Computational Advertising and Recommendation

Ye Chen (yec@microsoft.com)

CS294-1, Behavioral Data Mining, Berkeley

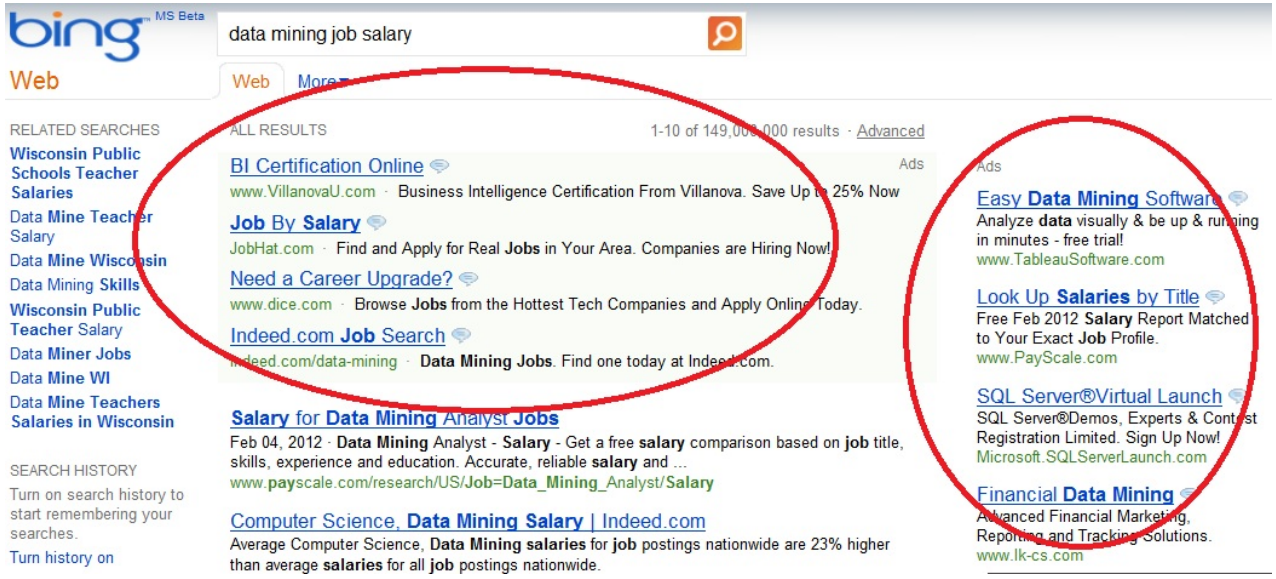
Outline

Computational advertising: Definition

- show the best ads to a user under a context;
- to optimize some utilities of publishers, advertisers, users, and intermediaries;
- an emerging subdiscipline that involves:
  - machine learning: clustering, classification and regression.
  - optimization: linear, integer, convex optimization.
  - information retrieval: query-ad selection, learning-to-rank.
  - economics: game theory, mechanism design, auction theory.
  - large-scale computing: hadoop, distributed computing.
  - recommender systems: content and collaborative filtering.

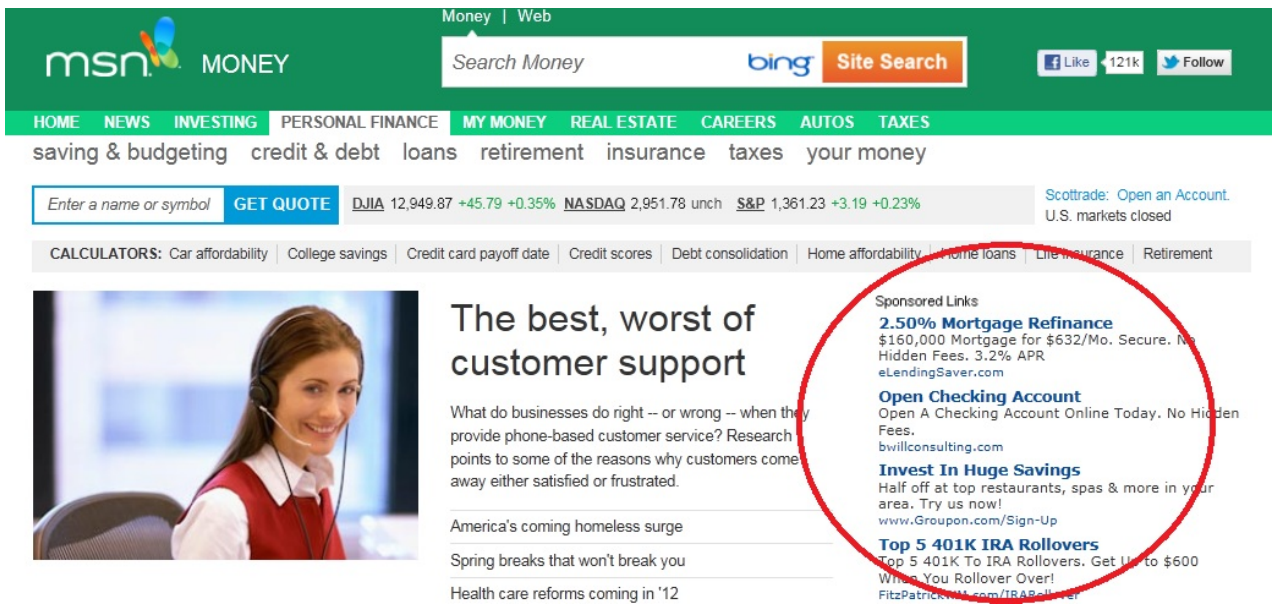
Computational advertising: Sponsored search

- sponsored search;
  - context: a user issues a query.
  - publishers: Google (AdWords), Bing (AdCenter), Yahoo!
  - max: publisher revenue, s.t. advertiser campaign goal, budget, user satisfaction.
  - marketplace: keyword-based GSP with cost-per-click (CPC) pricing.
  - system sketch: query analysis $\rightarrow$ ad selection and relevance $\rightarrow$ click prediction $\rightarrow$ GSP.



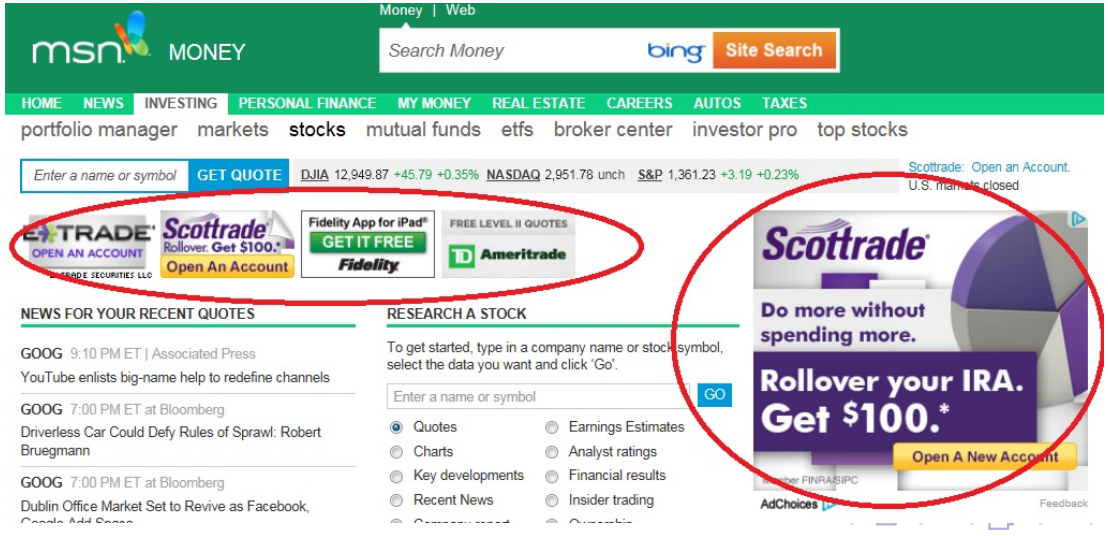
Computational advertising: Contextual ads

- contextual ads: an extension of sponsored search;
  - context: page content and user behavior.
  - publishers: content providers, and major search engines operate the marketplace.
  - system sketch: starts with keyword extraction in absence of user query.



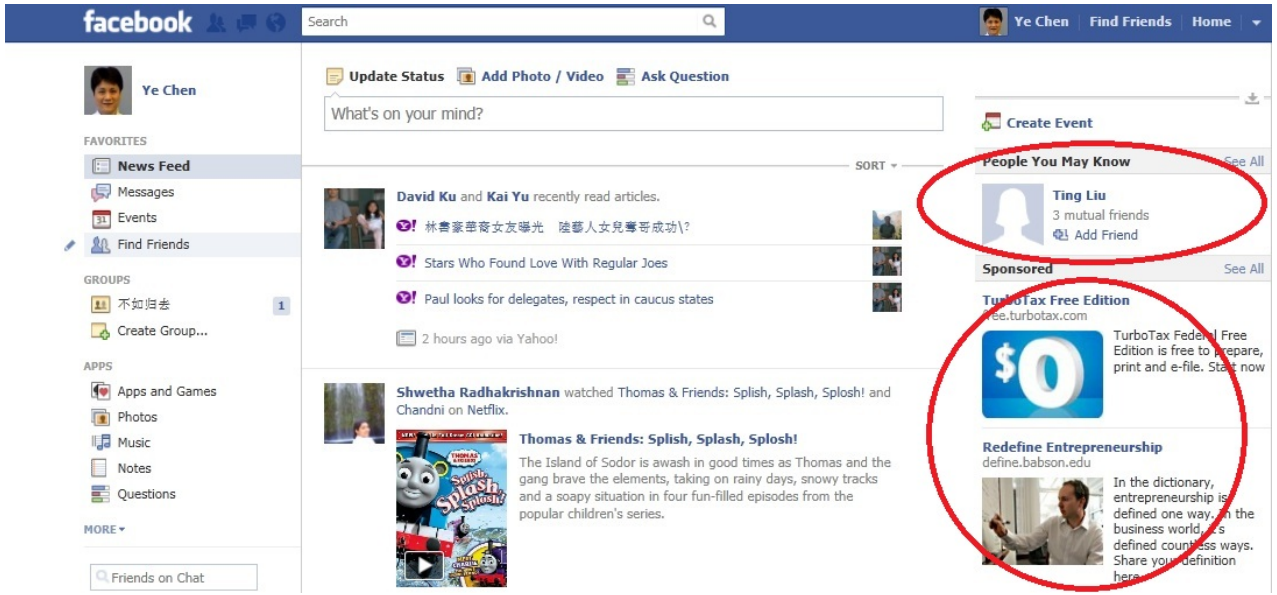
Computational advertising: Display ads

- display ads;
  - context: page, application and user behavior.
  - publishers: content providers in display ad-networks operated by Google (doubleclick), Microsoft (aquantive), and Yahoo! (rightmedia).
  - two types of display ads:
    1. reserved: delivery guaranteed, contracts negotiated upfront, pricing based on CPM (cost-per-(k)impression), e.g., brand ads, direct response.
    2. performance-based: max publisher revenue, s.t. advertiser budget, real-time bidding on exchange, pricing based on CPC/CPA (cost-per-action)/CPM.



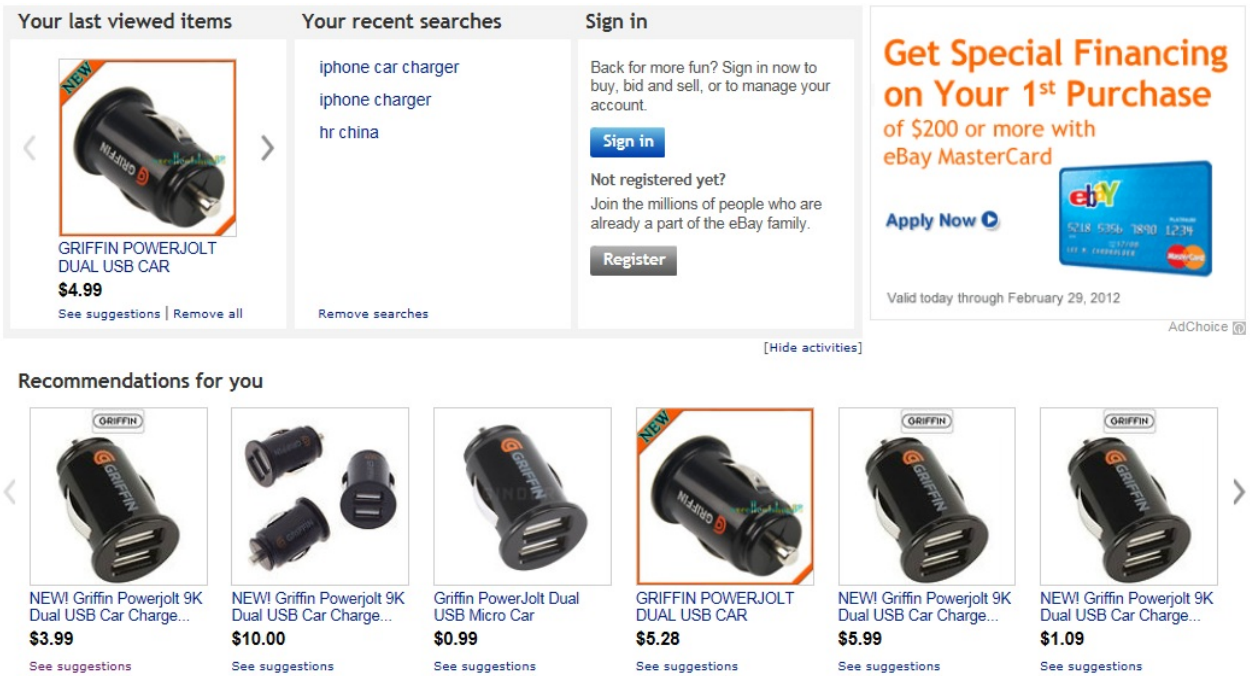
Computational advertising: Emerging formats

- emerging formats;
  - social targeting: Facebook, Twitter, and Linkedin have user profiles and social graphs.
  - mobile ads: ads in apps, real-time location.
  - local ads and deals: Groupon.



Recommendation: An alternate view

- show the best items to a user under a context;
- a classic problem in e-commerce domain, e.g., Amazon, eBay.
- some based on explicit feedbacks, e.g., the popular Netflix problem or rating-based recommender.
- many more rely on implicit feedbacks, e.g., clicks, purchases, dwell-time, social graphs.



A general methodology

- formulate the learning problem;
  - objective function: min quadratic or hinge loss; max precision/recall, AUC, log-likelihood, clicks, or revenue.
  - it is critical yet nontrivial to align with real business goals, e.g., revenue, long-term ROI, user engagement.
  - the gap usually reflects the challenges of non-objectively-measurable, e.g., CTR vs. brand recognition.
- feature representation for users, ads (advertisers), context (publishers).
  - mostly counts or categorical.
  - highly sparse.
  - very rare positive feedbacks.
  - right level of granularity vs. concept drifts.
- solve the optimization problem at large scale (offline) and real-time (online).
- experiments: offline evaluation (due diligence) vs. online A/B testing (true test).

Behavioral targeting: Problem definition

(Chen, Pavlov, and Canny, KDD'09, TKDD'10)

- Behavioral targeting (BT)
  - leverages historical user behavior to select the most relevant ads.
  - y: predicts and maximizes click-through rate (CTR).
  - $\mathbf{x}$: ad clicks and views, page views, search queries and clicks.
- Challenges:
  - large scale, e.g., Y! logged 9TB ad data with 500B entries on Aug'08.
  - sparse, e.g., the CTR of automotive display ads is 0.05%.
  - dynamic, i.e., user behavior changes over time.

Non-negative linear Poisson regression

- Poisson distribution for counts:

$$p(y) = \frac{\lambda^y \exp(-\lambda)}{y!}, \text{where } \lambda = \mathbf{w}^\top \mathbf{x}.$$

- MLE by multiplicative recurrence:

$$w'_j \leftarrow w_j \frac{\sum_i \frac{y_i}{\lambda_i} x_{ij}}{\sum_i x_{ij}}, \text{where } \lambda_i = \mathbf{w}^\top \mathbf{x}_i.$$

- CTR prediction:

$$\widehat{\mathsf{CTR}}_{ik} = \frac{\lambda_{ik}^{\mathsf{click}} + \alpha}{\lambda_{ik}^{\mathsf{view}} + \beta}.$$

- Notation:

y,λ	the observed and expected counts.
$\mathbf{w}$,$\mathbf{x}$	the weight and bag-of-words feature vector.
i, j, k	the indices of user, feature, and ad category.
α, β	the smoothing constants for clicks and views.

Large-scale implementation: Data reduction and information loss

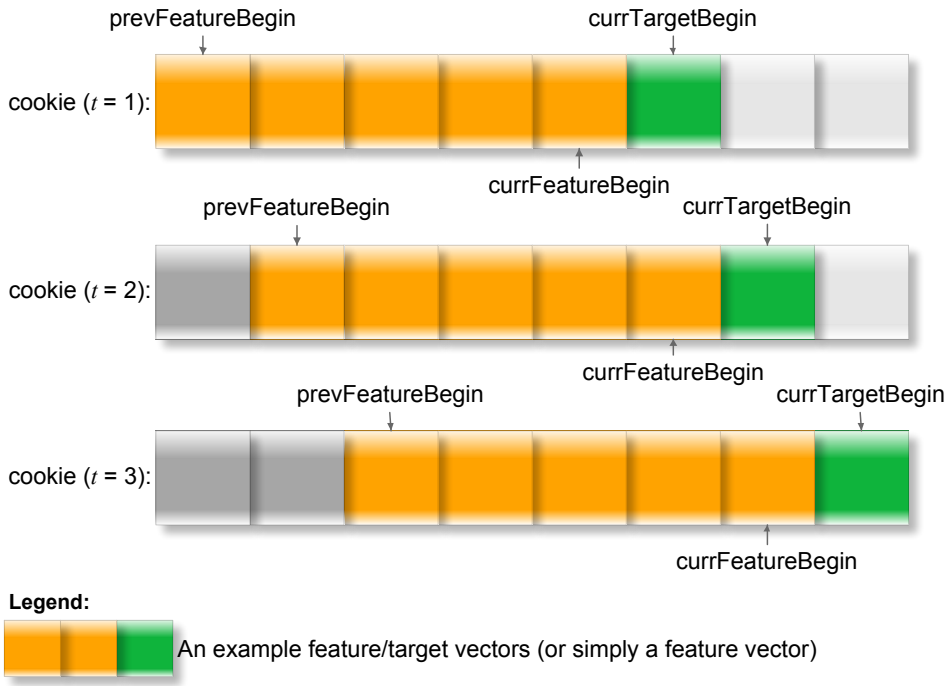
- Many practical learning algorithms are IO-bound and scan-bound.
- For BT, one needs to preprocess 20-30TB raw data feeds of ads and searches.
- Reduce data size at the earliest, by projection, aggregation and merging, e.g., on (cookie, time).
- Data prep should have minimum information loss and redundancy, e.g., time resolution.
- Data prep should be loosely coupled with specific modeling logics for data reusability, e.g., neither decays counts nor categorizes ads.
- After preprocessing, the data size is reduced to 2-3TB.

Large-scale implementation: Feature selection and indexing

- A data-driven approach is to use granular events as features.
- Frequency-based feature selection works almost best in practice for sparse data.
- Frequency is counted in cookie rather than event occurrence (robot filtering).
- Thresholding immediately after summing Mapper, locally and in-memory, thus cut the long tail of the power-law like sparse data.
- Output of feature selection is three dictionaries (ads, pages, queries), which collectively define an indexing of the feature space.

Large-scale implementation: Feature vector generation in $\mathcal{O}(1n)$

- Linear time algorithms are of great interest for large-scale learning.
- The scalar c of a linear complexity $O(cn)$ should be seriously taken into account when n is easily in the order of billion.
- To generate $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ in $O(1n)$ time:



Large-scale implementation: Data-driven weight initialization

- To exploit the sparseness, one shall use some data-driven approaches.
  1. feature-specific normalization (the idea of tf-idf):

$$w_{kj} \leftarrow \frac{\sum_i \frac{y_{ik} x_{ij}}{\sum_{j'} x_{ij'}}}{\sum_i x_{ij}}.$$

  2. target-specific normalization (respect the highly skewed distribution of traffic over categories):

$$w_{kj} \leftarrow \frac{\sum_i (y_{ik} x_{ij}) \sum_i y_{ik}}{\sum_{j'} [\sum_i (y_{ik} x_{ij'}) \sum_i x_{ij'}]}.$$

Large-scale implementation: Parallel multiplicative recurrence

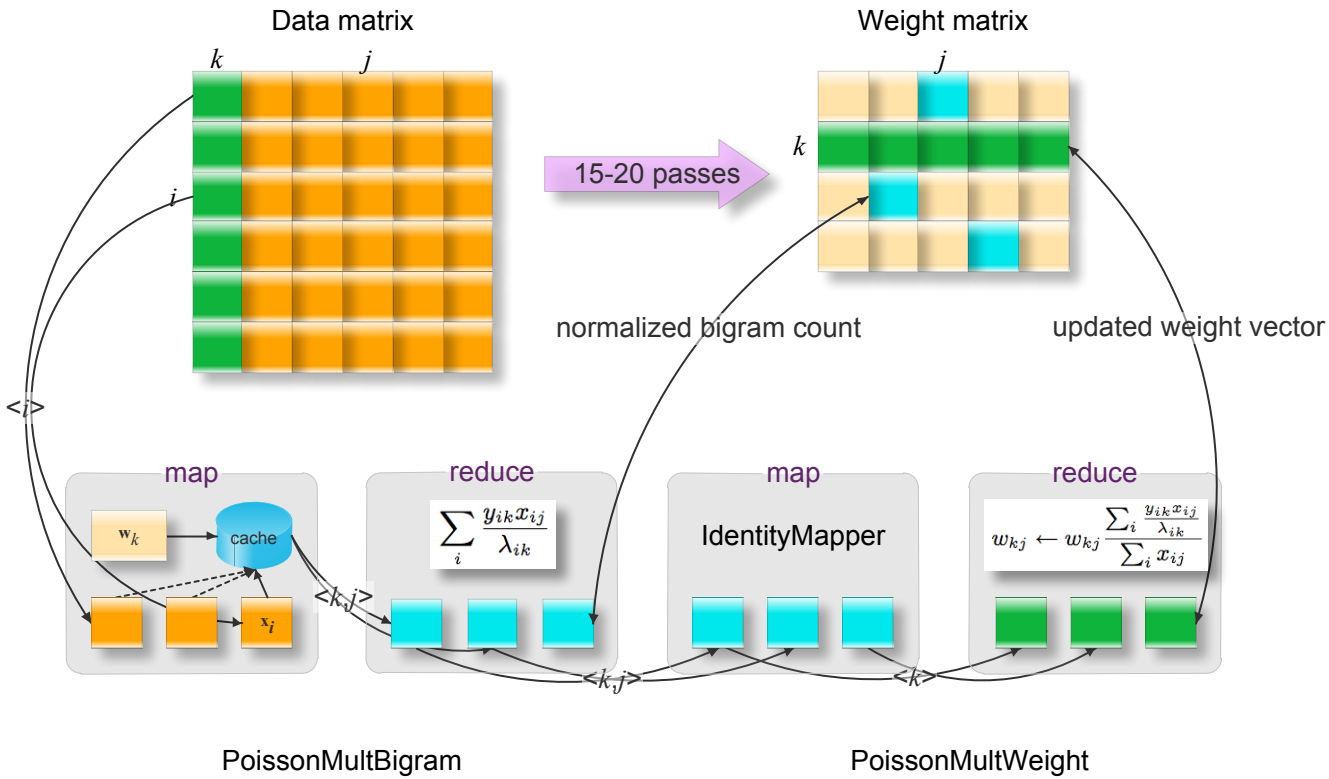
- Given $D = [Y\ X]$, solve $W^* = \text{argmax}_W \log p(Y^\top | W X^\top)$.
- An NMF problem $Y^\top \approx W X^\top$ where the quality of factorization is measured by log likelihood.
- Multiplicative update:

$$w'_j \leftarrow w_j \frac{\sum_i \frac{y_i}{\lambda_i} x_{ij}}{\sum_i x_{ij}}, \text{where } \lambda_i = \mathbf{w}^\top \mathbf{x}_i.$$

- Computational bottleneck: $\sum_i \frac{y_i}{\lambda_i} x_{ij}$.
- Parallel iterative algorithms typically suffer from synchronizing model parameters after each iteration.
- For BT, the final multiplicative update of $\mathbf{w}_k$ has to be carried out in a single node.

Large-scale implementation: "Fine-grained parallelization"

- Scalable data structures: $(\mathbf{x}_i, \mathbf{y}_i)$ sparse vectors, $\mathbf{w}_k$ dense vectors.
- Distribute counting co-occurrences by (k, j) which defines an entry in W.
- In-memory cache input examples (*not weights*), and retrieve relevant weight vectors on demand.



Legend:
1. Variables: x for feature counts, y for target counts, λ for expected target counts, w for model weights;
2. Indices: i for example, j for feature, k for target;
3. <key>: distributing by a single key;

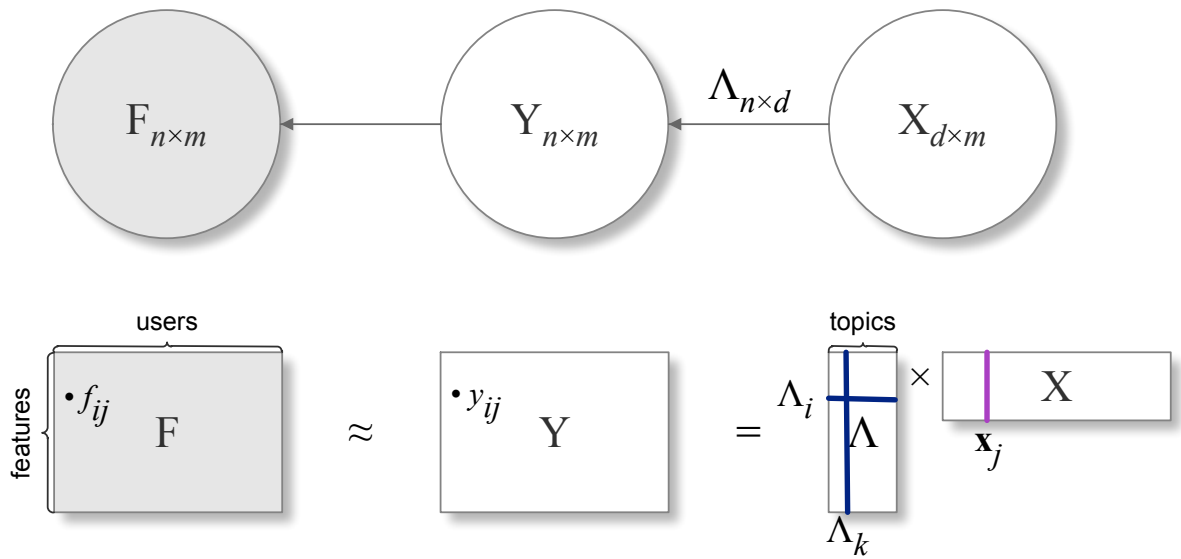
Factor modeling for CTR prediction: Problem definition

(Chen, Kapralov, Pavlov, and Canny, NIPS'09)

- Ad targeting $\text{ad}^* = \text{argmax}_{\text{ad}} f(\text{ad}, \text{user}, \mathbf{x})$
  - To select the ads most relevant to a user.
  - $y = f(\text{ad}, \text{user}, \mathbf{x})$: typically click-through rate (CTR).
  - $\mathbf{x}$: query, page content, user behavior, ad clicks and views.
  - The count data can be formed as a feature-by-user matrix F.
- Sponsored search (SS)
  - To place textual ads alongside algorithmic search results.
  - $y = p(\text{click}|\text{ad}, \text{user}, \text{query})$.
- Behavioral targeting (BT)
  - To select display ads based on historical user behavior.
  - $y = p(\text{click}|\text{ad}, \text{user}, \text{behavior})$.

The GaP factorization

- Notation
  - F is an $n \times m$ matrix of observed counts.
  - Y is an $n \times m$ matrix of expected counts, $F \sim \text{Poisson}(Y)$ element-wise.
  - X is a $d \times m$ matrix where the column $\mathbf{x}_j$ is a low-dimensional representation of user j, i.e., unnormalized $p(k|j)$.
  - Λ is an $n \times d$ matrix where the column Λ_k represents the kth topic as a vector of event probabilities $p(i|k)$, thus $Y = \Lambda X$.
- The graphical model



$$f_{ij} \sim \text{Poisson}(y_{ij}) \leftarrow y_{ij} \sim \text{mixture of Multinomial}(p(i|k)) \leftarrow x_{kj} \sim \text{Gamma}(\alpha_k, \beta_k)$$

The generative model

- To generate an observed event-user count f_{ij}:
  1. Generate $x_{kj} \sim \text{Gamma}(\alpha_k, \beta_k), \forall k$.
  2. Generate y_{ij} occurrences of event i from a mixture of Multinomial $(p(i|k))$ with outcome i, i.e., $y_{ij} = \Lambda_i \mathbf{x}_j$ where Λ_i is the ith row vector of Λ.
  3. Generate $f_{ij} \sim \text{Poisson}(y_{ij})$.
- x_{kj} is given a Gamma as an empirical prior, with pdf

$$p(x) = \frac{x^{\alpha-1} \exp(-x/\beta)}{\beta^{\alpha} \Gamma(\alpha)} \text{ for } x > 0 \text{ and } \alpha, \beta > 0.$$

- Given a latent vector $\mathbf{x}_j$, derive the expected count vector $\mathbf{y}_j$

$$\mathbf{y}_j = \Lambda \mathbf{x}_j.$$

- The observed count f_{ij} follows a Poisson with the mean y_{ij}

$$p(f) = \frac{y^f \exp(-y)}{f!} \text{ for } f \geq 0.$$

Parameter estimation

- The likelihood of a user count vector $\mathbf{f}$

$$p(\mathbf{f}|\Lambda, \mathbf{x}, \alpha, \beta) = \prod_{i=1}^{n} \frac{y_i^{f_i} \exp(-y_i)}{f_i!} \prod_{k=1}^{d} \frac{x_k^{\alpha_k - 1} \exp(-x_k/\beta_k)}{\beta_k^{\alpha_k} \Gamma(\alpha_k)}, \text{where } y_i = \Lambda_i \mathbf{x}.$$

- The log likelihood

$$\ell = \sum_i (f_i \log y_i - y_i - \log f_i!)$$
$$+ \sum_k ((\alpha_k - 1) \log x_k - x_k/\beta_k - \alpha_k \log(\beta_k) - \log \Gamma(\alpha_k)).$$

- Given $F = (\mathbf{f}_1, \ldots, \mathbf{f}_m)$, we wish to find the MLE of the model parameters (Λ, X).

$$\text{E-step: } x'_{kj} \leftarrow x_{kj} \frac{\sum_i (f_{ij} \Lambda_{ik}/y_{ij}) + (\alpha_k - 1)/x_{kj}}{\sum_i \Lambda_{ik} + 1/\beta_k};$$

$$\text{M-step: } \Lambda'_{ik} \leftarrow \Lambda_{ik} \frac{\sum_j (f_{ij} \overline{x}_{kj}/\overline{y}_{ij})}{\sum_j \overline{x}_{kj}}.$$

Rationale for GaP model

- GaP and LDA are very similar, except for one key difference.
  - In LDA, the choice of latent factor is made independently word-by-word.
  - In GaP, several items are chosen from each latent factor, i.e., that topics are locally related.
  - If x_k are independently distributed gamma(α_k, β) respectively, then the vector $(x_1/s, \ldots, x_d/s)$, where $s = \sum_k x_k$, follows a Dirichlet$(\alpha_1, \ldots, \alpha_d)$.
- Another reason for our preference for GaP is its simplicity.
  - LDA requires transcendental functions, e.g., the Ψ function in Eq.(8) in (Blei et al., 2003).
  - GaP requires only basic arithmetic.

Two variants for CTR prediction

- The standard GaP model fits discrete count data.
- We derive two variants for predicting CTR.
  1. To predict clicks and views independently, and then to construct the unbiased estimator of CTR, typically with Laplace smoothing:

$$F \approx Y = \Lambda X;$$

$$\widehat{\mathrm{CTR}}_{\mathrm{ad}(i)j} = \frac{\Lambda_{\mathrm{click}(i)}\mathbf{x}_j + \delta}{\Lambda_{\mathrm{view}(i)}\mathbf{x}_j + \eta}.$$

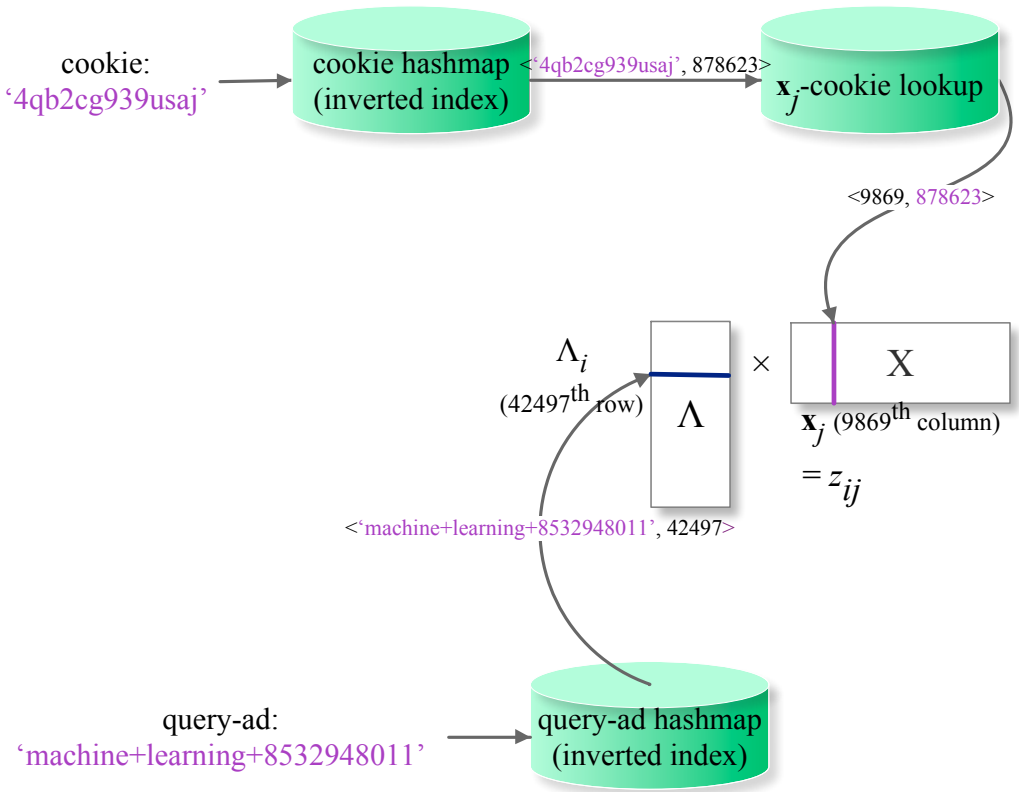
  2. To consider the relative frequency of counts in the GaP factorization. Let F be observed clicks, V be observed views, and Z be expected CTRs:

$$F \approx Y = V.Z = V.(\Lambda X);$$

$$\widehat{\mathrm{CTR}}_{\mathrm{ad}(i)j} = z_{ij} = \Lambda_i \mathbf{x}_j.$$

The GaP deployment for sponsored search

- Offline training. Given the observed F and V obtained from a corpus of historical user data, we derive Λ and X using the CTR-based GaP.
- Offline user profile updating. Given the global Λ and the user-local F and V, we update the user profiles X in a distributed and data-local manner, using E-step recurrence only.
- Online CTR prediction. Given a query issued by user j, the global Λ and the local $\mathbf{x}_j$, the predicted CTRs are obtained by a matrix-vector multiplication $\mathbf{z}_j^{\text{match}} = \Lambda^{\text{match}} \mathbf{x}_j$.



Positional normalization

- The observed CTR represents a conditional $p(\text{click}|\text{position})$, while we wish to learn a CTR normalized by position $p_{\text{rel}}(\text{click}|\text{examine})$.
- We assume an examination model with the Markov process:

$$p(\text{click}|\text{position}) = p_{\text{rel}}(\text{click}|\text{examine})p_{\text{pos}}(\text{examine}|\text{position}).$$

- Apply a GaP factorization with one inner dimension to feature-by-position F and V.
- Simple and empirically motivated.
  - Not dependent on the content of ads higher up, as with the cascade or DBN models.
  - For ads, the probability of clicking any ad link is extremely low.
  - In this case, the DBN positional prior degrades to a negative exponential function.

Large-scale implementation

- Recall the multiplicative recurrence

$$\text{E-step: } x'_{kj} \leftarrow x_{kj} \frac{\sum_i (f_{ij}\Lambda_{ik}/z_{ij}) + (\alpha_k - 1)/x_{kj}}{\sum_i v_{ij}\Lambda_{ik} + 1/\beta_k};$$

$$\text{M-step: } \Lambda'_{ik} \leftarrow \Lambda_{ik} \frac{\sum_j (f_{ij}\overline{x}_{kj}/\overline{z}_{ij})}{\sum_j v_{ij}\overline{x}_{kj}}.$$

- Data locality
  - Updating X in a distributed and data-local manner.
  - Training Λ by alternating 10 successive E-steps with one M-step.
  - For M-step summing over all users ($f_{ij}\overline{x}_{kj}/\overline{z}_{ij}$ and $v_{ij}\overline{x}_{kj}$) incrementally.
- Data sparsity
  - Only computing z when the corresponding f is non-zero (f_{ij}/z_{ij}).
  - Let N_c be the total number of non-zero f's, N_v be the total number of non-zero v's, and r be the number of EM iterations. Typically $N_v \gg N_c \gg m > n \gg d$, the complexity of offline training is $O(N_v dr)$.
- Scalability
  - Assuming $O(N_v dr) \approx 4N_v dr, m = 10M, d = 10, N_v = 100 \times m, r = 15, \ldots, 20$.
  - We have achieved 100 Mflops by a single-machine implementation with sparse matrix arithmetics.
  - Thus it takes 1.6-2.2 hours to train a model.

Remarks on scaling

- We observed that the running time on a 250-node cluster is no less than a single-node yet highly efficient implementation, after accounting for the different factors of users $4\times$ and latent dimension $2\times$, with a similar 100 Mflops.
- To deal with scaling, implementation issues (such as cache efficiency, number of references, data encapsulation) still cause orders-of-magnitude differences in performance and can more than overwhelm the additional nodes.
- The right principle of scaling up should start with a single node and achieve above 100 Mflops with sparse arithmetic operations.

Recommendation at long tail: Problem definition

(Chen and Canny, SIGIR'11)

- Recommendation for online marketplaces (e.g., eBay)
  - Items are ad-hoc listings, without product or catalog taxonomy.
  - Transactional data is very sparse (30-fold sparser than the Netflix data).
  - No user-item ratings, and transactional counts do not follow Gaussian.
- From items to products
  - Most items are unique, no links between user behaviors.
  - Map items to products, a clustering problem.
  - Topic models are not suitable.
    1. An item has 10 terms, little can be learned by projecting to lower-dim
       (e.g., Apple iPhone 4 - 32GB - Black (Unlocked) Smartphone).
    2. Item title terms are highly independent.
    3. The remaining term dependencies are entirely local (e.g., red iPhone?).

Generative clustering

- An item $\mathbf{x}$ is a 3-tuple of vectors: $\mathbf{x} = (\mathbf{b}, \mathbf{c}, \mathbf{g})$.
  1. For binary variables: $b_v \sim \text{Binom}(p_v), \forall v \in \{1, \ldots, V\}$;
  2. For categorical variables: $c_u \sim \text{Mult}(\theta_u), \forall u \in \{1, \ldots, U\}$;
  3. For continuous variables: $g_s \sim \mathcal{N}(\mu_s, \sigma^2), \forall s \in \{1, \ldots, S\}$.
- Given a latent product $\mathbf{z}_k$, the likelihood of an item $\mathbf{x}_i$ is:

$$p(\mathbf{x}_i|\mathbf{z}_k) = \prod_{v:b_{iv}=1} p_{kv} \prod_{v:b_{iv}=0} (1 - p_{kv}) \prod_u \theta_{ku}$$

$$\times \prod_s \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(g_{is} - \mu_{ks})^2}{2\sigma^2}\right).$$

Parameter estimation

- Given a set of items $I = \{\mathbf{x}_i\}_{i=1}^n$, we wish to learn a smaller set of latent products $P = \{\mathbf{z}_k\}_{k=1}^m$.

$$(\mathbf{z}_1^*, \ldots, \mathbf{z}_m^*) = \underset{(\mathbf{z}_1,\ldots,\mathbf{z}_m)}{\operatorname{argmax}} \sum_k \sum_i \gamma_{ik} \ell(\mathbf{x}_i | \mathbf{z}_k),$$

  where γ_{ik} is an indicator variable for item-product membership.

- We thus derive the following EM algorithm:

$$\text{E-step:} \qquad \gamma_{ik} \leftarrow \begin{cases} 1 & k = \underset{k'}{\operatorname{argmax}}\, \ell(\mathbf{x}_i | \mathbf{z}_{k'}), \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

$$\text{M-step:} \qquad \mathbf{z}_k \leftarrow \mathbb{E}\left(\mathbf{x}_i | \gamma_{ik} = 1\right). \tag{2}$$

- Smooth the local parameters $\mathbf{z}_k$ by the background probabilities $\mathbf{q}$:

$$\mathbf{z}_k \leftarrow (1 - \lambda)\mathbf{z}_k + \lambda \mathbf{q}.$$

Efficient inference

- The inferential task given a model trained:

$$k^* = \underset{k}{\operatorname{argmax}}\, \ell(\mathbf{x}'|\mathbf{z}_k).$$

- Let us define $p_a = (1-\lambda)p_{kv} + \lambda q_v$ and $p_b = \lambda q_v$,

$$\ell(\mathbf{x}_i|\mathbf{z}_k) = \sum_{v:p_{kv}>0, b_{iv}=1} \log\left(\frac{p_a(1-p_b)}{(1-p_a)p_b}\right) + \sum_{v:p_{kv}>0} \log\left(\frac{1-p_a}{1-p_b}\right)$$

$$+ \sum_{v} \log(1-p_b) + \sum_{v:b_{iv}=1} \log\left(\frac{p_b}{1-p_b}\right).$$

Naïve Bayes: Product-level recommendation model

- Recommendation can be made by naïve Bayes for ranking:

$$y^* = \operatorname*{argmax}_y p(y|x) \propto \operatorname*{argmax}_y p(x, y).$$

- The product-to-product preference probability:

$$p(y|x) = \frac{\alpha_1 C_{yx}^{\mathsf{bb}} + \alpha_2 C_{yx}^{\mathsf{pp}} + \alpha_3 C_{yx}^{\mathsf{cb}} + \zeta}{C_{yx}^{\mathsf{vb}} + \zeta / p(y)},$$

where the baseline popularity:

$$p(y) = \frac{\beta_1 C_y^{\mathsf{p}} + \beta_2 C_y^{\mathsf{b}} + \beta_3 C_y^{\mathsf{c}}}{\max\left(C_y^{\mathsf{v}}, \varepsilon\right)}.$$

Here C denotes co-occurrences, e.g., C_{yx}^{vb} is the number of users who bid on x and viewed y. We consider four types of co-occurrence patterns: (1) bid-bid (bb), (2) purchase-purchase (pp), (3) click-bid (cb), and (4) view-bid (vb).

Counting co-occurrences

- Implement as matrix multiplication, and exploit sparseness:

$$C^{\mathsf{vb}} = D_{\mathsf{v}} D_{\mathsf{b}}^{\top}, \forall(x, y) \text{ where } C_{yx}^{\mathsf{cb}} > 0.$$

- Use a $t \times w$ sliding window to count co-occurrences.
- Impose a empirical positional prior to normalize (multiply) view count:

$$p(r; \eta, \phi) = \frac{1}{Z_1} r^{-\eta}, r = 1, 2, \ldots, \phi.$$

Here η is a positive real number implying the rate at which a prior decreases as the positional rank r moves down in a search result page, ϕ is the lowest rank the prior covers, and $Z_1 = \sum_{r=1}^{\phi} r^{-\eta}$ is a normalizing constant.

Item-level ranking model

- We wish to have a probabilistic scoring function:

$$j^* = \operatorname*{argmax}_j p(j|i),$$

where i and j are seed and candidate items, respectively.

- The item-to-item recommendation score can be factorized as:

$$p(j|i) = \sum_{x,y} p(j|y)p(y|x)p(x|i)$$

where x and y denote latent products, correspondingly.

- Incorporate auction end time:

$$p(j, \Delta h(j)|i) = p(j|y)p(y|x)p(\Delta h(j)),$$

where $\Delta h(j)$ is the remaining auction time, and $p(\Delta h(j))$ is a smoothed and normalized exponential decay function.

Objective functions

- Ranking recommendations directly in purchase probability shall optimize number of purchases.
- But the probabilistic co-preference score can be extended to maximize other utilities.

$$\mathbb{E}\left(f(u_j)\right) = f(u_j)p(j, \Delta h(j)|i),$$

where u_j is the unit price of the target item j.

- To optimize revenue,

$$f_{\mathsf{rev}}(u) = a_1 \min\left(u, b_1\right) + a_2 \max\left(0, \left(\min\left(u, b_2\right) - b_1\right)\right)$$

$$+a_3 \max\left(0, \left(u - b_2\right)\right).$$

- To optimize user satisfaction,

$$f_{\mathsf{usr}}(u) = 1 + \log\left(\max\left(1, u\right)\right).$$

Summary

- Computational advertising is an emerging scientific subdiscipline that matters substantially to Internet monetization today.
- Recommendation system is a classic problem, yet becomes active as a formulation for ad targeting, and generally matching items to users.
- Industrial problems are rich, yet challenging in data scale, sparsity, real-time response time, and temporal dynamics.
- Often times, the deployment success relies as much on engineering excellence as algorithmic elegance.