



A Tutorial on Wikimedia Visual Resources and its Application to Neural Visual Recommender Systems

Denis Parra[1], Antonio Ossa-Guerra[1], Manuel Cartagena[1], **Patricio Cerda-Mardini**[2], Felipe del Río[1], Isidora Palma[1], Diego Saez-Trumper[3], and Miriam Redi[3]

1. Pontificia Universidad Católica de Chile
2. mindsdb
3. Wikimedia Foundation

21st IEEE International Conference on Data Mining

ICDM 2021
IEEE International Conference on Data Mining
7 – 10 DECEMBER 2021
AUCKLAND NEW ZEALAND

Session 3: Visual BPR methods

Denis Parra[1], Antonio Ossa-Guerra[1], Manuel Cartagena[1], **Patricio Cerda-Mardini**[2], Felipe del Río[1], Isidora Palma[1], Diego Saez-Trumper[3], and Miriam Redi[3]

1. Pontificia Universidad Católica de Chile
2. mindsdb
3. Wikimedia Foundation

21st IEEE International Conference on Data Mining

ICDM 2021
IEEE International Conference on Data Mining
7 – 10 DECEMBER 2021
AUCKLAND NEW ZEALAND

VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback

Ruining He
UC San Diego
r4he@ucsd.edu

Julian McAuley
UC San Diego
jmcauley@ucsd.edu

Context

- Previous work did not consider the *visual appearance* of the items

- Modeling derived from BPR, suitable to uncover visual factors

- Addressing cold-start problem

VBPR Key insights

- Each item is represented by its visual features, from a pretrained AlexNet

- Preference predictor is a complex term that can be simplified thanks to BPR

Model: Preference predictor

$$\hat{x}_{u,i} = \alpha + \beta_u + \beta_i + \gamma_u^T \gamma_i + \theta_u^T (\mathbf{E} f_i) + \beta'^T f_i$$

α — Global offset

$\beta_u + \beta_i$ — Bias terms

$\gamma_u^T \gamma_i$ — Latent factors (compatibility between user and item)

Model: Preference predictor

$$\hat{x}_{u,i} = \alpha + \beta_u + \beta_i + \gamma_u^T \gamma_i + \theta_u^T (\mathbf{E} f_i) + \beta'^T f_i$$

$\theta_u^T (\mathbf{E} f_i)$

Visual factors
(user preference over visual dimensions)

$\beta'^T f_i$

Users' visual bias
(user's overall opinion towards visual appearance of a given item)

Model: Preference predictor while training

$$\hat{x}_{uij} = \hat{x}_{u,i} - \hat{x}_{u,j}$$

$$\hat{x}_{u,i} = \alpha + \beta_u + \beta_i + \gamma_u^T \gamma_i + \theta_u^T (\mathbf{E} f_i) + \beta'^T f_i$$

$$\hat{x}_{u,j} = \alpha + \beta_u + \beta_j + \gamma_u^T \gamma_j + \theta_u^T (\mathbf{E} f_j) + \beta'^T f_j$$

Model diagram

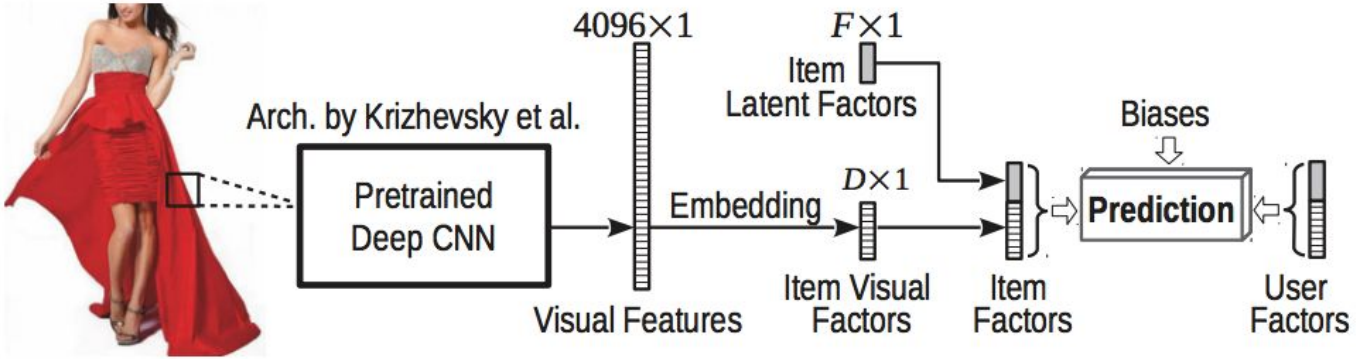


Figure 1: Diagram of our preference predictor. Rating dimensions consist of visual factors and latent (non-visual) factors. Inner products between users and item factors model the compatibility between users and items.

Loss function

$$\hat{x}_{u,i} = \beta_i + \gamma_u^T \gamma_i + \theta_u^T (E f_i) + \beta'^T f_i$$

Weights are learned using BPR-OPT (Rendle et al., 2009)

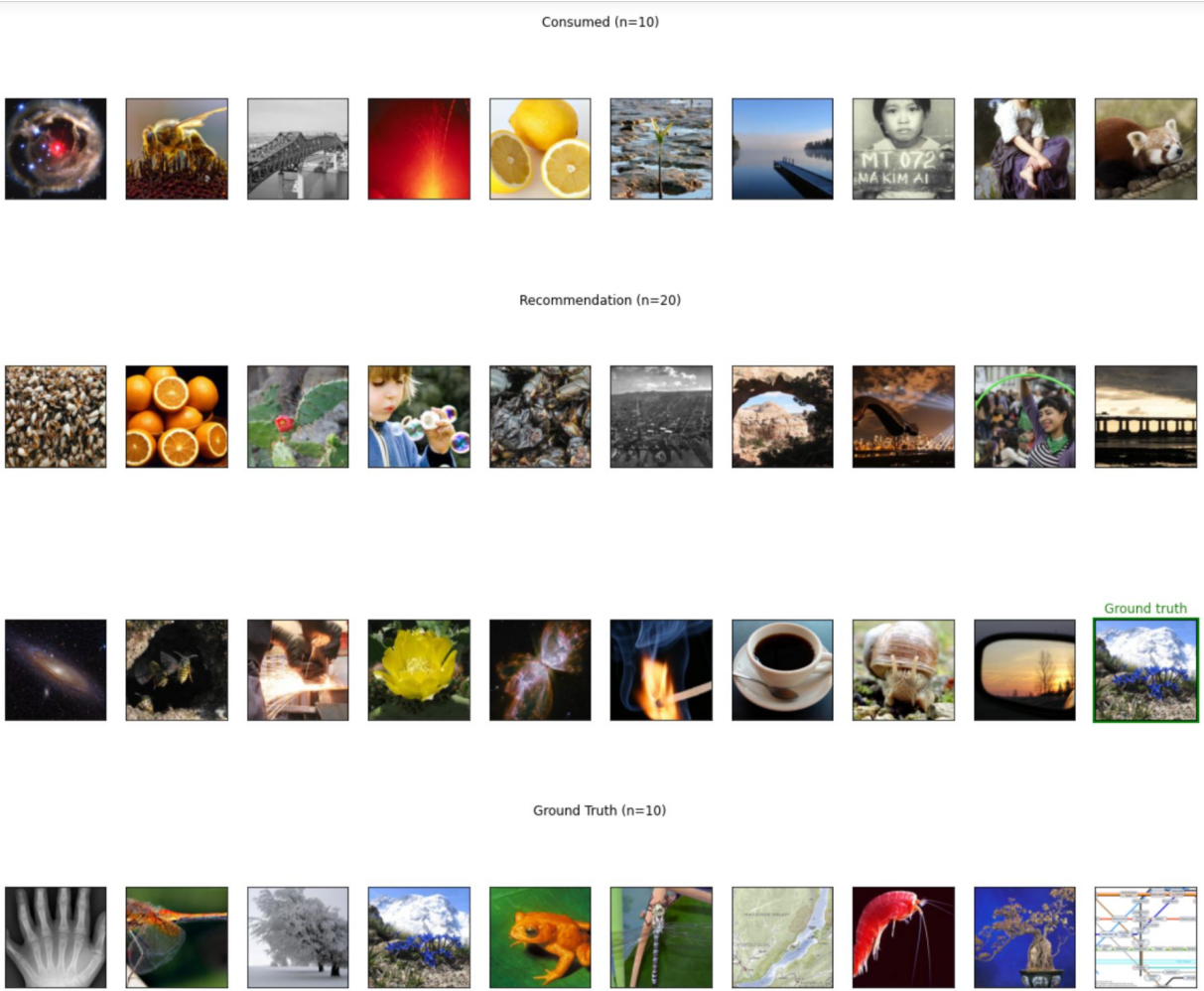
$$D_S = \{(u, i, j) | u \in U \wedge i \in I_u^+ \wedge j \in I \setminus I_u^+\}$$

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

Main Results

AUC	RR	R@20	P@20	nDCG@20	R@100	P@100	nDCG@100
0.75033	0.02351	0.00719	0.00429	0.02717	0.03742	0.00438	0.09059

Example recommendation



VBPR Takeaways

- Demonstrate usefulness of visual information for implicit feedback recommendation tasks

- Scalable, simple yet effective against cold start

Visually-Aware Fashion Recommendation and Design with Generative Image Models

Wang-Cheng Kang
UC San Diego
wckang@eng.ucsd.edu

Chen Fang
Adobe Research
cfang@adobe.com

Zhaowen Wang
Adobe Research
zhawang@adobe.com

Julian McAuley
UC San Diego
jmcauley@eng.ucsd.edu

a.k.a. *DVBPR*:
Deep Visually-aware Bayesian Personalized Ranking

About the paper

- Accepted at the International Conference on Data Mining (ICDM), 2017
- Authors: Adobe Research & Prof. McAuley's Lab @ UCSD
- Proposes fashion 1) recommendation and 2) design (through GANs)
- We focus on 1)

Methodology

Source: Kang et al. 2017

Context

- Fashion domain is complex: long tails, cold starts, evolving dynamics

- Content-aware recommender systems are well-suited to it

Source: Kang et al. 2017

DVBPR Key Insights

- Opt for “domain-aware” visual embeddings instead of “off-the-shelf” as in VBPR
- Joint training of visual embeddings and recommender system
- Recommending new items consistent with each user’s preference

Approach

- BPR framework: optimize rank of purchased vs non-purchased items
- Siamese trainable CNNs contrast positive and negative pairs
  - Images are retrieved and rescaled in the DataLoader
- Original datasets: Amazon fashion + Tradesy
- In this tutorial: updated Wikimedia Commons dataset

Model

- Users $u \in \mathcal{U}$
- Items $i \in \mathcal{I}$
- Positive items $\mathcal{I}_u^+$
- Item image X_i

VBPR:

$$x_{u,i} = \beta_i + \gamma_u^T \gamma_i + \theta_u^T (E \cdot f_i)$$

DVBPR:

$$x_{u,i} = \beta_i + \gamma_u^T \gamma_i + \theta_u^T \phi(X_i)$$

Model

VBPR:

Item bias

Latent CF factors

Visual factors

$$x_{u,i} = \beta_i + \gamma_u^T \gamma_i + \theta_u^T (E \cdot f_i)$$

Dimensionality reduction

Pre-trained embedding

DVBPR:

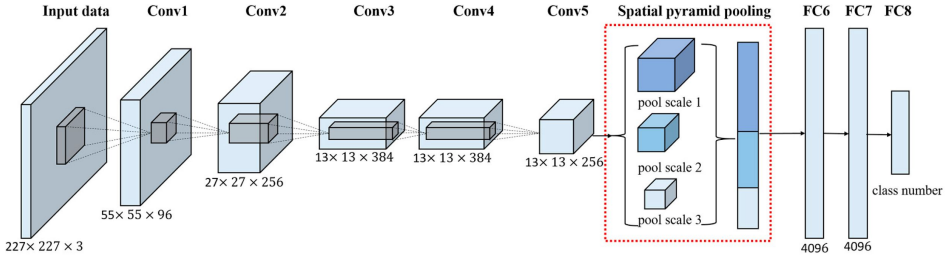
$$x_{u,i} = \beta_i + \gamma_u^T \gamma_i + \theta_u^T \phi(X_i)$$

We keep this!

Trainable CNN

Convolutional Neural Networks

- We use AlexNet with K=100



AlexNet. Source: Han et al. 2017

- Paper uses CNN-F with K=50

conv1	conv2	conv3	conv4	conv5	full6	full7	full8
64x11x11	256x5x5	256x3x3	256x3x3	256x3x3	4096	4096	K
st. 4, pad 0	st. 1, pad 2	st. 1, pad 1	st. 1, pad 1	st. 1, pad 1	drop-	drop-	-
x2 pool	x2 pool	-	-	x2 pool	out	out	-

CNN-F. Source: Kang et al. 2017

BPR Optimization

$$u \in \mathcal{U} \qquad i \in \mathcal{I}_u^+ \qquad j \in \mathcal{I} \setminus \mathcal{I}_u^+ \qquad (u,i,j) \in \mathcal{D}$$

$$\mathcal{D} = \{(u,i,j) | u \in \mathcal{U} \wedge i \in \mathcal{I}_u^+ \wedge j \in \mathcal{I} \setminus \mathcal{I}_u^+\}$$

$$\max \sum_{(u,i,j) \in \mathcal{D}} \ln \sigma(x_{u,i,j}) - \lambda_\Theta \|\Theta\|^2$$

$$x_{u,i,j} = x_{u,i} - x_{u,j}$$

Retrieval / Recommendation

$$\delta(u, c) = \underset{i \in X_c}{\text{argmax}}\ x_{u,i} = \underset{i \in X_c}{\text{argmax}}\ \beta_i + \gamma_u^T \gamma_i + \theta_u^T \phi(X_i)$$

Observations

- Model converges after 5 epochs (~12 hours on an 8-core CPU + GTX 1080 Ti)

- In our experience, latent CF factors are crucial for the model to learn

Datasets

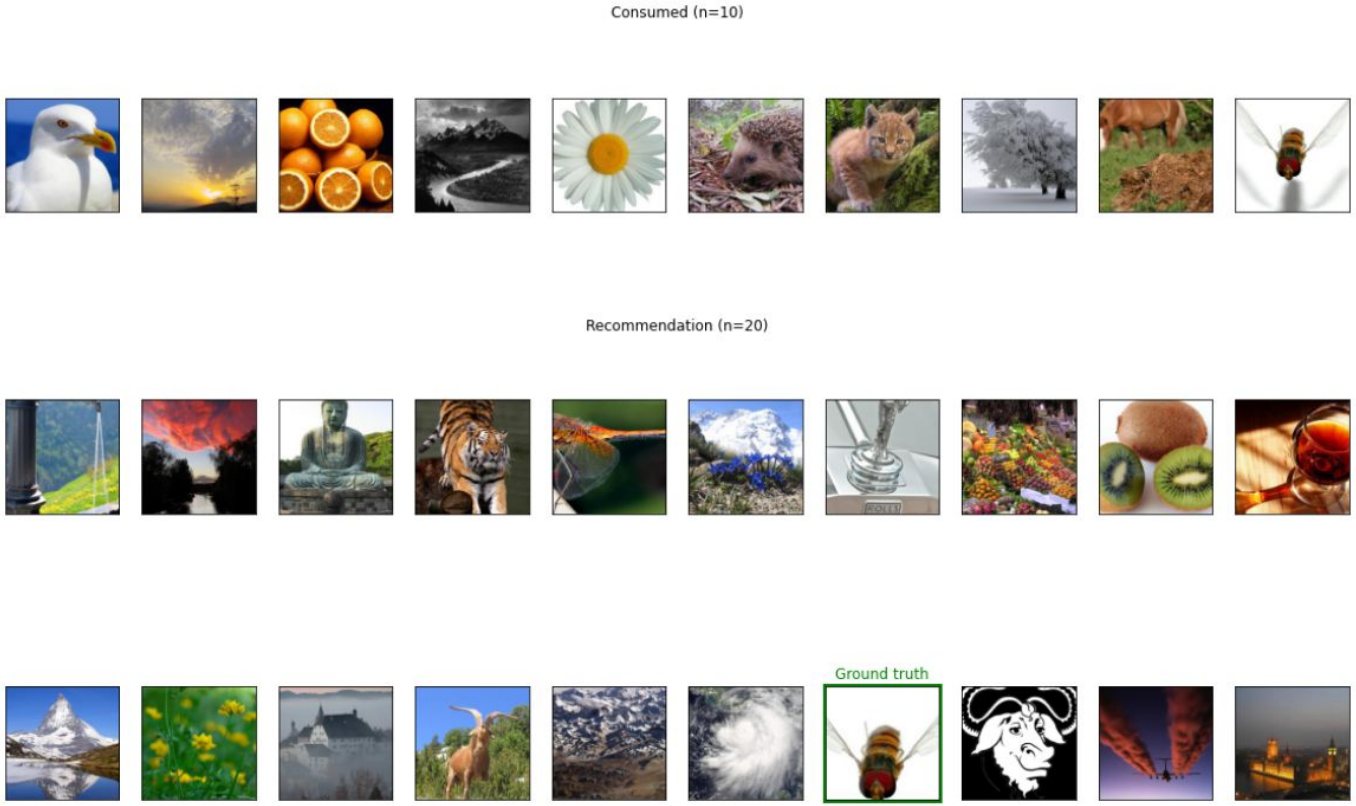
Dataset	# Users	# Items	# Interactions	# Categories
Amazon Fashion	64583	234892	513367	6
Amazon Women	97678	347591	827678	53
Amazon Men	34244	110636	254870	50
Tradesy.com	33864	326393	655409	N / A
Wikimedia (*update*)	1078	32959	96991	N / A

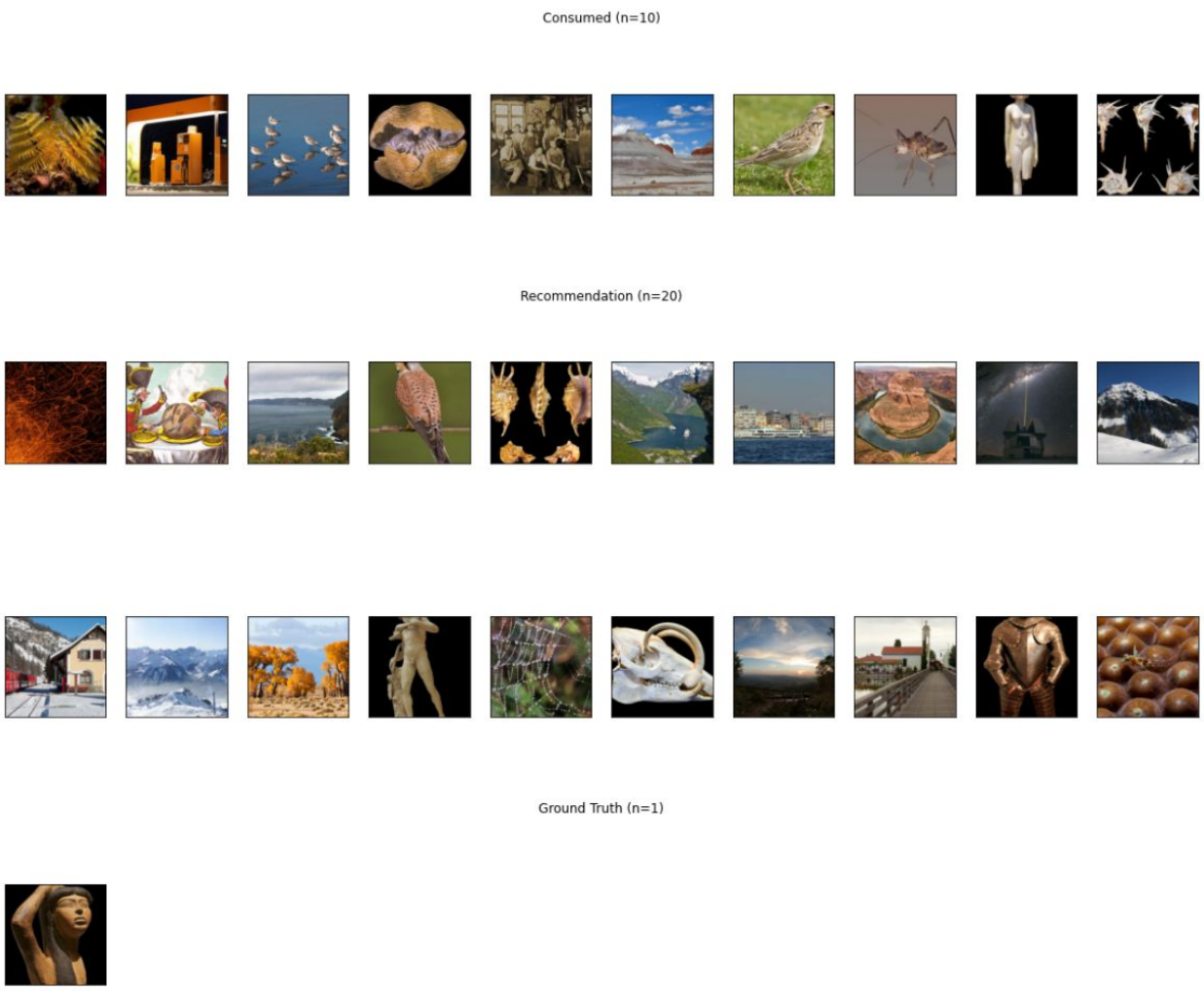
- Wikimedia interactions are related to image quality

Main Results

AUC	RR	R@20	P@20	nDCG@20	R@100	P@100	nDCG@100
0.79574	0.07086	0.03164	0.01809	0.09254	0.10099	0.01155	0.14307

Example recommendations




Consumed (n=10)
Recommendation (n=20)
Ground Truth (n=1)

DVBPR Takeaways

- Wikimedia dataset is different from Amazon and Tradesy
  - Image quality is primary concern
  - Content < Collaborative Filtering

- This might explain why latent non-visual factors are needed

- DVBPR approach is simple yet effective for visual recommendation in challenging domains

- Newer CNN architectures might be interesting to explore
  - EfficientNet
  - Lambda Networks

Session Conclusions

- BPR is a powerful method to train visual recommender systems

- VBPR leverages the power of highly descriptive features that a pre-trained CNN can generate

- DVBPR takes this one step further by *learning* the weights of the CNN module

References (VBPR)

[1] He, R., & McAuley, J. (2016, February). VBPR: visual bayesian personalized ranking from implicit feedback. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 30, No. 1).

[2] Messina, P., Dominguez, V., Parra, D., Trattner, C., & Soto, A. (2019). Content-based artwork recommendation: integrating painting metadata with neural and manually-engineered visual features. User Modeling and User-Adapted Interaction, 29(2), 251-290.

[3] Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2012). BPR: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:1205.2618.

References (DVBPR)

[1] Kang, W., Fang, C., Wang, Z., & McAuley, J. (2017). Visually-Aware Fashion Recommendation and Design with Generative Image Models. *2017 IEEE International Conference on Data Mining (ICDM)*, 207-216.

[2] Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. *Communications of the ACM, 60*, 84 - 90.

[3] Chatfield, K., Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Return of the Devil in the Details: Delving Deep into Convolutional Nets. *ArXiv, abs/1405.3531*.

HAI
—VIS

Session 3: Visual BPR methods

Questions? Shoot me an email at **pcerdam@uc.cl**

Denis Parra[1], Antonio Ossa-Guerra[1], Manuel Cartagena[1], **Patricio Cerda-Mardini**[2], Felipe del Río[1], Isidora Palma[1], Diego Saez-Trumper[3], and Miriam Redi[3]

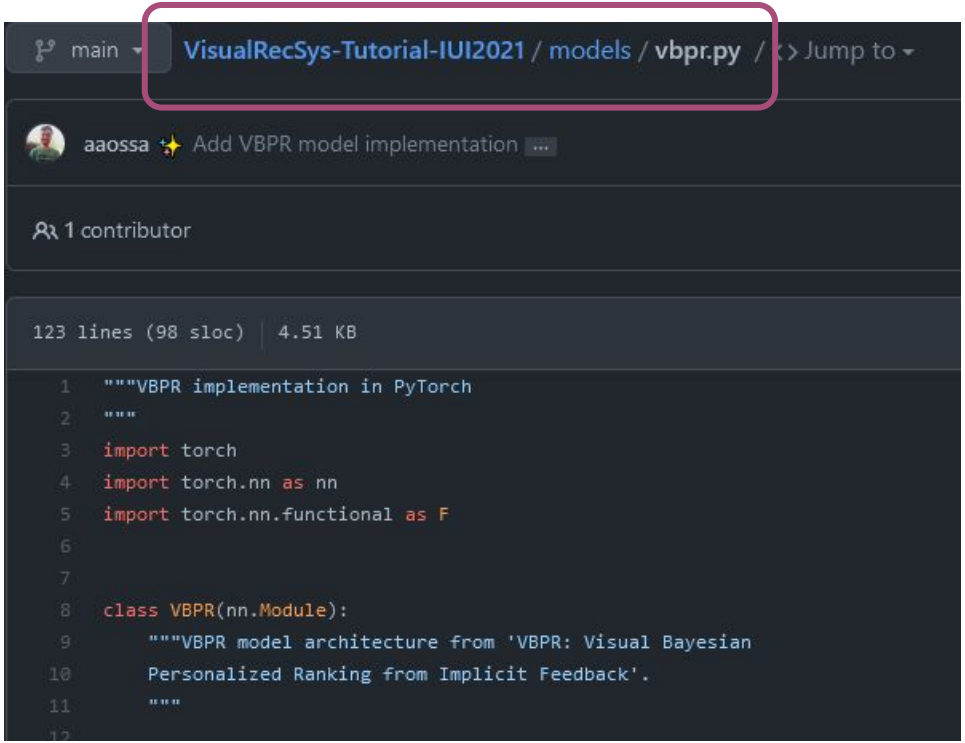
1. Pontificia Universidad Católica de Chile
2. mindsdb
3. Wikimedia Foundation

21st IEEE International Conference on Data Mining

ICDM 2021
IEEE International Conference on Data Mining
7 – 10 DECEMBER 2021
AUCKLAND NEW ZEALAND

Appendix - Implementation details

Implementation details



Implementation details (Bias terms)

```python
class VBPR(nn.Module):
    """VBPR model architecture from 'VBPR: Visual Bayesian
    Personalized Ranking from Implicit Feedback'.
    """

    def __init__(self, n_users, n_items, features, dim_gamma, dim_thet
        super().__init__()

        # Image features
        self.features = nn.Embedding.from_pretrained(features, freeze=Tru

        # Latent factors (gamma)
        self.gamma_users = nn.Embedding(n_users, dim_gamma)
        self.gamma_items = nn.Embedding(n_items, dim_gamma)

        # Visual factors (theta)
        self.theta_users = nn.Embedding(n_users, dim_theta)
        self.embedding = nn.Embedding(features.size(1), dim_theta)

        # Biases (beta)
        # self.beta_users = nn.Embedding(n_users, 1)
        self.beta_items = nn.Embedding(n_items, 1)
        self.visual_bias = nn.Embedding(features.size(1), 1)

        # Random weight initialization
        self.reset_parameters()
```

$$\beta_u + \beta_i$$

```python
def forward(self, ui, pi, ni):
    # User
    ui_latent_factors = self.gamma_users(ui)  # Latent factors of user u
    ui_visual_factors = self.theta_users(ui)  # Visual factors of user u
    # Items
    pi_bias = self.beta_items(pi)  # Pos. item bias
    ni_bias = self.beta_items(ni)  # Neg. item bias
    pi_latent_factors = self.gamma_items(pi)  # Pos. item visual factors
    ni_latent_factors = self.gamma_items(ni)  # Neg. item visual factors
    pi_features = self.features(pi)  # Pos. item visual features
    ni_features = self.features(ni)  # Neg. item visual features

    # Precompute differences
    diff_features = pi_features - ni_features
    diff_latent_factors = pi_latent_factors - ni_latent_factors

    # x_uij
    x_uij = (
        pi_bias - ni_bias
        + (ui_latent_factors * diff_latent_factors).sum(dim=1).unsqueeze(-1)
        + (ui_visual_factors * diff_features.mm(self.embedding.weight)).sum(dim
        + diff_features.mm(self.visual_bias.weight)
    )

    return x_uij.unsqueeze(-1)
```

Implementation details (Latent factors)

```python
class VBPR(nn.Module):
    """VBPR model architecture from 'VBPR: Visual Bayesian
    Personalized Ranking from Implicit Feedback'.
    """

    def __init__(self, n_users, n_items, features, dim_gamma, dim_theta):
        super().__init__()

        # Image features
        self.features = nn.Embedding.from_pretrained(features, freeze=Tru

        # Latent factors (gamma)
        self.gamma_users = nn.Embedding(n_users, dim_gamma)
        self.gamma_items = nn.Embedding(n_items, dim_gamma)

        # Visual factors (theta)
        self.theta_users = nn.Embedding(n_users, dim_theta)
        self.embedding = nn.Embedding(features.size(1), dim_theta)

        # Biases (beta)
        # self.beta_users = nn.Embedding(n_users, 1)
        self.beta_items = nn.Embedding(n_items, 1)
        self.visual_bias = nn.Embedding(features.size(1), 1)

        # Random weight initialization
        self.reset_parameters()
```

$$\gamma_u^T \gamma_i$$

```python
def forward(self, ui, pi, ni):
    # User
    ui_latent_factors = self.gamma_users(ui)  # Latent factors of user u
    ui_visual_factors = self.theta_users(ui)  # Visual factors of user u
    # Items
    pi_bias = self.beta_items(pi)  # Pos. item bias
    ni_bias = self.beta_items(ni)  # Neg. item bias
    pi_latent_factors = self.gamma_items(pi)  # Pos. item visual factors
    ni_latent_factors = self.gamma_items(ni)  # Neg. item visual factors
    pi_features = self.features(pi)  # Pos. item visual features
    ni_features = self.features(ni)  # Neg. item visual features

    # Precompute differences
    diff_features = pi_features - ni_features
    diff_latent_factors = pi_latent_factors - ni_latent_factors

    # x_uij
    x_uij = (
        pi_bias - ni_bias
        + (ui_latent_factors * diff_latent_factors).sum(dim=1).unsqueeze(-1)
        + (ui_visual_factors * diff_features.mm(self.embedding.weight)).sum(dim
        + diff_features.mm(self.visual_bias.weight)
    )

    return x_uij.unsqueeze(-1)
```

Implementation details (Visual factors)

```python
class VBPR(nn.Module):
    """VBPR model architecture from 'VBPR: Visual Bayesian
    Personalized Ranking from Implicit Feedback'.
    """

    def __init__(self, n_users, n_items, features, dim_gamma, dim_the
        super().__init__()

        # Image features
        self.features = nn.Embedding.from_pretrained(features, freeze=Tru

        # Latent factors (gamma)
        self.gamma_users = nn.Embedding(n_users, dim_gamma)
        self.gamma_items = nn.Embedding(n_items, dim_gamma)

        # Visual factors (theta)
        self.theta_users = nn.Embedding(n_users, dim_theta)
        self.embedding = nn.Embedding(features.size(1), dim_theta)

        # Biases (beta)
        # self.beta_users = nn.Embedding(n_users, 1)
        self.beta_items = nn.Embedding(n_items, 1)
        self.visual_bias = nn.Embedding(features.size(1), 1)

        # Random weight initialization
        self.reset_parameters()
```

$$\theta_u^T(\mathbf{E}f_i)$$

```python
def forward(self, ui, pi, ni):
    # User
    ui_latent_factors = self.gamma_users(ui)  # Latent factors of user u
    ui_visual_factors = self.theta_users(ui)  # Visual factors of user u
    # Items
    pi_bias = self.beta_items(pi)  # Pos. item bias
    ni_bias = self.beta_items(ni)  # Neg. item bias
    pi_latent_factors = self.gamma_items(pi)  # Pos. item visual factors
    ni_latent_factors = self.gamma_items(ni)  # Neg. item visual factors
    pi_features = self.features(pi)  # Pos. item visual features
    ni_features = self.features(ni)  # Neg. item visual features

    # Precompute differences
    diff_features = pi_features - ni_features
    diff_latent_factors = pi_latent_factors - ni_latent_factors

    # x_uij
    x_uij = (
        pi_bias - ni_bias
        + (ui_latent_factors * diff_latent_factors).sum(dim=1).unsqueeze(-1)
        + (ui_visual_factors * diff_features.mm(self.embedding.weight)).sum(dim
        + diff_features.mm(self.visual_bias.weight)
    )

    return x_uij.unsqueeze(-1)
```

Implementation details (Users' visual bias)

```python
class VBPR(nn.Module):
    """VBPR model architecture from 'VBPR: Visual Bayesian
    Personalized Ranking from Implicit Feedback'.
    """

    def __init__(self, n_users, n_items, features, dim_gamma, dim_theta):
        super().__init__()

        # Image features
        self.features = nn.Embedding.from_pretrained(features, freeze=Tru

        # Latent factors (gamma)
        self.gamma_users = nn.Embedding(n_users, dim_gamma)
        self.gamma_items = nn.Embedding(n_items, dim_gamma)

        # Visual factors (theta)
        self.theta_users = nn.Embedding(n_users, dim_theta)
        self.embedding = nn.Embedding(features.size(1), dim_theta)

        # Biases (beta)
        # self.beta_users = nn.Embedding(n_users, 1)
        self.beta_items = nn.Embedding(n_items, 1)
        self.visual_bias = nn.Embedding(features.size(1), 1)

        # Random weight initialization
        self.reset_parameters()
```

$$\beta'^{T} f_i$$

```python
def forward(self, ui, pi, ni):
    # User
    ui_latent_factors = self.gamma_users(ui)  # Latent factors of user u
    ui_visual_factors = self.theta_users(ui)  # Visual factors of user u
    # Items
    pi_bias = self.beta_items(pi)  # Pos. item bias
    ni_bias = self.beta_items(ni)  # Neg. item bias
    pi_latent_factors = self.gamma_items(pi)  # Pos. item visual factors
    ni_latent_factors = self.gamma_items(ni)  # Neg. item visual factors
    pi_features = self.features(pi)  # Pos. item visual features
    ni_features = self.features(ni)  # Neg. item visual features

    # Precompute differences
    diff_features = pi_features - ni_features
    diff_latent_factors = pi_latent_factors - ni_latent_factors

    # x_uij
    x_uij = (
        pi_bias - ni_bias
        + (ui_latent_factors * diff_latent_factors).sum(dim=1).unsqueeze(-1)
        + (ui_visual_factors * diff_features.mm(self.embedding.weight)).sum(dim
        + diff_features.mm(self.visual_bias.weight)
    )

    return x_uij.unsqueeze(-1)
```

Implementation details (Inference)

At this time, we don't need the first 2 terms, because they're constants shared across the recommendation list

$$\widehat{x}_{u,i} = \cancel{\alpha} + \cancel{\beta_u} + \beta_i + \gamma_u^T \gamma_i + \theta_u^T (\mathbf{E} f_i) + \beta'^T f_i.$$

```python
def recommend_all(self, user, cache=None, grad_enabled=False):
    with torch.set_grad_enabled(grad_enabled):
        # User
        u_latent_factors = self.gamma_users(user)  # Latent factors of user u
        u_visual_factors = self.theta_users(user)  # Visual factors of user u

        # Items
        i_bias = self.beta_items.weight  # Items bias
        i_latent_factors = self.gamma_items.weight  # Items visual factors
        i_features = self.features.weight  # Items visual features
        if cache is not None:
            visual_rating_space, opinion_visual_appearance = cache
        else:
            visual_rating_space = i_features.mm(self.embedding.weight)
            opinion_visual_appearance = i_features.mm(self.visual_bias.weight)

        # x_ui
        x_ui = (
            i_bias
            + (u_latent_factors * i_latent_factors).sum(dim=1).unsqueeze(-1)
            + (u_visual_factors * visual_rating_space).sum(dim=1).unsqueeze(-1)
            + opinion_visual_appearance
        )

        return x_ui
```

Implementation details - Model Class

```python
class DVBPR(nn.Module):
    def __init__(self, n_users, n_items, K=2048, use_cnnf=False):
        super().__init__()
        self.cache = None

        # CNN for learned image features
        if use_cnnf:
            self.cnn = CNNF(hidden_dim=K)  # CNN-F is a smaller CNN
        else:
            alexnet = models.alexnet(pretrained=False)
            final_len = alexnet.classifier[-1].weight.shape[1]
            alexnet.classifier[-1] = nn.Linear(final_len, K)
            self.cnn = alexnet

        # Visual latent preference (theta)
        self.theta_users = nn.Embedding(n_users, K)

        # Latent factors (gamma)
        self.gamma_users = nn.Embedding(n_users, 100)
        self.gamma_items = nn.Embedding(n_items, 100)

        # Random weight initialization
        self.reset_parameters()
```

Implementation details - Forward Pass

```
def forward(self, ui, pimg, nimg, pi, ni):
    # User
    ui_visual_factors = self.theta_users(ui)  # Visual factors of user u
    ui_latent_factors = self.gamma_users(ui)  # Latent factors of user u

    # Items
    pi_features = self.cnn(pimg)  # Pos. item visual features
    ni_features = self.cnn(nimg)  # Neg. item visual features

    pi_latent_factors = self.gamma_items(pi)  # Pos. item visual factors
    ni_latent_factors = self.gamma_items(ni)  # Neg. item visual factors

    x_ui = (ui_visual_factors * pi_features).sum(1) + (pi_latent_factors * ui_latent_factors).sum(1)
    x_uj = (ui_visual_factors * ni_features).sum(1) + (ni_latent_factors * ui_latent_factors).sum(1)

    return x_ui, x_uj
```

Implementation details - Optimization

```
# Forward pass
with torch.set_grad_enabled(phase == "train"):
    pos, neg = self.model(profile, pimg, nimg, pi, ni)
    output = pos-neg
    loss = self.criterion(output, target)
    loss += (1.0 * torch.norm(self.model.theta_users.weight))

    # Backward pass
    if phase == "train":
        loss.backward()
        self.optimizer.step()
```

Implementation details - Recommendation

```python
def recommend_all(self, user, cache, grad_enabled=False):
    with torch.set_grad_enabled(grad_enabled):
        # User
        u_visual_factors = self.theta_users(user)  # Visual factors of user u
        ui_latent_factors = self.gamma_users(user)

        # Items
        i_latent_factors = self.gamma_items.weight  # Items visual factors

        x_ui = ((i_latent_factors * ui_latent_factors).sum(dim=1).squeeze() + \
                (u_visual_factors * cache).sum(dim=2).squeeze())

        return x_ui
```