Using Discriminative Graphical Models for Insurance Recommender Systems

Teja Kanchinadam*, Maleeha Qazi*, Joseph Bockhorst*, Mary Y. Morell*, Katie Meisssner* and Glenn Fung*
**Machine Learning Research*
American Family Insurance, Strategic Data & Analytics
Madison, WI, USA
Email: {tkanchin,mqazi,jbockhor,mpeng,kmeissne,gfung}@amfam.com

***Abstract*—Recommender systems have become extremely important to various types of industries where customer interaction and feedback is paramount to the success of the business. For companies that face changes that arise with ever-growing markets, providing product recommendations to new and existing customers is a challenge. Furthermore, it is important to have an algorithm which is descriptive, scalable, agnostic to missing features, and robust in providing these recommendations. Directed graphical models meet all these demands; however, if the dimensionality of the features is high, structure learning and inference can become computationally prohibitive. In this work, we propose an algorithm with some novel aspects to learn the structure of a graphical model (e.g. Bayesian network), which considerably speeds up both training (from days to minutes in some cases) and inference run-times with respect to standard Bayesian structure learning approaches, while achieving similar accuracy. We also show that this approach produces more accurate predictions than a state-of-the-art matrix factorization algorithm in the absence of complete evidence on several insurance-related datasets.**

I. Motivation

In the insurance industry, the motivation for creating a recommendation system stems from the urge to provide value for customers. By connecting to customers through accurate and relevant recommendations, customer trust and loyalty could be increased.

The aim is to predict relevant insurance products for our customers based on what other similar people with similar portfolios have. We want to be able to do this for both our current & prospective customer base. To aid our agents in providing the best value to our customers, we created this recommendation engine to generate recommendations based on customer portfolio data, to give agents a good starting point for discussions with their customers. Currently the human interaction aspect is relied on for the best possible experience for the customer. Future versions of the system will allow for direct customer interaction & offers.

The first iteration of our system [1] that was previously deployed had a limited scope: we focused on 3 of our 19 sales states and each state got 2 models: one for auto, & one for property. Since each model was relatively small, we used a commercial Bayesian network tool to create and score the models: BayesiaLab [2]. Due to the success of our first wave of models, we decided to expand recommendations to all 19 of our sales states in this second iteration of the system. For simplicity, we also decided to create general models that cover all the 19 states at once. Due to the vast amounts of data that had to be considered when creating models at the national level, structure learning and inference became computationally infeasible with BayesiaLab. This motivated us to design a much more scalable approach to Bayesian network (BN) structure learning that better fits our task.

Bayesian Networks belong to a class of algorithms known as probabilistic graphical models. For such models, a directed graph expresses the conditional independence assumptions between random variables, which are represented as nodes, using the (lack of) arcs between nodes. They provide a compact representation of joint probability distributions. The advantage of BNs is that they can encode variable relationships that can be learned from data. Furthermore, the complex relationships embedded in the data can be visualized using the graphical representation of the models. From the machine learning perspective, BNs belong to the class of generative models, because they learn the joint probability distribution of features and target variables together.

Training a BN usually involves two steps. First, we learn the underlying graph structure. This can be done by learning it from the available training data or by using domain knowledge about the problem at hand [3, 4]. The former is used in this paper. The second step involves learning the conditional probability tables (CPTs) that represent the conditional distribution for each variable given its parents.

Learning the graph structure is usually the hardest part in this process. In general, this problem is NP-hard [5]. Even learning the structure using a greedy approach could result in a dense graph. The corresponding CPTs, whose size grows exponentially with the number of parents, explode in size, potentially making inference or even simply realizing the CPT in memory intractable.

Taking these two things into consideration, we propose an algorithm which learns a simpler structure in a way that both structure learning and inference are scalable and can be performed much faster. Our algorithm reduces the structure learning time from days (5) to minutes (20) for most of our models. Furthermore, in terms of accuracy this approach is on par with that of state-of-the-art recommendation algorithms.

There are several characteristics of our problem that made the BN approach a good fit in our case:

- Most of our features, derived from policy and customer data, have discrete values.
- Traditional recommendation systems deal with many products (e.g. Netflix, Amazon, etc.), but in our case we had a limited number of products to work with.
- We had a number of missing values in our dataset. BNs can predict in the presence of very little evidence if necessary. This gives us the best of both worlds - generative and discriminative.
- We could optimize the structure to predict the targets we chose, and only consider the top features relevant to each target as its supporting evidence.

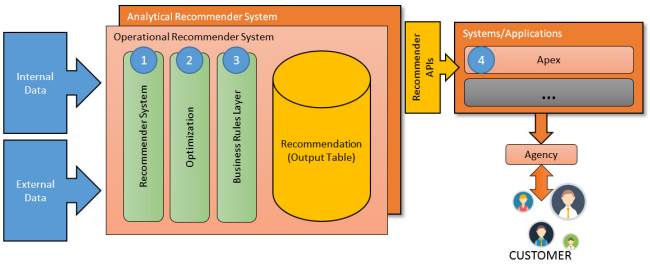

Fig. 1. Schema of our deployed recommender system (APEX is the agent system/interface)

II. Related Work

Some documented instances of recommender systems for the insurance domain include [6], [7], and [8], and for the financial industry include [9]. None of these systems have the same breadth of product/policy knowledge being tackled by our system (auto, property, life, and umbrella policies), and they have limited scope for usage (targeted to call center reps with limited knowledge of products [6], or web usage for a single product [8]). Reference [7] suggests that their system is based on a standard user-user collaborative filtering approach, which would not handle missing values as well as our approach does. Reference [9] has a high degree of interaction required to create recommendations for a customer, whereas ours is meant to run as a batch process on a fairly regular schedule without user input to create recommendations for humans to act on during their business day without delay (HILDA).

Low Rank Matrix Factorization (LRMF) models are widely used algorithms for recommender systems. We experimented with the Boosted Inductive Matrix Completion algorithms (BIMC) as described in [10]. We have applied these methods on our dataset, which is described in section IV, even though these methods take "side information" (i.e. non-target features/variables) into account in an inductive fashion to deal with the cold-start problem, they don't have the mechanism to take other "label information" (i.e. target features/variables) into account without having seen the example during training (transduction). Hence they perform poorly on the new unseen cases with missing "side information".

A paper that compared some collaborative filtering algorithms [11] showed that for a wide range of conditions, Bayesian networks outperformed other methods. Even though [11] is from 20 years ago and many advancements have been made to matrix factorization approaches, the paper highlights that depending on the nature of the data and the application, BNs have smaller memory requirements & allow for faster predictions, but require a learning phase that can take a significant amount of time (multiple hours, depending on the amount of data and features). The approach detailed in this paper is geared to reducing the burden of the learning phase as well as the inference phase of using BNs.

As explained earlier, learning a BN usually starts with learning the graph structure from the data, if it's not already known, and there has been a great deal of research done in finding the optimal Bayesian structure from data. But even under a wide variety of assumptions, this problem is known to be NP-hard [5, 12]. The algorithm in [13] maximizes the BDeu scoring function and finds the optimal Bayesian network, but is only feasible if the number of nodes/variables is in the tens (less than a hundred). In our insurance-inspired settings the number of variables usually exceeds that.

Exact structure learning algorithms like [14] and [15] have exponential run-times. The algorithm presented in [16] tries to find the posterior probabilities of structural features, and finding the optimal Bayesian structure is again restricted to a small number of variables (less than 50) & is not scalable beyond that. [17] tries a greedy approach to find the optimal Bayesian network, but if the data set is large, the run-time is exponential. The algorithm in [18], which finds the maximum spanning tree over a given set of variables, is a very fast approximation of the structure with the maximum in-degree of 1, but it gives very poor results during inference. Given that our dataset has close to 100 variables to start with, none of these algorithms can be used as-is.

III. Methodology

The following sections will describe our deployed system, and the algorithm we chose to use for creating the recommendations.

A. Deployed System Overview

Figure 1 shows the conceptual schema of our deployed recommender system. Our system consists of the following modules, executed in-order:

1) **The recommendation module:** This is the core recommender model, it takes the input customer data (portfolio, policy, etc.) and generates predictions for likely recommendations.
2) **The optimization module:** This takes as an input a set of possible recommendations from the recommendation module and prioritizes recommendations based on an optimization criteria predefined by the business. Possible choices for optimization strategies are: customer retention, customer satisfaction, & customer lifetime value (prediction or estimation of the net profit attributed to the entire future relationship with a customer).

3) **The business rule module:** This module filters recommendations according to business rules to ensure that the final recommendations are appropriate for the customer. For example, it filters out a recommendation if a similar one has been presented to the customer recently (this is based on the system's business requirement).

The final goal is to have the system deployed as a service that can be used in different company applications: as a tool to assist agents (our primary focus in this paper), as a customer facing recommendation engine, and as a tool to assist our marketing department, among others.

The remaining paper focuses on the algorithm we built for the recommendation module (labeled as "1" in Figure 1).

B. Model Scope & Data

For this second phase of the project, we wanted to expand the original scope of 3 sales states to all 19 sales states. And we also wanted to accommodate both our "Classic" and "Advance" line of products. But we kept our original decision to create separate auto and property models, due to the fact that the model features are line specific. This led to the creation of 4 models: 2 for auto, and 2 for property (i.e. each had a Classic and Advance version). All recommendations are made at the policy level.

The purpose of the system for this phase is to suggest new product offers (for both cross-sell and up-sell) for every existing customer. "Cross-sell" means a model recommends a different product line (e.g. auto model recommends property, life, or umbrella coverage), vs. "up-sell" which means a model recommends an additional coverage or endorsement in the same product line (e.g. property model recommends ID fraud coverage).

Both the property and auto models have 6 cross-sell targets/ products. The final recommendation for 5 of the 6 cross-sell targets that have predictions coming from both the auto and property models is made by combining both predictions in an optimal way.

In addition, each one of the 4 models (Classic auto, Classic property, Advance auto, Advance property) have between 3 to 7 up-sell targets/products (e.g. rental reimbursement, new car replacement coverage, sewer/sump pump overflow coverage, ID fraud coverage, etc.).

The auto data consisted of coverage details (limits, deductibles, discounts, etc.), customer demographics (age, gender, marital status, etc.), household characteristics, and various risk factors. The property data consisted of policy data, coverage details, options and endorsements, dwelling characteristics, household characteristics, customer demographics, sewer and wildfire risk, and geological data. In addition, we used other related policy indicators.

The data for this project came from multiple disparate sources, and the first task was to consolidate the data into a single database. The next step included data source analysis, raw/derived feature analysis, business work-flow analysis, statistical data checks, etc. to determine the features that are most likely to be available at the time of execution and contained the most relevant information for the task at hand.

Our data contained both discrete and continuous features. Some of the continuous features were manually discretized based on business logic, but most were binned using the K-Means algorithm [19]. Due to the nature of the data, missing values were kept as separate states for each node because their absence was considered significant.

C. Notation Used

Consider a set of insurance policies $P = \{p_1, p_2, \ldots, p_m\}$. For each policy $p_i \in P$, we have:

- a set of features $\{f_{i1}, f_{i2}, \ldots, f_{in}\}$. The features are characteristics about the policy / policy holder, e.g number of cars, type of house, premiums, policy holder characteristics, etc. Let's define a matrix $F \in \mathbb{R}^{m \times n}$ where the rows are the feature vectors for each one of the m policies.
- a set of targets or coverages $\{t_{i1}, t_{i2}, \ldots, t_{iz}\}$. Let's also define a matrix $T \in \mathbb{R}^{m \times z}$ where the rows are the coverages vectors for each one of the m policies.

An important consideration of our domain is that both the features and targets are frequently incomplete (i.e. have missing values). Consequently, we require a recommendation approach that is robust to missing data. Let $\tilde{F}$ and $\tilde{T}$ refer to incomplete versions of F and T. We aim to learn a recommender function, or classifier, r such that $r(\tilde{F}, \tilde{T}) \rightarrow \mathbb{R}^{m \times z}$, minimizes $N(r(\tilde{F}, \tilde{T}) - T)$ for a given norm N.

Note that we will use the notation F_i to refer to feature i in a general way, or to refer to the column i of F that represents a column vector that contains all the values for that feature for all m policies depending on the context. We will do the same for the targets T_i.

D. Recommendation problem as a Bayesian Network

Our goal is to use graphical models, specifically BNs, to model the joint probability distribution $\Pr(F, T)$ from the available training data. If we constrain a feature variable to have a single target variable as its parent the joint distribution factors as

$$\Pr(T)\Pr(F|T) = \Pr(T)\prod_{i}^{n}\Pr(F_i|T_{\text{par}(i)})$$

where $T_{\text{par}}(i)$ is the parent of F_i.

A key inference task for recommendation is to compute the posterior distribution for a target given evidence $\mathbf{e} = \{\mathbf{e}_T, \mathbf{e}_F\}$, where $\{\mathbf{e}_T, \mathbf{e}_F\}$ represent the observed targets and features respectively. Similarly let $\mathbf{h} = \{\mathbf{h}_T, \mathbf{h}_F\}$ represent the unobserved (or hidden) targets and features. Without loss of generality let our query variable be T_i. This gives the following

equation for single target inference:

$$
\begin{aligned}
Pr(T_i|\mathbf{e}) &\propto \Pr(T_i, \mathbf{e}) \\
&= \sum_{\mathbf{h}} \Pr(T_i, \mathbf{e}, \mathbf{h}) \\
&= \sum_{\mathbf{h_T}} \Pr(T_i, \mathbf{e_T}, \mathbf{h_T}) \sum_{\mathbf{h_F}} \Pr(\mathbf{e_F}, \mathbf{h_F}|T_i, \mathbf{e_T}, \mathbf{h_T}) \\
&= \sum_{\mathbf{h_T}} \Pr(T_i, \mathbf{e_T}, \mathbf{h_T}) \sum_{\mathbf{h_F}} \prod_{i \in \mathrm{c}(i)} \Pr(f_i|T_i) \\
&\qquad * \prod_{i \notin \mathrm{c}(i)} \Pr(f_i|t_{\text{par(i)}})
\end{aligned} \tag{1}
$$

where $\mathrm{c}(i)$ are the children of T_i. The second line uses the fact that the features are children of the targets and the third line results from the one parent per feature constraint.

Given input features F and output targets T, and a number of training examples P, a *generative model* explicitly models the joint probability distribution $Pr(F, T)$ and then uses the Bayes rule to compute $Pr(T_i \mid F)$ (when making predictions). A *discriminative model* would instead directly model $Pr(T_i \mid F)$. This allows a generative model to perform better with missing data. In our domain where we want to use the same models to give recommendations for both existing customers (for whom we will have some background data) and prospective customers (for whom we'd have very little or no data), BNs give us the best combination of generative and discriminative. They are generative models by definition, but since we query them to create recommendations, we are essentially using them in a discriminative fashion.

There are various ways to do inference, they all fall into two categories: exact or approximate inference. Both categories are expensive in different ways [20]. Exact inference is very sensitive to topology [21]. Approximate inference, especially Monte Carlo and related sampling approaches, can take a long time to converge [22]. Very complex networks are problematic to use for a production system because they cannot scale easily. Our method creates relatively simple BN structures, while not compromising on accuracy for predictions. The BN structure among the target variables is unconstrained, but non-target variables have Naive Bayes-like constraints in that all non-target variables have a single target variable as its parent. This allows for a more tree-like structure than would result if all variables had an unconstrained BN structure. This allows the resulting model to scale well for production level inference.

During BN structure learning we allow for the injection of prior knowledge of what the structure of the network should look like via the use of *white-* & *black-lists*. For our algorithm, *white-lists* are edges that are always included in the network & *black-lists* are edges that are never included in the network.

E. Performance metrics

There are several metrics often used to measure performance related to recommender systems [23]. One of the more widely used is precision and recall @k, this metric assumes that the recommendation problem can be interpreted as a ranking problem. In our case, since our number of targets is relatively small (in the tens instead of hundred of thousands) and our recommendations are binary in nature (e.g. recommend coverage A: y/n), our recommender problem can be related to a binary multi-label classification problem. Hence, a simple summary measure of binary precision/recall will be used: Receiver Operating Characteristic, or ROC curves [24] (we report Area Under the ROC curve - i.e. AUC - in the following sections).

Algorithm 1 Discriminative Graphical Model

Input: Matrix of Targets T - $\{T_1, T_2, \ldots, T_z\}$
Matrix of Features F - $\{F_1, F_2, \ldots, F_n\}$
Output: Directed Acyclic Graph (Structure)
BN = Learn joint probability distribution of $\{T_1, T_2, \ldots, T_z\}$ using the search and score method described in [25]
for $i = T_1$ to T_z **do**
Do feature selection for target i as described in section III-G2:
top-k = RF_feature_selection($\{F_1, F_2, \ldots, F_n\}, i$)
if any(top-k in BN) **then**
duplicate the features which are already present
end if
add the top-k features to the BN in a Naive Bayes fashion.
end for
return BN

Fig. 2. Algorithm for Discriminative Graphical Model

F. Algorithm Overview

Our proposed algorithm can be explained in two steps: first, it learns the underlying graph structure between the targets using any off-the-shelf structure learning algorithm, in our case we use the implementation proposed in [25]. Then for each target, we learn the best k features using any off-the-shelf feature selection algorithm and attach them to the target node in a Naive Bayes fashion, as shown in Figure 4. Once we have the final graph structure, we learn the conditional probability tables for each node in the graph using an off-the-shelf CPT learning algorithm. The summary of the proposed structure learning algorithm is presented in Figure 2, and a more detailed explanation of these steps is presented in subsequent sections.

G. Structure Learning

The main idea of this step is to restrict the space of possible graph configurations, so as to reduce the complexity considerably, while finding an approximation to the joint distribution $Pr(F, T)$ that works well for the task at hand. By doing this we can avoid dealing with the daunting task of exploring the entire search space like a general purpose graph structure approximation algorithm would face, which can be

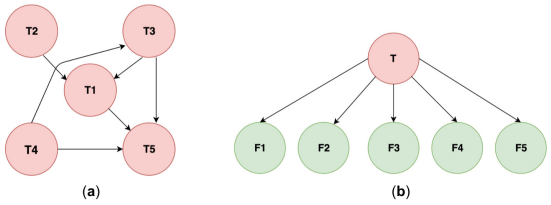

Fig. 3. (a): Sample structure among targets, (b): Naive Bayes graph connecting features and target.

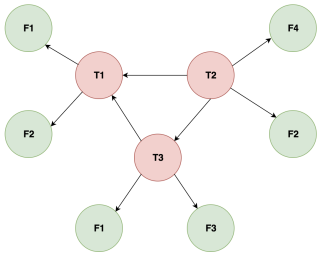

Fig. 4. This illustrates the structure described in section III-G. For example let's assume {T1,T2,T3} as targets and we choose to pick the top-2 features for each target. Notice how we are duplicating the node *F1* for {T1,T3} and *F2* for {T1,T2}

computationally expensive. Instead, we reduce the problem to the two sub-tasks described here.

1) Learning Structure Between Targets: To generate more accurate predictions it is important to understand and model the correlation between the targets as accurately as possible. As stated before, in our case the number of targets is relatively small (less than 20), hence learning the full joint conditional distribution among them is not computationally expensive.

We use the search and score method [26] implemented in [25] to find an approximation to the joint distribution of the targets $Pr(T)$. The search and score method transforms the Bayesian network structure learning problem into a shortest path problem over an *order graph*. This order graph is a lattice made up of layers of variable sets from the Bayesian network structure learning (BNSL) problem, with the root node having no variables, the leaf nodes having all variables, and layer i in the lattice having all subsets of variables of size i. Each path from the root to a leaf represents a certain topological sort of the variables, with the shortest path corresponding to the optimal topological sort and BN.

The resulting graph structure, which approximates the joint distribution of the targets, can have a more complex structure (see Figure 3 (a)) and it is used in the final graph construction discussed in the next section.

2) Feature Selection with Random Forests: The ensemble voting procedure of random forests allows for the calculation of an importance score for each variable in the model. There are many ways to score variable relevance, here we have used *Gini Impurity* as the variable importance score to rank features and select the more relevant ones for each target's discrimination task. Under this structure discovery framework, the feature selection problem is highly related to the problem of finding the Markov blanket for random variables when building a graphical model [27]. Our approach aims to find a simple approximation to the Markov blanket for every target in our BN, such that inference will be computationally tractable.

We construct our graph by starting with the structure between targets learned in section III-G1. Then taking the *top-k* features associated to each target, obtained by the process explained in section III-G2, and connecting them to each target in a *Naive-Bayes* fashion as illustrated in Figure 3 (b). By doing this the *top k* features attached to each target are considered independent of each other during inference, making the inference process more computationally tractable. However, we have observed that for some cases, even though constructing the graph in this manner will make the training time faster, the inference time can still be expensive. This is because a single feature may be important to more than one target, and so could have many incoming edges which leads to a complex graph. This in turn makes the inference costly because the CPT explodes in size.

3) Adding Evidence by Duplicating Features: In order to address the above problem, we made duplicate copies of features and attached them to each target separately (as shown in Figure 4). Since we are setting up the features to be independent of each other for a given target, we argue that making multiple copies of the same feature will have a similar effect as having one feature with multiple relationships. Especially since we are never interested in querying, or calculating posterior probabilities for, feature/non-target nodes. Furthermore, with this setup we can have a simple acyclic graph, which allows for faster inference time.

H. Inference

For BNs calculating posterior marginals on nodes is known to be NP-hard [28]. In our experiments, we have used *loopy belief propagation* [28], which calculates true posteriors for singly connected networks, and calculates approximate marginals on a network with loops.

IV. Experiments

A. Training of the Deployed Models

For each model, the corresponding training dataset was stratified split, based on sales state, into 70/30 train/test sets. As can be seen from Table I, there was much more data for the Classic models than the Advance models due to the fact that the Classic line is legacy, and the Advance line only recently (roughly 5 years ago) got fully deployed to all states. When building the models, we started by running our algorithm for various values of k (globally set to 2, 5, 7, & 10), to evaluate how this would impact the train/test AUCs for each target. By comparing the average AUC value per target for each value of k we determined the ideal setting of k for each target. The final models were then created with the local values of k per target that would give the best AUC. Sometimes additional iterations were necessary to refine the discretization bins for

TABLE I
DETAILS OF DATASETS

Line	Total Policies (all states)	Train Set	Test Set
Advance Auto	691,763	484,232	207,531
Advance Prop.	192,633	134,844	57,789
Classic Auto	2,528,391	1,769,872	758,519
Classic Prop.	1,754,699	1,228,287	526,412

some features/targets to help boost the AUCs. Discretization was done as described in Section III-B.

B. Pre-deployment Off-line Validation

This section focuses on reporting empirical results and comparisons for one of the deployed models: *Advance Auto*. In order to validate our approach, we compared our proposed approach to:

(i) The **Inductive Matrix Completion (IMC)** algorithm proposed in [10], which takes two matrices: a policy $\times$ product matrix and a feature matrix to learn two low rank matrices which can be used in a inductive fashion during testing time to provide recommendations for all the products in a policy.

(ii) Learning the complete (approximated) joint structure (features and targets) using the **Chow-Liu** algorithm [18] which aims to learn a fast approximation of the BN structure (joint probability distribution) in the form of a first-order dependency tree. This algorithm is very fast since it widely restricts the search space for candidate structures to approximate the joint distribution. Since Chow-Liu return a dependency tree with a maximum in-degree of 1, it is more difficult to enforce the business logic (white and black edge lists). The results shown in the table are hence unconstrained with respect to business logic.

For all methods, We have used hyper-parameter tuning using a validation set. More concretely we compared the performance of the three algorithms in the following scenarios:

1) **Comparisons of performance for all targets using IMC vs our BN approach:** results can be seen in Table II. Note that our proposed BN approach can make recommendations in a inductive way taking into account not only the available features but also current coverages as evidence. This is not the case with IMC, where once the model is trained recommendations for an unseen customer are made taking into account only the features for all possible coverages. Notice that the performance of our approach is equivalent to

TABLE II
PERFORMANCE COMPARISON FOR ALL TARGETS (AVERAGE AUC) BETWEEN IMC, CHOW-LIU AND OUR PROPOSED ALGORITHM.

IMC	BN	Chow-Liu
0.798	**0.800**	0.773

IMC in average (outperforming IMC in 7 out of 13 targets) and both of the approaches significantly outperformed the Chow-Liu approach. However as we will see below there are other advantages of using our proposed algorithm.

2) **Comparisons of performance when we have missing features:** Here we compare performance for IMC vs our BN approach when we randomly remove features as if they were missing. Note that we don't have to process the data differently with the BN approach since the features can be treated as unobserved evidence during inference. In contrast, the IMC approach has to impute the missing values in order to produce a prediction (mean imputation in our case). Results can be seen in Figure 5. The x-axis represents the percentage of missing features and the y-axis is the AUC on the testing set. For this experiment, in order to be fair with IMC, we are predicting all the targets at the same time, this means that no targets are observed at the moment of prediction, so our algorithm performs a slightly worse when only the features are observed since it is optimized to use both the features and the observable targets as evidence. Figure 5 shows that the proposed BN algorithm performance degrades much more slowly when features are missing and even produces reasonable predictions in the absence of almost all the features (98%). This characteristic is very important when making recommendations to prospects on our website where the data collected for recommendations can be scarce.

3) **Performance degradation of the BN approach when targets are unobserved:** results can be seen in Figure 6. The x-axis represents the number of targets scored (unobserved) at a time and the y-axis represents the average AUC on the testing set. For this experiment we are assuming that all features are observed. As discussed earlier in the section, if we want to use our BN models to recommend newer products to an existing customer based on their existing products, we can use existing product information as evidence during inference. As was the case with missing (unobserved) features, note that our algorithm degrades very gracefully with respect to unobserved targets as well. The algorithm performance ranges from around 80% to 75% average AUC, depending on the number of observed targets. Note that IMC never uses existing product information for inference and relies completely on features and it is optimized to only use them for predictions. For our deployed systems we are generating recommendations for each missing product in the portfolio observing all the other products already in the portfolio (one-at-a-time). However this may not be the case for other applications of this recommendation system inside the company.

4) **Training time and Inference time Comparisons:** It is evident from the above comparisons that the performance of our BN approach is at par to existing state-of-the-art algorithms such as matrix factorization. However, our algorithm is much faster than the existing BN approaches. On a 16GB OSX machine, the running time to train our model over 96 variables is 25 minutes. The matrix factorization approach took over 4 days to train and tune on the same dataset and architecture (tuning used 16 combinations, i.e. 360 min/combination), whereas Chow-Liu structure learning took 10 minutes to train (Table III). We were never able to

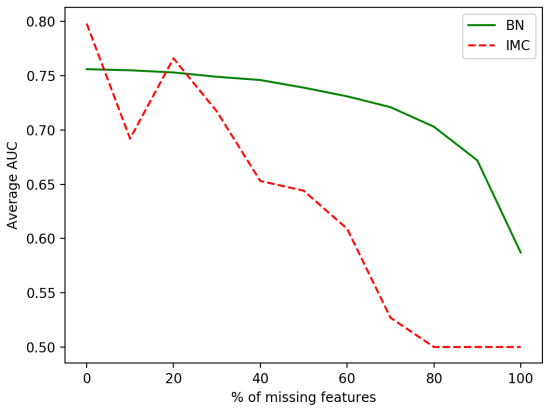

Fig. 5. Comparison between IMC and our algorithm scoring all the targets at once with missing features.

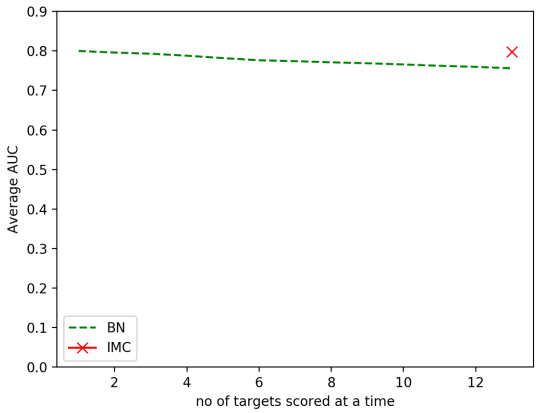

Fig. 6. Performance of targets when scored k-at-a-time using our algorithm.

train our model using exact structure learning approaches from [26] and greedy approach from [17] because the computer ran out of memory, and on a different machine we killed the process after 5 days (since it exceeded our time limits).

Our approach is also scalable when it comes to inference run-time. The inference run-time using loopy belief propagation, on a 16 GB OSX machine, and the model trained using the Chow-Liu approach for 100 records is 45 seconds. For the same setting, using our trained model, the inference time is 7.15 seconds which is approximately 7 times faster. The run-time for matrix factorization is 0.15 seconds, however, computationally speaking this is a different approach than the BN one.

For production, we tested our model on our Hadoop cluster which is running Spark, and for 700,000 records, the run-time was approximately 2 hours. This approximation is an upper bound because there are always high-priority production jobs running in this environment and the number of resources that were dedicated to our test were dynamically allocated.

TABLE III
RUN-TIMES FOR TRAINING AND INFERENCE FOR VARIOUS APPROACHES IN A DETERMINISTIC ENVIRONMENT. INFERENCE IS PERFORMED ON 100 RECORDS.

RunTime	**IMC**	**BN**	**Chow-Liu**
Training	4 days	**25 mins**	10 mins
Inference	**0.15 s**	7.15 s	45 s

C. *Post-deployment On-line Validation*

This second phase of our system was released to all 19 sales states, with Classic roll-out in Nov 2017 and Advance roll-out in Feb 2018. We evaluated this phase of our deployed system using a test-and-learn methodology [29].

The analysis performed was an Agent-level Natural Experiment that measures the incremental lift in sales / performances between agents who are actively engaged in utilizing the recommender system (RS) to similar agents who are not actively engaged, but have access to the RS. Natural Experiments are observational studies where randomization of test and control groups was not accomplished [30]. Synthetic controls are selected through a matching process to minimize differences between test and control groups.

The test agents were a subset of hundreds of agents who were "active" users of the system as of March 13, 2018. Here, "active" is defined as having a click rate, on recommendations generated by RS, greater than or equal to 1 per business day between launch and March 2018. The RS system was released to all agents across all states in Nov 2017.

Each test agent was matched to a varying number of control agents to ensure most similar controls are selected and less similar controls are not forced into being in a test agent's control group.

The analysis was performed at the monthly grain, and the timeframe was defined taking into account data from 12 months prior to the program launch and 3 months post system launch.

The analysis method, Difference-in-Difference (D-I-D) [31], makes the underlying assumption that whatever happened to the control group over time is what would have happened to the treatment group in the absence of the program.

The analysis results are based on both Classic & Advance since the deployed models where built for both lines. For all lines, we saw statistically significant increases in PIF counts and written premiums, respectively. No statistically significant differences between test and control groups for new policy counts and counts of policies attrited were detected.

We also looked at the umbrella product (i.e. additional liability coverage above the limits of your homeowners and auto insurance policies) separately and saw statistically significant increases for new product counts (9.18% lift). For this product we also had PIF counts and statistically significant decreases for counts of policies attrited (-9.54% lift; here negative is a good value). While directional in nature, we saw an increase

in written premiums as well.

V. Conclusions and Future Work

In summary, we have proposed, implemented and deployed a BN-based recommendation system framework that has several advantages including:

1) Faster training (structure discovery) and inference with respect to other approaches. We achieve training acceleration of the order of $\sim 700x$ (5 days for traditional BN structure learning to 25 minutes with our algorithm). This is paramount for future RS implementations where the number of possible recommendations can be expanded from best products to offer to best next actions (customer engagement, discounts, etc). This will also be an important factor if we want our RS to update frequently based on our daily interactions with our customers.
2) Dealing gracefully with missing values. This is very relevant for recommendations to prospective customers where customized recommendations can be generated on the fly given the available information about the potential insured.
3) Our method is scalable both for training and inference, and can be easily implemented in a distributed fashion.

During this process we also compared our results to some deep learning approaches, and we are still working to refine those models for the future.

References

[1] M. Qazi, G. M. Fung, K. J. Meissner, and E. R. Fontes, "An insurance recommendation system using bayesian networks," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, ser. RecSys '17. New York, NY, USA: ACM, 2017, pp. 274–278. [Online]. Available: http://doi.acm.org/10.1145/3109859.3109907

[2] "Bayesialab," http://www.bayesia.com/, versions: 5.4.3 & 6.0.2; Accessed: 1-Feb-2017.

[3] "Integrating expert knowledge with data in bayesian networks: Preserving data-driven expectations when the expert variables remain unobserved," *Expert Systems with Applications*, vol. 56, 2016.

[4] "A bayesian network decision model for supporting the diagnosis of dementia, alzheimers disease and mild cognitive impairment," *Computers in Biology and Medicine*, vol. 51, 2014.

[5] D. M. Chickering, D. Heckerman, and C. Meek, "Large-sample learning of bayesian networks is np-hard," *Journal of Machine Learning Research*, vol. 5, no. Oct, 2004.

[6] L. Rokach, G. Shani, B. Shapira, E. Chapnik, and G. Siboni, "Recommending insurance riders."

[7] B. P. Sanghamitra Mitra, Nilendra Chaudhari, "Leveraging hybrid recommendation system in insurance domain," *International Journal of Engineering and Computer Science*, vol. 3, Oct 2014.

[8] A. Gupta and A. Jain, "Life insurance recommender system based on association rule mining and dual clustering method for solving cold-start problem," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, Oct 2013.

[9] A. Felfernig and A. Kiener, "Knowledge-based interactive selling of financial services with fsadvisor," in *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence - Volume 3*, ser. IAAI'05. AAAI Press, 2005.

[10] N. Natarajan and I. S. Dhillon, "Inductive matrix completion for predicting gene-disease associations," *Bioinformatics*, vol. 30, jun 2014.

[11] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[12] D. M. Chickering, "Learning bayesian networks is np-complete," in *Learning from data*. Springer, 1996.

[13] A. P. Singh and A. W. Moore, "Finding optimal bayesian networks by dynamic programming," 2005.

[14] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine learning*, vol. 9, no. 4, 1992.

[15] N. Friedman and D. Koller, "Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks," *Machine learning*, vol. 50, no. 1-2, 2003.

[16] M. Koivisto and K. Sood, "Exact bayesian structure discovery in bayesian networks," *Journal of Machine Learning Research*, vol. 5, no. May, 2004.

[17] D. M. Chickering, "Optimal structure identification with greedy search," *Journal of machine learning research*, vol. 3, no. Nov, 2002.

[18] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE transactions on Information Theory*, vol. 14, no. 3, 1968.

[19] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[20] H. Guo and W. Hsu, "A survey of algorithms for real-time bayesian network inference," in *Join Workshop on Real Time Decision Support and Diagnosis Systems*, 2002.

[21] R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media, 2006.

[22] A. Doucet, N. De Freitas, and N. Gordon, "An introduction to sequential monte carlo methods," in *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 3–14.

[23] A. Gunawardana and G. Shani, "A survey of accuracy evaluation metrics of recommendation tasks," *J. Mach. Learn. Res.*, vol. 10, Dec. 2009.

[24] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.

[25] J. Schreiber, "pomegranate," https://github.com/jmschrei/pomegranate, 2016.

[26] C. Yuan, B. Malone, and X. Wu, "Learning optimal bayesian networks using a* search," in *IJCAI proceedings-international joint conference on artificial intelligence*, vol. 22, no. 3, 2011, p. 2186.

[27] T. Gao and Q. Ji, "Efficient markov blanket discovery and its application," *IEEE Transactions on Cybernetics*, vol. 47, no. 5, pp. 1169–1179, May 2017.

[28] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999.

[29] D. C. Montgomery, *Design and Analysis of Experiments*. John Wiley & Sons, 2006.

[30] E. A. Stuart, "Matching methods for causal inference: A review and a look forward," *Statist. Sci.*, vol. 25, no. 1, 02 2010.

[31] "Difference in difference estimation," https://www.mailman.columbia.edu/research/population-health-methods/difference-difference-estimation, accessed: 30-Apr-2018.