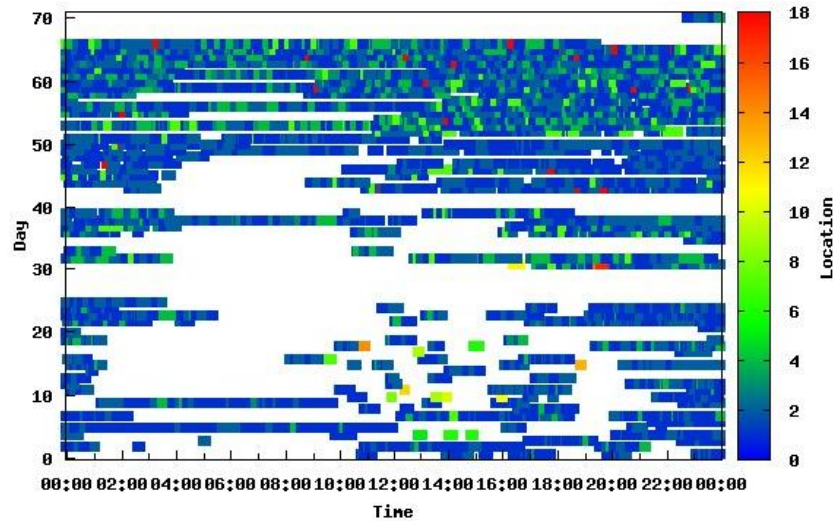


Figure 3-24 User 016 locations at 1-minute polls

Summary of characteristics of datasets with high prediction rates

The 1-minute data consists of 566 training files. Of these, 96 produced models with prediction rates greater than 90%. Analysis of the characteristics of the training datasets shows two criteria that are desirable for meeting high prediction results.

1. When using the techniques described in this chapter and a spatial resolution of a WiFi access point for locations, predictable datasets should not contain an average of more than three locations per timestamp. There are two situations that cause this characteristic. The first is a user who is moving around too randomly to be predictable. In this case, the model cannot be applied. The second case is that of a user who has changed patterns, perhaps due to a move or a new semester. In this situation, outdated information should be discarded and the model should be re-trained on less, but more recent data.

(Retraining intervals are discussed further in the Section 5.1.)

2. The time period encompassed by the training data must cover the time period of interest to the application. Given the spatial resolution of the available datasets, a model that contains only a few timestamps could have a high prediction rate if the application limits its queries to those particular timestamps and there are not too many unique locations for those timestamps. For general applications such as the test cases used here, the datasets that led to the best predictions had entries for the entire 24-hour day.

3.3.6 Drawbacks of the Markov Model for Predicting Location

The Markov model demonstrated in this chapter provides a simple yet effective method for predicting future locations. Its drawbacks become apparent when considering the design decisions required for a realistically useful application of the model.

Recent data only become incorporated into the model when the model is retrained. An actual application would need to determine how often the model is retrained. If the model is retrained often, it is more up-to-date but it is also using computational resources and could appear to be running slowly to the user, as it is may be off-line during retraining. If the model is not retrained often, then recent data are not incorporated, and the model could miss some predictions. These “misses” would be especially acute during times when someone’s routine changed radically, such as a vacation or the beginning of a new semester. The optimal retraining period would keep the model up-to-date without bogging down the device with constant retraining.

Changes in routine, such as vacations or new jobs, are an interesting problem in location models. Irregular data, such as an out-of-town trip, affects the predictions of the model by giving the model more choices for each prediction. The Markov model summarizes counts of locations seen in the past. Counts for new locations are added to the list of the counts for existing

locations. The counts for the existing, usual locations would likely outweigh the new locations enough that the model would continue to predict the usual, routine locations. The downside of this approach is that when someone's routine changes permanently, e.g., at the start of a new semester, the model will not reflect the new routine until the counts of the new locations approach and eventually outnumber the counts for the previous locations. This effect could be minimized by editing the historical data used for training to remove outdated records.

The metrics of support and confidence from Market Basket Analysis, which will be used in Chapter 4, can also be applied to the predictions produced by the Markov model. Support is a measure of how often this association has been seen in the historical data. An example of high support would be a prediction where the model has recorded Bob in the office at 9:00 am hundreds of times. An example of low support is the model recording Bob at the mall once. The confidence is the probability of a given prediction. Both of the examples given would have high probabilities, and therefore, high confidence, for Bob being at the office or the mall despite the different values of support. A useful implementation of this Markov model needs to communicate both the support and the confidence of its predictions to allow the user to have a measure of trust in the usefulness of the predictions.

3.4 Summary

This chapter discussed the problem of predicting future locations, especially in cases where the location data may be sparse. Here, sparse data are represented as timestamped locations that are used to train a variable-order Markov model with a maximum order of one. The time in question is used as the context, and the corresponding locations are the resulting predictions. Fallback to the zeroth-order (the most popular location) is used if the context time is not found. This model,

Predicting Future Locations

tested on data collected at UCSD, was able to predict location, given a context time, correctly from 78% to 94% of the time. In contrast, the Dartmouth study which predicted only the *next* wireless access point was able to correctly predict 65% to 72% correctly [48]. While most related projects rely on using continuous data to predict the next location in a sequence, the results reported here are not constrained by the requirement for continuous location records.

Chapter 4 Predicting Arrival Times

The previous chapter explored methods of predicting *where* an individual would be at a given time. This chapter explores the converse question; that of *when* an individual will be at a given location.

Section 4.1 begins by applying the location-predicting Markov model to temporal predictions. As described in Section 2.2.3, a sequence predictor like the Markov model is not a likely solution for a temporal predictor. As expected, the Markov model fails, but the investigation into *why* it fails reveals the difference between location- and temporal-predictions. Given that failure, Section 4.2 then introduces a heuristic model for temporal-prediction that accounts for those differences. The algorithm is explained in Section 4.3. Section 4.4 contains the result of the experiments using the new model. Section 4.5 introduces metrics for showing the confidence in individual predictions.

4.1 Temporal-Prediction using the Markov Model

As explained in Section 2.2.3, state or sequence predictors such as Markov models are not suitable for predicting multiple outputs such as a list of future times. Too many predictions are returned and there is a loss of contextual information in statistical models such as the Markov model that are based on counting events. This section applies the previous location-prediction model to temporal-prediction as an investigation into the differences between the two problems. The method described in Chapter 3 was to create sequences of *(time, location)* pairs, use them to train a low-order Markov model with fallback, and then predict the location that was most often

associated with a given time. In reality, the application of this model to temporal-prediction fails. At best, it predicts the correct time associated with a given location only 10% of the time.

The Markov model for temporal prediction was identical to the model used in Chapter 3. Instead of training on data that was formatted as a sequence of times and their corresponding locations,

$$\{time_0, location_0, time_1, location_1, \dots, time_n, location_n\},$$

the sequences were rearranged to form (location, time) pairs:

$$\{location_0, time_0, location_1, time_1, \dots, location_n, time_n\}$$

where $location_t$ is the context, or the corresponding location, for $time_t$.

As with the location-based model, the temporal-prediction model was trained and tested with the same runs of five weeks of training data with a corresponding week of testing data.

Fallback was not used in this model because it does not make logical sense. In the location application of the model, which is answering the *where* question, it may make sense for an application to fall back to the most common location. For the *when* question, such as, “When will Bob be in Hawaii?” it does not make sense for the model to respond, “I have no data for ‘Hawaii,’ but the most common time in the model is 10:00 am, so I will predict that Bob will be in Hawaii at 10:00 am.” In these fallback cases which are missing the context location, the model should refuse to predict.

If none of the temporal-predictions returned were correct, these incorrect predictions were then checked to see if they were within 10 or 20 minutes of the correct time. This corresponds to the checking of neighboring access points that was done for location- prediction.

Predicting Arrival Times

The results for using the Markov model to predict the time someone will be at a given location are shown in Figure 4-1. Initial results are dismal: less than 11% of the best predictions are correct for both datasets. Results improve slightly by allowing predictions to be within 10 or 20 minutes of the correct time.

One problem with using this approach can be seen by visualizing the data. The data for User 013 are shown in Figure 4-2. User 013 visits only three locations throughout the 10 weeks, and spends most of her time at Location #1 (blue). If the model is trained on User 013's data and asked to predict *where* she would be at any given time, Location #1 would almost always be a

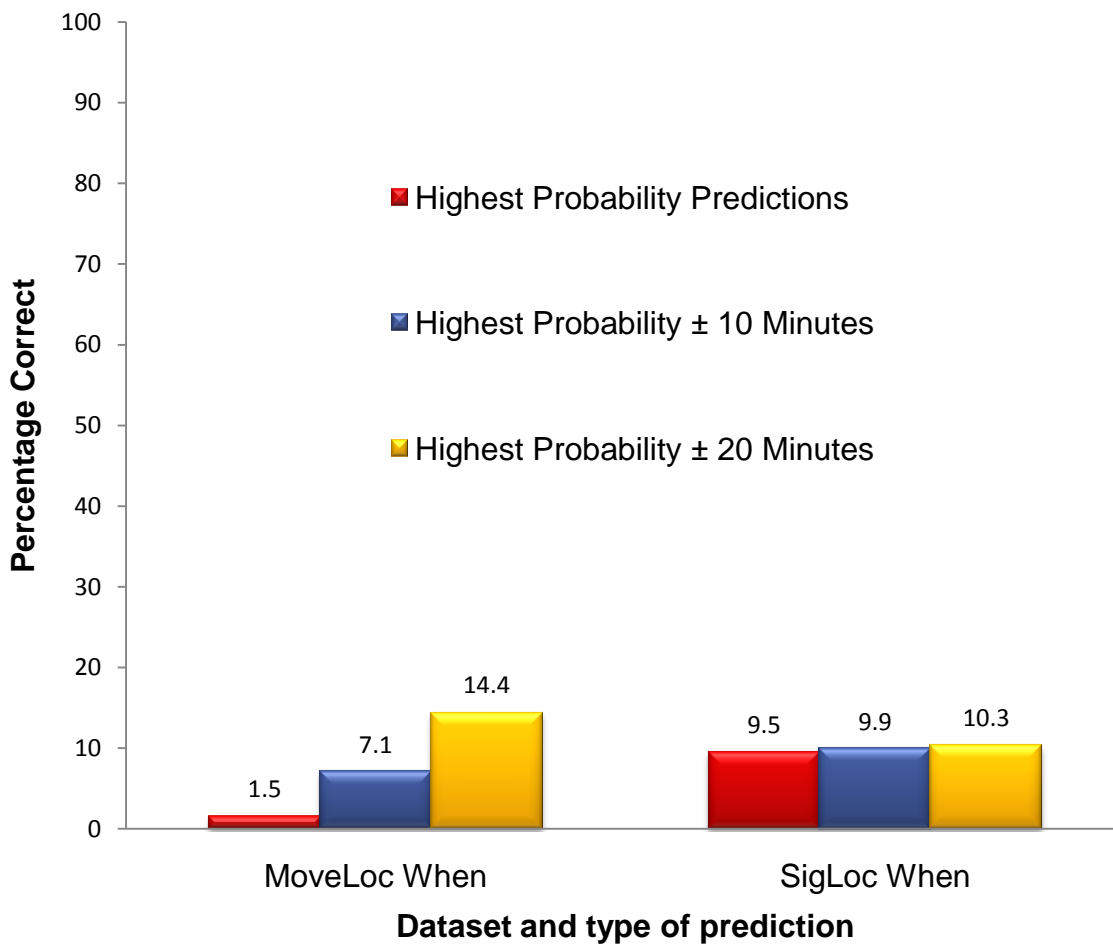


Figure 4-1 Predicting times at future locations using the Markov model

Predicting Arrival Times

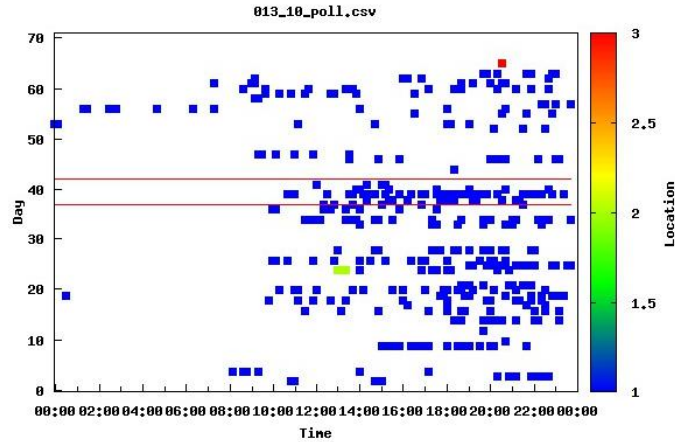


Figure 4-2 Data for User 013 (10-minute windows)

correct answer, leading to a very high prediction rate. And indeed, for the week #6 testing data, inside the red lines above, the model predicted her location 100% of the time. If the model is asked the converse question, “When is she most likely to be at Location 1?” there is no reliable answer, and the predictions fail.

One of the reasons for the differences in the location- and temporal-prediction results is apparent when the quantity of predictions returned is compared. The upper bound on the number of predictions returned is the maximum number of states. For temporal-predictions, the maximum number of states is 144 for the 10-minute SigLoc data, and 1,440 for the 1-minute MoveLoc data. The maximum number of locations varies by user. Table 4-1 shows the median number of unique locations and times in the training files. The number of location states is much less than the number of temporal states; hence, the model is more likely to get the location correct. The numbers of predictions returned listed in Table 4-2 also reflect this discrepancy. The number of

Table 4-1 Average number of unique locations and times in training files

Dataset	Ave. # Locations / Training File	Ave. # Times / Training File
MoveLoc (1-minute)	18	646
SigLoc (10-minutes)	5	68

predictions returned for location-prediction is much less than the number of predictions returned for time-prediction.

Table 4-2 Number of predictions returned for each test

DataSet	Question	Median number of Predictions returned	Maximum number of Predictions returned
MoveLoc	Where?	2	13
SigLoc	Where?	2	9
MoveLoc	When?	1439	1440
SigLoc	When?	144	144

The Markov model returns *all* of the predictions found for a given context. The initial results considered only the prediction(s) with the maximum probability. When the model considers all of the predictions returned, regardless of their probabilities, the results improve. Figure 4-3 shows the previous results with the addition of the percentage of correct predictions if all of the predictions returned by the model are considered. This high predictive accuracy is not very useful, as shown in Figure 4-3, where the results for “all predictions,” in most cases, cover an entire day.

Thresholds could be used to return a subset of predictions. Indeed, any practical application of a temporal predictor would need to adjust the number of results returned according to the needs of the user. Some applications may need to know the single most probable time that someone is in a given location, while others may need to display a range of probable times, where the probability threshold used to determine the range is set by the user. Thresholding was applied to the results from this section and are shown in further detail in Appendix B

Predicting Arrival Times

Applying a Markov model to temporal-prediction shows that temporal-prediction is a different problem than location-prediction. Location predictors return a very small number of discrete predictions while temporal predictors need to return a range of continuous values. Temporal-predictors need to communicate the probabilities of the values in the range, such as “There is a 50% chance probability that Bob will be in the office by 8:00am and a 95% chance he will be in the office by 9:00am.”

There is a notion of distance that is different when considering location versus time. The distance between two locations is fixed and does not change in any circumstances. In the

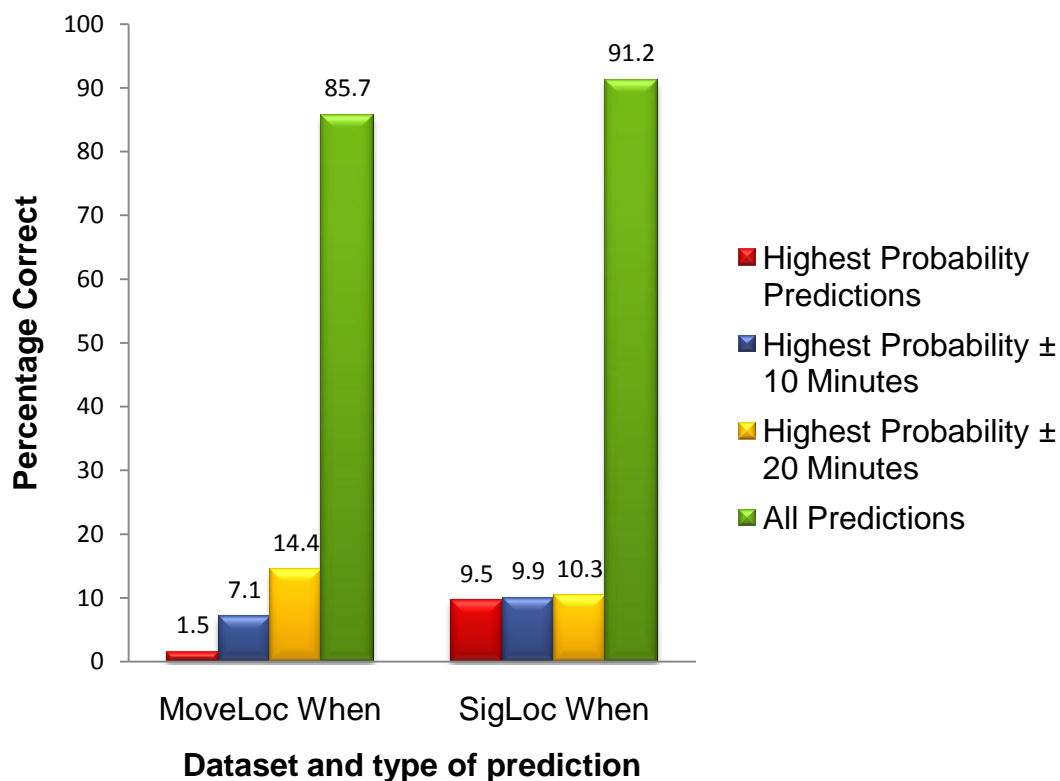


Figure 4-3 Predicting times at future locations using the Markov model including all predictions returned by the model.

application of temporal-prediction, the concept of the distance between two times may differ

depending on the circumstances. For example, if Bob shows up at the office every day at 8:00am and does not leave until 5:00pm, then asking a question about Bob at 4:00pm is the same as asking about Bob at 8:30am. In this sense, the time “4:00pm” is near the time “8:30am.” However, if Bob changes his location every five minutes, then the time “8:30am” could very different that “9:00am.” This concept of distance shows up in the changes in someone’s locations throughout the day. Sequence predictors such as the Markov model do not retain information about the order in which movements occur or when location changes. A new model is needed that incorporates and uses this information about the times when someone changes their location. The rest of this chapter describes this new model, the results when the model was applied to the data, and analysis.

4.2 The “Traversing the Sequence” Model

This section defines an algorithm that retains more contextual information. This model is called ‘Traversing the Sequence’ or the SEQ model, because it keeps the entire sequence of times and locations intact and, given contextual information such as a previous time or location, it traverse the sequences to predict a future time at a given location.

The SEQ model represents the user’s day as a sequence of (*arrival time, location*) pairs. Note that this is a different representation than the Markov model. In the Markov model, for example, if a user spends 30 minutes in one location, the result is three records in the SigLoc dataset, one for each 10-minute timeslot. In the SEQ model, this location is represented as one pair with the arrival time and the location and no duration information.

The SEQ model is an *unsupervised learning* model, in that it is not trained on historical data that have been labeled with correct answers. Instead, it stores the historical data without

modification. When it is given context, such as the location in question, it searches the historical data for matches. This is a modification of the template-matching used in pattern recognition. The difference is that this technique does not require exact matches of lengthy patterns. For example, the SEQ algorithm can search for a complicated set of conditions, such as, “Bob was seen at a coffee shop at 8:12am on a Monday; when is he likely to be in his office on a Monday?”

The differing amounts and types of contextual data are referred to as the “order” of the model. The model is always queried about a location. The zeroth order of the model is the case when no additional information is given. The first-order model is used when one item of additional context is supplied, either a previously-observed location or time. If both a previously-observed time and location are given, the second order algorithm is used. The following section describes the algorithm with an example that spans one day.

4.3 The Traversing-the-Sequence Algorithm

The algorithm begins by reading in an input file and storing the sequences. There is one sequence for each day. An example sequence for user Alice is shown here in Figure 4-4, with the time values shortened to the hour portion of the time.

8	Coffee	9	Office	12	Lunch	13	Office	14	Meeting	15	Office	16	Meeting	17	Office	18	Home
---	--------	---	--------	----	-------	----	--------	----	---------	----	--------	----	---------	----	--------	----	------

Figure 4-4 A sample sequence

Zeroth Order- No previous context

For the zeroth order, the model is given a destination location (also referred to as ‘quest-location’) and it returns a list of the times that location has been visited. In this example, if the quest-location was ‘Meeting,’ the model would return a set of times and their respective counts of the instances of ‘Meeting’. The set of counts always includes a count for ‘Never,’ which is a count of the number of sequences which did not contain the destination location. The count for ‘Never’ is used to give an indication of the support of the prediction. If the count for ‘Never’ is significantly higher than the other counts, it means that the person rarely goes to that destination location.

Figure 4-5 shows the results if the SEQ model was given the historical data from Figure 4-4 above and asked to predict when Alice will be at “Meeting.” Because there is no previous context given, the order is 0. The quest location is “Meeting”. The results show 1 count of being at a meeting at 1600 and one at 1400, so there is a 50% confidence for each. The support is 100% because the quest location was seen in all of the sequences in the training data (in this case, one sequence).

```
/home/Ingrid/crowdad/ucsd/seq> ./read_seq.py
0th Order: context: None, Quest_location: Meeting
{'16:00': 1, '14:00': 1, 'Never': 0}
```

Figure 4-5 Using the SEQ model to predict when Alice will be at a meeting location (0th Order)

First Order – One item of context

The first-order model is given one item of context, either a previous time or previous location. The process is the same for either. The model searches each sequence for the destination location. Of the sequences that contain the destination location, the model finds the sequences

Predicting Arrival Times

that also contain the context time or context location, and counts the number of times it sees the destination after each occurrence of the context time or location. Figure 4-6 includes several examples. In the first case, the context is the time 15:00. In essence, the model is being asked, “It’s 15:00. When will Alice be at the Meeting?” The model returns a set of counts: It ‘saw’ Alice at the meeting after 15:00 once, at 16:00.

<code>/home/Ingrid/crowdad/ucsd/seq>./read_seq.py</code>	
<code>1st Order:</code>	
<code>Context: 15:00, Quest_location: Meeting</code>	Q: When will Alice be at a meeting after 15:00?
<code>{'16:00': 1, 'Never': 0}</code>	A: At16:00
<code>Context: Lunch, Quest_location: Meeting</code>	Q: When is Alice’s first meeting after Lunch?
<code>{'14:00': 1, 'Never': 0}</code>	A: At14:00
<code>Context: Office, Quest_location: Meeting</code>	Q: When will Alice be at a meeting after being in the Office?
<code>Office, Meeting</code>	A: At 14:00 and at 16:00
<code>{'16:00': 1, '14:00': 1, 'Never': 0}</code>	

Figure 4-6 The SEQ model predicting times given previous context of a time or a location.

Second Order – Time and Location as context

The second-order model requires two items of context: a time and a location. The algorithm for the second-order model includes fuzzy logic, because the context times may not match exactly with the times in the sequence, and there are different types of situations which need to be addressed. A flowchart is in Figure 4-7.

Predicting Arrival Times

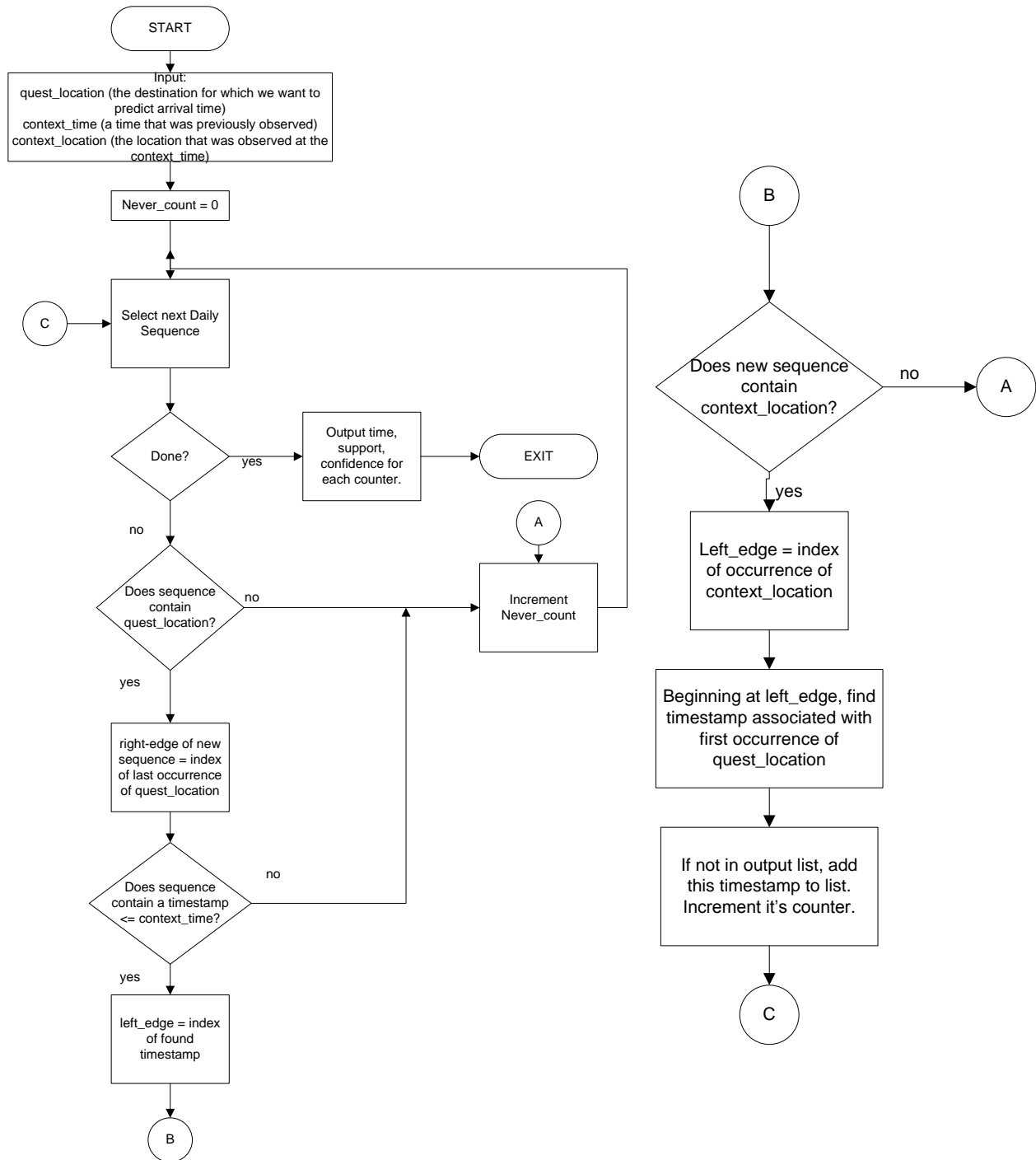


Figure 4-7 Flowchart: applying the SEQ model with both time and location as previous context.

Predicting Arrival Times

This is illustrated below. The illustration begins with our sample sequence and asks the model to predict when Alice will be at the meeting if she is in the Office at 10:00. In this case, the quest-location is “Meeting,” the context-time is “10:00” and the context-location is “Office.”

First, check if the quest-location, “Meeting,” is in the sequence.



8	Coffee	9	Office	12	Lunch	13	Office	14	Meeting	15	Office	16	Meeting	17	Office	18	Home
---	--------	---	--------	----	-------	----	--------	----	---------	----	--------	----	---------	----	--------	----	------

Figure 4-8 SEQ model 2nd order: check that the quest location is in the sequence

Find the time equal to or closest to (but earlier than) the context time. In this example, our context time is 10:00am, and the context time that is closest and earlier to 10:00am is the entry at 9:00am. The location of this time becomes the left edge of the sub-sequence to be searched.



8	Coffee	9	Office	12	Lunch	13	Office	14	Meeting	15	Office	16	Meeting	17	Office	18	Home
---	--------	---	--------	----	-------	----	--------	----	---------	----	--------	----	---------	----	--------	----	------

Figure 4-9 SEQ model 2nd order: find context time in sequence

Search the sub-sequence to find the **next** occurrence of the destination location. The right-edge of the sub-sequence is when the location changes from the destination location to another location.

Predicting Arrival Times

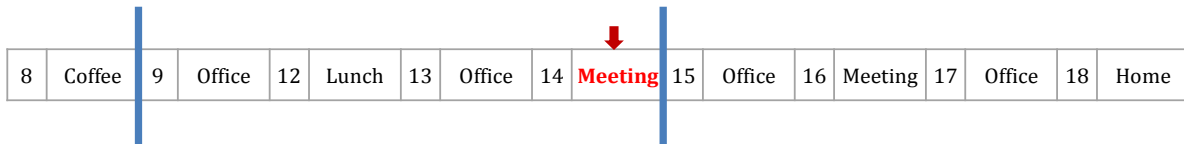


Figure 4-10 SEQ model 2nd order: find next occurrence of quest location

Search the sub-sequence to find the context location. If it is found within the sub-sequence, it means that the algorithm has found a sub-sequence that meets all of the input criteria: it includes the destination location (Meeting), the context time ($\leq 10:00\text{am}$) and the context location (Office). The timestamp that is paired with the destination location is returned as the predicted time. In this example, the algorithm predicts that, given that Alice was observed at the office at 10:00am, she will be at the Meeting location at 14 (2:00pm).

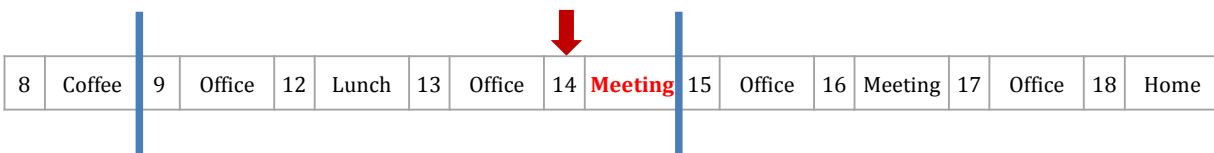


Figure 4-11 SEQ model 2nd order: time prediction

4.3.1 An Example of the SEQ Model

This section applies the algorithm to a simple example that spans 10 weekdays of historical data. The user, Bob, usually goes to the office at 9:00am after stopping for coffee. One day, he went to the office at 8:00am and the following day, he went to the office at 8:00am and had a meeting at 9:00am. Each day, he returns home at 5:00pm (17h00). In this example, “home” is considered the default location for the beginning and ending of each day.

Predicting Arrival Times

	08h00	09h00	10h00	11h00	noon	13h00	14h00	15h00	16h00	17h00
M	coffee	office								home
T	coffee	office								home
W	coffee	office								home
R	coffee	office								home
F	coffee	office								home
M	coffee	office								home
T	coffee	office								home
W	coffee	office								home
R	office									home
F	office	mtg	office							home

Figure 4-12 Two workweeks in Bob's life

Example #1 is a simple question: When will Bob be in the office? The algorithm searches the sequences to find all of the occurrences when Bob first arrives at the office.

Table 4-3 SEQ model counts for when Bob arrives at the office.

<i>Time corresponding to "Office"</i>	<i># times this location appears at given time</i>
08h00	8
09h00	2

Note that this example shows arrival time and does not include duration information, which could be added.

If there are days when Bob never goes into the office, they are also reported because they affect the support of the calculations. Support and confidence in the predictions is discussed in section 4.5.

There are two types of input into the sequence algorithm. The minimal, mandatory input is the destination location in question. In the previous example, the destination location is “Office.”

Predicting Arrival Times

When each sequence is searched, that location is used to form the right-hand boundary, as shown below in Figure 4-13.

	08h00	09h00	10h00	11h00	noon	13h00	14h00	15h00	16h00	17h00
M	coffee	office								home
T	coffee	office								home
W	coffee	office								home
R	coffee	office								home
F	coffee	office								home
M	coffee	office								home
T	coffee	office								home
W	coffee	office								home
R	office									home
F	office	mtg	office							home

Figure 4-13 Sequences truncated by the location in question

Optional inputs into the algorithm can be used to select a subset of sequences to be searched (such as ‘weekdays’) and can also be used to select the portion of the sequence which is searched. This additional context eliminates sequences that do not include the context and forms the left edge of the subsequence in sequences that include the context. If the additional context in our example is that Bob was observed in the coffee shop at 8:00 am, the last two sequences would be dropped from the search and the remaining sequences would have the left boundary shown in Figure 4-14.

Predicting Arrival Times

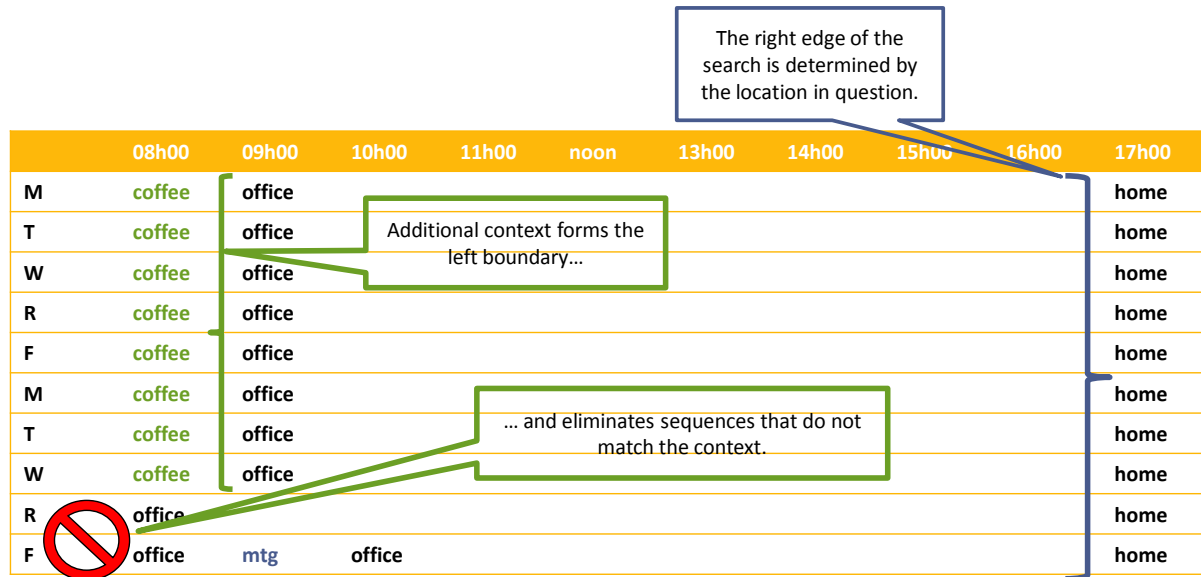


Figure 4-14 Sequences truncated or eliminated by additional context.

The additional context can be used to narrow down situations to get a more precise prediction.

For example, if Bob is in and out of his office during the day, asking when he will be in his office after lunch may result in fewer, but more accurate predictions.

4.3.2 Advantages and Disadvantages of the SEQ Model

The advantages and disadvantages of this approach are summarized in Table 4-4 and described below.

Table 4-4 Advantages and disadvantages of sequence approach

	Context: It is possible to add additional context, such as day-of-the-week, to the model and to filter sequences by this additional context.
	Unsupervised learning. Model can ignore outdated historical data without re-training.
	Adaptable to use partial context.
	Memory: Uses more memory than compression-based models, making searches slower.

The SEQ Model incorporates additional context

Sequence predictors, such as the compression-based models that are usually applied to problems of this type, use symbols from a single alphabet. In the case of location-predictors, this alphabet comprises location symbols. In the previous chapter, the Markov model predictor was expanded to use two types of input: time and location. However, it is not easily expandable to include further context. Each type of additional context introduces a new set of states in the Markov model, making it difficult to find patterns. For example, if day-of-the-week was incorporated into the Markov model, the state “Monday 8:00 am” would be as different from “Tuesday 8:00 am” as it is from “Saturday 10:00 pm.”

The additional context supported by the SEQ model can be used to label the daily sequences or specific entries. For example, the model can filter the daily sequences by day-of-the-week or season of the year, or any other date-related factors. One could even add seemingly unrelated context such as weather conditions.

The *(time, location)* entries within each sequence can also be tagged with additional context to refine the predictions. If the sequences are created from different sources of information, such as multiple location systems or online calendars, the *(time, location)* pairs could be tagged with a label indicating their source. This source information is then used to tune the predictions according to the credibility of the source. For example, locations determined by GPS could be considered more accurate than locations determined by cellular tower. In another example, online calendar entries are not as accurate as actual observations of someone’s location [67]. The model could use these tags to report its confidence in the predictions.

The SEQ model is unsupervised

Supervised models, such as the Markov model described in the previous chapter, need to be trained on known data before they are used to predict unknown data. Adding new data or discarding old data requires retraining. As discussed at the end of the previous chapter, the application developers would have to decide *a priori* on a retraining schedule that balanced the time and computation spent re-training versus the need to incorporate recent data.

The sequence model is not trained in the same way. The historical data are stored but not processed (or ‘learned’) by the model. This makes the model very flexible in that it can dynamically decide which sequences to use for the prediction and which to ignore. One example of this is a user who suddenly changes his location patterns. Previously, it may have been most effective to use the previous eight weeks of his location data to predict his future movements. However, if the model starts to see some noticeable changes to his patterns, the model may decide to use only the last few days’ worth of patterns to predict the next day. This unsupervised model has the ability to change the window size of the previous sequences searched to arrive at a prediction.

The SEQ model is adaptable to partial context

The sequence model works with questions where the previous state is only partially defined. This model can predict if only time is given as previous context or only location. For example, this model can answer questions where time is given as previous context such as, “It is currently noon. When will Bob be the office next?” or questions where location is given as previous context, such as, “Bob is at the coffee shop. When will he next be at the office?”

As mentioned previously, this dissertation refers to these different levels of previous context as different *orders* of the model. They are summarized in Table 4-5.

Table 4-5 Orders in the 'traversing the sequence' model

Order	Inputs:	Example
Zero Order	Destination location	“When will Bob be at the office?”
First Order – Time	Current time, destination location	“It’s noon. When will Bob next be in the office?”
First Order – Location	Current location, destination location	“Bob is at the coffee shop. When will he next be at the office?”
Second Order	Current time, current location, destination location	“It is 8:00am and Bob is at the coffee shop. When will he next be at the office?”

Memory and Execution Time of the SEQ Model

The purpose of compression- based algorithms is, obviously, to compress the input to consume less memory. The SEQ model does no compression, so it will use more memory storage. The current implementation stores the data as Python [68] strings.

The sequences are comprised of *(time, location)* pairs. A sequence is created for each day.

During execution, each sequence may be searched up to two times, once to see if it contains the destination location and once to search for any additional context. If the number of days of data, or sequences, is d , and the number of *(time, location)* pairs per sequence is n , the search time is $O(d * n^2)$. As will be shown, the SEQ algorithm performs better with coarser granularity. In addition, the location determination system may not record continuous data, so it is reasonable to assume that the value of n would, in reality, be smaller than 1,440. For the data used in these experiments, the average values for d and n are shown in Table 4-6.

Table 4-6 Average number of sequences and pairs per sequence

<i>DataSet</i>	<i>Average Number of Sequences (d)</i>	<i>Average Number of time,location pairs per sequence(n)</i>
SigLoc (10minute timestamps)	19	5
MoveLoc (1 minute timestamps)	25	29

Summary

This section discussed the advantages and disadvantages of the SEQ model. The advantages of this model include its ability to incorporate context in the model, its unsupervised approach and its flexibility in being able to take advantage of partial contextual knowledge when making a prediction.

The algorithm was then applied to the data. Details of the experiment and the results are described below.

4.4 Experiment and Results

The same source user data that were used for the Markov model were used for the SEQ model. Instead of using 16-bit symbols to encode time and location as was done for the Markov model, the data for the SEQ model are stored in human-readable, comma-delimited files. A portion of a file is shown in Figure 4-15. The SEQ model was written in Python [68] because of its rapid-development support and ease of handling text strings.

```
2002-10-02; 2; Wednesday; 09:50, 355, 12:20, 184, 13:20, 173, 15:30, 435
2002-10-03; 2; Thursday; 00:00, 355
2002-10-04; 2; Friday; 22:40, 355, 22:50, 368, 23:00, 355
2002-10-05; 2; Saturday; 00:00, 355, 10:30, 363, 10:50, 355, 12:00, 360,
12:10, 355, 13:00, 360, 13:10, 355
```

Figure 4-15 Portion of a training file used for the SEQ model

Predicting Arrival Times

The model was tested with different types of previous context, which is referred to as ‘order’. In the 0th order test, the only input is the location. The model is queried as to when the user will be at the test location. This location is referred to as the *quest location*. A previous time can be given as additional context; this is noted as ‘1-time’ order. A previous location can also be used as additional context; this is noted as ‘1-loc’ in the model. And finally, both a previous time and location can be used as additional context, creating the second order.

Aggregate results for the 1-minute and 10-minute datasets are shown in Figure 4-16 and Figure 4-17, respectively. The prediction accuracy has improved over the Markov model used in Section 4.1. The results include the rare situations where the model refused to make a prediction because the location in question was not found. These could be considered to be correct predictions as the model should refuse to make a prediction for a query such as “When is Bob on Mars?” Allowing for predictions within 10 or 20 minutes of the correct testing time yields an even higher predictive accuracy.

There are two causes for the improvement over the Markov model. First, the SEQ model predicts only arrival times. The Markov model indirectly encodes duration, in that records are repeated until the user changes location. The Markov data, therefore, has a chain of multiple records if a user is at a particular location for a length of time. Since the length of the user’s stay may vary, the tests on the records at the end of the chain of testing data may produce wrong predictions, reducing the average predictive accuracy. Additionally, the SEQ model uses a smaller dataset of arrival times and locations instead of polled timeslots and locations and returns fewer predictions.

Predicting Arrival Times

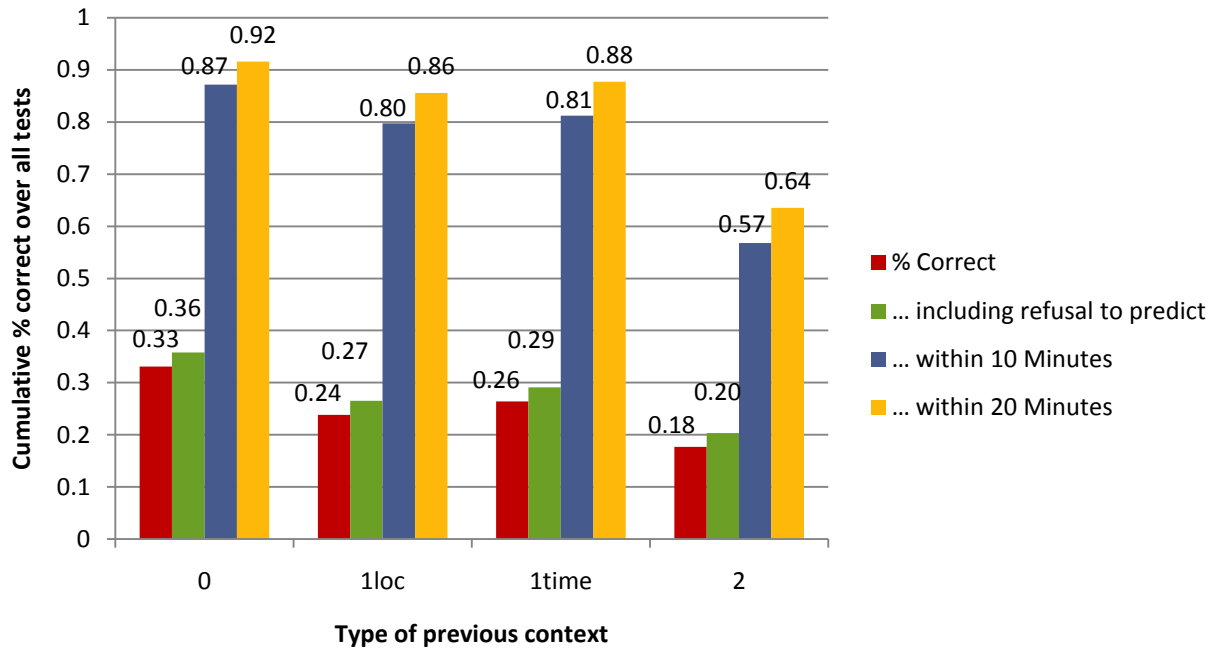


Figure 4-16 Aggregate results of the SEQ model on the 1-minute data

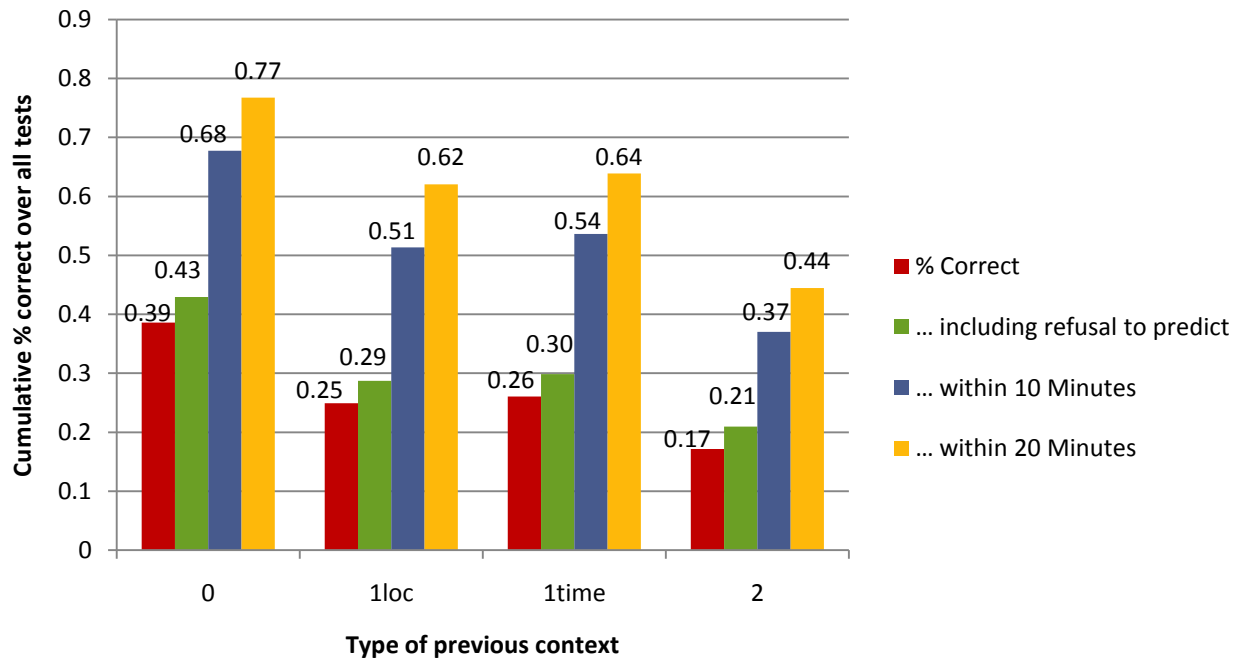


Figure 4-17 Aggregate results of the SEQ Model on the 10-minute data

Predicting Arrival Times

Figure 4-18 and Figure 4-19 show the normalized distribution of the prediction results at the zeroth order for the 1-minute and 10-minute data, respectively. Similar graphs for the other orders are in Appendix C .

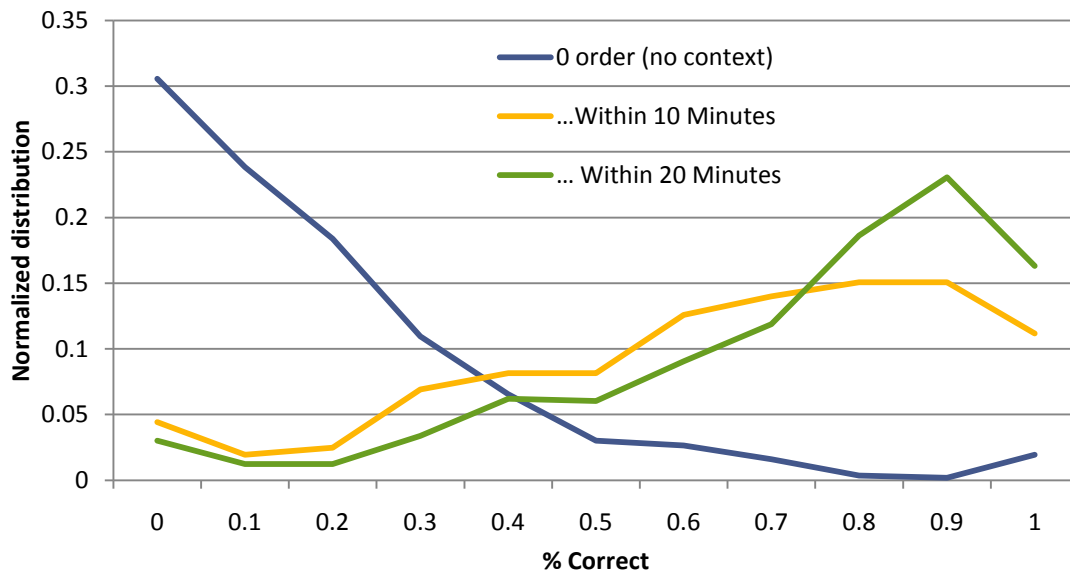


Figure 4-18 Normalized distribution of SEQ model prediction results for the zeroth order (1- minute data)

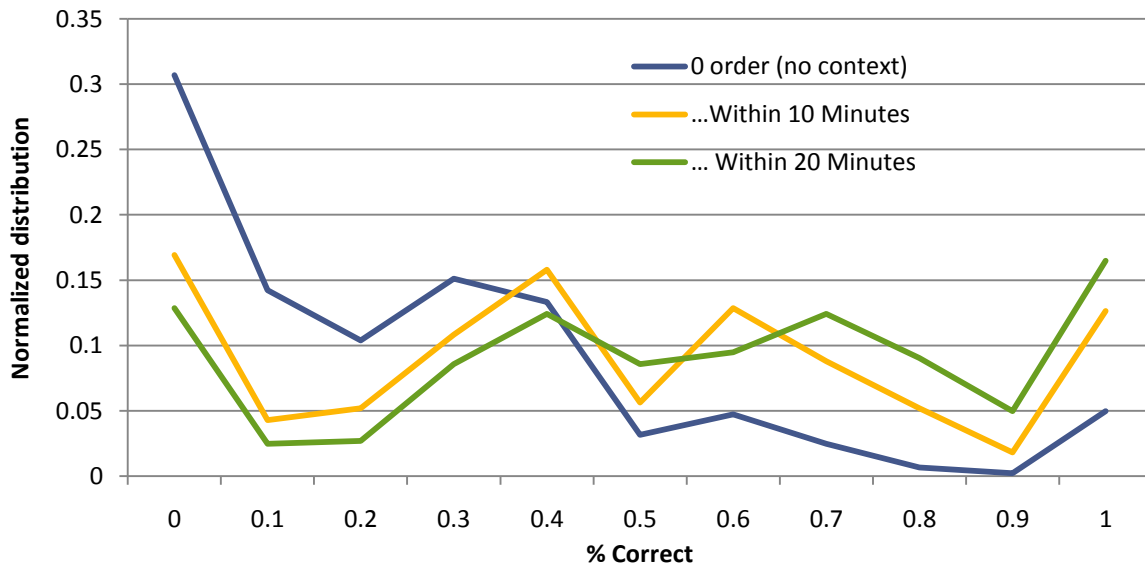


Figure 4-19 Normalized distribution of SEQ model prediction results for the zeroth order (10-minute data)

Figure 4-20 and Figure 4-21 show the cumulative distributions for the prediction results for the 1-minute and 10-minute data, respectively, for the 0-order tests. These graphs also include the prediction results for an ideal predictor which is never wrong and a predictor based on randomly guessing from the timestamps recorded in the training data. The SEQ model predictor performs better than random guesses. Allowing a 10- to 20-minute window increases the prediction results significantly.

4.5 Scoring Individual Predictions

These results are cumulative results over a set of tests. For individual predictions, the Market Basket Analysis metrics of *support* and *confidence* would be also be calculated to give a measure of the belief in the prediction.

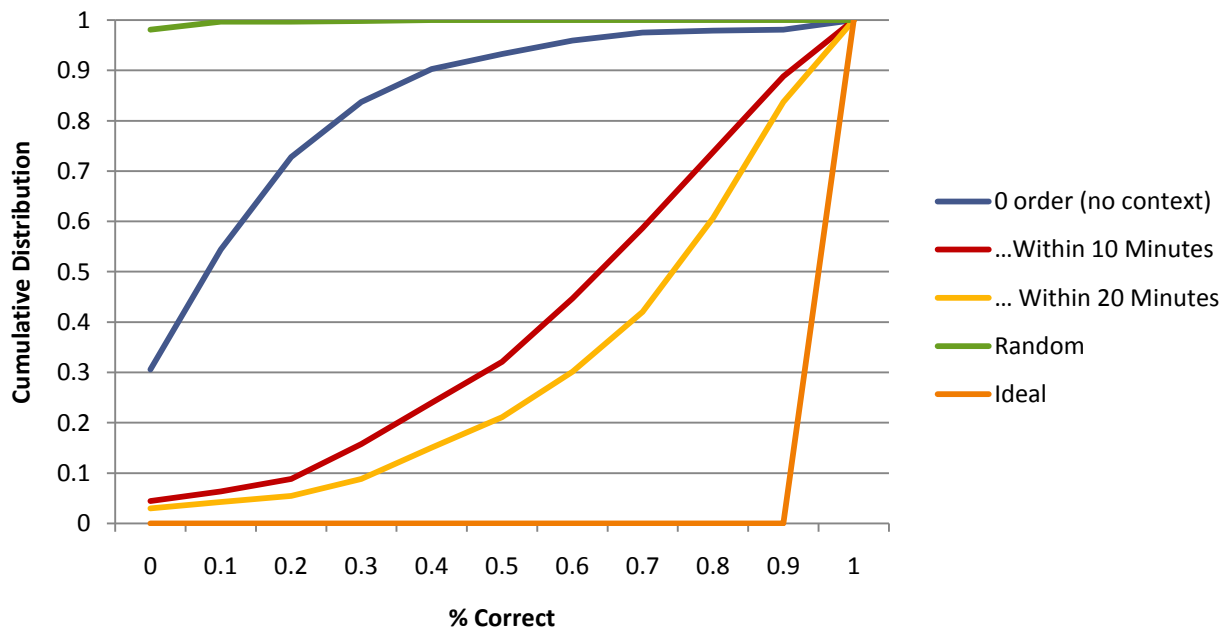


Figure 4-20 Cumulative distribution of SEQ model prediction results for the zeroth order (1- minute data)

Predicting Arrival Times

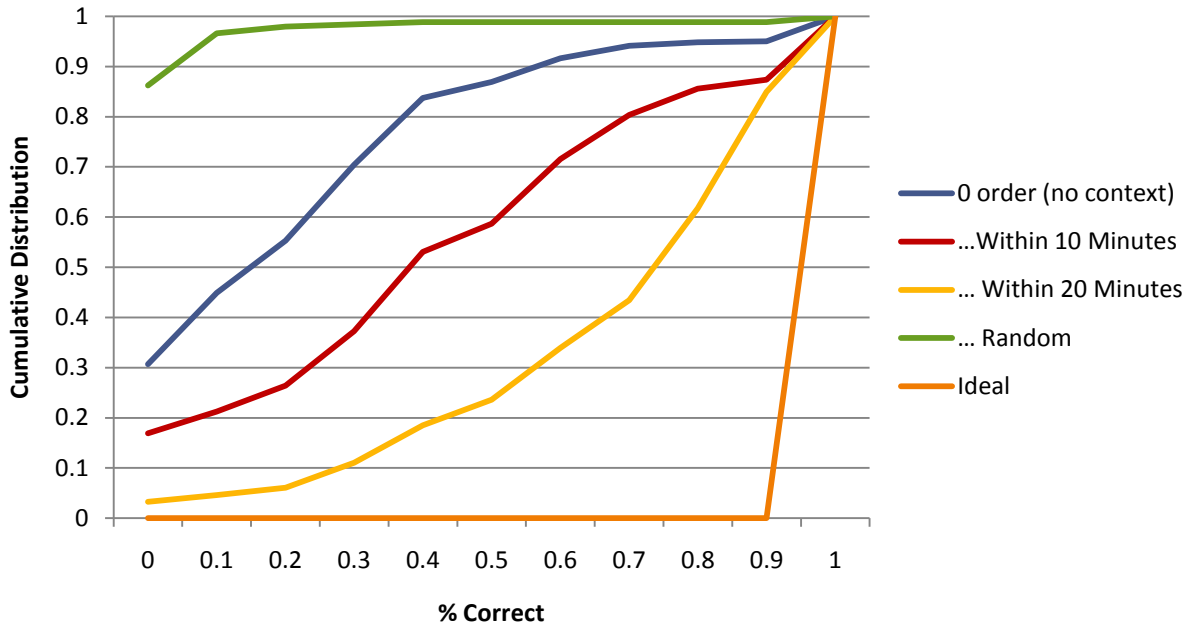


Figure 4-21 Cumulative distribution of results of SEQ model on 10-minute data

Market Basket Analysis [69], also known as affinity analysis, is an application of data mining which is used to find relationships between items in a large dataset. The classic example of Market Basket Analysis is that of analyzing shoppers' habits and discovering that when a male shopper buys diapers, he also buys beer. (This particular example, which discovered that men who bought diapers on Friday evenings also bought beer, includes a constraint on gender and time. Not all analyses include these constraints).

The output of a Market Basket Analysis is a set of rules of the form “If A, then B.” Two metrics, *support* and *confidence*, are used to sift and rank the rules. The probability of the A term occurring is referred to as the *support*, and the conditional probability of the B term occurring given the A term is referred to as the *confidence*.

Predicting Arrival Times

In the SEQ model, the support is defined as the number of sequences in which the quest location and any given context appears. For example, if the historical data covered 30 days, and Bob went to the office on 22 of those days, the support for a quest location of “Office” would be

$$\text{Support} = \frac{\text{number of sequences containing "Office"}}{\text{total number of sequences}} = \frac{22}{30} = 0.7 \quad (4-1)$$

The confidence measures the strength of the prediction given its support. In the current example, if Bob went to the office at 8:00am on 20 days, the confidence in the prediction that Bob will be at the office at 8:00am is

$$\text{Confidence} = \frac{\text{Number of instances of Bob at "Office" at 8:00am}}{\text{Number of sequences containing "Office"}} = \frac{20}{22} = 0.9 \quad (4-2)$$

For queries that include contextual information, the support and confidence calculations include the context. If the current example’s query changed to “When will Bob be in the office if he was seen at 7:00 am buying coffee?”, the support and confidence equations would change to include the context.

$$\text{Support} = \frac{\text{number of sequences containing (7:00 am, Coffee) and "Office"}}{\text{total number of sequences}} = \frac{15}{30} = 0.5 \quad (4-3)$$

$$\text{Confidence} = \frac{\text{Number of instances of Bob at "Office" at 8:00am}}{\text{Number of sequences containing (7:00 am, Coffee) and "Office"}} = \frac{12}{15} = 0.8 \quad (4-4)$$

In a typical Market Basket Analysis that produces thousands of rules, support is used as a threshold, and rules with low support are discarded. The confidence of the remaining rules is used to rank their validity. In this prediction application, both measures would be considered before using a prediction. Support is a measure of how common the location is in the user’s historical data. Confidence is a measure of how regularly the user appears at that location.

Support and confidence are comparative measures, not absolutes. As the amount of historical data grows, the absolute values for support and confidence decrease. As new unique locations

are added, the percentage for a specific location decreases, lowering its support. Likewise, as more timestamps are added, the total counts for times at a particular location (the denominator) increase, lowering the confidence for any particular timestamp. Even though the absolute values decrease, the confidence of predictions is still used to rank the predictions.

Figure 4-22 and Figure 4-23 show the average values for support and confidence for the times the model made a correct prediction. One reason for the high values for support is a low number of unique locations in the historical data. At the zeroth order, where the only given information is the location in question, support is high, meaning that there are many sequences that contain that location, but confidence in the time prediction for a particular timeslot is lower. As the amount of previous context is increased to include both a previous time and a previous location (second order), fewer sequences are found that match the context criteria (low support), but the confidence in those predictions is higher. For example, consider the query asking when Bob will arrive at the office. Bob arrives at the office almost every day, so there will be many daily sequences with “office” in them and therefore, support will be high. He may also have a range of arrival times, so the confidence in any one arrival time may be low. Assume that once each month, Bob brings doughnuts to the office before a meeting. These occasions may occur rarely, but when they do, Bob is almost always at the office by 9:00am. The support will be low, but the confidence in the prediction will be higher.

Predicting Arrival Times

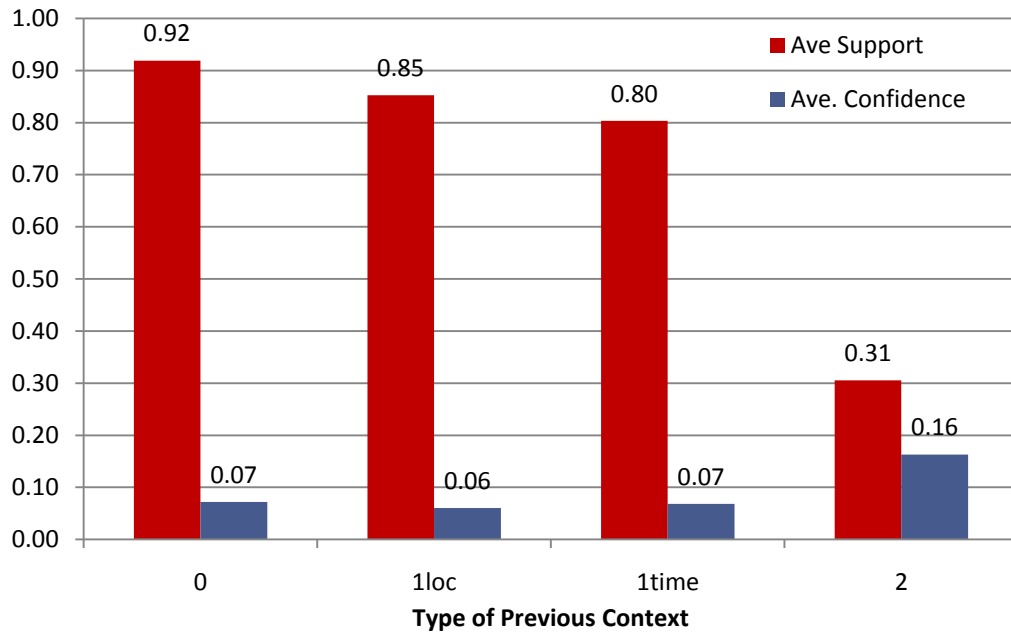


Figure 4-22 Average support and confidence metrics for correct predictions (1-minute data)

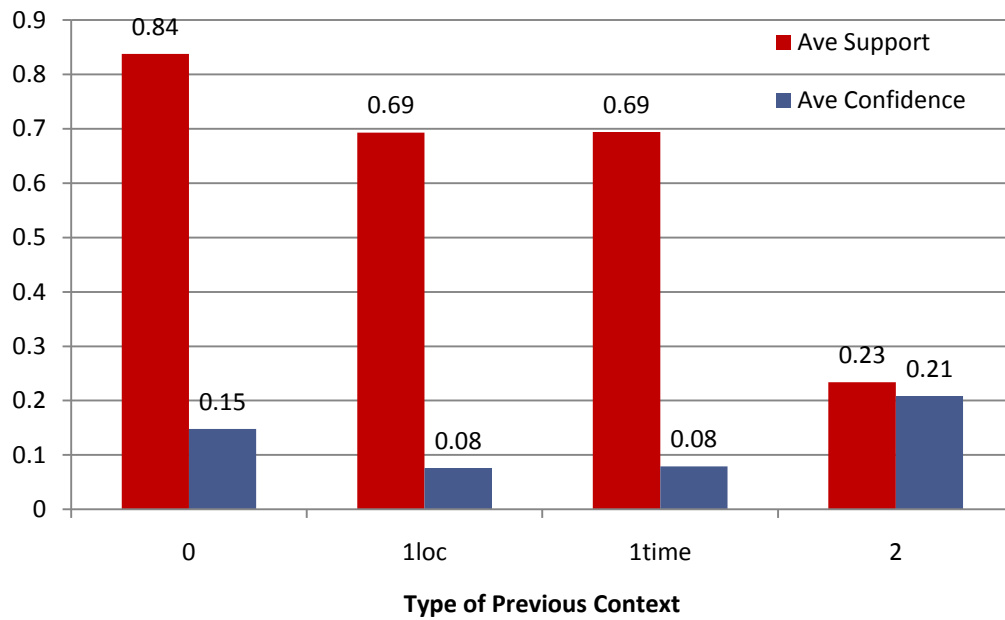


Figure 4-23 Average support and confidence metrics for correct predictions (10-minute data)

4.6 Summary

This dissertation is the first work to predict time of arrival to arbitrary locations. Temporal-prediction is a very different problem than location-prediction because of the large number of possible answers (states) and the mapping between location and time. (Someone can only be in one location at a given time, but they can at a given location at several different times.) An unsupervised model called the SEQ model was developed that retains all of the input data in order. These stored patterns of daily movements allow queries with context such as previous locations or times. When the model is given no previous context, it predicts the time of arrival with an accuracy of 39% for the 10-minute data and 33% for the 1-minute data. When a tolerance window of 20 minutes is allowed, the prediction rates rise to 77% and 92%, respectively. Additional context such as previous locations or times reduces the overall prediction rate to a worst case of 17% for the 10-minute data and 18% for the 1-minute data with the maximum previous context. These results are significantly better than random predictions, which have a prediction rate of 4% for the 10-minute data and 0.07% for the 1-minute data. Support and confidence metrics can be used to communicate the strength of individual predictions.

Chapter 5 Conclusions and Future Work

The previous chapters discuss two types of predictions: predicting *where* someone will be at a given time and predicting *when* someone is likely to be at a given location. A first-order Markov model in which the historical data is represented as *(timestamp, location)* pairs successfully predicts *where* someone will be at a given time. A heuristic model that searches daily sequences can predict *when* someone will be at a given location. These sequences represent the location data as a simple string of context information, such as the date and/or day-of-the-week, followed by all of the time, location sightings for that day. This section concludes the dissertation with a discussion of sparse data, the conclusions, contributions and potential future work.

5.1 Sparse Data

One of the contributions of this work is its ability to work with sparse location data.

This research on location- and temporal-prediction uses location data collected using WiFi access point logs. There are advantages: many devices have WiFi built-in and many buildings have WiFi infrastructure, making a location system based on WiFi relatively inexpensive. The granularity of WiFi location can also be an advantage. WiFi access point location can place someone within a portion of a building. The prediction applications listed in Section 2.1 can be supported with that level of granularity. For example, if the algorithm predicts a professor is in his home building on campus, the user can infer that he is either in his office or his lab. WiFi-based location systems also work indoors, which is where people spend most of their time.

The disadvantage of using WiFi access points for location is that the logs may have time gaps, leading to sparse data, which is one of the problems addressed in this dissertation. Initially, that would seem to be a problem as one would think that continuous, precise data would be the best foundation for a predictive algorithm.

GPS can conceivably provide that continuous data. However, with current technology, GPS can deplete the battery on a smart phone within a matter of hours, which is a source of irritation to users. In addition, GPS provides more detail than the prediction algorithm needs, requiring pre-processing to extract the *significant* locations. However, manual updates provided by location-based online social network sites such as Foursquare, Loopt, and Facebook Places may be a future source of data for predictive algorithms.

Location-based online social networking is rapidly growing and is predicted to attract 82 million subscribers by 2013 [70]. As of March 2011, Foursquare had almost 7.5 million registered users and almost 500 million check-ins in the previous year [71]. These updates may be sparse but they have the advantage of being locations of importance to the user. These updates also include place names. The popularity of these location-based-systems reinforces the effort put into creating prediction algorithms based on sparse data and encourages continued effort in this direction.

5.2 Conclusions

This dissertation presents an analysis and implementation of algorithms to predict people's future locations and the times they are at those future locations. While few people are completely predictable, in general, most people are predictable enough for simple machine learning algorithms to predict their future locations.

This dissertation solves two prediction problems:

- How to predict someone's future location, and
- How to predict when someone will be at a given location in the future.

A first-order Markov model with fallback is an effective predictor for predicting future locations. Instead of representing locations as a sequence of location symbols, a representation of a series of *(timestamp, location)* pairs is an effective way to represent sparse data for predictive models. With this representation and the Markov model, prediction accuracy in the range of 78%-92% was achieved.

The problem of predicting future times is more difficult because of the large number of potential 'answers' (times of the day), and the requirement to be 'inexact' or fuzzy with the data. This dissertation has shown that the same *(timestamp, location)* representation still applies, but a new, heuristic algorithm based on Market Basket Analysis is required. The advantage of the heuristic algorithm is its built-in support for any type of additional context. This algorithm achieved prediction rates of 33%-39% of our UCSD data. If a tolerance of plus/minus twenty minutes is allowed, the prediction rates rise to 77%-91%.

5.3 Contributions

The contributions of this research are:

- **The use of sparse data in location-prediction.** Most existing work uses continuous location data, usually from GPS or cellular operators. However, GPS data are not continuous when the device is indoors, and measurements based on cellular towers use large areas and large time periods. This work takes advantage of the data that are

available using WiFi position logs or, in the future, social-networking sites. These data require less pre-processing and may include useful place names. The requirement to support sparse data inspired the development of a new representation to represent location data as *(timestamp, location)* pairs instead of ordered location sequences.

- **Future location prediction.** Existing research focuses on predicting next location, i.e. the next value in a sequence of locations. Some of these works may predict the next few steps in a trajectory, which is similar to predicting in the future, but only predicts into the near future. This work is the first to take the approach of using time as context to predict out in the future without attempting to predict a location trajectory.
- **Analysis of predictable data.** This dissertation characterizes the historical data to show which features are important for successful predictions. Analysis shows that increasing the number of timestamps in a dataset improves predictions while increasing the number of locations per timestamps degrades predictions. These trends can indicate if the data have the minimum requirements needed to warrant training the model, or can indicate when the model needs to be retrained with recent data.
- **Time of arrival prediction.** This work is the first to predict *when* someone will be at a given location. Temporal predictions support rendezvous and collaboration. As a prediction problem, temporal-prediction is more difficult than location prediction because the number of possible temporal-predictions is much higher than the number of possible location-predictions. A new, heuristic solution for temporal-prediction was successfully developed. Prediction rates range from 33 - 39%, which is better than the rate of 0.1 - 4% achieved with random guesses. If a tolerance of up to 20 minutes is allowed, prediction rates using the SEQ model rise to 77 - 92%.

5.4 Future Work

This dissertation shows the potential of predicting people's future locations and times and develops the algorithms to do the predictions. The application of these algorithms to real world problems introduces several issues that have not been addressed in this initial research. This section will discuss two areas for future work: the features needed to create a useful application of this theory and the ability to measure changes in predictability for anomaly detection.

5.4.1 Requirements for Prediction Applications

Different applications based on predictions will have different requirements for the level of confidence and accuracy required. Some incorrect predictions may be tolerable for a social-networking application that helps you rendezvous with friends. However, an application that provides assistance to someone with a mental disability could cause emotional trauma with incorrect predictions. Applications need to determine if there are enough historical data of the right kind to warrant making a prediction. They need decision rules on whether to reveal a particular prediction to the user based on its confidence and support.

Communicating the results of a prediction is a non-trivial problem. In the case of location-prediction, the result may not be a single predicted location, but multiple locations with their corresponding probabilities, support and confidence measures, which need to be displayed.

People phrase their location differently depending on with whom they are communicating [72, 73]. For example, a person will mention their town's name if they are speaking with someone from out of town but may use a nickname, such as the name of a store or restaurant, if they are talking to a local friend. Therefore, even the name of the location used by the application is an important design decision.

Conclusions and Future Work

Expressing a temporal-prediction is more complicated. As mentioned previously, when the prediction algorithm is asked when someone will be at a given location, the result is not a discrete answer, but a range of results. Figure 5-1 shows an example output. Again, the needs of the application determine which of these results are revealed to the user and how to communicate the probability, confidence and support in each of the predictions.

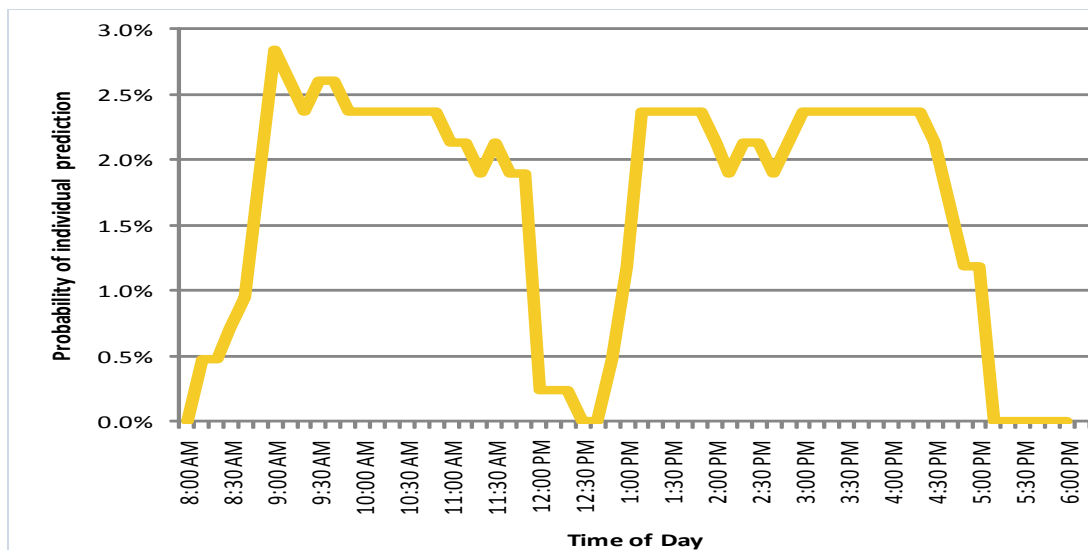


Figure 5-1 A hypothetical temporal prediction

In addition to display options, feedback from the user could be used to tune the algorithm. A user's frustration with incorrect predictions could adjust the application to not reveal a prediction below a certain level of confidence. The user could also indicate additional context that could be used to tune the application. For example, perhaps the model does not recognize that it should re-train due to a semester break. Feedback from the users could adjust the algorithms re-training logic.

5.4.2 Measuring Changes in Predictability

There are two situations in which one wants to know the predictability of a user: before training the model and while using the model. Before training the model, one must know if the quantity and quality of the data are sufficient for prediction. Additionally, the data could be filtered to exclude unusual patterns such as vacations or out-of-date information, such as the previous semester. This *a priori* characterization of the historical data was addressed in section 3.3.5. Changes in predictability that occur in real time while the model is running could signal anomalies, as mentioned in Section 1.1, or could indicate that the model needs re-training with recent data. Entropy, as defined in Section 3.3.4, could be used to signal changes in user patterns. An increase in conditional entropy signals a decrease in someone's predictability.

5.5 Summary

Context-awareness is defined as “an application’s ability to detect and react to environmental variables” [74]. The next step is development of devices and applications that are not only reactive triggers, but can proactively predict future situations. Bo Begole calls this “contextual intelligence” and writes that the result is “services that can help us reach each other at convenient times to talk, conserve energy resources, remember meetings and other important events in our lives, coach us about healthy life choices, and so on [3].” Location and time are important indicators of context, and research in location and temporal prediction is therefore a necessary step toward achieving the goal of contextually intelligent, proactive systems.

Appendix A Investigating the Ping-Pong Effect

The first step in developing a predictive model is to determine the steps required to extract the location features from the raw data. Preliminary studies of the UCSD data indicated that the data included ping-pong effects, which occur when a device is stationary but shifts association between two or more access points in a short period of time. Initially, steps were taken to remove these ping-pong records during pre-processing and are described in this section.

However, the final model negates the ping-pong effect in two ways. First, our 10-minute dataset filters out any records of less than 10 minutes in duration which removes the short ping-pong records. Secondly, the prediction algorithm can check neighboring access points if the initial prediction was incorrect. A ping-pong event, by definition, always happens between neighboring access points, so this check removes the need for ping-pong record removal during the pre-processing step.

The final version of the predictive model does not include any pre-processing to remove ping pong events. The discussion of ping-pong events is included in this appendix for completeness.

When investigating whether the number of ping-pong events was significant, a ping-pong event was initially defined as a set of sessions where a device switched from one AP to another AP and back to the original AP, where each of the sessions began within 30 seconds of the end of the previous session. Two of the ping-pong events for user 23 are indicated by a green background below:

Investigating the Ping-Pong Effect

User	Start Time of Session	Associated Access Point	Duration of Session
23	2002-09-26 01:20:07.00	357	00:05:08.00
23	2002-09-26 01:25:36.00	363	00:00:00.00
23	2002-09-26 01:25:56.00	357	00:06:27.00
23	2002-09-26 01:36:50.00	357	00:02:21.00
23	2002-09-26 01:39:32.00	363	00:00:00.00
23	2002-09-26 01:39:52.00	354	00:00:41.00
23	2002-09-26 01:40:54.00	357	00:16:24.00
23	2002-09-26 01:59:41.00	357	00:05:29.00
23	2002-09-26 02:05:30.00	354	00:00:21.00
23	2002-09-26 02:06:10.00	357	00:33:23.00
23	2002-09-26 12:32:00.00	357	00:00:00.00

Figure A-1 Example of ping-pong events. Records in green indicate a ping-pong between two access points; the records highlighted in blue indicate a ping-pong between three access points.

The events highlighted in blue could also be considered ping-pong events, where the user associated with two different access points before returning to the original access point. In the initial analysis, only single ‘bounces’ between access points were counted and it was found that out of 29,757 transitions between access points, 10,624 (36%) were ping-pong events.

I then looked at the time that was spent at the intermediary access point during the ping-pong event. 5,277 of the 10,624 ping-pong records had durations of 0 seconds, which means that intermediary access point was logged for only one poll.

The following histogram summarizes the amount of time the device spent associated with the intermediary access point.

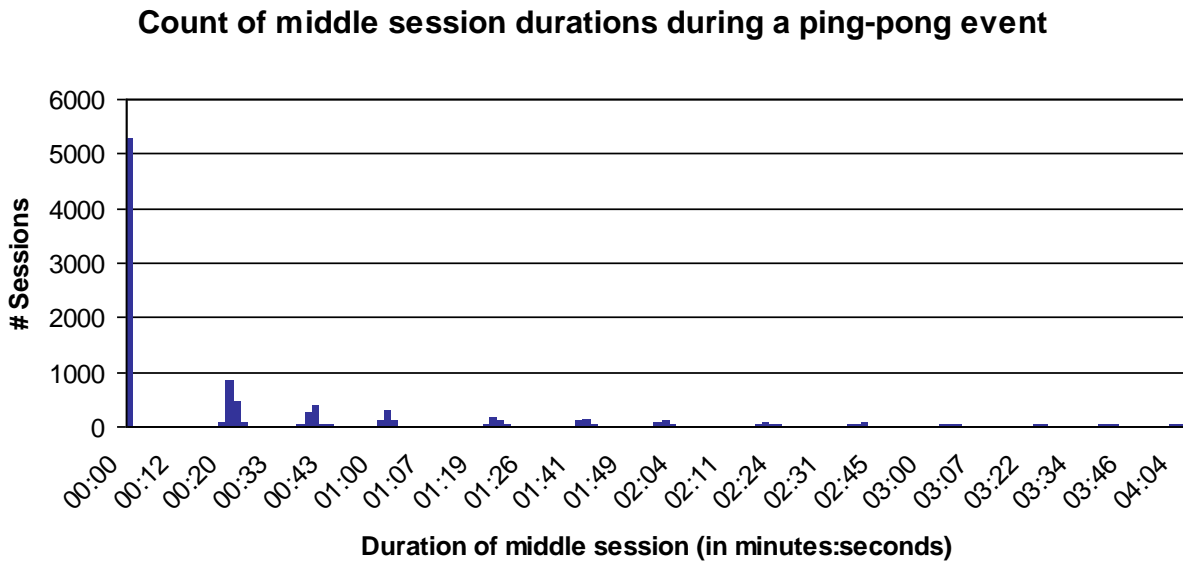


Figure A-2 Duration of the middle session during a ping-pong event

Enforcing a minimum threshold on the session duration could be used to remove ping-pong events. Removing ping-pong sessions with a duration of 1 minute or less removes approximately 75% of the calculated ping-pong events. Increasing the threshold value to 4 minutes removes over 90% of the ping-pong events. Using a time threshold also removes ping-pong events where the association bounces between more than two access points.

The next step was to go through the data records and purge the ping-pong sessions. Instead of simply deleting these short sessions, they were merged, if appropriate, with the sessions before and/or after. The user sessions were inspected three at a time. If the second session had a duration of less than one minute (the threshold value), it was considered a ping-pong record to be removed. These ping-pong records fall into four categories. If the sessions are labeled A, B, and C, with B being the short middle session, B was either deleted or merged with A or C, depending on certain criteria, as shown in Figure 7-3.

Investigating the Ping-Pong Effect

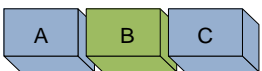
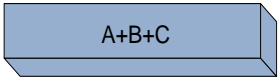

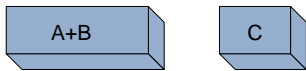

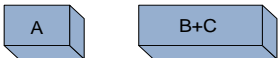
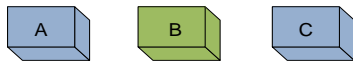

Situation	Resulting Action	# Records which fit this characteristic (out of 94,165 records)
A, B, C are contiguous in time. Ex. 	Merge A, B and C into A 	6892 (7 %)
A and B are contiguous, but there is a time gap before record C. Ex. 	Merge B into A 	2045 (2 %)
B and C are contiguous, with a break after A Ex. 	Merge B and C 	9075 (10 %)
None of the sessions are contiguous Ex. 	Delete record B. 	21641 (23%)

Figure A-3 Purging ping-pong records

Ping-pongs between multiple access points resulted in multiple records being deleted. The resulting data set was 49% of the size of the original dataset.

One more pass was made to the resulting dataset to merge records which were not ping-pongs, but were contiguous. If two consecutive records recorded the same access point, and the second record started within 30 seconds of the first, they were combined. Of 45,990 records, 292 were found to be contiguous and merged.

The result of this data mining is a set of files, one for each of the 275 users, containing a list of the user's sessions, meaning the starting time their device associated with an access point, the ID of the access point and the length of time the device was associated. Ping-pong events and sessions shorter than 30 seconds were removed.

Appendix B Thresholds applied to the Markov Model for predicting time

Section 4.1 applied the Markov model to temporal-prediction. In this appendix, thresholding is applied to the results.

Figure B-1 shows some thresholds that might be used to determine a subset of predictions to be considered. For example, the threshold at 90% would return the prediction that Alice would be likely to be in the office from 8:30am until noon and from 1:10pm until 4:40pm. This broader prediction could be more useful to an application than the single prediction of 9:00am that is returned with a tighter threshold of 0% (which returns the highest probability predictions only).

Thresholding was then implemented in the Markov model. Recall that the model returns a list of predictions, each labeled with its probability. The predictions are sorted in order of probability, from the highest probability to the lowest. To invoke a threshold, the top predictions are checked for correctness until their cumulative probability surpasses the probability threshold. For example, a threshold of 10% uses the top predictions until their cumulative probability is greater than 10%. A threshold of 0% uses only the most likely prediction, which is the same as the ‘best’ predictions in the Figure 4-3. A threshold of 100% uses all of the predictions returned, which is the equivalent of the using all of the probabilities, again as shown in Figure 4-3. Figure B-2 and Figure B-3 illustrate the results of using a probability threshold for the MoveLoc and SigLoc datasets.

Thresholds applied to the Markov Model for predicting time

The appearance of a ‘knee’ in Figure B-2 and Figure B-3 would indicate an optimal threshold value. One can see that here, none exists. The value of the threshold is dependent upon the

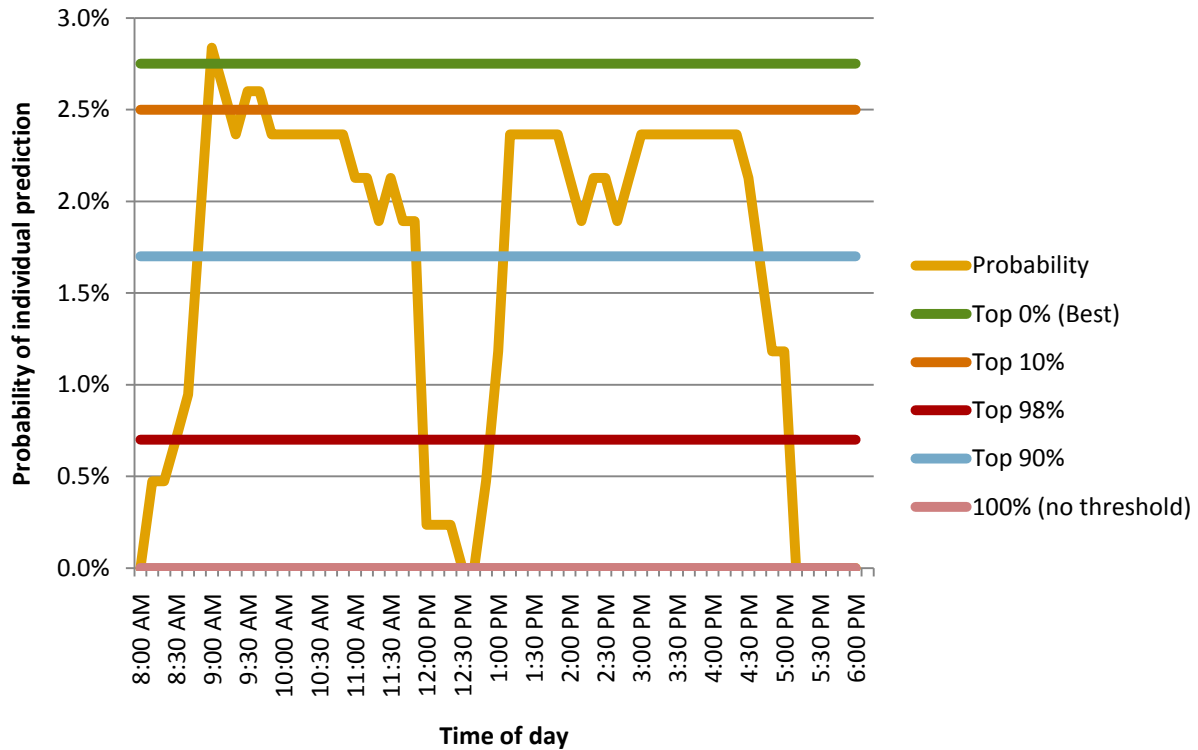


Figure B-1 Thresholds for a hypothetical set of predictions

goals of the application. One application may want only predictions with a high probability while another application may want to present all of the possibilities to the user. Imprecise knowledge of user intent may still be useful information [2]. An application may not need to know exactly when Alice is going to be at the office, but whether she will be there at all.

Thresholds applied to the Markov Model for predicting time

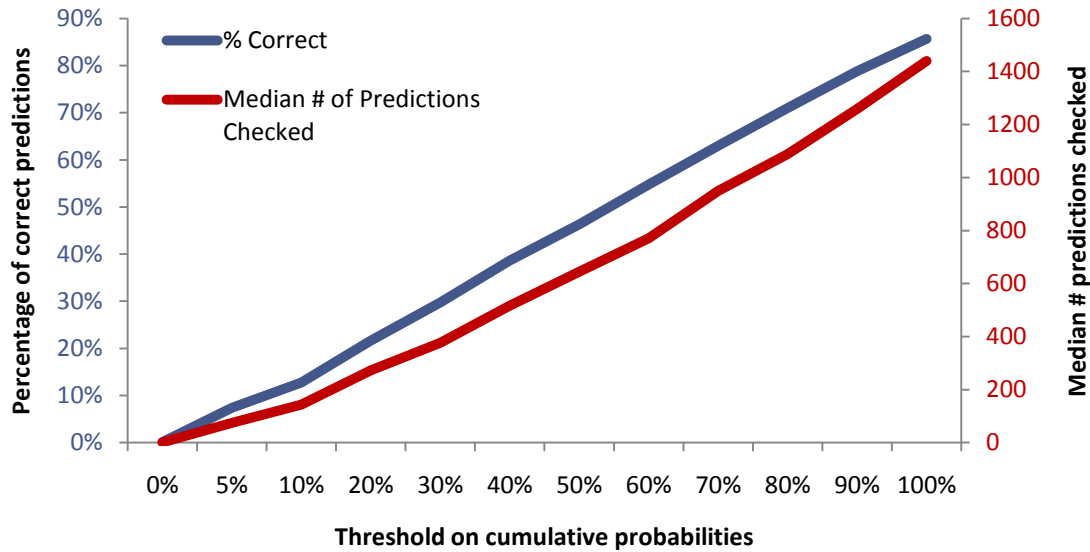


Figure B-2 Applying a probability threshold to time predictions for the 1-minute data

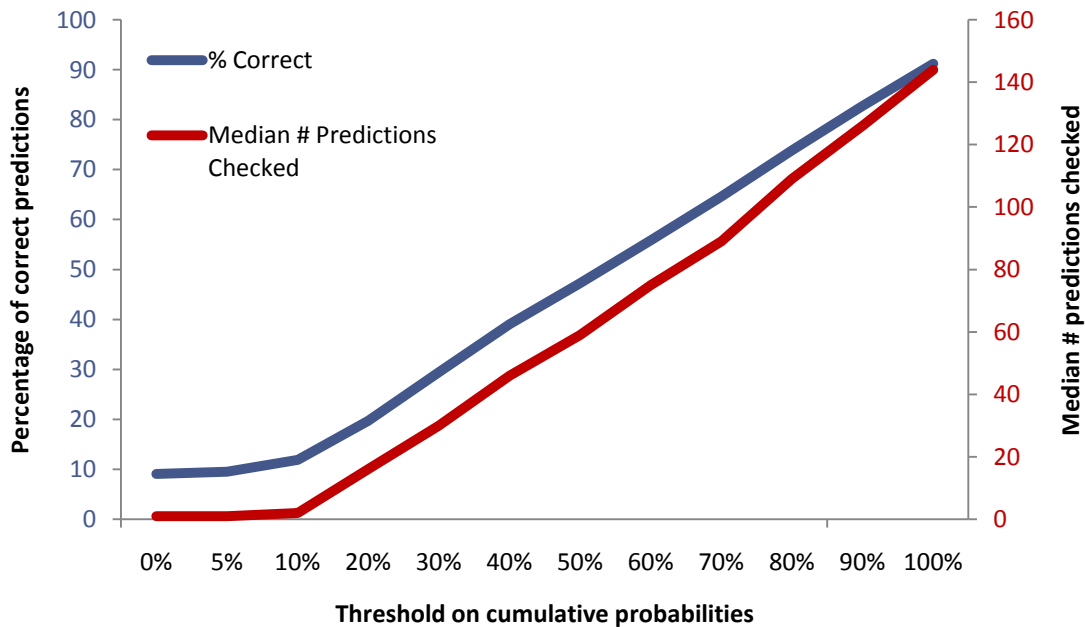


Figure B-3 Applying a probability threshold to time predictions for the 10-minute data

Currently, the tests done on the Markov model do not consider the predictions themselves, only their probabilities. In the future, contiguous predictions should be combined into a window of time. For example, if the returned predictions include {8:00am, 8:10am, 8:20am, ..., 10:00am}, they could be combined into one prediction of 8:00am – 10:00am. When contiguous predictions

Thresholds applied to the Markov Model for predicting time

are combined, there are several issues that must be considered when computing the resulting probability. If Bob only occasionally arrives at the office at 8:00am but he is almost always there at 9:00 am, that information should be clearly presented to the user. In some applications, the less-likely 8:00 am prediction should not be included in the results presented to the user. A proper visualization of the results can illustrate these probabilities, either with a temperature graph which uses different colors for different probabilities or a graph similar to Figure B-1.

Appendix C Additional Graphs of SEQ Model Results

This Appendix contains additional graphs of the normalized distributions of the SEQ Model applied to different orders for each of the datasets (SigLoc and MoveLoc).

SigLoc (10-minute) data

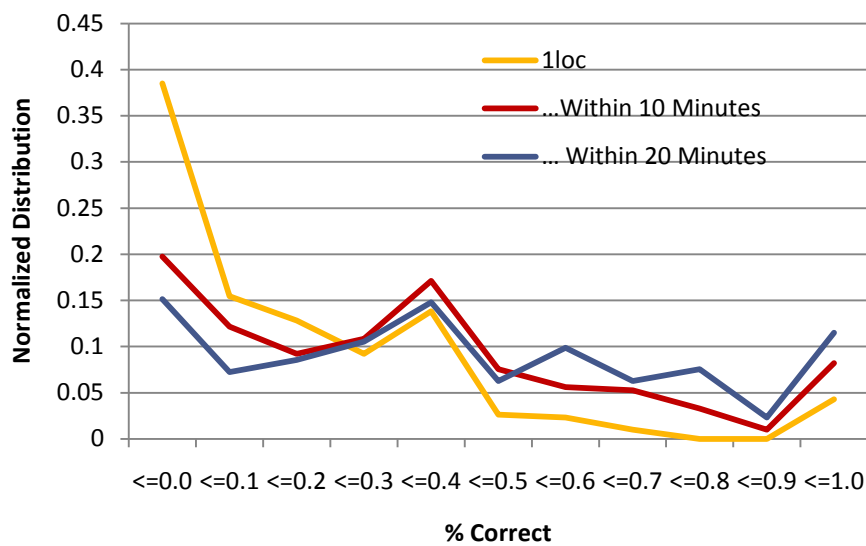


Figure C-1 Normalized distribution of prediction results of the SEQ model applied to the SigLoc data at 1loc order (Context is a previously noted location.)

Additional Graphs of SEQ Model Results

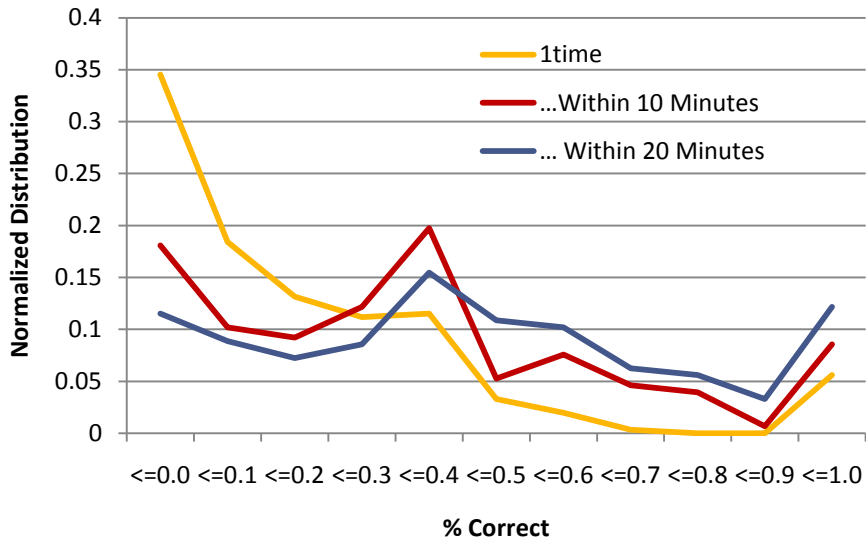


Figure C-2 Normalized distribution of prediction results of the SEQ model applied to the SigLoc data at 1time order (Context is a previously noted time.)

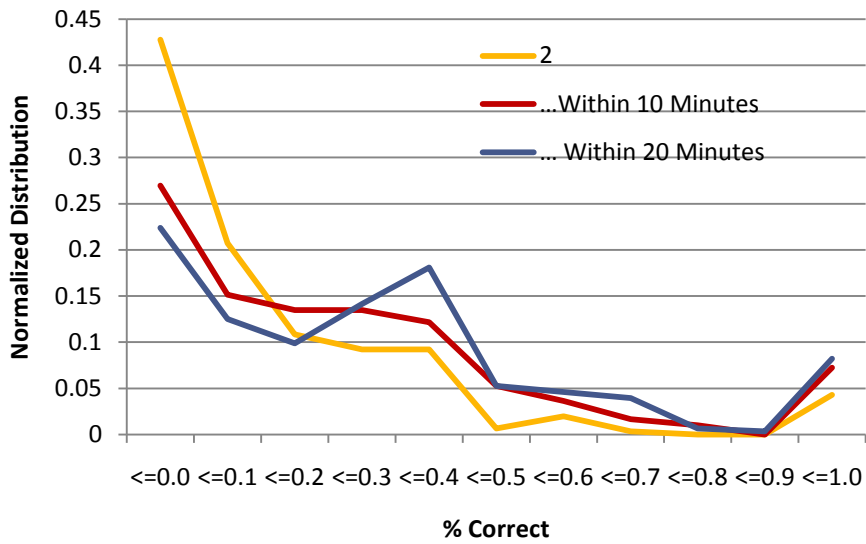


Figure C-3 Normalized distribution of prediction results of the SEQ model applied to the SigLoc data at 2nd order (Context is a previously noted time and location.)

Additional Graphs of SEQ Model Results

MoveLoc (1-minute) data

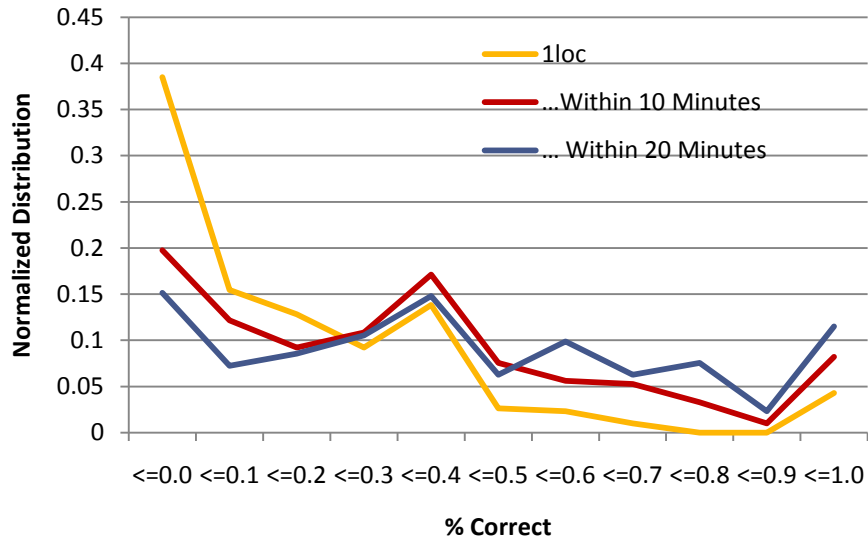


Figure C-4 Normalized distribution of prediction results of the SEQ model applied to the MoveLoc data at 1loc order (Context is a previously noted location.)

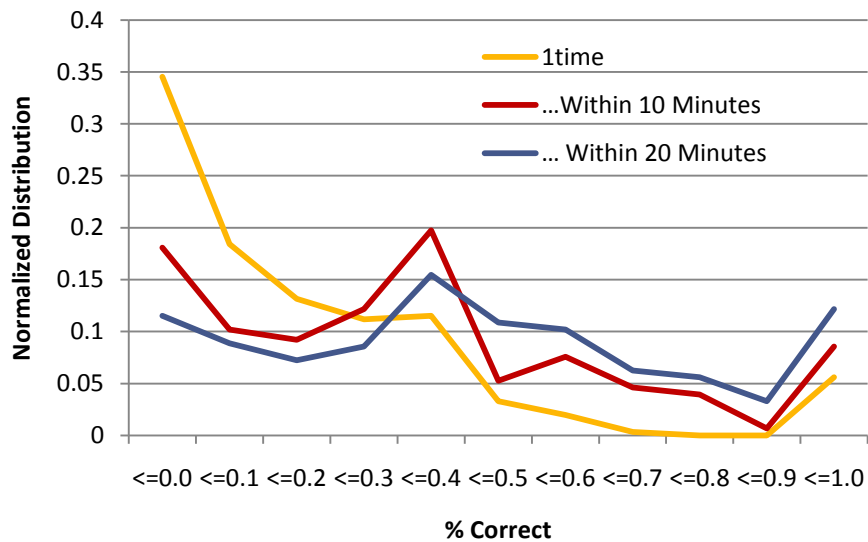


Figure C-5 Normalized distribution of prediction results of the SEQ model applied to the MoveLoc data at 1time order (Context is a previously noted time.)

Additional Graphs of SEQ Model Results

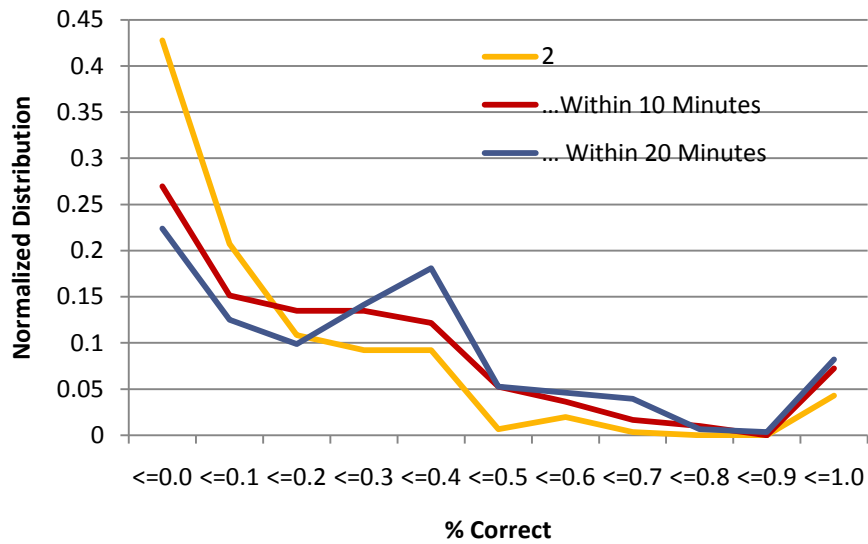


Figure C-6 Normalized distribution of prediction results of the SEQ model applied to the MoveLoc data at 2nd order (Context is a previously noted time and location.)

Appendix D Source Code for the Markov Model

The code for the Markov model is written in C and was developed using the Eclipse IDE.

Portions of the code are adapted from [66]. The following .c and .h files are used to build the Markov model. The output is an executable file called predict_Markov.exe.

The command line has the form predict_Markov -o <order> -f <training file> -p <testing file> -input_type binboxstrings [-v] [-when].

An example is:

```
predict_Markov -o 1 -f
/home/Ingrid/crawdad/ucsd/2010Mar/SigLoc/16bit/002wks2_6.dat -p
/home/Ingrid/crawdad/ucsd/2010Mar/SigLoc/16bit/002wks7_7.dat -
input_type binboxstrings > results.xml
```

The input flags are described below.

Flag	Argument	Description
-v	(none)	Verbose mode. Individual and cumulative results are output to stdout.
-o	Max order (max of 3)	This is the maximum order of the markov model. Because the representation alternates timestamps with locations, this value need to be odd. For the results in this dissertation, the order was set to 1.
-when	(none)	Run the model to predict time instead of location.
-f	Training file name, including path	This argument signifies the file used to train the model.
-p	Testing file name, including path	This argument signifies the file used to test the model.
-input_type	Binboxstrings	Used to indicate which representation is used. Other options include
-logloss	test filename	Calculate average logloss for the given test file.
-c	Confidence_level (-1,0 - 100)	Applicable only to the 'when' case, this argument returns prediction results with a confidence level threshold. A value of -1 ignores the confidence level. Values between 0

Source Code for the Markov Model

		and 100 are used to determine which of the returned predictions to use. 0 means use all predictions. Any other value means use all predictions whose probabilities sum to greater than or equal to the confidence level. Ex. If confidence level is 80% and the predictions returned 75%, 15%, 10%, it would use the first two predictions (sum=90%) but not the third.
--	--	---

I apologize for the lack of elegance in the code.

Source Code for the Markov Model

predict.c

```
/*
 * predict.c
 *
 * This module is the driver program for a variable order
 * finite context compression program. The maximum order is
 * determined by command line option. This particular version
 * also monitors compression ratios, and flushes the model whenever
 * the local (last 256 symbols) compression ratio hits 90% or higher.
 *
 * This code is based on the program comp-2.c, by Bob Nelson.
 * It has been adapted to do predictions instead of compression.
 * It builds the variable order markov model, but doesn't output
 * an encoded bit stream.
 *
 * To build this program, see similar command lines for comp-2.c.
 *
 * Command line options:
 *
 * -f text_file_name [defaults to test.inp]
 * -o order [defaults to 3 for model-2]
 * -logloss test_file_name          # Calculate average-log loss for the given test string.
 * -p test_file_name                # Run a prediction for each char of given test string.
 * -v                               # verbose mode (prints extra info to stdout
 * -delimiters string_of_delimiters # characters in this string are ignored in prediction
results.
 * (The -delimiters option is not supported in the 16bit version)
 * -input_type representation_type # denotes type of input. If verbose is set, outputs
change by representation used.
 * -when                            # if this argument is included, the code will flip
the sequence from a T1,L1,T2,L2 string
 *                                # to L1,T1,L2,T2... (where L=location and
T=time), train the model.
 *                                # This option ASSUMES that the
input_type is binboxstrings.
 * -c confidence_level              # Currently implemented only for the WHEN case.
confidence_level == -1 ignores the
 *                                # the confidence level. Values
between 0 and 100 are used to determine which of the
 *                                # returned predictions to use. 0
means use all predictions. Any other value means
 *                                # use all predictions whose
probabilities sum to greater than or equal to the confidence
 *                                # level. Ex. If confidence level is
80% and the predictions returned 75%, 15%, 10%, it would
 *                                # use the first two predictions
(sum=90%) but not the third.
 *
 *
 * 22Apr2010 ink Number of predictions are written to num_pred.xml
 *                                time deltas (in a wrong prediction, the difference between the
right
 *                                time answer (WHEN) and the predicted time (MostProbables) is
calculated and written
 *                                to time_deltas.xml.
 * 26Apr2010 ink Added the confidence_level option.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>          // for log10() function;
#include <assert.h>        // for assert()
#include "coder.h"
#include "model.h"
```

Source Code for the Markov Model

```
//include <bitio.h>
#include "predict.h"
#include "string16.h"
#include "mapping.h" // for ap mapping, ap neighbors, timeslot mapping
char
str_representations[][21]={"Unknown","Locstrings","Loctimestrings","Boxstrings","Binboxstrings",
"BindOWts"};
char str_mappings[][6] = {"LOC", "STRT", "DUR", "DELIM"};

#define COUNT_NUMBER_OF_PREDICTIONS_RETURNED // to write to num_pred.csv file.
/*
 * The file pointers are used throughout this module.
 */
FILE *training_file; // File containing string to train on.
FILE *test_file; // File to test against (form future predictions)
FILE *num_pred_file; // file to write out the number of predictions
// returned for each test.
FILE *time_deltas_file; // file to write out the time difference between
// time predictions andthe right answer.

char test_file_name[ 81 ];

char verbose = FALSE; // if true, print out lots of info
int confidence_level = -1; // 0 < value < 100, -1 means don't use it.
// confidence_level == 0 means use all
returned predictions. // confidence_level between 0 and 100 means
use most probable // predictions until sum of probabilities >
confidence_level. // current implementation uses
confidence_level in WHEN case only.
//unsigned int str_delimiters[10]; // delimiters to ignore in prediction tests.
int representation; // specified with -input_type argument.
int research_question; // WHERE or WHEN will someone be next.

/*
 * Counters used to count up the results of the predictions
 */
int num_tested=0, num_right=0; // test results
int NumTests = 0;
int FallbackNumCorrect = 0; // number of times it fell to level 0 and was still right.
int FallbackNum = 0; // number of times model went to level 0 for a prediction
//Counters for most likely predictions
int MostProb_NumCorrect = 0; // # times one of the most probably predictions was right
// (also used to count
confidence_level predictions)
int MostProb_NeighborCorrect = 0; // number of times the most prob prediction was wrong, but
one of // it's neighboring APs is
correct answer (WHERE only)
int MostProb_Within10Minutes = 0; // (WHERE case) # times right answer within 10 minutes of
one // of the predictions
int MostProb_Within20Minutes = 0; // (WHERE case) # times right answer is within 20 minutes
of // one of the most likely
predictions
int MostProb_MultiplePredictions = 0; // # times the Most Probable list included > 1 prediction.
// Counters for the less likely predictions
int LessProb_NumCorrect = 0; // # times one of the less probable predictions was right
int LessProb_NeighborCorrect = 0; // number of times one of the less prob prediction was
wrong, but one of // it's neighboring APs is
correct answer (WHERE only)
int LessProb_Within10Minutes = 0; // (WHERE case) # times right answer within 10 minutes of
one // of the predictions
int LessProb_Within20Minutes = 0; // (WHERE case) # times right answer is within 20 minutes
of // one of the most likely
predictions
```

Source Code for the Markov Model

```
int LessProb_MultiplePredictions = 0; // # times the Less Probable list included > 1 prediction.

int number_multiple_predictions = 0; //number fo times model made > 1 prediction for a given
time.
int number_times_neighbors_are_correct = 0; // number of times the prediction is a neighbor of
actual
// Counter for comparison with educated guess
int number_times_guess_is_correct = 0;

STRUCT_PREDICTION pred; // structure containing predictions

/*
 * The main procedure is similar to the main found in COMP-1.C.
 * It has to initialize the coder, the bit oriented I/O, the
 * standard I/O, and the model. It then sits in a loop reading
 * input symbols and encoding them. One difference is that every
 * 256 symbols a compression check is performed. If the compression
 * ratio exceeds 90%, a flush character is encoded. This flushes
 * the encoding model, and will cause the decoder to flush its model
 * when the file is being expanded. The second difference is that
 * each symbol is repeatedly encoded until a succesfull encoding
 * occurs. When trying to encode a character in a particular order,
 * the model may have to transmit an ESCAPE character. If this
 * is the case, the character has to be retransmitted using a lower
 * order. This process repeats until a succesful match is found of
 * the symbol in a particular context. Usually this means going down
 * no further than the order -1 model. However, the FLUSH and DONE
 * symbols do drop back to the order -2 model. Note also that by
 * all rights, add_character_to_model() and update_model() logically
 * should be combined into a single routine.
 */
int main( int argc, char **argv )
{
    SYMBOL_TYPE c, c1, c2;
    int function; // function to perform
    STRING16 * test_string;

    int i; // general purpose register

#ifdef THIS_SPACE_RESERVED_FOR_TEST_CODE
    test_timecode(); // Test routines
    exit( 0 );
#endif

    /* Initialize *****/
    printf("<Run>\n"); // start of XML element
    function = initialize_options( --argc, ++argv );
    initialize_model();
    test_string = string16(MAX_STRING_LENGTH+1);

#ifdef COUNT_NUMBER_OF_PREDICTIONS_RETURNED
    /** Use this code to count the number of predictions returned for each test.
     * They are written into a separate, comma-delimited file.
     */
    num_pred_file = fopen("num_pred.csv", "a"); // should really check for errors
    if (ftell(num_pred_file) == 0) {
        //fprintf(num_pred_file, "<?xml version=\"1.0\" standalone=\"yes\" ?>\n");
        if (confidence_level < 0)
            fprintf(num_pred_file, "test_file_name, num_best_predictions, num_less_predictions,
pred.num_predictions\n");
        else
            fprintf(num_pred_file, "test_file_name, confidence_level, num_conf_predictions,
total_num_predictions\n");
    }
    /*
     * End of code to count number of predictions.
     *****/
#endif

    /* Train the model on the given input training file *****/
    if (research_question == WHERE) {
```

Source Code for the Markov Model

```

    for ( ; ; )
    {
        if (fread(&c, sizeof(SYMBOL_TYPE),1,training_file) == 0)
            c = DONE;
        /* NOTE: fread() seems to skip over whitespace chars, so be careful what's in your
bin file. */
        //printf("Training on 0x%04x\n", c);
        /** The 16-bit version does not currently support delimiter-removal
        ** if (strchr(str_delimiters, c) != NULL) {
        **     //printf("Skipping training on '%c'\n",c);
        **     continue;
        ** }
        go onto the next character
        *****/
        // if ( c == EOF || c == '\n' || c == 0x0a) // skip line feeds
        //     c = DONE;
        clear_current_order();
        if ( c == DONE )
            break;
        update_model( c );
        counters in the level-0 table (I THINK)
        add_character_to_model( c );
    }
} else { // research_question == WHEN
    // This portion of the code ASSUMES that the input string is a 'binbox' representation
    // of the format T1,L1,T2,L2... where Tx is the time for pair x, and Lx is the location
at
    // that time. For the 'when' question, the pairs need to be flipped so that the
sequence
    // looks like L1,T1,L2,T2, etc. Instead of re-processing the input data, I'm going to
read
    // the sequence in pairs and flip them.
    for ( ; ; )
    {
        if (fread(&c1, sizeof(SYMBOL_TYPE),1,training_file) == 0) // Read first char
            c = DONE;
        clear_current_order();
        if ( c == DONE )
            break;
        if (fread(&c2, sizeof(SYMBOL_TYPE),1,training_file) == 0) // Read second char
            c = DONE;
        if ( c == DONE )
            break;

        // Train on second char in pair
        update_model( c2 );
        add_character_to_model( c2 );
        // Train on first char in pair
        clear_current_order();
        update_model( c1 );
        add_character_to_model( c1 );
    }
}

/** Print information about the model */
if (verbose) {
    print_model_allocation();
    print_model();
}

// for this research
question
}
count_model(research_question, verbose); // print out number of tables and
child tables

/*****/

/* Trying some probabilities....
printf("PROBABILITIES\n");
probability( 'r', "ab", verbose);
probability( 'a', "ac", verbose);
probability( 'a', "ad", verbose);

```

Source Code for the Markov Model

```
probability( 'a', "br", verbose);
probability( 'd', "ca", verbose);
probability( 'b', "da", verbose);
probability( 'c', "ra", verbose);
probability( 'b', "a", verbose);
probability( 'c', "a", verbose);
probability( 'd', "a", verbose);
probability( 'r', "b", verbose);
probability( 'a', "c", verbose);
probability( 'a', "d", verbose);
probability( 'a', "r", verbose);
probability( 'a', "", verbose);
probability( 'r', "ac", verbose);
probability( 'b', "a", verbose);
probability( 'r', "ab", verbose);
*8*****/
//      probability( 'V', "01", verbose);

/* Try some predictions
printf("PREDICTIONS\n");

// Try known predictions
predict_next("ab", & pred);
predict_next("br", & pred);
predict_next("a", & pred);
predict_next("b", & pred);
predict_next("", & pred);

// Try unseen context
predict_next("ar", & pred);

// Try an unknown symbol in the context
predict_next("xy", & pred);

*****/

switch (function) {
case PREDICT_TEST:
    // read test file
    i = freadl6( test_string, MAX_STRING_LENGTH, test_file);
    if (i == MAX_STRING_LENGTH)
        fprintf(stderr, "Test String may be over max length and may have been
truncated.\n");
    predict_test(test_string);
    break;
case LOGLOSS_EVAL:
    // read test file
    //fgets(test_string, MAX_STRING_LENGTH, test_file);
    i = freadl6( test_string, MAX_STRING_LENGTH, test_file);
    if (i == MAX_STRING_LENGTH)
        fprintf(stderr, "Test String may be over max length and may have been
truncated.\n");
    printf("%d, %f\n", max_order, compute_logloss(test_string, verbose));
    break;
case NO_FUNCTION:
default:
    break;
}
if (!verbose)
    printf("</Run>\n"); // End of xml element
#ifdef COUNT_NUMBER_OF_PREDICTIONS_RETURNED
    fclose(num_pred_file);
#endif
exit( 0 );
}

/*
* This routine checks for command line options, and opens the
* input and output files. The only other command line option
* besides the input and output file names is the order of the model,
* which defaults to 3.
```

Source Code for the Markov Model

```
*
* Returns the function to perform
*/
int initialize_options( int argc, char **argv )
{
    char training_file_name[ 81 ];
    //char test_file_name[ 81 ];
    int function = NO_FUNCTION;
    char str_type[41];
    char str_dir[41], *str_file = str_dir;
    int temp;

    /* Set Defaults */
#ifdef NOT_USED_IN_16_BIT_VERSION
    str_delimiters[0] = '\0';          // clear delimiter string
#endif
    str_type[0] = '\0';                // clear string_type
    representation = NONE;
    research_question = WHERE;          // 'Where will Bob be at 10:00?' type questions.

    // strcpy( training_file_name, "test.inp" );
    while ( argc > 0 ) {
        // -f <filename> gives the training file name
        if ( strcmp( *argv, "-f" ) == 0 ) {
            argc--;
            strcpy( training_file_name, *++argv );
            if (verbose)
                printf("Training on file %s\n", training_file_name);
        }
        // -p <filename> gives the test filename to predict against
        else if ( strcmp( *argv, "-p" ) == 0 ) {
            argc--;
            strcpy( test_file_name, *++argv );
            test_file = fopen( test_file_name, "rb" );
            if ( test_file == NULL )
            {
                printf( "Had trouble opening the testing file (option -p)\n" );
                exit( -1 );
            }
            setvbuf( test_file, NULL, _IOFBF, 4096 );
            function = PREDICT_TEST;
            if (verbose)
                printf("Testing on file %s\n", test_file_name);
            else
            {
                str_file = strrchr(test_file_name, '/') + 1; // pointer to the filename in string
                printf("    <TestFile>%s</TestFile>\n", str_file);
                strcpy(str_dir, test_file_name);
                str_dir[strlen(str_dir) - strlen(str_file)] = '\0';
                printf("    <SourceDir>%s</SourceDir>\n", str_dir);
            }
        }
        // -o <order>
        else if ( strcmp( *argv, "-o" ) == 0 )
        {
            argc--;
            max_order = atoi( *++argv );
            // IN this version of the code, where we put time in context and
            // ask the model to predict loc (assuming time,loc pairs), the max_order
            // needs to be an odd number.
            if (max_order%2 == 0)
                printf("max_order should be an odd value!\n");
        }
        // -v
        else if ( strcmp( *argv, "-v" ) == 0 )
        {
            verbose = TRUE;          // print out prediction information
        }
        // -logloss <test filename>
        else if ( strcmp( *argv, "-logloss" ) == 0 )
        {
            argc--;
            strcpy( test_file_name, *++argv );
        }
    }
}
```

Source Code for the Markov Model

```

test_file = fopen( test_file_name, "rb");
if ( test_file == NULL )
{
    printf( "Had trouble opening the testing file (option -logloss)\n" );
    exit( -1 );
}
setvbuf( test_file, NULL, _IOFBF, 4096 );
function = LOGLOSS_EVAL;
}
// -c <level> gives a confidence level.
// value needs to be -1 to shut it off or between 0 (use all predictions) and 100.
else if ( strcmp( *argv, "-c" ) == 0 ) {
    argc--;
    temp = atoi( ++argv );
    if (temp > 100) {
        fprintf(stderr, "Confidence level %d is out of range. Should be between 0 and 100
(inclusive) or -1\n", temp);
        fprintf(stderr, "Ignoring the confidence level argument.\n");
        temp = -1;
    }
    else if (temp < 0)
        temp = -1;
    confidence_level = temp;
    if (verbose)
        printf("Confidence level is %d\n", confidence_level);
}
#ifdef DELIMITER_CODE_NOT_SUPPORTED_IN_16BIT_MODEL
// -d <string> ignore prediction of delimiters in string
else if ( strcmp( *argv, "-d" ) == 0 ) {
    argc--;
    strcpy( str_delimiters, ++argv );
    if (verbose)
        printf("Ignoring delimiters \"%s\"\n", str_delimiters);
}
#endif
// -input_type <string_type> Indicate type of input strings used.
// Choices include "locstrings", "boxstrings", "loctimestrings"
else if ( strcmp( *argv, "-input_type" ) == 0 ) {
    argc--;
    strcpy( str_type, ++argv );
    if (strcmp(str_type, "locstrings") == 0)
        representation = LOCSTRINGS;
    else if (strcmp(str_type, "loctimestrings") == 0)
        representation = LOCTIMESTRINGS;
    else if (strcmp(str_type, "boxstrings") == 0)
        representation = BOXSTRINGS;
    else if (strcmp(str_type, "binboxstrings") == 0)
        representation = BINBOXSTRINGS;
    else if (strcmp(str_type, "bindowts") == 0)
        representation = BINDOWTS;
    else
        representation = NONE;
    if (verbose)
        printf("Input string type is %d (%s)\n",
            representation,
            str_representations[ representation]);
}
// -when
else if ( strcmp( *argv, "-when" ) == 0 ) {
    research_question = WHEN;
}
else
{
    fprintf( stderr, "\nUsage: predict [-o order] [-v] [-logloss predictfile] " );
    fprintf( stderr, "[-f text file] [-p predictfile] [-input_type string_type] [-
when]\n" );
    fprintf( stdout, "\nUsage: predict [-o order] [-v] [-logloss predictfile] " );
    fprintf( stdout, "[-f text file] [-p predictfile] [-input_type string_type] [-
when]\n" );
    exit( -1 );
}

```

Source Code for the Markov Model

```

        argc--;
        argv++;
    }
    if (verbose)
        printf("Research question is %s\n", (research_question==WHERE)? "WHERE":"WHEN");
    else
        printf("    <ResearchQuestion>%s</ResearchQuestion>\n", (research_question==WHERE)?
"WHERE":"WHEN");
    training_file = fopen( training_file_name, "rb" );
    if (verbose)
        fprintf(stdout,"%s\n", training_file_name);
    else
        printf("    <TrainingFile>%s</TrainingFile>\n", strrchr(training_file_name, '/')+1);
    if ( training_file == NULL )
    {
        printf( "Had trouble opening the input training file %s!\n", training_file_name );
        exit( -1 );
    }
    // Setup full buffering w/ a 4K buffer. (for speed)
    setvbuf( training_file, NULL, _IOFBF, 4096 );
    setbuf( stdout, NULL );
    return( function );
}

/*****
* predict_test
*
* Given a test string, test each character of the string.
* For example, if the test string is "abc", call
* predict_next() for each substring:
*     predict_next("", &pred);
*     predict_next("a", &pred);
*     predict_next("ab", &pred);
*
* This version (predict_MELT) has been modified to use the
* first character in each pair as context and the second to predict.
* (Of course, this assumes 1st order, and a representation of
* <time,loc> pairs (aka binboxstrings).)
* INPUTS:
*     test_string = pointer to string to test.
*
* RETURNS: nothing
* *****/
void predict_test( STRING16 * test_string){
    int i;                // index into test string
    int length;           // string length
    SYMBOL_TYPE predicted_char;
    STRING16 *str_sub;    // sub-strings (chunks of context)
    // the following are for testing the Markov predictions against educated guessing
    SYMBOL_TYPE fallback_char; // most common location (for comparison against educated
guess)
    STRUCT_PREDICTION pred_fallback;
    STRING16 *nullStr16;

    //printf("Original test string %s\n", format_string16(test_string));
    // initialize
    str_sub = string16(max_order);                // allocate memory for sub-string
    length = strlen16( test_string);
    build_test_string( test_string);              // if WHEN, flip string

    /*****
    * LOOP
    *****/
    // Go through test string, and try to predict every other symbol
    // using the context of the preceding symbol.
    // note that the loop starts with 1 because we are using char 0 for context only,
    // not prediction. (This loop works for higher orders.)

    for (i=max_order; i < length; i+= 2, NumTests++){
        // Copy the symbols preceding the test-symbol into str_sub.
        if (i < max_order)

```


Source Code for the Markov Model

```

        strncpy16( str_sub, test_string, 0, i);          // create test string
    else
        strncpy16( str_sub, test_string, i-max_order, max_order);

    /*****
     * DO THE PREDICTION
     *****/
    predicted_char = predict_next(str_sub, &pred);
    //printf("%s!\n\n", predicted_char == get_symbol( test_string, i) ? "RIGHT" :
"WRONG");

    nullStr16 = string16(1);
    set_strlen16(nullStr16, 0);
    fallback_char = predict_next(nullStr16, &pred_fallback);
    fallback_char = pred_fallback.sym[0].symbol;
    //fallback_char = 0x25FF;          // TEST TEST TEST
    //printf("fallback_char=%x\n", fallback_char);
    /*****
     * Analyze the results
     *****/
    analyze_pred_results( get_symbol(test_string, i), get_symbol(test_string, i-1),
fallback_char);
}
/*****
 * Output the results
 *****/
output_pred_results();
return;
} // end of predict_test

/*****
 * get_char_type
 *
 * Return the type of character just predicted.
 * INPUTS: representation
 *
 *          expected symbol
 *          index into input string
 * OUTPUTS: next_type_index is changed for loctimestrings
 * RETURNS: 0 = Location
 *          1 = Starting Time
 *          2 = Duration
 *          3 = Delimiter
 *****/
int get_char_type( SYMBOL_TYPE symbol, int index_into_input_string)
{
    switch (representation) {
        case LOCSTRINGS:
            return( get_locstring_type( symbol));
        case BOXSTRINGS:
            return( get_boxstring_type( index_into_input_string));
        case LOCTIMESTRINGS:
            return( get_loctimestring_type( symbol));
        case BINBOXSTRINGS:
            return( get_binboxstring_type( symbol));
        case BINDOWTS:
            return( get_bindowts_type( symbol));
        default:
            // If we don't know the string type, we can't figure out the character
type
            return DELIM;
    }
}

/*****
 * get_locstring_type
 * Given the character, return the character type (delimiter or location)
 *
 * INPUTS: symbol in input string
 * OUTPUTS: None
 * RETURNS: DELIM for a delimiter
 *          LOC for a location char
 *****/
int get_locstring_type( SYMBOL_TYPE symbol)

```

Source Code for the Markov Model

```

{
    if (symbol == (SYMBOL_TYPE) ':')
        return (DELIM);
    else
        return (LOC);
}

/*****
 * get_boxstring_type
 * Given the index into the test string, return the character type (delimiter or location)
 *
 * INPUTS: index into input string
 * OUTPUTS: None
 * RETURNS:
 *          LOC for a location char
 *          STRT for a starting time character
 *          DUR for a duration character
 *****/
int get_boxstring_type( int index_into_input_string)
{
    switch (index_into_input_string % 6)
    {
        case 0: case 1:          return( STRT);
        case 2: case 3:          return( LOC);
        case 4: case 5:          return( DUR);
        default:                 // Error!
            printf("Error in get_box_string_type\n");
            return -1;
    }
}

/*****
 * get_loctimestring_type
 * Given the character (symbol),
 * return the character type (delimiter, location, etc)
 *
 * Loctimestrings look like this:
 * L}tt:tt~dd:dd
 * where L is a location, tt:tt is the starting time and dd:dd is the duration.
 * INPUTS: symbol in input string
 *
 * OUTPUTS: None
 * RETURNS: DELIM for a delimiter
 *          LOC for a location char
 *****/
int get_loctimestring_type( SYMBOL_TYPE symbol)
{
    static int next_type_index = 0;
    static int types[] = {LOC, STRT, STRT, STRT, STRT, DUR, DUR, DUR, DUR};
    int result;

    if (symbol == (SYMBOL_TYPE) '}' || symbol == (SYMBOL_TYPE) ':' ||
        symbol == (SYMBOL_TYPE) '~' || symbol == (SYMBOL_TYPE) ';')
        return (DELIM);

    else {
        result = types[ next_type_index];
        next_type_index = (next_type_index + 1) % 9; // point to the next type
        return(result);
    }
}

/*****
 * get_binboxstring_type
 * Given the character, return the character type (delimiter or location)
 *
 * INPUTS: symbol in input string
 * OUTPUTS: None
 * RETURNS: DELIM for a delimiter
 *          LOC for a location char
 *****/

```

Source Code for the Markov Model

```
int get_binboxstring_type( SYMBOL_TYPE symbol)
{
    if (symbol >= INITIAL_START_TIME && symbol <= FINAL_START_TIME)
        return(STRT);
    if (symbol >= INITIAL_DURATION && symbol <= FINAL_DURATION)
        return(DUR);
    if (symbol >= INITIAL_LOCATION && symbol <= FINAL_LOCATION)
        return(LOC);

    // This is an error. We should never get here.
    return(DELMIM);
}
/*****
 * get_bindowts_type
 * Given the character, return the character type (delimiter or location)
 *
 * * The range of times is different for the DOWTS (day-of-week timeslot)
 * symbols.
 * INPUTS: symbol in input string
 * OUTPUTS: None
 * RETURNS: DELIM for a delimiter
 *          LOC for a location char
 *****/
int get_bindowts_type( SYMBOL_TYPE symbol)
{
    if (symbol >= INITIAL_START_TIME && symbol <= 0x25FF)
        return(STRT);
    if (symbol >= 0x2620 && symbol <= 0x26FF)
        return(LOC);

    // This is an error. We should never get here.
    return(DELMIM);
}

#ifdef THIS_IS_THE_CODE_TO_TEST_THE_STRING16_ROUTINES

// TEST STRING16 routines
int main( int argc, char **argv )
{
    STRING16 * str16;
    int i;
    FILE * test_file;
    SYMBOL_TYPE c;

    str16 = string16(MAX_STRING_LENGTH);          // allocate new struct
    // Fill the string
    /*
    for (i = 0; i < 5; i++)
        str16->s[i] = (SYMBOL_TYPE) i + 0x1000;
    str16->s[i] = 0x0;
    str16->length = i+1;

    printf("The string is: %s\n", format_string16( str16));

    for (i=0; i < 5; i++) {
        printf("The symbol at offset %d is 0x%04x.\n", i, get_symbol( str16, i));
    }

    for (i=0; i < 5; i++) {
        shorten_string16( str16);
        printf("After shortening, the string is: %s\n", format_string16( str16));
    }
    */

    test_file = fopen( "108wks01 05.dat", "rb");
    /* This is one way to read in a file: */
    i = fread16( str16, MAX_STRING_LENGTH, test_file);
    printf("The string is: %s\n", format_string16( str16));
    fclose( test_file);
    /**/
}
```

Source Code for the Markov Model

```

/** And this is another */
test_file = fopen( "108wks01_05.dat", "r");
do {
    //fscanf( test_file, "%x", &c);          // read an unsigned short integer (2 bytes)
    i = fread(&c, sizeof(SYMBOL_TYPE),1,test_file);
    if (i>0)
        printf("Training on 0x%04x\n", c);
} while (i > 0);

delete_string16( str16);
exit(0);
}
#endif // STRING16 Test Code

/*****
 *
 * neighboring_ap
 *
 * Given two symbols for locations (AP) return true if
 * the first one is a neighbor of the second.
 *
 *****/
unsigned char neighboring_ap( SYMBOL_TYPE predicted_ap, SYMBOL_TYPE actual_ap)
{
    unsigned int i;
    unsigned int actual_ap_number=0;

    // Translate from the ap symbol value to the actual ap number (1-524)
    // mappings are in the ap_map[] in mapping.h
    for (i=0;i < 525; i++)
        if (ap_map[i] == actual_ap) {
            actual_ap_number = i;
            break;
        }
    if (i == 525) {
        printf("Error: hit end of ap_map looking for 0x%x\n", actual_ap);
        return( FALSE );
    }

    // Look in the ap_neighbors array to see if the predicted_ap is a
    // neighbor of the actual_ap.
    i = 0;
    while (ap_neighbors[ actual_ap_number][i] != 0) {
        //printf("compare 0x%x to 0x%x - ", ap_neighbors[ actual_ap_number][i],
predicted_ap);
        if (ap_neighbors[ actual_ap_number][i] == predicted_ap) {
            //printf("return TRUE\n");
            return (TRUE);
        }
        //else printf("return FALSE\n");
        i++;
    }
    return(FALSE);
} // end of neighboring_ap

/*****
** get_hhmm_from_code
*
* Given a time code (ex. 0x2621), return the time ("00:00")
* INPUTS: code to convert, dest of where to put string
* OUTPUT: dest = string ex "10:23"
* RETURNS: TRUE for success, FALSE for failure (code not found)
*****/
int get_hhmm_from_code( SYMBOL_TYPE code, char * dest) {
    int hours, minutes;
    int i;
    // Find code in timeslot_map
    for (i=0; i < 1441; i++)

```

Source Code for the Markov Model

```

        if (timeslot_map[i] == code)
            break;
    if (i==1441) { // code not found
        printf("mapping.c: timecode 0x%4x not found.\n", code);
        return (FALSE);
    }
    else { // legit value found
        // calculate time: time="00:00" + index
        hours = i/60;
        minutes = i % 60;
        sprintf(dest, "%02d:%02d", hours, minutes);
        return (TRUE);
    }
}

/*****
 * test_timecode
 *
 * routine to test time routines
 *****/
void test_timecode() {
    SYMBOL_TYPE i;
    char s[8]; // time string

    for (i=0x2621; i <= 0x2d94; i++)
        if (get_hhmm_from_code(i, s))
            printf("%s\n", s);
}

/*
 * get_str_mappings
 * Given a mapping (LOC,etc.), return pointer to
 * the string to print.
 *
 * INPUTS: mapping (LOC, STRT...)
 * OUTPUT: pointer to string
 */
char * get_str_mappings(int mapping)
{
    return (str_mappings[mapping]);
}

/*****
 *
 * build_test_string
 *
 * Build test string for testing. If predicting WHERE, leave it alone.
 * If predicting WHEN, flip it to go LTLTLTL instead of TLTLTLTLTL
 * where L=location and T=time
 *
 * INPUTS: test_string = input test string
 * OUTPUTS: If WHEN, test_string has been flipped.
 * RETURNS: nothing
 *****/
void build_test_string(String16 * test_string)
{
    String16 * str_dest;
    int length;
    int i;

    length = strlen16(test_string);
    if (research_question == WHEN) { // allocate memory for actual test string
        str_dest = string16( length);
        for (i=0; i < length; i++)
            if (i%2) { // odd
                //Do this (16bit version): str_dest[i-1] = test_string[i];
                put_symbol(str_dest, i-1, get_symbol(test_string,i));
            }
            else { // even
                //Do this (16bit version): str_dest[i+1] = test_string[i];
                put_symbol(str_dest, i+1, get_symbol(test_string,i));
            }
    }
}

```

Source Code for the Markov Model

```

    }
    set_strlen16(str_dest, i); // terminate the string
    strncpy16( test_string, str_dest, 0, length); // copy into final location
}
if (verbose) {
    printf("Testing on string %s\n", format_string16(test_string));
}
} // end of build_test_string()
/*****
*
* analyze_pred_results
* Look at results and add to counters
* INPUT: correct_answer is the actual result from the test string
* All other inputs come from global variables.
* OUTPUTS: Global counters are incremented.
* RETURNS: void
*
* The pred structure returns ALL predictions (unless the
* code in model-2.c predict_next ()has been changed per the comment).
* If no predictions were made, the model may have resorted
* to falling back to order 0.
* The predictions that were made are listed in order from
* the most likely to the least likely. I want to first look
* at only the most likely results and count their stats and
* then look at the other, less likely results.
* GLOBAL: num_pred_file is file to output the number of predictions returned for each test.
* confidence_level is used to determine which predictions to use (WHEN case only)
* 1Feb2011 Included the fallback_answer (which is just the most probable location) to compare
results.
*****/
void analyze_pred_results( SYMBOL_TYPE correct_answer, SYMBOL_TYPE context, SYMBOL_TYPE
fallback_answer)
{
    int j; // counter into number of predictions
    unsigned char predicted_correctly; // true if one of the predictions is correct
    unsigned char is_neighbor = FALSE; // true if two APs are neighbors
    char str_time[8], str_time2[8]; // space to format time strings
    unsigned char bool_MultipleBest = FALSE; // true if > 1 most likely predictions
    unsigned char bool_MultipleLess = FALSE; // true if > 1 less likely predictions
    unsigned char is_within_10min = FALSE; // true if one of the predictions is
within 10 minutes
    unsigned char is_within_20min = FALSE; // true if one of the predictions is
within 20 minutes
    int num_best_predictions = 0; // number of most likely predictions
    int num_less_predictions = 0; // number of less than best predictions
    int index_last_best = 0; // number of predictions -1 that are
'most likely'
    int best_count; // numerator for most
likely predictions
    // Variables used for confidence_level approach
    float f_confidence; // confidence_level expressed
as a value between 0 and 1.
    float f_prob_sum; // sum of prediction
probabilities
    float f_previous_prob; // probability of previous prediction
    float f_current_prob; // probability of current prediction
    int i_current_numerator, i_prev_numerator; // numerator of current and previous
probabilities
    unsigned char bool_done = FALSE; // true to break out of confidence_level
loop.

    if (verbose) { // Print output
        printf("context,expected symbol, predicted symbol, # predictions, order, probability\n");

        for (j=0; j < pred.num_predictions; j++) {
            if (research_question == WHERE) {
                // test for pred.depth prevents printing out entire FALLBACK table
                // If the prediction is from falling back to 0, don't bother with
it
                // if it's returning a location instead of a time. (Remember, the
model doesn't

```

Source Code for the Markov Model

```

        // know there's a diff)
        if (pred.depth==0)
            if (get_char_type( pred.sym[j].symbol, 0) != LOC)
                continue;

    // this is a TIME, go onto the next LOC prediction.
    get_hhmm_from_code(context, str_time );
    //assert(get_char_type(correct_answer, 0)==LOC);
    //assert(get_char_type(pred.sym[j].symbol, 0) == LOC);
    printf("%s, 0x%04x, 0x%04x, %d, %d, %f, %s, %s, 0x%04x\n",
        str_time,
        correct_answer,
        pred.sym[j].symbol,
        //
        pred.num_predictions,
        pred.depth,
        // depth
        (float)
        pred.sym[j].prob_numerator/pred.prob_denominator,
        (correct_answer == pred.sym[j].symbol)? "CORRECT" :
        "--",
        (correct_answer == fallback_answer)? "GUESS_CORRECT"
        : "--",
        fallback_answer);
    }
    else //research_question == WHEN
    {
        // If the prediction is from falling back to 0, don't bother with
        it
        // if it's returning a location instead of a time. (Remember, the
        model doesn't
        // know there's a diff)
        if ((pred.depth==0))
            if (get_char_type( pred.sym[j].symbol, 0) == LOC)
                continue;

        // this is a LOC, go onto the next TIME prediction.
        get_hhmm_from_code(correct_answer, str_time );
        // convert
        expected symbol to time
        get_hhmm_from_code(pred.sym[j].symbol, str_time2);
        // convert prediction into a time
        printf("0x%04x, %s, %s, %d, %d, %f, %s\n",
            context,
            str_time,
            str_time2,
            // location
            // expected symbol
            // predicted symbol
            pred.num_predictions,
            pred.depth,
            //
            depth
            (float) pred.sym[j].prob_numerator/pred.prob_denominator,
            (correct_answer == pred.sym[j].symbol)? "CORRECT" : "--");
    }
} // end if verbose
/*****
 * Check if any predictions were returned.
 *****/
if (pred.num_predictions == 0) // no predictions were returned!
    return;

/*****
 * Check for fallback to 0 order.
 *****/
// Check for the fallback situation, where the model fell back to order 0
if (pred.depth == 0) { //Model fell back to 0th-order
    FallbackNum++;
    // See if best one of the fallback predictions is correct.
    for (j=0; j < pred.num_predictions; j++) { // Why did j start at
1 instead of a 0? I think that was a bug.
        if (pred.sym[j].prob_numerator == pred.sym[0].prob_numerator) {
            if (correct_answer == pred.sym[j].symbol) {

```

Source Code for the Markov Model

```

FallbackNumCorrect++;
//number_times_guess_is_correct++;           // count as an
educated guess
break;
}
}
return; // Don't need to check any farther.
} // End of fallback case.

/*****
 * Check types of predictions (MostLikely or LessLikely)
 *****/
// The first element in the pred structure is the most likely.
// It's count value is the numerator of it's probability, so
// any entries with the same count have the same probability.
best_count = pred.sym[0].prob_numerator; // highest count (probability)
index_last_best = 0; // assume only one
prediction.
num_best_predictions = 1;

// Do a quick check to see if more than one prediction was returned
// as the most likely.
// (index starts at 1 because we already know that [0] is the most likely)
for (j=1; j < pred.num_predictions; j++) {
    if (pred.sym[j].prob_numerator == best_count) {
        bool_MultipleBest = TRUE;
        index_last_best = j;
        num_best_predictions++;
    }
    else if (j > (index_last_best+1)) {
        bool_MultipleLess = TRUE; //found > 1 less likely prediction.
        num_less_predictions++;
    }
    else // this case for the first less likely
        num_less_predictions++;
}
if (bool_MultipleBest)
    MostProb_MultiplePredictions++;
if (bool_MultipleLess)
    LessProb_MultiplePredictions++;

if (confidence_level == -1 || research_question == WHERE) {
#ifdef COUNT_NUMBER_OF_PREDICTIONS_RETURNED
/*****
 * Output the number of predictions to a file.
 *****/
//fprintf(num_pred_file, " <Test value=\"%d\">\n", num_tested);
//fprintf(num_pred_file, " <NumBestPred>%d</NumBestPred>\n",
num_best_predictions);
//fprintf(num_pred_file, " <NumLessPred>%d</NumLessPred>\n",
num_less_predictions);
//fprintf(num_pred_file, " </Test>\n");
fprintf(num_pred_file, "%s, %d, %d, %d\n", test_file_name, num_best_predictions,
num_less_predictions, pred.num_predictions);
#endif

/*****
 * Check to see if an educated guess (the fallback answer) would have been
correct.

 * Recall that the fallback answer (the 0th order answer) is just the most
 * common location or time.
 *****/
if (correct_answer == fallback_answer)
    number_times_guess_is_correct++;

/*****
 * Check predictions for correctness or if they are close to correct.
 *****/
// Check the most likely predictions first.
predicted_correctly = FALSE; // Assume they are all wrong.
for (j=0; (j < index_last_best+1) && (!predicted_correctly); j++) {

```


Source Code for the Markov Model

```

        if (correct_answer == pred.sym[j].symbol) {
            predicted_correctly = TRUE;
            MostProb_NumCorrect++;
        }
    }
    if (!predicted_correctly) {
        is_neighbor = FALSE;
        is_within_10min = FALSE;
        is_within_20min = FALSE;
        // All the most likely predictions are wrong. See if some were close.
        for (j=0; j < index_last_best; j++) {
            // this prediction is wrong. See if it is close.
            if (research_question == WHERE) {
                is_neighbor = neighboring_ap( pred.sym[j].symbol,
correct_answer);

                if (is_neighbor)
                    break;
            }
            else { //research_question is WHEN
                is_within_10min = within_time_window( pred.sym[j].symbol,
correct_answer, 10);

                if (!is_within_20min)
                    // stop checking to see if one is within 20 minutes
if you already found one.
                    is_within_20min = within_time_window(
pred.sym[j].symbol, correct_answer, 20);
                if (is_within_10min)
                    break;
            }
        }
        // Update the counters for the MostProbable predictions
        if (is_neighbor)
            MostProb_NeighborCorrect++;
        if (is_within_10min)
            MostProb_Within10Minutes++;
        if (is_within_20min)
            MostProb_Within20Minutes++;
    }

    //Now check less likely predictions
    predicted_correctly = FALSE; // assume they are all wrong.
    for (j=index_last_best+1; (j < pred.num_predictions) && !predicted_correctly; j++)
    {
        if (correct_answer == pred.sym[j].symbol) {
            predicted_correctly = TRUE;
            LessProb_NumCorrect++;
        }
    }
    if (!predicted_correctly) {
        // All the less likely predictions are wrong. See if some were close.
        is_neighbor = FALSE;
        is_within_10min = FALSE;
        is_within_20min = FALSE;
        for (j=index_last_best+1; j < pred.num_predictions; j++) {
            // this prediction is wrong. See if it is close.
            if (research_question == WHERE) {
                is_neighbor = neighboring_ap( pred.sym[j].symbol,
correct_answer);

                if (is_neighbor)
                    break;
            }
            else { //research_question is WHEN
                is_within_10min = within_time_window( pred.sym[j].symbol,
correct_answer, 10);

                if (!is_within_20min)
                    // stop checking to see if one is within 20 minutes
if you already found one.
                    is_within_20min = within_time_window(
pred.sym[j].symbol, correct_answer, 20);
                if (is_within_10min)
                    break;
            }
        }
    }
}

```

Source Code for the Markov Model

```

    }
    // Update the counters for the LessProbable predictions
    if (is_neighbor)
        LessProb_NeighborCorrect++;
    if (is_within_10min)
        LessProb_Within10Minutes++;
    if (is_within_20min)
        LessProb_Within20Minutes++;
    }
    // end of checking predictions without using confidence level
else // use confidence level to determine which predictions to use.
{
    f_confidence = (float) confidence_level/100.0; // convert to
a value between 0 and 1 (inclusive)
    assert (f_confidence >= 0 && f_confidence <= 1); // check range
    predicted_correctly = FALSE; // Assume they are all wrong.
    bool_done = FALSE; // true if
confidence-level has been reached.
    f_previous_prob = 0.0; // probability of
previous prediction
    f_prob_sum = 0.0; // total
probability of the predictions used.
    i_prev_numerator = 0;

    /*****
    * The predictions are returned in order of highest probability, where the
probability
    * is calculated by dividing the numerator by the denominator.
    * We want to check all predictions until we hit the confidence level specified by
the user.
    * For example, if the confidence level is 75% and three predictions are returned
with probabilities
    * of 85%, 15% and 5%, we would use the first two (to get at least the top 75% of
predictions) and not the third.
    * If there are a bunch of predictions with the same probability, we check them
all, even if that takes us
    * over the threshold.
    *****/
    j=0;
    //printf("correct, prediction, prob, total_prob, confidence = %f\n",
f_confidence);
    // go through each prediction, stopping if we hit the confidence level or
    // if we hit a correct prediction.
    while ((j < pred.num_predictions) && (!bool_done)){
        // calculate probability of this prediction
        f_current_prob = (float) pred.sym[j].prob_numerator/ (float)
pred.prob_denominator;
        i_current_numerator = pred.sym[j].prob_numerator;
        // if its the same as the previous prediction or if we are below the
        // confidence threshold, check to see if its correct.
        if ((i_current_numerator == i_prev_numerator) || (f_prob_sum <=
f_confidence)) {
            //if ((f_current_prob == f_previous_prob) || (f_prob_sum <= f_confidence))
            {
                //printf(">>>");
                if (correct_answer == pred.sym[j].symbol) {
                    predicted_correctly = TRUE;
                    MostProb_NumCorrect++;
                }
            }
            //else
            //printf(" ");
            // add this prediction to the totals
            f_prob_sum += f_current_prob;
            if ((f_prob_sum > f_confidence) && (i_current_numerator !=
i_prev_numerator)) {
                //printf("bool_done = TRUE\n");
                bool_done = TRUE;
            }
        }
    }
}

```

Source Code for the Markov Model

```

        f_previous_prob = f_current_prob;
        i_prev_numerator = i_current_numerator;
        //printf("0x%x, 0x%x, %f, %f, %f, %s\n", correct_answer,
pred.sym[j].symbol, f_current_prob, f_prob_sum, f_confidence, (correct_answer ==
pred.sym[j].symbol)? "CORRECT" : "-----");
        j++;
    }
#ifdef COUNT_NUMBER_OF_PREDICTIONS_RETURNED
    /****
    * Output the number of predictions to a file.
    *****/
    //fprintf(num_pred_file, " <Test value=\"%d\">\n", num_tested);
    //fprintf(num_pred_file, " <NumConfPred>%d</NumConfPred>\n", j-1);
    //fprintf(num_pred_file, " </Test>\n");
    fprintf(num_pred_file, "%s, %d, %d, %d\n", test_file_name, confidence_level, j,
pred.num_predictions);
#endif

    } // end of confidence level tests.

} // end of analyze_results
/*****
* output_pred_results
* if verbose, display the results in words
* else in XML.
* INPUTS: global counters
* OUTPUTS: none
* RETURNS: void
*****/
void output_pred_results()
{
    if (verbose) {
        /* Print only the percentage of pairs correct & percentage when time is correct */
        printf("NumTests=%d, FallbackNumCorrect=%d, Number_fallbacks_to_zero=%d,
numCorrectGuesses=%d\n",
            NumTests, // number of tests
            FallbackNumCorrect, // number of times it fell back to level 0
            // but was still correct
            FallbackNum, // number of times it fell back to 0, wrong or right
            prediction
            number_times_guess_is_correct); // compare to fallback answer even if
a prediction was made
    }
    else {
        // Overall Results
        //printf(" <MaxOrder>%d</MaxOrder>\n", max_order);
        printf(" <NumTests>%d</NumTests>\n", NumTests);
        // Results for predictions that went to Fallback
        printf(" <FallbackNum>%d</FallbackNum>\n", FallbackNum);
        printf(" <FallbackNumCorrect>%d</FallbackNumCorrect>\n", FallbackNumCorrect);
        if (confidence_level < 0) { // normal output
            /****
            * Results for most likely predictions
            *****/
            printf(" <MostProb_NumCorrect>%d</MostProb_NumCorrect>\n",
MostProb_NumCorrect);
            if (research_question == WHERE)
                printf("
<MostProb_NeighborCorrect>%d</MostProb_NeighborCorrect>\n", MostProb_NeighborCorrect);
            else // research_question == WHEN
            {
                printf("
<MostProb_Within10Minutes>%d</MostProb_Within10Minutes>\n", MostProb_Within10Minutes);
                printf("
<MostProb_Within20Minutes>%d</MostProb_Within20Minutes>\n", MostProb_Within20Minutes);
            }
            printf("
<MostProb_MultiplePredictions>%d</MostProb_MultiplePredictions>\n",
MostProb_MultiplePredictions);
            /****
            * Results for less likely predictions

```

Source Code for the Markov Model

```

        *****/
        printf("    <LessProb_NumCorrect>%d</LessProb_NumCorrect>\n",
LessProb_NumCorrect);
        if (research_question == WHERE)
            printf("
<LessProb_NeighborCorrect>%d</LessProb_NeighborCorrect>\n", LessProb_NeighborCorrect);
        else // research_question == WHEN
        {
            printf("
<LessProb_Within10Minutes>%d</LessProb_Within10Minutes>\n", LessProb_Within10Minutes);
            printf("
<LessProb_Within20Minutes>%d</LessProb_Within20Minutes>\n", LessProb_Within20Minutes);
        }
        printf("
<LessProb_MultiplePredictions>%d</LessProb_MultiplePredictions>\n",
LessProb_MultiplePredictions);
        /*****
        * Results for educated guessing (using the fallback result as the
prediction)
        *****/
        if (research_question == WHERE)
            printf("    <correctGuesses>%d</correctGuesses>\n",
number_times_guess_is_correct);
        } // end of normal output
        else { // using confidence level
            /****
            * Results for most likely predictions
            *****/
            printf("    <ConfidenceLevel>%d</ConfidenceLevel>\n", confidence_level);
            printf("    <ConfidenceLevel_NumCorrect>%d</ConfidenceLevel_NumCorrect>\n",
MostProb_NumCorrect);
        }
    }
} // end of output_pred_results
/*****
* within_time_window
*
* Compare two time codes (symbols) and see if they are within
* the given range.
* INPUTS: time1 = 16bit code for a time
*         time2 = 16bit code for another time
*         range = number of MINUTES to see if they are in range.
* RETURNS: TRUE if they are in range (|symbol2 - symbol1| <= range)
*         FALSE if they are not or an error occurred.
*****/
unsigned char within_time_window( SYMBOL_TYPE time1, SYMBOL_TYPE time2, int range)
{
    int ts1=0, ts2=0, i;
    // Find codes in timeslot_map
    for (i=0; i < 1441; i++)
        if (timeslot_map[i] == time1) {
            ts1 = i;
            break;
        }
    if (i==1441) { // code not found
        printf("within_time_window: timecode 0x%4x not found.\n", time1);
        return (FALSE);
    }
    for (i=0; i < 1441; i++)
        if (timeslot_map[i] == time2) {
            ts2 = i;
            break;
        }
    if (i==1441) { // code not found
        printf("within_time_window: timecode 0x%4x not found.\n", time2);
        return (FALSE);
    }
    return (abs(ts2-ts1)<= range);
}

```

Source Code for the Markov Model

```
}
```

model-2.c

```
/*
 * Listing 12 -- model-2.c
 *
 * This module contains all of the modeling functions used with
 * comp-2.c and expand-2.c. This modeling unit keeps track of
 * all contexts from 0 up to max_order, which defaults to 3.
 * In addition, there is a special context -1 which is a fixed model
 * used to encode previously unseen characters, and a context -2
 * which is used to encode EOF and FLUSH messages.
 *
 * Each context is stored in a special CONTEXT structure, which is
 * documented below. Context tables are not created until the
 * context is seen, and they are never destroyed.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>      // for isprint() declaration
#include <math.h>        // for log10() function;
#include "coder.h"
#include "model.h"
#include "string16.h" // for handling 16-bit char 'strings'
#include "predict.h"  // for printing symbol type
/*
 * max_order is the maximum order that will be maintained by this
 * program. EXPAND-2 and COMP-2 both will modify this int based
 * on command line parameters.
 */
int max_order=3;
/*
 * *contexts[] is an array of current contexts. If I want to find
 * the order 0 context for the current state of the model, I just
 * look at contexts[0]. This array of context pointers is set up
 * every time the model is updated.
 */
CONTEXT **contexts;
/*
 * current_order contains the current order of the model. It starts
 * at max_order, and is decremented every time an ESCAPE is sent. It
 * will only go down to -1 for normal symbols, but can go to -2 for
 * EOF and FLUSH.
 */
int current_order;
/*
 * This variable tells COMP-2.C that the FLUSH symbol can be
 * sent using this model.
 */
int flushing_enabled=0;
/*
 * This table contains the cumulative totals for the current context.
 * Because this program is using exclusion, totals has to be calculated
 * every time a context is used. The scoreboard array keeps track of
 * symbols that have appeared in higher order models, so that they
 * can be excluded from lower order context total calculations.
 */
short int totals[ RANGE_OF_SYMBOLS+2 ];
char scoreboard[ RANGE_OF_SYMBOLS ];
int alloc_count=0;      // number of CONTEXT structs allocated.
/*
 * Global variables for routine that counts the tables once
 * the model is built. (I know, they shouldn't be global, but
 * it's the quickest way to implement, and this is grad student
 * code, not a professional release.
 */
```

Source Code for the Markov Model

```

*/
int num_context_tables=0;
int num_children[1500];          // number of children in each context table

/*
 * Local procedure declarations.
 */
void error_exit( char *message );
void update_table( CONTEXT *table, SYMBOL_TYPE symbol );
void rescale_table( CONTEXT *table );
void totalize_table( CONTEXT *table );
CONTEXT *shift_to_next_context( CONTEXT *table, SYMBOL_TYPE c, int order);
CONTEXT *allocate_next_order_table( CONTEXT *table,
                                   SYMBOL_TYPE symbol,
                                   CONTEXT *lesser_context );

/*
 * This routine has to get everything set up properly so that
 * the model can be maintained properly. The first step is to create
 * the *contexts[] array used later to find current context tables.
 * The *contexts[] array indices go from -2 up to max_order, so
 * the table needs to be fiddled with a little. This routine then
 * has to create the special order -2 and order -1 tables by hand,
 * since they aren't quite like other tables. Then the current
 * context is set to \0, \0, \0, ... and the appropriate tables
 * are built to support that context. The current order is set
 * to max_order, the scoreboard is cleared, and the system is
 * ready to go.
 */

void initialize_model()
{
    int i;
    CONTEXT *null_table;
    CONTEXT *control_table;

    current_order = max_order;
    contexts = (CONTEXT **) calloc( sizeof( CONTEXT * ), 10 );
    alloc_count += 10;
    if ( contexts == NULL )
        error_exit( "Failure #1: allocating context table!" );
    contexts += 2;
    null_table = (CONTEXT *) calloc( sizeof( CONTEXT ), 1 );
    alloc_count++;
    if ( null_table == NULL )
        error_exit( "Failure #2: allocating null table!" );
    null_table->max_index = -1;
    contexts[ -1 ] = null_table;
    for ( i = 0 ; i <= max_order ; i++ )
        contexts[ i ] = allocate_next_order_table( contexts[ i-1 ],
                                                    0,
                                                    contexts[ i-1 ] );

    alloc_count += max_order;
    handle_free( (char __handle *) null_table->stats );
    null_table->stats =
        (STATS __handle *) handle_calloc( sizeof( STATS ) * 500 ); // HERE was 256
    if ( null_table->stats == NULL )
        error_exit( "Failure #3: allocating null table!" );
    null_table->max_index = 499; // TEST TEST TEST was 256
    for ( i=0 ; i < 500 ; i++ ) // TEST TEST was 255
    {
        null_table->stats[ i ].symbol = (unsigned char) i;
        null_table->stats[ i ].counts = 1;
    }

    control_table = (CONTEXT *) calloc( sizeof(CONTEXT), 1 );
    if ( control_table == NULL )
        error_exit( "Failure #4: allocating null table!" );
    alloc_count++;
    control_table->stats =
        (STATS __handle *) handle_calloc( sizeof( STATS ) * 2 );

```

Source Code for the Markov Model

```
if ( control_table->stats == NULL )
    error_exit( "Failure #5: allocating null table!" );
contexts[-2] = control_table;
control_table->max_index = 1;
control_table->stats[ 0 ].symbol = -FLUSH;
control_table->stats[ 0 ].counts = 1;
control_table->stats[ 1 ].symbol = -DONE;
control_table->stats[ 1 ].counts = 1;

clear_scoreboard();
}
/*
 * This is a utility routine used to create new tables when a new
 * context is created. It gets a pointer to the current context,
 * and gets the symbol that needs to be added to it. It also needs
 * a pointer to the lesser context for the table that is to be
 * created. For example, if the current context was "ABC", and the
 * symbol 'D' was read in, add_character_to_model would need to
 * create the new context "BCD". This routine would get called
 * with a pointer to "BC", the symbol 'D', and a pointer to context
 * "CD". This routine then creates a new table for "BCD", adds it
 * to the link table for "BC", and gives "BCD" a back pointer to
 * "CD". Note that finding the lesser context is a difficult
 * task, and isn't done here. This routine mainly worries about
 * modifying the stats and links fields in the current context.
 */

CONTEXT *allocate_next_order_table( CONTEXT *table,
                                   SYMBOL_TYPE symbol,
                                   CONTEXT *lesser_context )
{
    CONTEXT *new_table;
    int i;
    unsigned int new_size;

    for ( i = 0 ; i <= table->max_index ; i++ )
        if ( table->stats[ i ].symbol == symbol )
            break;
    if ( i > table->max_index )
    {
        table->max_index++;
        new_size = sizeof( LINKS );
        new_size *= table->max_index + 1;
        if ( table->links == NULL )
            table->links = (LINKS __handle *) handle_calloc( new_size );
        else
            table->links = (LINKS __handle *)
                handle_realloc( (char __handle *) table->links, new_size );
        new_size = sizeof( STATS );
        new_size *= table->max_index + 1;
        if ( table->stats == NULL )
            table->stats = (STATS __handle *) handle_calloc( new_size );
        else
            table->stats = (STATS __handle *)
                handle_realloc( (char __handle *) table->stats, new_size );
        if ( table->links == NULL )
            error_exit( "Failure #6: allocating new table" );
        if ( table->stats == NULL )
            error_exit( "Failure #7: allocating new table" );
        table->stats[ i ].symbol = symbol;
        table->stats[ i ].counts = 0;
    }
    new_table = (CONTEXT *) calloc( sizeof( CONTEXT ), 1 );
    alloc_count++;
    if ( new_table == NULL )
        error_exit( "Failure #8: allocating new table" );
    new_table->max_index = -1;
    table->links[ i ].next = new_table;
    new_table->lesser_context = lesser_context;
    return( new_table );
}
```

Source Code for the Markov Model

```
/*
 * This routine is called to increment the counts for the current
 * contexts. It is called after a character has been encoded or
 * decoded. All it does is call update_table for each of the
 * current contexts, which does the work of incrementing the count.
 * This particular version of update_model() practices update exclusion,
 * which means that if lower order models weren't used to encode
 * or decode the character, they don't get their counts updated.
 * This seems to improve compression performance quite a bit.
 * To disable update exclusion, the loop would be changed to run
 * from 0 to max_order, instead of current_order to max_order.
 ***
 *** Ingrid: For example, with his code and the training string
 * 'abracadabra' the counts end up as 'a'=4, 'b'=1, 'r'=1, 'c'=1, 'd'=1
 *
 */
void update_model( SYMBOL_TYPE symbol )
{
    int local_order;

    if ( current_order < 0 )
        local_order = 0;
    else
        local_order = current_order;

    //Ingrid: Override his code
    local_order = 0;          // Ingrid
    // End of override changes

    if ( symbol >= 0 )
    {
        while ( local_order <= max_order )
        {
            if ( symbol >= 0 )
                update_table( contexts[ local_order ], symbol );
            local_order++;
        }
        current_order = max_order;
        clear_scoreboard();
    }
}
/*
 * This routine is called to update the count for a particular symbol
 * in a particular table. The table is one of the current contexts,
 * and the symbol is the last symbol encoded or decoded. In principle
 * this is a fairly simple routine, but a couple of complications make
 * things a little messier. First of all, the given table may not
 * already have the symbol defined in its statistics table. If it
 * doesn't, the stats table has to grow and have the new guy added
 * to it. Secondly, the symbols are kept in sorted order by count
 * in the table so as that the table can be trimmed during the flush
 * operation. When this symbol is incremented, it might have to be moved
 * up to reflect its new rank. Finally, since the counters are only
 * bytes, if the count reaches 255, the table absolutely must be rescaled
 * to get the counts back down to a reasonable level.
 */
void update_table( CONTEXT *table, SYMBOL_TYPE symbol )
{
    int i;
    int index;
    SYMBOL_TYPE temp;
    CONTEXT *temp_ptr;
    unsigned int new_size;

    /*
     * First, find the symbol in the appropriate context table. The first
     * symbol in the table is the most active, so start there.
     */
    index = 0;
```


Source Code for the Markov Model

```

while ( index <= table->max_index &&
        table->stats[index].symbol != symbol )
    index++;
if ( index > table->max_index )
{
    table->max_index++;
    new_size = sizeof( LINKS );
    new_size *= table->max_index + 1;
    if ( current_order < max_order )
    {
        if ( table->max_index == 0 )
            table->links = (LINKS __handle *) handle_calloc( new_size );
        else
            table->links = (LINKS __handle *)
                handle_realloc( (char __handle *) table->links, new_size );
        if ( table->links == NULL )
            error_exit( "Error #9: reallocating table space!" );
        table->links[ index ].next = NULL;
    }
    new_size = sizeof( STATS );
    new_size *= table->max_index + 1;
    if (table->max_index==0)
        table->stats = (STATS __handle *) handle_calloc( new_size );
    else
        table->stats = (STATS __handle *)
            handle_realloc( (char __handle *) table->stats, new_size );
    if ( table->stats == NULL )
        error_exit( "Error #10: reallocating table space!" );
    table->stats[ index ].symbol = symbol;
    table->stats[ index ].counts = 0;
}
/*
 * Now I move the symbol to the front of its list.
 */
i = index;
while ( i > 0 &&
        table->stats[ index ].counts == table->stats[ i-1 ].counts )
    i--;
if ( i != index )
{
    temp = table->stats[ index ].symbol;
    table->stats[ index ].symbol = table->stats[ i ].symbol;
    table->stats[ i ].symbol = temp;
    if ( table->links != NULL )
    {
        temp_ptr = table->links[ index ].next;
        table->links[ index ].next = table->links[ i ].next;
        table->links[ i ].next = temp_ptr;
    }
    index = i;
}
/*
 * The switch has been performed, now I can update the counts
 */
table->stats[ index ].counts++;
//if ( table->stats[ index ].counts == 255 )    // Ingrid: removed this - it sets level 0
counts to 0
//rescale_table( table );    // HERE these two lines were commented out until I hit a
large file.
}

/*****
 * clear_current_order
 *
 * Used for during training.
 *
 * INPUTS: None
 * OUTPUTS: current_order set to 0
 * RETURNS: void
 *****/
void clear_current_order(){

```

Source Code for the Markov Model

```
        current_order = 0;
        return;
    }
/*
 * This routine is called when a given symbol needs to be encoded.
 * It is the job of this routine to find the symbol in the context
 * table associated with the current table, and return the low and
 * high counts associated with that symbol, as well as the scale.
 * Finding the table is simple. Unfortunately, once I find the table,
 * I have to build the table of cumulative counts, which is
 * expensive, and is done elsewhere. If the symbol is found in the
 * table, the appropriate counts are returned. If the symbol is
 * not found, the ESCAPE symbol probabilities are returned, and
 * the current order is reduced. Note also the kludge to support
 * the order -2 character set, which consists of negative numbers
 * instead of unsigned chars. This insures that no match will every
 * be found for the EOF or FLUSH symbols in any "normal" table.
 */
int convert_int_to_symbol( SYMBOL_TYPE c, SYMBOL *s )
{
    int i;
    CONTEXT *table;

    table = contexts[ current_order ];
    totalize_table( table );
    s->scale = totals[ 0 ];
    if ( current_order == -2 )
        c = -c;
    for ( i = 0 ; i <= table->max_index ; i++ )
    {
        if ( c == table->stats[ i ].symbol )
        {
            if ( table->stats[ i ].counts == 0 )
                break;
            s->low_count = totals[ i+2 ];
            s->high_count = totals[ i+1 ];
            return( 0 );
        }
    }

    s->low_count = totals[ 1 ];
    s->high_count = totals[ 0 ];

    current_order--;
    return( 1 );
}

/*
 * This routine is called when decoding an arithmetic number. In
 * order to decode the present symbol, the current scale in the
 * model must be determined. This requires looking up the current
 * table, then building the totals table. Once that is done, the
 * cumulative total table has the symbol scale at element 0.
 */
void get_symbol_scale( SYMBOL *s )
{
    CONTEXT *table;

    table = contexts[ current_order ];
    totalize_table( table );
    s->scale = totals[ 0 ];
}

/*
 * This routine is called during decoding. It is given a count that
 * came out of the arithmetic decoder, and has to find the symbol that
 * matches the count. The cumulative totals are already stored in the
 * totals[] table, from the call to get_symbol_scale, so this routine
 * just has to look through that table. Once the match is found,
 * the appropriate character is returned to the caller. Two possible
 * complications. First, the character might be the ESCAPE character,
```

Source Code for the Markov Model

```
* in which case the current_order has to be decremented. The other
* complication is that the order might be -2, in which case we return
* the negative of the symbol so it isn't confused with a normal
* symbol.
*/
/**** not used in prediction *****/
int convert_symbol_to_int( int count, SYMBOL *s)
{
    int c;
    CONTEXT *table;

    table = contexts[ current_order ];
    for ( c = 0; count < totals[ c ] ; c++ )
        ;
    s->high_count = totals[ c-1 ];
    s->low_count = totals[ c ];
    if ( c == 1 )
    {
        current_order--;
        return( ESCAPE );
    }
    if ( current_order < -1 )
        return( (int) -table->stats[ c-2 ].symbol );
    else
        return( table->stats[ c-2 ].symbol );
}
/*****/

/*
* After the model has been updated for a new character, this routine
* is called to "shift" into the new context. For example, if the
* last context was "ABC", and the symbol 'D' had just been processed,
* this routine would want to update the context pointers to that
* contexts[1]=="D", contexts[2]=="CD" and contexts[3]=="BCD". The
* potential problem is that some of these tables may not exist.
* The way this is handled is by the shift_to_next_context routine.
* It is passed a pointer to the "ABC" context, along with the symbol
* 'D', and its job is to return a pointer to "BCD". Once we have
* "BCD", we can follow the lesser context pointers in order to get
* the pointers to "CD" and "C". The hard work was done in
* shift_to_context().
*/
void add_character_to_model( SYMBOL_TYPE c )
{
    int i;

    if ( max_order < 0 || c < 0 )
        return;
    contexts[ max_order ] =
        shift_to_next_context( contexts[ max_order ],
                               c, max_order );
    for ( i = max_order-1 ; i > 0 ; i-- )
        contexts[ i ] = contexts[ i+1 ]->lesser_context;
}

/*
* This routine is called when adding a new character to the model. From
* the previous example, if the current context was "ABC", and the new
* symbol was 'D', this routine would get called with a pointer to
* context table "ABC", and symbol 'D', with order max_order. What this
* routine needs to do then is to find the context table "BCD". This
* should be an easy job, and it is if the table already exists. All
* we have to in that case is follow the back pointer from "ABC" to "BC".
* We then search the link table of "BC" until we find the link to "D".
* That link points to "BCD", and that value is then returned to the
* caller. The problem crops up when "BC" doesn't have a pointer to
* "BCD". This generally means that the "BCD" context has not appeared
* yet. When this happens, it means a new table has to be created and
* added to the "BC" table. That can be done with a single call to
* the allocate_new_table routine. The only problem is that the
* allocate_new_table routine wants to know what the lesser context for
```

Source Code for the Markov Model

```
* the new table is going to be. In other words, when I create "BCD",
* I need to know where "CD" is located. In order to find "CD", I
* have to recursively call shift_to_next_context, passing it a pointer
* to context "C" and they symbol 'D'. It then returns a pointer to
* "CD", which I use to create the "BCD" table. The recursion is guaranteed
* to end if it ever gets to order -1, because the null table is
* guaranteed to have a for every symbol to the order 0 table. This is
* the most complicated part of the modeling program, but it is
* necessary for performance reasons.
*/
CONTEXT *shift_to_next_context( CONTEXT *table, SYMBOL_TYPE c, int order)
{
    int i;
    CONTEXT *new_lesser;

/*
* First, try to find the new context by backing up to the lesser
* context and searching its link table. If I find the link, we take
* a quick and easy exit, returning the link. Note that there is a
* special Kludge for context order 0. We know for a fact that
* the lesser context pointer at order 0 points to the null table,
* order -1, and we know that the -1 table only has a single link
* pointer, which points back to the order 0 table.
*/
    table = table->lesser_context;
    if ( order == 0 )
        return( table->links[ 0 ].next );
    for ( i = 0 ; i <= table->max_index ; i++ )
        if ( table->stats[ i ].symbol == c ) {
            if ( table->links[ i ].next != NULL )
                return( table->links[ i ].next );
            else
                break;
        }

/*
* If I get here, it means the new context did not exist. I have to
* create the new context, add a link to it here, and add the backwards
* link to *his* previous context. Creating the table and adding it to
* this table is pretty easy, but adding the back pointer isn't. Since
* creating the new back pointer isn't easy, I duck my responsibility
* and recurse to myself in order to pick it up.
*/
    new_lesser = shift_to_next_context( table, c, order-1 );

/*
* Now that I have the back pointer for this table, I can make a call
* to a utility to allocate the new table.
*/
    table = allocate_next_order_table( table, c, new_lesser );
    return( table );
}

/*
* Rescaling the table needs to be done for one of three reasons.
* First, if the maximum count for the table has exceeded 16383, it
* means that arithmetic coding using 16 and 32 bit registers might
* no longer work. Secondly, if an individual symbol count has
* reached 255, it will no longer fit in a byte. Third, if the
* current model isn't compressing well, the compressor program may
* want to rescale all tables in order to give more weight to newer
* statistics. All this routine does is divide each count by 2.
* If any counts drop to 0, the counters can be removed from the
* stats table, but only if this is a leaf context. Otherwise, we
* might cut a link to a higher order table.
*/
void rescale_table( CONTEXT *table )
{
    int i;

    if ( table->max_index == -1 )
        return;
    for ( i = 0 ; i <= table->max_index ; i++ )
```

Source Code for the Markov Model

```
        table->stats[ i ].counts /= 2;
if ( table->stats[ table->max_index ].counts == 0 &&
    table->links == NULL )
{
    while ( table->stats[ table->max_index ].counts == 0 &&
        table->max_index >= 0 )
        table->max_index--;
    if ( table->max_index == -1 )
    {
        handle_free( (char __handle *) table->stats );
        table->stats = NULL;
    }
    else
    {
        table->stats = (STATS __handle *)
            handle_realloc( (char __handle *) table->stats,
                sizeof( STATS ) * ( table->max_index + 1 ) );
        if ( table->stats == NULL )
            error_exit( "Error #11: reallocating stats space!" );
    }
}
}

/*
 * This routine has the job of creating a cumulative totals table for
 * a given context. The cumulative low and high for symbol c are going to
 * be stored in totals[c+2] and totals[c+1]. Locations 0 and 1 are
 * reserved for the special ESCAPE symbol. The ESCAPE symbol
 * count is calculated dynamically, and changes based on what the
 * current context looks like. Note also that this routine ignores
 * any counts for symbols that have already showed up in the scoreboard,
 * and it adds all new symbols found here to the scoreboard. This
 * allows us to exclude counts of symbols that have already appeared in
 * higher order contexts, improving compression quite a bit.
 */
void totalize_table( CONTEXT *table )
{
    int i;
    unsigned char max;
    // int num_excluded_symbols = 0; // Ingrid - count excluded symbols

    for ( ; ; )
    {
        max = 0;
        i = table->max_index + 2;
        totals[ i ] = 0;
        for ( ; i > 1 ; i-- )
        {
            totals[ i-1 ] = totals[ i ];
            if ( table->stats[ i-2 ].counts )
                if ( ( current_order == -2 ) ||
                    scoreboard[ table->stats[ i-2 ].symbol - LOWEST_SYMBOL ] == 0 )
                    totals[ i-1 ] += table->stats[ i-2 ].counts;
            if ( table->stats[ i-2 ].counts > max )
                max = table->stats[ i-2 ].counts;
        }
    }

    /*
     * Here is where the escape calculation needs to take place.
     */

    if ( max == 0 )
        totals[ 0 ] = 1;
    else
    {
        /* Ingrid - I commented out this code because I'm not sure what he's doing!
         * (and it gives me incorrect results for logloss numbers)
         */
        totals[ 0 ] = (short int) ( 256 - table->max_index );
        totals[ 0 ] *= table->max_index;
        totals[ 0 ] /= 256;
        totals[ 0 ] /= max;
    }
}
```

Source Code for the Markov Model

```

*          totals[ 0 ]++;
*          totals[ 0 ] += totals[ 1 ];
***** end of original code */
          /* Start of Ingrid's code */
          if (current_order == 0)
              totals[0] = totals[1] + table->max_index;
          else
              totals[0] = totals[1] + table->max_index + 1;
          /* End of Ingrid's changes */
      }
      if ( totals[ 0 ] < MAXIMUM_SCALE )
          break;
      rescale_table( table );
      print a message if it does.)
  }
  for ( i = 0 ; i < table->max_index ; i++ )
      // Ingrid - changed to <= (was <)
      // Careful: if it runs through the whole loop it will cause an ACCESS_VIOLATION
      if (table->stats[i].counts != 0) {
          // This is a bug fix hack -- don't know why we can sometimes get a table where
          this is not true:
          if (table->stats[i].symbol >= LOWEST_SYMBOL)
              scoreboard[ table->stats[ i ].symbol - LOWEST_SYMBOL ] = 1;
          printf("i=%d, max_index=%d, brackets=%d, max=%d\n", i, table->max_index, table-
>stats[ i ].symbol - LOWEST_SYMBOL, RANGE_OF_SYMBOLS);
      }
  }

/*
* This routine is called when the entire model is to be flushed.
* This is done in an attempt to improve the compression ratio by
* giving greater weight to upcoming statistics. This routine
* starts at the given table, and recursively calls itself to
* rescale every table in its list of links. The table itself
* is then rescaled.
*/
void recursive_flush( CONTEXT *table )
{
    int i;

    if ( table->links != NULL )
        for ( i = 0 ; i <= table->max_index ; i++ )
            if ( table->links[ i ].next != NULL )
                recursive_flush( table->links[ i ].next );
    rescale_table( table );
}

/*
* This routine is called to flush the whole table, which it does
* by calling the recursive flush routine starting at the order 0
* table.
*/
void flush_model()
{
    recursive_flush( contexts[ 0 ] );
}

void error_exit( char *message)
{
    putc( '\n', stdout );
    puts( message );
    exit( -1 );
}

/*
* count_model
*
* This routine is called to count the number of tables in the model.
* research_question = WHEN or WHERE (so we know what to count)
* verbose = TRUE to printf verbose output, rather than comma-sep values.
*/
void count_model(int research_question, int verbose)

```

Source Code for the Markov Model

```

{
    int i;
#ifdef ADD_IF_YOU_WANT_MAX_MIN
    int min, max;
#endif
    float total, average;
#ifdef ADD_IF_NEEDED
    float std_dev, median, var;
#endif
#ifdef ADD_THIS_IN_IF_YOU_WANT_IT
    int jj, max_count, max_index;
    int count[MAX_MODEL_COUNT];           //working array to hold codes for calculating mode.
    int done;
    int multimodal;                       // true if number of children has more than one mode.
#endif
    // count tables and children for this research_question.
    recursive_count(0, contexts[0], research_question, verbose);
    //printf("num_context_tables = %d\n", num_context_tables);
    //for (i=0; i <= num_context_tables; i++)
        //printf("num_children[%d]=%d\n", i, num_children[i]);

#ifdef ADD_IF_YOU_WANT_MAX_MIN
    // Find min, max
    min=10000;
    max = 0;
    for (i=1; i <= num_context_tables; i++){
        if (num_children[i] < min)
            min = num_children[i];
        if (num_children[i] > max)
            max = num_children[i];
    }
    // The tables are sorted by count, so the max number of children is always
    // the first value.
    if (verbose)
        printf("max num_children=%d\n", max);
    else
        printf("    <MaxNumChildren>%d</MaxNumChildren>\n", max);

    if (verbose)
        printf("min num_children=%d\n", min);
    else
        printf("    <MinNumChildren>%d</MinNumChildren>\n", min);
#endif

    // Calculate average number of children
    for (i=1, total=0; i <= num_context_tables; i++)
        total += num_children[i];
    average = total/num_context_tables;
    if (verbose) {
        printf("total num_children=%f, num_context_tables=%d\n", total,
num_context_tables);
        printf("average num_children=%f\n", average);
    }
    else{
        printf("    <AveNumChildren>%.2f</AveNumChildren>\n", average);
        printf("    <tableSize>%.0f</tableSize>\n", total);
    }
#ifdef ADD_IF_YOU_WANT_STD_DEVIATION
    // Calculate standard deviation
    for (i=1, total=0; i <= num_context_tables; i++ ) {
        var = num_children[i] - average;
        var *= var;                       //square
        total += var;
    }
    std_dev = sqrt(total/num_context_tables);
    if (verbose)
        printf("standard deviation=%f\n", std_dev);
    else
        printf("    <StdDevChildren>%.2f</StdDevChildren>\n", std_dev);
#endif
}

```

Source Code for the Markov Model

```

// Calculate median (middle value or arithmetic mean of two middle values)
if (num_context_tables % 2)
    // odd number of entries, median = middle value
    median = num_children[(num_context_tables+1)/2];
else
    // even number of entries, take ave of middle two values
    median = (num_children[num_context_tables/2] +
              num_children[num_context_tables/2+1])/2;
if (verbose)
    printf("median = %f\n", median);
else
    printf("    <MedianNumChildren>%.2f</MedianNumChildren>\n", median);
#endif

#ifdef ADD_THIS_IN_IF_YOU_WANT_IT
// Calculate mode (most common value).
// This part of the code assumes that the values are in order.
i=0;
max_count=0;
max_index = 0;
jj=1;
done = FALSE;
while (! done) {
    // count the number of times each value appears and
    // stick the count in the corresponding box in the count[] array.
    // (ex. if the first 5 values are the same, count[1]=count[2]=count[3]=count[4]=0
    // count[5] = 5
    while (++i <= num_context_tables &&
           (i < MAX_MODEL_COUNT) &&
           num_children[i] == num_children[i-1])
    {
        count[i-1]=0;
        jj++;
        // increment counter of how many times this value
        appears
    }
    if (i == MAX_MODEL_COUNT) {
        fprintf(stderr, "Hit an error calculating mode.\n");
        break;
    }
    // hit a spot where the value is different
    count[i-1] = jj;
    // store count at index where value can be found.
    // Is this the max we've seen so far?
    if (jj > max_count) {
        max_index = i-1;
        max_count = jj;
    }
    jj=1;
    if (i > num_context_tables)
        done = TRUE;
}
if (verbose)
    printf("Mode is %d.\n", num_children[ max_index ]);
else
    printf("    <ModeChildren>%d</ModeChildren>\n", num_children[max_index]);
// TODO: Add check for bimodal!
// Go back through table. if you find another value with the same count,
// output 'T' for 'multimodal?' variable, else output FALSE
for (multimodal=FALSE, i=1; i <= num_context_tables; i++) {
    // if there's a value with the same count and a different index,
    // then there is more than one mode.
    if (count[i] == max_count && i != max_index)
        multimodal = TRUE;
}
if (verbose)
    printf("Multimodal = %s\n", multimodal == TRUE? "true" : "false");
else
    printf("    <MultimodalChildren>%s</MultimodalChildren>\n", multimodal == TRUE?
    "True" : "False");
#endif

```


Source Code for the Markov Model

```

}

/*
 * print_model
 *
 * This routine is called when the entire model is to be printed.
 */
void print_model()
{
    recursive_print( 0, contexts[ 0 ] );
}
/*
 * recursive_count
 * Walk through the tables, and print what you find.
 * Count the given table, and then count it's children.
 *
 * Inputs: depth = level (how many tabs to print)
 *         table = points to the table to print
 *
 * NOTE that this routine assumes only 1st order tables
 * and a binboxstrings representation.
 */
void recursive_count(int depth, CONTEXT *table, int research_question, int verbose )
{
    int i, type;
    char tabs[] = "\t\t\t\t\t";
    char str_time[8]; // room to format time as hh:mm
    // the following variables are now globals.
    //static int num_context_tables=0;
    //static int num_children[200]; // number of children in each context table

    /* print tabs to create nested table */
    tabs[ depth+1 ] = '\0'; // create tab string */

    /* Print this table's information */
    if (verbose)
        printf("Depth: %d, %d child(ren)\n",
            depth,
            table->max_index+1);
    else
        if (depth==0)
            //printf(" <TotalNumChildren>%d</TotalNumChildren>\n",table-
>max_index+1);

    if (table->max_index == -1)
    {
        return;
    }
    if (depth > 0)
        num_children[ num_context_tables ] = table->max_index+1;

    for (i=0; i <= table->max_index; i++)
    {
        type = get_binboxstring_type( table->stats[i].symbol);
        /* If this table has links, print them */
        if (( table->links != NULL && depth < max_order ) &&
            ((research_question == WHEN && type == LOC) ||
            (research_question == WHERE && type == STRT))){
            if (verbose) {
                if (type == STRT)
                {
                    get_hhmm_from_code(table->stats[i].symbol, str_time );
                    printf("%sContext %s %s: ", tabs, str_time,
get_str_mappings(type));
                }
                else
                    printf("%sContext 0x%04x %s: ", tabs, table->stats[i].symbol,
get_str_mappings(type));
            }
            num_context_tables++;
            recursive_count( depth+1, table->links[i].next, research_question, verbose);

```

Source Code for the Markov Model

```

    }
    }
    //printf("num_context_tables=%d\n", num_context_tables);
}

/*
 * recursive_print
 * Walk through the tables, and print what you find.
 * Print the given table, and then print it's children.
 *
 * Inputs: depth = level (how many tabs to print)
 *         table = points to the table to print
 */
void recursive_print(int depth, CONTEXT *table )
{
    int i;
    char tabs[] = "\t\t\t\t\t";
    char str_time[8];

    /* print tabs to create nested table */
    tabs[ depth ] = '\0'; /* create tab string */

    if (table->max_index == -1)
        return;

    /* Print this table's information */
    for (i=0; i <= table->max_index; i++) {
        if (get_char_type(table->stats[i].symbol,0) == STRT){
            get_hhmm_from_code(table->stats[i].symbol, str_time ); //convert to time
string hhmm
            printf("%sSymbol: %s, counts: %d\n",
                    tabs,
                    str_time,
                    table->stats[i].counts);
            //table->stats[i].symbol,
        }
        else printf("%sSymbol: 0x%04x, counts: %d\n",
                    tabs,
                    table->stats[i].symbol,
                    table->stats[i].counts);

        /* If this table has links, print them */
        if ( table->links != NULL && depth < max_order )
            recursive_print( depth+1, table->links[i].next);
    }
}

/*****
** probability
**
** Given a context and a char, return the probability of that char.
**
**
** INPUTS:  string context (assumed to be shorter than max_order!)
**          character
**          pointer to symbol where results will be stored
**          verbose = true to print out results
** OUTPUTS: symbol is filled in with interval information
** RETURNS: probability (as a number between 0 and 1)
** Note that ESCAPE probability is NOT included.
*/
float probability( SYMBOL_TYPE c, STRING16 * context_string, char verbose)
{
    int i;
    CONTEXT *table;
    int prob_numerator = 1, prob_denominator = 1;
#ifdef TOOK_THIS_OUT_TO_BE_CONSISTENT
    int local_order=0;
#endif
    float fl_prob;
    int done = false;
    // final probability calculation

```

Source Code for the Markov Model

```

while (!done){
    // Traverse the tree, trying to find the context string.
    if (current_order >= 0)
        traverse_tree(context_string);
    table = contexts[current_order];    // point to best context

    // now find the character we want the probability of
    i = 0;
    while (i <= table->max_index &&
           table->stats[i].symbol != c)
        i++;

    if (i > table->max_index) {
        // If you got here, it means that you found the context string (or part of
it)
        // in the table, but can't find the test character c anywhere. You can
try a shorter
        // context or stop at level -1.
        if (current_order > 0)
            shorten_string16( context_string ); // remove the first character
from the context string and try again
        else {
            // This character wasn't found in the training data, so fall back
to level -1
            current_order = -1;
        }
    }
    else
        done = true;    // found the test character in the current context
table
    }
    // Brute force calculation of the probability

    // Numerator = counts for this character c
    prob_numerator = table->stats[i].counts;

#ifdef TOOK_THIS_OUT_TO_BE_CONSISTENT
    // Demoninator has two parts, it's the sum of all the counts + the number of elements in
the table
    // (His context[0] table has an extra entry in it, so don't add the extra '1'
    if (local_order == 0)
        prob_denominator = table->max_index;
    else
        prob_denominator = table->max_index + 1;    // this is the number of elements in
the table.
#else
    prob_denominator = 0;
#endif

    for (i=0; i <= table->max_index; i++)
        prob_denominator += table->stats[i].counts;

    fl_prob = (float) prob_numerator/(float) prob_denominator;
    printf("Pr( 0x%04x | %s) = %d/%d = %f\n", c, format_string16(context_string),
        prob_numerator,
        prob_denominator,
        fl_prob);
    return( fl_prob);
}

/*****
** predict_next
**
** Given a context, return:
**     the most likely next chars and their probabilities
**     the context level at which you found those chars (to indicate
**         (if you had to fall back to a lower level of context)
**
** INPUTS:  string context (only the last max_order bytes are used)

```

Source Code for the Markov Model

```

**                pointer to where results will be stored
** OUTPUTS: symbol is filled in with interval information
** RETURNS:
*****/
unsigned char predict_next( STRING16 * context_string, STRUCT_PREDICTION * results)
{
    int i;
    CONTEXT *table;
    int max_counts;                // maximum value for 'counts' found

    // Traverse the tree, trying to find the context string.
    // If the entire context string can't be found,
    // trim the first char off of the context string and try again.
    // At the end of this section,
    // the context[] array points to the current context
    // (example: context[1] points to the "a" context and
    // context[2] points to the "ab" context.
    // local_order is set to the depth where the context was found
    traverse_tree( context_string);
    if (current_order < 0)          // if the last char wasn't found at all, don't back down
all the way to -1
        current_order = 0;
    table = contexts[ current_order ];

    /* At this point, we have traversed the tree and we are
     * pointing to the best context that we can find.
     * table points to this context table
     * local_order is the depth (or order) of this model
     */
    // strcpy( results->context_string_used, &context_string[start_of_string]);
    results->depth = current_order;

    /* Find the symbols with the highest probability, given
     * this context.
     * Fortunately, this implementation of the model stores
     * the symbols in sorted order, so we can start at the
     * top and stop when we hit a max count or when the
     * counts (used to calc the probability) diminish.
     */

    // Denominator has two parts, it's the sum of all the counts + the number of elements in
the table
    // (His context[0] table has an extra entry in it, so don't add the extra '1'
#ifdef REMOVED_FOR_CONFIDENCE_LEVEL_TESTING
    if (current_order == 0)
        results->prob_denominator = table->max_index;
    else
        results->prob_denominator = table->max_index + 1;    // this is the number of
elements in the table.
#else
        results->prob_denominator = 0;                        // initialize
denominator
#endif
    max_counts = table->stats[0].counts; // max value for 'counts' is in first element
    // go through the table and store all symbols with the same value for 'counts' as the
    // first element.

    // the clause below [table->stats[i].counts== max_counts] is causing the model to only
    // return the most popular predictions, not all of the predictions.
    for (i=0;
        i <= table->max_index &&
        //table->stats[i].counts == max_counts &&                                // comment
this out to return ALL predictions instead of just the most popular.
        i < MAX_NUM_PREDICTIONS;
        i++)
    {
        // store information about this symbol
        results->sym[i].symbol = table->stats[i].symbol;
        results->sym[i].prob_numerator = table->stats[i].counts;
        results->prob_denominator += table->stats[i].counts;
    }
    results->num_predictions = i;

```

Source Code for the Markov Model

```

// Add up the rest of the counts in the table for the denominator
for ( ; i <= table->max_index; i++) {
    results->prob_denominator += table->stats[i].counts;
}
#endif
#ifdef DEBUG_MODEL
/* print results
*/
/** TEST TEST TEST TEST *****/
printf("predict_next: given context_string = \"%s\".\n",
format_string16(context_string));
printf("\tdepth = %d, denominator=%d, number of predictions = %d\n",
        results->depth,
        results->prob_denominator,
        results->num_predictions);
for (i=0; i < results->num_predictions; i++)
    printf("\tsymbol 0x%04x, numerator=%d\n", results->sym[i].symbol, results-
>sym[i].prob_numerator);
printf("\n");
#endif

return( results->sym[0].symbol);
}

/** print_model_allocation
 * print out the statistics on memory usage
 */
void print_model_allocation()
{
    printf("%d CONTEXT tables allocated.\n", alloc_count);
}

/*****
 * traverse_tree
 * Given a context string, traverse the tree.
 * Assumes context_string length < max_order
 * INPUTS: context_string (it may be empty, that's oK)
 * OUTPUTS: current_order is set to largest order where context
 *           was found
 *           contexts[] pointers set up (valid from 0 to current_order)
 * RETURNS: void
 *
 *****/
void traverse_tree( STRING16 * context_string) {
    int i;
    CONTEXT *table;
    SYMBOL_TYPE test_char; // char we are currently looking for in tree
    int local_order;
    int index_into_string;
    int done = false;

    // Traverse the tree, trying to find the context string.
    // If the entire context string can't be found,
    // trim the first char off of the context string and try again.
    // At the end of this section,
    // the context[] array points to the current context
    // (example: context[1] points to the "a" context and
    // context[2] points to the "ab" context.
    // local_order is set to the depth where the context was found
    local_order= 0; // Start with the order-0 model

    // initialize our moving index into the string. This
    // index is used to pull characters out of the string.
    index_into_string = 0;
    //printf("traverse_tree: context_string=\"%s\".\n", format_string16(context_string);

    // Slimy test for the blank string
    if (strlen16(context_string) == 0)
        done = true;

```

Source Code for the Markov Model

```

while (!done) {
    // traversing tree
    test_char = get_symbol(context_string, index_into_string );
    table = contexts[ local_order ];
    // Search this table for this character
    i = 0;
    while (i <= table->max_index &&
           table->stats[i].symbol != test_char &&
           table->max_index >= 0)
    {
        i++;
    }
    if ((i > table->max_index) ||
        ((table->links[i].next)->max_index == -1)) // didn't find this symbol in the
// there is no further
//
// (this second case only happens for
// the
// very end of the training
//
// string. You could have a branch
// of
// the tree that is only a few
//
// links long, so you can't assume you
// can
// always traverse down to
// a
// depth of max-order.
    {
        // didn't find it

        // if we are at the end of the string, we are out of luck
        if (strlen16( context_string) == 1) {
            printf("prob: couldn't find this char, going to level -1\n");
            local_order = -1;
            break; // stop traversing tree
        }
        else {
            shorten_string16( context_string);
            index_into_string = 0; // start over at beginning of shorter
//
// string
            local_order= 0;
        }
        continue;
    }

    // Found this character, go onto the next char
    if (++index_into_string == strlen16( context_string))
        done = true;
    local_order++;
    contexts[local_order] = table->links[i].next;
}
table = contexts[ local_order ];
current_order = local_order;

/* At this point, we have traversed the tree and we are
 * pointing to the best context that we can find.
 * table points to this context table
 * current_order is the depth (or order) of this model
 */
return;
}
/*****8
 * clear_scoreboard
 * Clear the scoreboard
 *****/
void clear_scoreboard() {
    int i;

    for ( i = 0 ; i < RANGE_OF_SYMBOLS ; i++ )

```

Source Code for the Markov Model

```

        scoreboard[ i ] = 0;
    }

/*****
 * compute_logloss
 *
 * Given a test string, calculate the average
 * log-loss for encoding the string.
 *
 * INPUTS:
 *     test_string = pointer to string to test.
 *
 * RETURNS: nothing
 * *****/
float compute_logloss( STRING16 * test_string, int verbose){
    int i;                // index into test string
    int length=0;         // string length
    SYMBOL s;             // interval information
    int escaped;           // true if we hit ESCAPE situation.
    double prob_numerator, prob_denominator; // for calculating probabilities for each char
    float fl_prob;        // probability as a float
    float summation = 0.0; // summation of the log-base-2(P())
    STRING16 * str_sub;

    if (verbose) {
        printf("compute_logloss: Testing on string \"%s\"\n", format_string_16(test_string));
    }

    // Create working substring
    str_sub = string16(max_order);

    // Calculate the probability of each character in the test string.
    // Since this calculation has to do with encoding, we need to include
    // the ESCAPE probabilities and the EXCLUSION mechanism, which
    // are handled by the convert_int_to_symbol routine.

    for (i=0; i < strlen16( test_string) ; i++) {

        // Create the context string, which is the max_order characters
        // before the character in question.
        // Ex. If the test_string is "abcdef" and i is 5, the
        // test character will be 'f' and the context string (for
        // a model order of 2) is the 2 characters before the 'f', which
        // are "de".

        if (i < max_order) {
            strncpy16( str_sub, test_string, 0, i);        // create test string
        }
        else {
            strncpy16( str_sub, test_string, i-max_order, max_order);
        }
        prob_numerator = 1;
        prob_denominator = 1;
        clear_scoreboard();
        if (verbose)
            printf("\t%d: log2(P(0x%04x|\"%s\"))",
                i, get_symbol(test_string, i), format_string16(str_sub));
        // print first part of line

        do {
            //printf("\t\ttraverse for \"%s\"\n", format_string16(str_sub));
            traverse_tree( str_sub);                // set pointers to best context
            escaped = convert_int_to_symbol( get_symbol(test_string,i), &s);
            //printf("\t\thigh=%d, low=%d, scale=%d\n", s.high_count, s.low_count,
s.scale);

            if (s.scale != 0) {
                prob_numerator *= (s.high_count - s.low_count);
                prob_denominator *= s.scale;
                //printf("\t\tnum=%f,denom=%f\n", prob_numerator, prob_denominator);
            }
        } while (escaped);
    }
}

```

Source Code for the Markov Model

```
        if (escaped){
            /* If the test char isn't found in this table, shorten the context
and try again. */
            //printf("escaped..");
            // was ==> if (strlen16(str_sub)== 0) {        // can't shorten
anymore
            if (strlen16(str_sub)<= 1) {                    // can't shorten anymore
                //printf(" abort\n");
                escaped=false;                                // abort if not found
            }
            else
                shorten_string16( str_sub);    // remove first char from
context
        }
    } while (escaped);

    fl_prob = (float) prob_numerator/(float) prob_denominator;
    //printf("fl_prob (%c)= %f\n", test_string[i], fl_prob);
    summation += log10(fl_prob);
    if (verbose)
        printf("= %f\n", log10(fl_prob)/log10(2.0));
}

// Convert logbase10 to log base 2 by diving by log-base-10(2)
summation /= log10(2.0);
// Take the average and change the sign
summation /= length;
summation *= -1.0;
if (verbose)
    printf("average log-loss is %f\n", summation);
return (summation);
} // end of compute_logloss
```

string16.c

```
/* *****
 * string16.c
 *
 * This module duplicates common string routines for arrays
 * of 16-bit characters.
 *
 * HISTORY:
 * 29Oct07 ink Created.
 *
 * *****/
#include <stdlib.h>           // for calloc(), free()
#include <assert.h>          // for assert()
#include <stdio.h>           // for sprintf()
#include "model.h"           // for MAX_STRING_LENGTH definition
#include "string16.h"

char printable_string16[5 * MAX_STRING_LENGTH+1];

/* Constructor string16
 * Create a string of 16-bit values; return a pointer to the string
 * NULL means error.
 * */
STRING16 * string16(int length){
    STRING16 * new_string16;
    SYMBOL_TYPE * ptr_array;
    ptr_array = (SYMBOL_TYPE *) calloc( sizeof( SYMBOL_TYPE ), length);
    if (ptr_array == NULL)
        return NULL;
    new_string16 = (STRING16 *) calloc(sizeof(STRING16), 1);
    new_string16->max_length = length;
    new_string16->s = ptr_array;
    return(new_string16);
}
```


Source Code for the Markov Model

```
/* Deconstructor - Delete string
*/
void delete_string16( STRING16 * str16_to_delete) {
    free( str16_to_delete->s);    // de-allocate the array
    free (str16_to_delete);        // de-allocate the structure
    str16_to_delete = NULL;
}

/* strlen16 - Return the length of the 16-bit 'string' */
int strlen16( STRING16 * s) {
    return s->length;
}

/* set_strlen16 - set the length of the 16-bit string */
void set_strlen16( STRING16 * s, int len) {
    s->length = len;
}

/* strncpy16 - Copy n elements
 * INPUT: dest = where to store the n elements (pointer to another STRING16)
 *        src = source
 *        offset = offset into src (where to start the copy)
 *        n = number of elements to copy
 */
STRING16 * strncpy16( STRING16 *dest, STRING16 *src, int offset, int n) {
    int i;

    assert(offset+n <= src->max_length);
    for (i=0; i < n; i++)
        dest->s[i] = src->s[offset+i];
    dest->length = n;
    return( dest);
}

/* format a STRING16 into a string of printable characters
 * FYI: Currently, this routine stores the result in global
 * memory to avoid memory leakage (if I allocated the string)
 */
char * format_string16( STRING16 *s16) {
    char * dest;
    SYMBOL_TYPE * src;
    int i, j;

    dest = printable_string16;
    src = s16->s;

    for (i = 0; i < s16->length && i < 5 * MAX_STRING_LENGTH; ) {
        for (j=0; j < 8 && (s16->length-i > 0); j++, i++) {
            sprintf( dest, "%04x ", *src);
            dest += 5;
            src++;
        }
        /*dest++ = '\n';
    }
    *dest = '\0'; // terminate string
    return (printable_string16);
}

/* Remove the first symbol from the string, shortening it by one symbol */
void shorten_string16( STRING16 *s16){
    int i;

    // (Use length+1 to copy the terminating null char)
    for (i=0; i < s16->length+1; i++)
        s16->s[i] = s16->s[i+1];
    s16->length--;
}

/* Return the symbol at the given offset of the given string. */
SYMBOL_TYPE get_symbol( STRING16 *s16, int offset){
    assert( offset <= s16->length);
```

Source Code for the Markov Model

```
        return(s16->s[offset]);
    }

/* Put the symbol at the given offset of the given string. */
void put_symbol( STRING16 *s16, int offset, SYMBOL_TYPE symbol){
    //assert( offset <= s16->length);
    s16->s[offset]=symbol;
}

/* read from a file into a STRING16 structure.
 * It's assumed that the file contains nothing but 16-bit signed
 * integers.
 */
int fread16( STRING16 *dest, int max_length, FILE *src_file){
    int i;

    i = fread( dest->s, sizeof( SYMBOL_TYPE), max_length, src_file);
    dest->length = i;
    dest->s[ i ] = 0x00;          // terminate 'string' with a null. (this should
                                // remove the final 0x0A linefeed
char.
    return(i);
}
```

coder.h

```
/*
 * Listing 1 -- coder.h
 *
 * This header file contains the constants, declarations, and
 * prototypes needed to use the arithmetic coding routines. These
 * declarations are for routines that need to interface with the
 * arithmetic coding stuff in coder.c
 */
#ifndef CODER_H_
#define CODER_H_

#define MAXIMUM_SCALE    16383 /* Maximum allowed frequency count */
#define ESCAPE           0xFFFF /* was 256 for char version */ /* The escape symbol */
#define DONE            -1 /* The output stream empty symbol */
#define FLUSH           -2 /* The symbol to flush the model */

/*
 * A symbol can either be represented as an int, or as a pair of
 * counts on a scale. This structure gives a standard way of
 * defining it as a pair of counts.
 */
typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

extern long underflow_bits; /* The present underflow count in */
                          /* the arithmetic coder. */

/*
 * Function prototypes.
 */
void initialize_arithmetic_decoder( FILE *stream );
void remove_symbol_from_stream( FILE *stream, SYMBOL *s );
```

Source Code for the Markov Model

```
void initialize_arithmetic_encoder( void );
void encode_symbol( FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( FILE *stream );
short int get_current_count( SYMBOL *s );

#endif
```

mapping.h (Example)

```
#ifndef MAPPING_H_
#define MAPPING_H_

unsigned int ap_map[525] = {0, 0x2321, 0x2322, 0x2323, 0x2324, 0x2325, 0x2326, 0x2327, 0x2328,
0x2329, 0x232a, 0x232b, 0x232d, 0x232e, 0x232f, 0x2330, 0x2331, 0x2332, 0x2333, 0x2334, 0x2335,
0x2336, 0x2337, 0x2338, 0x2339, 0x233a, 0x233c, 0x233d, 0x233e, 0x233f, 0x2340, 0x2341, 0x2342,
0x2343, 0x2344, 0x2345, 0x2346, 0x2347, 0x2348, 0x2349, 0x234a, 0x234b, 0x234c, 0x234d, 0x234e,
0x234f, 0x2350, 0x2351, 0x2352, 0x2353, 0x2354, 0x2355, 0x2356, 0x2357, 0x2358, 0x2359, 0x235a,
0x235b, 0x235c, 0x235d, 0x235e, 0x235f, 0x2360, 0x2361, 0x2362, 0x2363, 0x2364, 0x2365, 0x2366,
0x2367, 0x2368, 0x2369, 0x236a, 0x236b, 0x236c, 0x236d, 0x236e, 0x236f, 0x2370, 0x2371, 0x2372,
0x2373, 0x2374, 0x2375, 0x2376, 0x2377, 0x2378, 0x2379, 0x237a, 0x237b, 0x237c, 0x237d, 0x237e,
0x237f, 0x2380, 0x2381, 0x2382, 0x2383, 0x2384, 0x2385, 0x2386, 0x2387, 0x2388, 0x2389, 0x238a,
0x238b, 0x238c, 0x238d, 0x238e, 0x238f, 0x2390, 0x2391, 0x2392, 0x2393, 0x2394, 0x2395, 0x2396,
0x2397, 0x2398, 0x2399, 0x239a, 0x239b, 0x239c, 0x239d, 0x239e, 0x239f, 0x23a0, 0x23a1, 0x23a2,
0x23a3, 0x23a4, 0x23a5, 0x23a6, 0x23a7, 0x23a8, 0x23a9, 0x23aa, 0x23ab, 0x23ac, 0x23ad, 0x23ae,
0x23af, 0x23b0, 0x23b1, 0x23b2, 0x23b3, 0x23b4, 0x23b5, 0x23b6, 0x23b7, 0x23b8, 0x23b9, 0x23ba,
0x23bb, 0x23bc, 0x23bd, 0x23be, 0x23bf, 0x23c0, 0x23c1, 0x23c2, 0x23c3, 0x23c4, 0x23c5, 0x23c6,
0x23c7, 0x23c8, 0x23c9, 0x23ca, 0x23cb, 0x23cc, 0x23cd, 0x23ce, 0x23cf, 0x23d0, 0x23d1, 0x23d2,
0x23d3, 0x23d4, 0x23d5, 0x23d6, 0x23d7, 0x23d8, 0x23d9, 0x23da, 0x23db, 0x23dc, 0x23dd, 0x23de,
0x23df, 0x23e0, 0x23e1, 0x23e2, 0x23e3, 0x23e4, 0x23e5, 0x23e6, 0x23e7, 0x23e8, 0x23e9, 0x23ea,
0x23eb, 0x23ec, 0x23ed, 0x23ee, 0x23ef, 0x23f0, 0x23f1, 0x23f2, 0x23f3, 0x23f4, 0x23f5, 0x23f6,
0x23f7, 0x23f8, 0x23f9, 0x23fa, 0x23fb, 0x23fc, 0x23fd, 0x23fe, 0x23ff, 0x2421, 0x2422, 0x2423,
0x2424, 0x2425, 0x2426, 0x2427, 0x2428, 0x2429, 0x242a, 0x242b, 0x242c, 0x242d, 0x242e, 0x242f, 0x2430,
0x2431, 0x2432, 0x2433, 0x2434, 0x2435, 0x2436, 0x2437, 0x2438, 0x2439, 0x243a, 0x243c, 0x243d,
0x243e, 0x243f, 0x2440, 0x2441, 0x2442, 0x2443, 0x2444, 0x2445, 0x2446, 0x2447, 0x2448, 0x2449,
0x244a, 0x244b, 0x244c, 0x244d, 0x244e, 0x244f, 0x2450, 0x2451, 0x2452, 0x2453, 0x2454, 0x2455,
0x2456, 0x2457, 0x2458, 0x2459, 0x245a, 0x245b, 0x245c, 0x245d, 0x245e, 0x245f, 0x2460, 0x2461,
0x2462, 0x2463, 0x2464, 0x2465, 0x2466, 0x2467, 0x2468, 0x2469, 0x246a, 0x246b, 0x246c, 0x246d,
0x246e, 0x246f, 0x2470, 0x2471, 0x2472, 0x2473, 0x2474, 0x2475, 0x2476, 0x2477, 0x2478, 0x2479,
0x247a, 0x247b, 0x247c, 0x247d, 0x247e, 0x247f, 0x2480, 0x2481, 0x2482, 0x2483, 0x2484, 0x2485,
0x2486, 0x2487, 0x2488, 0x2489, 0x248a, 0x248b, 0x248c, 0x248d, 0x248e, 0x248f, 0x2490, 0x2491,
0x2492, 0x2493, 0x2494, 0x2495, 0x2496, 0x2497, 0x2498, 0x2499, 0x249a, 0x249b, 0x249c, 0x249d,
0x249e, 0x249f, 0x24a0, 0x24a1, 0x24a2, 0x24a3, 0x24a4, 0x24a5, 0x24a6, 0x24a7, 0x24a8, 0x24a9,
0x24aa, 0x24ab, 0x24ac, 0x24ad, 0x24ae, 0x24af, 0x24b0, 0x24b1, 0x24b2, 0x24b3, 0x24b4, 0x24b5,
0x24b6, 0x24b7, 0x24b8, 0x24b9, 0x24ba, 0x24bb, 0x24bc, 0x24bd, 0x24be, 0x24bf, 0x24c0, 0x24c1,
0x24c2, 0x24c3, 0x24c4, 0x24c5, 0x24c6, 0x24c7, 0x24c8, 0x24c9, 0x24ca, 0x24cb, 0x24cc, 0x24cd,
0x24ce, 0x24cf, 0x24d0, 0x24d1, 0x24d2, 0x24d3, 0x24d4, 0x24d5, 0x24d6, 0x24d7, 0x24d8, 0x24d9,
0x24da, 0x24db, 0x24dc, 0x24dd, 0x24de, 0x24df, 0x24e0, 0x24e1, 0x24e2, 0x24e3, 0x24e4, 0x24e5,
0x24e6, 0x24e7, 0x24e8, 0x24e9, 0x24ea, 0x24eb, 0x24ec, 0x24ed, 0x24ee, 0x24ef, 0x24f0, 0x24f1,
0x24f2, 0x24f3, 0x24f4, 0x24f5, 0x24f6, 0x24f7, 0x24f8, 0x24f9, 0x24fa, 0x24fb, 0x24fc, 0x24fd,
0x24fe, 0x24ff, 0x2521, 0x2522, 0x2523, 0x2524, 0x2525, 0x2526, 0x2527, 0x2528, 0x2529, 0x252a,
0x252b, 0x252d, 0x252e, 0x252f, 0x2530, 0x2531, 0x2532, 0x2533, 0x2534, 0x2535, 0x2536, 0x2537,
0x2538, 0x2539, 0x253a, 0x253b, 0x253c, 0x253d, 0x253e, 0x253f, 0x2540, 0x2541, 0x2542, 0x2543, 0x2544,
0x2545, 0x2546, 0x2547, 0x2548, 0x2549, 0x254a, 0x254b, 0x254c, 0x254d, 0x254e, 0x254f, 0x2550,
0x2551, 0x2552, 0x2553, 0x2554, 0x2555, 0x2556, 0x2557, 0x2558, 0x2559, 0x255a, 0x255b, 0x255c,
0x255d, 0x255e, 0x255f, 0x2560, 0x2561, 0x2562, 0x2563, 0x2564, 0x2565, 0x2566, 0x2567, 0x2568,
0x2569, 0x256a, 0x256b, 0x25
```

Source Code for the Markov Model

```

    {0x234c, 0x2340, 0x2346, 0x234f, 0x252b, 0x23b3, 0x237e, 0x2436, 0x2331, 0x2397, 0x24a5,
0x24a7, 0x24a9, 0x24b0, },
    {0},
    {0},
    {0x236f, 0x2361, 0x236d, 0x2372, 0x23b7, 0x23a4, 0x2339, 0x23b6, },
    {0},
    {0x23c4, 0x23f2, },
    {0x2332, },
    {0},
    {0x2364, 0x23bc, 0x2449, 0x23ba, 0x23bb, 0x2446, 0x244c, 0x23bd, 0x236a, },
    {0x2334, 0x2338, 0x2358, 0x237a, 0x235d, 0x239a, 0x2349, 0x23b3, 0x2357, 0x2377, },
    {0x252b, 0x24f5, 0x2529, 0x2534, 0x2353, 0x23e8, 0x24f6, 0x23df, 0x23e6, 0x23eb, 0x24f7,
0x2523, 0x252a, 0x2533, 0x23b1, 0x23e4, 0x23e7, 0x23ea, 0x24f4, 0x2333, 0x23d3, 0x252d, 0x23f1,
0x23af, 0x2336, 0x2454, },
    {0x2333, 0x254e, },
    {0x2332, },
    {0x2338, 0x2349, 0x2330, 0x2357, 0x236b, 0x2365, 0x237a, 0x24b0, 0x2550, 0x237c, 0x23d9,
0x2465, 0x23b3, 0x234a, 0x2535, 0x2342, 0x237e, 0x24aa, 0x24b4, 0x23de, 0x2358, 0x2345, 0x239a,
0x24a7, 0x2388, 0x2350, },
    {0x255a, 0x255c, 0x254b, 0x2556, 0x2369, 0x253e, 0x2399, 0x2559, 0x2367, 0x2557, 0x239b,
0x2565, 0x24a1, 0x2392, 0x238c, 0x2552, 0x2430, 0x2366, 0x24b0, 0x24aa, 0x24b4, 0x2394, 0x2378,
0x2383, 0x252e, 0x2364, 0x2553, 0x2551, 0x255b, 0x2389, },
    {0x2533, 0x24f5, 0x252b, 0x2331, 0x2353, 0x23df, 0x23e8, 0x252a, 0x23e6, 0x24f4, 0x23e4,
0x2529, 0x23d3, 0x23ea, 0x24f6, 0x24f7, 0x2534, 0x2428, 0x23e5, 0x23e7, 0x23eb, 0x2456, 0x2542,
0x2454, 0x2340, },
    {0},
    {0x2357, 0x237c, 0x24b0, 0x2334, 0x237a, 0x2349, 0x2330, 0x239a, 0x2358, 0x237e, 0x24aa,
0x24a7, 0x2350, },
    {0x236f, 0x2371, 0x2370, 0x255d, 0x2562, 0x2547, 0x2372, 0x2451, 0x2361, 0x24bb, 0x23e1,
0x23c8, 0x255f, 0x2424, 0x2329, 0x235a, 0x23ad, 0x2368, },
    {0},
    {0x255d, 0x2430, },
    {0},
    {0x2392, 0x2331, 0x2343, },
    {0x236a, 0x2344, 0x234f, 0x2362, 0x2346, 0x2364, 0x234d, },
    {0x234c, 0x234f, 0x24f5, 0x2346, 0x2363, 0x2331, 0x23b3, 0x2533, 0x23b1, 0x2326, 0x2546,
0x24f4, 0x252b, 0x2534, 0x2529, 0x2336, },
    {0x2379, 0x2347, 0x234b, },
    {0x2349, 0x234a, 0x2365, 0x236b, 0x2345, 0x2532, 0x2334, 0x23de, 0x2535, 0x2362, 0x236a,
0x234c, },
    {0x2540, 0x24f8, 0x2364, 0x233e, 0x254a, 0x2484, 0x2392, },
    {0x236a, 0x234c, 0x2364, 0x2362, 0x236b, 0x233f, 0x2365, 0x234d, 0x2346, 0x234f, },
    {0x236b, 0x2342, 0x2349, 0x2362, 0x2365, 0x236a, 0x2334, 0x234c, 0x235d, 0x234a, 0x23b3,
0x2535, 0x2348, },
    {0x2363, 0x233f, 0x236a, 0x234f, 0x2362, 0x2340, 0x234c, 0x2344, 0x236b, 0x24ca, 0x2364,
},
    {0},
    {0x236a, 0x2550, 0x23cf, 0x23d7, },
    {0x2334, 0x2338, 0x2365, 0x234a, 0x2550, 0x2342, 0x23cf, 0x236b, 0x237a, 0x2330, 0x2362,
0x2535, 0x2532, 0x2357, 0x237e, 0x237c, 0x23a6, 0x23d7, 0x23de, 0x23b3, 0x2345, 0x2333, 0x2348,
0x2388, },
    {0},
    {0},
    {0x2340, 0x2346, 0x2362, 0x2533, 0x2331, 0x23b3, 0x24f6, 0x2344, 0x234f, 0x24f5, 0x24f7,
0x2529, 0x252b, 0x2534, 0x2326, 0x235d, 0x2363, 0x2377, 0x2428, 0x236a, },
    {0x2344, 0x234f, 0x2364, 0x233f, 0x2326, },
    {0x2361, 0x236d, 0x2424, },
    {0x2362, 0x233f, 0x2346, 0x234d, 0x2340, 0x2344, 0x2363, 0x236a, 0x2445, 0x24f5, 0x2326,
0x252b, 0x234c, },
    {0x2383, 0x238c, 0x24be, 0x23cf, 0x2550, 0x2388, 0x2334, 0x2338, 0x2357, },
    {0},
    {0x23cd, 0x249b, 0x2444, },
    {0x2331, 0x23e8, 0x24f7, 0x23b1, 0x23e4, 0x23e7, 0x23df, 0x23ea, 0x23eb, 0x24f5, 0x24f6,
0x2523, 0x252a, 0x23e6, 0x2333, 0x2522, 0x23e5, 0x2534, 0x24f4, 0x2529, 0x2533, 0x252b, 0x2336,
},
    {0x2355, 0x2356, 0x23d0, 0x24a4, 0x23d6, 0x23da, 0x23db, 0x23d9, 0x2532, 0x2386, },
    {0x24c2, 0x2557, 0x2378, 0x2383, 0x238c, 0x255b, 0x2382, 0x2551, 0x2561, 0x2384, 0x253e,
0x23d2, 0x2553, 0x2558, 0x2399, 0x2430, 0x24c5, 0x254c, },
    {0x2335, 0x2399, 0x2355, 0x253e, 0x2557, },
    {0x2338, 0x237a, 0x239a, 0x24b0, 0x2349, 0x2334, 0x2330, 0x237e, 0x2358, 0x237c, 0x23de,
0x247b, 0x23b3, 0x24a7, 0x24aa, 0x24b6, 0x2388, },

```

Source Code for the Markov Model

```

{0x2330, 0x2334, 0x2338, 0x235d, 0x2357, 0x239a, 0x2349, 0x237a, },
{0},
{0},
{0x24f7, },
{0},
{0x2377, 0x2330, 0x2358, 0x2342, 0x234c, 0x2326, 0x2349, 0x2334, 0x23b4, 0x23b3, 0x23dd,
},
{0},
{0},
{0},
{0x236d, 0x23e1, 0x23b6, 0x2339, 0x2524, 0x236f, 0x234e, 0x2368, 0x2424, 0x2329, 0x2371,
0x23a4, },
{0x236b, 0x2344, 0x233f, 0x236a, 0x234c, 0x234f, 0x2345, 0x2349, 0x2346, 0x2365, 0x2363,
0x2364, 0x234d, },
{0x2346, 0x236b, 0x2362, 0x24ca, 0x234c, 0x2340, 0x234f, },
{0x2344, 0x23bc, 0x236a, 0x2362, 0x233f, 0x253f, 0x234d, 0x232f, 0x24f8, 0x254a, 0x2445,
0x2552, 0x2326, 0x2450, 0x234c, 0x244e, 0x23ba, 0x244d, 0x2343, 0x2392, 0x2346, 0x2484, 0x2388,
0x2387, },
{0x236a, 0x236b, 0x2345, 0x2349, 0x2334, 0x2346, 0x2362, 0x2342, 0x234c, 0x234a, 0x23b3,
0x2388, },
{0x2369, 0x2399, 0x2556, 0x255a, 0x255c, 0x2367, },
{0x2550, 0x236a, 0x255c, 0x255d, 0x2335, 0x2552, 0x255a, 0x2399, 0x253e, 0x2369, 0x2559,
},
{0x236f, 0x2361, 0x236d, 0x24d3, },
{0x2366, 0x255c, 0x2367, 0x2556, },
{0x2364, 0x2344, 0x2345, 0x2550, 0x2342, 0x2380, 0x2362, 0x2365, 0x23a6, 0x2349, 0x255d,
0x23d7, 0x234c, 0x2367, 0x253f, 0x23a5, 0x23ce, 0x23d4, 0x2445, 0x2532, 0x234a, 0x2346, 0x24f8,
0x255c, 0x2484, 0x23de, 0x244d, 0x2326, 0x2552, 0x2450, 0x23cb, 0x2425, 0x2399, 0x2348, 0x2546,
0x2381, 0x2388, 0x238a, 0x238b, 0x2387, },
{0x2363, 0x2342, 0x2345, 0x2349, 0x2362, 0x2365, 0x236a, 0x24ca, 0x2334, 0x234a, 0x2344,
0x23b3, 0x2535, 0x238a, 0x2388, },
{0x2544, 0x2322, },
{0x2361, 0x2368, 0x234e, 0x23b6, 0x2371, 0x23c7, 0x23c8, 0x2424, 0x2339, 0x236f, 0x23e1,
0x23a4, 0x23c6, },
{0x2427, },
{0x2339, 0x2370, 0x2371, 0x235a, 0x2372, 0x255f, 0x2562, 0x2451, 0x255d, 0x24bb, 0x24cb,
0x2361, 0x23e1, 0x2381, 0x23ad, },
{0x2339, 0x236f, 0x2371, 0x2372, 0x23e1, 0x235a, 0x2451, 0x255f, },
{0x2339, 0x236f, 0x2370, 0x235a, 0x2372, 0x2451, 0x23e1, 0x2368, 0x236d, 0x23e0, 0x255f,
0x23b6, 0x2562, 0x2329, 0x23c8, 0x23ad, 0x2361, },
{0x2370, },
{0x2481, },
{0},
{0x23b3, 0x23de, },
{0x2330, 0x239a, 0x23de, 0x247c, 0x237a, 0x2375, 0x237c, },
{0x235d, 0x23dd, },
{0x2383, 0x238c, 0x2557, 0x255b, 0x254b, 0x2552, 0x2335, 0x2399, 0x253e, 0x2394, 0x2430,
0x23d2, 0x24c2, 0x2551, 0x2556, 0x2355, 0x2559, },
{0x2341, },
{0x24b3, 0x24b5, 0x24b6, 0x2443, 0x247b, 0x24a7, 0x239a, 0x247c, 0x2330, 0x237e, 0x24b0,
0x24aa, 0x2338, 0x2357, 0x2358, 0x24b4, 0x24a8, 0x24ad, 0x237c, 0x245b, 0x24a6, 0x24b2, 0x24a5,
0x24ac, 0x24ab, 0x24a9, 0x24af, 0x237f, 0x24c0, 0x2334, 0x2349, 0x2376, 0x233d, 0x23de, 0x2535,
0x246b, 0x2388, },
{0},
{0x237f, 0x237e, 0x2357, 0x237a, 0x24b0, 0x24b1, 0x24a7, 0x24a9, 0x239a, 0x24ab, 0x247c,
0x2326, 0x24a8, 0x23cd, 0x2546, 0x2352, 0x247b, 0x2338, 0x233d, 0x24a5, 0x24ac, 0x2376, },
{0x2341, 0x234b, 0x2347, 0x23ef, 0x2379, },
{0x237f, 0x237c, 0x237a, 0x239a, 0x24b5, 0x2330, 0x2334, 0x2338, 0x2349, 0x2357, 0x23d5,
0x2546, 0x233d, 0x2535, 0x23de, 0x247c, },
{0x237c, 0x24b1, 0x237e, 0x233d, 0x2546, },
{0x236a, 0x23d4, 0x255d, 0x23ce, 0x23aa, 0x23ac, 0x23cb, 0x23a7, 0x23ab, 0x2381, },
{0x236a, 0x2380, 0x23ad, 0x245a, 0x2386, 0x24a4, 0x2339, },
{0x2384, 0x2561, 0x2355, },
{0x238c, 0x255b, 0x2350, 0x2551, 0x2557, 0x2558, 0x23d2, 0x24c2, 0x2378, 0x24d7, 0x2553,
0x246e, 0x2556, 0x2355, 0x254c, 0x253e, 0x2399, 0x242e, },
{0x2382, 0x2561, 0x2355, },
{0x2329, 0x2361, 0x236d, },
{0x2546, 0x236a, 0x23d7, 0x2550, 0x23d8, 0x23d1, 0x23d6, 0x23da, 0x24a4, 0x23cb, 0x2348,
0x23d4, 0x2388, 0x2532, 0x23a6, 0x23a5, 0x23cc, 0x23ce, 0x2425, 0x2380, 0x2381, 0x23d0, 0x23d9,
0x23db, 0x2349, },
{0},

```

Source Code for the Markov Model

```

    {0x2535, 0x23d7, 0x2532, 0x2550, 0x2386, 0x23a5, 0x236a, 0x23cf, 0x23a6, 0x2342, 0x2365,
0x238a, 0x2334, 0x2367, 0x2381, 0x2380, 0x2399, 0x2349, 0x236b, 0x2425, 0x2350, },
    {0x2335, 0x255c, 0x2556, },
    {0x2349, 0x2362, 0x2365, 0x236b, 0x2388, 0x2342, 0x234a, 0x236a, },
    {0},
    {0x2383, 0x2551, 0x2557, 0x255b, 0x2350, 0x2558, 0x2378, 0x23d2, 0x2553, 0x254c, 0x2394,
0x2399, 0x2355, 0x24c2, 0x2382, 0x2559, 0x2565, 0x242e, },
    {0x2391, 0x2394, 0x2399, },
    {0x238f, 0x2437, 0x2439, 0x243a, 0x243e, 0x24f1, 0x243c, 0x23ee, },
    {0x23ee, 0x2437, 0x243a, 0x238e, 0x243e, 0x243c, 0x24f1, },
    {0x23ee, 0x2437, 0x243c, },
    {0x2399, 0x2557, 0x2398, 0x2430, 0x2394, 0x238d, },
    {0x24a3, 0x2540, 0x2556, 0x242d, 0x2549, 0x2438, 0x24a1, 0x2343, 0x2432, 0x2335, 0x24f8,
},
    {0x23fa, 0x23fd, 0x237b, 0x2359, 0x23ff, },
    {0x238d, 0x2391, 0x2399, 0x2559, 0x2552, 0x2430, 0x2378, 0x254b, },
    {0x2426, },
    {0},
    {0x2426, 0x23b2, 0x2326, 0x2331, 0x23b0, 0x23f1, 0x2435, 0x2436, 0x2336, 0x2434, },
    {0x2430, 0x2399, 0x2391, 0x238d, },
    {0x2557, 0x2430, 0x2367, 0x2335, 0x2378, 0x2394, 0x253e, 0x254b, 0x255c, 0x23d2, 0x2556,
0x2552, 0x2559, 0x239b, 0x255a, 0x238c, 0x2383, 0x2398, 0x2565, 0x2391, 0x2369, 0x236a, 0x2355,
0x233c, 0x255d, 0x2380, 0x238d, 0x2356, },
    {0x24b6, 0x237a, 0x237e, 0x247c, 0x2357, 0x2338, 0x247b, 0x2443, 0x24b3, 0x24a7, 0x24b0,
0x237c, 0x2376, 0x23de, 0x24aa, 0x2330, 0x245b, 0x24c0, 0x24b5, 0x237f, 0x2358, },
    {0x2335, 0x255c, 0x253e, 0x2556, 0x2399, 0x2430, 0x254b, 0x2559, 0x2378, 0x255a, 0x236a,
0x2367, },
    {0x2556, 0x255a, 0x239b, 0x255c, },
    {0x254b, 0x2560, },
    {0},
    {0x23bb, 0x23bc, 0x23c1, 0x2448, 0x2449, 0x244e, 0x23ba, },
    {0x24fe, 0x24ff, 0x24b8, },
    {0},
    {0x24c7, 0x238e, 0x2437, },
    {0},
    {0},
    {0x236a, 0x23a6, 0x2550, 0x2349, 0x2388, 0x2386, },
    {0x2349, 0x236a, 0x2550, 0x2345, 0x23a5, 0x23d7, 0x2326, 0x2365, 0x2532, 0x234a, 0x2535,
0x2348, 0x2388, 0x2386, },
    {0x2380, 0x23ac, 0x255d, 0x23aa, 0x23ab, },
    {0},
    {0x2524, },
    {0x23a7, 0x23ac, 0x23ab, 0x2380, },
    {0x2380, 0x23a7, 0x23ac, 0x255d, 0x23aa, },
    {0x23a7, 0x23aa, 0x23ab, },
    {0x2339, 0x23ed, 0x2381, 0x236f, 0x2371, },
    {0x23d3, 0x23b1, 0x252b, 0x2340, 0x24f5, 0x2529, 0x24f6, 0x2533, 0x2534, 0x2333, 0x23df,
0x24f4, 0x24f7, 0x23af, 0x2331, },
    {0x2529, 0x252b, 0x2533, 0x2534, 0x2331, 0x24f5, 0x24f6, },
    {0x2397, 0x2436, },
    {0x23af, 0x2534, 0x2331, 0x23ae, 0x24f7, 0x252a, 0x23df, 0x252b, 0x24f4, 0x24f5, 0x23d3,
0x23ea, 0x24f6, 0x2529, 0x2533, 0x252d, 0x23e8, 0x23e6, 0x235b, 0x2523, 0x23cd, },
    {0x23b0, },
    {0x234c, 0x2331, 0x2377, 0x24f6, 0x252b, 0x23de, 0x235d, 0x2342, 0x255e, 0x2340, 0x23d3,
0x24f4, 0x24f5, 0x24f7, 0x2529, 0x2534, 0x2326, 0x2345, 0x23b4, 0x2545, 0x2533, 0x2388, 0x238a,
0x23dd, 0x2375, },
    {0x23b3, 0x235d, 0x2377, },
    {0},
    {0x236d, 0x2361, 0x23c8, 0x2329, 0x236f, 0x2424, 0x23e1, },
    {0x23a4, 0x2329, },
    {0},
    {0},
    {0x23bc, 0x2450, 0x23bb, 0x23c5, 0x232f, 0x23c1, 0x2364, 0x244c, 0x244e, },
    {0x23bc, 0x2446, 0x232f, 0x239f, 0x2449, 0x23ba, },
    {0x232f, 0x2364, 0x236a, 0x23ba, 0x23bb, 0x2449, 0x23c5, 0x2448, 0x239f, 0x244c, 0x23c2,
},
    {0x232f, 0x23bb, },
    {0x23c1, 0x244e, 0x23ba, },
    {0x23c0, },
    {0},
    {0x23ba, 0x239f, 0x23c5, 0x2448, 0x23be, 0x244e, },

```

Source Code for the Markov Model

```

    {0x23bb, 0x23bc, },
    {0},
    {0x232b, 0x23e3, },
    {0x244e, 0x23ba, 0x23c1, 0x2448, 0x239f, 0x23bc, 0x23be, },
    {0x2361, 0x236d, },
    {0},
    {0x2339, 0x23b6, 0x2361, 0x23e1, 0x236f, 0x2370, 0x2371, },
    {0x2451, 0x255f, 0x235f, 0x255d, 0x2339, 0x23ed, 0x23ad, },
    {0},
    {0x23d4, 0x23ce, 0x24a4, 0x23d0, 0x23db, 0x23da, 0x2326, 0x2425, 0x2386, },
    {0x23cb, 0x23d6, 0x23d8, 0x23da, 0x23d1, 0x23d9, 0x2550, 0x23db, 0x2349, 0x2425, 0x236a,
0x24a4, 0x2532, 0x23d7, 0x2535, 0x2386, },
    {0x2352, 0x23b1, },
    {0x23cb, 0x2380, 0x23da, 0x24a4, 0x23d4, 0x2550, },
    {0x2550, 0x2349, 0x23d7, 0x2334, 0x23a6, 0x2532, 0x234a, 0x2535, 0x2386, 0x2388, 0x23d8,
0x2350, },
    {0x23da, 0x23db, 0x23ce, 0x237e, 0x2354, 0x23d6, 0x24a4, 0x2532, 0x23d5, 0x23cb, 0x23d9,
0x23e3, 0x2386, },
    {0x23d8, 0x2550, 0x23d9, 0x2532, 0x23cc, 0x23d6, 0x23db, 0x23da, 0x24a4, 0x23d7, 0x236a,
},
    {0x2383, 0x238c, 0x2551, 0x255b, 0x2558, 0x246e, 0x24c2, 0x2557, 0x254c, 0x2378, 0x2553,
0x2355, 0x2382, 0x253e, },
    {0x2534, 0x2533, 0x24f6, 0x252b, 0x24f7, 0x24f5, 0x24f4, 0x2529, },
    {0x23cb, 0x255d, 0x23ce, 0x2380, 0x2381, 0x2386, },
    {0x24a4, 0x23db, 0x23d0, },
    {0x23da, 0x23db, 0x23cc, 0x23d8, 0x23d9, 0x23d0, 0x2354, 0x23d1, 0x2550, 0x2386, },
    {0x23cf, 0x2550, 0x2349, 0x2532, 0x236a, 0x23a6, 0x23d8, 0x23a5, 0x23d1, 0x2535, 0x2386,
0x2388, },
    {0x23cc, 0x23d1, 0x23d9, 0x2425, 0x2550, 0x236a, 0x23d7, 0x23db, 0x23d6, 0x23da, 0x2465,
0x2535, 0x2532, 0x23cf, 0x2386, 0x23a6, },
    {0x23d1, 0x23db, 0x23da, 0x23cc, 0x23d8, 0x2550, 0x23d6, 0x23cf, 0x23d0, 0x2532, 0x23cb,
0x24a4, 0x2386, 0x2535, 0x2388, },
    {0x23d0, 0x23d6, 0x23db, 0x23cc, 0x23d1, 0x2550, 0x23d9, 0x2354, 0x23cb, 0x23d8, 0x23d4,
0x24a4, 0x2425, 0x2386, },
    {0x23cc, 0x23d9, 0x23da, 0x23d1, 0x23d6, 0x23d0, 0x23cb, 0x23d8, 0x2550, 0x2425, 0x24a4,
0x23d4, 0x2354, 0x2535, 0x2386, 0x2546, },
    {0x2441, },
    {0x23b3, 0x23de, },
    {0x23b3, 0x2376, 0x237a, 0x239a, 0x255e, 0x235d, 0x2545, 0x23b4, 0x2342, 0x2357, 0x2349,
0x2330, 0x2334, 0x237c, 0x2345, 0x236b, 0x2428, 0x2463, 0x24b0, 0x24b6, 0x247c, 0x2377, 0x2375,
},
    {0x2331, 0x24f4, },
    {0},
    {0x2361, 0x23c8, 0x2339, 0x236f, 0x2370, 0x2371, 0x2368, 0x236d, 0x2451, 0x23b6, },
    {0},
    {0x23c4, 0x232b, 0x243f, },
    {0x2353, 0x23e8, 0x2331, 0x23e5, 0x23e7, 0x23eb, 0x235b, 0x2336, },
    {0x2353, 0x23e4, 0x23e8, 0x23e7, 0x23eb, 0x2331, 0x23e6, },
    {0x2331, 0x2353, 0x23e8, 0x23ea, 0x2533, 0x24f5, 0x24f6, 0x2534, 0x23df, 0x24f4, 0x24f7,
0x252b, 0x2529, 0x23e5, 0x23e7, 0x23eb, 0x23d3, 0x2523, 0x252d, 0x252a, 0x235b, 0x2336, },
    {0x23e5, 0x2353, 0x23e4, 0x23e8, 0x23eb, 0x2331, 0x23e6, },
    {0x2331, 0x2353, 0x24f4, 0x24f5, 0x24f7, 0x252b, 0x252a, 0x23e6, 0x23df, 0x23e4, 0x23e7,
0x23eb, 0x23e5, 0x23ea, 0x24f6, 0x2522, 0x2533, 0x23b1, 0x2529, 0x2333, 0x2336, 0x2453, 0x2456,
0x2454, },
    {0x2427, },
    {0x2331, 0x23e6, 0x252b, 0x24f5, 0x252d, 0x2533, 0x2534, 0x24f6, 0x24f7, 0x2529, 0x252a,
0x23b1, 0x23d3, 0x23e8, 0x2353, 0x24f4, 0x23df, 0x23ae, 0x235b, 0x2336, },
    {0x2353, 0x23e5, 0x23e7, 0x23e8, 0x23e4, 0x2331, 0x23e6, 0x23ea, },
    {0},
    {0x2339, 0x2371, 0x2372, },
    {0x2423, 0x243c, 0x2437, 0x238e, 0x238f, 0x243a, 0x2390, },
    {0},
    {0},
    {0},
    {0},
    {0},
    {0x2397, },
    {0},
    {0x243d, },
    {0x234b, },
    {0},

```

Source Code for the Markov Model

```

{0x23fb, 0x23f4, 0x24b3, 0x24b5, },
{0},
{0x23f4, 0x2422, 0x23f8, },
{0},
{0},
{0},
{0},
{0},
{0x2436, 0x2397, 0x23f8, 0x24d0, 0x242f, 0x23f9, },
{0x242a, 0x2390, },
{0x234e, 0x2361, 0x236d, 0x23b6, 0x2339, 0x236f, 0x24c6, 0x23a4, 0x2368, 0x23c8, 0x2329,
0x23e1, 0x2385, },
{0x23cc, 0x23ce, 0x23d8, 0x23db, 0x2550, 0x23cb, },
{0x2433, 0x2395, 0x2397, 0x2422, },
{0x23e9, 0x236e, },
{0x2340, 0x234c, 0x2336, },
{0},
{0},
{0},
{0},
{0x2383, 0x238c, },
{0},
{0x2399, 0x2557, 0x2552, 0x2559, 0x2398, 0x253e, 0x255c, 0x2391, 0x255d, 0x2335, 0x2355,
0x238d, 0x2394, },
{0x238a, },
{0x2392, 0x243e, },
{0x2426, 0x2422, 0x242f, },
{0x242f, },
{0x2397, 0x23b0, 0x2436, },
{0x2422, 0x2435, },
{0x243c, 0x238e, 0x238f, 0x2439, 0x243a, 0x243e, 0x23ee, 0x24f1, },
{0x2392, 0x24a3, 0x2343, },
{0x243a, 0x243e, 0x238e, 0x238f, },
{0x238e, 0x238f, 0x2437, 0x243c, 0x243e, 0x2439, 0x2492, 0x24f1, },
{0x238e, 0x23ee, 0x2437, 0x238f, 0x243a, 0x24f1, },
{0x23f7, },
{0x238e, 0x2492, 0x2437, 0x238f, 0x2439, 0x243a, 0x243c, 0x23a2, 0x24c7, 0x24f1, 0x2432,
},
{0x23e3, },
{0x2442, },
{0x256c, 0x23dc, 0x256b, 0x256d, },
{0x2440, },
{0x237a, 0x247b, 0x24b3, 0x24a7, 0x239a, 0x247c, 0x24b6, 0x24b5, 0x245b, 0x23de, },
{0},
{0x2447, 0x244d, 0x244f, },
{0x244c, 0x244b, 0x244f, 0x232f, 0x244a, },
{0x244d, 0x2445, 0x244f, 0x2448, },
{0x244d, 0x2449, 0x244e, },
{0x244a, 0x23ba, 0x23bc, 0x232f, 0x2450, 0x2448, 0x2446, 0x2445, 0x244d, },
{0x2449, 0x2446, 0x244b, 0x244c, 0x244e, },
{0x2446, 0x244a, 0x244f, 0x244e, 0x2447, },
{0x244a, 0x23bd, 0x2446, 0x2450, 0x244b, },
{0x2447, 0x2445, 0x2448, 0x24f8, 0x2449, },
{0x2450, 0x23c5, 0x244a, 0x244b, 0x2448, 0x2343, 0x23ba, },
{0x244b, 0x2447, 0x2446, 0x2445, },
{0x2448, 0x244e, 0x244c, },
{0x2339, 0x236f, 0x255f, 0x2370, 0x2371, 0x2562, 0x2372, 0x2547, 0x23b9, 0x23c9, 0x23e1,
0x2368, 0x23ed, },
{0},
{0},
{0},
{0x2522, },
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0x2461, },
{0x245a, 0x2459, },

```


Source Code for the Markov Model

```
{0x2487, 0x248f, 0x24d1, 0x2494, },  
{0x2496, },  
{0x23a3, 0x24d9, },  
{0},  
{0},  
{0},  
{0x237a, },  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0x237a, 0x247c, 0x2357, 0x237c, 0x239a, 0x24b3, 0x24b6, 0x2443, },  
{0x237a, 0x239a, 0x247b, 0x24b6, 0x24b0, 0x2443, 0x2357, 0x237c, 0x2330, 0x237e, 0x237f,  
0x2376, },  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0},  
{0x2364, 0x236a, },  
{0},  
{0},  
{0},  
{0},  
{0x23e2, 0x23a2, 0x252e, 0x2392, 0x2549, 0x2480, },  
{0},  
{0},  
{0},  
{0},  
{0x2563, },  
{0},  
{0},  
{0},  
{0},  
{0x2499, },  
{0x2487, 0x248f, 0x24d1, },  
{0},  
{0},  
{0},  
{0x2468, },  
{0},  
{0},  
{0x2352, 0x23cd, 0x2444, },  
{0},  
{0},  
{0},  
{0},  
{0},  
{0x2335, 0x2392, },  
{0},  
{0x2392, 0x2540, 0x2438, 0x2549, 0x2343, 0x2335, },
```

Source Code for the Markov Model

	{0x23d0,	0x2354,	0x237e,	0x23c4,	0x23db,	0x2546,	0x23d9,	0x23da,	0x23d5,	0x23d6,	0x23cb,	
0x23d1,	0x2550,	0x2535,	0x2465,	0x255d,	0x23ce,	0x23cc,	0x2386,	0x2532,	0x2388,	}	,	
	{0x24b1,	0x24a9,	0x24ac,	0x24a6,	0x24b0,	0x24a7,	0x24ae,	0x24a8,	0x24af,	0x24ab,	0x24b4,	
0x24ad,	0x237a,	0x2483,	0x24b3,	0x24e9,	0x24aa,	0x2546,	0x24b6,	0x2326,	}	,		
	{0x24ad,	0x24aa,	0x24b0,	0x24ac,	0x24af,	0x24a7,	0x24a5,	0x24b1,	0x24ae,	0x24ab,	0x24a9,	
0x24a8,	0x237a,	0x24b4,	0x24b2,	}	,							
	{0x24b0,	0x24b2,	0x24a9,	0x24ad,	0x24b1,	0x24b4,	0x24a8,	0x24a5,	0x24a6,	0x237a,	0x24aa,	
0x24ab,	0x237a,	0x2483,	0x24b3,	0x24b5,	0x2357,	0x24ae,	0x24af,	0x247c,	0x24c0,	0x2483,	0x237f,	
0x237e,	0x247b,	0x237c,	0x2326,	}	,							
	{0x24ad,	0x24a7,	0x24b0,	0x24b4,	0x24aa,	0x24ac,	0x24a5,	0x24b5,	0x24b3,	0x24a9,	0x24b1,	
0x24ab,	0x24b6,	0x24b2,	0x237a,	0x24a6,	}	,						
	{0x24b0,	0x24b1,	0x24a7,	0x24ad,	0x24aa,	0x24ab,	0x24a8,	0x24a5,	0x24ae,	0x24a6,	0x24ac,	
0x24b4,	0x24af,	0x24b6,	0x24b5,	0x2546,	}	,						
	{0x24a7,	0x24a8,	0x24b4,	0x24b0,	0x24af,	0x24a6,	0x24ad,	0x24b1,	0x24a5,	0x237a,	0x24b3,	
0x24a9,	0x2334,	0x2338,	0x2335,	0x255a,	0x255c,	0x24b2,	0x24ab,	0x24ac,	}	,		
	{0x24a5,	0x24ac,	0x24ae,	0x24af,	0x24b1,	0x24a6,	0x24b0,	0x24a7,	0x24ad,	0x24a9,	0x24b4,	
0x24aa,	0x24b5,	0x2483,	0x24d4,	0x24e7,	0x24e8,	0x24e9,	0x24b6,	0x2546,	0x24b3,	0x24b2,	}	,
	{0x24a5,	0x24ab,	0x24a7,	0x24ae,	0x24a6,	0x24a9,	0x24af,	0x237a,	0x24b1,	0x24b2,	0x24b0,	
0x24aa,	0x24b4,	0x24a8,	0x24ad,	0x24b6,	0x24b5,	}	,					
	{0x24a7,	0x24a8,	0x24b0,	0x24b5,	0x24b2,	0x24aa,	0x24b4,	0x24a5,	0x237a,	0x24b6,	0x24ac,	
0x24b1,	0x24a9,	0x24b3,	0x24af,	0x24a6,	0x24ab,	0x24ae,	}	,				
	{0x24a5,	0x24a6,	0x24af,	0x24b0,	0x24b1,	0x24ab,	0x24ad,	0x24ac,	0x24a7,	0x24a9,	0x24b2,	
0x2483,	0x237a,	}	,									
	{0x24a6,	0x24b0,	0x24aa,	0x24a7,	0x24ae,	0x24ab,	0x24ac,	0x24a5,	0x24b1,	0x24ad,	0x24a9,	
0x2483,	0x24d4,	0x24b6,	0x237a,	0x2546,	0x24e7,	0x24b2,	0x24b3,	}	,			
	{0x24b1,	0x24b4,	0x24a7,	0x24a8,	0x24a9,	0x24ab,	0x24aa,	0x24ad,	0x24a5,	0x24a6,	0x237a,	
0x24ac,	0x24b2,	0x24b6,	0x247b,	0x24b3,	0x24b5,	0x24ae,	0x24af,	0x2357,	0x239a,	0x237c,	0x237e,	
0x247c,	0x2483,	0x24c0,	}	,								
	{0x24a5,	0x24a9,	0x24a7,	0x24a8,	0x24ac,	0x24b0,	0x24ae,	0x24a6,	0x24ab,	0x24af,	0x24aa,	
0x24ad,	0x2483,	0x24b4,	0x2546,	0x237c,	0x2326,	0x2321,	}	,				
	{0x24ac,	0x24b0,	0x24a7,	0x24b5,	0x24ad,	0x237a,	0x24b6,	0x24aa,	0x24b3,	0x247b,	0x24a5,	
0x24b4,	0x24a8,	0x24c0,	}	,								
	{0x24b6,	0x24b5,	0x24b2,	0x24a7,	0x247b,	0x237a,	0x2443,	0x24b0,	0x24aa,	0x24b4,	0x24ad,	
0x24a6,	0x24a8,	0x247c,	0x239a,	0x24ac,	0x237e,	0x24c0,	0x2357,	0x24a5,	0x23de,	0x24ab,	0x2572,	
0x2571,	0x2574,	}	,									
	{0x24a7,	0x24a8,	0x24ad,	0x24b0,	0x24aa,	0x24a5,	0x24a6,	0x24af,	0x237a,	0x24b6,	0x24ab,	
0x24a9,	0x24b1,	}	,									
	{0x											

Source Code for the Markov Model

```
{0x24ac, },
{0},
{0},
{0x238c, },
{0},
{0x2461, },
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0x2472, 0x2493, 0x235c, },
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0x24f0, 0x24fd, },
{0},
{0x243a, 0x243c, 0x2437, 0x2360, },
{0},
{0},
{0x23ae, 0x23b1, 0x2533, 0x2331, 0x23df, 0x2529, 0x252b, 0x2534, },
{0x2331, 0x23af, 0x23d3, 0x23df, 0x24f4, 0x24f7, 0x2533, 0x2534, 0x24f6, 0x2529, 0x252b,
0x23e6, 0x23ae, 0x23ea, 0x23b1, 0x2523, 0x2340, 0x252d, },
{0x23ae, 0x23d3, 0x23df, 0x24f5, 0x23b1, 0x252b, 0x24f4, 0x2534, 0x2529, 0x2533, 0x234c,
},
{0x2353, 0x23e6, 0x2523, 0x23e8, 0x252b, 0x2533, 0x2331, 0x23d3, 0x24f5, 0x23b1, 0x2529,
0x2534, 0x24f4, 0x24f6, 0x252a, },
{0x2364, 0x236a, 0x2450, 0x2343, 0x253f, 0x254a, 0x2392, 0x2484, 0x244e, },
{0},
{0},
{0},
{0},
{0x24ef, 0x24f0, 0x235e, 0x24fc, 0x24ff, 0x23a0, 0x24b7, },
{0},
{0},
{0},
{0x2353, 0x2331, 0x23e8, 0x23f1, 0x252a, },
{0},
{0x23a9, 0x2361, 0x2329, },
{0},
{0},
{0},
{0},
{0x2331, 0x24f4, 0x24f5, 0x252b, 0x2533, 0x2534, 0x23ae, 0x23b1, 0x24f6, 0x2531, 0x2530,
0x23d3, 0x23df, 0x2340, 0x23ea, 0x23af, 0x252a, 0x235b, 0x24f7, 0x2336, 0x2337, },
{0x2331, 0x2353, 0x23df, 0x23e8, 0x2522, 0x23e6, 0x252b, 0x2533, },
{0x2529, 0x2534, 0x2331, 0x24f6, 0x2533, 0x23ae, 0x24f5, 0x23b1, 0x23ea, 0x23d3, 0x24f7,
0x2530, 0x23df, 0x2336, },
{0},
{0x252f, 0x2489, 0x24c9, 0x254d, 0x242d, 0x23f3, 0x243e, },
{0x252e, 0x243e, },
{0x24f6, 0x252b, 0x2531, },
{0x2530, 0x24f5, 0x24f6, 0x252b, 0x2533, 0x2534, },
{0x2550, 0x23d7, 0x23cf, 0x23d1, 0x23d8, 0x2535, 0x237a, 0x2349, 0x23a6, 0x236a, 0x2354,
0x2386, 0x2388, 0x23d5, 0x23db, },
{0x23d3, 0x252b, 0x2534, 0x2529, 0x23ae, 0x24f5, 0x24f6, 0x23ea, 0x2331, 0x24f4, 0x24f7,
0x23b1, 0x2523, 0x23e6, 0x23e8, 0x2533, 0x23af, },
{0x2331, 0x24f5, 0x2529, 0x252b, 0x23b1, 0x2531, 0x23ae, 0x24f6, 0x23af, 0x23d3, 0x2340,
},
```

Source Code for the Markov Model

```

    {0x2349, 0x23cf, 0x2550, 0x23a6, 0x23d7, 0x234a, 0x2334, 0x2532, 0x2465, 0x236a, 0x2388,
0x23d8, 0x23d9, },
    {0},
    {0x2478, },
    {0},
    {0},
    {0},
    {0},
    {0x2564, },
    {0x2556, 0x255a, 0x2561, 0x2335, 0x2399, 0x2430, 0x2551, 0x2378, 0x2383, 0x255b, 0x2382,
0x254b, 0x238c, 0x2559, 0x255c, 0x2565, 0x2355, 0x239b, 0x2394, 0x2389, 0x2553, 0x2356, },
    {0x236a, 0x24f8, 0x2364, 0x2484, },
    {0x2392, 0x24a3, 0x2343, 0x2438, 0x2549, },
    {0},
    {0x2336, },
    {0},
    {0x246c, 0x2322, 0x233a, },
    {0x23de, },
    {0x24a4, 0x23d0, 0x237e, 0x237c, 0x237f, 0x24a7, 0x24a5, 0x24b0, 0x24b1, 0x24a9, 0x255d,
0x236a, 0x2380, 0x2340, },
    {0},
    {0},
    {0x23e2, 0x2489, },
    {0x233e, 0x2343, 0x2392, },
    {0x2552, 0x2559, 0x239d, 0x2558, 0x255a, 0x253e, 0x2399, 0x2378, 0x2557, 0x2553, 0x255b,
},
    {0x2553, 0x246e, 0x2554, 0x2558, 0x2557, },
    {0},
    {0},
    {0},
    {0x236a, 0x23a6, 0x2349, 0x23d7, 0x23a5, 0x234a, 0x2380, 0x23cf, 0x2546, 0x23d1, 0x23d8,
0x2532, 0x23d9, 0x23cc, 0x23db, 0x23d6, 0x255d, 0x23da, 0x2535, 0x2425, 0x2345, 0x23de, 0x2326,
0x236b, 0x23d0, 0x2364, 0x2348, 0x2386, 0x2388, 0x2350, },
    {0x253e, 0x2561, 0x2559, 0x2557, 0x2553, 0x2558, 0x255c, 0x238c, 0x255b, 0x2383, 0x2384,
0x23f0, 0x2382, 0x24c2, 0x23d2, 0x2378, 0x2355, 0x2335, 0x2556, },
    {0x2367, 0x254b, 0x2559, 0x2557, 0x239d, 0x2560, 0x255a, 0x2335, 0x255c, 0x2565, 0x253e,
0x2556, 0x2553, },
    {0x254c, 0x2552, 0x2557, 0x2558, 0x254b, 0x239e, 0x246e, 0x2554, 0x255b, 0x246f, },
    {0},
    {0x23a9, 0x24bb, },
    {0x255a, 0x253e, 0x2335, 0x2369, 0x255c, 0x2399, 0x238c, 0x2559, 0x2430, 0x239b, 0x2551,
0x2565, 0x2552, 0x254b, 0x239c, 0x2561, 0x2378, 0x2389, 0x2356, },
    {0x2551, 0x255b, 0x254b, 0x2552, 0x2553, 0x2558, 0x2383, 0x238c, 0x2399, 0x2378, 0x253e,
0x2559, 0x2335, 0x2430, 0x23d2, 0x246e, 0x24c2, 0x254c, 0x2561, 0x2355, 0x2565, 0x2394, 0x255a,
0x23ec, 0x2356, },
    {0x2551, 0x255b, 0x255c, 0x2559, 0x2553, 0x2557, 0x2355, 0x254b, 0x2552, 0x246e, 0x246f,
0x254c, 0x2561, },
    {0x254b, 0x255b, 0x2399, 0x255c, 0x2557, 0x2551, 0x2335, 0x253e, 0x2565, 0x2556, 0x255a,
0x2367, 0x2552, 0x236a, },
    {0x2556, 0x254b, 0x2558, 0x2335, 0x239c, 0x255c, 0x2557, 0x2553, 0x2565, 0x2369, 0x253e,
0x2552, 0x2559, },
    {0x2551, 0x2558, 0x2557, 0x2559, 0x238c, 0x2383, 0x254b, 0x246e, 0x2553, 0x23d2, 0x24c2,
0x2355, 0x2561, 0x254c, },
    {0x2335, 0x2399, 0x255a, 0x2367, 0x2556, 0x2559, 0x253e, 0x236a, 0x239b, 0x2366, 0x2369,
0x2552, 0x2565, 0x254b, 0x2380, 0x2430, 0x2389, },
    {0x2380, 0x23d4, 0x255f, 0x233c, 0x236a, 0x2339, 0x23a7, 0x23cb, 0x24bc, 0x24d5, 0x236f,
0x2371, 0x23ce, 0x2425, 0x23d0, 0x2430, 0x2550, 0x24bb, 0x2546, 0x237e, 0x24a4, 0x2370, },
    {0x23b4, 0x23de, 0x2376, 0x235d, 0x2377, 0x23b3, 0x2342, 0x2349, },
    {0x2451, 0x2339, 0x255d, 0x24bb, 0x24bc, 0x2371, 0x236f, 0x23c9, 0x2370, 0x2540, },
    {0x2557, },
    {0x2551, 0x2382, 0x2384, 0x253e, 0x23fa, 0x23f0, 0x255b, 0x2399, 0x2355, },
    {0x2339, 0x2371, 0x235a, 0x236f, 0x2370, 0x23b9, 0x23ed, },
    {0},
    {0x2563, },
    {0x238c, 0x2399, 0x2335, 0x2378, 0x2383, 0x2430, 0x253e, 0x255c, },
    {0},
    {0},
    {0},
    {0},
    {0},
    {0},
    {0},

```

Source Code for the Markov Model

```
{0x256b, },
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
{0},
};

// ONE minute timeslots from 00:00 until 23:59
unsigned int timeslot_map[1441] = {0x2621, 0x2622, 0x2623, 0x2624, 0x2625, 0x2626, 0x2627,
0x2628, 0x2629, 0x262a, 0x262b, 0x262d, 0x262e, 0x262f, 0x2630, 0x2631, 0x2632, 0x2633, 0x2634,
0x2635, 0x2636, 0x2637, 0x2638, 0x2639, 0x263a, 0x263c, 0x263d, 0x263e, 0x263f, 0x2640, 0x2641,
0x2642, 0x2643, 0x2644, 0x2645, 0x2646, 0x2647, 0x2648, 0x2649, 0x264a, 0x264b, 0x264c, 0x264d,
0x264e, 0x264f, 0x2650, 0x2651, 0x2652, 0x2653, 0x2654, 0x2655, 0x2656, 0x2657, 0x2658, 0x2659,
0x265a, 0x265b, 0x265c, 0x265d, 0x265e, 0x265f, 0x2660, 0x2661, 0x2662, 0x2663, 0x2664, 0x2665,
0x2666, 0x2667, 0x2668, 0x2669, 0x266a, 0x266b, 0x266c, 0x266d, 0x266e, 0x266f, 0x2670, 0x2671,
0x2672, 0x2673, 0x2674, 0x2675, 0x2676, 0x2677, 0x2678, 0x2679, 0x267a, 0x267b, 0x267c, 0x267d,
0x267e, 0x267f, 0x2680, 0x2681, 0x2682, 0x2683, 0x2684, 0x2685, 0x2686, 0x2687, 0x2688, 0x2689,
0x268a, 0x268b, 0x268c, 0x268d, 0x268e, 0x268f, 0x2690, 0x2691, 0x2692, 0x2693, 0x2694, 0x2695,
0x2696, 0x2697, 0x2698, 0x2699, 0x269a, 0x269b, 0x269c, 0x269d, 0x269e, 0x269f, 0x26a0, 0x26a1,
0x26a2, 0x26a3, 0x26a4, 0x26a5, 0x26a6, 0x26a7, 0x26a8, 0x26a9, 0x26aa, 0x26ab, 0x26ac, 0x26ad,
0x26ae, 0x26af, 0x26b0, 0x26b1, 0x26b2, 0x26b3, 0x26b4, 0x26b5, 0x26b6, 0x26b7, 0x26b8, 0x26b9,
0x26ba, 0x26bb, 0x26bc, 0x26bd, 0x26be, 0x26bf, 0x26c0, 0x26c1, 0x26c2, 0x26c3, 0x26c4, 0x26c5,
0x26c6, 0x26c7, 0x26c8, 0x26c9, 0x26ca, 0x26cb, 0x26cc, 0x26cd, 0x26ce, 0x26cf, 0x26d0, 0x26d1,
0x26d2, 0x26d3, 0x26d4, 0x26d5, 0x26d6, 0x26d7, 0x26d8, 0x26d9, 0x26da, 0x26db, 0x26dc, 0x26dd,
0x26de, 0x26df, 0x26e0, 0x26e1, 0x26e2, 0x26e3, 0x26e4, 0x26e5, 0x26e6, 0x26e7, 0x26e8, 0x26e9,
0x26ea, 0x26eb, 0x26ec, 0x26ed, 0x26ee, 0x26ef, 0x26f0, 0x26f1, 0x26f2, 0x26f3, 0x26f4, 0x26f5,
0x26f6, 0x26f7, 0x26f8, 0x26f9, 0x26fa, 0x26fb, 0x26fc, 0x26fd, 0x26fe, 0x26ff, 0x2721, 0x2722,
0x2723, 0x2724, 0x2725, 0x2726, 0x2727, 0x2728, 0x2729, 0x272a, 0x272b, 0x272c, 0x272d, 0x272e, 0x272f,
0x2730, 0x2731, 0x2732, 0x2733, 0x2734, 0x2735, 0x2736, 0x2737, 0x2738, 0x2739, 0x273a, 0x273b,
0x273d, 0x273e, 0x273f, 0x2740, 0x2741, 0x2742, 0x2743, 0x2744, 0x2745, 0x2746, 0x2747, 0x2748,
0x2749, 0x274a, 0x274b, 0x274c, 0x274d, 0x274e, 0x274f, 0x2750, 0x2751, 0x2752, 0x2753, 0x2754,
0x2755, 0x2756, 0x2757, 0x2758, 0x2759, 0x275a, 0x275b, 0x275c, 0x275d, 0x275e, 0x275f, 0x2760,
0x2761, 0x2762, 0x2763, 0x2764, 0x2765, 0x2766, 0x2767, 0x2768, 0x2769, 0x276a, 0x276b, 0x276c,
0x276d, 0x276e, 0x276f, 0x2770, 0x2771, 0x2772, 0x2773, 0x2774, 0x2775, 0x2776, 0x2777, 0x2778,
0x2779, 0x277a, 0x277b, 0x277c, 0x277d, 0x277e, 0x277f, 0x2780, 0x2781, 0x2782, 0x2783, 0x2784,
0x2785, 0x2786, 0x2787, 0x2788, 0x2789, 0x278a, 0x278b, 0x278c, 0x278d, 0x278e, 0x278f, 0x2790,
0x2791, 0x2792, 0x2793, 0x2794, 0x2795, 0x2796, 0x2797, 0x2798, 0x2799, 0x279a, 0x279b, 0x279c,
0x279d, 0x279e, 0x279f, 0x27a0, 0x27a1, 0x27a2, 0x27a3, 0x27a4, 0x27a5, 0x27a6, 0x27a7, 0x27a8,
0x27a9, 0x27aa, 0x27ab, 0x27ac, 0x27ad, 0x27ae, 0x27af, 0x27b0, 0x27b1, 0x27b2, 0x27b3, 0x27b4,
0x27b5, 0x27b6, 0x27b7, 0x27b8, 0x27b9, 0x27ba, 0x27bb, 0x27bc, 0x27bd, 0x27be, 0x27bf, 0x27c0,
0x27c1, 0x27c2, 0x27c3, 0x27c4, 0x27c5, 0x27c6, 0x27c7, 0x27c8, 0x27c9, 0x27ca, 0x27cb, 0x27cc,
0x27cd, 0x27ce, 0x27cf, 0x27d0, 0x27d1, 0x27d2, 0x27d3, 0x27d4, 0x27d5, 0x27d6, 0x27d7, 0x27d8,
0x27d9, 0x27da, 0x27db, 0x27dc, 0x27dd, 0x27de, 0x27df, 0x27e0, 0x27e1, 0x27e2, 0x27e3, 0x27e4,
0x27e5, 0x27e6, 0x27e7, 0x27e8, 0x27e9, 0x27ea, 0x27eb, 0x27ec, 0x27ed, 0x27ee, 0x27ef, 0x27f0,
0x27f1, 0x27f2, 0x27f3, 0x27f4, 0x27f5, 0x27f6, 0x27f7, 0x27f8, 0x27f9, 0x27fa, 0x27fb, 0x27fc,
0x27fd, 0x27fe, 0x27ff, 0x2821, 0x2822, 0x2823, 0x2824, 0x2825, 0x2826, 0x2827, 0x2828, 0x2829,
0x282a, 0x282b, 0x282d, 0x282e, 0x282f, 0x2830, 0x2831, 0x2832, 0x2833, 0x2834, 0x2835, 0x2836,
0x2837, 0x2838, 0x2839, 0x283a, 0x283c, 0x283d, 0x283e, 0x283f, 0x2840, 0x2841, 0x2842, 0x2843,
0x2844, 0x2845, 0x2846, 0x2847, 0x2848, 0x2849, 0x284a, 0x284b, 0x284c, 0x284d, 0x284e, 0x284f,
0x2850, 0x2851, 0x2852, 0x2853, 0x2854, 0x2855, 0x2856, 0x2857, 0x2858, 0x2859, 0x285a, 0x285b,
0x285c, 0x285d, 0x285e, 0x285f, 0x2860, 0x2861, 0x2862, 0x2863, 0x2864, 0x2865, 0x2866, 0x2867,
0x2868, 0x2869, 0x286a, 0x286b, 0x286c, 0x286d, 0x286e, 0x286f, 0x2870, 0x2871, 0x2872, 0x2873,
0x2874, 0x2875, 0x2876, 0x2877, 0x2878, 0x2879, 0x287a, 0x287b, 0x287c, 0x287d, 0x287e, 0x287f,
0x2880, 0x2881, 0x2882, 0x2883, 0x2884, 0x2885, 0x2886, 0x2887, 0x2888, 0x2889, 0x288a, 0x288b,
0x288c, 0x288d, 0x288e, 0x288f, 0x2890, 0x2891, 0x2892, 0x2893, 0x2894, 0x2895, 0x2896, 0x2897,
0x2898, 0x2899, 0x289a, 0x289b, 0x289c, 0x289d, 0x289e, 0x289f, 0x28a0, 0x28a1, 0x28a2, 0x28a3,
0x28a4, 0x28a5, 0x28a6, 0x28a7, 0x28a8, 0x28a9, 0x28aa, 0x28ab, 0x28ac, 0x28ad, 0x28ae, 0x28af,
0x28b0, 0x28b1, 0x28b2, 0x28b3, 0x28b4, 0x28b5, 0x28b6, 0x28b7, 0x28b8, 0x28b9, 0x28ba, 0x28bb,
0x28bc, 0x28bd, 0x28be, 0x28bf, 0x28c0, 0x28c1, 0x28c2, 0x28c3, 0x28c4, 0x28c5, 0x28c6, 0x28c7,
0x28c8, 0x28c9, 0x28ca, 0x28cb, 0x28cc, 0x28cd, 0x28ce, 0x28cf, 0x28d0, 0x28d1, 0x28d2, 0x28d3,
0x28d4, 0x28d5, 0x28d6, 0x28d7, 0x28d8, 0x28d9, 0x28da, 0x28db, 0x28dc, 0x28dd, 0x28de, 0x28df,
0x28e0, 0x28e1, 0x28e2, 0x28e3, 0x28e4, 0x28e5, 0x28e6, 0x28e7, 0x28e8, 0x28e9, 0x28ea, 0x28eb,
0x28ec, 0x28ed, 0x28ee, 0x28ef, 0x28f0, 0x28f1, 0x28f2, 0x28f3, 0x28f4, 0x28f5, 0x28f6, 0x28f7,
0x28f8, 0x28f9, 0x28fa, 0x28fb, 0x28fc, 0x28fd, 0x28fe, 0x28ff, 0x2921, 0x2922, 0x2923, 0x2924,
0x2925, 0x2926, 0x2927, 0x2928, 0x2929, 0x292a, 0x292b, 0x292c, 0x292d, 0x292e, 0x292f, 0x2930, 0x2931,
0x2932, 0x2933, 0x2934, 0x2935, 0x2936, 0x2937, 0x2938, 0x2939, 0x293a, 0x293b, 0x293c, 0x293d, 0x293e,
0x293f, 0x2940, 0x2941, 0x2942, 0x2943, 0x2944, 0x2945, 0x2946, 0x2947, 0x2948, 0x2949, 0x294a,
```

Source Code for the Markov Model

[illegible]**model.h**

/*

Source Code for the Markov Model

```
* Listing 8 -- model.h
*
* This file contains all of the function prototypes and
* external variable declarations needed to interface with
* the modeling code found in model-1.c or model-2.c.
*/
#ifndef MODEL_H_
#define MODEL_H_

/*
 * External variable declarations.
 */
extern int max_order;
extern int flushing_enabled;

#include "string16.h"
#include "coder.h"

/*
 * Definitions
 */
#define MAX_NUM_PREDICTIONS 1500 // Maximum number of predictions
#define MAX_DEPTH 3 // Maximum depth of model (which is also the
longest context)
#define MAX_MODEL_COUNT 500 // used in count_model
#define true 1
#define false 0
#define MAX_STRING_LENGTH 30000

// The symbols ranges for the binary box strings
// (from build_16bit_boxstrings.py)
// 27Apr2009 Changed for 1 minute thresholds!!!
#define INITIAL_START_TIME 0x2620 // was 0x2120
#define FINAL_START_TIME 0x2DFF // was 0x21FF
#define INITIAL_DURATION 0x2220 // not used for MELT version
#define FINAL_DURATION 0x22FF
#define INITIAL_LOCATION 0x2320
#define FINAL_LOCATION 0x25FF
#define LOWEST_SYMBOL INITIAL_LOCATION // was INITIAL_START_TIME
#define RANGE_OF_SYMBOLS FINAL_START_TIME-LOWEST_SYMBOL // was FINAL_LOCATION-
LOWEST_SYMBOL

/*
 * This program consumes massive amounts of memory. One way to
 * handle large amounts of memory is to use Zortech's __handle
 * pointer type. So that my code will run with other compilers
 * as well, the handle stuff gets redefined when using other
 * compilers.
 */
#ifdef __ZTC__
#include <handle.h>
#else
#define __handle
#define handle_calloc( a ) calloc( (a), 1 )
#define handle_realloc( a, b ) realloc( (a), (b) )
#define handle_free( a ) free( (a) )
#endif

/*
 * A context table contains a list of the counts for all symbols
 * that have been seen in the defined context. For example, a
 * context of "Zor" might have only had 2 different characters
 * appear. 't' might have appeared 10 times, and 'l' might have
 * appeared once. These two counts are stored in the context
 * table. The counts are stored in the STATS structure. All of
 * the counts for a given context are stored in an array of STATS.
 * As new characters are added to a particular context, the STATS
 * array will grow. Sometimes, the STATS array will shrink
 * after flushing the model.
 */
typedef struct {
    SYMBOL_TYPE symbol; // was unsigned char
```

Source Code for the Markov Model

```

        int counts;                                // was 'unsigned char', but rescaling set some level
0 counts to 0
    } STATS;

/*
 * Each context has to have links to higher order contexts. These
 * links are used to navigate through the context tables. For example,
 * to find the context table for "ABC", I start at the order 0 table,
 * then find the pointer to the "A" context table by looking through
 * then LINKS array. At that table, we find the "B" link and go to
 * that table. The process continues until the destination table is
 * found. The table pointed to by the LINKS array corresponds to the
 * symbol found at the same offset in the STATS table. The reason that
 * LINKS is in a separate structure instead of being combined with
 * STATS is to save space. All of the leaf context nodes don't need
 * next pointers, since they are in the highest order context. In the
 * leaf nodes, the LINKS array is a NULL pointers.
 */
typedef struct {
    struct context *next;
} LINKS;

/*
 * The CONTEXT structure holds all of the known information about
 * a particular context. The links and stats pointers are discussed
 * immediately above here. The max_index element gives the maximum
 * index that can be applied to the stats or link array. When the
 * table is first created, and stats is set to NULL, max_index is set
 * to -1. As soon as single element is added to stats, max_index is
 * incremented to 0.
 *
 * The lesser context pointer is a navigational aid. It points to
 * the context that is one less than the current order. For example,
 * if the current context is "ABC", the lesser_context pointer will
 * point to "BC". The reason for maintaining this pointer is that
 * this particular bit of table searching is done frequently, but
 * the pointer only needs to be built once, when the context is
 * created.
 */
typedef struct context {
    int max_index;
    LINKS __handle *links;
    STATS __handle *stats;
    struct context *lesser_context;
} CONTEXT;

/*
 * This structure holds the results of a prediction, including the predicted
 * next symbol and it's probability (prob_numerator/prob_denominator).
 */
typedef struct {
    unsigned short int symbol;                // The symbol
    int prob_numerator; // numerator of the probability
                                           // the denominator is in the containing structure
} STRUCT_PREDICTED_SYMBOL;

/*
 * This structure holds a prediction, including a list of predicted symbols, with
 * their associated probabilities, the context string used to find the context,
 * and the context level at which the context was found. (context_level = length
 * of the context string) We are returning the context string in case (in the future)
 * we manipulate
 * the given context string to find a similar string with better probabilities.
 */
typedef struct {
    struct {
        SYMBOL_TYPE symbol;                // The symbol
        int prob_numerator; // numerator of the probability
                                           // the denominator is in the containing structure
    } sym[MAX_NUM_PREDICTIONS];
    //unsigned char context_string_used[MAX_DEPTH];

```


Source Code for the Markov Model

```
    int depth;                                // context level at which this prediction was made
    int num_predictions; // number of elements in sym[] array.
    int prob_denominator; // denominator of the probability
} STRUCT_PREDICTION;

/*
 * Prototypes for routines that can be called from MODEL-X.C
 */
void initialize_model( void );
void update_model( SYMBOL_TYPE symbol );
void clear_current_order(void);
int convert_int_to_symbol( SYMBOL_TYPE c, SYMBOL *s );
//void get_symbol_scale( SYMBOL *s );
//int convert_symbol_to_int( int count, SYMBOL *s );
void add_character_to_model( SYMBOL_TYPE c );
void flush_model( void );
void recursive_count(int depth, CONTEXT *table, int research_question, int verbose );
void count_model(int research_question, int verbose);
void print_model(void);
void recursive_print( int depth, CONTEXT * table);
float probability( SYMBOL_TYPE c, STRING16 * context_string, char verbose);
unsigned char predict_next(STRING16 * context_string, STRUCT_PREDICTION * results);
void print_model_allocation();
void traverse_tree( STRING16 * context_string);
void clear_scoreboard(void);
float compute_logloss( STRING16 * test_string, int verbose);
#endif
```

predict.h

```
#ifndef PREDICT_H_
#define PREDICT_H_

#define FALSE 0
#define TRUE ~FALSE

/*
 * Declarations for local procedures.
 */
int initialize_options( int argc, char **argv );
int check_compression( void );
//void print_compression( void );
void predict_test( STRING16 * test_string);
#ifdef NOTUSEDIN16BITVERSION
void remove_delimiters( char * str_input, char * str_purge);
void strpurge( char * str_in, char ch_purge);
#endif
int get_char_type( SYMBOL_TYPE symbol, int index_into_input_string);
int get_locstring_type( SYMBOL_TYPE symbol);
int get_boxstring_type( int index_into_input_string);
int get_loctimestring_type( SYMBOL_TYPE symbol);
int get_binboxstring_type( SYMBOL_TYPE symbol);
int get_bindowts_type( SYMBOL_TYPE symbol);
unsigned char neighboring_ap( SYMBOL_TYPE predicted_ap, SYMBOL_TYPE actual_ap);
int get_hhmm_from_code( SYMBOL_TYPE code, char * dest);
void test_timecode();
char * get_str_mappings(int mapping);
void build_test_string(STRING16 * test_string);
void analyze_pred_results( SYMBOL_TYPE correct_answer, SYMBOL_TYPE context, SYMBOL_TYPE
fallback_answer);
void output_pred_results(void);
unsigned char within_time_window( SYMBOL_TYPE time1, SYMBOL_TYPE time2, int range);

/* Function Types */
#define NO_FUNCTION 0
#define PREDICT_TEST 1
#define LOGLOSS_EVAL 2
```

Source Code for the Markov Model

```
/* String Types (types of input strings) */
#define NONE 0
#define LOCSTRINGS 1
#define LOCTIMESTRINGS 2
#define BOXSTRINGS 3
#define BINBOXSTRINGS 4
#define BINDOWTS 5

/* Character (Symbol) Types */
#define LOC 0 // location
#define STRT 1 // starting time
#define DUR 2 // duration
#define DELIM 3 // delimiter

/* Research Question Types */
/* These values are used in 'research_question' variable. */
#define WHERE 1 // "Where will Bob be at 10:00?"
#define WHEN 2 // "When will Bob be at location x?"
#endif /*PREDICT_H_*/
```

string16.h

```
/* *****
 * string16.h
 *
 * Prototypes for routines that are like string routines
 * but operate on strings (arrays) made up of 16-bit elements.
 * *****/

#ifndef STRING16_H_
#define STRING16_H_

#include <stdio.h> // for file I/O

typedef signed short int SYMBOL_TYPE;

typedef struct {
    int max_length; // allocated length of array
    SYMBOL_TYPE *s; // pointer to allocated memory
    int length; // length of string
} STRING16;

/* Function Prototypes */
STRING16 * string16(int length);
void delete_string16( STRING16 * str16_to_delete);
int strlen16( STRING16 * s);
void set_strlen16( STRING16 * s, int len);
STRING16 * strncpy16( STRING16 *dest, STRING16 *src, int offset, int n);
char * format_string16( STRING16 *s16);
void shorten_string16( STRING16 *s16);
SYMBOL_TYPE get_symbol( STRING16 *s16, int offset);
int fread16( STRING16 *dest, int max_length, FILE *src_file);
void put_symbol( STRING16 *s16, int offset, SYMBOL_TYPE symbol);
#endif /*STRING16_H_*/
```

Appendix E Utilities

The following routines were used to create the input files. The first section shows snippets of the data files.

001Pbox.dat (used with 16-bit Markov model)

```

Date, binary string with timeslot, location pairs
2002-09-22,e*$E+$ + $
2002-09-
23,]('$q('$($...('$□('$™('$£('$-$('$·('$Á('$Ë('$Ö('$B('$é('$ó('$ý('$()('$3)('$>)('$H)('$R)('$A)('$f)('$p)
('$z)('$„)('$ž)('$-)('$¢)('$¬)('$¶)('$Ä)('$f)('$p)('$z)('$„)('$ž)('$-)('$¢)('$¬)('$¶)('$Ä)('$Ö)('$P)('$
2002-09-24,Ä('$ø+$ú+$:-$
2002-09-25,™('$£('$Ÿ+$-
2002-09-26,...('$□('$Ö('$
2002-09-27,-(x#
2002-10-02,p)İ#ú+$
2002-10-03,Q*$e*$o*$y*$f*$□*$-$*$
2002-10-04,o*$y*$f*$-
*-$i*$μ*$ç*$Ë*$ó*$Ÿ*$ç*$ñ*$ü*$&+$1+$<+$F+$P+$Z+$d+$n+$x+$,$+$E+$x+$,$+$E+$-
+$+$a+$+$+$34+$E+$-+$+$a+$+$34+$E+$Ö+$

```

The following files for user 1 are used with the SEQ model.

user1.csv

```
1,09-22,13:07:09,354,31,1,1
1,09-22,13:07:09,363,13,1,0
1,09-22,13:07:30,354,20,1,1
1,09-22,13:07:30,355,12,1,0
1,09-22,13:07:30,363,14,1,0
1,09-22,13:07:58,354,24,1,1
1,09-22,13:07:58,363,14,1,0
1,09-22,13:08:18,354,20,1,1
1,09-22,13:08:18,363,16,1,0
1,09-22,13:08:38,354,22,1,1
1,09-22,13:08:38,363,10,1,0
1,09-22,13:08:58,354,22,1,1
1,09-22,13:08:58,363,10,1,0 ...
```

userb1.csv

1,	354,	2002-09-22	13:07:09.00,	00:04:50.00,	2002-09-22	13:11:59.00
1,	354,	2002-09-22	13:12:30.00,	00:02:22.00,	2002-09-22	13:14:52.00
1,	363,	2002-09-22	13:15:13.00,	00:03:23.00,	2002-09-22	13:18:36.00
1,	367,	2002-09-22	13:31:47.00,	00:09:12.00,	2002-09-22	13:40:59.00
1,	367,	2002-09-22	15:51:44.00,	00:11:39.00,	2002-09-22	16:03:23.00
1,	367,	2002-09-22	19:19:48.00,	00:03:18.00,	2002-09-22	19:23:06.00
1,	367,	2002-09-22	19:25:35.00,	00:04:34.00,	2002-09-22	19:30:09.00
1,	367,	2002-09-22	20:13:09.00,	00:15:29.00,	2002-09-22	20:28:38.00
1,	355,	2002-09-22	20:28:59.00,	00:01:01.00,	2002-09-22	20:30:00.00
1,	367,	2002-09-22	20:40:20.00,	00:11:55.00,	2002-09-22	20:42:15.00
1,	355,	2002-09-22	20:42:35.00,	00:03:24.00,	2002-09-22	20:45:59.00

Utilities

dur1.csv

```
1, 2002-09-22 13:07:09.00, 354, 00:03:50.00
1, 2002-09-22 13:11:28.00, 363, 00:00:20.00
1, 2002-09-22 13:12:09.00, 354, 00:00:00.00
1, 2002-09-22 13:12:30.00, 363, 00:00:00.00
1, 2002-09-22 13:12:50.00, 354, 00:02:02.00
1, 2002-09-22 13:15:13.00, 363, 00:02:22.00
1, 2002-09-22 13:17:55.00, 367, 00:00:00.00
1, 2002-09-22 13:18:15.00, 355, 00:00:21.00
1, 2002-09-22 13:31:47.00, 367, 00:09:12.00
1, 2002-09-22 15:51:44.00, 367, 00:11:39.00
1, 2002-09-22 19:19:48.00, 367, 00:03:18.00
1, 2002-09-22 19:25:35.00, 367, 00:03:54.00
1, 2002-09-22 19:29:50.00, 360, 00:00:20.00
1, 2002-09-22 20:13:09.00, 367, 00:15:29.00
...
```

userc1.csv

```
1, 367, 2002-09-22 15:51:44.00, 00:11:39.00, 2002-09-22 16:03:23.00
1, 367, 2002-09-22 20:13:09.00, 00:15:29.00, 2002-09-22 20:28:38.00
1, 367, 2002-09-22 20:30:20.00, 00:11:55.00, 2002-09-22 20:42:15.00
1, 355, 2002-09-23 08:27:21.00, 00:11:02.00, 2002-09-23 08:38:23.00
1, 355, 2002-09-23 08:44:44.00, 00:10:51.00, 2002-09-23 08:55:35.00
1, 367, 2002-09-23 08:55:55.00, 02:14:44.00, 2002-09-23 11:10:39.00
1, 355, 2002-09-23 11:00:45.00, 00:10:08.00, 2002-09-23 11:10:53.00
1, 367, 2002-09-23 11:10:35.00, 02:41:33.00, 2002-09-23 13:52:08.00
1, 367, 2002-09-23 12:16:59.00, 01:30:20.00, 2002-09-23 13:47:19.00
1, 355, 2002-09-23 13:40:41.00, 00:16:33.00, 2002-09-23 13:57:14.00
1, 367, 2002-09-23 13:55:57.00, 00:31:34.00, 2002-09-23 14:27:31.00
1, 367, 2002-09-24 10:03:00.00, 00:14:45.00, 2002-09-24 10:17:45.00
1, 367, 2002-09-24 21:59:18.00, 00:23:55.00, 2002-09-24 22:23:13.00
...
```

The rest of this Appendix contains utility programs used to convert one form of the user data to another.

build_16bit_boxcombos5.py

```
#!/usr/bin/python
```

```
""" build_16bit_boxcombos5.py
```

```
Create records with combinations of time, location pairs where time is 'rounded' to 10 minute intervals and only records with duration >= 10 minutes are used.
```

```
This routine reads in a user file (such as userc077.csv) and builds an output file called XXXPbox.dat. The output file contains strings of the timeslot and location (in 10 minute units).
```

```
The input (user) file has the following format:
```

```
0: user
1: AP
2: start time
3: duration
4: end time
```

```
HISTORY:
```

```
30Mar10 Created from build_16bit_combos4.py
```

```
"""
```

```
from mx import DateTime
import sys # for command line arguments
import string
```

Utilities

```
import re
import os
import struct          # for writing in binary
from parse_input_line import parse_input_line
from pickle import load

#base directory name
SOURCE_PATH = r"..\user"
OUTPUT_PATH = r"..\16bit"

# Offsets into user file
USER      = 0
AP        = 1
START     = 2
DURATION  = 3
END       = 4

# Global
dict_timeslot = {}
dict_ap = {}

#
# Error class
#
class MyError(Exception):
    def __init__(self,value):
        self.value = value
    def __str__(self):
        return repr(self.value)

# end of Error class

#
# createFile
# Create a file for this user. Write out the header line.
# INPUTS: user = user number (3 digits)
#         fields= which fields are included (L, T, D)
#
def createFile( user, fields ) :
    filename = (OUTPUT_PATH+"\\%s%sbox.dat" % ( user, fields))
    print "createFile: " + filename
    ofile = open(filename, "wb")
    ofile.write("Date, binary string with timeslot, location pairs\n" )
    return ofile

#
# map_to_dictionary
#
# Given a dictionary, indices, map, and item, map the item to a 2 character
# string.
# Return the mapping and the new indices.
#
def map_to_dictionary(dict, index, max_index, item) :
    special_characters = ['\n',' ',' ',';','\f'] # don't embed these values

    #if we've run out of mapping characters, return
    if index == max_index:
        #
        return END_OF_DAY
        raise MyError,'Out of map chars'

    # don't add it if it's already in the dictionary
    if (not dict.has_key(item)):
        dict[ item] = index
        #print("Mapping %s to 0x%x" % (item, dict[item]))
        # increment to next mapping symbol
        index += 1

    # if either byte of the index maps to a special character,
    # like a linefeed or comma or a control character, skip it
    while ((chr(index & 0x00FF) in special_characters) or \
```

Utilities

```
        chr(index / 0x100) in special_characters or \
        index & 0x00FF < 0x21 or \
        index / 0x100 < 0x21) and \
        index <= max_index):
            index += 1
            #print "Hit a special character, incrementing the index\n"
    if index == max_index :
        print("MAXED OUT");

    return (dict[item], index)    # return mapping

# add_ap_to_dictionary
#
# Add the given access point to the dictionary and determine it's mapping
# Return the mapping.
#
def add_ap_to_dictionary(ap) :
    global dict_ap # dictionary
    return( dict_ap[int( ap) ])

#
# add_timeslot_to_dictionary
#
# FORMERLY: Add the given timeslot to the dictionary and determine it's mapping
# Return the mapping. (Return END_OF_DAY if we're out of available
# mapping chars)
# NOW: Return mapping for a given timeslot string ("hh:mm")
#
def add_timeslot_to_dictionary(ts) :
    global dict_timeslot # dictionary
    return( dict_timeslot[ ts] )

#
# add_duration_to_dictionary
#
# Add the given duration to the dictionary and determine it's mapping
# Return the mapping. (Return END_OF_DAY if we're out of available
# mapping chars)
#
def add_duration_to_dictionary(dur) :
    global dict_duration # dictionary
    global next_dur_index

    (map_symbol, next_dur_index) = \
        map_to_dictionary( dict_duration, \
            next_dur_index, FINAL_DURATION, dur)
    return map_symbol    # return mapping

#
# timeslot
#
# Given a time/date string, return the closest (earlier) timeslot
# within a 1-minute interval. For example, if the input is "09:14.36",
# "09:10" is returned.
# inputs: start = DateTime object containing the starting time
# returns: string containing time (ex. "08:20" )
#
def timeslot( startTime):
    ts = startTime
    if ts.minute in [0,10,20,30,40,50]:
        mins = ts.minute
    else:
        mins = ts.minute/10 * 10
    ts += DateTime.RelativeDateTime(minute=mins, second=0)
    # return the time (hh:mm) portion of the datetime object
    return( str(ts)[11:16])

#
# build_segment
#
# Given a timeslot and a list of tuples, where each tuple contains
# (user, ap, start, dur, end)
# where start,dur,end are DateTime objects and
```

Utilities

```
# user and ap are strings,
# create a string (ex, 'abc') where
# the first 16-bit symbol is the mapped timeslot ('a')
# the second 16-bit symbol is the mapped location
# the third 16-bit symbol is the mapped duration/10
# 6Nov07 - Changed so that the symbols are determined by the 'fields' string
# 'TLD' means timeslot, location, duration
# 7NOV07 - Added 'P' option
#
def build_segment( ts, ll, fields ) :
    segment_string = []

    # Sort the list so that the entry with the longest duration
    # is first
    ll.sort(lambda a,b:cmp(str(b[DURATION]),str(a[DURATION])))

    # Get the location
    location = ll[0][AP]

    # Get the number of 10-minute segments in the duration
    # Divide the duration by 10 minutes
    # (NOTE: dur.minute is the number of minutes past the hour, while
    # dur.minutes is the total number of minutes.
    # For ex. at 02:36:06.00,
    # dur.minute is 36, while dur.minutes is 156.10)
    dur = ll[0][DURATION]
    if int(dur.minute) in [0,10,20,30,40,50]:
        mins = int(dur.minutes)/10
    else:
        mins = int(dur.minutes)/10    # was +1

    for i in fields:
        if (i == 'T' or i == 't'):
            # Map the timeslot
            segment_string.append( add_timeslot_to_dictionary( ts ))
        elif (i == 'L' or i == 'l'):
            # Map the location
            segment_string.append( add_ap_to_dictionary( ll[0][AP]))
        elif (i == 'D' or i == 'd'):
            segment_string.append(add_duration_to_dictionary(mins))
        elif (i == 'P' or i == 'p'):
            segment_string.append( add_poll( ts, mins, location, 'P'))
        elif (i == 'S' or i == 's'):
            segment_string.append( add_poll( ts, mins, location, 'S'))

    return(segment_string)

#
# build_strings
#
# Given a file, do the above for this user.
#
# INPUT: filename (on one user's file)
#         fields = which fields (location, start time, duration) to include
# OUTPUT: results written to an output file
#
def build_strings( filename, fields ) :
    global dict_ap, dict_timeslot, dict_duration    # dictionaries

    print "Reading file " + filename
    ifile = open(filename, "r")

    # Initialize values
    first = True    # flag to tell if we're on the first line
    enoughForOutput = False    # true when an output file has been created.
    location_string = []    # array containing one day's time & locations
    ll = []    # List of all records in this timeslot

    # Walk through file, building a string for each day
```

Utilities

```

for line in ifile.readlines():
    # HERE IS WHERE WE SKIP RECORDS WHERE
    # DURATION IS LESS THAN 10 MINUTES!!!
    temp = parse_input_line(line)
    if int(temp[DURATION].minutes) < 10:
        continue # onto next line
    prev = parse_input_line( line )
    current_date = str(prev[START])[0:10]
    current_timeslot = timeslot( prev[START])

    # Create output file if needed
    if (first):
        # Create output file using user number
        ofile = createFile( "%03d" % (int(prev[USER])), fields)
        first = False
        prev_date = current_date
        working_timeslot = current_timeslot
        enoughForOutput = True

    # If the timeslot has changed, create the string for
    # the working timeslot
    if (current_timeslot != working_timeslot or \
        prev_date != current_date) :
        # if there is something in the list
        if len(ll) > 0 :
            # orig: location_string.append(build_segment( working_timeslot, ll,
fields))

            for ii in build_segment( working_timeslot, ll, fields):
                location_string.append(ii)

            # Clear data to start on new timeslot
            working_timeslot = current_timeslot
            ll = []
        if int(prev[DURATION].minutes) >= 10:
            ll.append( prev ) # add this line to the list

    # If the date has changed, write out the last day's worth of info
    if (prev_date != current_date ) and (len(location_string)>0):
        ofile.write("%s," % (prev_date))
        for bb in location_string:
            # use this for L,T,D => bbb = bb
            # And comment out the following line:
            for bbb in bb:
                # convert to unsigned short(2 bytes)
                data = struct.pack('H',bbb)
                ofile.write(data)
        ofile.write("\n")
        # Clear location string
        location_string = []
        prev_date = current_date

    # End of loop to read file

# If the list is not empty, add the current info to the location string
if len(ll) > 0 :
    location_string += build_segment( working_timeslot, ll, fields)
    ofile.write("%s," % (prev_date))
    for bb in location_string:
        for bbb in bb:
            data = struct.pack('H',bbb)
            ofile.write(data)
    ofile.write("\n")
ifile.close() # Close input file
if enoughForOutput:
    ofile.close()
## end of build_strings() #####
#
# increment_ts
# Given a timestring (like "08:10") return the string
# with 10 minutes added to it (ex. "08:20")

```


Utilities

```
# if the ts is ("23:50"), don't increment (to next day)
def increment_ts( ts):

    if (ts == "23:50") :
        return(ts)

    # Convert ts string to a list
    tl = []
    for i in ts:
        tl.append(i)

    tl[3] = str(int(tl[3]) + 1)          # increment 10-minute digit
    if tl[3] == '6':
        # rollover and increment hours
        tl[3] = '0'
        if tl[1] == '9':
            tl[1] = '0'
            tl[0] = str(int(tl[0])+1)
        else:
            tl[1] = str(int(tl[1]) + 1)    # increment 1s digit of hours

    # Convert list back into a string
    next_ts = ""
    for i in tl:
        next_ts += i
    #print "Incrementing %s to %s\n" % ( ts, next_ts)
    return( next_ts)

# add_poll
# Given a starting time (ts), the number of 10-minute units (dur_units)
# and a location (ap), build a string of encoded time, loc, time, loc
# until the duration has expired.
# For example, if ts = "08:10" and dur_units = 3 and loc = '37',
# the returned string would have 3 entries,
# 08:10, '37', 08:20, '37', 08:30, '37'
# (Of course, these entries are all mapped)
#
# 13NOV07 Added field.  If field=='P', add the timeslot, else just do loc
#
def add_poll( ts, dur_units, ap, field):
    poll_string = []                      # The string we are building
    loc = add_ap_to_dictionary(ap)        # map the location
    tss = ts                              # get a local copy of the time
    for i in range(dur_units):
        #print("add_poll: %s %s" % (tss,ap))
        if (field == 'P'):
            poll_string.append( add_timeslot_to_dictionary(tss) )
        poll_string.append( loc )
        tss = increment_ts(tss)           # Add 10-minutes to the time_
    return(poll_string)

def test():
    # TO TEST, JUST DO ONE USER
    full_filename = SOURCE_PATH+'\\'+ "userc034.csv"
    build_strings( full_filename, "P")

""" Main
"""
print "usage: .\\build_16bit_boxcombos3.py"
print "Output is in %s\n" % (OUTPUT_PATH)
startTime = DateTime.now()
print "Started at " + str(startTime) + "\n"

# argv[1] is a string containing the fields and order
# ex. "LTD" means LOC, START TIME, DURATION in that order
# if "P" is in the arg string, then it can't have the other choices
# if len(sys.argv[1]) == 0:
#     printf("No fields specified")
#     exit(0)
```

Utilities

```
# Read in the dictionaries for access points and timeslots

f = open('dictionaryAP.dat')
dict_ap = load(f)
f.close()

f = open('dictionaryTimeslots.dat')
dict_timeslot = load(f)
f.close()

"""
test()                # test on one file

"""
# For each user
for file in os.listdir(SOURCE_PATH):
    # find files that match "userc*.csv"
    if re.search(r".csv",file) and re.search("userc",file):
        full_filename = SOURCE_PATH+'\\'+file
        if os.path.getsize(full_filename) > 0 :
            build_strings( full_filename, "P")    # sys.argv[1] )
print ("Time to process: %s\n" % str(DateTime.now() - startTime))

# END OF FILE
```

count_data_stats.py

```
#!/usr/bin/python
```

```
""" count_data_stats.py

This routine reads in an input file used for training or testing and outputs information about
the file, namely
- the number of unique locations (& possibly counts for each)
- the number of unique timeslots (and possibly counts for each)
- the number of timeslot, location entries.
```

Currently, this routine assumes that the input file (ex 077wks01_05.dat) has the form t,l,t,l where t and l are both 16-bit values and the t is a timeslot (ex. 9:00) and l is a location.

(To get other information, like the mappings, you would need the original input file that was used in pull_weeks_binary.py).

Usage:
./count_data_stats.py <filename>

The counts are output to the screen.

HISTORY:
21Apr08 Created. (from pull_weeks_binary.py)
28Apr08 Copied from count_data_stats.py and wrote out more info.
Then copied to count_data_stats.py and changed info output

```
"""

#from mx import DateTime
import string
import re
import os
import sys
import struct          # for reading and writing binary

#base directory name
#SOURCE_PATH = r"c:\cygwin\home\iburbey\crawdad\ucsd\user\output\binboxstrings"
#OUTPUT_PATH = r"c:\cygwin\home\iburbey\crawdad\ucsd\user\output\binboxstrings"

#####
# count_entries
#
# Given a file, do the above for this user.
```

Utilities

```
#
# INPUT: filename (on one user's file)
# OUTPUT: results
#
def count_entries( filename ) :

    dict_ap          = {}      # dictionary of access points (translated) & counts
    dict_timeslot    = {}      # dictionary of timeslots

    #print "Reading file " + filename
    ifile = open(filename, "rb")

    # The file has only one line, which ends with a trailing newline.
    line = str.rstrip( ifile.readline() ) # read line & strip trailing newline

    length = len(line)

    # Take the characters in the string two by two and convert to 16bit value
    i = 0
    while i < len(line):
        time_chars = line[i] + line[i+1]      # LSB first
        time = struct.unpack('H', time_chars)  # convert to 16bit value
        if dict_timeslot.has_key( time ) :
            dict_timeslot[time] += 1
        else :
            dict_timeslot[time] = 1
        # add to dictionary
        loc_chars = line[i+2]+line[i+3]        # LSB first
        loc = struct.unpack('H',loc_chars)
        if dict_ap.has_key( loc):
            dict_ap[ loc ] += 1
        else:
            dict_ap[ loc] = 1
        i += 4

    ifile.close()

    # To print out the values and counts in the dict, uncomment this section
    #print 'times:'
    #print dict_timeslot
    #print 'locs:'
    #print dict_ap

    # calculate percentage for each location (the level 0 table)
    # iterate through dictionary to get total of location counts.
    total_ap_count = 0
    max_ap_count = 0
    print                                     # linefeed
    for ii in dict_ap:
        total_ap_count += dict_ap[ii]
        if dict_ap[ii] > max_ap_count :
            max_ap_count = dict_ap[ii]
    #    print ii, total_ap_count

    #for ii in dict_ap:
    #    print "loc: %s count: %d %.1f%%" % (hex(int(ii[0])),
dict_ap[ii],100.*dict_ap[ii]/total_ap_count)

    if (total_ap_count > 0) :
        print '%s, %d, %d, %d, %.1f' % \
            (filename, length/4, len(dict_timeslot), len(dict_ap), \
            100.*max_ap_count/total_ap_count ),          # no line feed at end
    else :
        print '%s, %d, %d, %d, 0.0' % \
            (filename, length/4, len(dict_timeslot), len(dict_ap)),

##### end of cull_weeks #####

def test():
    # TO TEST, JUST DO ONE USER
```

Utilities

```
full_filename = '032wks01_05.dat'
count_entries( full_filename)

""" Main
"""
# Check command line arguments
if (len(sys.argv) < 2) :
    print("Missing a command line argument")
    print "Usage: .\count_data_stats.py <filename>"
    print "      where <filename> begins with the 3-digit user number"
    sys.exit(1)

# Parse command line arguments
filename = sys.argv[1]
user = filename[0:3]
count_entries( filename)
# END OF FILE #####
```

entropy.py

```
#!/usr/bin/python
```

```
""" entropy.py
```

Calculate Entropy values training files.

NOTE: Output is only one element in XML -- The XML wrappers need to be added for the file to be read by EXCEL.
Output is in XML.

(To get other information, like the mappings, you would need the original input file that was used in pull_weeks_binary.py).

Usage:

```
./entropy.py <filename>
```

The counts are output to the screen.

HISTORY:

21Apr08 Created. (from pull_weeks_binary.py)

28Apr08 Copied from count_data_stats.py and wrote out more info.

Then copied to count_data_stats.py and changed info output

1Apr10 Changed to xml

5Apr10 Changed name from data_stats_xml.py to file_stats_xml.py to be clearer. Added type and source to element name

2Nov10 Copied from file_stats_xml.py and modified to calc entropy

16Nov10 Added conditional entropy calculation to 16bit module

28Jan11 Changed name of 'actualEntropy' to 'pairsEntropy'

```
"""
```

```
#from mx import DateTime
import string
import re
import os
import sys
import struct          # for reading and writing binary
import math
```

```
#
```

```
#      GLOBAL VARIABLES
```

```
#
```

```
dict_ap = {}    # dictionary of access points (locations)
dict_timeslot = {}    # dictionary of times
dict_tuples = {}    # dictionary of (time,loc) tuples
dict_transitions = {} # dictionary of transitions between locations
numPairs = 0    # Number of (time,location) tuples in file
```

Utilities

```
#####
# count_entries_16bit
#
# Given a file, do the above for this user.
#
# INPUT: filename (on one user's file)
#       file_type ("16bit" or "seq")
#       file_source ("MoveLoc" or "SeqLoc")
# OUTPUT: results
#
def count_entries_16bit( filename, file_type, file_source) :
    global dict_ap, dict_timeslot, dict_tuples, numPairs

    #print "Reading file " + filename
    ifile = open(filename, "rb")

    # The file has only one line, which ends with a trailing newline.
    line = str.rstrip( ifile.readline()) # read line & strip trailing newline

    length = len(line)

    # Take the characters in the string two by two and convert to 16bit value
    i = 0
    while i < len(line):
        time_chars = line[i] + line[i+1]      # LSB first
        time = struct.unpack('H', time_chars)  # convert to 16bit value
        time=time[0]
        if dict_timeslot.has_key( time) :
            dict_timeslot[time] += 1
        else :
            dict_timeslot[time] = 1
        # add to dictionary
        loc_chars = line[i+2]+line[i+3]        # LSB first
        loc = struct.unpack('H',loc_chars)
        loc=loc[0]
        if dict_ap.has_key( loc):
            dict_ap[ loc ] += 1
        else:
            dict_ap[ loc] = 1

        # build the (time,loc) tuple and add to dictionary
        tup = time,loc
        if dict_tuples.has_key(tup):
            dict_tuples[ tup] += 1
        else:
            dict_tuples[ tup] = 1

        i += 4
    ifile.close()

    numPairs = length/4

#####
# count_entries_seq
#
# Given a file, do the above for this user.
#
# INPUT: filename (on one user's file)
#       file_type ("16bit" or "seq")
#       file_source ("MoveLoc" or "SeqLoc")
# OUTPUT: results
#
def count_entries_seq( filename, file_type, file_source) :
    global dict_ap, dict_timeslot, numPairs

    total_length = 0      # total length of time.loc seq

    #print "Reading file " + filename
    ifile = open(filename, "rb")
```

Utilities

```

for line in ifile:
    line = line.strip()    # remove trailing newline
    if line=="":
        break
    #each line has the format date; wk_num; day-of-week; time,loc,time,loc,...
    # we want just the time,loc,time,loc list
    str_date, str_wknum, str_dow, str_seq = line.split(';')

    # now split the time,loc,time,loc string into a list
    ll_seq = str_seq.split(',')
    length = len(ll_seq)
    total_length += length

    # add each element to it's corresponding dictionary
    last_loc = ""
    for i in range(0, length, 2):
        # Add time to dictionary
        time = ll_seq[i]
        if dict_timeslot.has_key( time) :
            dict_timeslot[time] += 1
        else :
            dict_timeslot[time] = 1

        # add ap to dictionary
        loc = ll_seq[i+1]
        if dict_ap.has_key( loc):
            dict_ap[ loc ] += 1
        else:
            dict_ap[ loc] = 1

        # build the (time,loc) tuple and add to dictionary
        tup = time,loc
        if dict_tuples.has_key(tup):
            dict_tuples[ tup] += 1
        else:
            dict_tuples[ tup] = 1

        # build the (last_loc,loc) tuple and add it
        # to the dictionary of transitions.
        loc_tup = last_loc, loc
        if dict_transitions.has_key(loc_tup):
            dict_transitions[loc_tup] += 1
        else:
            dict_transitions[loc_tup]=1
        last_loc = loc

ifile.close()
numPairs = total_length/2

#####
#
#     generateOutput
#
#     Output values in XML format.
#
#####
def generateOutput(file_type):
    # To print out the values and counts in the dict, uncomment this section
    #print 'times:'
    #print dict_timeslot
    #print 'locs:'
    #print dict_ap

    #
    # Output results in XML (to the std output)
    #
    #print '<?xml version="1.0"?>'    # XML declaration
    print '    <FileStats filename="%s\" type="%s\" source="%s\">' % \
        (filename, file_type, file_source)
    print '\t<NumPairs>%d</NumPairs>' % (numPairs)
    print '\t<NumUniqueTimes>%d</NumUniqueTimes>' % len(dict_timeslot)

```

Utilities

```
print '\t<NumUniqueLocs>%d</NumUniqueLocs>' % len(dict_ap)

#
# Calculate RandomEntropyLoc
#
randomEntropyLoc = 0.0
randomEntropyLoc = math.log(len(dict_ap),2)
print '\t<randomEntropyLoc>%f</randomEntropyLoc>' % (randomEntropyLoc)

#
# Calculate randomEntropyTime
#
randomEntropyTime = 0.0
randomEntropyTime = math.log(len(dict_timeslot),2)
print '\t<randomEntropyTime>%f</randomEntropyTime>' % (randomEntropyTime)

#
# Calculate uncEntropyLoc
#
log = 0.0
prob = 0.0
sum = 0.0
for value in dict_ap.itervalues():
    prob = float(value)/numPairs
    log = math.log(prob,2)
    sum += prob * log
sum = -sum
print '\t<uncEntropyLoc>%f</uncEntropyLoc>' % (sum)

#
# Calculate uncEntropyTime
#
log = 0.0
prob = 0.0
sum = 0.0
for value in dict_timeslot.itervalues():
    prob = float(value)/numPairs
    log = math.log(prob,2)
    sum += prob * log
sum = -sum
print '\t<uncEntropyTime>%f</uncEntropyTime>' % (sum)

#
# Calculate Actual Entropy
#
log = 0.0
prob = 0.0
sum = 0.0
for value in dict_tuples.itervalues():
    prob = float(value)/numPairs
    log = math.log(prob,2)
    sum += prob * log
sum = -sum
print '\t<pairsEntropy>%f</pairsEntropy>' % (sum)

#
#
# Calculate Conditional Entropy
#
#at each timeslot, for each location calculate prob and entropy
sum_outer=0.0
for time in dict_timeslot.iterkeys():
    dd_loc={} # temp dictionary for counters
    for ti, li in dict_tuples.iterkeys():
        if ti == time:
            dd_loc[li] = dict_tuples[(ti,li)]
    #print "for time = %d, dd_loc: " % (time),
    #print dd_loc

    #for each location, calculate math
```

Utilities

```
log = 0.0
prob = 0.0
sum_inner = 0.0
denom = dict_timeslot[time]
for value in dd_loc.itervalues():
    #print '\tfor location ', value, ' at time ', time
    #print '\tcount = ', value, ' , denom= ', denom
    prob = float(value)/denom
    #print '\tprob: ', prob
    log = math.log(prob,2)
    #print '\tlog: ', log
    sum_inner += prob * log
    #print '\tsum_inner: ', sum_inner
prob_time = float(dict_timeslot[time])/numPairs
#print ' prob_time: ', prob_time
sum_outer += prob_time * sum_inner
#print ' sum_outer: ', sum_outer
sum_outer = -sum_outer
print '\t<conditionalEntropy>%f</conditionalEntropy>' % (sum_outer)

#
# For SEQ format, output transitionEntropy
# Entropy between location transitions
#
if (file_type.strip() == 'seq'):
    log = 0.0
    prob = 0.0
    sum = 0.0
    for value in dict_transitions.itervalues():
        prob = float(value)/numPairs
        log = math.log(prob,2)
        sum += prob * log
    sum = -sum
    print '\t<locTransEntropy>%f</locTransEntropy>' % (sum)

print ' </FileStats>'

def test():
    global file_source, full_filename
    # TO TEST, JUST DO ONE USER
    file_type = "16bit"
    file_source = "MoveLoc/entropy"
    full_filename = '100wks1_2.dat'
    count_entries_16bit( full_filename, file_type, file_source)
    generateOutput(file_type) # write the output xml

# test seq files
#count_entries_seq('032wks1_5.cvs', "seq", "SigLoc")
#generateOutput(file_type) # write the output xml

#test() # to test

""" Main
"""
# Check command line arguments
if (len(sys.argv) < 3) :
    print("Missing a command line argument")
    print "Usage: .\\file_stats_xml.py <filename> <file_type> <source>"
    print " where <filename> begins with the 3-digit user number"
    print " <file_type> is '16bit' or 'seq'"
    sys.exit(1)

# Parse command line arguments
filename = sys.argv[1]
user = filename[0:3]
file_type = sys.argv[2]
file_source = os.path.abspath('') #sys.argv[3]
```


Utilities

```
if file_type.strip() == "16bit":
    count_entries_16bit( filename, file_type, file_source)
else:
    count_entries_seq( filename, file_type, file_source)
generateOutput(file_type)      # write the output xml

# END OF FILE #####
```

file_stats_xml.py

```
#!/usr/bin/python
```

```
""" file_stats_xml.py
```

This routine reads in an input file used for training or testing and outputs information about the file, namely

NOTE: Output is only one element in XML -- The XML wrappers need to be added for the file to be read by EXCEL.
Output is in XML.

Currently, this routine assumes that the input file (ex 077wks01_05.dat) has the form t,l,t,l where t and l are both 16-bit values and the t is a timeslot (ex. 9:00) and l is a location.

(To get other information, like the mappings, you would need the original input file that was used in pull_weeks_binary.py).

Usage:

```
./file_stats_XML.py <filename>
```

The counts are output to the screen.

HISTORY:

21Apr08 Created. (from pull_weeks_binary.py)

28Apr08 Copied from count_data_stats.py and wrote out more info.

Then copied to count_data_stats.py and changed info output

1Apr10 Changed to xml

5Apr10 Changed name from data_stats_xml.py to file_stats_xml.py to be clearer. Added type and source to element name

```
"""
```

```
#from mx import DateTime
import string
import re
import os
import sys
import struct          # for reading and writing binary
```

```
#####
```

```
# count_entries_16bit
```

```
#
```

```
# Given a file, do the above for this user.
```

```
#
```

```
# INPUT: filename (on one user's file)
```

```
#       file_type ("16bit" or "seq")
```

```
#       file_source ("MoveLoc" or "SeqLoc")
```

```
# OUTPUT: results
```

```
#
```

```
def count_entries_16bit( filename, file_type, file_source) :
```

```
    dict_ap          = {}      # dictionary of access points (translated) & counts
```

```
    dict_timeslot    = {}      # dictionary of timeslots
```

```
    #print "Reading file " + filename
```

```
    ifile = open(filename, "rb")
```

```
    # The file has only one line, which ends with a trailing newline.
```

```
    line = str.rstrip( ifile.readline()) # read line & strip trailing newline
```

Utilities

```

length = len(line)

# Take the characters in the string two by two and convert to 16bit value
i = 0
while i < len(line):
    time_chars = line[i] + line[i+1]      # LSB first
    time = struct.unpack('H', time_chars)  # convert to 16bit value
    if dict_timeslot.has_key( time) :
        dict_timeslot[time] += 1
    else :
        dict_timeslot[time] = 1
    # add to dictionary
    loc_chars = line[i+2]+line[i+3]        # LSB first
    loc = struct.unpack('H',loc_chars)
    if dict_ap.has_key( loc):
        dict_ap[ loc ] += 1
    else:
        dict_ap[ loc] = 1
    i += 4

ifile.close()

# To print out the values and counts in the dict, uncomment this section
#print 'times:'
#print dict_timeslot
#print 'locs:'
#print dict_ap

# calculate percentage for each location (the level 0 table)
# iterate through dictionary to get total of location counts.
total_ap_count = 0
max_ap_count = 0
#print                                # linefeed
for ii in dict_ap:
    total_ap_count += dict_ap[ii]
    if dict_ap[ii] > max_ap_count :
        max_ap_count = dict_ap[ii]
#    print ii, total_ap_count

#for ii in dict_ap:
#    print "loc: %s count: %d %.1f%%" % (hex(int(ii[0])),
dict_ap[ii],100.*dict_ap[ii]/total_ap_count)

"""
if (total_ap_count > 0) :
    print '%s, %d, %d, %d, %.1f' % \
        (filename, length/4, len(dict_timeslot), len(dict_ap), \
        100.*max_ap_count/total_ap_count ),          # no line feed at end
else :
    print '%s, %d, %d, %d, 0.0' % \
        (filename, length/4, len(dict_timeslot), len(dict_ap)),
"""

# Output results in XML (to the std output)
#print '<?xml version="1.0"?>'          # XML declaration
print '    <FileStats filename="%s\" type="%s\" source="%s\">' % \
    (filename, file_type, file_source)
print '\t<NumPairs>%d</NumPairs>' % (length/4)
print '\t<NumUniqueTimes>%d</NumUniqueTimes>' % len(dict_timeslot)
print '\t<NumUniqueLocs>%d</NumUniqueLocs>' % len(dict_ap)
if (total_ap_count > 0):
    print '\t<PercAtMostPopularAP>%.1f</PercAtMostPopularAP>' \
        % (100.*max_ap_count/total_ap_count)
print '    </FileStats>'

#####
#####
# count_entries_seq
#
# Given a file, do the above for this user.
#

```

Utilities

```
# INPUT: filename (on one user's file)
#       file_type ("16bit" or "seq")
#       file_source ("MoveLoc" or "SeqLoc")
# OUTPUT: results
#
def count_entries_seq( filename, file_type, file_source) :

    dict_ap      = {}      # dictionary of access points (translated) & counts
    dict_timeslot = {}      # dictionary of timeslots

    #print "Reading file " + filename
    ifile = open(filename, "rb")

    for line in ifile:
        line = line.strip()      # remove trailing newline
        if line=="":
            break
        #each line has the format date; wk_num; day-of-week; time,loc,time,loc,...
        # we want just the time,loc,time,loc list
        str_date, str_wknum, str_dow, str_seq = line.split(';')

        # now split the time,loc,time,loc string into a list
        ll_seq = str_seq.split(',')
        length = len(ll_seq)

        # add each element to it's corresponding dictionary
        for i in range(0, length, 2):
            # Add time to dictionary
            time = ll_seq[i]
            if dict_timeslot.has_key( time) :
                dict_timeslot[time] += 1
            else :
                dict_timeslot[time] = 1

            # add ap to dictionary
            loc = ll_seq[i+1]
            if dict_ap.has_key( loc):
                dict_ap[ loc ] += 1
            else:
                dict_ap[ loc] = 1

    ifile.close()

    # To print out the values and counts in the dict, uncomment this section
    #print 'times:'
    #print dict_timeslot
    #print 'locs:'
    #print dict_ap

    # calculate percentage for each location (the level 0 table)
    # iterate through dictionary to get total of location counts.
    total_ap_count = 0
    max_ap_count = 0
    #print                                     # linefeed
    for ii in dict_ap:
        total_ap_count += dict_ap[ii]
        if dict_ap[ii] > max_ap_count :
            max_ap_count = dict_ap[ii]
    #    print ii, total_ap_count

    #for ii in dict_ap:
    #    print "loc: %s count: %d %.1f%%" % (hex(int(ii[0])),
    dict_ap[ii],100.*dict_ap[ii]/total_ap_count)

    """
    if (total_ap_count > 0) :
        print '%s, %d, %d, %d, %.1f' % \
            (filename, length/4, len(dict_timeslot), len(dict_ap), \
              100.*max_ap_count/total_ap_count ),          # no line feed at end
    else :
        print '%s, %d, %d, %d, 0.0' % \
            (filename, length/4, len(dict_timeslot), len(dict_ap)),
```

Utilities

```
"""

# Output results in XML (to the std output)
#print '<?xml version="1.0"?>'          # XML declaration
print '    <FileStats filename="%s\" type="%s\" source="%s\">' % \
      (filename, file_type, file_source)
print '\t<NumPairs>%d</NumPairs>' % (length/4)
print '\t<NumUniqueTimes>%d</NumUniqueTimes>' % len(dict_timeslot)
print '\t<NumUniqueLocs>%d</NumUniqueLocs>' % len(dict_ap)
if (total_ap_count > 0):
    print '\t<PercAtMostPopularAP>%.1f</PercAtMostPopularAP>' \
          % (100.*max_ap_count/total_ap_count)
print '    </FileStats>'

def test():
    # TO TEST, JUST DO ONE USER
    file_type = "16bit"
    file_source = "MoveLoc"
    full_filename = '032wks01_05.dat'
    count_entries_16bit( full_filename, file_type, file_source)

""" Main
"""
# Check command line arguments
if (len(sys.argv) < 3) :
    print("Missing a command line argument")
    print "Usage: .\\file_stats_xml.py <filename> <file_type> <source>"
    print "      where <filename> begins with the 3-digit user number"
    print "      <file_type> is '16bit' or 'seq'"
    sys.exit(1)

# Parse command line arguments
filename = sys.argv[1]
user = filename[0:3]
file_type = sys.argv[2]
file_source = os.path.abspath('')      #sys.argv[3]
if file_type.strip() == "16bit":
    count_entries_16bit( filename, file_type, file_source)
else:
    count_entries_seq( filename, file_type, file_source)

# END OF FILE #####
```

parse_input_line.py

```
from mx import DateTime

#      parse_input_line
# parse a line from the input file into (user, ap, start, duration, end)
# INPUTS: line
# OUTPUTS: tuple: (user, ap, start, duration, end)
# The start, duration and end fields are DateTime objects.
#
def parse_input_line( line ):
#      print "Parsing line: %s" % (line)
    if (line == "") :
        # hit the end of the file
        pUser = ap3 = start = dur = end = ""
    else:
        pUser, ap3, pStartTime, pDuration, pEnd = line.split(',')
        start = DateTime.DateTimeFrom(pStartTime)
        dur = DateTime.DateTimeDeltaFrom(pDuration)
        end = DateTime.DateTimeFrom(pEnd)
    return( pUser, str.strip(ap3), start, dur, end)
```

test_stats_xml.py

```
#!/usr/bin/python

""" test_stats_xml.py

This routine reads in two files: a testing file and a training file and outputs
the number of locations and the number of times that are in the test file that are NOT in the
training file.
NOTE: Output is only one element in XML -- The XML wrappers need to be added for the file to be
read by EXCEL.
Output is in XML.

(To get other information, like the mappings, you would need the original input file that was
used in pull_weeks_binary.py).

Usage:
    ./test_stats_XML.py <train_filename> <test_filename> <type>

The counts are output to the screen.

HISTORY:
    5Apr10 Created (from file_stats_XML.py)
"""

#from mx import DateTime
import string
import re
import os
import sys
import struct          # for reading and writing binary

#####
# count_entries_16bit
#
#    Given a 16bit file and two dictionaries,
#    fill in the location and time dictionary
#
# INPUT: filename, dictionary for aps, dictionary for times
# OUTPUT: dictionaries are filled in.
#
def count_entries_16bit( filename, dict_ap, dict_timeslot ) :
    #print "Reading file " + filename
    ifile = open(filename, "rb")

    # The file has only one line, which ends with a trailing newline.
    line = str.rstrip( ifile.readline() ) # read line & strip trailing newline

    length = len(line)

    # Take the characters in the string two by two and convert to 16bit value
    i = 0
    while i < len(line):
        time_chars = line[i] + line[i+1]          # LSB first
        time = struct.unpack('H', time_chars)      # convert to 16bit value
        if dict_timeslot.has_key( time ) :
            dict_timeslot[time] += 1
        else :
            dict_timeslot[time] = 1
        # add to dictionary
        loc_chars = line[i+2]+line[i+3]            # LSB first
        loc = struct.unpack('H',loc_chars)
        if dict_ap.has_key( loc ):
            dict_ap[ loc ] += 1
        else:
            dict_ap[ loc] = 1
        i += 4

    ifile.close()
```

Utilities

```
# To print out the values and counts in the dict, uncomment this section
#print 'times:'
#print dict_timeslot
#print 'locs:'
#print dict_ap
#####
# count_entries_seq
#
# Given a seq file, build the ap and times dictionaries
#   for that file
#
# INPUT: filename, dict for aps, dict for times.
# OUTPUT: dict_ap and dict_timeslot are filled in.
#
#####
def count_entries_seq( filename, dict_ap, dict_timeslot) :
    #print "Reading file " + filename
    ifile = open(filename, "rb")

    for line in ifile:
        line = line.strip()    # remove trailing newline
        if line=="":
            break
        #each line has the format date; wk_num; day-of-week; time,loc,time,loc,...
        # we want just the time,loc,time,loc list
        str_date, str_wknum, str_dow, str_seq = line.split(';')

        # now split the time,loc,time,loc string into a list
        ll_seq = str_seq.split(',')
        length = len(ll_seq)

        # add each element to it's corresponding dictionary
        for i in range(0, length, 2):
            # Add time to dictionary
            time = ll_seq[i]
            if dict_timeslot.has_key( time) :
                dict_timeslot[time] += 1
            else :
                dict_timeslot[time] = 1

            # add ap to dictionary
            loc = ll_seq[i+1]
            if dict_ap.has_key( loc):
                dict_ap[ loc ] += 1
            else:
                dict_ap[ loc] = 1

    ifile.close()

    # To print out the values and counts in the dict, uncomment this section
    #print 'times:'
    #print dict_timeslot
    #print 'locs:'
    #print dict_ap
#####
#
#   compare_dict
#
#   Compare two dictionaries
#
# INPUT: dict_train is dictionary for train file
#         (locations or times)
#   dict_test is dictionary for testing file
#         (locations or times)
# RETURNS: number of keys in the test_dict that are
#   not in the train_dict
#####
def compare_dicts( dict_train, dict_test):
    countNotFound = 0
    for test_key in dict_test.keys():
        if not dict_train.has_key( test_key):
            countNotFound += 1
```

Utilities

```
        return countNotFound

#####
# Main
#####
#
# Check command line arguments
#
if (len(sys.argv) < 4) :
    print("Missing a command line argument")
    print "Usage: .\\test_stats_xml.py <train_filename> <test_filename> <file_type>"
    print "      where <train_filename> is the training file"
    print "      where <test_filename> is the testing file"
    print "      <file_type> is '16bit' or 'seq'"
    sys.exit(1)

#
# Parse command line arguments
#
train_filename = sys.argv[1]
test_filename = sys.argv[2]
file_type = sys.argv[3]
file_source = os.path.abspath('')      #sys.argv[3]
#
# Initialize dictionaries
#
dict_train_aps = {}
dict_train_timeslots = {}
dict_test_aps = {}
dict_test_timeslots = {}
#
# Fill in the dictionaries for the trainin and testing files.
#
if file_type.strip() == "16bit":
    count_entries_16bit( train_filename, dict_train_aps, dict_train_timeslots)
    count_entries_16bit( test_filename, dict_test_aps, dict_test_timeslots)
else:
    count_entries_seq( train_filename, dict_train_aps, dict_train_timeslots)
    count_entries_seq( test_filename, dict_test_aps, dict_test_timeslots)

#
# Compare the dictionaries to find how many entries that are in the test
# file are missing from the training file.
NumLocsNotFound = compare_dicts( dict_train_aps, dict_test_aps)
NumTimesNotFound = compare_dicts( dict_train_timeslots, dict_test_timeslots)
#
# Output to XML
#
print '    <TestStats train_filename="%s\" test_filename="%s\" type="%s\" source="%s\">' % \
    (train_filename, test_filename, file_type, file_source)
print '\t<NumTestLocsNotFound>%d</NumTestLocsNotFound>' % (NumLocsNotFound)
print '\t<NumTestTimesNotFound>%d</NumTestTimesNotFound>' % (NumTimesNotFound)
print '    </TestStats>'
# END OF FILE #####
```

Appendix F SEQ Model Source Code

seq_model.py

```
#!/usr/bin/python

"""seq_model.py

Read in a sequence file and built a list of all the sequences in the file.
nd then predict.
The input file contains a string of time,location pairs. See training_seq.txt for a simple
example.

ASSUMPTIONS:
This code assumes that it run in the same directory as the sequence files.
This code assumes that the testing file contains only 1 week's worth of data. It reads the week
number from the first line and uses that week number of all of the predictions in that testing
data file.

HISTORY:
27Jan2010 Ink Created.
14Feb2010 ink Changed so that input files include date, day of week, week #
25Feb2010 ink Allow you to predict against the same week you've trained on
1Mar2010 ink Added 'QuestLocationNotFound' flag. If this is true, then the correct result is to
refuse to predict, which should be counted as a correct answer.
1Mar2010 Ink Added count of how many wrong predictions were within 10 minutes and how many were
within 20 minutes
1Mar2010 ink Added counters. Once the training sequences have been filtered, count how many
time,loc pairs are in the resulting list and how many unique locations.
20Apr2010 ink Output in XML
3May2010 ink Write out number of predictions returned per test to output file.
5May2010 ink 10, 20 minute test was only correct for SigLoc data so I fixed it.
Added TimeDelta output (the number of minutes between the best prediction and the
actual time).
19Oct2010 ink Added support and confidence calculations. support = The number of times the
context is seen / # training sequences. (e.g. "I've seen 'OFFICE' in 12 out of 24 sequences.)
confidence = # times I saw the prediction/ # times I've seen the context (e.g. I saw "Office at
8:00" 10 times/I saw "office" 12 times).
I am no longer counting "refuse to predict" as a correct prediction!
22Feb2011 ink Actually added aveSupport and aveConfidence. and numUniqueTimes. (Name is
incorrect, since I'm not calculating averages but summing numerators and denominators.)
"""

# Load Python packages
import sys # for command line arguments
import re # real expressions
import os # for handling files and directories
import getopt # for command-line arguments in the unix style
from mx import DateTime # for calculating diffs between two times.

# Global Variables
list_seq = [] # list of training sequences
# [date, day-of-week, week# [time,loc tuples]]
list_sub_seq = [] # list of training sequences that fit week, dow criteria
# [(time,loc tuples), (more time,loc tuples)]
verbose = False # true for verbose output
str_order = "" # The order to use (0, 1time, 1loc, 2)
str_trainingfile = "" # input training file
str_testingfile = "" # file to test against (to predict)
num_preceding_weeks = 5 # number of preceding weeks to test against.
```


SEQ Model Source Code

```
quest_location_not_found = True      # If this is true, the model should refuse to predict.
numerSupport=0                      # numerator of support calculation (# sequences containing the given
quest_location AND the context)

#####
#
#       Error Class
#       For formatting error messages.
#
#####
class MyError(Exception):
    def __init__(self,value):
        self.value = value
    def __str__(self):
        return repr(self.value)
# end of Error class

#####
#
#       usage
#       print out how to call the program
#####3
def usage():
    print "Usage: seq_model <arguments>"
    print "\t-o, --order, order [0,1time,1loc, 2]"
    print "\t-f, --filename, input (training file)"
    print "\t-p, --predict, file to test against"
    print "\t-w, --weeks, for number of preceding weeks to use"
    print "\t-v, --verbose for verbose output"
    return

#####
#
#       getCommandLineArgs
#
#       Read the command line arguments, check
#       them for validity.
#
#####
def getCommandLineArgs(argv):
    global verbose, str_order, str_trainingfile, str_testingfile
    global num_preceding_weeks

    try:
        opts, args = getopt.getopt( argv,
"o:f:p:w:v",["order=", "file=", "predict=", "weeks=", "verbose"])
    except getopt.GetoptError:
        usage()
        sys.exit(2)

    for opt, arg in opts:
        #print opt, arg

        if opt in ("-o", "--order"):
            str_order=arg
        elif opt in ("-f", "--file"):
            str_trainingfile= arg
        elif opt in ("-p", "--predict"):
            str_testingfile= arg
        elif opt in ("-v", "--verbose"):
            verbose = True
        elif opt in ("-w", "--weeks"):
            num_preceding_weeks = int(arg)

    # Check that required arguments are present.
    if str_order not in ["0", "1time", "1loc", "2"]:
        print >> sys.stderr, "Error: Order (-o) must be 0, 1time, 1loc, 2"
        return False

    if str_trainingfile == "":
```

SEQ Model Source Code

```
print >> sys.stderr, "Error: Specify training file using -f option"
return False

# Check if input file exists
# Note that this is not considered a secure approach.
if os.path.exists( str_trainingfile ) == False:
    if verbose:
        print("Training file %s does not exist" % (str_trainingfile))
    return False

if str_testingfile == "":
    print >> sys.stderr, "Error: Specify file to test against (or predict) using -p
option."
    return False

# Check if test file exists
# Note that this is not considered a secure approach.
if os.path.exists( str_testingfile ) == False:
    if verbose:
        print("Testing file %s does not exist" % (str_testingfile))
    return False

# Validate other command line arguments, if necessary.

# Check for valid range in number of weeks
if ((num_preceding_weeks < 0) or (num_preceding_weeks > 9)):
    print >> sys.stderr, "Error: number of preceding weeks is out of range; defaulting
to 9."
    #set to default value
    num_preceding_weeks = 9

# Everything checks out
return True

#####
#
#     fn_zero_order_search
#
#     Given a quest location, search the
#     sequences without any context.
#
#     INPUTS: quest_location (string)
#     RETURNS: dictionary of counts
#     key =time or "Never"
#     value = count
#####
def fn_zero_order_search(quest_location):
    global list_sub_seq          # list of historical sequences
    global quest_location_not_found
    global numerSupport

    numerSupport = 0              # # of times context seen in a sequence
                                # this value is the numerator of the support

    dict_count = {}
    dict_count["Never"]=0 # create entry for this location not seen
    quest_location_not_found = True

    #Search the List of Sequences
    for line in list_sub_seq:
        ll_seq = line.split(',')
        #print "fn_zero: ll_seq = ", ll_seq
        # First, check to see if the quest_location is even in this
        # sequence at all.
        if ll_seq.count(quest_location) == 0:
            dict_count["Never"] += 1
            continue          # onto next sequence
        numerSupport += 1
        quest_location_not_found = False      # found it at least once
        # Add additional occurrences of quest_loc to the dictionary
        count_quest_locs = ll_seq.count(quest_location)
        index_loc = 0
```

SEQ Model Source Code

```

index_absolute = 0
for i in range(0, count_quest_locs):
    index_absolute += 1
    # find the relative index for the next occurrence of quest_location
    index_loc = ll_seq[index_absolute:].index(quest_location)
    index_absolute += index_loc
    key = ll_seq[ index_absolute - 1]
    if dict_count.has_key( key):
        dict_count[ key ] += 1 # increment counter
    else:
        dict_count[key] = 1    # create entry in dictionary
return dict_count

#####
#
#   fn_1st_order_time_search
#
#   Given a context
#   time and a quest
#   location, search the
#   sequences for the quest location that
#   occurs after the context.
#   First check to see if the quest_location
#   is in the sequence, then use the times
#   where the quest_location happens after
#   the context. (ex. when Bob
#   goes to the office after coffee)
#
#   INPUTS:
#       context (string)
#       quest_location (string)
#   RETURNS: dictionary of counts
#       key =time or "Never"
#       value = count
#####
def fn_1st_order_time_search(context, quest_location):
    global list_sub_seq      # list of historical sequences
    global quest_location_not_found
    global numerSupport

    numerSupport = 0          # # of times context seen in a sequence
                              # this value is the numerator of the support

    dict_count = {}
    dict_count["Never"]=0 # create entry for this location not seen
    quest_location_not_found = True

    #Search the List of Sequences
    for line in list_sub_seq:
        ll_seq = line.split(',')
        # First, check to see if the quest_location is even in this
        # sequence at all.
        if ll_seq.count(quest_location) == 0:
            dict_count["Never"] += 1
            continue          # onto next sequence

        quest_location_not_found = False      # found it at least once
        # Check if the context is in this sequence.
        success, search_index = fn_find_time_in_list( context, ll_seq)
        if success == False:
            dict_count["Never"] += 1
            continue          # onto next sequence
        numerSupport += 1
        if (ll_seq[search_index:].count(quest_location) > 0):
            quest_index = ll_seq[search_index:].index( quest_location)
            search_index += quest_index
            key = ll_seq[ search_index - 1]
            if dict_count.has_key( key):
                dict_count[ key ] += 1 # increment counter
            else:
                dict_count[key] = 1    # create entry in dictionary

```

SEQ Model Source Code

```

        else:
            # Quest not found after context
            dict_count["Never"] += 1
        return dict_count
#####
#
#     fn_1st_order_search
#
#     Given a context location OR a context
#     time and a quest
#     location, search the
#     sequences for the quest location that
#     occurs after the context.
#     First check to see if the quest_location
#     is in the sequence, then use the times
#     where the quest_location happens after
#     the context. (ex. when Bob
#     goes to the office after coffee)
#
#     INPUTS:
#         context (string)
#         quest_location (string)
#     RETURNS: dictionary of counts
#         key =time or "Never"
#         value = count
#####
def fn_1st_order_search(context, quest_location):
    global list_sub_seq          # list of historical sequences
    global quest_location_not_found
    global numerSupport

    numerSupport = 0              # # of times context seen in a sequence
                                  # this value is the numerator of the support

    dict_count = {}
    dict_count["Never"]=0 # create entry for this location not seen
    quest_location_not_found = True

    #Search the List of Sequences
    for line in list_sub_seq:
        ll_seq = line.split(',')
        # First, check to see if the quest_location is even in this
        # sequence at all.
        if ll_seq.count(quest_location) == 0:
            dict_count["Never"] += 1
            continue          # onto next sequence

        quest_location_not_found = False      # found it at least once
        # Check if the context is in this sequence.
        if ll_seq.count(context) == 0:
            dict_count["Never"] += 1
            continue          # onto next sequence

        numerSupport += 1
        # For each occurrence of the context,
        # see if the quest_location happens afterwards.
        search_index = 0
        while (search_index < len(ll_seq)-2):
            if ll_seq[search_index:].count(context) == 0:
                break      # stop if there are no more context
            context_index = ll_seq[search_index:].index( context)
            search_index += context_index

            if (ll_seq[search_index:].count(quest_location) > 0):
                quest_index = ll_seq[search_index:].index( quest_location)
                search_index += quest_index
                key = ll_seq[ search_index - 1]
                if dict_count.has_key( key):
                    dict_count[ key ] += 1 # increment counter
                else:
                    dict_count[key] = 1      # create entry in dictionary

```

SEQ Model Source Code

```

        search_index += 1      # to avoid loops
    else:
        break                  # with next sequence

    return dict_count
#####
#
#     fn_find_time_in_list
#
#     Given a time string (ex. "08:00"),
#     and a sequence, find the time closest
#     to that time (but equal to or earlier)
#     in the sequence. Return the index.
#     ASSUMPTIONS: that the time strings
#     are at the even-numbered locations
#     in the sequence.
#     INPUTS:
#         context_time (string)
#         ll_seq (list)
#     RETURNS:
#         success, index tuple
#         where 'success' is TRUE if
#         a time is found and FALSE if not
#         index = index into the sequence.
#####
def fn_find_time_in_list(context_time, ll_seq):
    success = False
    index = 0          # dummy value
    for (i, item) in enumerate(ll_seq):
        if (i % 2) == 0:    # if i is even (a time)
            if item <= context_time:
                success=True    # Found a candidate
                index = i
            else:
                break
    #print "find_time: context_time=%s, found %s at %d" % (context_time, ll_seq[index],
index)
    return(success, index)

#####
#
#     fn_2nd_order_search
#
#     Given a context time AND a context
#     time and a quest location, search the
#     sequences for the quest locatin that
#     occurs after the context.
#     First, verify that the quest_location
#     is in the sequence. If not, quit.
#     The remaining steps are these:
#     1. Find the time in the sequence closest
#     to but earlier than the context_time. (This
#     spot marks the left ('early') edge of the
#     sequence to search.
#     2.Find the first occurrence of the quest_
#     location after the context_time. If it
#     doesn't exist, quit. If it does exist,
#     this marks the right ('late') edge of the
#     search space.
#     3. See if the context_location is in that
#     limited search space. If not, quit. If
#     it is, use the time associated with the
#     quest_location.
#
#     ASSUMPTIONS: That the time strings are
#     formatted with leading zeros (ex. "08:00"
#     and not "8:00")
#
#     INPUTS:
#         context_time (string)
#         context_loc (string)

```

SEQ Model Source Code

```

#         quest_location (string)
#     RETURNS: dictionary of counts
#     key =time or "Never"
#     value = count
#####
def fn_2nd_order_search(context_time, context_location, quest_location):
    global list_sub_seq          # list of historical sequences
    global quest_location_not_found
    global numerSupport

    numerSupport = 0              # # of times context seen in a sequence
                                  # this value is the numerator of the support

    dict_count = {}
    dict_count["Never"]=0 # create entry for this location not seen
    quest_location_not_found = True

    #Search the List of Sequences
    for line in list_sub_seq:
        ll_seq = line.split(',')
        # First, check to see if the quest_location is even in this
        # sequence at all.
        if ll_seq.count(quest_location) == 0:
            #print "quest_location not in sequence"
            dict_count["Never"] += 1
            continue          # onto next sequence

        quest_location_not_found = False      # found it at least once
        # Step 1: Search the sequence to find the time closest to the
        # context time.
        # This context_time_index is also the new left index
        # of our search space
        success, left_index = fn_find_time_in_list(context_time, ll_seq)
        if success == False:
            #print "context time not found"
            dict_count["Never"] += 1
            continue          # move to next sequence

        # Step 2: Find the quest_location in the sequence,
        # AFTER the context_time.
        if ll_seq[left_index:].count(quest_location) == 0:
            # Not found
            #print "quest location not found after context_time"
            dict_count["Never"] += 1
            continue          # move to next sequence
        right_index = left_index + ll_seq[left_index:].index(quest_location)
        #print "Left is %d, right is %d" % (left_index, right_index)

        # Step 3: See if the context location is in our
        # sub-sequence.
        if ll_seq[left_index:right_index].count(context_location) == 0:
            # not found, quit searching this sequence
            #print "context location not found"
            dict_count["Never"] += 1
            continue

        numerSupport += 1
        # At this point, we have met all of the conditions for
        # our "second-order" search
        # The predicted time is the time right before
        # the quest location, which is at right_index.
        key = ll_seq[right_index-1]
        if dict_count.has_key( key ):
            dict_count[key] += 1
        else: # add it to dictionary
            dict_count[key] = 1

    return dict_count
#####
#
#     within_10or20_Min

```

SEQ Model Source Code

```
#
#   Given the dictionary and the time,
#   return true if the dictionary contains
#   a time within 10 minutes or 20 minutes
#   Also calculates and returns the smallest
#   time difference between the time and the predictions.
#   input: dict_cc is the dictionary of counts
#          str_time is the time in question
#   returns: ten = true if within 10 minutes
#            twenty = true if within 20 minutes
#            delta = number of minutes
#####
def within_10or20_min( dict_cc, str_time):
    ten = False
    twenty = False
    min_delta = 1500 # minutes between best prediction and correct time

    correct_dtime = DateTime.DateTimeFrom(str_time) # correct answer
    # calculate the delta for each prediction and keep the smallest delta
    for item in dict_cc:
        pred_dtime = DateTime.DateTimeFrom(item)
        diff = abs(int((pred_dtime - correct_dtime).minutes))
        if diff < min_delta:
            min_delta = diff
    if min_delta <= 10:
        ten = True
    if min_delta <= 20:
        twenty = True
    return (ten, twenty, min_delta)

#####
#
#   Main
#
#####
def main(argv):
    global list_sub_seq

    bool_outcome = getCommandLineArgs(argv)
    if bool_outcome == False: # some problem with the command line?
        sys.exit(2)

    #
    #   Train the model by building a list
    #   of the training sequences.
    #

    ifile = open(str_trainingfile,"r") # read file of training sequences
    l2 = [] # temp list to hold each line of file
    l3 = []
    for line in ifile:
        line = line.strip() # remove line feed
        l3 = line.split(';')
        # Remove white space, line feeds
        for (i,item) in enumerate(l3):
            l3[i] = item.strip()
        if (len(l3[0]) > 0): # skip blank lines
            l2 = l3[3].split(',') # get time,loc pairs
            # Remove white space, line feeds
            for (i,item) in enumerate(l2):
                l2[i] = item.strip()
            l3[3] = ','.join(l2)
            list_seq.append(l3)
    #print 'training list_seq: ', list_seq
    ifile.close()
    #
    #   Read the testing file and predict it
    #
    ifile = open(str_testingfile,"r") # read file of testing sequences
    # Reset all of the counters to 0 #####
```

SEQ Model Source Code

```

num_correct=0
num_tests = 0
num_refuse_to_predict = 0      # refuse to predict if quest location not found in any of
the input sequences.
num_within_10min = 0          # number of wrong predictions within 10 minutes
num_within_20min = 0          # # of wrong predictions w/in 20 minutes
aveSupportNumer = 0            # numerator = sum(numberSupport)
aveSupportDenom = 0            # denominator = sum(denomSupport)
aveConfidenceNumer = 0         # numerator = sum(count of correct prediction)
numConfidence = 0              # only here to prevent python error if all tests fail.
test_list = [] # complete test file
l2 = []
l4 = [] # temp list to hold testing file
# Start XML output
#print '<? xml version="1.0" standalone="yes" ?>'
#print "<SeqModel>"

for line in ifile:
    test_seq = [] # just the time, loc tuples in the test file
    line = line.strip() # remove trailing line feed
    if line == '':
        continue
    l4 = line.split(';') # parse by comma-delimiters
    # Remove whitespace, linefeeds
    for (i,item) in enumerate(l4):
        l4[i] = item.strip()
    if (len(l4[0]) > 0): #skip blank lines
        test_list.append(l4)
        #test_seq.append(l4[3]) # just the time,loc tuples
    # Build a test_seq that only contains the time,loc pairs
    l2 = l4[3].split(',') # parse by comma-delimiters
    # Remove white space, line feeds
    for (i,item) in enumerate(l2):
        l2[i] = item.strip()
        if (len(l2[i]) > 0): # skip blank lines
            test_seq.append(l2[i])

    #print 'test_seq: ', test_seq
    #
    # Get the week number from the test file.
    # See if it's in range.
    #
    test_week = int(test_list[0][1]) # get the week # of the test week
    #print 'test_week: ', test_week
    if test_week <= num_preceding_weeks:
        print >> sys.stderr, "Error: Week number in testing data (%d) is too
large. Should be less than %d." % (test_week, num_preceding_weeks)
        sys.exit(3)

    #
    # The test file contains date, day-of-week, week#, tuples
    # Use the num_preceding_weeks value to
    # figure out which lines to test.
    #
    # Go through the master list and pull all the related
    # sequences and put them into list_sub_seq. The following code
    # simply checks against the sequences listed in list_sub_seq.
    # For counting metrics, also build a list of only the locations
    # in the list_sub_seq. This list is used to count the number
    # of time,loc pairs in the 'training' data and to count the
    # number of unique locations.

    list_sub_seq=[]
    location_list=[] # list of only locations inlist_sub_seq
    time_list=[] # count unique times

    #####
    for i in range(0, len(list_seq)):
        l7=[]
        date, wk, dow, sequence = list_seq[i]
        week = int(wk)

```


SEQ Model Source Code

```

# print i, date, dow, week, sequence
# Keep this entry if it's within the week range.
# print 'week, range: ', week, range(test_week-num_preceding_weeks,
test_week)

if week in range(test_week-num_preceding_weeks, test_week):
    # print 'adding this week'
    list_sub_seq.append( sequence)
    l7 = sequence.split(',')
    for (ii, iitem) in enumerate(l7):
        if ii%2 == 1:
            # print 'add loc ', iitem
            location_list.append(iitem)
        else:
            # print 'add time ', iitem
            time_list.append(iitem)
# if num_preceding_weeks == 0, it's a special case
# of testing the model against its training data.
elif num_preceding_weeks == 0:
    if week == test_week:
        list_sub_seq.append(sequence)

# print 'in main: list_sub_seq:', list_sub_seq
# denominator of support is the length of the list_sub_seq
# print "denomSupport = %d" % (len(list_sub_seq))
denomSupport=len(list_sub_seq)

# Write out the number of predictions returned to another file
ofile = open("num_pred_seq.csv","a")

# Go through the line and try to predict
#####
#         ZERO ORDER
#####
if (str_order == "0"):
    for i in range(0,len(test_seq)-1,2):
        delta = 0
        str_time = test_seq[i]
        str_loc = test_seq[i+1]
        if verbose:
            print "0th Order: Quest_location is %s, answer is %s" %
(str_loc, str_time),

        dict_0counts = fn_zero_order_search(str_loc)
        # print dict_0counts
        num_tests += 1
        if quest_location_not_found:
            num_refuse_to_predict += 1
        # if quest_location_not_found or \
        if dict_0counts.has_key(str_time):
            num_correct += 1
            # We have a correct prediction; add the counts to the
totals
            numerConfidence = dict_0counts[str_time]
            aveSupportNumer += numerSupport
            aveSupportDenom += denomSupport
            aveConfidenceNumer += numerConfidence
            if verbose:
                print "YES!!!!"
                print "numerSupport=%d, denomSupport=%d,count=%d" %
(numerSupport, denomSupport, dict_0counts[str_time])
                print dict_0counts
                # print "Support=%f, Confidence=%f" %
((float)numerSupport/(float)denomSupport, dict_0counts[str_time]/(float)numerSupport)
            else:
                ten, twenty, delta = within_10or20_min( dict_0counts,
str_time)

                if ten:
                    num_within_10min += 1
                if twenty:
                    num_within_20min += 1
                if verbose:
                    print "NO."

```

SEQ Model Source Code

```

                                print "numerSupport=%d, denomSupport=%d" %
(numerSupport, denomSupport)
                                str_out = "%s, %s, %d, %d\n" % (str_testingfile, str_order,
len(dict_0counts), delta)
                                ofile.write(str_out)
                                #####
                                # FIRST ORDER - Use Location
                                #####
                                elif (str_order == "lloc"):
                                    for i in range(2,len(test_seq)-1,2):
                                        delta = 0
                                        str_context_loc = test_seq[i-1]
                                        str_time = test_seq[i]
                                        str_loc = test_seq[i+1]
                                        if verbose:
                                            print "1st Order: Context = %s, Quest_location is %s,
answer is %s" % (str_context_loc, str_loc, str_time)
                                        dict_lloc_counts = fn_1st_order_search(str_context_loc, str_loc)
                                        #print dict_lloc_counts
                                        num_tests += 1
                                        if quest_location_not_found:
                                            num_refuse_to_predict += 1
                                        # if quest_location_not_found or \
                                        if dict_lloc_counts.has_key(str_time):
                                            num_correct += 1
                                            numerConfidence = dict_lloc_counts[str_time]
                                            aveSupportNumer += numerSupport
                                            aveSupportDenom += denomSupport
                                            aveConfidenceNumer += numerConfidence
                                            if verbose:
                                                print "YES!!!!!"
                                        else:
                                            ten, twenty, delta = within_10or20_min( dict_lloc_counts,
str_time)
                                            if ten:
                                                num_within_10min += 1
                                            if twenty:
                                                num_within_20min += 1
                                            if verbose:
                                                print "NO."
                                            str_out = "%s, %s, %d, %d\n" % (str_testingfile, str_order,
len(dict_lloc_counts), delta)
                                            ofile.write(str_out)
                                            #####
                                            # FIRST ORDER - Use Time
                                            #####
                                            elif (str_order == "ltime"):
                                                for i in range(2,len(test_seq)-1,2):
                                                    delta = 0
                                                    str_context_time = test_seq[i-2]
                                                    str_time = test_seq[i]
                                                    str_loc = test_seq[i+1]
                                                    if verbose:
                                                        print "1st Order: Context = %s, Quest_location is %s,
answer is %s" % (str_context_time, str_loc, str_time)
                                                        dict_ltime_counts =
fn_1st_order_time_search(str_context_time, str_loc)
                                                        #print dict_ltime_counts
                                                        num_tests += 1
                                                        if quest_location_not_found:
                                                            num_refuse_to_predict += 1
                                                        #if quest_location_not_found or \
                                                        if dict_ltime_counts.has_key(str_time):
                                                            num_correct += 1
                                                            numerConfidence = dict_ltime_counts[str_time]
                                                            aveSupportNumer += numerSupport
                                                            aveSupportDenom += denomSupport
                                                            aveConfidenceNumer += numerConfidence
                                                            if verbose:
                                                                print "YES!!!!!"
                                                        else:

```

SEQ Model Source Code

```

ten, twenty, delta = within_10or20_min( dict_1time_counts,
str_time)
    if ten:
        num_within_10min += 1
    if twenty:
        num_within_20min += 1
    if verbose:
        print "NO."

    str_out = "%s, %s, %d, %d\n" % (str_testingfile, str_order,
len(dict_1time_counts), delta)
    ofile.write(str_out)

#####
# SECOND ORDER - Use previous time and location
#####
elif (str_order == "2"):
    for i in range(2, len(test_seq)-1, 2):
        delta = 0
        str_context_time = test_seq[i-2]
        str_context_loc = test_seq[i-1]
        str_time = test_seq[i]
        str_loc = test_seq[i+1]
        if verbose:
            print "2nd Order: Context = %s,%s, Quest_location is %s,
answer is %s" % (str_context_time, str_context_loc, str_loc, str_time)
        dict_2_counts =
fn_2nd_order_search(str_context_time, str_context_loc, str_loc)
        #print dict_2_counts
        num_tests += 1
        if quest_location_not_found:
            num_refuse_to_predict += 1
        #if quest_location_not_found or \
        if dict_2_counts.has_key(str_time):
            num_correct += 1
            numerConfidence = dict_2_counts[str_time]
            aveSupportNumer += numerSupport
            aveSupportDenom += denomSupport
            aveConfidenceNumer += numerConfidence
            if verbose:
                print "YES!!!!!"
        else:
            ten, twenty, delta = within_10or20_min( dict_2_counts,
str_time)
            if ten:
                num_within_10min += 1
            if twenty:
                num_within_20min += 1
            if verbose:
                print "NO."

            str_out = "%s, %s, %d, %d\n" % (str_testingfile, str_order,
len(dict_2_counts), delta)
            ofile.write(str_out)
            #####
            if num_tests > 0:
                num_training_pairs = len(location_list)
                num_unique_locations = len(set(location_list))
                num_unique_times = len(set(time_list))

                if verbose:
                    print "%s, %d training_pairs, %d unique locations, %d weeks, %d/%d,
%%Correct = %.1f, refuse_to_predict=%d, %d w/in 10 minutes, %d w/in 20 minutes" %
(str_testingfile, num_training_pairs, num_unique_locations, num_preceding_weeks, num_correct,
num_tests, 100*num_correct/num_tests, num_refuse_to_predict, num_within_10min, num_within_20min)

                else:
                    #print "%s, %d, %d, %d, %d, %d, %.1f, %d, %d, %d" % (str_testingfile,
num_training_pairs, num_unique_locations, num_preceding_weeks, num_correct, num_tests,
100*num_correct/num_tests, num_refuse_to_predict, num_within_10min, num_within_20min)
                    # Print results in XML
                    print "<Run>"

```

SEQ Model Source Code

```
print "    <TrainingFile>%s</TrainingFile>" % (str_trainingfile)
print "    <TestFile>%s</TestFile>" % (str_testingfile)
print "    <numSequences>%d</numSequences>" % (denomSupport)
print "    <NumTrainingPairs>%d</NumTrainingPairs>" % (num_training_pairs)
print "    <NumUniqueLocations>%s</NumUniqueLocations>" %
(num_unique_locations)
print "    <NumUniqueTimes>%s</NumUniqueTimes>" % (num_unique_times)

# print "    <NumPrecedingWeeks>%d</NumPrecedingWeeks>" %
(num_preceding_weeks)
print "    <Order>%s</Order>" % (str_order)
print "    <NumCorrect>%d</NumCorrect>" % (num_correct)
print "    <NumTests>%d</NumTests>" % (num_tests)
print "    <NumRefuseToPredict>%d</NumRefuseToPredict>" %
(num_refuse_to_predict)
print "    <NumWithin10Minutes>%d</NumWithin10Minutes>" %
(num_within_10min)
print "    <NumWithin20Minutes>%d</NumWithin20Minutes>" %
(num_within_20min)
print "    <aveSupportNumer>%d</aveSupportNumer>" % (aveSupportNumer)
print "    <aveSupportDenom>%d</aveSupportDenom>" % (aveSupportDenom)
print "    <aveConfidenceNumer>%d</aveConfidenceNumer>" %
(aveConfidenceNumer)
print "</Run>"
# print "</SeqModel>" # end of XML
ifile.close()

if __name__ == "__main__":
    main(sys.argv[1:]) # remove first argument and run code.

# End of File
```

REFERENCES

1. Weiser, M.: The Computer for the Twenty-First Century. *Scientific American* 265, 94-104 (1991)
2. Satyanarayanan, M.: Pervasive computing: vision and challenges. *IEEE Personal Communications* 8, 10-17 (2001)
3. Begole, B.: It's time to reap the context-aware harvest, <http://blogs.parc.com/blog/2010/09/its-time-to-reap-the-context-aware-harvest/>, (2010)
4. Bellotti, V., Begole, B., Chi, E.H., Ducheneaut, N., Fang, J., Isaacs, E., King, T., Newman, M.W., Partridge, K., Price, B., Rasmussen, P., Roberts, M., Schiano, D.J., Walendowski, A.: Activity-based serendipitous recommendations with the Magitti mobile leisure guide. In: *Proceedings of CHI. ACM, Florence, Italy* (2008)
5. Holbach, W.: Personal Communication, (2010)
6. Patterson, D.J., Liao, L., Gajos, K., Collier, M., Livic, N., Olson, K., Wang, S., Fox, D., Kautz, H.: Opportunity Knocks: A System to Provide Cognitive Assistance with Transportation Services. In: *Proceedings of UbiComp*, pp. 433-450. Springer-Verlag GmbH (2004)
7. Estrin, D., Chandy, K.M., Young, R.M., Smarr, L., Odlyzko, A., Clark, D., Reding, V., Ishida, T., Sharma, S., Cerf, V.G., Izle, U., Barroso, L.A., Mulligan, G., Hooke, A., Elliott, C.: Internet Predictions. *IEEE Internet Computing* 14, 12-42 (2010)
8. Chang, Y.-J., Liu, H.-H., Wang, T.-Y.: Mobile social networks as quality of life technology for people with severe mental illness. *IEEE Wireless Communications* 16, 34-40 (2009)
9. Axup, J., Viller, S., MacColl, I., Cooper, R.: Lo-Fi Matchmaking: A Study of Social Pairing for Backpackers. In: *Proceedings of UbiComp*, pp. 351-368. SpringerLink, Orange County, CA, USA (2006)
10. Clifford, S.: Linking Customer Loyalty With Social Networking. *The New York Times*. New York Times, New York (2010)
11. Ashbrook, D., Starner, T.: Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing* 7, 275-286 (2003)
12. Schilit, B., Hong, J., Gruteser, M.: Wireless location privacy protection. *Computer* 36, 135-137 (2003)
13. Kraut, R.E., Fish, R., Root, R., Chalfonte, B.: Informal communication in organizations: Form, function, and technology. In: Oskamp, S., Scacapan, S. (eds.): *Human reactions to technology: Claremont symposium on applied social psychology*. Sage Publications, Beverly Hills, CA (1990)
14. Tullio, J.: Exploring the Design and Use of Forecasting Groupware Applications with an Augmented Shared Calendar. *Computer Science, Doctor of Philosophy*. Georgia Institute of Technology (2005) 202

REFERENCES

15. Begole, J.B., Tang, J.C., Hill, R.: Rhythm modeling, visualizations and applications. In: Proceedings of UIST, pp. 11 - 20 ACM Press, Vancouver, Canada (2003)
16. Partridge, K., Begole, J.: Activity-based Advertising: techniques and challenges. In: Proceedings of Workshop of Pervasive Advertising. PARC, Nara, Japan (2009)
17. Katsaros, D., Manolopoulos, Y.: Prediction in wireless networks by Markov chains. *Wireless Communications*, IEEE 16, 56-64 (2009)
18. Burbey, I., Martin, T.L.: Predicting future locations using prediction-by-partial-match. In: Proceedings of MELT. ACM, San Francisco, California, USA (2008)
19. Ashbrook, D., Starner, T.: Learning significant locations and predicting user movement with GPS. In: Proceedings of International Symposium on Wearable Computers, pp. 101 (2002)
20. Fisk, D.: Beer and Nappies -- A Data Mining Urban Legend. (2006)
21. Antunes, C.M., Oliveira, A.L.: Temporal Data Mining: An Overview. In: Proceedings of KDD 2001 Workshop of Temporal Data Mining, held in conjunction with the 7th ACM SIGDDK International Conference on Knowledge Discovery and Data Mining (KDD-2001), San Francisco, CA, USA (2001)
22. Jakkula, V.R., Cook, D.J.: Using Temporal Relations in Smart Environment Data for Activity Prediction. In: Proceedings of 24th International Conference on Machine Learning, Corvallis, OR (2007)
23. Elnekave, S., Last, M., Maimon, O.: Predicting future locations using clusters' centroids. In: Proceedings of Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, pp. 1-4. ACM, Seattle, Washington (2007)
24. Hsu, W., Lee, M.L., Wang, J.: Temporal and Spatio-temporal Data Mining. IGI Publishing, (2008)
25. Roddick, J.F., Spiliopoulou, M.: A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering* 14, 750-767 (2002)
26. Moerchen, F.: Time series knowledge mining. Doctorate. Philipps-University, Marburg, Germany (2006) 178
27. Sun, R., Giles, C.L.: Sequence learning: from recognition and prediction to sequential decision making. *IEEE Intelligent Systems* 16, 67-70 (2001)
28. Lymberopoulos, D., Bamis, A., Savvides, A.: Extracting spatiotemporal human activity patterns in assisted living using a home sensor network. In: Proceedings of Proceedings of the 1st international conference on Pervasive Technologies Related to Assistive Environments, pp. 1-8. ACM, Athens, Greece (2008)
29. Petzold, J., Bagci, F., Trumler, W., Ungerer, T.: Next Location Prediction Within a Smart Office Building. In: Proceedings of Pervasive 2005, Munich, Germany (2005)
30. Das, S.K., Cook, D.J., Battacharya, A., Heierman, E.O., III, Tze-Yun, L.: The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications* 9, 77-84 (2002)
31. Skyhook Wireless, www.skyhookwireless.com
32. Otsason, V., Varshavsky, A., LaMarca, A., de Lara, E.: Accurate GSM Indoor Localization. In: Beigl, M., Intille, S., Rekimoto, J., Tokuda, H. (eds.): *UbiComp 2005: Ubiquitous Computing*, Vol. 3660. Springer Berlin / Heidelberg (2005) 903-903

REFERENCES

33. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G., Schilit, B.: Place Lab: Device Positioning Using Radio Beacons in the Wild.: In: Proceedings of Pervasive 2005, pp. 116-133. Springer Berlin / Heidelberg, Munich, Germany (2005)
34. Kim, M., Fielding, J.J., Kotz, D.: Risks of using AP locations discovered through war driving. In: Proceedings of Fourth International Conference on Pervasive Computing (Pervasive), pp. 67-82. Springer-Verlag, Dublin, Ireland (2006)
35. Kotz, D., Henderson, T.: CRAWDAD: A Community Resource for Archiving Wireless Data at Dartmouth. *IEEE Pervasive Computing* 4, 12-14 (2005)
36. UCSD Wireless Topology Discovery Trace, <http://sysnet.ucsd.edu/wtd/>
37. Hightower, J., Borriello, G.: Location systems for ubiquitous computing. *Computer* 34, 57-66 (2001)
38. Hightower, J., Consolvo, S., LaMarca, A., Smith, I., Hughes, J.: Learning and Recognizing the Places We Go. In: Proceedings of Ubicomp 2005, pp. 159-176. Springer-Verlag (2005)
39. Zhou, C., Ludford, P., Frankowski, D., Terveen, L.: An experiment in discovering personally meaningful places from location data. In: Proceedings of CHI '05 extended abstracts on Human factors in computing systems. ACM Press, Portland, OR, USA (2005)
40. Priyantha, N., Chakraborty, A., Balakrishnan, H.: The Cricket Location-Support System. In: Proceedings of MOBICOM. ACM, Boston, MA (2000)
41. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The Active Badge Location System. *ACM Transactions on Information Systems* 10, 91-102 (1992)
42. CRAWDAD data set ibm/watson (v. 2003-02-29), Downloaded from <http://crawdad.cs.dartmouth.edu/ibm/watson>
43. Cook, D.J., Youngblood, M., Heierman, E.O., III, Gopalratnam, K., Rao, S., Litvin, A., Khawaja, F.: MavHome: an agent-based smart home. In: Proceedings of First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), pp. 521-524 (2003)
44. Bhattacharya, A., Das, S.K.: LeZi-update: an information-theoretic approach to track mobile users in PCS networks. In: Proceedings of 5th annual ACM/IEEE international conference on Mobile computing and networking. ACM Press, Seattle, Washington, United States (1999)
45. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* 24, 530-536 (1978)
46. Roy, A., Das, S.K., Basu, K.: A Predictive Framework for Location-Aware Resource Management in Smart Homes. *IEEE Transactions on Mobile Computing* 6, 1284-1283 (2007)
47. Hariharan, R., Toyama, K.: Project Lachesis: Parsing and Modeling Location Histories. In: Proceedings of Second International Conference on Geographic Information Science, pp. 106-124. Springer, Adelphi, MD (2004)
48. Song, L., Kotz, D., Jain, R., He, X.: Evaluating location predictors with extensive Wi-Fi mobility data. In: Proceedings of INFOCOMM, pp. 1414-1424. IEEE, Hong Kong (2004)

REFERENCES

49. Petzold, J.: Zustandsprädiktoren zur Kontextvorhersage in ubiquitären Systemen. der Fakultät für Angewandte Informatik, Doctorate. Universität Augsburg, Augsburg (2005) 168
50. Petzold, J., Pietzowski, A., Bagci, F., Trumler, W., Ungerer, T.: Prediction of Indoor Movements Using Bayesian Networks, (2005)
51. Petzold, J., Bagci, F., Trumler, W., Ungerer, T., Vintan, L.: Global State Context Prediction Techniques Applied to a Smart Office Building. In: Proceedings of Communications Networks and Distributed Systems Modeling and Simulation Conference (CNDS), San Diego, CA (2004)
52. Petzold, J., Bagci, F., Trumler, W., Ungerer, T.: Confidence Estimation of the State Predictor Method. Lecture Notes in Computer Science, Vol. 3295/2004. Springer, Berlin / Heidelberg (2004) 375-386
53. Eagle, N., Pentland, A.S.: Eigenbehaviors: Identifying Structure in Routine. MIT Media Lab: VisMod Group (2006)
54. Eagle, N., Clauset, A., Quinn, J.A.: Location Segmentation, Inference and Prediction for Anticipatory Computing. In: Proceedings of AAAI Spring Symposium on Technosocial Predictive Analytics, Stanford, CA (2009)
55. Ziebart, B.D., Maas, A.L., Dey, A.K., Bagnell, J.A.: Navigate like a cabbie: probabilistic reasoning from observed context-aware behavior. In: Proceedings of UbiComp, pp. 322-331. ACM, Seoul, Korea (2008)
56. Bauer, M., Deru, M.: Motion-Based Adaptation of Information Services for Mobile Users. In: Ardissono, L., Brna, P., Mitrovic, A. (eds.): User Modeling 2005, Vol. 3538. Springer Berlin / Heidelberg (2005) 271-276
57. Vu, L., Do, Q., Narhstedt, K.: Jyotish: A Novel Framework for Constructing Predictive Model of People Movement from Joint Wifi/Bluetooth Trace. In: Proceedings of IEEE International Conference on Pervasive Computing and Communications (PERCOM), pp. 54-62. IEEE, Seattle, WA (2011)
58. Vu, L., Do, Q., Nahrstedt, K.: Exploiting Joint Wifi/Bluetooth Trace to Predict People Movement, <https://www.ideals.illinois.edu/handle/2142/16944>, (2011)
59. Calabrese, F., Di Lorenzo, G., Ratti, C.: Human mobility prediction based on individual and collective geographical preferences. In: Proceedings of 13th International IEEE Conference on Intelligent Transportation Systems (ITSC) pp. 312-317. IEEE, Madeira Island, Portugal (2010)
60. Monreale, A., Pinelli, F., Trasarti, R., Giannotti, F.: WhereNext: a location predictor on trajectory pattern mining. In: Proceedings of 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 637-646. ACM, Paris, France (2009)
61. Scellato, S., Musolesi, M., Mascolo, C., Latora, V., Campbell, A.T.: NextPlace: A Spatio-Temporal Prediction Framework for Pervasive Systems. In: Proceedings of Ninth International Conference on Pervasive Computing (Pervasive 2011). Springer Verlag, San Francisco, CA (2011)
62. Begleiter, R., El-Yaniv, R., Yona, G.: On Prediction Using Variable Order Markov Models. Journal of Artificial Intelligence Research 22, 385-421 (2004)
63. Cleary, J., Witten, I.: Data Compression Using Adaptive Coding and Partial String Matching. IEEE Transactions on Communications 32, 396-402 (1984)

REFERENCES

64. MacKay, D.J.C.: Information Theory, Inference & Learning Algorithms. Cambridge University Press, (2002)
65. Song, C., Qu, Z., Blumm, N., Barabás, A.-L.: Limits of Predictability in Human Mobility. *Science* 327, 1018 - 1021 (2010)
66. Arithmetic Coding + Statistical Modeling = Data Compression, <http://www.dogma.net/markn/articles/arith/part2.htm>
67. Mynatt, E., Tullio, J.: Inferring calendar event attendance. In: Proceedings of Proceedings of the 6th international conference on Intelligent user interfaces, pp. 121-128. ACM, Santa Fe, New Mexico, United States (2001)
68. Foundation, P.S.: Python Programming Language -- Official Website, <http://www.python.org/>
69. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer, New York, NY, (2001)
70. ABResearch: 82 Million Location-based Mobile Social Networking Subscriptions by 2013, <http://www.abiresearch.com/press/3064-82+Million+Location-based+Mobile+Social+Networking+Subscriptions+by+2013>, (2011)
71. Foursquare: It all started with a check-in: the vision for #4sq3 and beyond, <http://blog.foursquare.com/2011/03/08/foursquare-3/>, (2011)
72. Tang, K.P., Lin, J., Hong, J.I., Siewiorek, D.P., Sadeh, N.: Rethinking location sharing: exploring the implications of social-driven vs. purpose-driven location sharing. In: Proceedings of Proceedings of the 12th ACM international conference on Ubiquitous computing, pp. 85-94. ACM, Copenhagen, Denmark (2010)
73. Lin, J., Xiang, G., Hong, J.I., Sadeh, N.: Modeling people's place naming preferences in location sharing. In: Proceedings of Proceedings of the 12th ACM international conference on Ubiquitous computing, pp. 75-84. ACM, Copenhagen, Denmark (2010)
74. Barkhuus, L., Dey, A.: Is Context-Aware Computing Taking Control away from the User? Three Levels of Interactivity Examined, (2003)