Automatic Derivation of the Multinomial PCA Algorithm

Wray Buntine
Helsinki Institute for IT
buntine@hiit.fi

Bernd Fischer
RIACS / NASA Ames
fisch@email.arc.nasa.gov

Alexander G. Gray
Carnegie Mellon University
agray@cs.cmu.edu

Abstract

Machine learning has reached a point where probabilistic methods can be understood as variations, extensions, and combinations of a small set of abstract themes, e.g., as different instances of the EM algorithm, or as exponential family methods. This allows the automatic derivation of algorithms customized for different models. One interesting new model is a multinomial version of PCA which has received attention due to its ability to better model documents as having multiple topics. Here we explain how AutoBayes, an automatic synthesis system for machine learning programs, derives from a high-level statistical specification of this model code similar to the original non-negative matrix factorization algorithm of Lee and Seung. The derivation combines multiple statistical schemes to solve a new problem not originally envisaged. This demonstrates that the approach can be scaled up from text-book problems to research-level algorithm design.

1 INTRODUCTION

AUTOBAYES. In earlier work [5, 7] we described AutoBayes, an automatic program synthesis system which can compile a statistical model specification into a custom *algorithm design* and then further down into a working program implementing that design. AutoBayes can be loosely thought of as "part theorem prover, part Mathematica, part statistics textbook, and part Numerical Recipes." It is an *algorithm compiler* which provides much more flexibility than a fixed code repository such as a Matlab toolbox, and allows the creation of efficient algorithms which have never before been implemented, or even written down. The system is intended to automate the application of known algorithmic principles in novel contexts, from the simple to the complex.

Multinomial PCA. In order to explore and extend the capabilities of AutoBayes, we have applied it to several problems from the statistical algorithm literature, as described in [7]. The most complex and challenging example among these is the recent *multinomial Principle Components Analysis* (multinomial PCA) model. Several authors have proposed this multinomial analogue to PCA to handle discrete or positive only data; methods and names for the problem include non-negative matrix factorization [12], probabilistic latent semantic analysis [9], latent Dirichlet allocation [1], generative aspect models [14]. A good discussion of the motivation for these techniques can be found in [9].

Overview of this paper. In [5, 7] we introduced the system and gave examples of its capabilities across several statistical problems ranging from simple maximum likelihood and Bayesian textbook problems to recently published algorithms from the literature [13, 6, 18, 3]. In [7] we focused on illustrating the ability of the system to derive pure EM algorithms, i.e., algorithms which result from the triggering of only the EM schema, possibly with analytic solutions along the way; the detailed derivation example we presented was the case of the standard EM algorithm for Gaussian mixtures. Here we consider the system's behavior at its current limit along the spectrum of complexity, through the derivation of multinomial PCA. Aside from its interest as a detailed example of a research-level algorithm design, the automated multinomial PCA derivation demonstrates a key advantage inherent in the design of AutoBayes: the automatic combination of *multiple* abstract schemas, allowing algorithmic principles to be used as fundamental building blocks, at arbitrary points in a derivation. The process of examining the multinomial PCA case in detail illustrates a number of interesting points and spawns several technical goals for the continued development of AutoBayes.

2 DERIVING STATISTICAL ALGORITHMS BY COMPUTER

The problem of designing statistical algorithms has an inherently combinatorial nature, since subparts of a function may be optimized recursively and in different ways. It also involves the use of new data structures or approximations to gain performance. As the research in statistical algorithms advances, its creative focus should move beyond ultimately mechanical aspects and towards extending the abstract applicability of algorithmic principles like EM, Expectation-Propagation and variational methods, and inventing radically new principles. The application of these principles should (and can) be left to the computer.

2.1 THE STATISTICAL ALGORITHM DESIGN PROBLEM

Why is computerized algorithm design even possible? A number of factors unite to allow us to solve the algorithm design problem computationally.

The existence of fundamental building blocks. Standardized distributions and their combinations (e.g., mixture models) are useful for describing the real world. Standard algorithms (e.g., Newton's method or linear programming) developed in numerical analysis, operations research, and computer science can be instantiated and combined to solve optimization problems. Moreover, specialized algorithms have been developed for certain classes of models, e.g., back-propagation for networks, top-down construction for trees, forward-backward dynamic programming for Markov models, or EM for latent-variable models.

The existence of a common representation. Graphical models have emerged as a general framework capable of representing probabilistic models from neural networks to decision trees to hidden Markov models, specifying the composition of individual distributions into an arbitrarily complex probabilistic model [17, 2, 16]. Such a common framework does not exist, however, for optimization algorithms; in our method we provide one, where all algorithms are represented as *schemas* (described below).

The formalization of known constraints. Combining optimizers with models correctly requires one more piece of information: the applicability constraints for each schema. Some optimizers are more dependent on the specific form of the optimization problem than others. For example, Nead-Melder simplex, simulated annealing, and Gibbs sampling require only the ability to evaluate the function to be maximized, conjugate gradient requires both first-derivative and Hessian information, while EM presupposes a latent-variable structure. In AutoBayes part of the definition of each optimization schema is the set of known constraints on its applicability, or *guards*.

2.2 CHALLENGES

The design of the AutoBayes-system posed a number of major challenges, in particular (*i*) the choice of a scalable synthesis methodology, (i.e., schema-based synthesis as described below), (*ii*) the extension of Bayesian networks in a practical manner to include specification and reasoning about vectors, matrices, and tensors, and (*iii*) the integration of symbolic reasoning about mathematical formulae and concepts like positivity, non-zeroness etc. into the synthesis process.

2.3 RELATED SYSTEMS

Several existing systems tackle related problems but without approaching the broad functionality of AutoBayes. Code libraries are common in statistics and learning (e.g., Matlab or S), but they lack the high level of automation achievable only by symbolic reasoning. The Bayes Net Toolbox [15] is a Matlab library which allows users to program in models but it cannot derive algorithms or generate code; the need to generate fast inner loops which has emerged recently could be addressed by AutoBayes. The Bugs-system [17] also allows users to program in models but it is specialized for Gibbs sampling, a universal but less efficient Bayesian inference technique. Gibbs sampling could be integrated into AutoBayes as an additional algorithm schema. The system of Ellman and Murata [4] also synthesizes programs from a high-level specification, but for the domain of differential equation modeling.

3 THE AutoBayes SYSTEM

Our approach uses techniques from computational logic. It has its roots in the general idea of *theorem proving*: synthesis begins with an initial goal and a set of initial assertions, or axioms, and adds new assertions, or theorems, by repeated application of the axioms, until the goal is proven. In our context, the goal is given by the statistical model; the derived algorithms are side effects of constructive theorems proving the existence of algorithms for the goal.

3.1 COMBINING SCHEMA-BASED SYNTHESIS AND BAYES NETS

Schema-based synthesis. While attractive from a first-principles point of view, the computational cost of full-blown theorem proving grinds simple tasks to a halt while elementary and intermediate facts are reinvented from scratch. To achieve the scale of deduction required by algorithm derivation, we thus utilize a *schema-based synthesis* technique which breaks away from strict theorem proving. Instead, we effectively encode high-level knowledge, such as the general EM strategy, as *schemas*, or auxiliary lemmas with explicitly specified preconditions. The first core element which makes automatic algorithm derivation feasible is the fact that we can use Bayesian networks to efficiently encode the preconditions of complex algorithms such as EM, and to drive the process of generating arbitrary probability expressions as needed.

First-order logic representation of Bayesian networks. A theory of indexed Bayesian networks was developed in [8]; here indices are represented as Prolog variables and networks correspond to backtrack free datalog programs, allowing the dependencies to be efficiently computed. We have extended these results to work with non-ground probability queries since we seek to determine probabilities over vectors and matrices, for instance sample likelihoods and posterior probabilities. Tests for independence on these indexed Bayesian networks are easily developed in Lauritzen's framework which uses ancestral sets and set separation [11] and is more amenable to a theorem prover than the double negatives of the more widely known d-separation criteria. Given a Bayesian network, some probabilities can easily be extracted by enumerating the component probabilities at each node:

Lemma 1. *Let U, V be sets of variables over a Bayesian network with $U \cap V = \emptyset$. Then $V \cap descendents(U) = \emptyset$ and $parents(U) \subseteq V$ hold in the corresponding dependency graph iff the following probability statement holds:*

$$Pr(U \,|\, V) = Pr(U \,|\, \mathrm{parents}(U)) = \prod_{u \in U} Pr(u \,|\, \mathrm{parents}(u)).$$

Symbolic probabilistic inference. How can probabilities not satisfying these conditions be converted to symbolic expressions? While many general schemes for inference on networks exist, our principal hurdle is the need to perform this over symbolic expressions incorporating real and integer variables from disparate real or infinite-discrete distributions. For instance, we might wish to compute the full *maximum a posteriori* probability for the mean and variance vectors of a Gaussian mixture model under a Bayesian framework. While the sum-product framework of [10] is perhaps closer to our formulation, we have out of necessity developed another scheme that lets us extract probabilities on a large class of mixed discrete and real, potentially indexed variables, where no integrals are needed and all marginalization is done by summing out discrete variables. We give the non-indexed case below; this is readily extended to indexed variables (i.e., vectors).

Lemma 2. *$V \cap descendents(U) = \emptyset$ holds and $ancestors(V)$ is independent of U given V iff there exists a set of variables U' such that Lemma 1 holds if we replace U by $U \cup U'$. Moreover, the unique minimal set U' satisfying these conditions is given by $ancestors(U) \,/ (ancestors(V) \cup V)$.*

Lemma 3. *Let V' be a subset of $V/descendents(U)$ such that $ancestors(V')$ is independent of $(U \cup V)/(V' \cup ancestors(V'))$ given V'. Then Lemma 2 holds if we replace U by $U \cup V/V'$ and V by V'. Moreover, there is a unique maximal set V' satisfying these conditions.*

Lemma 2 lets us evaluate a probability by a summation:

$$Pr(U \,|\, V) = \sum_{u' \in \mathrm{Dom}(U')} Pr(U' = u', U \,|\, V)$$

while Lemma 3 lets us evaluate a probability by a summation and a ratio:

$$Pr(U \,|\, V) = \frac{Pr(U \cup V/V' \,|\, V')}{Pr(V/V' \,|\, V')}$$

Since the lemmas also show minimality of the sets U' and V/V', they also give the minimal conditions under which a probability can be evaluated by discrete summation without integration. These inference lemmas are operationalized as network decomposition schemas. However, we usually attempt to decompose a probability into independent components before applying this schema.

Computer algebra. The second core element which makes automatic algorithm derivation feasible is the fact that we can mechanize the required symbol manipulation, using computer algebra methods. General *symbolic differentiation* and *expression simplification* are capabilities fundamental to our approach, for instance checking legality of expressions such as X/Y or $\log(X)$ where domain constraints apply, simplifying expressions such as $\sum_{i=1,\ldots,I} x_j$ (e.g., to $I * x_j$), and symbolically generating Jacobians and Hessians as needed. AutoBayes contains a computer algebra engine using term rewrite rules which are an efficient mechanism for substitution of equal quantities or expressions and thus well-suited for this task. Popular symbolic packages such as Mathematica contain known errors allowing unsound derivations; they also lack support for reasoning about vector and matrix quantities. Extending Mathematica with the required support turned out to be impractical given the tight level of integration we required.

3.2 IMPLEMENTATION

Levels Of Representation. Internally, our system uses three conceptually different levels of representation. *Probabilities* (including logarithmic and conditional probabilities) are the most abstract level. They are processed via methods for Bayesian network decomposition or matches with core algorithms such as EM. *Formulae* are introduced when probabilities of the form $Pr(U \mid \text{parents}(U))$ are detected, either in the initial network, or after the application of network decompositions. Atomic probabilities (i.e., U is a single variable) are directly replaced by formulae based on the given distribution and its parameters. General probabilities are decomposed into sums and products of the respective atomic probabilities using the lemmas just given. Formulae are ready for immediate optimization using symbolic or numeric methods but sometimes they can be decomposed further into independent subproblems. Finally, we use an imperative *intermediate code* as the lowest level to represent both program fragments within the schemas as well as the completely constructed programs. All transformations we apply operate on or between these levels.

Transformations For Optimization. A number of different kinds of transformations are available. *Decomposition* of a problem into independent subproblems is always done. Decomposition of probabilities is driven by the Bayesian network; we have a separate system for handling decomposition of formulae. A formula can be decomposed along a loop, e.g. the problem "optimize $\vec{\theta}$ for $\prod_i f(\theta_i)$" is transformed into a for-loop over subproblems "optimize θ_i for $f(\theta_i)$." More commonly, "optimize θ, ϕ for $f(\theta) + g(\phi)$" is transformed into the two subprograms "optimize θ for $f(\theta)$" and "optimize ϕ for $g(\phi)$." The lemmas given earlier are applied to change the level of representation and thus for *simplification of probabilities*. Examples of general *expression simplification* include simplifying the log of a formula, moving a summation inwards, and so on. When necessary, *symbolic differentiation* is performed. In the initial specification or in intermediate representations, *likelihoods* (i.e. subexpressions of the form $log \prod_i Pr(x_i \mid \theta))$ are identified and simplified into linear expression with terms such as $mean(x_i)$ and $mean(x_i^2)$. The *statistical algorithm schemas* currently implemented include EM, k-means, and discrete model selection. Adding a Gibbs sampling schema would yield functionality comparable to that of BUGS [17]. Usually, the schemas require a particular form of the probabilities involved; they thus interact with decomposition and simplification transformations. For example, EM is a way of dealing with situation where Lemma 2 applies but where U' is indexed identically to the data, so-called "hidden" data.

Code Generation. From the intermediate code, code in a particular *target language* may be generated. Currently, AutoBayes can generate C++ and C which can be used in a stand-alone fashion or linked into Octave or Matlab (as a mex file). During this code-generation phase, most of the vector and matrix expressions are converted into for-loops, and various *code optimizations* are performed which are impossible for a standard compiler. Our tool does not only generate *efficient* code, but also highly readable, *documented* programs: model- and algorithm-specific comments are generated automatically during the synthesis phase. For most examples, roughly 30% of the produced lines are comments. These comments provide explanation of the algorithm's derivation. The intermediate code for multinomial PCA is presented later.

3.3 TESTING

AutoBayes also automatically generates a program for *sampling* from the specified model, so that closed-loop testing with synthetic data of the assumed distributions can be done. This uses simple forward sampling because test data needs to be generated along with the "true" model parameters to generate a full test case. Linking to Octave means we can use Octave a test scripting language.[1] For instance, we developed a specification of trajectory clustering [6] where clusters of curves exist in 2-D and the task combines regression with mixture modeling. For this, (1) the AutoBayes-generated sampler linked into to Octave generates a "true" set of curves, and samples from them, (2) the AutoBayes-generated learner linked into to Octave takes the sample and estimates a set of curves, (3) Octave is used to plot the "true" and estimated curves, and the data, and generate summary statistics, and (4) multiple trials of the above process are done. In this manner, we test our generated code. Examples of the statistical problems tested through the system appear in [7].

4 OVERVIEW OF MULTINOMIAL PCA

Consider the problem where a sample of documents are input as data. First sample a C-dimensional probability vector $\vec{m}$ that represents the proportional weighting of topics/components in a document, and then mix this with a C dimensional set of W-dimensional probability vectors $\vec{\Omega}$ representing the W word probabilities for each topics/component.

$$\vec{m} \quad \sim \quad Dirichlet(\vec{\alpha}) \quad ,$$

[1] We use Octave rather than Matlab because it is open source.

```
1 model mpca as 'Multinomial PCA';
% Dimensions
2 const int n_points as 'nr. of documents'
3   with 0 < n_points;
4 const int n_classes as 'nr. of classes'
5   with 0 < n_classes and n_classes << n_points;
6 const int n_feats as 'nr. of words in each document'
7   with 0 < n_feats;
8 const int n_words as 'nr. of distinct words'
9   with 0 < n_words;
% Model Parameters
10 const double alpha(0..n_classes-1) as 'Dirichlet parameters';
11 double omega(0..n_classes-1,0..n_words-1) as 'components'
12   with 0 =< omega(_,_)
13    and 1 = sum(J:=0..n_words-1,omega(_,J));
% Hidden Variables
14 double m(0..n_points-1,0..n_classes-1) as 'doc proportions'
15   with 0 =< m(_,_) and 1 = sum(J:=0..n_classes-1,m(_,J));
16 vector(J:=0..n_classes-1,m(I,J)) ~
17   dirichlet(vector(K:=0..n_classes-1,alpha(K)));
18 int k(0..n_points-1,0..n_feats-1) as 'component for word'
19   with 0 =< k(_,_) < n_classes;
20 k(I,_) ~ discrete(vector(J:=0..n_classes-1,m(I,J)));
% Data
21 data double x(0..n_points-1,0..n_feats-1) as 'data'
22   with 0 =< x(_,_) < n_words;
23 x(I,J) ~ discrete(vector(K:=0..n_words-1,omega(k(I,J),K)));
% Objective
24 max pr(x|omega) wrt omega;
```

Figure 1: AutoBayes Model Code for MPCA

$$\vec{b} \sim Multinomial(\vec{m}\vec{\Omega}, F) ,$$

where F is the total number of words in the document. This gives $\vec{b}$ as a multinomial sample for the bag of words representation of a document. Equivalently, we can assign each word position (or feature) in sequence to a unique topic according to $\vec{m}$, and then sample a word according to the word probabilities for that topic. This introduces an additional hidden variable $\vec{k}$ which is the assigned topics for each word position. This models each document as a vector of word indexes as in Blei, Ng and Jordan [1].

This second version is the model given in Figure 1, which is written in the modeling language for AutoBayes. In the figure, keywords have been underlined and line numbers have been added for reference in the text. The `as`-keyword allows annotations to variables which end up as comments in the generated code. Constants may remain, unspecified, e.g., `n_points`. The "$\sim$" symbol denotes "distributed as". Note the constraint that the sum of the class probabilities must equal one (line 15) along with others (lines 9 and 13, etc.) that make optimization of the model well-defined. The last line specifies the goal inference task: maximize the conditional probability $\mathrm{pr}(\vec{x}|\vec{\Omega})$ with respect to the parameters $\vec{\Omega}$. Note that moving the parameters across to the left of the conditioning bar converts this from a maximum likelihood to a *maximum a posteriori* problem.

This second representation is used because it has documents as a vector of word indexes; our system currently does not support sparse vectors needed for the bag of words representation.

5 DERIVING THE MULTINOMIAL PCA ALGORITHM

1. User specifies model First, the user specifies the model of interest in a high-level specification language (not a *programming* language). This is given in Figure 1, From the specification, the underlying Bayesian network (dependency graph) of the model is derived and represented internally as a directed graph.

2. System observes hidden-variable structure in Bayesian network First the system attempts to decompose the optimization goal into independent parts, but finds that it cannot. However, it then finds that the probability in the initial optimization statement matches the conditions of Lemma 2. Moreover, $U' = \{\vec{k}, \vec{m}\}$ has the same dimensions as the data vector $\vec{x}$ and thus is bona-fide "hidden data".

3. System triggers generalized k-means when EM fails However, the hidden data $\vec{m}$ is continuous and thus cannot be handled by the EM scheme which needs discrete hidden data. The system falls back on a generalized k-means scheme which modifies the optimization problem to introduce the real-valued hidden data into the optimization goal as $\mathrm{pr}(\vec{x}|\vec{\Omega}, \vec{m})$, a known biased method that nonetheless can be used to derive many k-means algorithms. The remaining hidden data is now discrete.

4. System invokes abstract EM-family schema This triggers the EM-schema, whose overall structure is shown. The syntactic structure of the cur-

```
schema(max Pr(U|V) wrt V,C)  :-
   ...
   C = "[initialize W];
        while([converging(V)]){
          /* E-step */ [calculate Pr(W|U,V)];
          /* M-step */ [max Exp_W log Pr(W,U|V) wrt V];
        }"
```

rent subproblem must match the first argument of the schema; if additional applicability constraints (not shown here) hold, this schema is executed. It constructs a piece of code which is returned in the variable C. This code fragment can contain itself calls to other schemas (denoted by [...]) which return code for subproblems which then is inserted into the schema, such as `converging`, a generic convergence criterion here imposed over the variables $V = \{\vec{\Omega}, \vec{m}\}$. The system identifies the discrete variable $\vec{k}$ as the single hidden variable, i.e. $W = \{\vec{k}\}$. For representation of the distribution of the hidden variable a matrix $\vec{q}$ is generated, where $q_{p,f,c}$ is the probability that the f-th feature (i.e., word) in the p-th document falls into the c-th class. AutoBayes then constructs the new distribution `k(I,J)` $\sim$ `discrete(vec(K:=1..n_classes, q(I,J,K))` which replaces the original distribution in the following recursive calls of AutoBayes. The

$$
\begin{array}{l}
\texttt{while}(\texttt{converging}(\vec{\Omega},\vec{m}))\{\\
\quad \texttt{for}\ p=0:n\_points-1, f=0:n\_feats-1\\
\quad\quad \texttt{for}\ c=0:n\_classes-1\\
\quad\quad\quad q_{p,f,c} = Pr(x_{p,f}\,|\,k_{p,f}=c,\vec{\Omega},\vec{m});\\
\quad \texttt{max}\ Exp_{\vec{k}\sim discrete(\vec{q})} \log Pr(x_{p,f},k_{p,f}\,|\,\vec{\Omega},\vec{m})\ \texttt{wrt}\ \{\vec{\Omega},\vec{m}\}\\
\}
\end{array}
$$

freshly introduced distribution for $k_{p,f}$ implies that $k_{p,f}$ can be eliminated from the known variables by summing over $q_{p,f,*}$. This invokes the lemmas given in Section 3.1 to develop formula for the various probabilities.

5. M-step: System recursively decomposes optimization problem AutoBayes is recursively called with the new goal `max` $\log Pr(\vec{x},\vec{k} \mid \vec{\Omega},\vec{m})$ `wrt` $\{\vec{\Omega},\vec{m}\}$. Now, the Bayesian network decomposition schema applies with $U = \{\vec{x},\vec{k}\}$, $V = \{\vec{\Omega},\vec{m}\}$, revealing that $\vec{m}$ is independent of $\vec{\Omega}$, thus the optimization problem can be decomposed into two optimization subproblems: `max` $\log Pr(\vec{k} \mid \vec{m})$ `wrt` $\{\vec{m}\}$ and `max` $\log Pr(\vec{x} \mid \vec{\Omega},\vec{k})$ `wrt` $\{\vec{\Omega}\}$.

6. System unrolls i.i.d. vectors (twice) The first subgoal from the decomposition schema, `max` $\log Pr(\vec{k} \mid \vec{m})$ `wrt` $\{\vec{m}\}$, can be unrolled over the independent and identically distributed vector $\vec{x}$ using an index decomposition schema which moves expressions out of loops (sums or products) when they are not dependent on the loop index. Since $\vec{k}$ and $\vec{x}$ are co-indexed, unrolling proceeds over both (also independent and identically distributed) vectors in parallel. Likewise, the second subgoal from the decomposition schema, `max` $\log Pr(\vec{x} \mid \vec{\Omega},\vec{k})$ `wrt` $\{\vec{\Omega}\}$, can be unrolled over the classes because the expected value over classes effectively makes them independent.

7. System identifies and solves Lagrange multiplier problems Both remaining subgoals are converted to formula optimization problems. For instance for the `n_points` problems of maximizing each documents proportions $\vec{m}$, this reduces to maximizing $P\sum_{c=0}^{C-1}\left(\log(m_{p,c})\sum_{f=0}^{F-1} q_{p,f,c}\right)$ w.r.t. the variables $\{m_{p,c} : c = 0..C-1\}$ under the constraint $\sum_{c=0}^{C-1} m_{p,c} = 1$. This is solved by an application of the Lagrange multiplier schema. This in turn results in two subproblems for a single instance $m_{p,c}$ and for the multiplier which are both solved symbolically. The problem to optimize $\vec{\Omega}$ is handled similarly.

8. System checks and optimizes pseudo-code During the synthesis process, AutoBayes accumulates a number of constraints which have to hold to ensure proper operation of the code (e.g., absence of divide-by zero errors, legal array accesses, etc.). Unless these constraints can be resolved against the model, AutoBayes automatically inserts a run-time check into the code. Before finally generating C/C++ code, the pseudo-code is optimized using information from the specification (e.g., $\sum_c m_{p,c} = 1$) and the domain. Thus, deep optimizations beyond the capability of a regular compiler can be done. The pseudo-code generated for the declarations and initialization is given in Figure 2. This is the intermediate code generated

```
proc(mpca){
  const:
    // nr. of words in each document (n_feats)
    int F := size(x, 2);
    // nr. of documents (n_points)
    int P := size(x, 1);
    double tolerance := 0.003;
  input:
    // nr. of classes (n_classes)
    int C;
    // dirichlet pars
    double alpha[0:C-1];
    // nr. of distinct words (n_words)
    int W;
    // data
    int x[0:P-1, 0:F-1];
  output:
    // doc proportions
   double m[0:P-1, 0:C-1];
    // components
    double omega[0:C-1, 0:W-1];
  local:
    // component for word
    int k[0:P-1, 0:F-1];
    // local distribution used in EM-loop
    double q[0:P-1, 0:F-1, 0:C-1];
  {
    // Initialization
    local:  int p, f, c;
    //  The initialization is class-based; for each point the value
    //  of the hidden variable k is chosen first. Then, values
    //  for the local distribution are calculated.
    for(p=0..P-1, f=0..F-1)
      k(p,f) := random_int(0,C-1);

    //  Initialize the local distribution q;
    //  the initialization is "sharp", i.e., q is set to zero almost
    //  everywhere and to one at the index positions
    //  determined by the initial values of the hidden variable.
    for(p=0..P-1, f=0..F-1) {
      for(c=0..C-1) q(p,f,c) := 0;
      q(p,f, k(p,f)) := 1;
    }
  }
```

Figure 2: Declarations and Initialization

by AutoBayes reformatted with loop variable names changed, and constraint checks as asserts eliminated and some scoping (declarations and blocks) modified in order to fit the page. For instance, apart from this reformatting, the comments are entirely automatically generated. The pseudo-code generated for the core

```
// EM-loop
while( converging([matrix(p=0..P-1,c=0..C-1,m(p,c)),
                   matrix(c=0..C-1,w=0..W-1,omega(c,w))]) ) {
  // Decomposition I;
  // the problem to optimize the conditional probability
  // pr(k,x| m,omega) w.r.t. the variables m and omega can
  // under the given dependencies by Bayes rule be
  // decomposed into independent subproblems.
  // The conditional probability pr(x| k,omega) is under
  // the dependencies given in the model equivalent to
  //    prod(p=0..P-1,f=0..F-1, pr(x(p,f)| k(p,f), omega))
  // The probability occuring here is atomic and can thus be
  // replaced by probability density functions given
  // in the model. After summing out the expected variable ...
  local: int p,f,c;
  for(c=0,C-1) {
    // The expression
    //   C * sum(p=0..P-1, f=0..F-1,
    //           log(omega(c,x(p,f))) * q(p,f,c))
    // is maximized w.r.t. the variables omega(c,*)
    // under the constraint
    //     1 == sum(w=0..W-1, omega(c,w))
    // using the Lagrange-multiplier l.
    local:  double l;
            int w;
    l := C * sum(p=0..P-1, f=0..F-1,
                 cond(x(p,f) <= W-1, q(p,f,c), 0));
    for(w=0..W-1)
      // The function
      //   C * sum(p=0..P-1, f=0..F-1,
      //           log(omega(c, x(p,f))) * q(p,f,c))
      //   - l * sum(w=0..W-1, omega(c,w))
      // is then symbolically maximized w.r.t. the goal
      // variable omega(c,w) by differentiating.
      omega(c,w) := C * sum(p=0..P-1, f=0..F-1,
                            cond(x(p,f)==w, q(p,f,c), 0)) / l;
  }
  // The conditional probability pr(k| m) is under the
  // dependencies given in the model equivalent to
  //    prod(p=0..P-1, f=0..F-1, pr(k(p,f)| m(p,*)))
  // The probability occuring here is atomic and can thus be
  // replaced by probability density functions given
  // in the model. After summing out the expected variable ...
  for(p=0..P-1) {
    // The expression
    //   P * sum(c=0..C-1,
    //           log(m(p,c)) * sum(f=0..F-1, q(p,f,c)))
    // is maximized w.r.t. the variable m(p,*) under
    // the constraint
    //     1 == sum(c=0..C-1, m(p,c))
    // using the Lagrange-multiplier l2.
    local: double l2;
    l2 := F * P;
    for(c=0..C-1)
      // The function
      //   P * sum(c1=0..C-1,
      //           log(m(p,c1)) * sum(f=0..F-1, q(p,f,c1)))
      //   - l2 * sum(c1=0..C-1, m(p,c1))
      // is then symbolically maximized w.r.t. the goal
      // variable m(p,c) by differentiating.
      m(p,c) := P * sum(f=0..F-1, q(p,f,c)) / l2;
  }
  for(p=0..P-1, f=0..F-1, c=0..C-1)
    q(p,f,c) := m(p,c)*omega(c, x(p,f))
      / sum(c1=0..C-1, m(p,c1)*omega(c1, x(p,f)));
 }
}
```

Figure 3: Main EM Loop

loop is given in Figure 3.

9. System translates pseudo-code to real code in desired language Finally, AutoBayes converts the intermediate code into code of the desired target system. The source code contains thorough comments detailing the mathematics implemented. A regular compiler containing generic performance optimizations not repeated by AutoBayes turns the code into an executable program. A program for sampling from a multinomial PCA model is also synthesized for testing purposes.

10. Users tests in Octave using sampler and code In our case, we used the generated sampler to get 1000 sample "documents" each containing 50 words from a vocabulary of 50 words. There are 5 components in the model. The matrix `omega` is generated using Dirichlet sampling, and then the data `x` generated with forward sampling using the synthesized routine. The synthesized statistical algorithm for this converges in in about 5 minutes on a desktop Linux PC over a number of trials, and inspection shows the "truth" is approximately reconstructed. The algorithm tends to zero low probability words in the components, a bias one might expect from this kind of algorithm.

6 DISCUSSION

The derivation given in the previous section is nicely reflected in the code. For instance, Step 6, unrolling the vectors, corresponds to the first two major loops inside the main EM loop, "`for(c=0..C-1)`" and "`for(p=0..P-1)`".

By inspection of the code in Figures 2 and 3, and from knowledge of a comparable implementation there are three major weaknesses:

- Typically documents are variable length, or the bag of words model might be more appropriate. These data types would need corresponding support in the synthesis system.
- The intermediate array `q(_,_,_)` can in fact have its first dimension eliminated. This requires fairly standard code optimization operations on the low-level code (i.e., after the sums have been expended to loops) to discover–loop rearrangements, and code motion. We do not do these currently.
- The sum to recompute `omega` should have a special data structure to compile out the conditional `x(p,f)==w`.

Note each of these are critical for scale-up of the algorithm: the second dramatically reduces space requirements and the third dramatically reduces time. The first two, at least, correspond to improvements we already plan for the system. Note a considerable advantage of the synthesis approach is the high-level of analysis and allowed code modifications which facilitate optimizations generally unthinkable in standard language compilers.

Of course, the algorithm is quite dated by research standards. It is clear to us that adding more sophisticated schemas, in this case for Kullback-Leibler approximations and other general methods now applied to the problem is not difficult, thought time consuming. It is more important at this stage to make the system numerically robust and general purpose with the existing high-level schemas because this is what is required for a complete end-to-end proof that the system can generate code worthy of inclusion, for instance, in embedded software in a satellite.

7 CONCLUSION

From a synthesis perspective, the multinomial PCA algorithm generated demonstrates the ability of the system to combine existing schemas in a manner to overcome the restrictions of any one. We believe this demonstrates our main thesis: good mechanical combination of existing statistical methods is a powerful tool. There are of course several shortcomings in the algorithm generated as well, as discussed above, and these suggest various areas were well-understood modules can be added to the system. Also, just on this one problem, many different algorithms are possible once the newer methods are added to our system, and this makes the tyranny of "choice" a problem to be addressed as well.

References

[1] D. Blei, A.Y. Ng, and M. Jordan. Latent Dirichlet allocation. To appear in *NIPS*14*, 2002.

[2] W.L. Buntine. Operations for learning with graphical models. *JAIR*, 2:159–225, 1994.

[3] W.L. Buntine. Variational extensions to EM and multinomial PCA. To appear in *ECML 2002*.

[4] T. Ellman and T. Murata. Deductive synthesis of numerical simulation programs from networks of algebraic and ordinary differential equations. *Automated Software Engineering*, 5(3):291–319, 1998.

[5] B. Fischer and J. Schumann. AutoBayes: A System for Generating Data Analysis Programs from Statistical Models. *J. Functional Programming*, 2002. To appear. Preprint available at `http://ase.arc.nasa.gov/people/fischer/`.

[6] G.S. Gaffney and P. Smyth. Trajectory clustering using mixtures of regression models. In *5th KDD*, pp. 63–72, 1999.

[7] A. Gray, B. Fischer, J. Schumann and W. Buntine. Automatic Derivation of Statistical Algorithms: The EM Family and Beyond. Submitted, 2002.

[8] P. Haddawy and A.M. Frisch. Modal logics of higher-order probability. In *UAI 10*, 1994.

[9] T. Hofmann. Probabilistic latent semantic indexing. In *Research and Development in Information Retrieval*, 50–57, 1999.

[10] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2):498–519, 2001.

[11] S.L. Lauritzen, A.P. Dawid, B.N. Larsen, and H.-G. Leimer. Independence properties of directed Markov fields. *Networks*, 20:491–505, 1990.

[12] D. Lee and H. Seung. Learning the parts of objects by non-negative matrix factorization. Nature 401, 788–791, 1999.

[13] D.J.C. Mackay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1991.

[14] T. Minka and J. Lafferty. Expectation-Propagation for the Generative Aspect Model In *UAI 2002*, Edmonton, 2002.

[15] K. Murphy. Bayes Net Toolbox 2.0 for Matlab 5, 2000. Internet Report.

[16] P. Smyth, D. Heckerman, and M. Jordan. Probabilistic independence networks for hidden Markov models. *Neural Computation*, 9(2):227–269, 1997.

[17] A. Thomas, D.J. Spiegelhalter, and W.R. Gilks. BUGS: A program to perform Bayesian inference using Gibbs sampling. In *Bayesian Statistics 4*, pp. 837–842, 1992.

[18] D.A. van Dyk. The nested EM algorithm. *Statistica Sinica*, 10:203-225, 2000.