VisRec Tutorial
Session 2: Pipeline + VisRank + VBPR

Denis Parra, **Antonio Ossa-Guerra**, Manuel Cartagena, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces iui2021

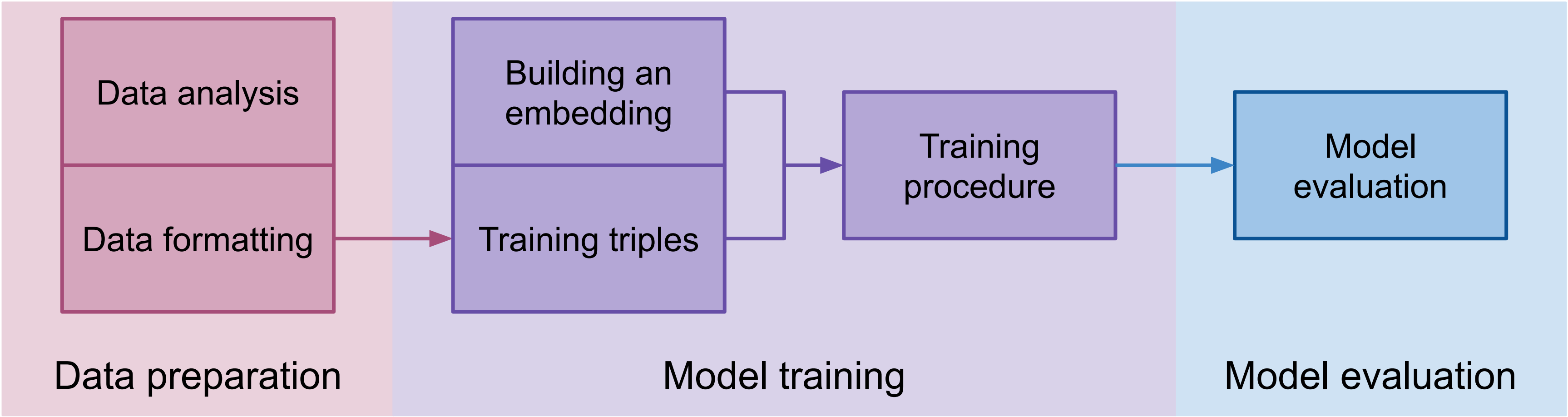
VisRec Tutorial
Session 2: **Pipeline** + VisRank + VBPR

Denis Parra, **Antonio Ossa-Guerra**, Manuel Cartagena, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

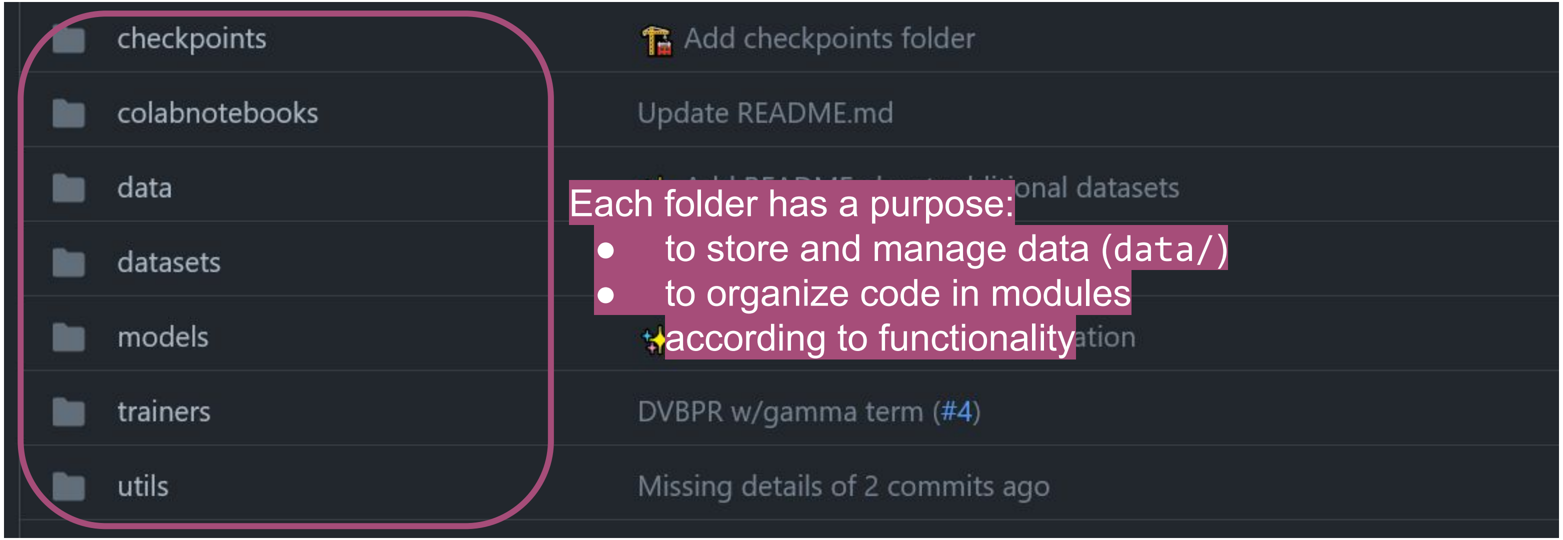
26th ACM Conference on Intelligent User Interfaces IUI 2021

Table of Contents

Organization of our pipeline



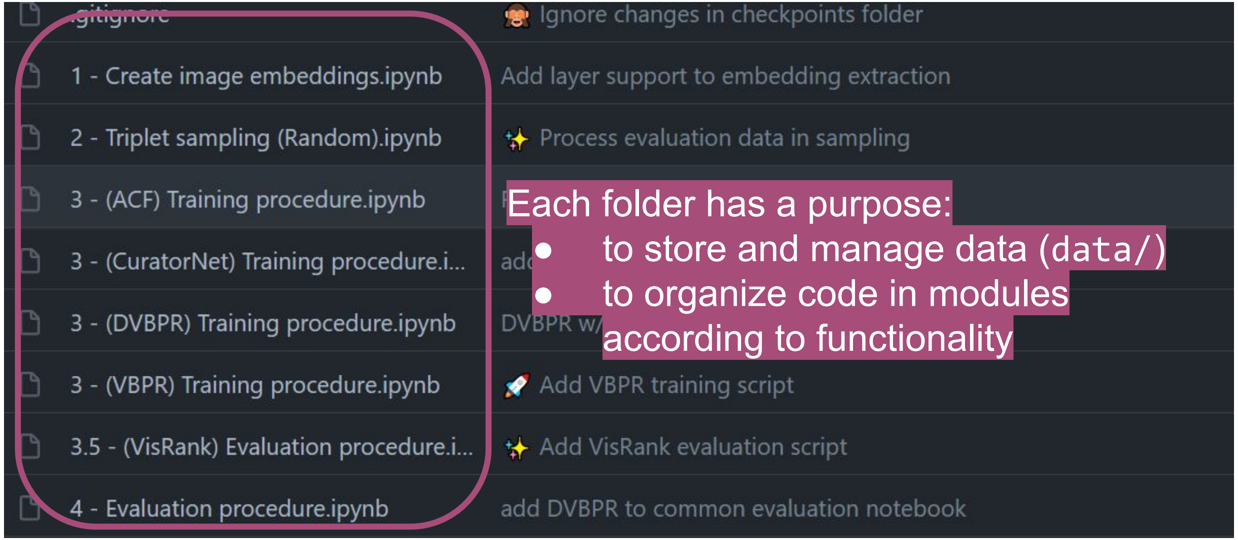
Our repository



Each folder has a purpose:

- to store and manage data (`data/`)
- to organize code in modules according to functionality

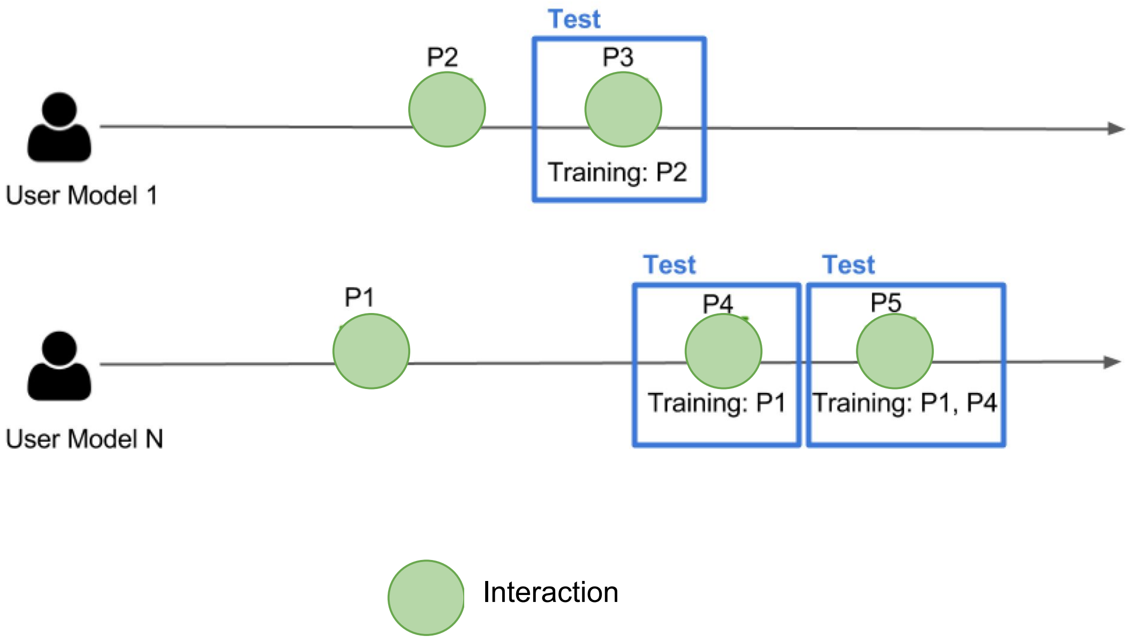
Our repository



Each folder has a purpose:

- to store and manage data (`data/`)
- to organize code in modules according to functionality

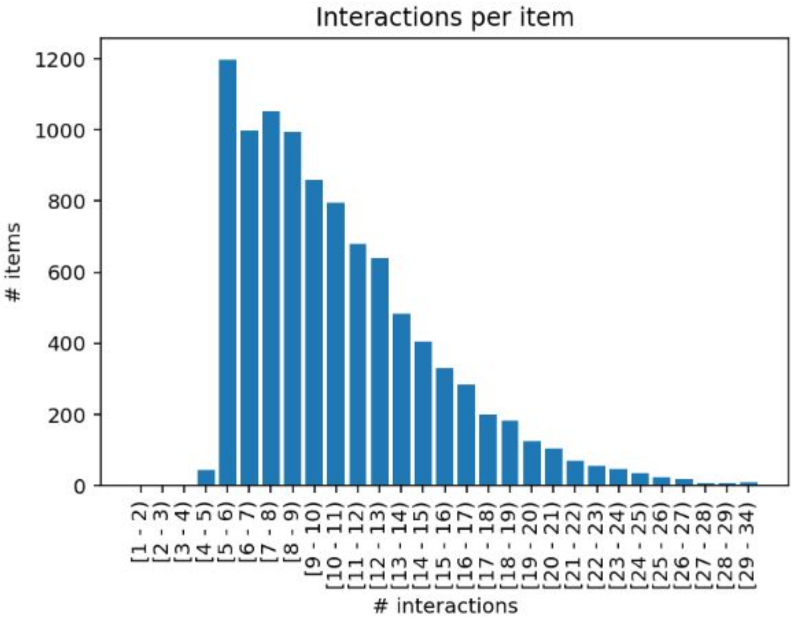
Data preparation: Data analysis



Data preparation: Data analysis

	user	image_id	timestamp	evaluation
0	1	200502005	1108503300	False
1	1	200504028	1113243060	False
2	1	20		
3	1	20		
4	1	20		

	user	image_id	timestamp	evaluation
0	1	200602085	1140370560	True
1	6	200510005	1128099960	True
2	11	200604035	1143603900	True
3	12	200805003	1208850300	True
4	13	201011221	1290851640	True



Data preparation: Data formatting

Key properties of format:

- Specific column names
- Interactions sorted by timestamp
- Column to identify evaluation rows

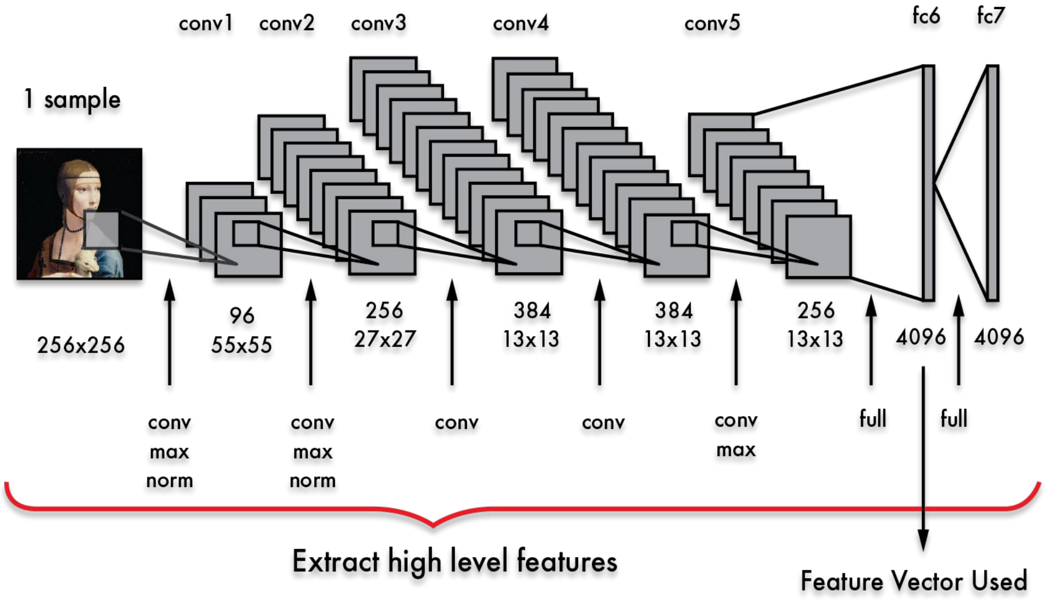
This structure is assumed further into the pipeline, so we must be consistent

	user_id	item_id	timestamp	evaluation
0	30	200501002	1105490700	False
1	12	200501002	1105521180	False
2	31	200501002	1105568700	False
3	6	200501002	1105646820	False
4	14	200501002	1105738260	False
...	...	...	...	...
96986	2738	201904242	1556319720	False
96987	2738	201904241	1556319780	True
96988	7298	201904241	1556338260	False
96989	7298	201904242	1556338380	True
96990	5578	201904242	1556347920	True

Model training: Building an embedding

- Each image in the dataset is mapped to a latent feature vector
- We store the embeddings in a *.npy file to load it in our models

```
[
    ("item_id1", <vector>),
    ("item_id2", <vector>),
    ...
]
```



Model training: User rating vs BPR

Pointwise approach:

- Looks a **single item** at a time while training, trying to predict how relevant is it fr the current query
- Requires to know how relevant the item really is: **user rating**

$$r_{ui}$$

Pairwise approach:

- Looks a **pair of items** at a time, trying to learn what's the optimal ordering of said pair
- Just needs **implicit feedback** to infere the ordering

$$x_{uij} = x_{ui} - x_{uj}$$

Model training: Triples for training

The output of this stage contains:

- **pi** and **ni**: positive and negative items (index)
- **ui**: user identifier (index)
- **profile**: index of already consumed items

We generate 5 millions triples for training and 500k for validation

	profile	pi	ni	ui
0	220 234 231 232	232	6890	0
1	23 28 29	29	4242	0
2	238 239 242 276	276	7961	0
3	234 231 232 233	233	4969	0
4	236 237 238 239	239	5866	0
...	...	...	...	...
5000081	9455 9451	9451	7814	1078
5000082	9512 9524 9525	9525	7372	1078
5000083	9455	9455	3328	1078
5000084	9511 9485 9490	9490	7872	1078
5000085	9490 9509 9526	9526	8933	1078

Model training: Data for evaluation

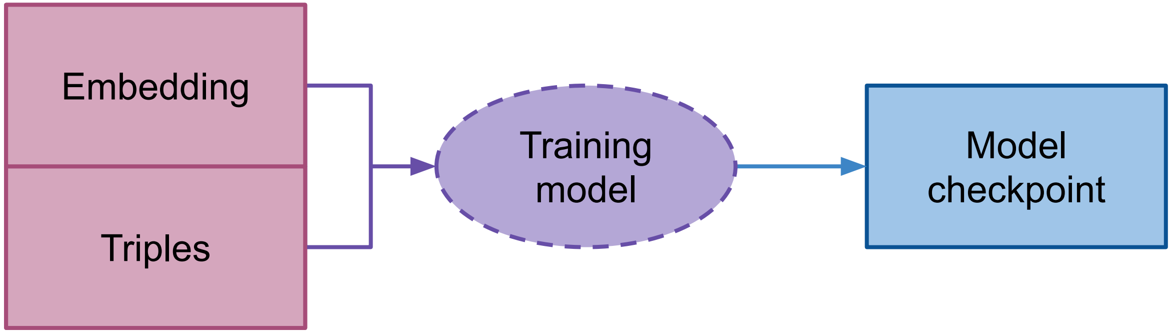
The evaluation data contains:

- **profile** and **predict**: already known interactions and ground truth (index)
- **user_id**: user identifier (index)
- **timestamp**: Time of ground truth interactions

We evaluate using the last item for each user in the dataset

	profile	predict	user_id	timestamp
0	0 29 28 27	26	14	1113423840
1	8 40 35 41	42	15	1114095000
2	25 0 30 38	43	34	1114273560
3	0 33 45 39	43	13	1114783500
4	0 48 70 69	71	30	1116466140
...	...	...	...	...
1073	9534 9528	9530	910	1556298360
1074	9486 9494	9534	984	1556305080
1075	9528 9534	9533	677	1556319780
1076	9528 9533	9534	1065	1556338380
1077	9528 9522	9534	853	1556347920

Model training: Training procedure



Input Model training Output

Differences in training and inference

Training

$$(user, item_i, item_j)$$

$$x_{uij} = x_{ui} - x_{uj}$$

Inference

$$(user, item)$$

$$x_{ui}$$

Model evaluation

In this stage we only load a trained model checkpoint

Then, for every user:

1. Predict each item score
2. Sort items by score
3. Calculate metrics

```
evaluation_df["profile"] = evaluation_df["profile"].map(tuple)
grouped_evals = evaluation_df.groupby(["profile", "user_id"]).agg({"predict":
for i, row in tqdm(enumerate(evaluation_df.itertuples()), total=len(evaluation_
    # Load data into tensors
    profile = torch.tensor(row.profile).to(device, non_blocking=True).unsqueeze
    user_id = torch.tensor([int(row.user_id)]).to(device, non_blocking=True)
    predict = torch.tensor(row.predict).to(device, non_blocking=True)
    # Prediction
    if MODEL == "ACF":
        acf_profile = profile + 1 # In ACF items   ed starting at 1
        scores = model.recommend_all(user_id, ac   squeeze()
    elif MODEL == 'DVBPR':
        scores = model.recommend_all(user_id, img   he=cache)
    elif MODE_PROFILE == "profile":
        scores = model.recommend_all(profile, cache=cache)
    elif MODE_PROFILE == "user":
        scores = model.recommend_all(user_id, cache=cache).squeeze()

    # Ranking
    pos_of_evals = (torch.argsort(scores, descend   ..., None] == predic
    if not PREDICT_ALL:
        pos_of_profi = (torch.argsort(scores, des   rue)[..., None] == pr
        # Relevant dimensions
        _a, _b = pos_of_evals.size(0), pos_of_profi.size(0)
        # Calculate shift for each eval item
        shift = (pos_of_profi.expand(_a, _b) < pos_of_evals.reshape(_a, 1).expa
        # Apply shift
        pos_of_evals -= shift.squeeze(0)
    # Store metrics
    AUC[i] = auc_exact(pos_of_evals, N_ITEMS)
    RR[i] = reciprocal_rank(pos_of_evals)
    R20[i] = recall(pos_of_evals, 20)
    P20[i] = precision(pos_of_evals, 20)
```

Tips and recommendations

In case that you want to use our pipeline:

- You’ll need a GPU (Google Colaboratory helps a lot!)
- Make sure to define an explicit criteria to choose evaluation rows
- Be consistent with the data format (ideally, it should not be different)
- Start with simple models to build your baseline
- If you add your own model implementation, follow the current structure
- Please read the README files in our repository

All of this recommendations will be available in the repository

VisRec Tutorial
Session 2: Pipeline + **VisRank** + VBPR

Denis Parra, **Antonio Ossa-Guerra**, Manuel Cartagena, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces iui2021

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

VisRank
(but main focus on DVBPR)

Context

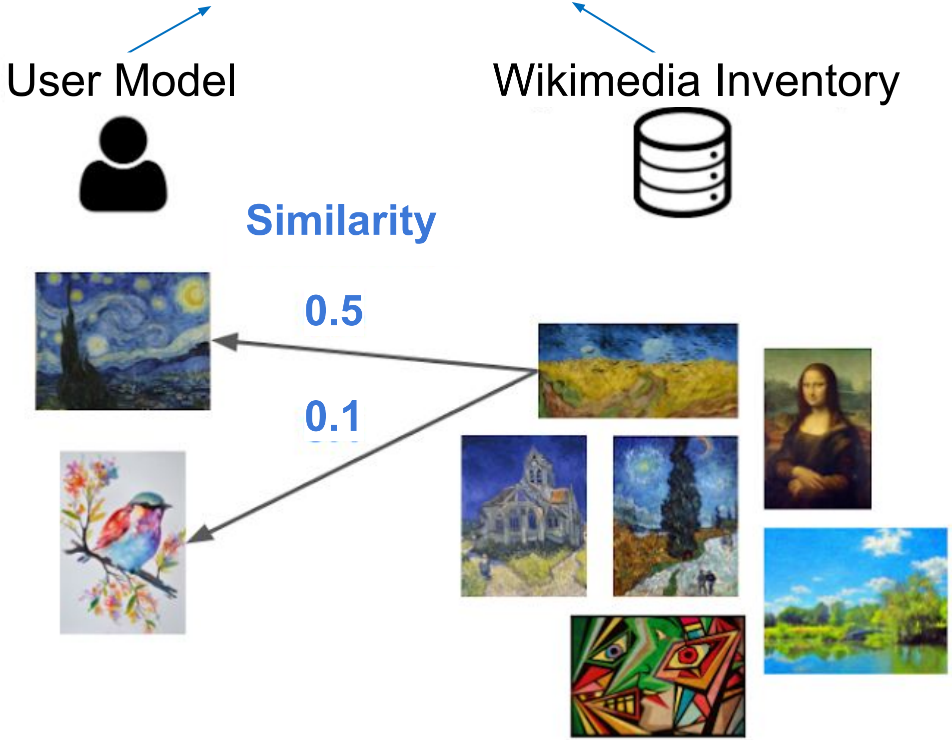
- To define a simple and performant baseline model based on similarity

- “Nearest neighbor” style recommendation

- Images are ranked according to their average distance to user items

Content-based recommendation: VisRank (baseline)

$$s(u,i) = score(\ ContentBasedProfile(u),\ Content(i)\ )$$



Mathematical formulation

Calculates similarity between the items as the cosine similarity between the vector representations

$$sim(V_i, V_j) = cos(V_i, V_j) = \frac{V_i \cdot V_j}{\|V_i\|\|V_j\|}$$

$$score(u,i)_X = \begin{cases} \max\limits_{j \epsilon P_u}\{sim(V_i^X, V_j^X)\} & (maximum) \\ \frac{\sum\limits_{j \in P_u} sim(V_i^X, V_j^X)}{|P_u|} & (average) \\ \frac{\sum\limits_{r=1}^{\min\{K,|P_u|\}} \max\limits_{j \epsilon P_u}{}^{(r)}\{sim(V_i^X, V_j^X)\}}{\min\{K,|P_u|\}} & (average\ top\ K) \end{cases}$$

Metrics for evaluation

AUC

$$AUC = \frac{1}{|\mathcal{U}|} \sum_{u} \frac{1}{|E(u)|} \sum_{(i,j) \in E(u)} \delta(\widehat{x}_{u,i} > \widehat{x}_{u,j})$$

Mean Reciprocal Rank

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i}$$

Metrics for evaluation

Precision@K

$$P@k = \frac{1}{|U_r|}\sum_{u\in U_r}\left(\frac{1}{|T_u|}\sum_{t\in T_u} p@k(t)\right) \qquad p@k(t) = \frac{|r_t^k \cap R_t|}{k}$$

Recall@K

$$R@k = \frac{1}{|U_r|}\sum_{u\in U_r}\left(\frac{1}{|T_u|}\sum_{t\in T_u} r@k(t)\right) \qquad r@k(t) = \frac{|r_t^k \cap R_t|}{|R_t|}$$

nDCG@K

$$nD@k = \frac{1}{|U_r|}\sum_{u\in U_r}\left(\frac{1}{|T_u|}\sum_{t\in T_u} \frac{DCG@k(t)}{iDCG@k(t)}\right) \qquad DCG@k(t) = \sum_{z=1}^{k}\frac{2^{B_t(i_{t,z})}-1}{log_2(1+z)}$$

Main Results

AUC	RR	R@20	P@20	nDCG@20	R@100	P@100	nDCG@100
0.60491	0.02788	0.04267	0.00213	0.03020	0.06215	0.00062	0.03376

- Reasonable result in AUC, better than random (0.5)

- Good MRR and nDCG values (we'll see other models later)

VisRec Tutorial
Session 2: Pipeline + VisRank + **VBPR**

Denis Parra, **Antonio Ossa-Guerra**, Manuel Cartagena, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces

VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback

Ruining He
UC San Diego
r4he@ucsd.edu

Julian McAuley
UC San Diego
jmcauley@ucsd.edu

a.k.a. Visual BPR

Context

- Previous work did not consider the *visual appearance* of the items
- Modeling derived from BPR, that’s 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

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

α	Global offset
$\beta_u + \beta_i$	Bias terms
$\gamma_u^T \gamma_i$	Latent factors (compatibility between user and item)

Model: Preference predictor

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

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

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

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

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

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

Implementation details

main VisualRecSys-Tutorial-IUI2021 / models / vbpr.py / Jump to

aaossa ✨ Add VBPR model implementation ...

1 contributor

123 lines (98 sloc) | 4.51 KB

```python
"""VBPR implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


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

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)

```
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()
```

$$\beta'^T f_i$$

```
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
```

Main Results

AUC	RR	R@20	P@20	nDCG@20	R@100	P@100	nDCG@100
0.77846	0.02169	0.05565	0.00278	0.02684	0.13821	0.00138	0.04105

- Big improvement in AUC (VisRank: 0.60491)

- Ranking metrics @100 also improved significantly

References

[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.

VisRec Tutorial
Session 2: Pipeline + VisRank + **VBPR**

Denis Parra, **Antonio Ossa-Guerra**, Manuel Cartagena, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces

IUI2021

Hands-On

VisRec: A Hands-on Tutorial on Deep Learning for Visual Recommender Systems

Denis Parra, **Antonio Ossa-Guerra**, Manuel Cartagena, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces iUI 2021

VBPR model diagram

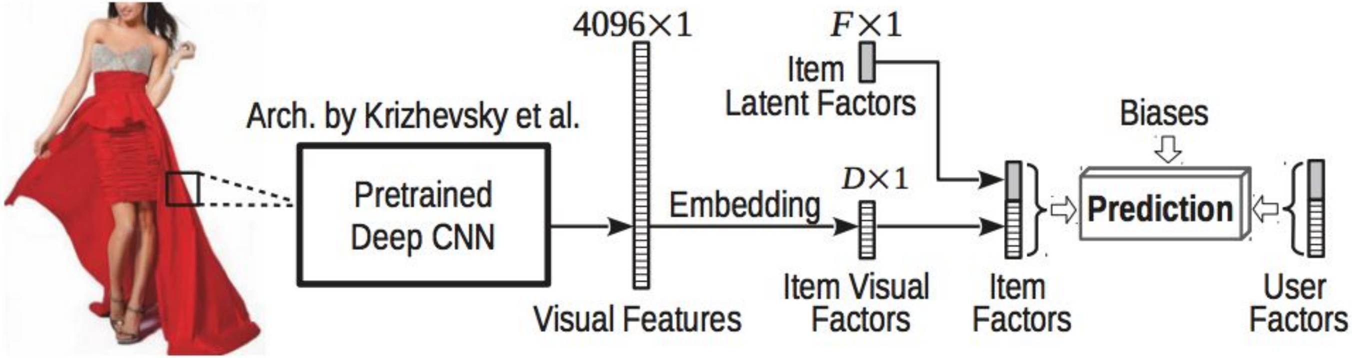


Figure 1: Diagram of our preference predictor. Rating dimensions consist of visual factors and latent (non-visual) factors. Inner products between users and item factors model the compatibility between users and items.

VBPR loss function

VBPR: Visual Bayesian Personalized Ranking (R. He & McAuley, 2016)

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

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

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

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