Mach Learn (2013) 93:357–380
DOI 10.1007/s10994-013-5356-5

Sequential event prediction

Benjamin Letham · Cynthia Rudin · David Madigan

Received: 8 November 2011 / Accepted: 11 April 2013 / Published online: 8 June 2013

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 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 events, but

Editors: Eyke Hüllermeier and Johannes Fürnkranz.

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

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. 2012). 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 (ERM). We apply our algorithms to data from three applications: an online grocery store recommender system, email recipient recommendation, 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, 2006b). 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.

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. 2012). 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 handle 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. For instance, the online grocery store recommender system problem has no intrinsic order in which groceries should be added to the basket, and in email recipient recommendation the order of the addresses is likely of little importance. Only the *set* of past items are useful for predicting the remaining sequence. Figure 1 gives an illustration of this point using the online grocery store recommender system. 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

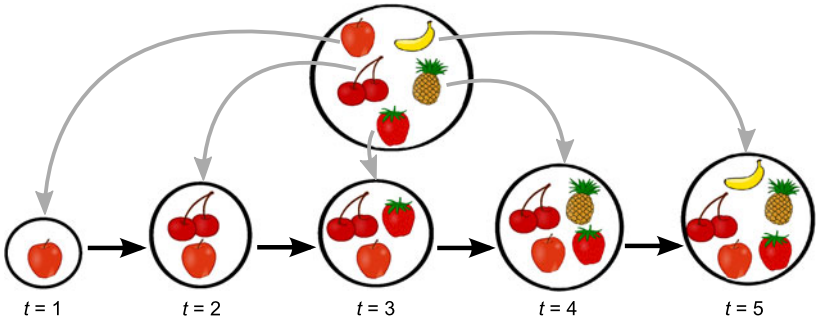

Fig. 1 An illustration of the online grocery store recommender system, in which items from an unordered shopping list are sequentially added to the user's basket. At each time step, the set of items in the basket is used to predict future items that will be added

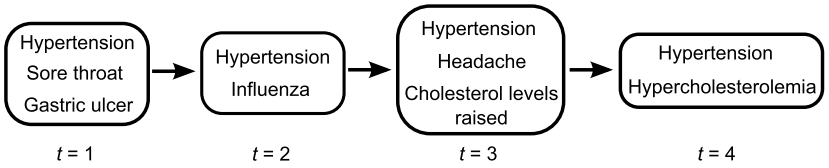

Fig. 2 An illustration of the medical condition prediction problem, in which collections of medical conditions occur at various time steps. At each time step, we use past collections of conditions to predict conditions that will subsequently be presented

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 $t = 3$ to say that elevated cholesterol preceded Headache. On the surface, the online grocery store recommender system and the medical condition prediction problem seem quite different, but the methodology we develop for each problem derives from a general formulation which 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 gives a formal introduction to sequential event prediction and provides the notation that we will use throughout the paper. Section 3 presents our ERM-based method for sequential event prediction. In Sect. 4 we apply the ERM-based algorithms to email recipient recommendation, in which the sequence is one of email addresses. In Sect. 5 we study patient condition prediction, and the sequences are of sets of medical conditions. The third and final application is in Sect. 6, where we apply our methods to an online grocery store recommender system. In that application we allow the recommendations to influence the order of the sequence, and provide algorithms for performing ERM. Our approach synthe-

sizes 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 Sequential event prediction

We begin by presenting the formal framework and notation of sequential event prediction problems, and discussing previously developed algorithms for sequential event prediction based on association rules.

We suppose that we have access to a collection of m sequences, which in our applications would be m visits from a grocery store customer, m emails, or m patient histories. The items in the sequence (e.g., grocery items, email addresses, or medical conditions) come from a library of N items, $\mathcal{Z}$ being the set of these items. Sequence i in the sequence database (e.g., visit i, email i, or patient i) includes a total of T_i time steps, with items (or sets of items) occurring in the sequence at time steps $t = 1, \ldots, T_i$. The item (or set of items) added to the sequence at time t is denoted $z_{i,t}$. As illustrated in Fig. 2, each step in the sequence may be a set of items and thus we consider $z_{i,t}$ to be a set in general. In many applications, such as the online grocery store recommender system and email recipient recommendation, $z_{i,t}$ will contain only one item. The observed part of the sequence at time t is denoted $x_{i,t} = \{z_{i,j}\}_{j=1,\ldots,t}$. The full sequence, x_{i,T_i}, is denoted X_i. We denote the collection of m training sequences as X_1^m.

It is not clear how to adapt standard modeling techniques (e.g., logistic regression) to sequential event prediction problems because they estimate full probabilities rather than partial probabilities. The difficulties in using regression for sequential event prediction are discussed in detail in Rudin et al. (2012), where we propose algorithms for sequential event prediction based on association rules.

Association rules have the advantage of being able to model the conditional probabilities directly. In this context, an association rule is a rule "$a \rightarrow b$," meaning that itemset a in the sequence implies item b is also in the sequence. We define the *confidence* of rule "$a \rightarrow b$" to be the proportion of training sequences with itemset a that also have item b: $\text{Conf}(a \rightarrow b) = \hat{\mathbb{P}}(b|a) = \frac{\#(a \text{ and } b)}{\#a}$. 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 observed portion of the sequence and right-hand side b a potential future item in the sequence. (2) For each right-hand side b, find the rule with the maximum confidence. (3) Rank the right-hand sides (potential future items in the sequence) in order of descending confidence, and use this ranked list for predictions. This is the "max-confidence" algorithm, used throughout the association rule literature and applied to sequential event prediction by Rudin et al. (2012).

In this work, we develop a framework for using ERM techniques in sequential event prediction. The ERM-based algorithms give increased flexibility over association rule algorithms by allowing the loss function to be tailored to the requirements of the specific application, and the ERM learning procedure leads to better predictions.

3 Empirical risk minimization for sequential event prediction

We present a general framework for using ERM in sequential event prediction, and then show how the framework can be specified to specific applications by presenting email recip-

ient recommendation, the online grocery store recommender system, and medical condition prediction as case studies.

The core of our ERM-based approach to sequential event prediction is a ranking model of the relationship between items in the observed part of the sequence and potential future items. The ranking model is a scoring function $f(x_{i,t}, a)$ that, given the observed part of the sequence $x_{i,t}$, scores each item $a \in \mathcal{Z}$ according to the predicted likelihood that it is a future item in the sequence. Ideally we would like $f(x_{i,t}, a)$ to be related to $\mathbb{P}(a|x_{i,t})$, the conditional probability of item a being in the sequence given that the items in $x_{i,t}$ are in the sequence. 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. We present here two possible scoring models, which we call the *one-stage model* and the *ML-constrained model*.

3.1 The one-stage model

Our first scoring model 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 be in the sequence, for each itemset-item pair that we are willing to consider. 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 $b \in \mathcal{Z}$. We require $\varnothing \in \mathcal{A}$ so that every (partial) sequence contains at least one itemset from $\mathcal{A}$. If itemset a and item b are likely to be present in the sequence 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 generally found in the same sequence. The influences of the itemsets in the observed part of the sequence are combined linearly to yield the score for a given item. For example, suppose the observed sequence is $x_{i,t} = \{a_1, a_2\}$ and $\mathcal{A} = \varnothing \cup \mathcal{Z}$. In other words, $\mathcal{A}$ includes the empty itemset and all itemsets consisting of a single item. 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 observed sequence $x_{i,t}$, the score of item b is:

$$f(x_{i,t}, b; \boldsymbol{\lambda}) := \lambda_{\varnothing,b} + \sum_{j=1}^{t} \sum_{\substack{a \subseteq z_{i,j} \\ a \in \mathcal{A} \setminus \varnothing}} \lambda_{a,b}. \tag{1}$$

In applications such as medical condition prediction, events in the sequence that are far in the past may be less influential than recent events. The effect of time can be incorporated into (1) by inserting a weighting factor before the inner sum that is inversely proportional to the elapsed time since sequence step j. In some applications, it may be important to capture nonlinear effects of combinations of items. Feature variables for those combinations of items, such as $\lambda_{\{a \text{ and } b\},c}$, allow the model to express more complex inter-item relationships while maintaining the computational benefits of a linear model.

We call this the one-stage model because all of the variables $\boldsymbol{\lambda}$ are fit simultaneously in a single optimization problem. The model uses a total of $|\mathcal{A}|N$ variables: $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{A}|N}$. A straightforward implementation is to take $\mathcal{A}$ as itemsets of size less than or equal to 1, which is $\mathcal{A} = \varnothing \cup \mathcal{Z}$. 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 sequence. 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.

3.2 The ML-constrained model

Our second model, the ML-constrained model, reduces the dimensionality by, for every non-empty itemset a, forcing each $\lambda_{a,b}$ to be proportional to $\hat{\mathbb{P}}(b|a)$, the maximum likelihood (ML) estimate of the conditional probability of having item b in the sequence given that itemset a is in the sequence. Specifically, we set

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

where μ_a is a free variable that does not depend on b. $\hat{\mathbb{P}}(b|a)$ is estimated directly from the training data, prior to any optimization for model fitting, as described in Sect. 2. Then, the ML-constrained model is:

$$f_{\mathrm{ML}}(x_{i,t}, b; \boldsymbol{\lambda}_{\varnothing,b}, \boldsymbol{\mu}) := \boldsymbol{\lambda}_{\varnothing,b} + \sum_{j=1}^{t} \sum_{\substack{a \subseteq z_{i,j} \\ a \in \mathcal{A} \setminus \varnothing}} \mu_a \hat{\mathbb{P}}(b|a). \tag{2}$$

To use this strategy, we first compute the ML estimates of the conditional probabilities. Then the N base scores $\lambda_{\varnothing,b}$ and the $|\mathcal{A}|$ proportionality coefficients μ_a are fit during ERM, 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.

3.3 The general loss function

We use the training set and the ERM principle to fit vector $\boldsymbol{\lambda}$. For the sequence i at time step t, we define a set of items $L_{i,t} \subset \mathcal{Z}$ that should be ranked strictly higher than some other set of items $K_{i,t} \subset \mathcal{Z}$. For instance, $L_{i,t}$ might be the remaining items in the sequence and $K_{i,t}$ might be items not in the sequence. 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 $K_{i,t}$ appropriate for our specific applications. The most general loss function, evaluated on the training set of m sequences, is:

$$R_{0-1}\left(f, X_1^m; \boldsymbol{\lambda}\right) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i - 1} \frac{1}{T_i} \frac{1}{|K_{i,t}|} \frac{1}{|L_{i,t}|} \sum_{l \in L_{i,t}} \sum_{k \in K_{i,t}} \mathbb{1}_{[f(x_{i,t},k;\boldsymbol{\lambda}) \geq f(x_{i,t},l;\boldsymbol{\lambda})]}. \tag{3}$$

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

$$\begin{aligned} &R_{\exp}\left(f, X_1^m; \boldsymbol{\lambda}\right) \\ &\quad := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T_i - 1} \frac{1}{T_i} \frac{1}{|K_{i,t}|} \frac{1}{|L_{i,t}|} \sum_{l \in L_{i,t}} \sum_{k \in K_{i,t}} e^{f(x_{i,t},k;\boldsymbol{\lambda}) - f(x_{i,t},l;\boldsymbol{\lambda})} + \beta \|\boldsymbol{\lambda}\|_2^2, \end{aligned} \tag{4}$$

where β is a parameter that determines the amount of regularization. Minimizing the loss function in (4) will produce model parameters $\boldsymbol{\lambda}$ that make accurate predictions across the sequence. Although we expressed these loss functions using f and $\boldsymbol{\lambda}$ as with the one-stage model, they apply directly to the ML-constrained model f_{ML} and its parameters $\boldsymbol{\lambda}_{\varnothing}$ and $\boldsymbol{\mu}$.

3.4 Scalability

In most cases, such as our email recipient recommendation and patient condition prediction examples, the loss function $R_{\exp}$ in (4) is convex in $\boldsymbol{\lambda}$ and thus fitting the scoring model to data requires only convex minimization in $|\mathcal{A}|N$ variables for the one-stage model, or $|\mathcal{A}| + N$ variables for the ML-constrained model. There are a number of efficient algorithms for convex minimization whose scalability has been addressed (Bertsekas 1995). Our ERM-based algorithms inherit the scalability of whichever convex minimization algorithm is used for model fitting, subject to the dimensionality of the chosen model. Our examples show that the ERM-based algorithms can be applied to real datasets with thousands of sequences and millions of model variables. In our online grocery store recommendation example, we consider a situation where the sequence order depends directly on the recommendations. In this case the loss function is not convex, however we present algorithms based on convex programming and gradient descent. Variants of gradient descent, particularly stochastic gradient descent, are known to have excellent scalability properties in large-scale learning problems (Bottou 2010).

3.5 Baseline algorithms

In our experiments we compare the performance of our ERM-based algorithms to two baselines: the max-confidence association rule algorithm described in Sect. 2 and an item-based collaborative filtering algorithm. 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 $R_{i,b}$ to item b. To adapt it to sequential recommendations, we let $R_{i,b} = 1$ if sequence i contains item b, and 0 otherwise. For each item b, we construct the binary "ratings" vector $\mathbf{R}_b = [R_{1,b}, \ldots, R_{m,b}]$ and then compute the cosine similarity between every pair of items a and b:

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

For each item b, we define the neighborhood of b, $\text{Nbhd}(b; k)$, as the k most similar items to b. To make a prediction from a partial sequence $x_{i,t}$, we score each item b by adding the similarities for all of the observed items in the sequence that are also in the neighborhood of b, and normalizing:

$$f_{sim}(x_{i,t}, b; k) := \frac{\sum_{a \in \bigcup_{j=1}^{t} z_{i,j} \cap \text{Nbhd}(b;k)} sim(a, b)}{\sum_{a \in \text{Nbhd}(b;k)} sim(a, b)}. \tag{5}$$

In Sect. 7, 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 our experiments. In our experiments, we used neighborhood sizes of 20, 40, and all items (Sarwar et al. 2001; Herlocker et al. 1999). Any ties when determining the top k most similar items were broken randomly.

4 Application 1: email recipient recommendation

In this application we study the sequence in which recipients are added to an email. Given a partial set of recipients, the goal is to predict the remaining recipients. In this application, each item in the sequence, $z_{i,t}$, is a single email address. An email recipient recommender

system knows who the sender of the email is, thus we initialize the sequence by setting $z_{i,0}$ as the address of the sender of email i. We then construct the rest of the sequence using the T_i addresses placed in the "To:" and "CC:" fields, in the order that they appear in the email.

To apply our ERM-based algorithms to this application, we must specify the sets $L_{i,t}$ and $K_{i,t}$ used in the loss function. A natural goal for this problem setting is, at each time step t, to attempt to rank all of the actual recipients that have not yet been added higher than all of the non-recipients. This goal is expressed by taking

$$L_{i,t} = \bigcup_{j=t+1}^{T_i} z_{i,j}$$

$$K_{i,t} = \mathcal{Z} \setminus \bigcup_{j=0}^{T_i} z_{i,j}.$$

We call this the *list loss*, as it tries to put the entire set of remaining recipients at the top of the recommendation list. For notational convenience we define $Z_i := \bigcup_{j=0}^{T_i} z_{i,j}$ to be the complete collection of email addresses in the sequence. The general loss function in (3) can then be rewritten

$$R_{0-1}^{\text{list}}\big(f, X_1^m; \boldsymbol{\lambda}\big) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\frac{1}{T_i(N-T_i)(T_i-t)}\sum_{l=t+1}^{T_i}\sum_{k\in\mathcal{Z}\setminus Z_i}\mathbb{1}_{[f(x_{i,t},k;\boldsymbol{\lambda})\geq f(x_{i,t},z_{i,l};\boldsymbol{\lambda})]} \tag{6}$$

and (4) then becomes:

$$R_{\text{exp}}^{\text{list}}\big(f, X_1^m; \boldsymbol{\lambda}\big) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\frac{1}{T_i(N-T_i)(T_i-t)}\sum_{l=t+1}^{T_i}\sum_{k\in\mathcal{Z}\setminus Z_i} e^{f(x_{i,t},k;\boldsymbol{\lambda})-f(x_{i,t},z_{i,l};\boldsymbol{\lambda})} + \beta\|\boldsymbol{\lambda}\|_2^2. \tag{7}$$

We applied our algorithm to the Enron email dataset, a collection of about 500,000 email messages from about 150 users (http://www.cs.cmu.edu/~enron/). 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, 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 evaluated algorithm performance across 10 iterations, each iteration using randomly selected training and test sets of 500 emails each. For each iteration, we chose the allowed itemsets (features) $\mathcal{A}$ by applying the FP-growth algorithm (Borgelt 2005), a frequent itemset mining algorithm, to the training set. We mined itemsets of size up to 4, with a minimum support requirement of 3 emails. The median number of allowed itemsets across the 10 iterations was 625.5 (minimum 562, maximum 649), including the empty set. Thus the median number of variables in the one-stage model was 750,600 ($\mathcal{A}N$) and the median number of variables in the ML-constrained model was 1,825.5 ($\mathcal{A}+N$).

We used the training and test sets to evaluate the performance of the one-stage model, ML-constrained model, max-confidence association rules, and cosine similarity item-based collaborative filtering methods. For our ERM-based algorithms, we found $\boldsymbol{\lambda}$ (or, $\boldsymbol{\lambda}_\varnothing$ and $\boldsymbol{\mu}$ for the ML-constrained model) that minimized (7) on the training set using L-BFGS-B, the limited memory implementation of the Broyden-Fletcher-Goldfarb-Shanno algorithm (Byrd et al. 1995; Zhu et al. 1997). We set the amount of ℓ_2-norm regularization, β, using 10-fold

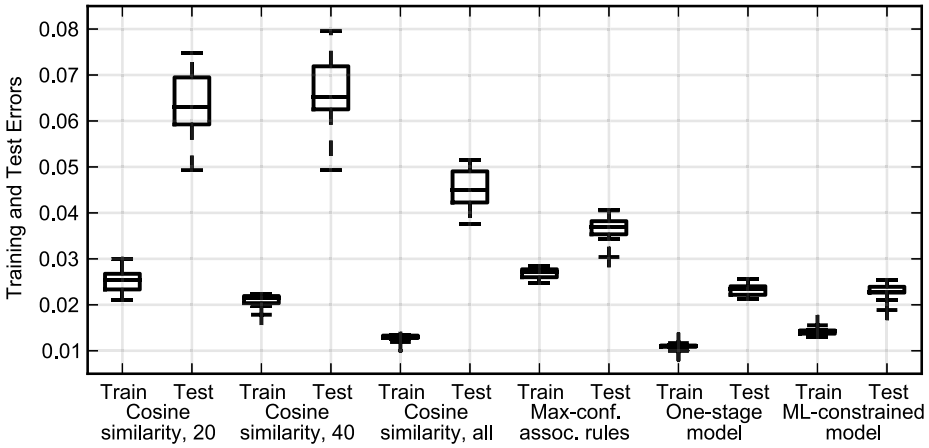

Fig. 3 Training and test errors for email recipient recommendation

cross validation on each training set separately with $\beta = 0, 0.001, 0.01$, and 0.1. For both the one-stage model and the ML-constrained model, 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 Fig. 3 we evaluated performance using the zero-one loss in (6). When evaluating the test error in Fig. 3, we excluded email addresses that were not encountered in the training set because these recipients were impossible to predict and resulted in a constant error for all methods. The results show that our ERM-based algorithms performed very well compared to the baseline algorithms. Cosine similarity, at all neighborhood sizes, had a tendency to overfit the data, with much higher test error than training error. The performance of the ML-constrained model was very close to that of the one-stage model, despite using many fewer variables.

We additionally evaluated performance using 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 measured average precision at each prediction (that is, each step in the sequence) and computed mean average precision by averaging over both time steps and sequences. We followed the procedure of McSherry and Najork (2008) to account for the presence of ties in the ranked lists. Figure 4 shows the mean average precision for each of the 10 iterations. Even though our methods were not optimized to maximize mean average precision, they performed well relative to both max confidence association rules and cosine similarity item-based collaborative filtering (shown in the figure only for the "all items" neighborhood, which was the best performing neighborhood size).

5 Application 2: patient condition prediction

Here we tailor the 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 medi-

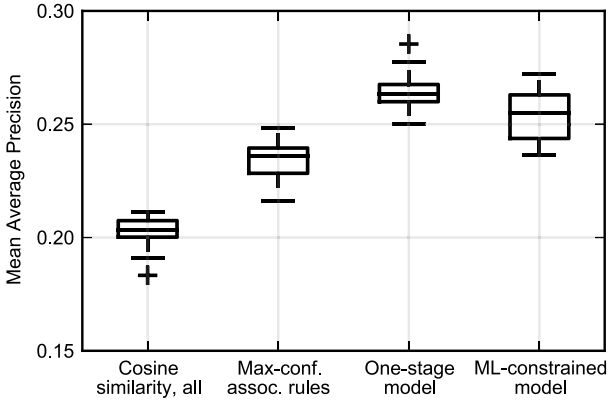

Fig. 4 Mean average precision for email recipient recommendation. *Larger numbers* indicate better performance

cal 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.

In this application, each event in the sequence is a set of conditions, as opposed to the single items in the email recipient recommendation application. The patient visits are ordered in time, however, each visit itself consists of a set of symptoms which we treat as unordered. Also, the same condition can occur at multiple visits throughout the patient history, unlike email recipient recommendation, in which addresses are not repeated in a sequence. Because of this, it is important to be able to predict condition recurrence. We thus estimate $\hat{\mathbb{P}}(b|a)$ used by the max-confidence association rule algorithm and the ML-constrained model as the probability of having condition b *later in the sequence* given that a has been observed in the sequence. In this application it is not natural to make a prediction before the patient's first visit ($t = 0$), thus we make predictions only at visits $t = 1, \ldots, T_i - 1$.

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 those 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 (3) and (4) 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 (3) becomes:

$$R_{0-1}^{\text{cond}}\big(f, X_1^m; \boldsymbol{\lambda}\big) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{T_i-1}\Bigg[\frac{1}{(T_i-1)(N-|z_{i,t+1}|)} \times \sum_{k\in\mathcal{Z}\setminus z_{i,t+1}}\sum_{l\in\tilde{z}_{i,t+1}}\frac{1}{|\tilde{z}_{i,t+1}|}\mathbb{1}_{[f(x_{i,t},k;\boldsymbol{\lambda})\geq f(x_{i,t},l;\boldsymbol{\lambda})]}\Bigg]. \tag{8}$$

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 (4) for this application as:

$$R_{\exp}^{\text{cond}}\left(f, X_1^m; \boldsymbol{\lambda}\right) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{T_i-1}\Bigg[\frac{1}{(T_i-1)(N-|z_{i,t+1}|)} \times \sum_{k\in\mathcal{Z}\setminus z_{i,t+1}}\sum_{l\in\tilde{z}_{i,t+1}}\frac{1}{|\tilde{z}_{i,t+1}|}e^{f(x_{i,t},k;\boldsymbol{\lambda})-f(x_{i,t},l;\boldsymbol{\lambda})}\Bigg] + \beta\|\boldsymbol{\lambda}\|_2^2. \tag{9}$$

We used L-BFGS-B to minimize (9) to fit the one-stage and ML-constrained models 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 were 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. We took $\mathcal{A}$ as all itemsets of size 1, plus the empty set. Fitting model variables required an optimization problem on 3,476,360 variables for the one-stage model (N^2+N) and 3,728 variables for ML-constrained model ($2N$).

To illustrate the behavior of our models, and the differences between the one-stage model and the ML-constrained model, in Fig. 5 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 Fig. 5 shows $\text{Conf}(a\rightarrow b)$ for each pair of items in row a and column b, which is equivalent to the conditional probability estimate $\hat{\mathbb{P}}(b|a)$. The high confidence values on the diagonal indicate that the conditional probability of having these conditions in the future given their past occurrence 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 obtained by weighting each row a of the association rule confidence matrix by μ_a. However, the main features of the ML-constrained model variables are different from those of the association rule confidence matrix, and in fact the ML-constrained variables are similar to those of the one-stage model, $\lambda_{a,b}$. With both models, the strength with which the recurrence of a condition is predicted (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 models, 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 more varied so it is difficult to interpret the relation between Nutritional support and Prophylaxis, Headache, and Hypertension.

To evaluate the performance of the ERM-based algorithms, we performed ten iterations of random sub-sampling with training and test sets each of 500 patients. We applied the cosine similarity algorithm with varying neighborhood sizes (20, 40, and all items), the

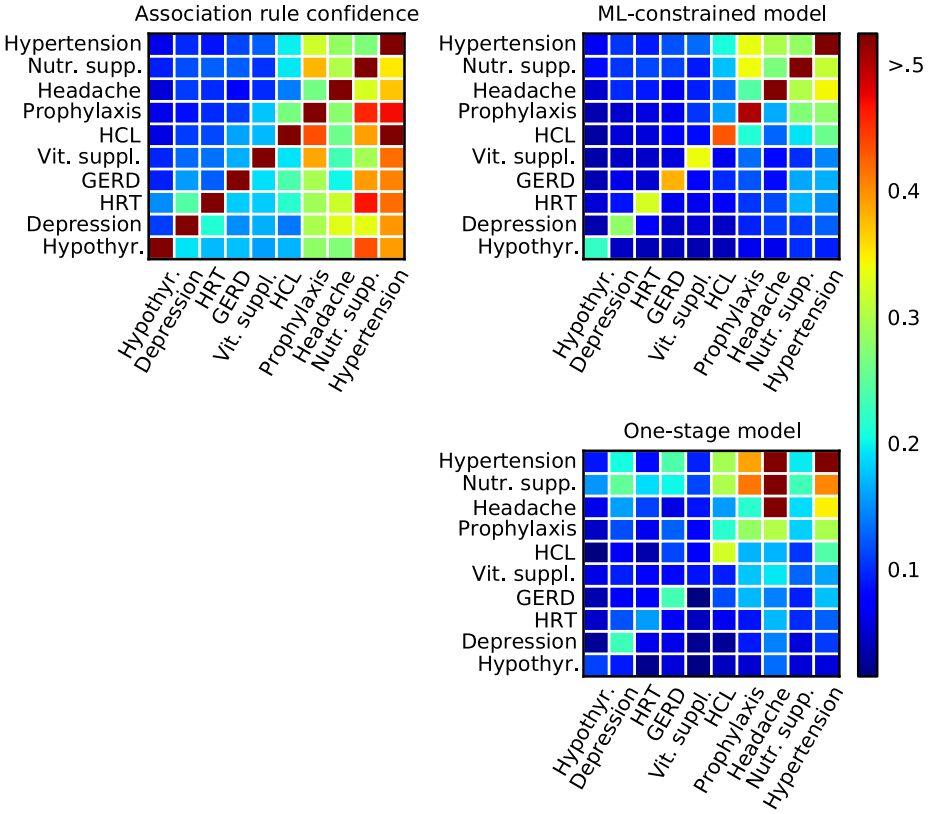

Fig. 5 An example of fitted model variables for the ten most frequent conditions in the patient condition prediction problem, for the one-stage and ML-constrained models, together with the association rule confidence matrix. This figure illustrates the differences between the fitted variables of the two models. *Row a column b* is: Conf($a \rightarrow b$) for association rules; $\mu_a \hat{\mathbb{P}}(b|a)$ for the ML-constrained model; and $\lambda_{a,b}$ for the one-stage model. Abbreviated symptoms are Nutritional support (Nutr. supp.), Hypercholesterolaemia (HCL), Vitamin supplementation (Vit. suppl.), Gastroesophageal reflux disease (GERD), Hormone replacement therapy (HRT), and Hypothyroidism (Hypothyr.)

max-confidence association rule algorithm, the one-stage model, and the ML-constrained model. 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 and test sets was evaluated using (8), and boxplots of the results across all 10 iterations are in Fig. 6. When evaluating the test error in Fig. 6, 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.

Our method, using both models, performed very well compared to the baselines, which seem to have had an overfitting problem judging from the difference between training and test results. The ML-constrained model used far fewer variables than the one-stage model (about 3000 compared to about 3.5 million) and generalized well.

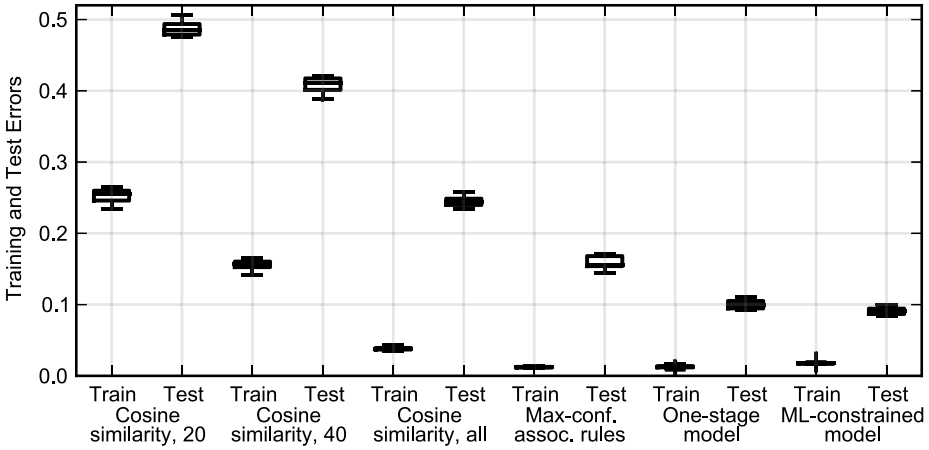

Fig. 6 Training and test errors for patient condition prediction

6 Application 3: online grocery store recommender system

In this application a customer comes to the online grocery store with a shopping list, and sequentially adds the items from his or her shopping list into a shopping basket. The goal is to, at each time step, make predictions about the other items the customer wishes to purchase. 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. Similar to the email recipient recommendation application, the sequence is of single, non-repeated items. The sequence is initialized as an empty basket: $z_{i,0} = \varnothing$. In this application we assume that the customer's shopping list (the collection of items that form the sequence) has no inherent order. Rather, we assume that the order in which the customer adds items to the basket depends directly on the recommendations made to the customer.

6.1 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. For convenience, denote the contents of the basket at time t as $Z_{i,t} := \bigcup_{j=1}^{t} z_{i,j}$ and the contents of the shopping list as $Z_i := \bigcup_{j=1}^{T_i} z_{i,j}$. We then order the items according to:

$$z_{i,t+1} \in \operatorname*{argmax}_{b \in Z_i \setminus Z_{i,t}} f(x_{i,t}, b; \boldsymbol{\lambda}). \tag{10}$$

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 (10) could be replaced by an application-specific model of user behavior; (10) 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 (4) with respect to variables $\boldsymbol{\lambda}$, using (10). The qualitative effect of (10) 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 (10). 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 (10), 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.

6.2 Specifying the loss function

In this application we use two loss functions. First, we use the list loss that was defined in (6) and (7) for the email recipient recommendation. This loss function has the goal of placing all of the items that remain on the shopping list at the top of the recommendations. In some situations, however, we may not need the entire shopping list near the top of the recommendations, 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 at least one item on the shopping list ranked higher than all items that are not on the shopping list:

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

This can be realized using by taking $L_{i,t} = \arg\max_{l\in Z_i\setminus Z_{i,t}} f(x_{i,t},l;\boldsymbol{\lambda})$ and $K_{i,t} = \mathcal{Z}\setminus Z_i$. The general loss function in (3) then becomes

$$R_{0-1}^{\text{item}}\big(f, X_1^m;\boldsymbol{\lambda}\big) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\frac{1}{T_i(N-T_i)}\sum_{k\in\mathcal{Z}\setminus Z_i}\mathbb{1}_{[f(x_{i,t},k;\boldsymbol{\lambda})\geq\max_{l\in Z_i\setminus Z_{i,t}} f(x_{i,t},l;\boldsymbol{\lambda})]} \tag{11}$$

and (4) becomes

$$R_{\text{exp}}^{\text{item}}\big(f, X_1^m;\boldsymbol{\lambda}\big) := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i-1}\frac{1}{T_i(N-T_i)}\sum_{k\in\mathcal{Z}\setminus Z_i} e^{f(x_{i,t},k;\boldsymbol{\lambda})-\max_{l\in Z_i\setminus Z_{i,t}} f(x_{i,t},l;\boldsymbol{\lambda})} + \beta\|\boldsymbol{\lambda}\|_2^2. \tag{12}$$

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, lemon
Shopping list 5: flour, oil, baking powder
Shopping list 6: flour, sugar, vanilla

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 1–4. 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 1–4, and the loss is one for each of shopping lists 5 and 6. 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 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 our experiments, we found that minimizing item loss produced 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.

6.3 ERM for the online grocery store recommender system

The model variables $\boldsymbol{\lambda}$ are chosen to minimize the loss on the training set by minimizing the list loss (7) or item loss (12). Using the assumption in (10), the basket at any time step $Z_{i,t}$ is itself a function of the recommender system, i.e., of $\boldsymbol{\lambda}$. Small changes in $\boldsymbol{\lambda}$ can change which item has the highest score, thus changing $z_{i,t+1}$. Because the predictions at $t + 1$ and all future time steps depend on $z_{i,t+1}$, this can significantly alter the value of the loss. Depending on $\boldsymbol{\lambda}$, different possibilities for $z_{i,t+1}$ could change $f(x_{i,t+1}, b; \boldsymbol{\lambda})$ by arbitrarily large amounts. This is why the loss functions in (7) and (12) are, subject to the assumption in (10), generally discontinuous.

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. It is when there are ties that epsilon changes in $\boldsymbol{\lambda}$ can alter $z_{i,t+1}$ and thus significantly change all future predictions. These discontinuities partition the variable space into regions that correspond to different orderings. Figure 7 is an illustration of how the space of $\boldsymbol{\lambda}$ is partitioned by different orderings,

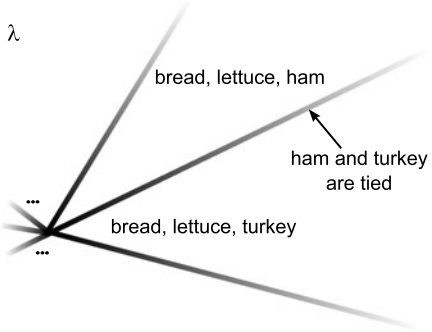

Fig. 7 An illustration of how the model variables can be partitioned into *regions* that lead to different orderings of the items in each shopping basket. The *borders* between *regions* correspond to selections of model variables for which the argmax in (10) is not unique, i.e., there is a tie. The *regions* are polyhedral, and the objective function is convex over each *region* but discontinuous at the *borders*

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 *Let Λ_{z^*} be the set of $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{A}|N}$ in the one-stage model or $(\boldsymbol{\lambda}_\varnothing, \boldsymbol{\mu}) \in \mathbb{R}^{|\mathcal{A}|+N}$ in the ML-constrained model that can produce the specific ordering $\{z^*_{i,t}\}_{i,t}$ under the assumption of* (10)*. Then, Λ_{z^*} is a polyhedron.*

Proof A particular ordering z^* is produced when, for each training list i and at each time step t, the next item in the ordering $z^*_{i,t+1}$ has the highest score of all of the items remaining on the shopping list. In other words, to choose $z^*_{i,t+1}$ 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 $z^*_{i,k}$:

$$f\left(x^*_{i,t}, z^*_{i,t+1}; \boldsymbol{\lambda}\right) \geq f\left(x^*_{i,t}, z^*_{i,k}; \boldsymbol{\lambda}\right), \quad \forall k > t+1.$$

These constraints in fact define Λ_{z^*}:

$$\Lambda_{z^*} := \Bigg\{ \boldsymbol{\lambda} : \lambda_{\varnothing, z^*_{i,t+1}} + \sum_{j=1}^{t} \sum_{\substack{a \subseteq z^*_{i,j} \\ a \in \mathcal{A} \setminus \varnothing}} \lambda_{a, z^*_{i,t+1}} \geq \lambda_{\varnothing, z^*_{i,k}} + \sum_{j=1}^{t} \sum_{\substack{a \subseteq z^*_{i,j} \\ a \in \mathcal{A} \setminus \varnothing}} \lambda_{a, z^*_{i,k}},$$
$$i = 1, \ldots, m,\ t = 0, \ldots, T_i - 2,\ k = t+2, \ldots, T_i \Bigg\}. \tag{13}$$

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 for the ML-constrained model by replacing $\lambda_{a,b}$ with $\mu_a \hat{\mathbb{P}}(b|a)$. □

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, $\boldsymbol{\lambda}_\varnothing$ and $\boldsymbol{\mu}$) are constrained to a particular ordering Λ_{z^*}, the list loss in (7) and the item loss in (12) are convex. We will now describe two optimization strategies for minimizing (7) and (12) subject to the assumption in (10). For notational convenience we will describe the algorithms in terms of $\boldsymbol{\lambda}$, but the algorithms can be directly applied to the ML-constrained model.

6.3.1 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 a 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 $\boldsymbol{\lambda}$ under the assumption of (10)

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 (Eq. (13))
Solve the convex program $\boldsymbol{\lambda}^* \in \operatorname{argmin}_{\boldsymbol{\lambda}\in\Lambda_z} R_{\exp}(f, X_1^m; \boldsymbol{\lambda})$ (Eq. (7) or (12))
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 $\boldsymbol{\lambda}'^* \in \operatorname{argmin}_{\boldsymbol{\lambda}\in\Lambda_{z'}} R_{\exp}(f, X_1^m; \boldsymbol{\lambda})$
  Sample a number q uniformly at random from $[0, 1]$
  if $\exp((R_{\exp}(f, X_1^m; \boldsymbol{\lambda}^*) - R_{\exp}(f, X_1^m; \boldsymbol{\lambda}'^*))/\mathit{Temp}(t)) > q$ **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**
      $\boldsymbol{\lambda}_{\text{best}} = \boldsymbol{\lambda}^*$

6.3.2 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 model which requires $|\mathcal{A}|N$

variables. It may be more efficient to use an unconstrained first-order method such as pure gradient descent. 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 $\boldsymbol{\lambda}$ under assumption (10)

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 $R_{\text{exp}}(f, X_1^m; \boldsymbol{\lambda}_0)$ (Eq. (7) or (12))
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**
    $l = l + 1$
  $t = t + 1$

6.4 Experimental results

Our online grocery store 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. We limited our experiments to the 250 most frequently occurring ingredients. This excluded only very rare ingredients that appeared in less than 5 recipes in the dataset, for instance "alligator." We took the allowed itemsets $\mathcal{A}$ as individual items, plus the empty set. The ML-constrained model thus required an optimization problem on 500 variables ($2N$) whereas the one-stage model required solving an optimization problem on 62,500 variables ($N^2 + N$).

We fit the one-stage model and the ML-constrained model, using both list loss in (7) and item loss in (12). 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 (6) or (11). Training and test sets were sampled independently for 20 trials to provide a distribution of training and test losses. 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 the one-stage model, chosen values of β

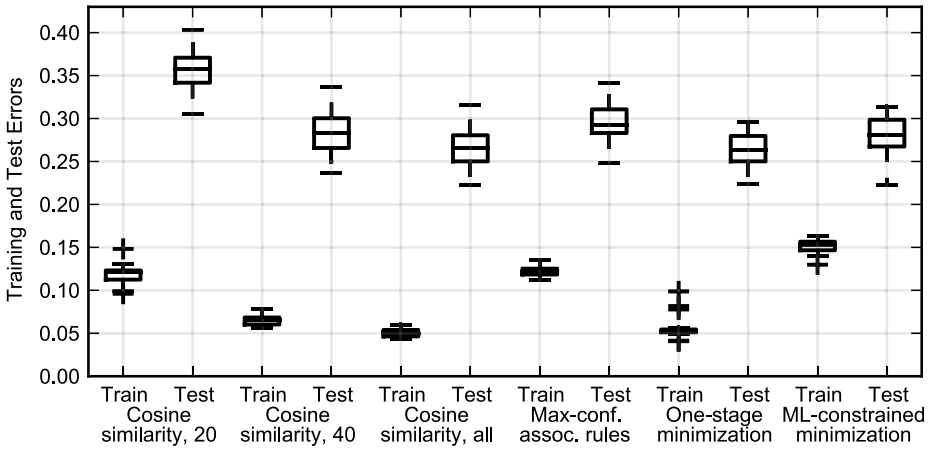

Fig. 8 List loss training and test errors for the online grocery store recommender system

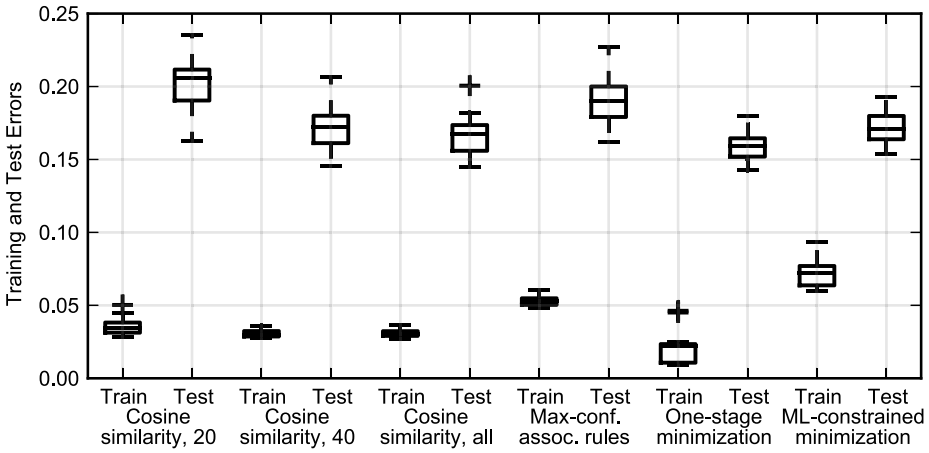

Fig. 9 Item loss training and test errors for the online grocery store recommender system

ranged from 0.001 to 0.1, with 0.001 chosen most frequently. With list loss and the ML-constrained model, chosen values of β ranged from 0.01 to 1, with 0.1 chosen most frequently. With item loss and the one-stage model, chosen values of β ranged from 0.0001 to 0.01, with 0.001 chosen most frequently. With item loss and the ML-constrained model, 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 Figs. 8 and 9 for list loss and item loss respectively. As before, the test errors in Figs. 8 and 9 exclude items that were not present in the training set, as these items necessarily cannot be well predicted and provided a constant bias.

The large difference between training and test errors suggests 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 meth-

ods), although if it were desirable to fit a model individually to each customer 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 performed well compared to the cosine similarity and association rule baselines. The one-stage model performed slightly better than the ML-constrained model, although it does so at a much higher computational cost. This is consistent with the results of the other experiments in this paper, which have shown that the ML-constrained model is able to provide close to the same performance as the one-stage model.

7 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 (Sect. 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. These differences help explain why in our experiments, particularly email recipient recommendation and patient condition prediction, 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. (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; 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 model 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 (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 Chang et al. (2012). Our formulation could potentially be adapted to optimize other evaluation metrics, as is done in Yue et al. (2007) and Chang et al. (2012), 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).

8 Conclusions

We have presented a supervised ranking framework for sequential event prediction that can be adapted to fit a wide range of applications. We proposed two ranking models, and showed how to specify our general loss function to applications in email recipient recommendation, patient condition prediction, and an online grocery store recommender system. In the online grocery store recommender system application, we allowed the predictions to alter the sequence of events resulting in a discontinuous loss function. 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. In all of our experiments, our ERM-based algorithms performed well, better than the max-confidence and cosine similarity baselines. Our ML-constrained model in particular provided good performance while keeping the dimensionality of the optimization problem small. There are many other applications where the *set* of past events matters for predicting the future, rather than the order of

past events. Our ERM-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 29th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'06* (pp. 19–26).

Agichtein, E., Brill, E., Dumais, S., & Ragno, R. (2006b). Learning user interaction models for predicting web search result preferences. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'06* (pp. 3–10).

Berchtold, A., & Raftery, A. E. (2002). The mixture transition distribution model for high-order Markov chains and non-Gaussian time series. *Statistical Science*, *17*(3), 328–356.

Bertsekas, D. P. (1995). *Nonlinear programming*. Belmont: Athena Scientific.

Bigal, M. E., Liberman, J. N., & Lipton, R. B. (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'10* (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'05* (pp. 89–96).

Byrd, R. H., 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, T. Y., Tsai, M. F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on machine learning, ICML'07* (pp. 129–136).

Carvalho, V. R., & Cohen, W. W. (2008). Ranking users for intelligent message addressing. In *Proceedings of the 30th European conference on IR research, ECIR'08* (pp. 321–333).

Chang, A., Rudin, C., Cavaretta, M., Thomas, R., & Chou, G. (2012). How to reverse-engineer quality rankings. *Machine Learning*, *88*(3), 369–398.

Chapelle, O., & Keerthi, S. S. (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* (pp. 305–312).

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

Cohen, W. W., Schapire, R. E., & Singer, Y. (1999). Learning to order things. *The Journal of Artificial Intelligence Research*, *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, D. A., Chawla, N. V., Christakis, N. A., & Barabasi, A. L. (2010). Time to CARE: a collaborative engine for practical disease prediction. *Data Mining and Knowledge Discovery*, *20*, 388–415.

Dekel, O., Manning, C. D., & Singer, Y. (2004). Log-linear models for label ranking. In: *Advances in neural information processing systems 16* (pp. 497–504).

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*, pp. 159–168). Springer, Berlin.

Freund, Y., Iyer, R., Schapire, R. E., & 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 *Proceedings of the 14th European conference on machine learning, ECML'03* (pp. 145–156).

Herbrich, R., Graepel, T., & Obermayer, K. (1999). Support vector learning for ordinal regression. In *Proceedings of the 9th international conference on artificial neural networks, ICANN'99* (pp. 97–102).

Herlocker, J. L., Konstan, J. A., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'99* (pp. 230–237).

Hu, Q., Huang, X., Melek, W., & Kurian, C. (2010). A time series based method for analyzing and predicting personalized medical data. In Y. Yao, R. Sun, T. Poggio, J. Liu, N. Zhong, & J. Huang (Eds.), *Lecture notes in computer science*. *Brain informatics* (pp. 288–298). Springer, Berlin.

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. 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 8th ACM SIGKDD international conference on knowledge discovery and data mining, KDD'02* (pp. 133–142).

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'02* (pp. 363–370).

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

McCormick, T. H., Rudin, C., & Madigan, D. (2012). Bayesian hierarchical modeling for predicting medical conditions. *Annals of Applied Statistics*, *6*(2), 652–668.

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

Pal, C., & McCallum, A. (2006). Cc prediction with graphical models. In *Proceedings of the 3rd conference on email and anti-spam, CEAS'06*.

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'08* (pp. 784–791).

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, R. E. (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'11* (pp. 615–634).

Rudin, C., Letham, B., Kogan, E., & Madigan, D. (2012) *A learning theory framework for sequential event prediction and association rules*. (Working paper OR 394-12). Massachusetts Institute of Technology, Operations Research Center.

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, R. I. (2005). An MDP-based recommender system. *Journal of Machine Learning Research*, *6*, 1265–1295.

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, A. G. (2006). Efficient margin-based rank learning algorithms for information retrieval. In *Proceedings of the 5th international conference on image and video retrieval, CIVR'06* (pp. 113–122).

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, R. H., 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.