Collaborative Metric Learning

홍성훈

Outline

Introduction

Notion of ‘Distance’

- ‘Distance’는 여러 ML algorithm의 핵심 개념이 되어 왔다.

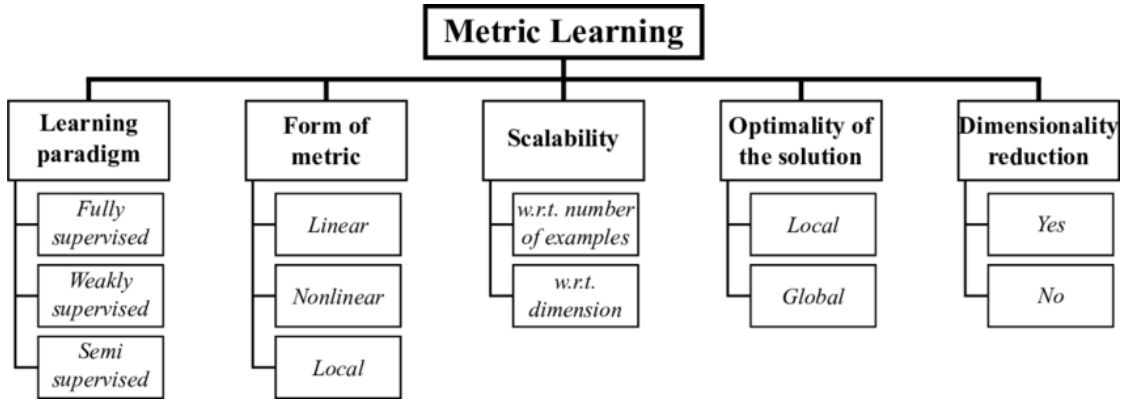
KNN	Input data와의 Distance가 작은 k개의 neighboring data를 기준으로 판단
K-means	각 centroid와 distance가 최소가 되도록 clustering함
SVMs	Hyperplane과의 distance에 따라 support vector가 결정되고, 이에 따라 classify됨

Introduction

Metric Learning Algorithm

- Data 간 중요한 relationship을 함의하는 distance metric을 생성하는 알고리즘

- Image Classification, Document retrieval, Protein function prediction 등 machine learning에 있어 필수적인 technique

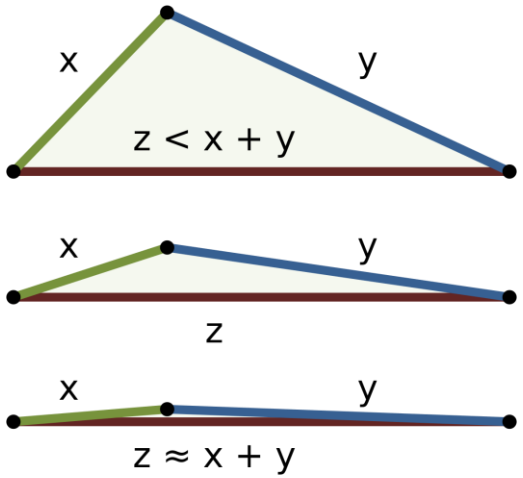


본 논문에서는 collaborative filtering problem에 metric learning을 적용하고, 그 결과를 최근 collaborative filtering approach와 비교

Introduction

Triangle Inequality

- 임의의 세 object에 대해, 두 개의 pairwise distance의 합은 나머지 pairwise distance보다 크거나 같아야 함



- 이는 x가 y와 z 모두와 유사할 때, 학습된 metric은 (x,y), (x,z) 뿐만 아니라 (y,z)도 같이 close하도록 pull되어야 함을 의미

- 이는 알려진 similarity information을 알려지지 않은 pair에 전달하는 similarity propagation이라고 볼 수 있음

Introduction

Triangle Inequality – MF

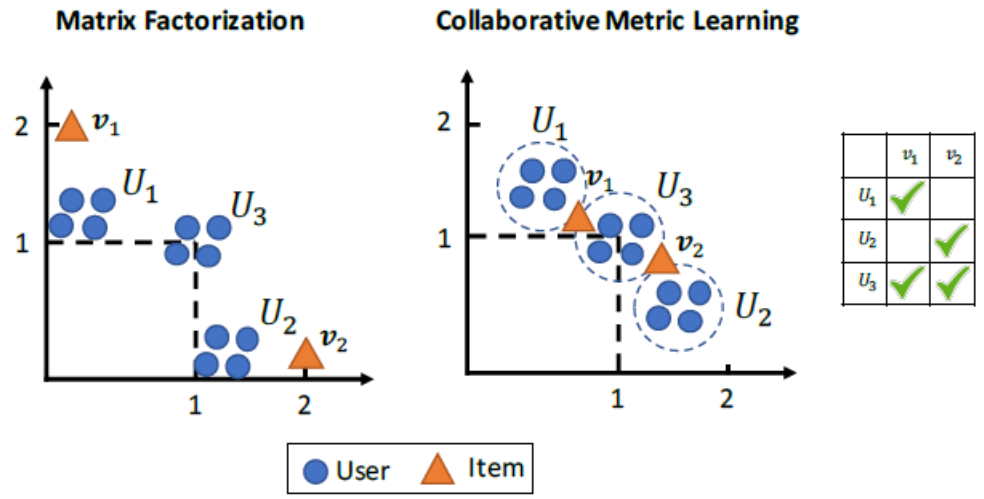
User U1,U2,U3

Item v1, v2

U1 – v1 선호, v2 선호 X

U2 – v1 선호 X, v2 선호

U3 – v1 선호, v2 선호



$$Matrix = \begin{bmatrix} 0 & 2 \\ 2 & 0 \\ 2 & 2 \end{bmatrix} \; LatentUser = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \; LatentItem = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$$

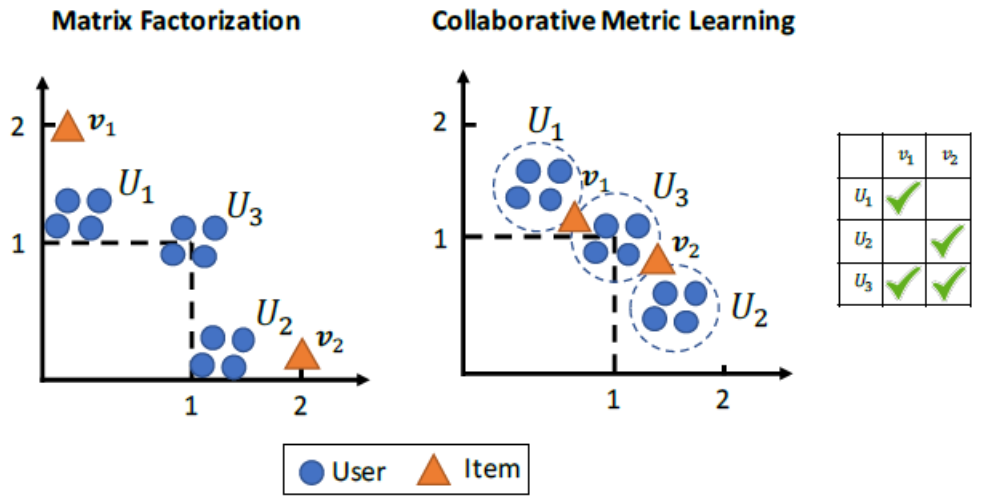
Introduction

Matrix Factorization의 문제점

- Matrix Factorization의 경우, user vector와 item vector의 내적을 통해 알려진 rating을 파악하고, unknown rating을 예측하기 위해 이런 vector의 내적을 사용

- 그러나 dot product는 triangle inequality를 만족하지 못하므로, matrix factorization은 metric learning approach로 사용하기 어려우며, 실제로 suboptimal performance를 보여줌

- 기존의 matrix factorization에만 의존해서 user-user, item-item의 관계를 파악하는 것은 model의 interpretability를 제한



Introduction

Collaborative Metric Learning

- CML은 user의 선호도 간 fine-grained relationship을 얻을 수 있음

- 여러 종류의 item feature에 대해서 모두 적용 가능함

- 또한 Top-K recommendation에서 Locality-Sensitive Hashing을 이용해, 효율성 역시 증가시킬 수 있음

이러한 Matrix Factorization의 단점을 보완하는 CML의 특징을 통해, 본 논문에서는 CML을 구현하고, 다양한 data에 대해 CML의 performance를 확인한다.

Outline

Theoretical Background

Metric Learning

- Metric Learning에서 label information은 similar pair와 dissimilar pair로 설명

$$S = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are considered similar}\}$$

$$D = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are considered dissimilar}\}$$

- Metric Learning approach는 Mahalanobis distance metric을 학습하는 것을 목표로 함

$$d_A(x_i, x_j) = \sqrt{(x_i - x_j)^T A (x_i - x_j)},$$

where $A \in \mathbb{R}^{m \times m}$ is a positive semi-definite matrix

- 이는 각 input x를 유클리드 거리가 constraint를 만족하도록 space $\mathbb{R}^m$에 project 시키는 것과 equivalent함

$$\min_A \sum_{(x_i, x_j) \in S} d_A(x_i - x_j)^2$$

$$s.t. \sum_{(x_i, x_j) \in D} d_A(x_i, x_j)^2 \geq 1 \text{ and } A \succcurlyeq 0$$

Theoretical Background

LMNN

- similar pair을 pull together하고, dissimilar pair을 push apart하는 것을 목표로 함

- x와 가장 가까워야 되는 data point들을 target neighbors, 주위에 침입하는 다르게 label된 input을 imposter라고 부르며, 이러한 imposter의 수를 최소로 하는 것을 optimization의 목표로 함

- 두 개의 loss term을 이용해 위 idea를 formulate한 것이 **LMNN(Largest Margin Nearest Neighbor)**이며, loss term에는 pull loss와 push loss가 존재

$$L_{pull}(d) = \sum_{j \rightsquigarrow i} d(x_i, x_j)^2$$

$$L_{push}(d) = \sum_{i,j \rightsquigarrow i} \sum_k (1 - y_{ik})[1 + d(x_i, x_j)^2 - d(x_i, x_k)^2]_+$$

$j \rightsquigarrow i$: input j가 input i의 target neighbor이다.

Indicator function $y_{ik} = 1$ if input i and k same class, otherwise 0

$[z]_+ = \max(z, 0)$: standard hinge loss

Theoretical Background

Collaborative filtering with implicit feedback

- Original MF model은 user의 explicit feedback에서 user vector와 item vector를 latent vector space로 mapping하여 unknown rating을 예측

$$\min_{u_*, v_*} \sum_{r_{ij} \in K} (r_{ij} - {u_i}^T v_j)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_i\|^2$$

where K is the set of known ratings, λ_u, λ_v are hyperparameters.

- 그러나 이를 똑같이 implicit data에 적용할 경우, unobserved data에 관해 문제가 발생
- Unobserved feedback이 negative feedback인지, 단순히 user가 인지하지 못한 것인지를 단정지을 수 없음

- 이러한 문제를 해결하기 위해 WRMF(Weighted Regularized Matrix Factorization)을 적용

$$\min_{u_*, v_*} \sum_{r_{ij} \in K} c_{ij} (r_{ij} - {u_i}^T v_j)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_i\|^2$$

- c_{ij}는 observed positive feedback에 대해서 더 큰 값을 가지고, unobserved interaction에 대해서는 더 작은 값을 가짐으로서 uncertain sample의 impact를 줄임

Theoretical Background

BPR

- Implicit feedback에 대해서 rating의 개념이 다소 부정확함을 확인할 수 있음

- 따라서 MF model은 다른 item들 간에 상대적인 선호도에 집중하기 시작했으며 BPR(Bayesian Personalized Ranking)이 대표적인 예시

$$\min_{u_*, v_*} \sum_{i \in I} \sum_{(j,k) \in D_i} -log\sigma\left(u_i{}^T v_j - u_i{}^T v_k\right) + \lambda_u \|u_i\|^2 + \lambda_v \|v_i\|^2$$

- D_i 는 (j,k) item pair의 집합으로, j는 user i와 interaction이 있었던 item, k는 user i와 interaction이 없었던 item이며, user i는 item j에 더 흥미가 더 있다고 가정함, σ는 sigmoid function

- 그러나 BPR loss는 lower rank에 있는 item에 대해 충분히 penalize하지 못하며, 이는 Top-K recommendation에서 suboptimal한 결과를 가져옴

Outline

Model Formulation

idea

- 관찰된 implicit feedback을 positive한 relationship을 가진 user-item pair set S로 모델링하고, 이런 관계를 encode하는 user-item joint metric을 학습

- 학습된 metric은 S에 있는 pair을 가깝게 pull하고, 다른 pair들을 상대적으로 더 멀게 push

- 이런 process는 triangle inequality로 인해 같은 item을 선호하는 user와, 같은 user에게 선호되는 item을 clustering하게 됨

→ 알려진 positive relationship을 따르는 metric을 기반으로, 알려지지 않은 user-user pair, item-item pair 역시 관계를 파악할 수 있음

Model Formulation

Loss Function

- User i와 item v의 vector를 그들의 Euclidean distance로 표현할 수 있음

$$d(i,j) = \|u_i - v_j\|$$

- user i가 item j를 선호할 경우 이 distance는 가까워지고, 선호하지 않을 경우 멀어지게 됨.

- 이러한 constraint들을 고려하여 다음과 같은 loss function을 formulate할 수 있음.

$$L_m(d) = \sum_{(i,j)\in S} \sum_{(i,j)\notin S} w_{ij}[m + d(i,j)^2 - d(i,k)^2]_+$$

- Item j는 user i가 선호하는 item이며, item k는 user i가 선호하지 않는 item
- w_{ij}는 ranking loss weight, m>0은 safety margin size를 의미함

- LMNN과 다르게, L_{pull} term이 존재하지 않음
  -> item이 여러 user에게 선호될 수 있으며 모든 user에게 가까워지도록 pull하는 것은 불가능함
  -> L_{push} term이 imposter가 있을 경우 user에게 더 positive한 item을 가깝게 pull하게 됨

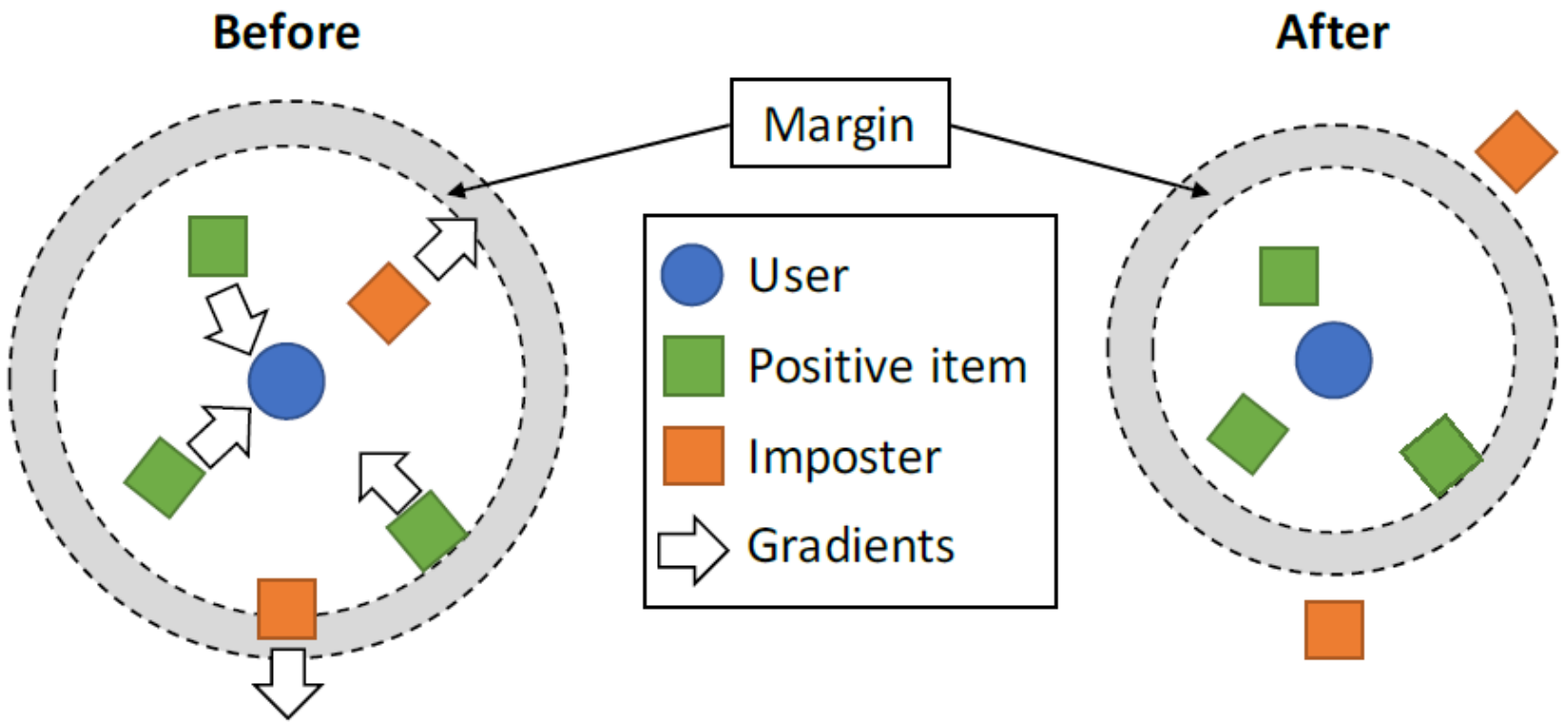
Model Formulation

Loss Fur[illegible]

- User i와 i[illegible]
  로 표현할 [illegible]

- 이때, user
  워지고, 선[illegible]
  constrain
  formulate

$$L_m(d) =$$



ıser i가 선호하

gin size를 의

user에게 가

게 더 positive

Model Formulation

Approximated Ranking Weight

- w_{ij} 를 통해 Lower rank에 있는 item에 대해서 penalize 할 수 있음

- J를 item의 총 개수라 하고, $rank_d(i,j)$를 user i의 recommendation에서 item j의 rank라 할 경우, w_{ij}를 다음과 같이 setting 할 수 있음

$(0 \leq rank_d(i,j) < \mathrm{J})$

$$w_{ij} = \log(rank_d(i,j) + 1)$$

- 그러나 gradient step마다 $rank_d(i,j)$ 를 계산하는 것은 expensive함.
- (user와 positive 각각 한 쌍, 그리고 여러 negative sample을 필요로 함)
- $rank_d(i,j)$에 대해 $\left\lfloor \frac{J}{N} \right\rfloor$로 이를 근사

(N은 하나의 imposter를 찾을 때까지 반복한 횟수)

- 각 item이 뽑힐 확률은 1/J이므로, geometric distribution에 따라 평균에 근사시키면 $\frac{J}{N}$가 됨.
- N의 크기가 더 커질 수 있기 때문에 최종적으로 $\left\lfloor \frac{J \times M}{U} \right\rfloor$로 근사함

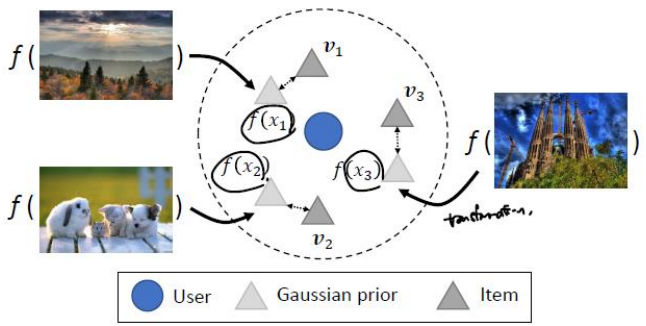
(U는 sample 개수, M은 sample에서의 imposter 개수)

Model Formulation

Integrating Item Features : Feature Loss

- Metric learning에서의 original idea는 raw input을 Euclidean space에 project하는 것이며, 이를 이용하여 item의 tag, image pixel과 같은 item feature을 recommendation system에 integrate할 수 있음

- $x_j \in R^m$을 item j의 raw feature vector라 하면, 이를 joint user-item space에 project하는 transformation function f를 학습



$$L_f(\theta, v_*) = \sum_j \left\| f(x_j, \theta) - v_j \right\|^2$$

- $f(x_j)$를 v_j의 Gaussian prior로 취급하며 v_j와의 deviation을 L2 loss function으로 penalize함. 본 논문에서는 transformation function f를 MLP(Multi-Layer Perception)로 고름.

- 이를 통해 L_m을 통해 학습되는 item vector v_j 와 MLP의 output 간의 distance를 minimize하는 것을 목표로 함

- 이 결과로 item feature가 유사한 item끼리 clustering되는 것을 확인할 수 있음

Model Formulation

Regularization

- KNN을 기반으로 한 모델은 data point가 너무 넓게 퍼져 있을 경우 high-dimensional model에 대해 매우 비효율적

- 그러므로 user, item vector를 unit sphere에 bound해야 됨

$$\|u_*\|^2 \leq 1, \|v_*\|^2 \leq 1$$

Model Formulation

Regularization- Matrix Factorization과 비교

- MF의 경우 정보가 알려진 user와 item에 대해 overfitting을 피하기 위해 L2 norm을 적용

$$\min_{u_*,v_*} \sum_{r_{ij}\epsilon K}(r_{ij} - u_i^T v_j)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_j\|^2$$

- L2 norm은 user vector와 item vector를 origin으로 pull 하게 되는데 CML의 경우 origin이 specific한 의미를 가지고 있지 않음

  -> covariance regularization을 사용

- 학습된 metric에서의 상관관계를 줄이기 위해 covariance regularization을 사용
- Covariance matrix C는 다음과 같이 정의

$$C_{ij} = \frac{1}{N}\sum_n (y_i^n - \mu_i)(y_j^n - \mu_j),$$

y_i^n: i번째 row, n번째 feature의 item 또는 user vector

이때 N은 batch size, n은 batch index를 의미

- Loss function L_C는 다음과 같이 정의할 수 있음

$$L_C = \frac{1}{N}\left(\|C\|_f - \|diag(C)\|_2^2\right)$$

이때, $\|.\|_f$는 Frobenius norm

Model Formulation

Training Procedure

- 위 과정을 모두 integrate하여 만든 loss term은 다음과 같다

$$\min_{\theta,u_*,v_*} L_m + \lambda_f L_f + \lambda_c L_c$$
$$s.t. \quad \|u_*\|^2 \leq 1 \text{ and } \|v_*\|^2 \leq 1$$

- 이때, λ_f와 λ_c는 loss term에서 weight를 조절하며, 이러한 objective function을 mini-Batch SGD로 학습시킴

- Learning rate의 경우 AdaGrad를 사용하여 조절

- Set S로부터 positive sample N 뽑음
- 위에서 뽑은 각 sample인 (user, item) pair에 대해 Negative Sample U를 뽑음
- 각 pair에 대해 hinge loss를 극대화할 수 있는 negative item K개를 유지하고 mini-batch size N을 만듬
- AdaGrad를 사용하여 gradient를 계산하고, parameter를 업데이트

- $y' = \frac{y}{\max(\|y\|,1)}$로 u_*, v_* 값을 조정
- 수렴할 때까지 위 procedure를 반복

Model Formulation

AdaGrad

- Adagrad는 learning rate가 일정하는 데서 오는 문제를 학습 과정에서 이를 감소시킴으로서 해결한 알고리즘

- h에 이전 기울기의 제곱을 누적해서 더하며, 이의 제곱근에 반비례하여 learning rate가 감소하게 됨

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

Outline

Experiment

Experiment 목표

1. CML의 recommendation domain에서의 superior accuracy를 보이고자 함

2. CML이 Top-K recommendation의 속도를 향상시킴을 보이고자 함

3. 학습된 metric이 user의 fine-grained preference와 underlying preference spectrum을 찾을 수 있음을 보이고자 함

Experiment

Datasets

- Evaluation의 난이도와 size가 모두 다른 6개의 different domain에서의 dataset을 사용

Table 1: Dataset Statistics.

	CiteULike	BookCX	Flickr	Medium	MovieLens20M	EchoNest
Domain	Paper	Book	Photography	News	Movie	Song
# Users	7,947	22,816	43,758	61,909	129,797	766,882
# Items	25,975	43,765	100,000	80,234	20,709	260,417
# Ratings	142,794	623,405	1,372,621	2,047,908	9,939,873	7,261,443
Concentration[a]	33.47%	33.10%	13.48%	55.38%	72.52%	65.88%
Features Type	Tags	Subjects	Image Features	Tags	Genres, Keywords	NA
# Feature Dim.	10,399	7,923	2,048	2,313	10,399	NA

[a]Concentration is defined as the percentage of the ratings that concentrate on the Top 5% of the items.

Experiment

Evaluation Methodology

- User의 rating을 60%/20%/20%의 비율로 training, validation, test set으로 분할
- 5개보다 작은 rating을 가진 user는 training set에만 포함
- 여러 model의 ranking은 Top-K recommendation의 recall rate로 평가

CML을 3개의 CF model과 비교

- WRMF(Weighted Regularized Matrix Factorization)
- BPR(Bayesian Personalized Ranking)
- WARP(Weighted Approximate-Rank Pairwise)

또한 item feature을 integrate시킨 CML+F을 3개의 hybrid CF와 비교

- FM(Factorization Machine)
- VBPR(Visual BPR)
- CDL(Collaborative Deep Learning)

Experiment

Hyperparameter setting

- 모든 model의 loss는 WARP loss를 사용했으며, CDL, VBPR, CML에서는 똑같은 MLP model을 사용

- Hyperparameter는 각 model에 대해 validation set에서 best performance를 보여주도록 setting 되었으며 CML은 $m = 0.5, \lambda_f = 1, \lambda_c = 10$로 setting 됨(MovieLens data에 대해서만 $\lambda_f = 0.5$)

- MLP의 경우 256-dimensional hidden layer, 50% dropout과 ReLu를 activation function으로 사용

- MF baseline들에 대해서는 latent vector size에 대해 10~100의 범위에서 evaluate 했으며, 비슷하게 결과를 확인

- Space의 이점이 있으므로 r=100의 vector size를 사용

Experiment

Result – Recommendation Accuracy

Table 2: Recall@50 and Recall@100 on the test set. (# dimensions $r = 100$) The best performing method is boldfaced. $*$, $**$, $***$ indicate $p \leq 0.05$, $p \leq 0.01$, and $p \leq 0.001$ based on the Wilcoxon signed rank test suggested in [41].

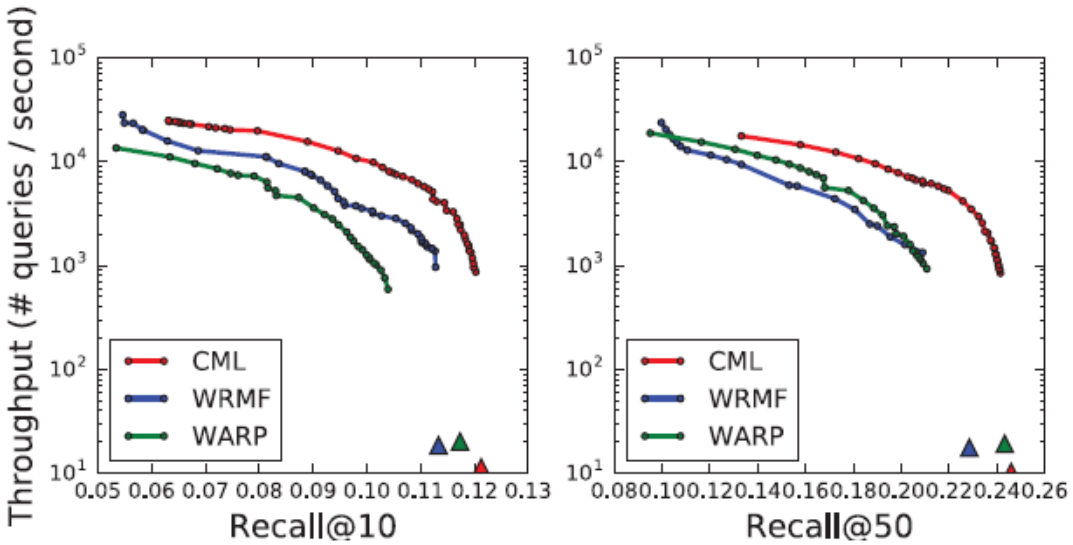
	WRMF	BPR	WARP	CML	*ours vs. best*	FM	VBPR	CDL	CML+F	*ours vs. best*
					Recall@50					
CiteULike	0.2437	0.2489	0.1916	**0.2714*****	9.03%	0.1668	0.2807	**0.3375****	0.3312	-1.86%
BookCX	0.0910	0.0812	0.0801	**0.1037*****	13.95%	0.1016	0.1004	0.0984	**0.1147*****	12.89%
Flickr	0.0667	0.0496	0.0576	**0.0711*****	6.59%	NA	0.0612	0.0679	**0.0753*****	10.89%
Medium	0.1457	0.1407	0.1619	**0.1730*****	6.41%	0.1298	0.1656	0.1682	**0.1780*****	5.82%
MovieLens	0.4317	0.3236	0.4649	**0.4665**	0.34%	0.4384	0.4521	0.4573	**0.4617***	0.96%
EchoNest	0.2285	0.1246	0.2433	**0.2460**	1.10%	NA	NA	NA	NA	NA
					Recall@100					
CiteULike	0.3112	0.3296	0.2526	**0.3411*****	3.37%	0.2166	0.3437	0.4173	**0.4255****	1.96%
BookCX	0.1286	0.1230	0.1227	**0.1436*****	11.66%	0.1440	0.1455	0.1428	**0.1712*****	17.66%
Flickr	0.0821	0.0790	0.0797	**0.0922*****	12.30%	NA	0.0880	0.0909	**0.1048*****	15.29%
Medium	0.2112	0.2078	0.2336	**0.2480*****	6.16%	0.1900	0.2349	0.2408	**0.2531*****	5.10%
MovieLens	0.5649	0.4455	0.5989	**0.6022**	0.55%	0.5561	0.5712	0.5943	**0.5976**	0.55%
EchoNest	0.2891	0.1655	0.3021	**0.3022**	0.00%	NA	NA	NA	NA	NA

Experiment

Top-K recommendation with LSH

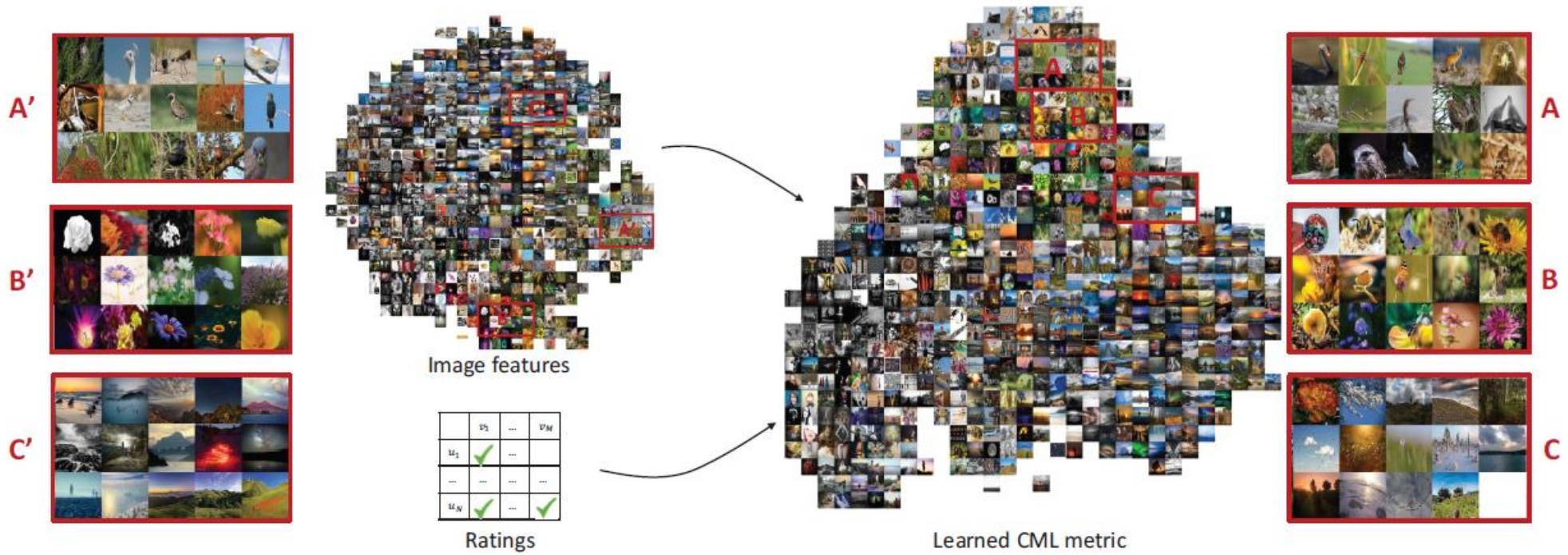
- MF에서 Top-K recommendation search problem은 maximum-inner-product search problem과 같음

- Asymmetric LSH for sublinear time MIPS 논문에서 사용한 LSH를 CML에 적용시켰을 때, brute-force search에서 제일 느렸던 CML이 LSH를 적용시키면 속도가 제일 빨라짐을 알 수 있음



Experiment

Metric Visualization using t-SNE



Outline

Implementation

Data

- 본 논문에서는 CML을 다양한 data에 적용하여 performance를 확인

- 그 중에 가장 data 정리가 잘 되어 있는 citeulike data를 통해 performance를 확인

```
2 15135 2058
1 9271
2 14664 1888
1 11583
1 1357
1 19774
1 24611
1 2283
13 1445 14247 15
1 10192
1 7078
1 18998
1 24655
3 17661 25304 11
1 18036
2 14059 20147
2 12468 22186
1 5667
1 25243
1 17177
2 20612 22651
1 543
```

```
4 193 11908 12727 14760
24 1506 2248 2307 2571 4196 5079 6425 8257 12550 13257 1571
23 1736 1870 2192 2847 3288 4333 4349 4539 4878 4969 5740 8
3 6193 14123 21147
29 480 3261 4247 6603 7517 9563 9934 9969 10541 11860 12053
4 6071 12723 13776 14631
17 1209 2533 3158 3303 4970 6000 7652 8727 9066 13881 14687
4 7621 8063 12244 22943
7 424 479 1751 6051 9850 13560 19697
4 5697 15701 16519 19854
12 3107 3834 5788 8803 9400 9810 9864 9865 12541 19633 2288
7 2483 13578 17111 17898 20318 22111 23155
3 2250 2648 19865
4 2510 4792 16512 25601
73 247 841 1336 1383 1388 1894 1929 2050 2696 2751 2770 326
54 54 319 329 899 1094 1942 2336 2560 2943 3624 5038 5749 6(
4 11448 12099 25446 25558
7 903 5731 8398 14693 16847 17826 23591
15 145 2738 7017 8576 8682 9303 9601 12543 13506 16390 1809
6 6205 6786 12020 13036 13992 16826
7 3909 7824 8428 9977 17247 21792 25638
7 1556 3563 4526 7779 8249 15892 25300
```

Implementation

Code Implementation

```
class CML(object):
    def __init__(self,
                 n_users,
                 n_items,
                 embed_dim=20,
                 features=None,
                 margin=1.5,
                 master_learning_rate=0.1,
                 clip_norm=1.0,
                 hidden_layer_dim=128,
                 dropout_rate=0.2,
                 feature_l2_reg=0.1,
                 feature_projection_scaling_factor=0.5
                 use_rank_weight=True,
                 use_cov_loss=True,
                 cov_loss_weight=0.1
                 ):
```

```
    def loss(self):
        loss = self.embedding_loss + self.feature_loss
        if self.use_cov_loss:
            loss += self.covariance_loss
        return loss
```

```
def optimize(self):
    gds = []
    gds.append(tf.train
               .AdagradOptimizer(self.master_learning_rate)
               .minimize(self.loss, var_list=[self.user_embeddings, self.item_embeddings]))
    if self.feature_projection is not None:
        gds.append(tf.train
                   .AdagradOptimizer(self.master_learning_rate)
                   .minimize(self.feature_loss / self.n_items))

    with tf.control_dependencies(gds):
        return gds + [self.clip_by_norm_op]
```

Implementation

Implementation Result

```
Recall on (sampled) validation set: 0.0006967735627777572
Optimizing...: 100%|██████████| 30/30 [02:09<00:00,  4.30s/it]

Training loss 617801.0

Recall on (sampled) validation set: 0.19447021694140124
Optimizing...: 100%|██████████| 30/30 [01:51<00:00,  3.73s/it]

Training loss 504339.1875

Recall on (sampled) validation set: 0.2142913240461077
Optimizing...: 100%|██████████| 30/30 [01:51<00:00,  3.70s/it]

Training loss 492096.53125

Recall on (sampled) validation set: 0.2166208372831288
Optimizing...: 100%|██████████| 30/30 [01:49<00:00,  3.65s/it]

Training loss 485784.1875

Recall on (sampled) validation set: 0.2214539806925115
Optimizing...: 100%|██████████| 30/30 [01:51<00:00,  3.70s/it]

Training loss 482359.5625

Recall on (sampled) validation set: 0.229101887057704
Optimizing...: 100%|██████████| 30/30 [01:50<00:00,  3.69s/it]

Training loss 480076.78125

Recall on (sampled) validation set: 0.22428700723882117
Optimizing...: 100%|██████████| 30/30 [02:00<00:00,  4.02s/it]

Training loss 479004.21875

Recall on (sampled) validation set: 0.2197872543194661
Optimizing...: 100%|██████████| 30/30 [01:51<00:00,  3.72s/it]
```

recall@50(CML)

```
Recall on (sampled) validation set: 0.000800319622012229
Optimizing...: 100%|██████████| 30/30 [06:53<00:00, 13.78s/

Training loss 576240.6875

Recall on (sampled) validation set: 0.21203636862261985
Optimizing...: 100%|██████████| 30/30 [06:37<00:00, 13.25s/

Training loss 449606.0625

Recall on (sampled) validation set: 0.22160340735433712
Optimizing...: 100%|██████████| 30/30 [06:32<00:00, 13.10s/

Training loss 436600.84375

Recall on (sampled) validation set: 0.22783224550689038
Optimizing...: 100%|██████████| 30/30 [06:38<00:00, 13.28s/

Training loss 429758.5625

Recall on (sampled) validation set: 0.228989125596117
Optimizing...: 100%|██████████| 30/30 [06:56<00:00, 13.87s/

Training loss 426032.78125

Recall on (sampled) validation set: 0.23221923099651517
Optimizing...: 100%|██████████| 30/30 [06:40<00:00, 13.36s/

Training loss 423569.125

Recall on (sampled) validation set: 0.2283501823766861
Optimizing...: 100%|██████████| 30/30 [06:38<00:00, 13.27s/
```

recall@50(CML+)

```
Recall on (sampled) validation set: 0.2956054183983315
Optimizing...: 100%|██████████| 30/30 [01:48<00:00,  3.63s/it]

Training loss 480790.6875

Recall on (sampled) validation set: 0.2962173714210118
Optimizing...: 100%|██████████| 30/30 [01:50<00:00,  3.69s/it]

Training loss 478624.09375

Recall on (sampled) validation set: 0.29323261748839663
Optimizing...: 100%|██████████| 30/30 [01:49<00:00,  3.66s/it]

Training loss 477666.4375

Recall on (sampled) validation set: 0.29434443819589734
Optimizing...: 100%|██████████| 30/30 [01:49<00:00,  3.65s/it]

Training loss 475513.5625

Recall on (sampled) validation set: 0.3036378341034471
Optimizing...: 100%|██████████| 30/30 [01:48<00:00,  3.63s/it]

Training loss 475107.9375

Recall on (sampled) validation set: 0.29700542648044836
Optimizing...: 100%|██████████| 30/30 [01:49<00:00,  3.65s/it]

Training loss 473969.46875

Recall on (sampled) validation set: 0.2895455312049477
Optimizing...: 100%|██████████| 30/30 [01:48<00:00,  3.63s/it]

Training loss 473946.28125
```

recall@100(CML)

```
Recall on (sampled) validation set: 0.008491427203065132
Optimizing...: 100%|██████████| 30/30 [02:05<00:00,  4.19s/it]

Training loss 601795.3125

Recall on (sampled) validation set: 0.2392285970070502
Optimizing...: 100%|██████████| 30/30 [01:58<00:00,  3.96s/it]

Training loss 486102.59375

Recall on (sampled) validation set: 0.2709344520659167
Optimizing...: 100%|██████████| 30/30 [01:50<00:00,  3.70s/it]

Training loss 476021.4375

Recall on (sampled) validation set: 0.27479190174487816
Optimizing...: 100%|██████████| 30/30 [01:48<00:00,  3.60s/it]

Training loss 471038.15625

Recall on (sampled) validation set: 0.2905078079703886
Optimizing...: 100%|██████████| 30/30 [01:47<00:00,  3.60s/it]

Training loss 468601.84375

Recall on (sampled) validation set: 0.28995030099444946
Optimizing...: 100%|██████████| 30/30 [01:47<00:00,  3.59s/it]
```

recall@100(CML+)

Outline

Conclusion

Conclusion

- Experiment에 나온대로, recommendation의 performance, efficiency를 향상시키고, user의 fine-grained preference와 underlying preference spectrum을 더 잘 보여줄 수 있는 metric learning method

- MF에 비해, user-item pair의 관계를 더 직관적으로 파악하며, 이 정보를 보다 효과적으로 알려지지 않은 pair에 propagate함

- 본 논문에서는 item의 feature만 고려했지만, user의 feature 역시 individual usage trace를 통해 고려할 수 있어야 한다고 제시