Efficient Thompson Sampling for Online Matrix-Factorization Recommendation

추천 시스템 Boot Camp 4기
강석우

Abstract

CF의 중요한 과제인 cold-start 문제는 아직 해결 되지 않았다.

본 논문에서는 덜 추천된 아이템 (혹은 새로운 아이템)에 대한 Explore로 가장 관련성 있는 아이템을 추천하는 새로운 알고리즘인 online MF을 보인다.

Particle Thompson Sampling for MF(PTS)는 톰슨 샘플링에 기반하지만, Rao-Blackwelliezed particle filter에 기반한 online 베이지안 확률 MF 메소드와 결합된 접근 법이다.

PTS가 CF를 적용한 SOTA에 비해 월등한 성능을 보인다는 것을 증명한다.

Introduction

MF는 강력하지만, 새로운 유저/아이템에 대해 cold-start 문제가 있다. 그리고 online 환경에서 유저 피드백을 빠르게 받아들여 추천을 해야하는 다른 문제도 있다.

본 논문에서는 matrix completion과 bandit 알고리즘으로 online low-rank matrix completion 문제를 해결한다.

bandit은 Explore와 Exploit 균형을 맞춤으로 cold start 문제를 자연적으로 해결한다. 그래서 최근 Thompson Sampling이 재조명을 받았다. 하지만 샘플링은 실제 사용하기에는 비용이 부담되고 실제로는 TS를 근사하는 방법으로 사용 하게 된다.

Particle Thompson Sampling for matrix factorization(PTS)은

온라인 베이지안 파라미터 추정에 파티클 필터(시스템에 적절하게 제안된 확률분포로 임의로 생성된 입력을 여럿 가하여보고 그것들을 종합하여 시스템의 정보를 추측하는 것이다.)를 적용한다. 만약 posterior에 베이지안 추정에 알맞지 않은 형태를 가지면 TS 사용하는 계산 효율적인 알고리즘이다.

Introduction

파티클 필터는 weighted 샘플 셋을 사용해 posterior density를 추정하므로 거대한 파라미터 공간을 가지는 문제가 있다.

문제 해결을 위해 Rao-Blackwellization과 디자인한 Monte-Carlo 커널로 데이터가 새롭게 추가될 때, 파티클 셋 업데이트를 최적화(계산 및 통계적으로)한다.

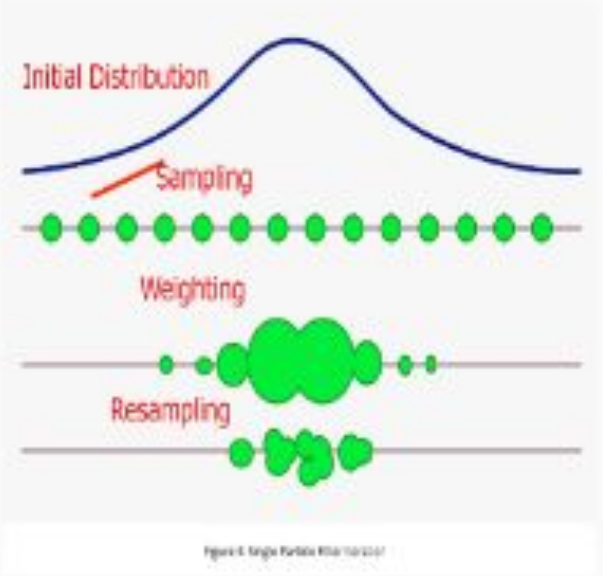
기존에는 single point 추정으로 아이템의 latent feature의 posterior를 추정했지만, PTS는 다양한 파티클 셋을 사용해서 latent feature의 posterior를 추정한다. 다섯가지 실제 추천 데이터셋을 사용해서 누적 regret을 개선함을 보였다.

Particle Filter

Bayesian Filtering은 Markov Chain을 따르는 시스템의 상태를 hidden variable로 가정하고 가정으로 부터의 관측 값을 이용해 시간에 따른 시스템의 상태를 나타내는 확률 밀도 함수를 Recursive로 추정하는 기법이다.

Particle Filter는 베이지안 필터의 근사 방법으로 특정 시간 t에서의 측정 값을 이용해서 가중치를 가지는 파티클 들을 만들고 실제 값의 확률 밀도 함수를 추정한다. 파티클 필터 알고리즘은

1. 초기화
2. 샘플링
   - 이전 시간 t-1의 파티클로 새로운 파티클을 샘플링을 만듬
3. 가중치 업데이트
   - 실제 값과의 비교로 각 파티클의 가중치를 변화 시킴
4. 리샘플링
   - 가중치가 변화함으로 파티클 집합의 분포가 일부 지점으로 수렴하는 것을 방지한다.



2. Probabilistic Matrix Factorization

probabilistic linear model with Gaussian Observation noise.

prior -> U{DxM(유저 수)}, V{DxN(아이템 수)} // conditional distribution -> Rating

Maximize Posterior Distribution with U, V == Minimize Error Function with Regularization

But, Computationarally very expensive

$$p(R|U,V,\alpha) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|U_i^T V_j, \alpha^{-1})\right]^{I_{ij}} \quad (1)$$

$$p(U|\alpha_U) = \prod_{i=1}^{N}\mathcal{N}(U_i|0,\alpha_U^{-1}I) \quad (2)$$

$$p(V|\alpha_V) = \prod_{j=1}^{M}\mathcal{N}(V_j|0,\alpha_V^{-1}I), \quad (3)$$

$$\ln p(U,V|R,\alpha,\alpha_V,\alpha_U) = \ln p(R|U,V,\alpha) + \\ + \ln p(U|\alpha_U) + \ln p(V|\alpha_V) + C,$$

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\left(R_{ij} - U_i^T V_j\right)^2 \\ + \frac{\lambda_U}{2}\sum_{i=1}^{N} \| U_i \|_{\mathrm{Fro}}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \| V_j \|_{\mathrm{Fro}}^2, \quad (4)$$

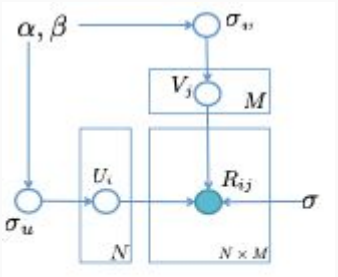


Figure 1: Graphical model of probabilistic matrix factorization model

3. Matrix-Factorization Recommendation Bandit

배포된 추천 시스템은 계속해서 누적되는 기대 보상을 최대화 하는 방향으로 아이템을 추천해야한다. 이에 밴딧 세팅은 시간이 지남에 따라 최고 rating을 가지는 아이템을 유저에게 추천하고 동시에 모든 아이템을 exploration 하므로 다루어 지게 되었다.

Matrix-Factorization Recommendation Bandit

Rating은 (U^*, V^*)의 알려지지 않은 고정된 Latent Features 로 부터 독립 동일 분포 $N(U_i^T V_j, \sigma^2)$로 얻어진다.

시간 t에, environment에서 선택된 유저 i_t에게 시스템(학습된 agent)은 아이템 j_t를 추천해야 한다. 유저는 추천된 아이템에 rating r_{i_t,j_t} $\mathcal{N}(U_{i_t}^{*T} V_{j_t}^*, \sigma^2)$을 하고 시스템은 rating을 보상으로 받는다. 보상을 줄여서 $r_t^o = r_{i_t,j_t}$라 한다.

시스템은 아이템 j_t 추천을 policy를 사용해서 한다. 시간 t 이전에 관측된 rating을 $r_{1:t-1}^o$로 하고 $r_{1:t-1}^o = (i_k, j_k, r_k^o)_{k=1}^t$ 라 한다. 시간 t에 시스템이 얻을 수 있는 최대 기대 보상은 $j^*(i) = max_j U_i^{*T} V_j^*$으로, 최적 아이템 $j^*(i) = argmax_j U_i^{*T} V_j^*$가 추천 되었을 때 받을 수 있다. (U^*, V^*)를 모르기 때문에, 최적 아이템 $j^*(i)$도 사전에 알려지지 않는다. 추천 시스템의 퀄리티는 $expected\ cumulative\ regret$ 으로 측정한다.

$$CR = \mathbb{E}\left[\sum_{t=1}^{n} [r_t^o - r_{i_t,j^*(i_t)}]\right] = \mathbb{E}\left[\sum_{t=1}^{n} [r_t^o - \max_j U_{i_t}^{*\top} V_j^*]\right]$$

expectation은 알고리즘에 의해 시간 t에서 유저 선택과 무작위로 추천될 아이템으로 정해진다.

Particle Thompson Sampling for Matrix Factorization Bandit

CR을 직접적으로 최적화 하기는 어렵기 때문에 톰슨 샘플링을 MF Bandit에 사용한다. 톰슨 샘플링을 MF 밴딧에 사용하는 것의 어려운 점은 보상 구조를 컨트롤하는 latent feature (U, V)를 posterior로 incrementally update 하는 것이다.

Probabilistic Matrix Factorization 모델을 exploit 하기 위해서 설계한 Rao-Blackwellized Particle Filter를 설명하자면

Theta 를 $\theta = (\sigma, \alpha, \beta)$로 컨트롤 파라미터로 한다. 시간 t의 posterior를 $p_t = \Pr(U, V, \sigma_U, \sigma_V, |r^o_{1:t}, \theta)$ 로 한다. 표준 파티클 필터는 모든 파라미터 $(U, V, \sigma_U, \sigma_V)$를 샘플링 한다.

하지만, 바닐라 PF는 σ_U, σ_V를 알고 있다고 가정해도 파티클이 degeneracy(일부 파티클의 가중치가 반복적으로 증가함으로써 파티클 집합 의 분포가 일부 지점으로 수렴하는 퇴보)하는 문제가 있다.

RBPF 알고리즘은 Posterior 분포를 다음과 같이 유지하는데, 각 파티클은 V, σ_U(합쳐질 수 있다.)의 점 밀도를 표현한다.

그래서 $\hat{p}_t = \frac{1}{D} \sum_{d=1}^{D} \delta_{(V^{(d)}, \sigma_U^{(d)})}$ t 시간에서의 posterior 분포를 다음으로 근사한다. (D - 파티클 수)

Particle Thompson Sampling for Matrix Factorization Bandit

결정적으로 파티클 필터는 시간에 변하지 않는 파라미터 들을 추정하기 때문에, 효과적이고 효율적인 MCMC-kernel을 $K_t(V^{'}, \sigma_U^{'}; V, \sigma_U)$로 고정시킨다. p_t에 대해서 필수적이다.

move kernel K_t는 두가지 관측에 기반 해서 설계한다.

첫번째 관측 : U와 σ_V를 보조 변수로 사용해서 효과적으로 $U, \sigma_V | V, \sigma_U \sim p_t(U, \sigma_V | V, \sigma_U)$ 와 $U', \sigma_V^{'} | U, \sigma_V \sim p_t(V', \sigma_U^{'} | U, \sigma_V)$를 샘플링 한다.

이러한 move는 업데이트 마다 샘플링을 하기 때문에 변수가 많으면 엄청 비-효율적 일 수가 있기는 하다.

두번째 관측 : 현재 유저 U_{-i_t}를 제외한 모든 유저에 대한 latent feature 들은 현재 관찰된 rating과 독립적이다. $r_t^o : p_t(U_{-i_t} | V, \sigma_U) = p_{t-1}(U_{-i_t} | V, \sigma_U)$, 그래서 시간 t에서 U_i만 리-샘플 하면 되고 U_{-i_t}는 리-샘플 하지 않아도 된다. (효율적인 구현의 핵심)

그러므로, 현재의 아이템 latent feature V_{jt}를 리-샘플 하기에 충분하다. 이것이 RBPF의 효율적인 구현을 만든다. 각 파티클은 사실 U, V, σ_U, σ_V를 저장하고 (U, σ_V)는 보조 변수, 그리고 커널 이동 K_t에 대해 우리는 $U_{i_t} | V, \sigma_U$ 을 샘플링 하고 다음 $V_{j_t}^{'} | U, \sigma_V$을 샘플링 하고 다음 $\sigma_U^{'} | U, \alpha, \beta$ 을 샘플링을 하면 된다.

Particle Thompson Sampling for Matrix Factorization Bandit

Algorithm 1 Particle Thompson Sampling for Matrix Factorization (PTS)
Global control params: $\sigma, \sigma_U, \sigma_V$; for Bayesian version (PTS-B): σ, α, β

$\hat{p}_0 \leftarrow$ InitializeParticles()
$R^o = \emptyset$
for $t = 1, 2 \ldots$ **do**
  $i \leftarrow$ current user
  Sample $d \sim \hat{p}_{t-1}.w$
  $\tilde{V} \leftarrow \hat{p}_{t-1}.V^{(d)}$
  [If PTS-B] $\tilde{\sigma}_U \leftarrow \hat{p}_{t-1}.\sigma_U^{(d)}$
  Sample $\tilde{U}_i \sim \Pr(U_i | \tilde{V}, \tilde{\sigma}_U, \sigma, r^o_{1:t-1})$ ▷ sample new U_i due to Rao-Blackwellization
  $\hat{j} \leftarrow \arg\max_j \tilde{U}_i^\top \tilde{V}_j$
  Recommend $\hat{j}$ for user i and observe rating r.
  $r_t^o \leftarrow (i, \hat{j}, r)$
  $\hat{p}_t \leftarrow$ UpdatePosterior$(\hat{p}_{t-1}, r^o_{1:t})$
end for
procedure UPDATEPOSTERIOR$(\hat{p}, r^o_{1:t})$
  ▷ $\hat{p}$ has the structure $(w, \text{particles})$ where particles$[d] = (U^{(d)}, V^{(d)}, \sigma_U^{(d)}, \sigma_V^{(d)})$.
  $(i, j, r) \leftarrow r_t^o$
  $\forall d,\ \Lambda_i^{u(d)} \leftarrow \Lambda_i^u(V^{(d)}, r^o_{1:t-1}), \zeta_i^{u(d)} \leftarrow \zeta_i^u(V^{(d)}, r^o_{1:t-1})$ ▷ see Eq. (3)
  $\forall d, w_d \propto \Pr(R_{ij} = r | V^{(d)}, \sigma_U^{(d)}, \sigma, r^o_{1:t-1})$, see Eq.(5), $\sum w_d = 1$ ▷ Reweighting; see Eq.(5)
  $\forall d,\ i \sim \hat{p}.w; \hat{p}'.\text{particles}[d] \leftarrow \hat{p}.\text{particles}[i];\ \forall d, \hat{p}'.w_d \leftarrow \frac{1}{D}$ ▷ Resampling
  for all d **do** ▷ Move
    $\Lambda_i^{u(d)} \leftarrow \Lambda_i^{u(d)} + \frac{1}{\sigma^2} V_j V_j^\top;\ \zeta_i^{u(d)} \leftarrow \zeta_i^{u(d)} + r V_j$
    $\hat{p}'.U_i^{(d)} \sim \Pr(U_i | \hat{p}'.V^{(d)}, \hat{p}'.\sigma_U^{(d)}, \sigma, r^o_{1:t})$ ▷ see Eq. (2)
    [If PTS-B] Update the norm of $\hat{p}'.U^{(d)}$
    $\Lambda_j^{v(d)} \leftarrow \Lambda_j^v(V^{(d)}, r^o_{1:t}), \zeta_j^{v(d)} \leftarrow \zeta_i^u(V^{(d)}, r^o_{1:t})$
    $\hat{p}'.V_j^{(d)} \sim \Pr(V_j | \hat{p}'.U^{(d)}, \hat{p}'.\sigma_V^{(d)}, \sigma, r^o_{1:t})$
    [If PTS-B] $\hat{p}'.\sigma_U^{(d)} \sim \Pr(\sigma_U | \hat{p}'.U^{(d)}, \alpha, \beta)$ ▷ see Eq.(4)
  end for
  return $\hat{p}'$
end procedure

Particle Thompson Sampling for Matrix Factorization Bandit

시간 t 마다의 복잡도는 $\mathcal{O}(((\hat{N}+\hat{M})K^3+K^3)D)$ 이다.

N과 M은 한 아이템에 rating을 매긴 최대 유저 수 그리고 같은 유저에 rating된 최대 아이템 수이다.

세제곱 K에 대한 의존도는 precision matrix를 역행렬로 만드는데 따른 것인데 rank K는 그냥 작다.

알고리즘의 24번 줄 이후는 캐싱을 통한 incremental update 로 수정할 수 있다.

알고리즘의 22번 줄 이후에는 현재 유저 i가 이전에 rating한 모든 아이템 j에 대해 Λ_j^v and ζ_j^v 를 incremental update 할 수 있다. 그러면 복잡도를 $\mathcal{O}((\hat{M}K^2+K^3)D)$, 로 줄일 수 있다. 실제 추천 시스템에서는 각 유저가 소수의 아이템에 대해 rating을 매기기 때문에 잠재적으로는 유의미한 개선이라고 할 수 있다.