Scalable Multidimensional Hierarchical Bayesian Modeling on Spark

Robert Ormandi, Hongxia Yang and Quan Lu

Yahoo!
Sunnyvale, CA

2015
Click-Through-Rate (CTR) Prediction

- Estimating the probability of *click* given
  - The *user* who opens
  - a *publisher’s page* with
  - an *advertiser’s ad* in it.

- That is, estimating

\[ P(\text{click}|\text{user}, \text{publisher}, \text{advertiser}) \]

- Used for
  - selecting the most promising ad at serving time,
  - calculating the *bid price* in a second-price auction.
Challenges

- **Extreme Sparsity**
  - Huge amount of system level observations, but very few at the individual user, publisher or advertiser level

- **Heavily Imbalanced Click-Non-click Ratio**
  - Huge amount of system level observations, but very few at the individual *user, publisher* or *advertiser* level

- **Highly Interdependent Data**
  - Known hierarchies, e.g. Advertiser → Campaign → Ad; Exchange → Publisher → Page; etc.

- **Sampling Bias, Heavy-Tailed Distributions, Dynamic Environment, etc.**
Our Contributions—Highlight

- Model MadHab:
  - Bayesian which
  - jointly models multiple dimensions (*user*, *publisher* or *advertiser*); and
  - on multiple level of resolution (hierarchical).

- Algorithm MadHab-Spark:
  - efficient (in-memory); and
  - highly scalable (distributed).
The model is Bayesian, multidimensional and hierarchical. Why these?

- Hierarchical: The original data is organized in hierarchies (e.g. Advertiser $\rightarrow$ Campaign $\rightarrow$ Ad). We want to leverage this by “borrowing information” from multiple directions in the hierarchy of the model through the posterior sampling.
- Bayesian: We are interested in the whole and true posterior distribution to support e.g. the bid price calculation.
- Multidimensional: We want a joint model for better handling of the extreme sparsity.
Main Assumptions:

- The clicks follow a Bernoulli distribution:

\[ \text{click}_{\text{user}, \text{page}, \text{ad}} \sim \text{Bernoulli}(q_{\text{user}, \text{publisher}, \text{advertiser}}) \]

- The CTR follows a Beta distribution:

\[ q_{\text{user}, \text{publisher}, \text{advertiser}} \sim \text{Beta}(c \cdot q_{\text{user}}q_{\text{publisher}}q_{\text{advertiser}}, c(1 - q_{\text{user}}q_{\text{publisher}}q_{\text{advertiser}})) \]

- That is, the CTR distribution is a rank-one tensor decomposition of the three component-wise latent CTRs.

- The latent CTRs are learnt as well with their own model structure.
Madhab—Hierarchical Beta Prior

- Modeling the latent CTR for the \textit{advertiser} and \textit{publisher} components, we assume two \textit{Hierarchical Beta Prior} structures:
  - \textbf{Advertiser Component}:
    
    \[
    q_{ad} \sim \text{Beta}(c_3 \cdot q_{line}, c_3(1 - q_{line}))
    
    q_{line} \sim \text{Beta}(c_2 \cdot q_{campaign}, c_2(1 - q_{campaign}))
    
    q_{campaign} \sim \text{Beta}(c_1 \cdot q_{advertiser}, c_1(1 - q_{advertiser}))
    
    q_{advertiser} \sim \text{Beta}(c_0 \cdot q_a, c_0(1 - q_a))
    \]
  - \textbf{Where}:
    - \textit{campaign} denotes a campaign of the advertiser \textit{advertiser},
    - \textit{line} a targeting bucket of the campaign \textit{campaign} and
    - \textit{ad} is an ad assigned to the line \textit{line}
  - \textbf{resulting a tree}.
  - \textbf{For the publisher dimension we have a similar structure with the following levels}: exchange, publisher and page
User Component: we have a mixture model on the top of a predefined clustering with a logit link function

\[
\text{logit}(q_{\text{cluster(user)}}) \sim \text{Normal}(x_{\text{cluster(user)}}^T \cdot \beta_{\pi_j}, U_{\pi_j}^2)
\]

Where

- \(\text{cluster(.)}\) denotes the predefined clustering of the users,
- \(\pi_j\) is the latent mixture index variable, and
- \((\beta_j, U_j^2)\) are the parameters of the \(j\)th latent component.
Madhab—The Model

- Multi-resolution (hierarchies),
- Multidimensional
  - User: A mixture of Gaussians with a logit link function
  - Advertiser: Hierarchical Beta Prior
  - Publisher: Hierarchical Beta Prior

- Bayesian model which jointly models the above defined dimensions
- by introducing component-wise latent CTRs along with their model structure.
Fitting the Model

- Applying MCMC-based posterior sampling (recap):

\[
\theta^{(0)} = (\theta_1^{(0)}, \ldots, \theta_K^{(0)}) \leftarrow \text{arbitrary}
\]

\[
\text{foreach } t \rightarrow \infty, i = 1 \ldots K
\]

\[
\theta_i^{(t+1)} \sim P(\theta_i | \theta_1^{(t)}, \ldots, \theta_{i-1}^{(t)}, \theta_{i+1}^{(t)}, \ldots, \theta_K^{(t)}, S)
\]

end

- Where \textit{theta} is a vector containing all the parameters and \textit{S} the set of observations.

- The set of \{\theta^{(l)}, \theta^{(l+d)}, \theta^{(l+2d)}, \ldots\} is a sample from the posterior.

- Please see the paper for the exact posterior updates corresponding to the model parameters.
Algorithms

- Algoritms:
  - Centralized: the naive implementation of the MCMC algorithm on a single machine → not scalable
  - MadHab-MapReduce: a Map-Reduce based approximation of the original MCMC
  - MadHab-Spark: the scalable yet efficient implementation

- Each applies the MCMC method, but in different environments under different scalability assumptions.
Map-Reduce-based Implementation:

- Each mapper responsible for performing posterior sampling $P(\theta_i|X_i)$ on an $X_i$ subset of the data.
- Because of the nature of the distributed framework Map-Reduce, these subset samplers run in parallel.
- We apply one reducer which combines the subset-samples applying a Weierstrass transformation (ensemble).

Advantages: simple, fits well to the widely used Map-Reduce framework.

Drawback: the performance-scalability trade-off of the algorithm heavily depends on the number of mappers applied.
MadHab-Spark—The Proposed Algorithm

- **Spark** distributed computation model:
  - Distributed
  - In-memory
  - Highly parallel

- **GraphX library**: A lightweight library on the top of Spark making possible to store and operate on large distributed graphs
MadHab-Spark—The Proposed Algorithm

Spark/GraphX-based Implementation:

- Building an intermediate graph representation of the original model, called *blanket graph*
- Nodes are the Markov blankets of the original Bayesian network
- Directed edges between the blanket nodes iff there is an edge between the corresponding in the Bayesian network of the model
- Individual MCMC update: in-parallel as a graph node operation
Observations regarding the Spark-based implementation:

- The edge definition describes the *computational dependencies*.
- The MCMC update can be run as *node operations* in the blanket graph.
- The algorithm runs the posterior update algorithms on each Markov blanket as a distributed graph node program and spread the updated values along the blanket edges within *Gather-Apply-Scatter (GAS)* cycles iteratively.
- One can apply different *scheduler* for the GAS program resulting different consistency model between the nodes.
Implementing the model as a:
- highly scalable,
- fast (=in-memory because of the MCMC) and efficient,
- general Bayesian modeling framework which
- fits into the current system/data architecture.
Experimental Evaluation

▶ Algorithms:
1. Regularized Generalized Linear Model using Lasso Elastic Net with calibration through isotonic regression;
2. MadHab-MapReduce without calibration;
3. MadHab-Spark without calibration.

▶ Dataset:
- From running campaign with life cycle spanning from 01/19/2015 till 03/31/2015
- Campaign has multiple lines with each line consisting of several ads
- Each line has its specific targeting criteria with allocated budgets
Score Calibration

- As we showed applying some theoretical reasoning the model MadHab does not require score calibration
- However the non-MadHab-based approaches estimates the probabilities far from the true distributions
Thank you for your attention!