OPERATIONS RESEARCH CENTER

Working Paper

Sequential Event Prediction

by

Benjamin Letham
Cynthia Rudin
David Madigan

OR 387-11 **November 2011**

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Noname manuscript No.
(will be inserted by the editor)

Sequential event prediction

Benjamin Letham · Cynthia Rudin · David Madigan

Received: date / Accepted: date

Abstract In *sequential event prediction*, we are given a "sequence database" of past event sequences to learn from, and we aim to predict the next event within a current event sequence. We focus on applications where the *set* of the past events has predictive power and not the specific *order* of those past events. Such applications arise in recommender systems, equipment maintenance, medical informatics, and in other domains. Our formalization of sequential event prediction draws on ideas from supervised ranking. We show how specific choices within this approach lead to different sequential event prediction problems and algorithms. In recommender system applications, the observed sequence of events depends on user choices, which may be influenced by the recommendations, which are themselves tailored to the user's choices. This chicken-and-egg problem leads to sequential event prediction algorithms involving a non-convex optimization problem. We apply our approach to an online grocery store recommender system, email recipient recommendation, and a novel application in the health event prediction domain.

Keywords Sequential Event Prediction · Supervised Ranking · Recommender Systems

1 Introduction

Sequential event prediction refers to a wide class of problems in which a set of initially hidden events are sequentially revealed. The goal is to use the set of revealed

B. Letham
Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, USA
E-mail: bletham@mit.edu

C. Rudin
MIT Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA
E-mail: rudin@mit.edu

D. Madigan
Department of Statistics, Columbia University, New York, NY, USA
E-mail: madigan@stat.columbia.edu

events, but not necessarily their order, to predict the remaining (hidden) events in the sequence. We have access to a "sequence database" of past event sequences that we can use to design the predictions. Predictions for the next event are updated each time a new event is revealed. There are many examples of sequential prediction problems. Medical conditions occur over a timeline, and the conditions that the patient has experienced in the past can be used to predict conditions that will come (McCormick et al, 2011). Music recommender systems, *e.g.* Pandora, use a set of songs for which the user has revealed his or her preference to construct a suitable playlist. The playlist is modified as new preferences are revealed. Online grocery stores such as Fresh Direct (in NYC) use the customer's current shopping cart to recommend other items. The recommendations are updated as items are added to the basket. Motivated by this application, "sequential event prediction" was formalized by Rudin et al (2011, 2012), who created a theoretical foundation along with some simple algorithms based on association rules. In this work, we present optimization-based algorithms for sequential event prediction. These algorithms are based on the principle of empirical risk minimization. We apply our algorithms to data from two applications: an online grocery store recommender system, and medical condition prediction.

Recommender systems are a particularly interesting example of sequential event prediction because the predictions are expected to influence the sequence (*e.g.*, Senecal and Nantel, 2004), and any realistic algorithm should take this into account. For instance, there has recently been work showing that measurements of user behavior can be used to improve search engine rankings (Agichtein et al, 2006a,b). For an online grocery store recommender system, items are added to the basket one at a time. The customer may not have an explicit preference for the order in which items are added, rather he or she may add items in whichever order is most convenient. In particular, the customer may add items in the order provided by the recommender system, which means the predictions actually alter the sequence in which events appear. Our formulation allows for models of user behavior to be incorporated while we learn the recommender system. In the case where users add items top-down according to the recommendations, the result is a non-standard optimization problem. Specifically, there are smooth, convex regions where the order in which items are added to the basket remains the same, punctuated with discontinuities. Even though the optimization problem is discontinuous, we exploit the structure in the objective to show how a good solution can be found. Namely, we use convex programming within smooth regions and a simulated annealing strategy to traverse discontinuities.

The same formulation used for the online grocery store recommender system can be directly applied to email recipient recommendation. Given a partial list of recipients on an email, we wish to predict the remaining recipients. An email recipient recommendation algorithm can be a very useful tool; an algorithm for this purpose was recently implemented on a very large scale by Google and is integrated into the Gmail system used by millions of people (Roth et al, 2010).

Medical condition prediction is a new yet active area of research in data mining (Davis et al, 2010; McCormick et al, 2011). Accurate predictions of subsequent patient conditions will allow for better preventative medicine, increased quality of life, and reduced healthcare costs. Rather than a sequence of single items, the data comprise a sequence of sets of conditions. Our formulation can be made to handle

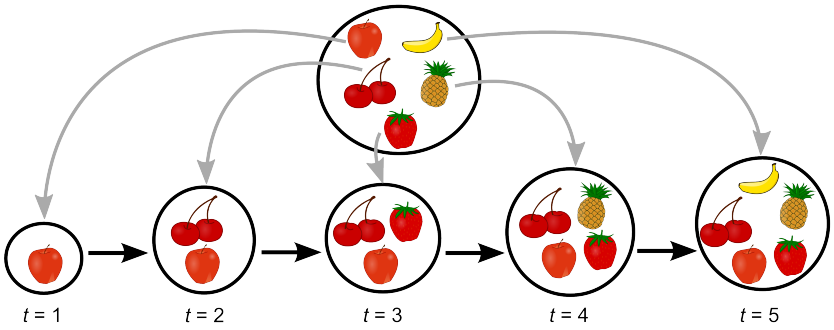

Fig. 1 An illustration of the online grocery store recommender system, in which items from an unordered shopping list are sequentially added to the user's basket. At each time step, the set of items in the basket is used to predict future items that will be added.

sequences of sets, and we apply it to a medical dataset consisting of individual patient histories.

The sequential event prediction problems we consider here are different from time-series prediction problems, that one might handle with a Markov chain. The recommender system problem has no intrinsic order in which groceries should be added to the basket, and only the *set* of items in the basket are useful for predicting the next one rather than the order in which those items were placed into the basket. Figure 1 gives an illustration of this point. For instance, at time $t = 2$, apples and cherries are in the basket and are together used to predict what will be added next. The fact that apples were added before cherries is not necessarily useful. In the medical condition prediction problem, collections of conditions occur at different time steps, and we use all past collections of conditions to predict the next collection. Figure 2 shows a sequence of these collections of conditions as they occur over time. For instance, at time $t = 1$, we use the entire collection of conditions {Hypertension, Sore throat, Gastric Ulcer} to make a prediction about the next collection. At time $t = 2$, we use the two collections {Hypertension, Sore Throat, Gastric Ulcer} and {Hypertension, Influenza} to make a prediction about the following time step. The collections of conditions occur sequentially in a certain order, however each collection is itself an unordered set of conditions. For example, it might not be sensible at $t = 2$ to say that Hypertension preceded Influenza. On the surface, the online grocery store recommender system and the medical condition prediction problem seem quite different, but the methodology we develop for both problems is quite similar, and could be adapted to a wide range of other sequential event prediction problems.

We treat each step of sequential event prediction as a supervised ranking problem. Given a set of revealed events from the current sequence, our algorithms rank all other possible events according to their likelihood of being a subsequent event in the sequence. The accuracy of our prediction is determined by how far down the list we need to look in order to find the next item(s) to be added.

Section 2.1 gives the formulation for the online grocery store recommender system. Section 2.2 develops the formulation for medical condition prediction. Section 2.4 contains optimization strategies and algorithms. Section 3 contains experiments

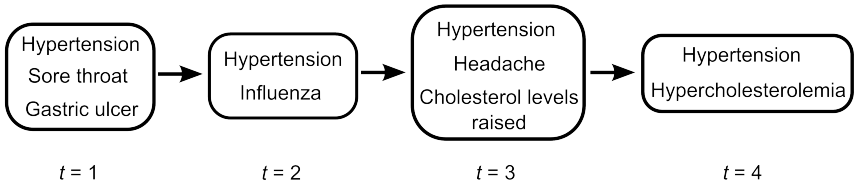

Fig. 2 An illustration of the medical condition prediction problem, in which collections of medical conditions occur at various time steps. At each time step, we use past collections of conditions to predict conditions that will subsequently be presented.

for the three applications. Our approach synthesizes ideas from supervised ranking in machine learning, convex optimization, and customer behavior modeling to produce flexible and powerful methods that can be used broadly for sequential event prediction problems.

2 Supervised Ranking and Sequential Events

We start with the formulation for the online grocery store recommender system, and then develop the formulation for medical condition prediction. We then show how these formulations fit into a unified framework.

2.1 Online grocery store recommender system

The online store has an inventory of N items, $\mathcal{Z}$ being the set of these items. Customer i comes to the store with a shopping list of items to be purchased $X_i \subseteq \mathcal{Z}$. The length of a particular shopping list, $|X_i|$, can vary from 1 to N and is denoted T_i. A session begins with an empty basket and items from the list are sequentially added to the basket at time steps $t = 1, \ldots, T_i$. The item added to basket i at time t is denoted $z_{i,t}$. The entire basket at time t is denoted $x_{i,t}$ and is equal to $\cup_{j=0}^{t} z_{i,j}$. The items in the basket $x_{i,t}$ are also called "revealed events." The items that are on the shopping list that are not yet in the basket, $X_i \setminus x_{i,t}$, are called "hidden events." The basket is initially empty, $x_{i,0} = z_{i,0} = \varnothing$, and the final basket x_{i,T_i} is equal to the shopping list X_i. Training is done using a collection of m shopping lists, denoted X_1^m. For the purpose of this paper, the recommender system is designed to be a tool to assist the customer, *i.e.*, there is no motive to recommend higher priced items, promote sale items, etc., although these could be incorporated in an extended version of our formulation.

The core of our supervised ranking method for sequential event prediction is a ranking model of the relationship between items in the basket (revealed events) and items that have not yet been added to the basket (hidden events). The ranking model is a scoring function $f(x_{i,t}, a)$ that, given the current basket $x_{i,t}$, scores each item $a \in \mathcal{Z} \setminus x_{i,t}$ according to the predicted likelihood that it is on the shopping list. Ideally we would like $f(x_{i,t}, a)$ to be related to $\mathbb{P}(a|x_{i,t})$, the conditional probability of adding a given that the items in $x_{i,t}$ have been added. The predictions will be made by ordering the items in descending score, so we need only that

$f(x_{i,t}, a)$ is monotonically related to $\mathbb{P}(a|x_{i,t})$ in order for the predictions to be accurate. The list of items ranked according to the scoring function will be called the "recommendation list." Note that the recommendation list is ordered, whereas the shopping list is unordered.

Our scoring function relies on a set of real-valued variables $\{\lambda_{a,b}\}_{a,b}$ to model the influence that itemset a has on the likelihood that item b will subsequently be added, for each itemset-item pair that we are willing to consider. Throughout the paper we let $\mathcal{A}$ be the allowed set of itemsets, and we introduce a variable $\lambda_{a,b}$ for every $a \in \mathcal{A}$ and for every $b \in \mathcal{Z}$. If itemset a and item b are likely to be purchased together, $\lambda_{a,b}$ will be large and positive. Also, $\lambda_{a,b}$ can be negative in order to model negative correlations between items that are not purchased together. The influences of the itemsets in the basket are combined linearly to yield the score for a given item. For example, suppose the basket contains items a_1 and a_2 and $\mathcal{A}$ contains all itemsets with size less than or equal to 1. Item b is then scored as $f(\{a_1, a_2\}, b; \boldsymbol{\lambda}) = \lambda_{\varnothing,b} + \lambda_{a_1,b} + \lambda_{a_2,b}$. For a general basket $x_{i,t}$, the score of item b is:

$$f(x_{i,t}, b; \boldsymbol{\lambda}) := \sum_{\substack{a \in \mathcal{A} \text{ s.t.} \\ a \in x_{i,t}}} \lambda_{a,b}. \tag{1}$$

Our scoring model uses a total of $|\mathcal{A}|N$ variables: $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{A}|N}$. We focus primarily on the straightforward implementation $\mathcal{A} = \{\text{all itemsets of size less than or equal to } 1\}$. The itemsets of size 1 give variables $\lambda_{a,b}\ \forall a, b \in \mathcal{Z}$ that describe pairwise influences between items. The empty itemset gives rise to "base" scores $\lambda_{\varnothing,b}$ that model the likelihood of choosing item b in the absence of any information about the basket. In this implementation, the number of variables is $|\mathcal{A}|N = N^2 + N$.

The dimensionality of the problem can be controlled by limiting the set $|\mathcal{A}|$, for instance using a maximum itemset size or a minimum support requirement, where elements of $\mathcal{A}$ must be found often enough in the dataset. Alternatively, the dimensionality of the problem could be reduced by separating items into categories and using $\lambda_{A,b}$ to model the influence of having *any* item from category A on item b. For example, a_1 and a_2 could represent individual flavors of ice cream, and A the category "ice cream." The choice of which itemsets to consider is a feature selection (or model selection) problem.

We will use the training set to fit vector $\boldsymbol{\lambda}$. This means that we will construct $\boldsymbol{\lambda}$ so that the scoring function f makes accurate predictions on the training set. For the i^{th} shopping list at time step t, we will define a set of items $L_{i,t} \subset \mathcal{Z}$ that should be ranked strictly higher than some other set of items $K_{i,t} \subset \mathcal{Z}$ on the recommendation list. The value of the loss function depends on how much this is violated; specifically, we lose a point every time an item in $K_{i,t}$ is ranked above an item in $L_{i,t}$. We will subsequently explore different definitions of $L_{i,t}$ and $K_{i,t}$ and their effect on the recommendation list. The loss function, evaluated on the training set of m shopping lists, is:

$$R_{0\text{-}1}(f, X_1^m; \boldsymbol{\lambda}) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i - 1} \frac{1}{T_i} \frac{1}{|K_{i,t}|} \frac{1}{|L_{i,t}|} \sum_{l \in L_{i,t}} \sum_{k \in K_{i,t}} \mathbb{1}_{[f(x_{i,t},k;\boldsymbol{\lambda}) \geq f(x_{i,t},l;\boldsymbol{\lambda})]}. \tag{2}$$

In our algorithms, we use the exponential loss (used in boosting), a smooth upper bound on $R_{0\text{-}1}$. Specifically, we use that $\mathbb{1}_{[b \geq a]} \leq e^{b-a}$, and add an ℓ_2-norm

regularization term:

$$R_{\exp}(f, X_1^m; \boldsymbol{\lambda}) := \\ \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\frac{1}{T_i}\frac{1}{|K_{i,t}|}\frac{1}{|L_{i,t}|}\sum_{l\in L_{i,t}}\sum_{k\in K_{i,t}} e^{f(x_{i,t},k;\boldsymbol{\lambda})-f(x_{i,t},l;\boldsymbol{\lambda})} + \beta||\boldsymbol{\lambda}||_2^2, \quad (3)$$

where β is a parameter that determines the amount of regularization.

We now discuss how the sequence of items will be ordered during training, and then will specify this formulation using two choices of $L_{i,t}$ and $K_{i,t}$.

2.1.1 Fitting a sequential prediction model to an unordered set

Although the shopping list is unordered, the predictions at each time step depend on the set of items that have already been added to the basket, and thus depend indirectly on the order in which items are added to the basket. To fit the model variables to the training data, we must impose an order for the items to be added to the basket. Here we allow the predictions to influence the ordering of the items. Specifically, we assume that the customer prefers convenience, in that the next item added to the basket is the most highly recommended item on their shopping list. We then order the items according to:

$$z_{i,t+1} \in \underset{a\in X_i\setminus x_{i,t}}{\operatorname{argmax}} f(x_{i,t}, a; \boldsymbol{\lambda}). \quad (4)$$

It may be that the argmax is not unique, *i.e.*, there is a tie. Here we break ties randomly to choose the next item. The order in which items are added to the basket is a function of the model variables $\boldsymbol{\lambda}$. When fitting the model variables, we do not order the items *a priori*, rather we allow the ordering to change during fitting, together with the model variables. Our assumption in (4) could be replaced by an application-specific model of user behavior; (4) is not an accurate assumption for all applications. On the other hand, a recommender system trained using this assumption has properties that are useful in real situations, as we discuss below.

We will train the machine learning model to minimize the loss function (3) with respect to variables $\boldsymbol{\lambda}$, using (4). The qualitative effect of (4) is to put the items that are (conditionally) more likely to be purchased into the basket sooner, while leaving unlikely items for later. Once these items are in the basket, they will be used for making the subsequent predictions. Thus the model variables that generally play the largest role in the learning, and that are most accurately estimated, correspond to items that are more likely to be purchased.

One could imagine training the model using all permutations of each shopping list in the training set as an alternative to (4). As another alternative, one could randomly permute the shopping lists and include only that ordering. Even though these approaches potentially capture some realistic situations that our ordering assumption does not, we argue that it is not a good idea to do either of these. First, the number of possible permutations on even a moderately sized training set makes it computationally intractable to train using all possible permutations. Second, if the users do adhere, even loosely, to a behavioral strategy such as our assumption in (4), the model would be forced to fit many permutations that would rarely occur, and would treat those rare situations as equally important to the ones

more likely to occur. For example, a randomly permuted shopping list could place conditionally unlikely items at the beginning of the ordering. This could actually create bias against the correct recommendations.

Under the assumption of (4), we will now work with two natural constructions of the sets $L_{i,t}$ and $K_{i,t}$ used to fit and evaluate the model. The resulting loss functions can just as easily be applied to ordered data, where the sequence of events is not influenced by the recommender system, without the assumption of (4), as we show in our email recipient recommendation experiment.

2.1.2 List loss: recommend the whole list

In the first loss formulation, which we call the *list loss*, the model attempts to rank the entire shopping list higher than all items that are not on the shopping list, at every time step. A perfect model would have $f(x_{i,t}, l; \boldsymbol{\lambda}) > f(x_{i,t}, k; \boldsymbol{\lambda})$, for all $l \in X_i \setminus x_{i,t}$, and $k \in \mathcal{Z} \setminus X_i$, and for all i and t. To do this, we use (3) with $L_{i,t}$ as the set of all items remaining on the shopping list, that is, $L_{i,t} := X_i \setminus x_{i,t}$, and $K_{i,t}$ as the set of items not on the shopping list, $K_{i,t} := \mathcal{Z} \setminus X_i$. Equation (2) becomes:

$$R^{\text{list}}_{\text{0-1}}(f, X_1^m; \boldsymbol{\lambda}) := \\ \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i - 1} \frac{1}{T_i(N - T_i)(T_i - t)} \sum_{l=t+1}^{T_i} \sum_{k \in \mathcal{Z} \setminus X_i} \mathbb{1}_{[f(x_{i,t}, k; \boldsymbol{\lambda}) \geq f(x_{i,t}, z_{i,l}; \boldsymbol{\lambda})]} \tag{5}$$

and (3) then becomes:

$$R^{\text{list}}_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}) := \\ \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i - 1} \frac{1}{T_i(N - T_i)(T_i - t)} \sum_{l=t+1}^{T_i} \sum_{k \in \mathcal{Z} \setminus X_i} e^{f(x_{i,t}, k; \boldsymbol{\lambda}) - f(x_{i,t}, z_{i,l}; \boldsymbol{\lambda})} + \beta ||\boldsymbol{\lambda}||_2^2. \tag{6}$$

2.1.3 Item loss: how far down the list is the next item?

We may not need the entire shopping list near the top of the recommendation list as in the list loss, rather we might need only that one item from the shopping list is highly ranked. In this way, the loss associated with a particular basket will depend only on the highest ranked item that is still on the shopping list. We call this formulation the *item loss*. Under the item loss, a perfect prediction would have the top ranked item on the shopping list, that is $\max_{a \in X_i \setminus x_{i,t}} f(x_{i,t}, a; \boldsymbol{\lambda})$, at the top of the recommendation list:

$$\max_{a \in X_i \setminus x_{i,t}} f(x_{i,t}, a; \boldsymbol{\lambda}) > f(x_{i,t}, k; \boldsymbol{\lambda}), \text{ for all } k \in \mathcal{Z} \setminus X_i \text{ and for all } i \text{ and } t.$$

This can be realized using (2), where $L_{i,t}$ is the highest ranked item that remains on the shopping list, which by assumption (4) is $z_{i,t+1}$, and where $K_{i,t}$ is the set of items that are not on the shopping list, $\mathcal{Z} \setminus X_i$. Equation (2) is then equal to:

$$R^{\text{item}}_{\text{0-1}}(f, X_1^m; \boldsymbol{\lambda}) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i - 1} \frac{1}{T_i(N - T_i)} \sum_{k \in \mathcal{Z} \setminus X_i} \mathbb{1}_{[f(x_{i,t}, k; \boldsymbol{\lambda}) \geq f(x_{i,t}, z_{i,t+1}; \boldsymbol{\lambda})]} \tag{7}$$

and (3) becomes:

$$R_{\text{exp}}^{\text{item}}(f, X_1^m; \boldsymbol{\lambda}) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i-1} \frac{1}{T_i(N-T_i)} \sum_{k \in \mathcal{Z} \setminus X_i} e^{f(x_{i,t},k;\boldsymbol{\lambda}) - f(x_{i,t},z_{i,t+1};\boldsymbol{\lambda})} + \beta ||\boldsymbol{\lambda}||_2^2. \tag{8}$$

As an extreme example, suppose the recommendation list has a single item from the shopping list at the top, and the rest of the shopping list at the bottom. The list loss would be large, while the item loss would be small or zero. Qualitatively, item loss forces a form of rank diversity which we will now discuss.

At the first time step $t = 0$, there is no knowledge of the event sequence so the same recommendation list will be used for all shopping lists. Let us consider how this recommendation list might be constructed in order to achieve a low item loss for the following collection of example shopping lists:

Shopping list 1: onion, garlic, beef, peppers
Shopping list 2: onion, garlic, chicken
Shopping list 3: onion, garlic, fish
Shopping list 4: onion, garlic, lemon
Shopping list 5: onion, chicken, peppers
Shopping list 6: flour, oil, baking powder
Shopping list 7: flour, sugar, vanilla
Shopping list 8: flour, sugar, yeast

In these shopping lists, the three most frequent items are onion, garlic, and flour. Using item loss, we incur loss for every shopping list that does not contain the highest ranked item. A greedy strategy to minimize item loss places the most common item, onion, first on the recommendation list, thus incurring 0 loss for shopping lists 1-5. The second place in the recommendation list will not be given to the second most frequent item (garlic), rather it will be given to the most frequent item among shopping lists that do not contain onion. This means the second item on the recommendation list will be flour. With onion ranked first and flour ranked second, we incur 0 loss on shopping lists 1-5, and the loss is one for each of shopping lists 6, 7, and 8. The ranks of the remaining items do not matter for this time step, as these two ingredients have satisfied every shopping list. This greedy strategy is the same as the greedy strategy for the maximum coverage problem, in which we are given a collection of sets with some elements in common and choose k sets to cover as many elements as possible. This algorithm has been used for rank diversification (see, for instance, Radlinski et al, 2008). This is also the same strategy as used in decision lists (Rivest, 1987). This greedy strategy would be an efficient strategy to minimize item loss if we made a prediction only at $t = 0$, however, it might not truly minimize loss, and even if it does happen to minimize loss at time $t = 0$, it might not minimize loss over all time steps. In practice, we find that minimizing item loss produces a diverse ranked list at each time step: it attempts to ensure that an item from each shopping list is ranked highly, as opposed to simply ranking items based on popularity.

The loss functions in (6) and (8) can also be used in the case of ordered data, that is, when the sequence $\{z_{i,t}\}_t$ is pre-specified rather than modeled using (4). In our email recipient recommendation experiment, we treat the data as ordered and use exactly the loss function in (6).

2.2 Patient condition prediction

Here we tailor this formulation to patient condition prediction in the context of data from a large clinical trial. In this trial, patients visited the doctor periodically and reported all medical conditions for which they were taking medications. The names of the medical conditions were taken from the Medical Dictionary for Regulatory Activities (MedDRA). The dataset includes activities such as vitamin supplementation and flu shots as medical "conditions," but mainly consists of conditions that are not voluntarily chosen by the patients. We chose to predict both voluntary and involuntary conditions/activities.

The notation is similar to that of the formulation in (2): Patient i makes T_i visits to a doctor at time steps $t = 1, \ldots, T_i$. At time t, he or she reports a set of conditions $z_{i,t} \subseteq \mathcal{Z}$ where $\mathcal{Z}$ is the set of all possible conditions and $|\mathcal{Z}| = N$. Note that in the online grocery store recommender system $z_{i,j}$ was a single item, whereas for this problem $z_{i,j}$ is a set of conditions. A patient history is a record of which conditions were reported at which times and forms a sequence of sets. We treat the patient visits as ordered in time, as this order may be a useful feature for prediction. However, each visit itself consists of a set of symptoms which we treat as unordered. We denote the full patient history for patient i at time t as $x_{i,t} = \{z_{i,j}\}_{j=1,\ldots,t}$. Given the current patient history, we predict the next set of conditions in the sequence, that is, we use $x_{i,t}$ to predict the set $z_{i,t+1}$. As before, we use a scoring function $f^{\text{cond}}(x_{i,t}, b)$ to score the likelihood that condition b is in $z_{i,t+1}$ given the patient history $x_{i,t}$. The score of condition b is the base score of b, namely $\lambda_{\varnothing,b}$, combined linearly with the influences of conditions from the patient history:

$$f^{\text{cond}}(x_{i,t}, b; \boldsymbol{\lambda}) := \lambda_{\varnothing,b} + \sum_{j=1}^{t} \sum_{a \in z_{i,j}} \lambda_{a,b}. \tag{9}$$

This model can be specialized further if desired. For instance, it may be that conditions that occurred far in the past are less informative about the future than conditions that occurred recently. In this case, we could add weights to (9) which reduce the influence of conditions as the elapsed time increases. Additionally, the score in (9) can easily be extended to include itemsets of size larger than 1.

We denote a complete patient history as $X_i = x_{i,T_i}$ and train the algorithm using m randomly selected complete patient histories. Unlike the online grocery store recommender system, in patient symptom prediction it is not natural to make a prediction before the patient's first visit ($t = 0$), thus we make predictions only at visits $t = 1, \ldots, T_i - 1$. Also, the same condition can occur at multiple visits throughout the patient history, whereas in the online grocery store recommender system items appear on a shopping list only once.

Some patients present *chronic, pre-existing* conditions that were present before their first visit and persisted after their last visit. Common chronic, pre-existing conditions include Hypertension (high blood pressure), Hypercholesterolaemia (high cholesterol), and Asthma. It is possible for a condition to be chronic, pre-existing in one patient, but not in another. For instance, some patients developed Hypertension during the study, so Hypertension was not pre-existing in these patients. We denote the set of chronic, pre-existing conditions for patient i as $C_i \subseteq \mathcal{Z}$, and place each chronic, pre-existing condition in the set of conditions for each visit: $c \in z_{i,j}$ for all $c \in C_i$, for $j = 1, \ldots, T_i$, and for all i. Chronic, pre-existing

conditions were used to make predictions for subsequent conditions, but we did not attempt to predict them because predicting the recurrence of a chronic condition is trivial. We removed chronic, pre-existing conditions from the loss function by defining $\tilde{z}_{i,j} = z_{i,j} \setminus C_i$ as the set of reported conditions excluding chronic, pre-existing conditions. We then adapt the framework of (2) and (3) for training by setting $L_{i,t} = \tilde{z}_{i,t+1}$, the correct, subsequent set of non-trivial conditions, and $K_{i,t} = \mathcal{Z} \setminus z_{i,t+1}$, all other possible conditions. Then (2) becomes:

$$R_{0-1}^{\text{cond}}(f^{\text{cond}}, X_1^m; \boldsymbol{\lambda}) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T_i-1} \left[\frac{1}{(T_i - 1)(N - |z_{i,t+1}|)} \times \right.$$
$$\left. \sum_{k \in \mathcal{Z} \setminus z_{i,t+1}} \sum_{l \in \tilde{z}_{i,t+1}} \frac{1}{|\tilde{z}_{i,t+1}|} \mathbb{1}_{[f^{\text{cond}}(x_{i,t},k;\boldsymbol{\lambda}) \geq f^{\text{cond}}(x_{i,t},l;\boldsymbol{\lambda})]} \right]. \tag{10}$$

If at a particular visit the only conditions reported were chronic, pre-existing conditions, then $\tilde{z}_{i,t+1} = \varnothing$ and the inner most sum is simply not evaluated for that i and t to avoid dividing by zero with $|\tilde{z}_{i,t+1}|$. We further write (3) as:

$$R_{\text{exp}}^{\text{cond}}(f^{\text{cond}}, X_1^m; \boldsymbol{\lambda}) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T_i-1} \left[\frac{1}{(T_i - 1)(N - |z_{i,t+1}|)} \times \right.$$
$$\left. \sum_{k \in \mathcal{Z} \setminus z_{i,t+1}} \sum_{l \in \tilde{z}_{i,t+1}} \frac{1}{|\tilde{z}_{i,t+1}|} e^{f^{\text{cond}}(x_{i,t},k;\boldsymbol{\lambda}) - f^{\text{cond}}(x_{i,t},l;\boldsymbol{\lambda})} \right] + \beta ||\boldsymbol{\lambda}||_2^2. \tag{11}$$

2.3 A general view of sequential event prediction

The patient condition prediction problem in Section 2.2 generalizes the idea in Section 2.1 from each event being a single item to each event being a set of items. In other words, $z_{i,t}$ can be considered generally as a set, potentially of cardinality 1. The loss functions in (5), (7), and (10) were each shown to be specifications of the general loss function (2), which can be considered to be a unified loss function for sequential event prediction. Many problems can be expressed in this framework, often without requiring much additional specification as we show with email recipient recommendation. The loss function in (2) should be modified to reflect the evaluation metric relevant to each particular problem, as we discuss further when discusing related works in Section 4.

2.4 Optimization strategies and algorithms

The model variables $\boldsymbol{\lambda}$ are chosen to minimize the loss on the training set by optimizing the appropriate loss function (Equation 6, 8, or 11). Our formulation involves $N^2 + N$ variables (more generally, $|\mathcal{A}|N$), which may be prohibitive, both because of the computational complexity of the optimization problem as well as the ability to accurately determine so many variables from a limited training set. First we will present a strategy for constraining many of the variables and significantly reducing the size of the optimization problem. Then we will present algorithms

for optimization separately for ordered problems, like patient condition prediction, and for unordered problems under the assumption of (4). Our optimization algorithm for unordered problems are written for the purpose of exposition for itemsets with size 0 or 1, but can easily be extended to general itemsets.

2.4.1 ML-constrained minimization

We have used two main strategies for fitting the model variables to the training set. In our first strategy, which we will call *one-stage* minimization, each variable $\lambda_{a,b}$ is a free variable in the minimization problem, leading to an optimization problem involving $|\mathcal{A}|N$ variables. Our second strategy, which we call *Maximum likelihood (ML)-constrained* minimization, reduces the size of the optimization problem by, for every non-empty itemset a, forcing each $\lambda_{a,b}$ to be proportional to $\hat{\mathbb{P}}(b|a)$, the ML estimate of the conditional probability of having item b in a basket given that itemset a is in the basket. Specifically, we set

$$\lambda_{a,b} = \mu_a \hat{\mathbb{P}}(b|a) \tag{12}$$

where μ_a is a free variable fit during the optimization and does not depend on b. $\hat{\mathbb{P}}(b|a)$ is estimated directly from the training data, prior to any optimization, as the proportion of shopping lists containing b that also contain a. Then, the ML-constrained model is:

$$f_{\mathrm{ML}}(x_{i,t}, b; \boldsymbol{\lambda}_\varnothing, \boldsymbol{\mu}) := \lambda_{\varnothing,b} + \sum_{\substack{a \in \mathcal{A}\setminus\varnothing \text{ s.t.} \\ a \in z_{i,t}}} \mu_a \hat{\mathbb{P}}(b|a). \tag{13}$$

We are in essence modeling the full conditional probability $\mathbb{P}(b|x_{i,t})$ as a linear mixture of the single-itemset conditional probabilities $\mathbb{P}(b|a)$ for every allowed itemset a in $x_{i,t}$. To use this strategy, we first compute the ML estimates of the conditional probabilities. Then the N base scores $\lambda_{\varnothing,a}$ and the $|\mathcal{A}|$ proportionality coefficients μ_a are fit during the minimization, for an optimization problem on $|\mathcal{A}| + N$ variables. Appropriate restrictions on $|\mathcal{A}|$ (for example, itemsets of size less than or equal to 1) lead to an optimization problem over $O(N)$ variables. These variables are fit by minimizing the analogous list loss or item loss function:

$$R^{\mathrm{list}}_{\exp}(f_{\mathrm{ML}}, X_1^m; \boldsymbol{\lambda}_\varnothing, \boldsymbol{\mu}) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\left[\frac{1}{T_i(N-T_i)(T_i-t)}\times \right.$$
$$\left.\sum_{l=t+1}^{T_i}\sum_{k\in\mathcal{Z}\setminus X_i} e^{f_{\mathrm{ML}}(x_{i,t},k;\boldsymbol{\lambda}_\varnothing,\boldsymbol{\mu}) - f_{\mathrm{ML}}(x_{i,t},z_{i,l};\boldsymbol{\lambda}_\varnothing,\boldsymbol{\mu})}\right] + \beta||\boldsymbol{\lambda}_\varnothing||_2^2 + \beta||\boldsymbol{\mu}||_2^2. \tag{14}$$

$$R^{\mathrm{item}}_{\exp}(f_{\mathrm{ML}}, X_1^m; \boldsymbol{\lambda}_\varnothing, \boldsymbol{\mu}) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\left[\frac{1}{T_i(N-T_i)}\times \right.$$
$$\left.\sum_{k\in\mathcal{Z}\setminus X_i} e^{f_{\mathrm{ML}}(x_{i,t},k;\boldsymbol{\lambda}_\varnothing,\boldsymbol{\mu}) - f_{\mathrm{ML}}(x_{i,t},z_{i,t+1};\boldsymbol{\lambda}_\varnothing,\boldsymbol{\mu})}\right] + \beta||\boldsymbol{\lambda}_\varnothing||_2^2 + \beta||\boldsymbol{\mu}||_2^2. \tag{15}$$

For patient condition prediction, $\hat{\mathbb{P}}(b|a)$ is the ML estimate of the probability that condition b will occur at a future visit given that condition a is currently presented. The ML-constrained model corresponding to (9) is then

$$f_{\mathrm{ML}}^{\mathrm{cond}}(x_{i,t}, b; \boldsymbol{\lambda}_{\varnothing}, \boldsymbol{\mu}) := \boldsymbol{\lambda}_{\varnothing,b} + \sum_{j=1}^{t} \sum_{a \in z_{i,j}} \mu_a \hat{\mathbb{P}}(b|a), \tag{16}$$

and the loss function to be minimized is defined similarly to (11) as follows:

$$R_{\mathrm{exp}}^{\mathrm{cond}}(f_{\mathrm{ML}}^{\mathrm{cond}}, X_1^m; \boldsymbol{\lambda}_{\varnothing}, \boldsymbol{\mu}) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T_i - 1} \sum_{k \in \mathcal{Z} \setminus z_{i,t+1}} \left[\frac{1}{(T_i - 1)(N - |z_{i,t+1}|)} \times \right.$$

$$\left. \sum_{l \in \tilde{z}_{i,t+1}} \frac{1}{|\tilde{z}_{i,t+1}|} e^{f_{\mathrm{ML}}^{\mathrm{cond}}(x_{i,t}, k; \boldsymbol{\lambda}_{\varnothing}, \boldsymbol{\mu}) - f_{\mathrm{ML}}^{\mathrm{cond}}(x_{i,t}, l; \boldsymbol{\lambda}_{\varnothing}, \boldsymbol{\mu})} \right] + \beta ||\boldsymbol{\lambda}_{\varnothing}||_2^2 + \beta ||\boldsymbol{\mu}||_2^2. \tag{17}$$

In the sections that follow, we will describe optimization algorithms in terms of $\boldsymbol{\lambda}$. All of these algorithms are suitable for the optimization step in the ML-constrained minimization strategy by simply replacing $\lambda_{a,b}$ with $\mu_a \hat{\mathbb{P}}(b|a)$, where $\hat{\mathbb{P}}(b|a)$ has already been computed.

2.4.2 Optimization for ordered problems

For patient condition prediction, (11) and (17) are convex in $\boldsymbol{\lambda}$ and the minimization can be done easily using any algorithm for unconstrained nonlinear optimization. For the recommender system, when the items are ordered (as they are in our email recipient recommendations experiment) the loss functions (6), (8), (14), and (15) are all convex. Because the number of variables can be very large, particularly using the one-stage minimization strategy, here we use the limited memory implementation of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (Byrd et al, 1995; Zhu et al, 1997).

2.4.3 Optimization for the online grocery store recommender system

For the recommender system using the assumption in (4), the basket at any time step is itself a function of the recommender system, *i.e.*, of $\boldsymbol{\lambda}$. Small changes in $\boldsymbol{\lambda}$ can change $z_{i,t+1}$, which in turn changes the basket, and thus changes all subsequent predictions. This can significantly alter the value of the loss, and is why the loss functions in (6) and (8) are generally discontinuous. To illustrate the discontinuity, suppose there are three items in the inventory, bread, milk, and eggs, and that a customer has only bread and milk on his or her shopping list. We will take the variables $\lambda_{a,b}$ to be, from row a to column b:

$\lambda_{a,b}$	bread	milk	eggs
bread	0	1	0
milk	0	0	1
eggs	0	0	0

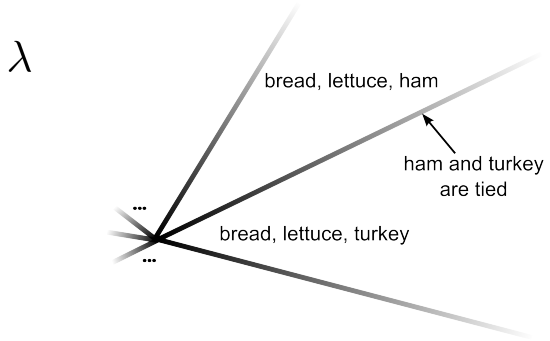

Fig. 3 An illustration of how the model variables can be partitioned into regions that lead to different orderings of the items in each shopping basket. The borders between regions correspond to selections of model variables for which the argmax in (4) is not unique, *i.e.*, there is a tie. The regions are polyhedral, and the objective function is convex over each region but discontinuous at the borders.

Now consider two vectors of base scores $[\lambda_{\varnothing,\text{bread}}, \lambda_{\varnothing,\text{milk}}, \lambda_{\varnothing,\text{eggs}}]$ which are arbitrarily close: $\boldsymbol{\lambda}^b_\varnothing = [0.5+\epsilon, 0.5, 0]$ and $\boldsymbol{\lambda}^m_\varnothing = [0.5, 0.5+\epsilon, 0]$. We now show that even though these sets of variables are arbitrarily close, they give very different values of the objective function due to the ordering assumption of (4). Using $\boldsymbol{\lambda}^b_\varnothing$, bread is the highest ranked item and will be the first item added to the basket, whereas using $\boldsymbol{\lambda}^m_\varnothing$, milk is the highest ranked item and will be the first item added to the basket. On the next prediction in the sequence, if bread is in the basket then the scores of milk and eggs are $\lambda_{\varnothing,\text{milk}} + \lambda_{\text{bread,milk}} = 1.5$ and $\lambda_{\varnothing,\text{eggs}} + \lambda_{\text{bread,eggs}} = 0$ respectively. Milk correctly has the higher score, and the loss is $e^{-1.5}$. However, if milk is the first item added, then on the next prediction in the sequence the scores of bread and eggs are $\lambda_{\varnothing,\text{bread}} + \lambda_{\text{milk,bread}} = 0.5$ and $\lambda_{\varnothing,\text{eggs}} + \lambda_{\text{milk,eggs}} = 1$ respectively. Eggs is mistakenly given the highest score and we incur a larger loss of $e^{1-0.5} = e^{0.5}$. Thus, an ϵ-sized change in $\boldsymbol{\lambda}$ led to a large change in the value of the objective.

The discontinuities occur at values of $\boldsymbol{\lambda}$ where there are ties in the ranked list, that is, where the model is capable of producing multiple orderings. These discontinuities partition the variable space into regions that correspond to different orderings. Figure 3 is an illustration of how the space of $\boldsymbol{\lambda}$ is partitioned by different orderings, with ties between items on the borders. The loss function is convex over each region and discontinuities occur only at the region borders. We now show that these regions are convex, which will lead to an optimization strategy.

Proposition 1 *Take $\mathcal{A}$ the set of itemsets of size 1, along with the empty set. Let Λ_{z^*} be the set of $\boldsymbol{\lambda} \in \mathbb{R}^{N^2+N}$ in one-stage minimization or $(\boldsymbol{\lambda}_\varnothing, \boldsymbol{\mu}) \in \mathbb{R}^{2N}$ in ML-constrained minimization that can produce an ordering $\{z^*_{i,t}\}_{i,t}$ under the assumption of (4). Then, Λ_{z^*} is a polyhedron.*

Proof A particular ordering z^* is produced when, at each time step, the next item $z^*_{i,t+1}$ has the highest score of the items in the shopping list. This needs to be true for all training baskets i. The region of $\boldsymbol{\lambda}$ corresponding to ordering z^* is, using

one-stage minimization:

$$\begin{aligned}\Lambda_{z^*} &:= \left\{ \boldsymbol{\lambda} : \forall i, t,\ \ z^*_{i,t+1} \in \operatorname*{argmax}_{a \in X_i \setminus x^*_{i,t}} f(x^*_{i,t}, a; \boldsymbol{\lambda}) \right\} \\ &= \left\{ \boldsymbol{\lambda} : \forall i, t,\ \ z^*_{i,t+1} \in \operatorname*{argmax}_{a \in X_i \setminus x^*_{i,t}} \sum_{j=0}^{t} \lambda_{z^*_{i,j}, a} \right\}\end{aligned}$$

where $x^*_{i,t}$ is $z^*_{i,0} \cup \ldots \cup z^*_{i,t}$. In other words, because $z^*_{i,t+1}$ was chosen before $z^*_{i,k}$ for all $k > t+1$, it must be true that the score of $z^*_{i,t+1}$ is greater than or equal to the score of $z^*_{i,k}$: $f(x^*_{i,t}, z^*_{i,t+1}) \geq f(x^*_{i,t}, z^*_{i,k}), \forall k > t+1$. These constraints are another way to define Λ_{z^*}:

$$\begin{aligned}\Lambda_{z^*} := \Bigg\{ \boldsymbol{\lambda} : \sum_{j=0}^{t} \lambda_{z^*_{i,j}, z^*_{i,t+1}} \geq \sum_{j=0}^{t} \lambda_{z^*_{i,j}, z^*_{i,k}}, \\ i = 1, \ldots, m,\ t = 0, \ldots, T_i - 2,\ k = t+2, \ldots, T_i \Bigg\}.\end{aligned} \tag{18}$$

Thus Λ_{z^*} can be defined by a set of $\sum_{i=1}^{m} \frac{1}{2}(T_i - 1)T_i$ linear inequalities and is a polyhedron. The result holds when using the ML-constrained minimization strategy, for which

$$\begin{aligned}\Lambda_{z^*} := \Bigg\{ \boldsymbol{\lambda}_\varnothing, \boldsymbol{\mu} : \lambda_{\varnothing, z^*_{i,t+1}} + \sum_{j=1}^{t} \mu_{z^*_{i,j}} \hat{\mathbb{P}}(z^*_{i,t+1} | z^*_{i,j}) \geq \lambda_{\varnothing, z^*_{i,k}} + \sum_{j=1}^{t} \mu_{z^*_{i,j}} \hat{\mathbb{P}}(z^*_{i,k} | z^*_{i,j}), \\ i = 1, \ldots, m,\ t = 0, \ldots, T_i - 2,\ k = t+2, \ldots, T_i \Bigg\},\end{aligned} \tag{19}$$

which is also a collection of linear inequalities. □

The proposition is true for each ordering z^* that can be realized, and thus the whole space can be partitioned into polyhedral regions. When the variables $\boldsymbol{\lambda}$ (or equivalently $\boldsymbol{\mu}$) are constrained to a particular ordering Λ_{z^*}, (6), (8), (14), and (15) are convex because they are fixed positively weighted sums of convex functions.

2.4.4 Convex optimization within regions, simulated annealing between regions

Because Λ_{z^*} is convex for each z^*, it is possible to find the optimal $\boldsymbol{\lambda}$ within Λ_{z^*} using convex programming. Our goal is to minimize the loss across all possible orderings z^*, so we need also to explore the space of possible orderings. Our first approach is to use simulated annealing, as detailed in Algorithm 1, to hop between the different regions, using convex optimization within each region.

Simulated annealing is an iterative procedure where $\boldsymbol{\lambda}$ is updated step by step. Steps that increase the loss are allowed with a probability that depends on a "temperature" variable. The temperature is decreased throughout the procedure so that steps that increase the loss become increasingly improbable. The algorithm begins with an initial ordering, then the minimizer within that region is found by convex optimization. Then we use a simulated annealing step to move to a

neighboring ordering, and the process is repeated. There are many "unrealizable" orderings that can be achieved only by a trivial model in which all of the variables λ equal the same constant so that the items are tied at every prediction. Thus, randomly permuting the ordering as is usually done in simulated annealing will often yield only trivial neighbors. An alternative strategy is to choose a direction in the variable space (for example, the direction of gradient descent) and to step in that direction from the current position of $\boldsymbol{\lambda}$ until the ordering changes. This new ordering is a realizable neighbor and can be used to continue the simulated annealing. Additional neighbors can be discovered by stepping in the variable space in different directions, for instance orthogonal to the gradient. The move to the new ordering is accepted with a probability that depends on the change in loss between the optimal solutions for the two orderings, and the temperature variable. This is done for a fixed number of steps, and finally the output is the best solution that was encountered during the search.

Algorithm 1: A combination of convex optimization and simulated annealing for fitting $\boldsymbol{\lambda}$.

Data: Training set X_1^m, number of simulated annealing steps T_S, annealing schedule $Temp$
Result: $\boldsymbol{\lambda}_{\text{best}}$
Begin with an initial ordering $\{z_{i,t}\}_{i,t}$
Form the constraints Λ_z associated with this ordering (Equation 18 or 19)
Solve the convex program $\boldsymbol{\lambda}^* \in \text{argmin}_{\boldsymbol{\lambda} \in \Lambda_z} R_{\exp}(f, X_1^m; \boldsymbol{\lambda})$ (Equation 6, 8, 14, or 15)
Set $\boldsymbol{\lambda}_{\text{best}} = \boldsymbol{\lambda}^*$
for $t = 1$ **to** T_S **do**
  Find a neighboring ordering $\{z'_{i,t}\}_{i,t}$
  Form the constraints $\Lambda_{z'}$ associated with the new ordering
  Solve the convex program $\boldsymbol{\lambda}'^* \in \text{argmin}_{\boldsymbol{\lambda} \in \Lambda_{z'}} R_{\exp}(f, X_1^m; \boldsymbol{\lambda})$
  Sample a number q uniformly at random from $[0, 1]$
  if $\exp((R_{\exp}(f, X_1^m; \boldsymbol{\lambda}^*) - R_{\exp}(f, X_1^m; \boldsymbol{\lambda}'^*))/Temp(t)) > q$ **then**
    Accept this move: $\boldsymbol{\lambda}^* = \boldsymbol{\lambda}'^*$
    if $R_{\exp}(f, X_1^m; \boldsymbol{\lambda}^*) < R_{\exp}(f, X_1^m; \boldsymbol{\lambda}_{\text{best}})$ **then**
      $\boldsymbol{\lambda}_{\text{best}} = \boldsymbol{\lambda}^*$

2.4.5 Gradient descent

When N is large, it can be expensive to solve the convex program at each step of simulated annealing in Algorithm 1, particularly using the one-stage optimization strategy which requires $N^2 + N$ variables. One of the difficulties is operating on the Hessian matrix, which contains more than N^4 elements. It is possible to use a first-order, constrained method such as projected gradient descent to solve the convex program in Algorithm 1. However, projected gradient descent is known to behave poorly when optimizing over polyhedral sets (Bertsekas, 1995), and we found that it did not perform well in our experiments. Improvements on projected gradient descent require a scaling matrix, which in this case would again have more than N^4 elements and could be prohibitively expensive for N large (Bertsekas, 1995). It may be more efficient to use an unconstrained first-order method such as pure

gradient descent. Variants of gradient descent, particularly stochastic gradient descent, are known to have excellent scalability properties in large-scale learning problems (Bottou, 2010). It is likely that a gradient descent algorithm will cross the discontinuities, and there are no convergence guarantees. In Algorithm 2, we ensure that the gradient descent terminates by imposing a limit on the number of steps that increase the loss. We take as our result the best value that was encountered during the search.

Algorithm 2: A gradient descent algorithm to fit $\boldsymbol{\lambda}$.

Data: Training set X_1^m, maximum number of steps that increase loss T_G, step size γ
Result: $\boldsymbol{\lambda}_{\text{best}}$
Begin with some initial $\boldsymbol{\lambda}_0$ and the associated $R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_0)$ (Equation 6, 8, 14, or 15).
Set: $\boldsymbol{\lambda}_{\text{best}} = \boldsymbol{\lambda}_0$
$t = 0$ (an index for all steps)
$l = 0$ (an index for steps that increase loss)
while $l < T_G$ **do**
  Take a step of gradient descent:
  $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \gamma \bigtriangledown R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_t)$
  if $R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_{t+1}) < R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_{\text{best}})$ **then**
    $\boldsymbol{\lambda}_{\text{best}} = \boldsymbol{\lambda}_{t+1}$
  if $R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_{t+1}) > R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_t)$ **then**
    $l = l + 1$
  $t = t + 1$

2.5 Baseline algorithms

There are not many baseline algorithms for sequential event prediction in the literature, and it is not clear how to adapt standard modeling techniques (*e.g.*, logistic regression) because they estimate full probabilities rather than partial probabilities. The difficulties in using regression for the online grocery store recommender system are discussed in Appendix A, and a more detailed discussion is in Rudin et al (2012).

2.5.1 Association rules

Association rules are a tool from data mining that have been used for sequential event prediction because they have the advantage of being able to model the conditional probabilities directly. We will use a classic associative classification algorithm as a baseline. In this context, an association rule is a rule "$a \rightarrow b$," meaning that itemset a on the shopping list implies item b is also on the shopping list. We define the *confidence* of rule "$a \rightarrow b$" to be the proportion of training lists with itemset a that also have item b: $\text{Conf}(a \rightarrow b) = \hat{\mathbb{P}}(b|a) = \frac{\#(a \cup b)}{\#a}$. A natural strategy for using association rules for sequential event prediction is to: 0) Specify a set $\mathcal{A}$ of allowed itemsets. 1) Form all rules with left-hand side a an allowed itemset in the basket and right-hand side b an item not in the basket. 2) For each

right-hand side b, find the rule with the maximum confidence. 3) Rank the right-hand sides (items not in the basket) in order of descending confidence, and use this ranked list for predictions. This is the "max-confidence" algorithm, described also by Rudin et al (2012).

In the online grocery store recommender system experiment, we set $\mathcal{A}$ to be the set of itemsets of size less than or equal to 1. We use the email recipient recommendation experiment to demonstrate how all of the models and loss functions generalize naturally to larger itemsets. In that experiment, we mine itemsets of size up to 4 and use a minimum support requirement to control dimensionality, where itemsets in $\mathcal{A}$ are required to appear in at least a certain fraction examples in the dataset.

For patient condition prediction, a rule "$a \to b$" means that condition a implies condition b at some future visit. For the max-confidence algorithm, we form all rules with left-hand side a being a condition in the patient's medical history, and right-hand side b any of the non-chronic conditions.

2.5.2 Item-based collaborative filtering

There are a large number of collaborative filtering algorithms that have been used for recommender systems. It is possible to use an item-based collaborative filtering algorithm to make a prediction at each step in the sequence and so we use cosine similarity item-based collaborative filtering (Sarwar et al, 2001) as a baseline method. Cosine similarity is intended for a setting in which user i applies a rating $R_{i,a}$ to item a. To adapt it to sequential recommendations, we let $R_{i,a} = 1$ if user i purchased item a, and 0 otherwise. For each item a, we construct the binary "ratings" vector $\mathbf{R}_a = [R_{1,a}, \ldots, R_{m,a}]$. We then compute the cosine similarity between every pair of items a and b:

$$sim(a, b) = \frac{\mathbf{R}_a \cdot \mathbf{R}_b}{||\mathbf{R}_a||_2 ||\mathbf{R}_b||_2}. \tag{20}$$

To make a prediction from a partial sequence $x_{i,t}$, we score each item b as:

$$f_{sim}(x_{i,t}, b) := \frac{\sum_{a \in x_{i,t}} sim(a, b)}{\sum_{a \in \mathcal{Z}} sim(a, b)}. \tag{21}$$

In Section 4, we discuss in depth why item-based collaborative filtering is not a natural fit for sequential event prediction problems. Nevertheless, since it is commonly used for similar problems, we use it as a baseline in one of our experiments.

3 Experiments and Discussion

We apply our method first to the patient condition prediction problem, then to the online grocery store recommendation problem, and finally to the email recipient recommendation problem. Each of the sequential event prediction strategies presented here (association rules, one-stage minimization, and ML-constrained minimization) uses a set of values to describe the influence of event a on the subsequent occurrence of event b. For association rules, the influence of a on b is Conf($a \to b$),

which is equivalent to the ML estimate $\hat{\mathbb{P}}(b|a)$. Under ML-constrained minimization, the influence variables are $\mu_a\hat{\mathbb{P}}(b|a)$, and the matrix of variables is thus obtained by scaling each row of the association rule confidence matrix by μ_a. With one-stage minimization, the influence variables are $\lambda_{a,b}$. We will give examples of these influence variables for patient condition prediction and the online grocery store recommender system, and discuss their interpretability. We then present results from a large set of experiments showing that our method performs well compared to the max-confidence association rule algorithm, and in the final set of experiments, compared to cosine similarity item-based collaborative filtering.

3.1 Patient condition prediction

We applied our method to the medical histories of 2433 patients. Each patient made multiple visits to the doctor, at an average of 6.4 visits per patient (standard deviation, 3.0). At each visit, multiple conditions are reported, with an average of 3.2 conditions per visit (standard deviation, 2.0). We perform patient level predictions, meaning for each patient we predict the conditions that the patient will experience in the future. Conditions came from a library of 1864 possible conditions. Fitting model variables required an optimization problem on 3,476,360 variables for the one-stage minimization strategy and 3,728 variables for ML-constrained minimization.

To illustrate the behavior of our model, in Figure 4 we show the model influence variables corresponding to the ten most frequent conditions, fitted to a randomly selected set of 2190 ($= 0.9 \times 2433$) patients and normalized.

The association rule confidence matrix in Figure 4 shows $\text{Conf}(a \rightarrow b)$ for each pair of items in row a and column b. The high confidence values on the diagonal indicate that the conditional probability of having these conditions in the future given their past occurence is high. In many instances, but not all, these conditions are chronic, pre-existing conditions. In addition to the high confidence values along the diagonal, the rules with Hypertension and Nutritional support on the right-hand side have higher confidences, in part because Hypertension and Nutritional support are the most common conditions. The ML-constrained influence variables, $\mu_a\hat{\mathbb{P}}(b|a)$, are a row-weighted version of the association rule confidence matrix, yet the main features are different, and in fact the ML-constrained influence variables are similar to those of one-stage minimization, $\lambda_{a,b}$. With both minimization strategies, the strength with which each condition predicts itself (the variables on the diagonal) is greatly reduced. This is because in many instances these are chronic, pre-existing conditions, and so they are excluded from the loss function and the model has no reason to predict them. For both minimization strategies, the variables along the top row show that Hypertension most strongly predicts Hypercholesterolaemia, Prophylaxis, and Headache. Hypercholesterolaemia (high cholesterol) is correlated with obesity, as is Hypertension, so they often occur together. Prophylaxis is preventative medicine which in this context almost always means taking medications, such as aspirin, to preempt heart problems. Hypertension is a risk factor for heart problems, and so the connection with Prophylaxis is also relevant. Finally, the frequency of Headaches is also known to increase with obesity (Bigal et al, 2006). In our dataset, the reasons for Nutritional support are

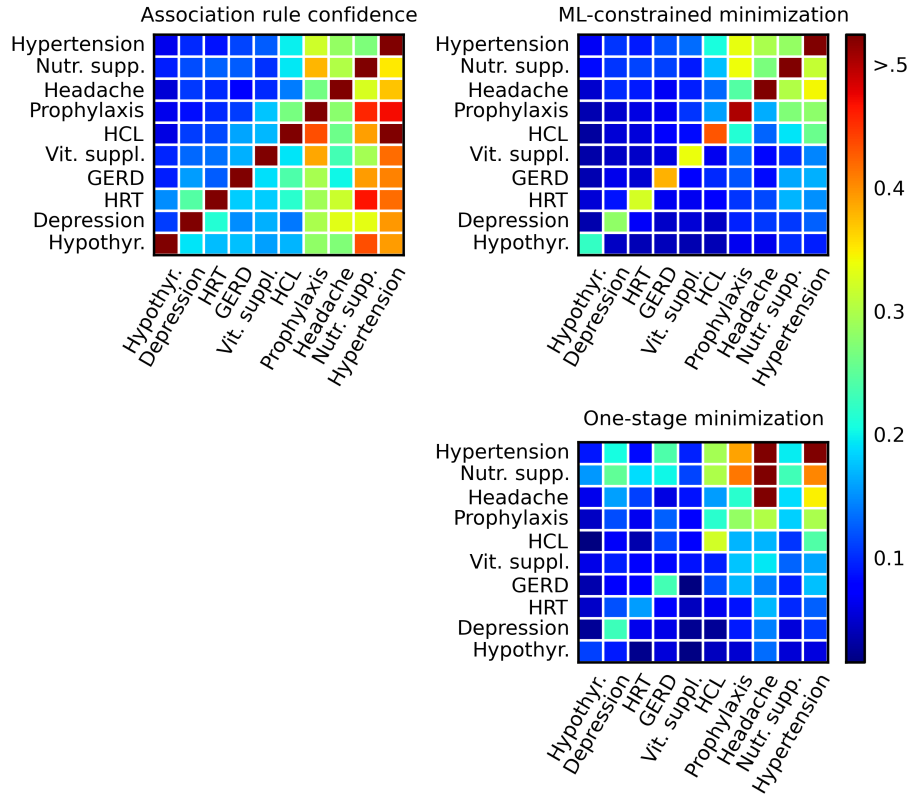

Fig. 4 An example of fitted model variables for the ten most frequent conditions in the patient condition prediction problem, for one-stage minimization and ML-constrained minimization, together with the association rule confidence matrix. This figure illustrates the differences between the fitted variables of the different models. Row a column b is: $\mathrm{Conf}(a \to b)$ for association rules; $\mu_a \hat{\mathbb{P}}(b|a)$ for ML-constrained minimization; and $\lambda_{a,b}$ for one-stage minimization. Abbreviated symptoms are Nutritional support (Nutr. supp.), Hypercholesterolaemia (HCL), Vitamin supplementation (Vit. suppl.), Gastroeophageal reflux disease (GERD), Hormone replacement therapy (HRT), and Hypothyroidism (Hypothyr.).

more varied so it is difficult to interpret the relation between Nutritional support and Prophylaxis, Headache, and Hypertension.

To evaluate the performance of our method, we performed ten iterations of random sub-sampling. For each iteration, we randomly sampled (without replacement) a training set of 500 patients and a test set of 500 patients and applied the max-confidence association rule algorithm, one-stage minimization, and ML-constrained minimization. To set the amount of ℓ_2-norm regularization in the loss function, β, we did 10-fold cross validation on each training set separately with $\beta = 0.001, 0.005, 0.01, 0.05$, and 0.1. We then set β to the value that minimized the mean error over the validation sets. With one-stage minimization, chosen values of β ranged from 0.001 to 0.05, with $\beta = 0.005$ chosen most frequently. with ML-constrained minimization $\beta = 0.01$ was always chosen. The error on the training

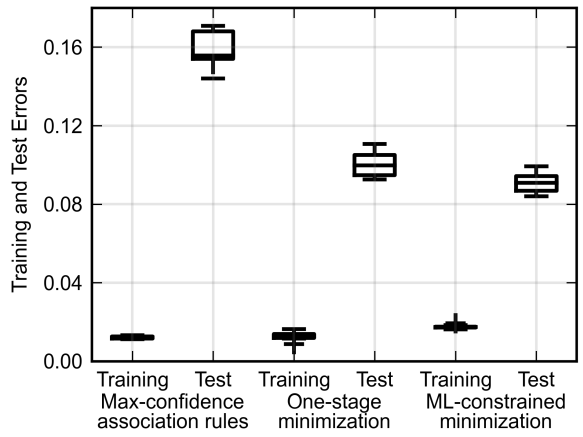

Fig. 5 Training and test errors for patient condition prediction.

and test sets were evaluated using (10), and boxplots of the results across all 10 iterations are in Figure 5. When evaluating the test error in Figure 5, we excluded conditions that were not encountered in the training set because these conditions were impossible to predict and resulted in a constant error for all methods. The median error including conditions not encountered in the training set was 0.22 for max-confidence association rules, 0.14 for one-stage minimization, and 0.14 for ML-constrained minimization.

Our method, using both minimization strategies, performed very well compared to max-confidence association rules, which seems to have had an overfitting problem, judging from the difference between training and test results. ML-constrained minimization used far fewer variables than one-stage minimization (about 3000 compared to about 3.5 million) and generalized well.

3.2 Online grocery store recommender system

Our recommender system dataset is derived from the publicly available ICCBR Computer Cooking Contest recipe book (ICCBR, 2011). The original dataset is 1490 recipes, each of which, among other things, contains a list of ingredients. We treated the ingredients in each recipe as unordered items on a shopping list.

To illustrate the effect of the minimization strategy and the loss function, we first fit the model to a reduced dataset consisting of the most popular 15 ingredients and recipes containing more than one of the top 15 ingredients. A randomly selected set of 100 recipes was used to train the model using the convex programming within regions, simulated annealing between regions algorithm (Algorithm 1). The model was trained using both the one-stage minimization and ML-constrained minimization strategies, for both the list loss and the item loss.

Figure 6 shows the ordered recommendations provided by the model to a user with an empty basket ($t = 0$). These ranked lists are determined entirely by the base scores $\boldsymbol{\lambda}_{\varnothing}$.

	ML-constrained minimization		One-stage minimization	
Frequency	Item loss	List loss	Item loss	List loss
Salt	Salt	Salt	Salt	Salt
Onion	Garlic	Garlic	Garlic	Onion
Garlic	Sugar	Onion	Sugar	Garlic
Flour	Onion	Oil	Onion	Oil
Sugar	Oil	Water	Flour	Sugar
Water	Water	Sugar	Milk	Water
Butter	Bell pepper	Bell pepper	Oil	Flour
Pepper	Pepper	Milk	Egg	Butter
Oil	Parsley	Parsley	Water	Pepper
Egg	Milk	Egg	Pepper	Bell pepper
Bell pepper	Flour	Butter	Bell pepper	Egg
Vanilla	Egg	Pepper	Butter	Milk
Milk	Butter	Flour	Parsley	Parsley
Parsley	Lemon juice	Lemon juice	Lemon juice	Vanilla
Lemon Juice	Vanilla	Vanilla	Vanilla	Lemon Juice

Fig. 6 Ranked recommendations to a user with an empty basket, using a 15 item dataset. In the first column, items are ranked according to their frequency. The remaining columns present the ranked recommendations provided by the model for ML-constrained and one-stage minimization, each under both the list loss and the item loss.

Subsequent recommendations are influenced by the items in the basket according to $\lambda_{a,b}$ or $\mu_a\hat{\mathbb{P}}(b|a)$. The normalized, fitted values of these variables are shown in Figure 7 for the list loss. The main features of the influence variables in Figure 7 are numbered. The different features in Figure 7 are rooted in the relative base scores (Figure 6) and the assumption from (4) that items are always added in the order in which they are recommended. We now describe each numbered feature individually:

Feature 1: There are two main clusters of ingredients that are commonly used together: a baking cluster (vanilla, flour, sugar, egg, milk, and butter), and a cooking cluster (onion, garlic, pepper, oil, bell pepper, parsley, and lemon juice). The frequency ranking in Figure 6 shows that cooking recipes are more popular than baking recipes.

Feature 2: The model variables associated with the cooking cluster are weak. This is because, as seen in Figure 6, cooking ingredients have higher base scores and will be chosen anyway.

Feature 3: The model variables associated with the baking cluster are much larger than those of the cooking cluster because baking ingredients must overcome the bias of lower base scores.

Feature 4: Lemon juice, parsley, and bell pepper have a negative influence on the other cooking ingredients (salt, onion, garlic, pepper, and oil). These three ingredients have the lowest base scores of the cooking ingredients and will be added to the basket only after all other cooking ingredients from the recipe have been added, so the negative influence variables often have no effect. Because these are also the least common of the ingredients, their corresponding variables are rarely used and are not fit accurately. (Recall that the size of the training set is 100 recipes.)

Feature 5: Baking ingredients predict the rare cooking ingredients (lemon juice, parsley, and bell pepper), although not as strongly as they predict other baking ingredients. Given one baking ingredient in the basket, there are likely to be more

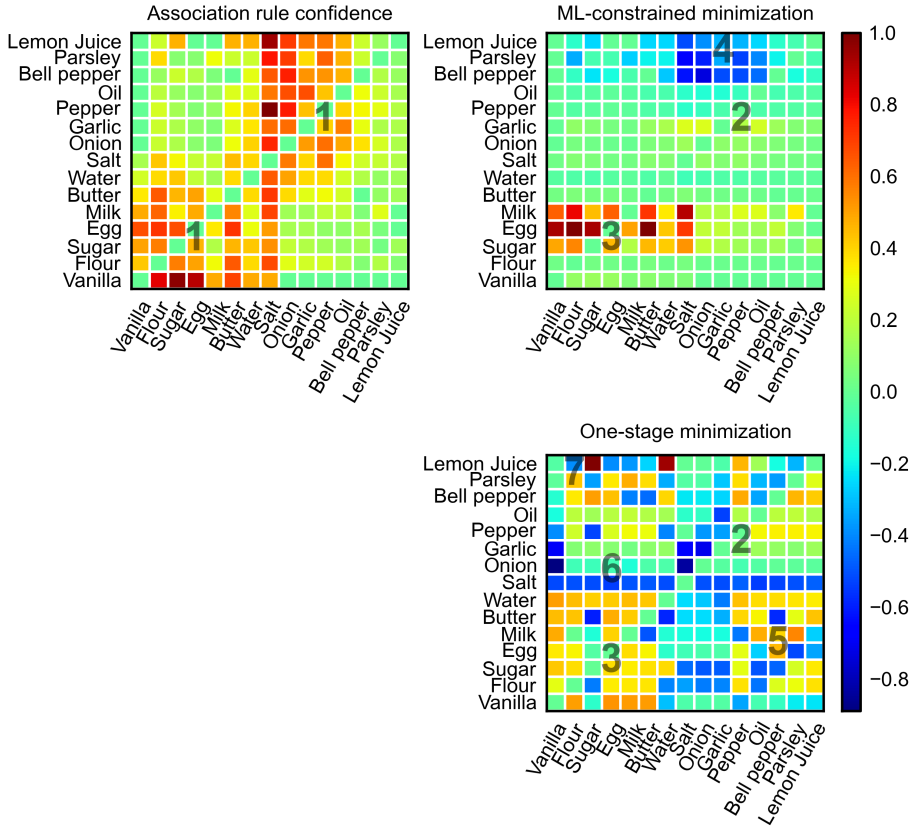

Fig. 7 Model influence variables for the online grocery store recommender system fit to a 15 item dataset using list loss. The element in row a column b is: $\text{Conf}(a \to b)$ for association rules; $\mu_a \hat{\mathbb{P}}(b|a)$ for ML-constrained minimization; and $\lambda_{a,b}$ for one-stage minimization. The large, overlayed numbers correspond to main features which are described in the text.

baking ingredients in the basket. Once all of the baking ingredients have been added to the basket, the rare cooking ingredients are the next most likely ingredients in a baking recipe. This feature cannot be present in the ML-constrained minimization variables because there each row must be proportional to the corresponding row in the association rule confidence matrix.

Feature 6: Salt is given a fairly constant, negative influence. From the association rule confidence matrix, it is clear that although salt is popular, it is a poor predictor of other ingredients. In recipes that contain salt, the influence of salt on other items is constant, thus salt is essentially not being used for subsequent predictions. The fact that the score is negative comes from the fitting and is arbitrary; the same effect holds no matter what the score of salt is, so long as it is constant across all items.

Feature 7: The very strong influence from lemon juice to sugar and water is probably spurious. Lemon juice is the rarest item and would usually have been added last. As a result, its influence variables were not used much in the fitting and were not reliably estimated. The model variables for Parsley and Bell Pepper are ques-

tionable for the same reason. This shows how the one-stage minimization model may be prone to overfitting on items that are rare in the training set.

The $t = 0$ recommendation list in Figure 6 demonstrates the connection between item loss and rank diversity discussed in Section 2.1.3. In this case, rank diversity entails having both a cooking ingredient and a baking ingredient high on the recommendation list. With item loss, the first recommendation is the most likely ingredient, salt, which is used in both baking and cooking. The next two ingredients are a cooking ingredient (garlic) and a baking ingredient (sugar), even though onion is ranked higher than sugar in frequency.

To study the test performance of the different methods, we did an experiment using the 250 most frequently occurring ingredients. This excluded only very rare ingredients that appeared in less than 5 recipes, for instance "alligator." Again, each recipe was treated as a shopping list containing certain items from $\mathcal{Z}$, the set of 250 ingredients.

We fit the model using one-stage minimization and ML-constrained minimization, using both list loss and item loss. Training and test sets each of 100 shopping lists were selected using random sub-sampling without replacement. The models were evaluated using the zero-one loss in (5) or (7). Training and test sets were sampled independently for 20 trials to provide a distribution of training and test losses for a total of (20 trials)$\times$(2 loss functions)$\times$(100 training lists + 100 test lists) = 8,000 shopping list evaluations for each minimization strategy. The results for Algorithm 1 (convex programming / simulated annealing) and Algorithm 2 (gradient descent) were very similar. We found that Algorithm 2 scaled better with the dimensionality of the dataset, so we report the results of Algorithm 2 here. The amount of ℓ_2-norm regularization in the loss function, β, was set using 3-fold cross validation on each training set, separately with $\beta = 0.0001, 0.001, 0.01, 0.1, 1$, and 10. We then set β to the value that minimized the mean error over the validation sets. With list loss and one-stage minimization, chosen values of β ranged from 0.001 to 0.1, with 0.001 chosen most frequently. With list loss and ML-constrained minimization, chosen values of β ranged from 0.01 to 1, with 0.1 chosen most frequently. With item loss and one-stage minimization, chosen values of β ranged from 0.0001 to 0.01, with 0.001 chosen most frequently. With item loss and ML-constrained minimization, chosen values of β ranged from 0.01 to 1, with 0.01 chosen most frequently. The training and test errors across the 20 trials are shown as boxplots in Figures 8 and 9 for list loss and item loss respectively.

As before, the test error presented in the figures excludes items that were not present in the training set, as these items necessarily cannot be well predicted using any algorithm. To provide some level of detail about this, the median list loss test error including items not encountered in the training set was 0.41 for max-confidence association rules, 0.34 for one-stage minimization, and 0.35 for ML-constrained minimization. The median item loss test error was 0.22 for max-confidence association rules, 0.18 for one-stage minimization, and 0.18 for ML-constrained minimization.

The large difference between training and test errors, particularly using item loss, suggests that that there is some overfitting despite the ℓ_2-norm regularization. This is not surprising given the number of possible items (250) and the number of shopping lists used for training (100). A larger training set would lead to better generalization (and less of an observable difference between the methods), although if it were desirable to fit a model individually to each customer

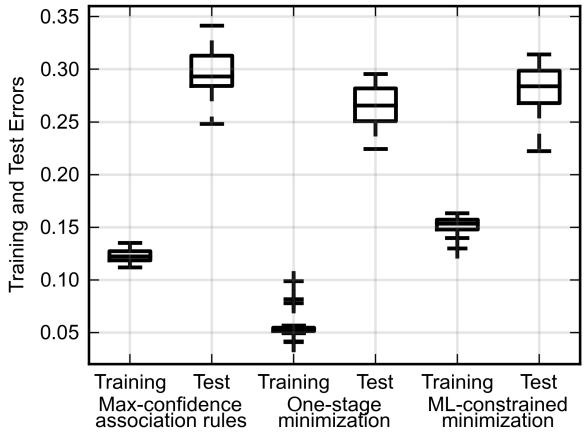

Fig. 8 List loss training and test errors for the online grocery store recommender system.

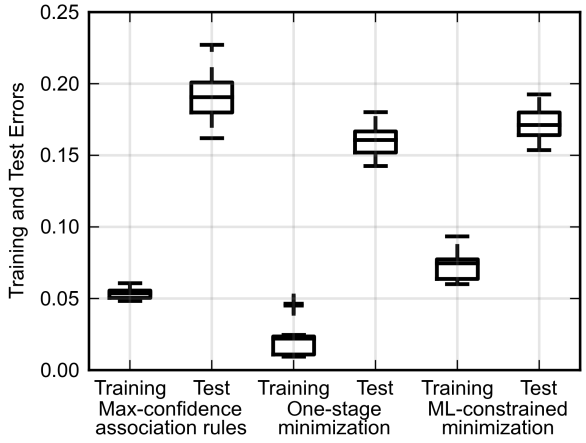

Fig. 9 Item loss training and test errors for the online grocery store recommender system.

the training data may truly be very limited. This is related to the "cold start" problem in recommender systems, when predictions need to be made when data are scarce.

For both loss functions, our method performs very well compared to the association rule baseline. One-stage minimization tends to perform slightly better, although it does so at a much higher computational cost. With this dataset, ML-constrained minimization solves an optimization problem on 500 variables whereas one-stage minimization requires solving an optimization problem on 62,500 variables. The best choice of minimization strategy depends on the application. One-stage minimization may be better for applications with few variables and plenty of computational power. ML-constrained minimization is probably a better choice for high-dimensional problems, or when the user is willing to sacrifice some accuracy for a significantly faster runtime.

3.3 Email recipient recommendation

For the email recipient recommendation experiment, we use the Enron email dataset (http://www.cs.cmu.edu/∼enron/), a collection of about 500,000 email messages from about 150 users. We limited our experiments to the "sent" folders of the 6 users who had more than 2000 emails in their "sent" folders and only considered emails with more than 2 recipients (using the "To" and "CC" fields), yielding a reduced dataset of 1845 emails with a total of 1200 unique recipients. The number of recipients per email ranged from 3 to 6. We formed a sequence of recipients by placing the sender in the first position in the sequence, and then appending the addresses in the "To" and "CC" fields in the order in which they appear in the email. The sender is always known in an email recommendation system, which is why we chose to predict the sequence starting from the sender being known.

We evaluated algorithm performance across 10 iterations, each iteration using randomly selected training and test sets of 500 emails each. For each iteration, we applied the FP-Growth algorithm (Borgelt, 2005) to the training set to find itemsets of size up to 4, with a minimum support requirement of 3 emails. These itemsets formed the allowed set $\mathcal{A}$ used for one-stage minimization, ML-constrained minimization, and max-confidence association rules. The median number of allowed itemsets across the 10 iterations was 625.5 (minimum 562, maximum 649), including the empty set. We used the training and test sets to evaluate the performance of one-stage minimization, ML-constrained minimization, max-confidence association rules, and cosine similarity item-based collaborative filtering. For one-stage minimization and ML-constrained minimization, we used the list loss in (6) and (14) respectively, and we set the amount of ℓ_2-norm regularization in the loss function, β, using 10-fold cross validation on each training set separately with $\beta = 0, 0.001, 0.01$, and 0.1. For both one-stage minimization and ML-constrained minimization, for all iterations, $\beta = 0$ minimized mean error over the validation sets and was chosen. The minimum support requirement when choosing the itemsets serves as a form of regularization, which may be why ℓ_2-norm regularization was not necessary. In Figure 10 we evaluated performance using the zero-one loss in (5). As before, the test error in Figure 10 excludes recipients who were not present in the training set as these cannot be predicted from the training data and form a constant bias. Including these recipients, the median test errors for cosine similarity item-based collaborative filtering, max-confidence association rules, one-stage minimization, and ML-constrained minimization were 0.218, 0.213, 0.201, and 0.203 respectively.

For this experiment, we additionally evaluated performance using the mean average precision. Mean average precision is a combination of precision and recall that is frequently used to evaluate ranking performance in information retrieval (Järvelin and Kekäläinen, 2000; Yue et al, 2007). The average precision of a ranked list is the average of the precision values computed at each of the relevant items. The average precision across many ranked lists is averaged to obtain the mean average precision. We measure average precision at each prediction (that is, each step in the sequence) and compute mean average precision by averaging over both time steps and sequences. We follow the procedure of McSherry and Najork (2008) to account for the presence of ties in the ranked lists. Figure 11 shows the mean average precision for each of the 10 iterations. Even though our methods were not

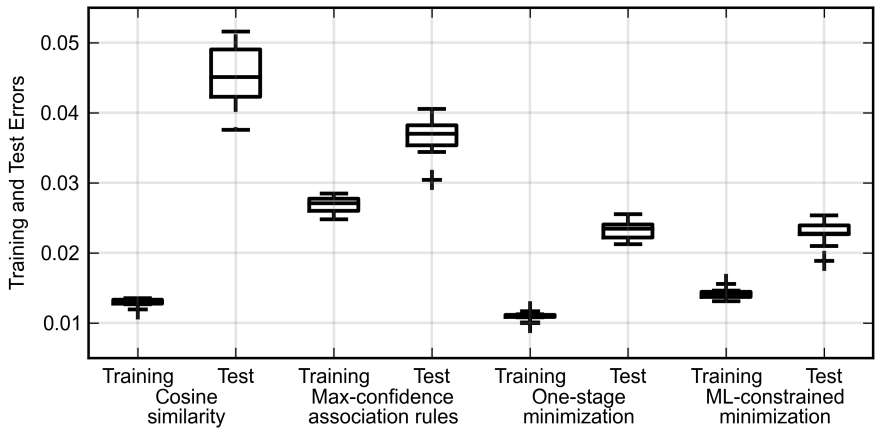

Fig. 10 Training and test errors for email recipient recommendation.

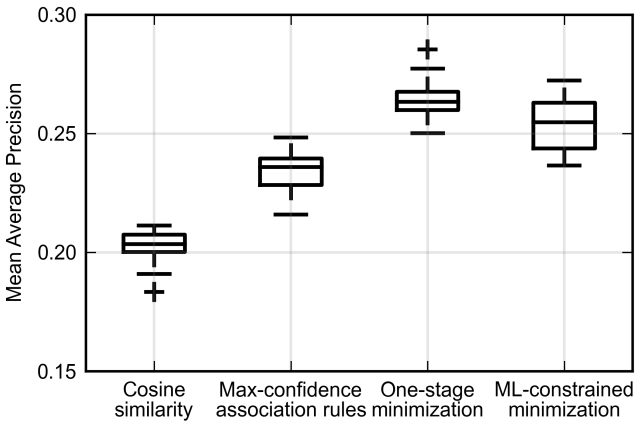

Fig. 11 Mean average precision for email recipient recommendation. Larger numbers indicate better performance.

optimized to maximize mean average precision, they perform well relative to both max confidence association rules and cosine similarity item-based collaborative filtering.

4 Related Work

This work is related to previous work on recommender systems, medical condition prediction, time-series modeling and supervised ranking.

There are many different approaches to recommender systems. Adomavicius and Tuzhilin (2005) give a review of current methods. Shani et al (2005) work with sequential recommendations using Markov decision processes, which differs from our approach in that our approach does not assume the Markov property.

Collaborative filtering methods have been especially common in recommender systems (see Sarwar et al, 2001, for a review). Some collaborative filtering methods rely on additional user information such as demographics and are not appropriate for our setting. Item-based collaborative filtering methods, cosine similarity in particular, are an extremely popular type of recommender system that are related to our approach as they consider only relations between various items in the sequence database (Sarwar et al, 2001; Linden et al, 2003). However, item-based collaborative filtering is generally not appropriate for these sequential prediction problems. Collaborative filtering algorithms are generally evaluated according to regression criteria (measuring accuracy in ratings) rather than ranking criteria, and is thus designed for a completely different type of learning framework. Also, when applying item-based collaborative filtering using the weighted sum method (Section 3.2.1 in Sarwar et al, 2001), we needed to compute an inner product of the similarities with the "ratings" for all co-rated items. However, for an incomplete basket, we do not have the ratings for all co-rated items, since there is no natural way to differentiate between items that have not yet been purchased in this transaction and items that will not be purchased in this transaction, as both have a "rating" of 0 at time t. Thus, the only ratings that are available are ratings of "1" indicating that an item is in the basket. In other words, our approach is intrinsically sequential, whereas it is unnatural to force item-based collaborative filtering into a sequential framework. Additionally, cosine similarity in particular is a symmetric measure ($sim(a, b) = sim(b, a)$) and thus not related to the conditional probability of b given a. In general, item-based collaborative filtering is not based in a machine learning framework, in that (20) and (21) are not based on either loss minimization (as our ranking algorithms are) or probabilistic modeling (as the association rule approach is). These differences help explain why in our email recipient recommendation experiment cosine similarity item-based collaborative filtering was outperformed by our methods, both in terms of our loss function and average precision.

Medical recommender systems are discussed by Davis et al (2008, 2010). The output of their system is a ranked list of conditions that are likely to be subsequently experienced by a patient, similar to the ranked recommendation lists that we produce. Their system is based on collaborative filtering rather than bipartite ranking loss which is the core of our method. Duan et al (2011) develop a clinical recommender system which uses patient conditions to predict suitable treatment plans. Much of the work in medical data mining uses explanatory modeling (*e.g.*, finding links between conditions), which is fundamentally different from predictive modeling (Shmueli, 2010). Most work in medical condition prediction focuses on specific diseases or data sets (see Davis et al (2010) for a literature review). Email recipient recommendation has been studied with several approaches, often incorporating the email content using language models, or finding clusters in the network of corresponding individuals (Dom et al, 2003; Pal and McCallum, 2006; Balasubramanyan et al, 2008; Carvalho and Cohen, 2008; Roth et al, 2010).

A large body of research on time series modeling dates back at least to the 1960's and provides many approaches for sequential prediction problems. Recent applications to medicine in general and patient level prediction in particular include Enright et al (2011), Stahl and Johansson (2009), and Hu et al (2010). Our ML-constrained minimization strategy was motivated by the mixture transition distribution developed by Berchtold and Raftery (2002) to model high-order

Markov chains. However, as we discussed earlier, typical time-series approaches focus specifically on the *order* of past events whereas in our applications the historical order seems of peripheral importance.

Our model and fitting procedure derive from previous work on supervised ranking. Many approaches to ranking have been proposed, including methods based on classification algorithms (Herbrich et al, 1999; Chapelle and Keerthi, 2010; Joachims, 2002; Freund et al, 2003; Burges et al, 2005), margin maximization (Shashua and Levin, 2002; Yan and Hauptmann, 2006), order statistics (Lebanon and Lafferty, 2002; Clémençon and Vayatis, 2008), and others (Cao et al, 2007; Rudin, 2009). The loss functions that we use derive from the bipartite misranking error, and the exponential upper bound is that used in boosting. Our list loss is in fact exactly the misranking error; thus minimizing list loss corresponds to maximizing the area under the ROC curve (Freund et al, 2003). Other loss functions can be substituted as is appropriate for the problem at hand, for example our item loss is a good fit for problems where only one relevant item needs to be at the top. Minimizing misranking error does not imply optimizing other evaluation metrics, such as average precision and discounted cumulative gain as illustrated in Yue et al (2007) and Bertsimas et al (2011). Our formulation could potentially be adapted to optimize other evaulation metrics, as is done in Yue et al (2007) and Bertsimas et al (2011), if these metrics are the quantity of interest. The theoretical framework underpinning ranking includes work in statistics, learning theory, and computational complexity (Cohen et al, 1999; Freund et al, 2003; Clémençon et al, 2008; Cossock and Zhang, 2008; Rudin and Schapire, 2009). Our work is also related to the growing fields of preference learning and label ranking (Fürnkranz and Hüllermeier, 2003; Hüllermeier et al, 2008; Dekel et al, 2004; Shalev-Shwartz and Singer, 2006).

5 Conclusions

We have presented a supervised ranking framework for sequential event prediction that can be adapted to fit a wide range of applications. We provided formulations for an online grocery store recommender system (unordered sequences of events), and for medical condition prediction (ordered sets of events). We showed how the online grocery store recommender system formulation could be directly applied to email recipient recommendation (ordered sequence of events). The same methods could be also be extended to other applications, such as unordered sequences of sets. In the recommender system setup, the predictions alter the sequence of events and the loss function is not continuous. Using the fact that the variable space can be partitioned into convex sets over which the loss function is convex, we presented two algorithms for approximately minimizing the loss. For applications in both the recommender system domain and the medical condition prediction domain, the supervised ranking models performed well in experiments, and better than the max-confidence and cosine similarity baselines.

There are many other applications where the *set* of past events matters for predicting the future, rather than the order of past events. Our optimization-based methodology is a direct and practical approach for prediction tasks with this property.

References

Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17(6):734–749

Agichtein E, Brill E, Dumais S (2006a) Improving web search ranking by incorporating user behavior information. In: Proceedings of the ACM Conference on Research and Development on Information Retrieval (SIGIR)

Agichtein E, Brill E, Dumais S, Ragno R (2006b) Learning user interaction models for predicting web search result preferences. In: Proceedings of the ACM Conference on Research and Development on Information Retrieval (SIGIR)

Balasubramanyan R, Carvalho VR, Cohen W (2008) Cutonce - recipient recommendation and leak detection in action. In: The AAAI 2008 Workshop on Enhanced Messaging

Berchtold A, Raftery AE (2002) The mixture transition distribution model for high-order markov chains and non-gaussian time series. Statistical Science 17(3):328–356

Bertsekas DP (1995) Nonlinear Programming. Athena Scientific, Belmont, Massachusetts

Bertsimas D, Chang A, Rudin C (2011) Integer optimization methods for supervised ranking. Operations Research Center Working Paper Series OR 388-11, MIT

Bigal ME, Liberman JN, Lipton RB (2006) Obesity and migraine. Neurology 66(4):545–550

Borgelt C (2005) An implementation of the fp-growth algorithm. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, OSDM '05, pp 1–5

Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010), pp 177–187

Burges C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender G (2005) Learning to rank using gradient descent. In: Proceedings of the 22nd International Conference on Machine Learning (ICML), pp 89–96

Byrd RH, Lu P, Nocedal J, Zhu C (1995) A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing 16(5):1190–1208

Cao Z, Qin T, Liu TY, Tsai MF, Li H (2007) Learning to rank: from pairwise approach to listwise approach. In: Proceedings of the 24th International Conference on Machine Learning (ICML)

Carvalho VR, Cohen WW (2008) Ranking users for intelligent message addressing. In: Proceedings of the IR research, 30th European conference on Advances in information retrieval, ECIR'08, pp 321–333

Chapelle O, Keerthi SS (2010) Efficient algorithms for ranking with SVMs. Information Retrieval 13(3):201–215

Clémençon S, Vayatis N (2008) Empirical performance maximization for linear rank statistics. In: Advances in Neural Information Processing Systems 22

Clémençon S, Lugosi G, Vayatis N (2008) Ranking and empirical minimization of U-statistics. Annals of Statistics 36(2):844–874

Cohen WW, Schapire RE, Singer Y (1999) Learning to order things. Journal of Artificial Intelligence 10:243–270

Cossock D, Zhang T (2008) Statistical analysis of bayes optimal subset ranking. IEEE Transactions on Information Theory 54(11):5140–5154

Davis DA, Chawla NV, Blumm N, Christakis N, Barabasi AL (2008) Predicting individual disease risk based on medical history. In: Proceedings of the ACM Conference on Information and Knowledge Management

Davis DA, Chawla NV, Christakis NA, Barabasi AL (2010) Time to CARE: a collaborative engine for practical disease prediction. Data Mining and Knowledge Discovery 20:388–415

Dekel O, Manning CD, Singer Y (2004) Log-linear models for label ranking. In: Proceedings of NIPS 2004

Dom B, Eiron I, Cozzi A, Zhang Y (2003) Graph-based ranking algorithms for e-mail expertise analysis. In: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD '03, pp 42–48

Duan L, Street W, Xu E (2011) Healthcare information systems: data mining methods in the creation of a clinical recommender system. Enterprise Information Systems 5(2):169–181

Enright C, Madden M, Madden N, Laffey J (2011) Clinical time series data analysis using mathematical models and dbns. In: Peleg M, Lavrac N, Combi C (eds) Artificial Intelligence in Medicine, Lecture Notes in Computer Science, vol 6747, Springer Berlin / Heidelberg, pp 159–168

Freund Y, Iyer R, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research 4:933–969

Fürnkranz J, Hüllermeier E (2003) Pairwise preference learning and ranking. In: Machine Learning: ECML 2003, vol 2837/2003, pp 145–156

Herbrich R, Graepel T, Obermayer K (1999) Support vector learning for ordinal regression. In: Proceedings of ICANN 1999, pp 97–102

Hu Q, Huang X, Melek W, Kurian C (2010) A time series based method for analyzing and predicting personalized medical data. In: Yao Y, Sun R, Poggio T, Liu J, Zhong N, Huang J (eds) Brain Informatics, Lecture Notes in Computer Science, vol 6334, Springer Berlin / Heidelberg, pp 288–298

Hüllermeier E, Fürnkranz J, Cheng W, Brinker K (2008) Label ranking by learning pairwise preferences. Artificial Intelligence 172(16–17):1897–1916

ICCBR (2011) International Conference on Case-based Reasoning (ICCBR) Computer cooking contest recipe book. URL http://liris.cnrs.fr/ccc/ccc2011/

Järvelin K, Kekäläinen J (2000) Ir evaluation methods for retrieving highly relevant documents. In: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '00, pp 41–48

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)

Lebanon G, Lafferty J (2002) Cranking: Combining rankings using conditional probability models on permutations. In: Proceedings of the 19th International Conference on Machine Learning (ICML)

Linden G, Smith B, York J (2003) Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing 7(1):76–80

McCormick T, Rudin C, Madigan D (2011) A hierarchical model for association rule mining of sequential events: An approach to automated medical symptom prediction. SSRN eLibrary URL http://ssrn.com/paper=1736062

McSherry F, Najork M (2008) Computing information retrieval performance measures efficiently in the presence of tied scores. In: Proceedings of the IR research, 30th European conference on Advances in information retrieval, ECIR'08, pp 414–421

Pal C, McCallum A (2006) Cc prediction with graphical models. In: Proceedings of the third conference on email and anti-spam, CEAS

Radlinski F, Kleinberg R, Joachims T (2008) Learning diverse rankings with multi-armed bandits. In: Proceedings of the 25th International Conference on Machine Learning (ICML)

Rivest RL (1987) Learning decision lists. Machine Learning 2(3):229–246

Roth M, Ben-David A, Deutscher D, Flysher G, Horn I, Leichtberg A, Leiser N, Matias Y, Merom R (2010) Suggesting friends using the implicit social graph. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '10, pp 233–242

Rudin C (2009) The P-norm Push: A simple convex ranking algorithm that concentrates at the top of the list. Journal of Machine Learning Research 10:2233–2271

Rudin C, Schapire RE (2009) Margin-based ranking and an equivalence between AdaBoost and RankBoost. Journal of Machine Learning Research 10:2193–2232

Rudin C, Letham B, Salleb-Aouissi A, Kogan E, Madigan D (2011) Sequential event prediction with association rules. In: Proceedings of the 24th Annual Conference on Learning Theory (COLT)

Rudin C, Letham B, Kogan E, Madigan D (2012) A learning theory framework for association rules and sequential events, In preparation

Sarwar B, Karypis G, Konstan J, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th international conference on World Wide Web, WWW '01, pp 285–295

Senecal S, Nantel J (2004) The influence of online product recommendations on consumers' online choices. Journal of Retailing 80:159–169

Shalev-Shwartz S, Singer Y (2006) Efficient learning of label ranking by soft projections onto polyhedra. Journal of Machine Learning Research 7:1567–1599

Shani G, Heckerman D, Brafman RI (2005) An MDP-based recommender system. Journal of Machine Learning Research 6:1265–1295

Shashua A, Levin A (2002) Taxonomy of large margin principle algorithms for ordinal regression problems. In: Proceedings of NIPS 2002

Shmueli G (2010) To explain or to predict? Statistical Science 25(3):289–310

Stahl F, Johansson R (2009) Diabetes mellitus modeling and short-term prediction based on blood glucose measurements. Mathematical Biosciences 217(2):101 – 117

Yan R, Hauptmann AG (2006) Efficient margin-based rank learning algorithms for information retrieval. In: International Conference on Image and Video Retrieval (CIVR)

Yue Y, Finley T, Radlinski F, Joachims T (2007) A support vector method for optimizing average precision. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '07, pp 271–278

Zhu C, Byrd RH, Lu P, Nocedal J (1997) Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software 23(4):550–560

A Regression and Sequential Event Prediction

When using our model or association rules for sequential event prediction, we assume that, at each time step, only the *set* of items in the basket is useful for the prediction and not the order in which they were added. In this section we will discuss a natural regression approach that does not make this assumption.

Let X_i be an indicator variable that is 1 if item i is in the current basket and 0 otherwise. Suppose we wish to apply regression (e.g., logistic regression) to create a model for each item separately. Consider the model for the last item (item m), where the predictor variables will be X_i for $i \in \{1, \ldots, m-1\}$, and X_m will be the response variable. This model would provide:

$$P(X_m = 1 | X_1 = x_1, \ldots, X_{m-1} = x_{m-1}) = \frac{1}{1 + \exp(f)},$$

where $f = \sum_{i=1}^{m-1} \lambda_i x_i + \lambda_{0,m}$, with each $x_i \in \{0, 1\}$.

Because the data are being revealed sequentially, the correct application of this technique is not straightforward. Only a *partial* basket is available when predictions need to be made. It is incorrect to substitute the current state of the basket directly into the formula above. For instance, if the current basket contains items 1 and 2, so $X_1 = 1$ and $X_2 = 1$, it is incorrect to write $P(X_m | X_1 = 1, X_2 = 1) = \frac{1}{1+\exp(f)}$, where $f = \lambda_1 + \lambda_2 + \lambda_{0,m}$. This statement would be equivalent to the expression:

$$P(X_m = 1 | X_1 = 1, X_2 = 1) = P(X_m = 1 | X_1 = 1, X_2 = 1, X_3 = 0, \ldots, X_{m-1} = 0),$$

which is clearly false in general.

On the other hand, it is possible to integrate in order to obtain conditional probability estimates:

$$P(X_m = 1 | X_1 = 1, X_2 = 1) = \sum_{x_3 = \{0,1\}, \ldots, x_{m-1} = \{0,1\}} P(X_m = 1 | X_1 = 1, X_2 = 1, X_3 = x_3 \ldots, X_{m-1} = x_m) \times P(X_3 = x_3, \ldots, X_{m-1} = x_m),$$

where estimates of $P(X_3 = x_3, \ldots, X_{m-1} = x_m)$ would need to be made also for every one of the 2^{m-3} combinations of $x_3, \ldots x_{m-1}$. Thus, this approach would rely on a large number of uncertain estimates (given limited data, and even moderately large m), each introducing errors into the final estimate.

The difficulties in using regression methods for sequential event prediction are discussed in further detail in Rudin et al (2012).