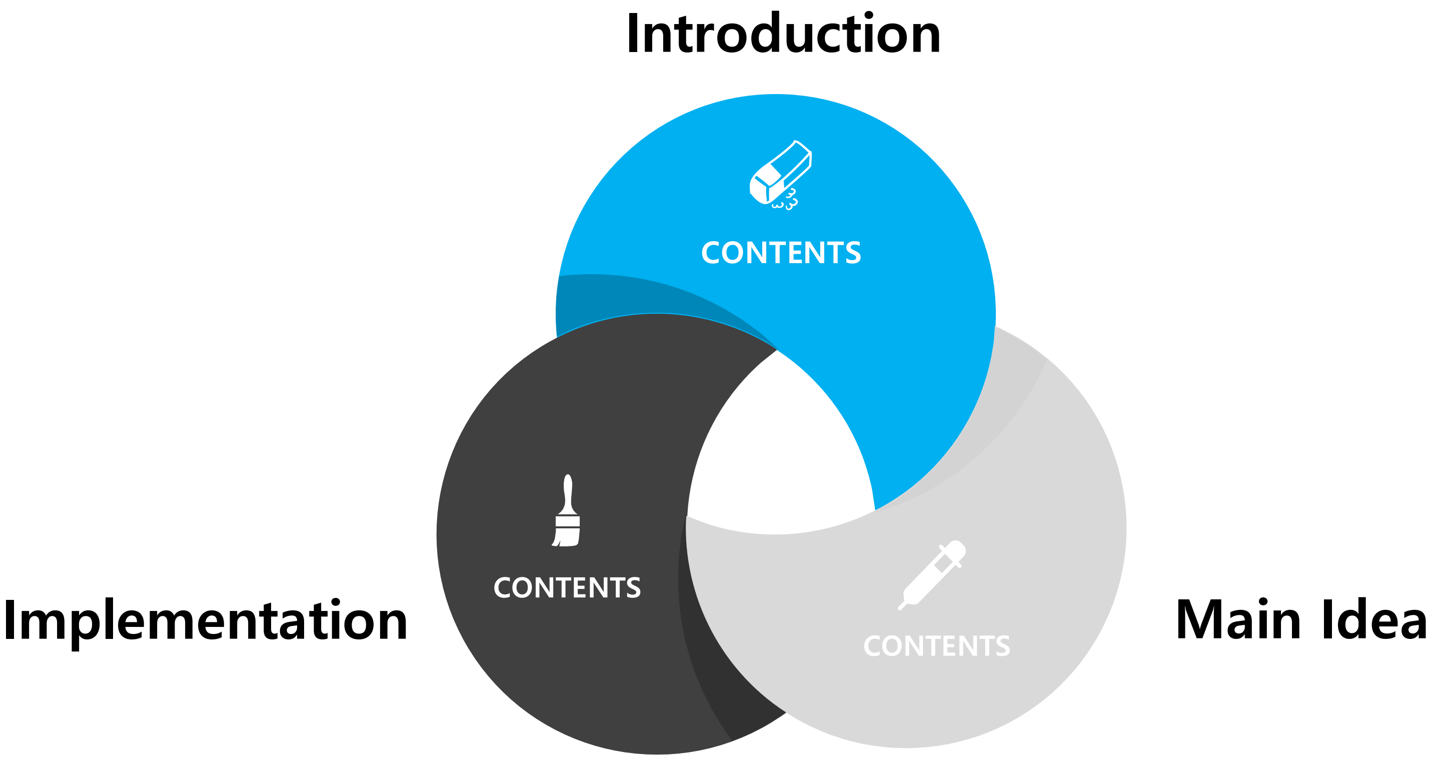
BPR: Bayesian Personalized Ranking from Implicit Feedback

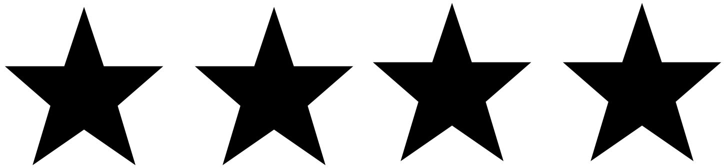
Sangmin Park

Contents


Introduction
CONTENTS
Implementation
CONTENTS
CONTENTS
Main Idea

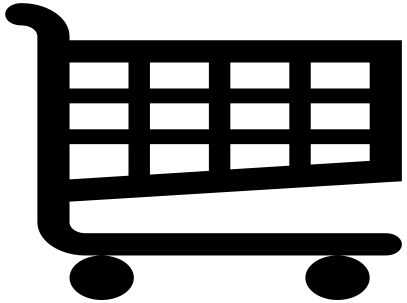
Introduction : Implicit feedback

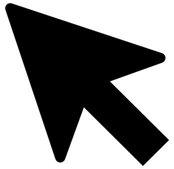
Explicit Feedback



더 가시적이다.

Implicit Feedback





더 얻기 쉽다.

Introduction : purpose

MF(Matrix Factorization)

User x Item Matrix에 latent features를 고려함으로써 User matrix, Item matrix로 분해하여 예측한다.

KNN(k-nearest-neighbor)

Similarity matrix를 기준으로 k개의 인접한 데이터를 기반하여 예측한다.

이전의 연구는 **특정 item에 대한 구매 여부**에 관심이 있었다면,

위 연구는 items 사이의 선호도에 기반한 **Personalized Ranking**에 더욱 중점을 둔다.

Introduction : map

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

$$= \frac{p(y|\theta)p(\theta)}{\int_\theta p(y|\theta)p(\theta)d\theta} \qquad \left(\because p(y) = \int_\theta p(y|\theta)p(\theta)d\theta\right)$$

$$\propto p(y|\theta)p(\theta)$$

Posterior ∝ Likelihood X Prior

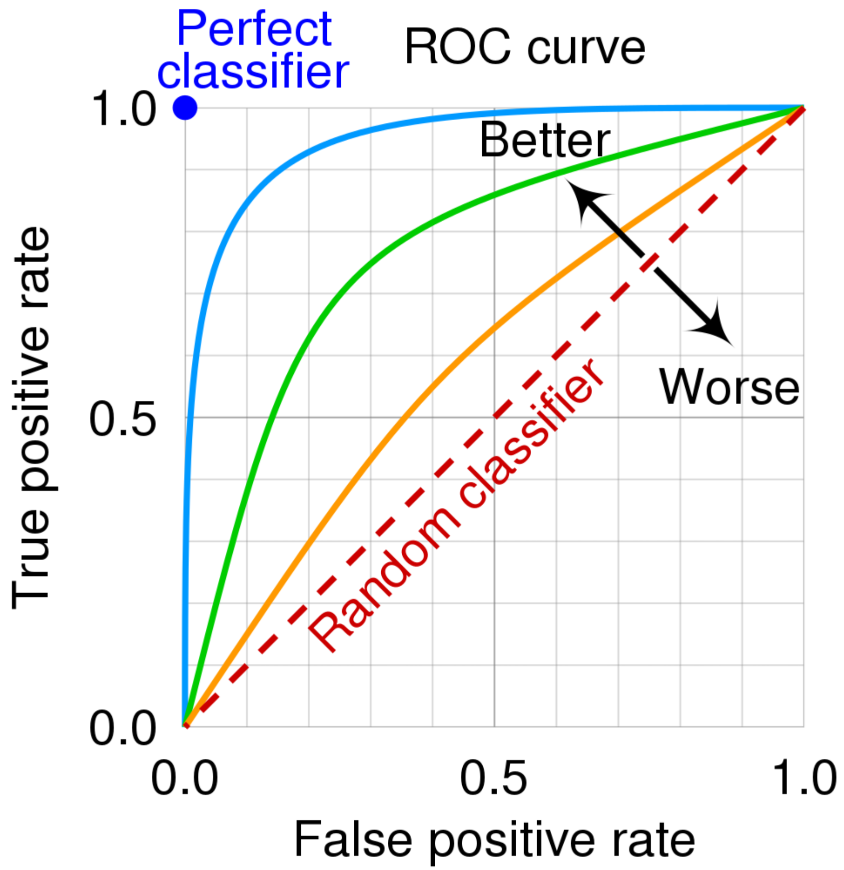
$p(\theta)$: Prior - subjective belief about θ

$p(y|\theta)$: Likelihood – observation (data) regarding θ

$p(\theta|y)$: Posterior - Updated belief about θ with the data

Introduction : Roc & Auc

	Predicted: Abnormal	Predicted: Normal
Actual: Abnormal	True Positive	False Negative
Actual: Normal	False Positive	True Negative



BPR : Main Idea

Main Idea : Formalization

U : 모든 users

I : 모든 items

$S \subseteq U \times I$: Implicit feedback이 관찰된 (u,i) 쌍의 집합

Iu +: : user u가 implicit feedback을 남긴 items

Ui + : item i에 implicit feedback을 남긴 users

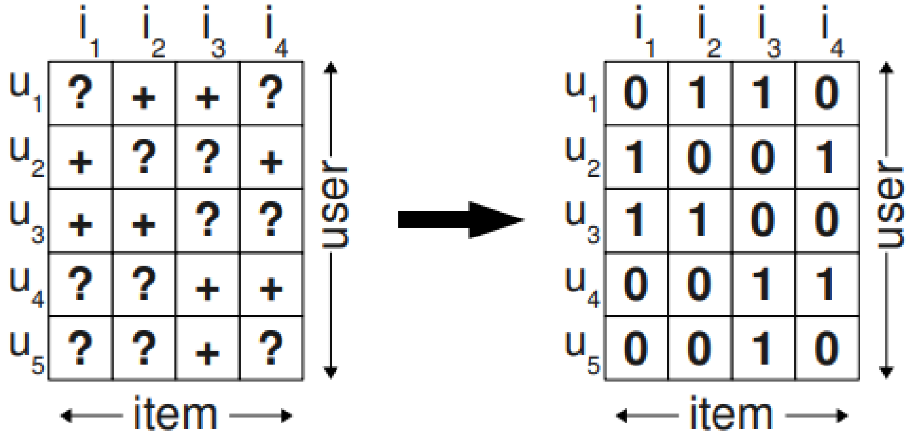
$>u$: user u의 선호도 비교

$$\forall i, j \in I : i \neq j \Rightarrow i >_u j \vee j >_u i \qquad (totality)$$

$$\forall i, j \in I : i >_u j \wedge j >_u i \Rightarrow i = j \qquad (antisymmetry)$$

$$\forall i, j, k \in I : i >_u j \wedge j >_u k \Rightarrow i >_u k \qquad (transitivity)$$

Main Idea : Formalization

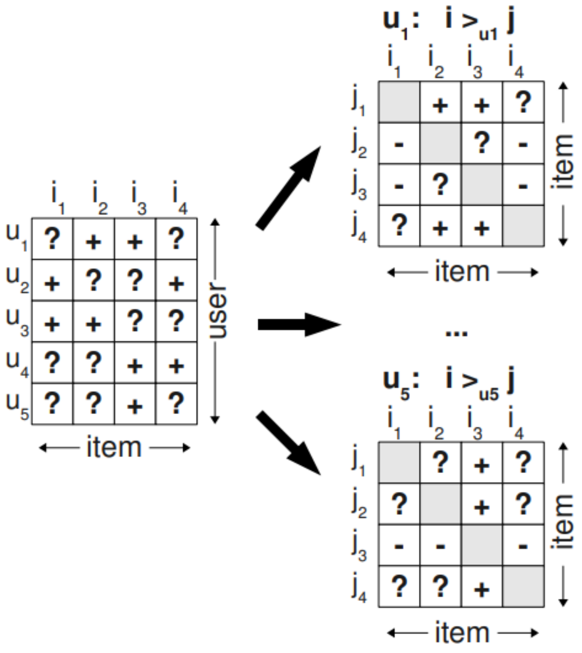


기존의 연구는 user x item matrix에 대해 각각의 선호도를 알아보며 sorting을 진행하는 방식이었다.

이에 따라 정말로 관심이 없는 **negative implicit feedback**과

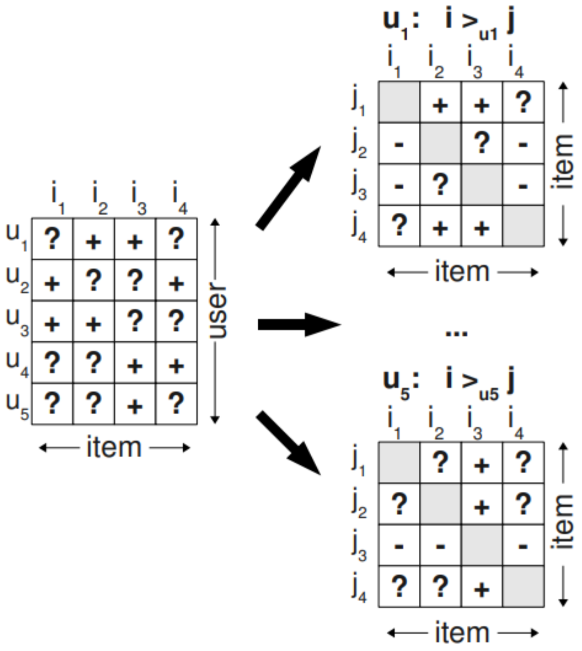
나중에 구매할 의향이 있을 **future missing value**를 구분하지 않고 **모두 0으로** 간주하게 되었다.

Main Idea : Formalization



각 user 별로 **item끼리 비교를 하면서** 선호 정도를 직접적으로 알아보자.

Main Idea : Formalization

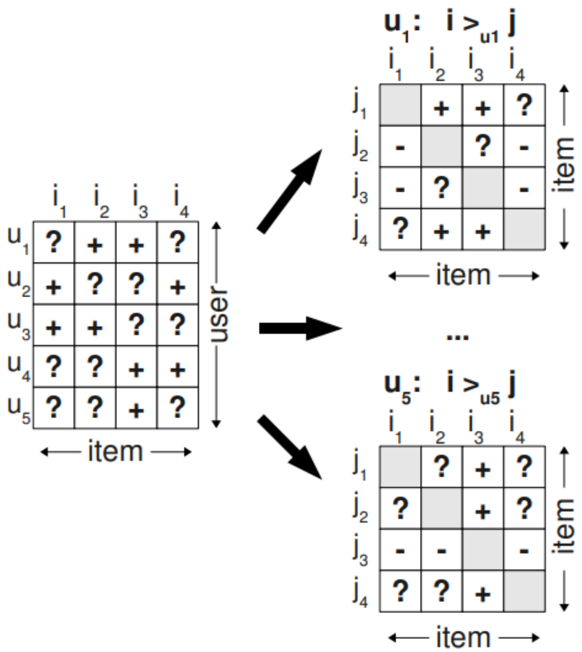


3가지 가정

1. 관측된 item은 관측되지 않은 item보다 선호된다.
2. 관측된 item사이 비교는 불가능하다.
3. 관측되지 않은 item사이의 비교 역시 불가능하다.

이전과 달리 negative feedback과 missing value가 구분된다.

Main Idea : Formalization



$$D_S := \{(u,i,j) | i \in I_u^+ \wedge j \in I \setminus I_u^+\}$$

$\wedge$: *and*
$\vee$: *or*
$\setminus$: *minus*

Recall : map

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

$$= \frac{p(y|\theta)p(\theta)}{\int_\theta p(y|\theta)p(\theta)d\theta} \qquad \left(\because p(y) = \int_\theta p(y|\theta)p(\theta)d\theta\right)$$

$$\propto p(y|\theta)p(\theta)$$

Posterior ∝ Likelihood X Prior

$p(\theta)$: Prior - subjective belief about θ

$p(y|\theta)$: Likelihood – observation (data) regarding θ

$p(\theta|y)$: Posterior - Updated belief about θ with the data

Likelihood와 prior를 구하자.

Main Idea : BPR Opt Criterion

$$p(\Theta | >_u) \propto p(>_u | \Theta)\, p(\Theta)$$

$$\prod_{u \in U} p(>_u | \Theta) = \prod_{(u,i,j) \in U \times I \times I} p(i >_u j | \Theta)^{\delta((u,i,j) \in D_S)} \cdot (1 - p(i >_u j | \Theta))^{\delta((u,j,i) \notin D_S)}$$

$$\delta(b) := \begin{cases} 1 & \text{if } b \text{ is true,} \\ 0 & \text{else} \end{cases}$$

각 User끼리 가지는 item들에 대한 선호도는 independet.

각 Item pair끼리의 비교 순서 역시 independent.

Main Idea : BPR Opt Criterion

$$p(\Theta| >_u) \propto p(>_u |\Theta)\, p(\Theta)$$

$$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in U \times I \times I} p(i >_u j|\Theta)^{\delta((u,i,j) \in D_S)} \cdot (1 - p(i >_u j|\Theta))^{\delta((u,j,i) \notin D_S)}$$

$$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j|\Theta)$$

By totality, antisymmetry

Main Idea : BPR Opt Criterion

$$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j|\Theta)$$

$$p(i >_u j|\Theta) := \sigma(\hat{x}_{uij}(\Theta))$$

Sigmoid function + real-valued function

Properties 충족

u,i,j 사이 relationship estimate.

Ex) MF, KNN

Main Idea : BPR Opt Criterion

$$p(\Theta) \sim N(0, \Sigma_\Theta)$$

$$\Sigma_\Theta = \lambda_\Theta I$$

$$N(\Theta|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(\Theta - \mu)^T \Sigma^{-1}(\Theta - \mu))$$

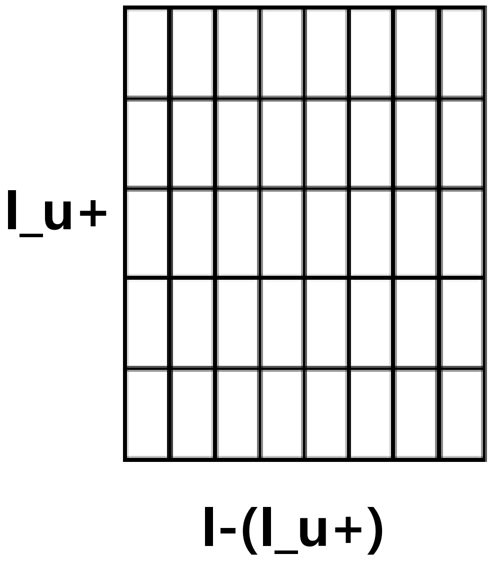
$$\propto exp(-\frac{1}{2}\Theta^T(\frac{1}{\lambda_\Theta} I)\Theta)$$

$$= exp(-\frac{1}{2\lambda_\Theta}\Theta^T\Theta) \simeq exp(-\lambda_\Theta \|\Theta\|^2)$$

Main Idea : BPR Opt Criterion

$$
\begin{aligned}
\text{BPR-Opt} :&= \ln p(\Theta | >_u) \\
&= \ln p(>_u | \Theta)\, p(\Theta) \\
&= \ln \prod_{(u,i,j) \in D_S} \sigma(\hat{x}_{uij})\, p(\Theta) \\
&= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
&= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2
\end{aligned}
$$

Main Idea : AUC analogy



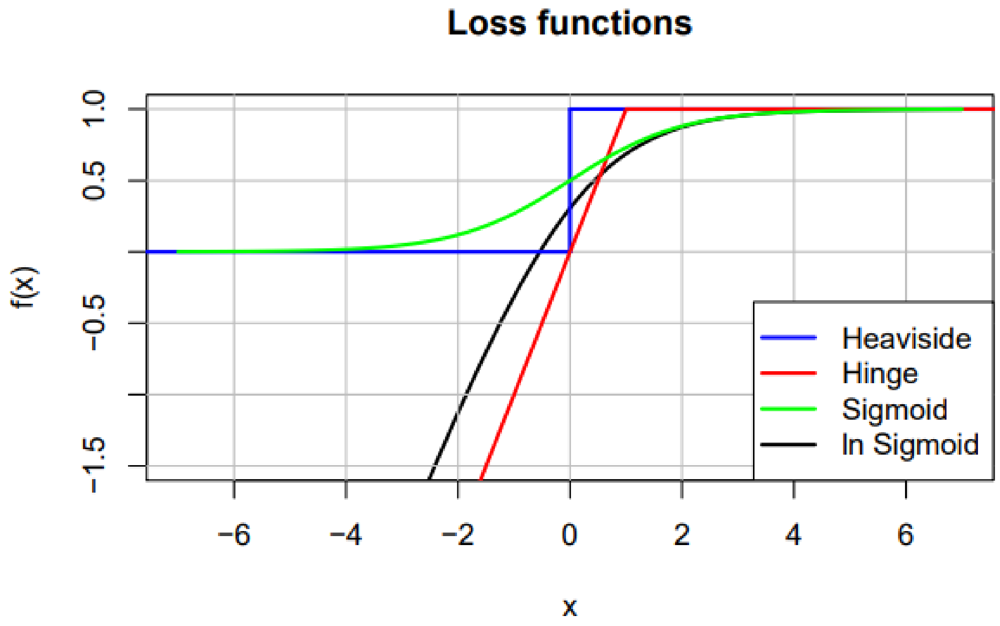
$$\mathrm{AUC}(u) := \frac{1}{|I_u^+|\,|I \setminus I_u^+|} \sum_{i \in I_u^+} \sum_{j \in |I \setminus I_u^+|} \delta(\hat{x}_{uij} > 0)$$

$$\mathrm{AUC} := \frac{1}{|U|} \sum_{u \in U} AUC(u)$$

$$\mathrm{AUC}(u) = \sum_{(u,i,j) \in D_S} z_u \, \delta(\hat{x}_{uij} > 0) \qquad \delta(x > 0) = H(x) := \begin{cases} 1, & x > 0 \\ 0, & \text{else} \end{cases}$$

z와 loss function만의 차이를 보인다.

Main Idea : AUC analogy



Main Idea : BPR Learning Algorithm

$$\frac{\partial \text{BPR-Opt}}{\partial \Theta} = \sum_{(u,i,j)\in D_S} \frac{\partial}{\partial \Theta} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \frac{\partial}{\partial \Theta} ||\Theta||^2$$

$$\propto \sum_{(u,i,j)\in D_S} \frac{-e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} - \lambda_\Theta \Theta$$

Full gradient descent

계산량이 많다.

learning rate를 낮게 잡아야한다. (observed data << non ovserved data ; skewness of x_uij)

Stochastic Gradient descent with Bootstrap

계산량이 상대적으로 적다.

User-wise, item-wise같이 consecutive한 update을 가능한 회피한다.

조기종료 가능

Main Idea : Learning models with BPR

$$\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$$

x_ui, x_uj를 MF/KNN을 접목하여 얻고,

둘의 차를 criterion으로 두어 optimize한다.

Main Idea : BPR with MF

$$\hat{X} := WH^t \quad W : |U| \times k \qquad H : |I| \times k$$

$$\hat{x}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} \cdot h_{if}$$

$$\frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} (h_{if} - h_{jf}) & \text{if } \theta = w_{uf}, \\ w_{uf} & \text{if } \theta = h_{if}, \\ -w_{uf} & \text{if } \theta = h_{jf}, \\ 0 & \text{else} \end{cases}$$

Main Idea : BPR with KNN

$$\hat{x}_{ui} = \sum_{l \in I_u^+ \wedge l \neq i} c_{il}$$

$C : I \times I$ item-similarity matrix

$$c_{i,j}^{\text{cosine}} := \frac{|U_i^+ \cap U_j^+|}{\sqrt{|U_i^+| \cdot |U_j^+|}}$$

$$\frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} +1 & \text{if } \theta \in \{c_{il}, c_{li}\} \wedge l \in I_u^+ \wedge l \neq i, \\ -1 & \text{if } \theta \in \{c_{jl}, c_{lj}\} \wedge l \in I_u^+ \wedge l \neq j, \\ 0 & \text{else} \end{cases}$$

Main Idea : BPR vs WRMF

$$\sum_{u \in U} \sum_{i \in I} c_{ui}(\langle w_u, h_i \rangle - 1)^2 + \lambda ||W||_f^2 + \lambda ||H||_f^2$$

SVD + regularization term (overfitting 방지)

Weights on Positive implicit feedback

그러나 **one – item level** (not pair)이고

SVD(square-loss)기반은 quantative problem에 적합하기에

Qualitative(classification) problem에 logistic approach보다 불리하다.

Main Idea : BPR vs MMMF

$$\sum_{(u,i,j)\in D_s} \max(0, 1-\langle w_u, h_i - h_j\rangle)+\lambda_w||W||_f^2+\lambda_h||H||_f^2$$

Explicit feedback기반 approach.

Matrix factorization만이 접목 가능(범용성이 상대적으로 낮다.)

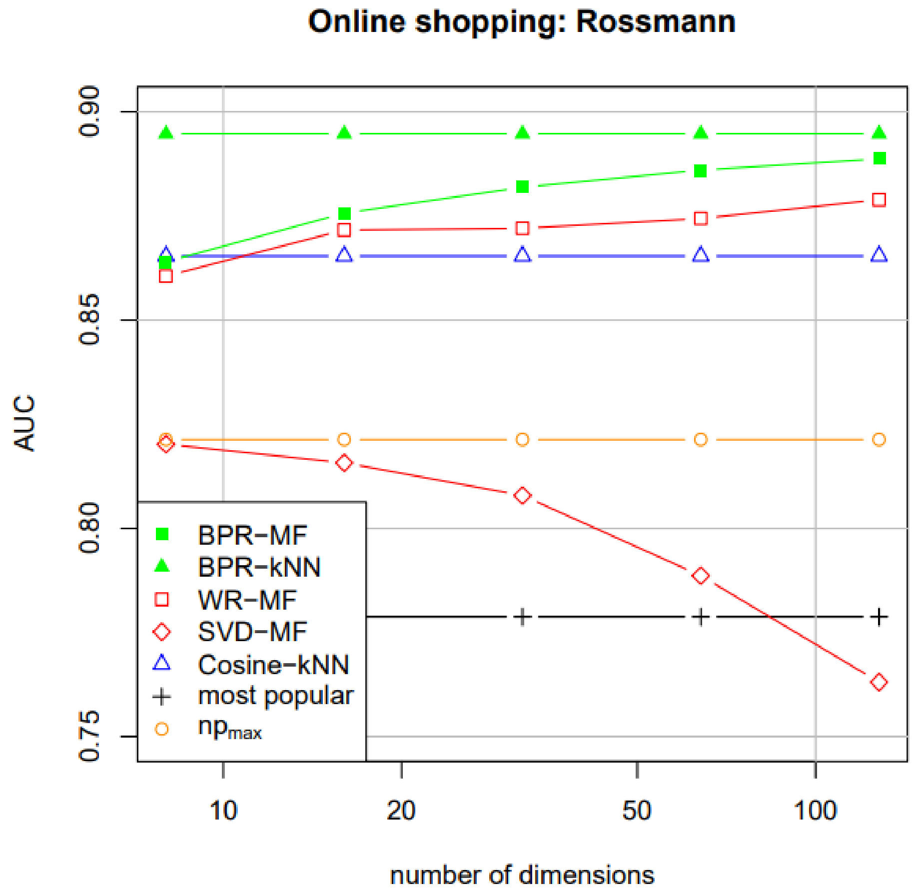
Main Idea : Evaluation

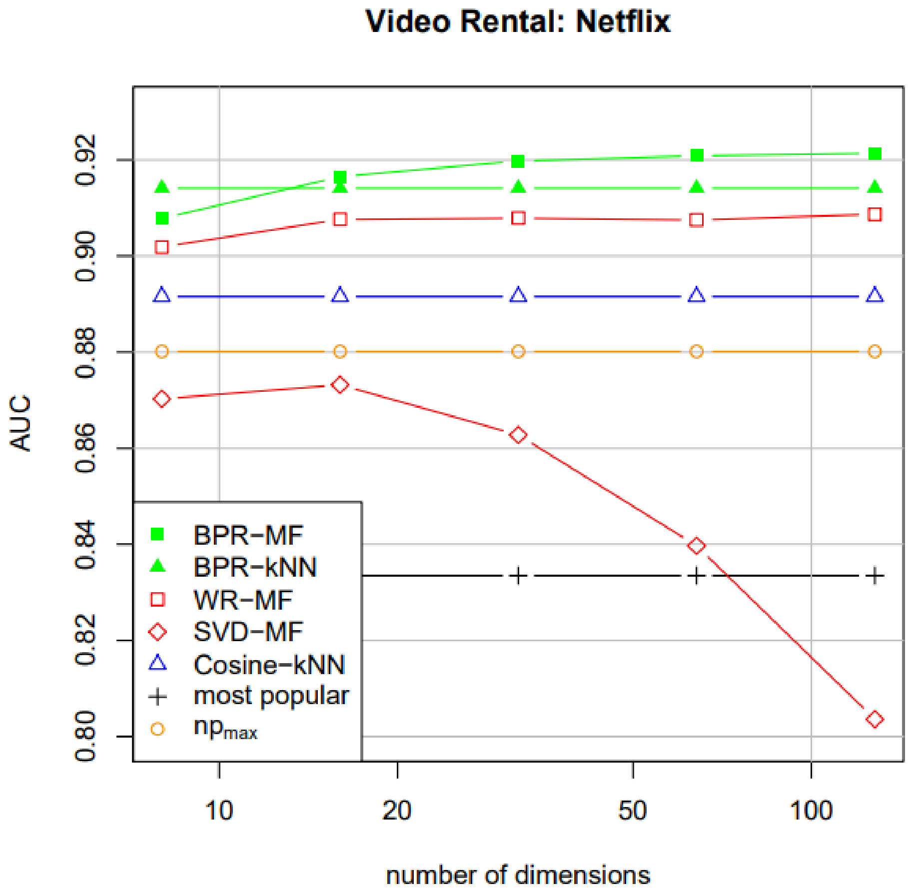
$$\mathrm{AUC} = \frac{1}{|U|} \sum_{u} \frac{1}{|E(u)|} \sum_{(i,j) \in E(u)} \delta(\hat{x}_{ui} > \hat{x}_{uj})$$

LOO scheme 이용.

Rated Users<10 인 Items or Rating Items < 10 Users는 제외.

Main Idea : Conclusion





Main Idea : Conclusion

SVD-MF,WR-MF,BPR-MF 모두 MF를 활용하였으나

그 중에서 특히 BPR-MF의 AUC 값이 월등히 높았다.

어떠한 criterion을 가지고 optimize하는지가 굉장히 중요하다.

Ranking에 대해서는 pair끼리의 비교를 기준으로 하는 것이 유효하다.

BPR : Implementation

Implementation : Preprocessing

```
ratings = pd.read_csv(os.path.join(INPUT_DIR, 'ratings.csv'))
ratings_df = ratings.drop(columns=['timestamp'])

#userId x moveId , cell : rating인 matrix
table_df = ratings_df.pivot_table(index='userId', columns='movieId', values ='rating')
df_matrix = table_df.to_numpy()
df_matrix = np.nan_to_num(df_matrix)

#implicit : 0과 1로만 표현
df_matrix[df_matrix>0]=1
```

User x item matrix >> Item x Item matrix (each users)

Binary to be Implicit

Implementation : Preprocessing

```
row_sums = np.sum(df_matrix,axis=1)
col_sums = np.sum(df_matrix,axis=0)

row_keep = row_sums>=10
col_keep = col_sums>=10

user_movie_data = df_matrix[row_keep][:,col_keep]

test_set = np.zeros(user_movie_data.shape)
test_pair = []
for row in range (user_movie_data.shape[0]):
  test_ele = np.random.randint(0,user_movie_data.shape[1])
  test_set[row][test_ele] = user_movie_data[row][test_ele]
  user_movie_data[row][test_ele] = 0
  test_pair.append([row,test_ele])
```

Rate users < 10 Items & Rate Items < 10 Users 배제

Test data를 loo scheme으로 구성.

Implementation : BPR-MF

```python
class BPR_MF:
  def __init__(self,learning_rate = 0.01, features = 20, lbd = 0.01, data = None):
    self.learning_rate = learning_rate
    self.features = features
    self.lbd = lbd
    self.data = data
    self.user_feat = None
    self.item_feat = None
    print("features")
    print(self.features)
```

Implementation : BPR-MF

```
def fit(self):
    #일단 user x feature, item x feature를 초기에 random하게 생성
    user_feat = np.random.random((self.data.shape[0],self.features))
    item_feat = np.random.random((self.data.shape[1],self.features))
    auc = []
    predicted = []
    truevalue = []
    epochs = []
```

Implementation : BPR-MF

```python
for itr in range(1000001):
  u = np.random.choice(range(self.data.shape[0]))
  u_feat = copy.deepcopy(self.data[u])
  i = np.random.choice(np.where(u_feat==1)[0])
  j = np.random.choice(np.where(u_feat==0)[0])

  w_u = user_feat[u,:]
  h_i = item_feat[i,:]
  h_j = item_feat[j,:]

  x_uij = np.dot(w_u,h_i) - np.dot(w_u,h_j)
  exp = np.exp(-x_uij) / (1 + np.exp(-x_uij))

  grad_wu = exp * (h_i-h_j) + self.lbd * w_u
  user_feat[u,:] = user_feat[u,:] + self.learning_rate * grad_wu

  grad_hi = exp * (w_u) + self.lbd * h_i
  item_feat[i,:] = item_feat[i,:] + self.learning_rate * grad_hi

  grad_hj = exp * (-w_u) + self.lbd * h_j
  item_feat[j,:] = item_feat[j,:] + self.learning_rate * grad_hj
```

$$\frac{\partial}{\partial\theta}\hat{x}_{uij} = \begin{cases} (h_{if} - h_{jf}) & \text{if } \theta = w_{uf}, \\ w_{uf} & \text{if } \theta = h_{if}, \\ -w_{uf} & \text{if } \theta = h_{jf}, \\ 0 & \text{else} \end{cases}$$

Implementation : BPR-MF

```python
  if(x_uij>0):
    predicted.append(1)
  else:
    predicted.append(0)
  #해당 유저의 movie i와 movie j를 비교해서 보자.
  if(self.data[u][i]==1 and self.data[u][j]==0):
    truevalue.append(1)
  else:
    truevalue.append(0)
  if(itr%100000==0):
    if not(sum(truevalue)==0 and itr !=0):
      epochs.append(itr)
      auc.append(sum(predicted)/sum(truevalue))
      print(str(itr) + "th auc is : " + str(auc[-1]))



self.user_feat = user_feat
self.item_feat = item_feat

return epochs, auc
```

Training data set에 대하여 AUC 계산

Implementation : BPR-MF

```
def test(self,pair,test_set):
  #pair에 있는 애들만 가지고 진행한다.
  test_predict=[]
  gr_truth = []
  tst_epchs = []
  rss = 0

  auc_test = 0
  for m in range(len(pair)):
    #나머지 중에 item고르기
    num_range = np.array(list(range(len(pair))))
    filter_num = np.delete(num_range, np.where(num_range==pair[m][1]))
    another = np.random.choice(filter_num)
    test_user_feat = self.user_feat[m,:]
    test_i_feat = self.item_feat[pair[m][1],:]
    test_j_feat = self.item_feat[another,:]
    test_xuij = np.dot(test_user_feat,test_i_feat) - np.dot(test_user_feat,test_j_feat)
```

만들어진 train matrix에 대입

Implementation : BPR-MF

```
    if(test_set[m][pair[m][1]]==1 and self.data[m][another]==0):
      gr_truth.append(1)
      if(test_xuij>0):
        test_predict.append(1)
      else:
        test_predict.append(0)

auc_test = np.sum(test_predict)/np.sum(gr_truth)
return auc_test
```

Ground truth이 1인 경우에 prediction은 어떤 결과를 도출했는지를 비율로써 비교한다.

Implementation : BPR-KNN

```
class BPR_KNN:
  def __init__(self,learning_rate = 0.01, lbd = 0.01, data = None):
    self.learning_rate = learning_rate
    self.lbd = lbd
    self.data = data
    self.c_item = np.random.random((self.data.shape[1],self.data.shape[1]))
```

Implementation : BPR-KNN

```python
u = np.random.choice(range(self.data.shape[0]))
u_feat = copy.deepcopy(self.data[u])
i = np.random.choice(np.where(u_feat==1)[0])
j = np.random.choice(np.where(u_feat==0)[0])

#C에서 읽도록 u가 ratings에서 (i제외) 평가했던 항목들의 위치를 읽어온다.
compare_list_temp_i = np.nonzero(self.data[u])[0]
compare_location_i = compare_list_temp_i[compare_list_temp_i!=i]

#c_il 들 의 위치를 위해 우선 ith row를 지정하고 나머지는 compare list의 위치로 찾아준다.
compare_list_i = self.c_item[i][compare_location_i]
x_ui = np.sum(compare_list_i)

#j에대해 동일과정 반복. [0]은 (list,공백)인 형태로 나와서 해준거다.
compare_list_temp_j = np.nonzero(self.data[u])[0]
compare_location_j = compare_list_temp_j[compare_list_temp_j!=j]

compare_list_j = self.c_item[j][compare_location_j]
x_uj = np.sum(compare_list_j)

x_uij = x_ui-x_uj
```

$$\hat{x}_{ui} = \sum_{l \in I_u^+ \wedge l \neq i} c_{il}$$

Implementation : BPR-KNN

```python
u = np.random.choice(range(self.data.shape[0]))
u_feat = copy.deepcopy(self.data[u])
i = np.random.choice(np.where(u_feat==1)[0])
j = np.random.choice(np.where(u_feat==0)[0])

#C에서 읽도록 u가 ratings에서 (i제외) 평가했던 항목들의 위치를 읽어온다.
compare_list_temp_i = np.nonzero(self.data[u])[0]
compare_location_i = compare_list_temp_i[compare_list_temp_i!=i]

#c_il 들 의 위치를 위해 우선 ith row를 지정하고 나머지는 compare list의 위치로 찾아준다.
compare_list_i = self.c_item[i][compare_location_i]
x_ui = np.sum(compare_list_i)

#j에대해 동일과정 반복. [0]은 (list,공백)인 형태로 나와서 해준거다.
compare_list_temp_j = np.nonzero(self.data[u])[0]
compare_location_j = compare_list_temp_j[compare_list_temp_j!=j]

compare_list_j = self.c_item[j][compare_location_j]
x_uj = np.sum(compare_list_j)

x_uij = x_ui-x_uj
```

$$\hat{x}_{ui} = \sum_{l \in I_u^+ \wedge l \neq i} c_{il}$$

Implementation : BPR-KNN

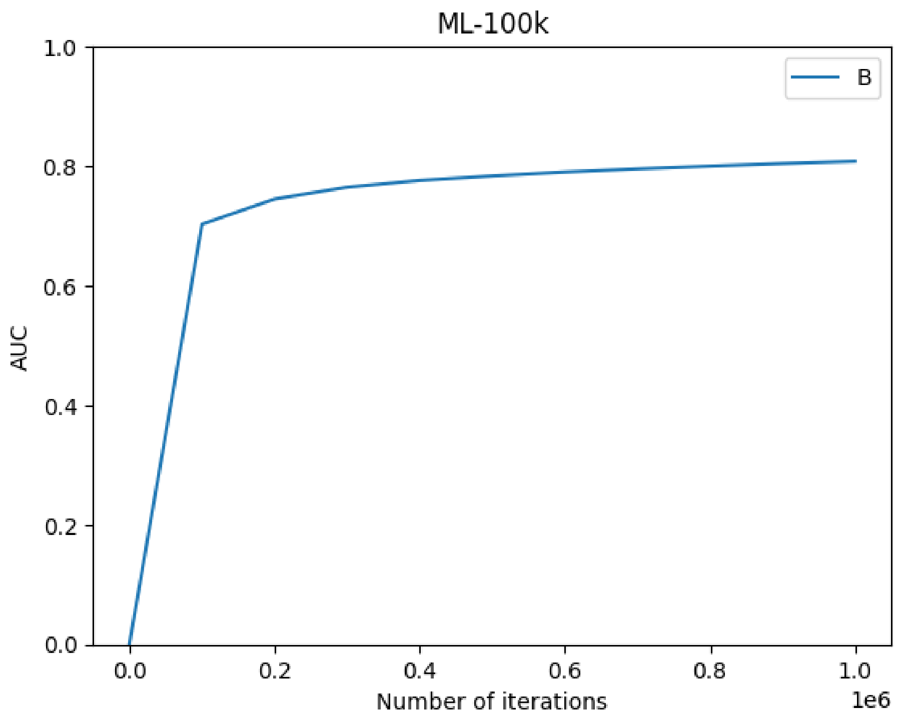
```
x_uij = x_ui-x_uj
exp = np.exp(-x_uij) / (1 + np.exp(-x_uij))

grad_ci = exp * 1 + self.lbd * self.c_item[i, compare_location_i]
grad_cj = exp * (-1) + self.lbd * self.c_item[j, compare_location_j]

#이제 갱신을 해주었다. c를
self.c_item[i,compare_location_i] = self.c_item[i,compare_location_i] + self.learning_rate * grad_ci
self.c_item[j,compare_location_j] = self.c_item[j,compare_location_j] + self.learning_rate * grad_cj
```

$$\frac{\partial}{\partial\theta}\hat{x}_{uij} = \begin{cases} +1 & \text{if } \theta \in \{c_{il}, c_{li}\} \wedge l \in I_u^+ \wedge l \neq i, \\ -1 & \text{if } \theta \in \{c_{jl}, c_{lj}\} \wedge l \in I_u^+ \wedge l \neq j, \\ 0 & \text{else} \end{cases}$$

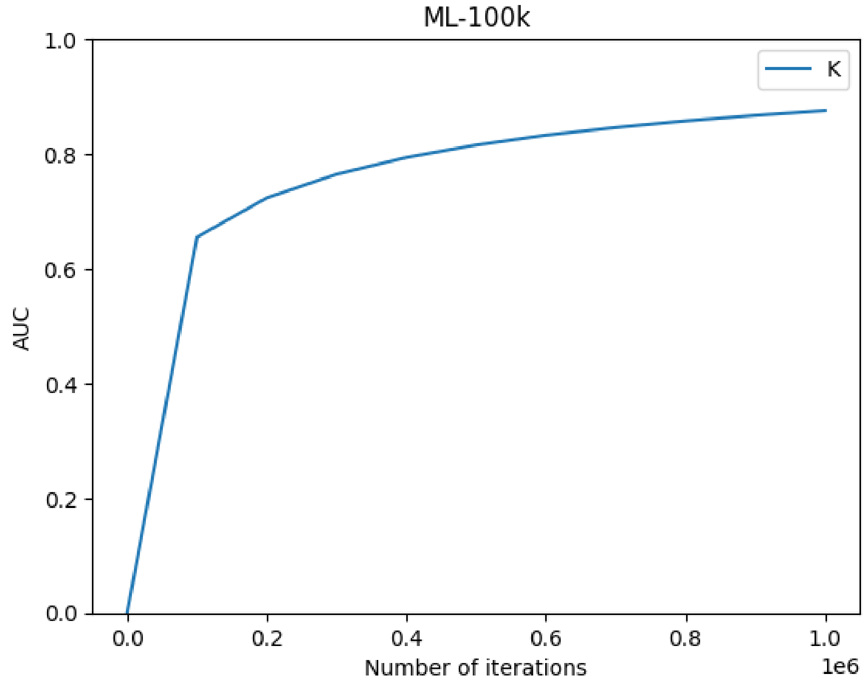
Implementation : BPR-MF



Train AUC : 0.81

Test AUC : 0.59

Implementation : BPR-KNN



Train AUC : 0.88

Test AUC : 0.64

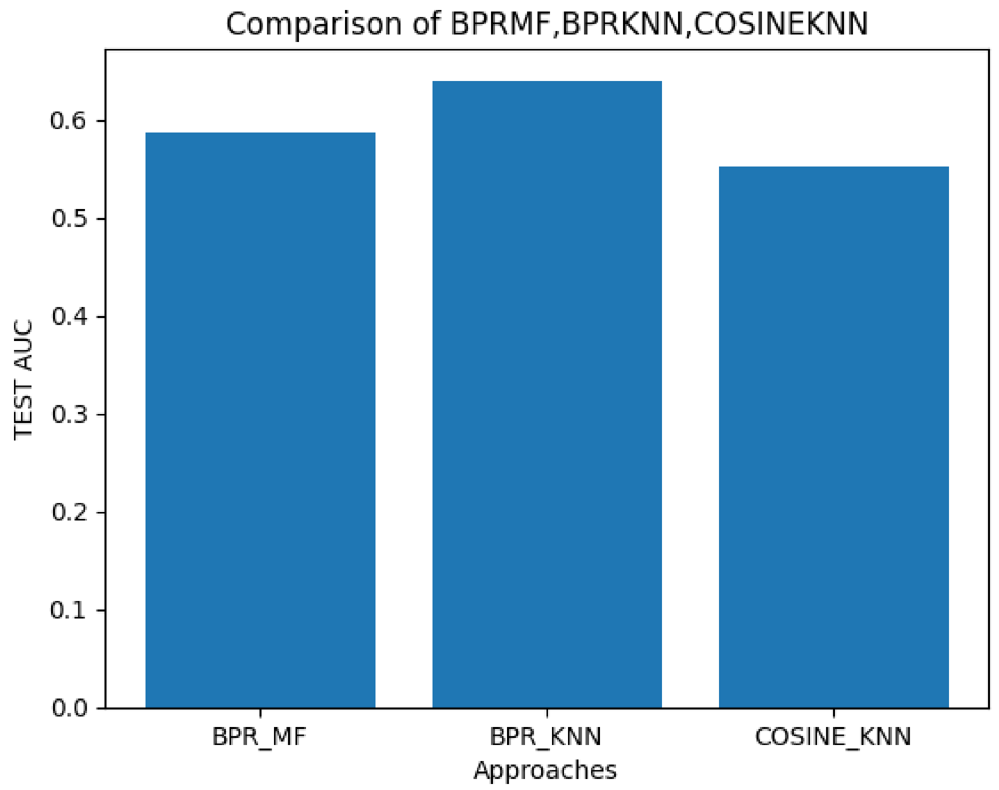
Implementation : COSINE-KNN

```
class COSINE_KNN:
  def __init__(self,learning_rate = 0.01, lbd = 0.01, data = None):
    self.learning_rate = learning_rate
    self.lbd = lbd
    self.data = data
    item_data = data.T
    product = np.dot(item_data,item_data.T)
    product_norm = np.linalg.norm(product, ord=1, axis=1, keepdims=True)
    norm = np.linalg.norm(item_data, ord=1, axis=1, keepdims=True)
    self.c = product / np.sqrt(norm * norm.T)
```

Train AUC : 0.88

Test AUC : 0.55

Implementation : Results



Thank you