OFFLINE RETRIEVAL EVALUATION WITHOUT EVALUATION METRICS

Recherche d'information

22 mai 2023

Ben Kabongo & Sofia Borchani

M1 DAC - Sorbonne Université

Machine Learning & Deep Learning for Information Access

RECALL-PAIRED PREFERENCE

Introduction et motivations

Problématique

- Comparaison entre systèmes : métriques d'évaluation
- Métriques traditionnelles (AP, NDCG, MRR, etc.) = scalaire
- **Problème de l'efficacité des étiquettes** : perte des informations sur la manière dont deux classements diffèrent
- **Problème de faible robustesse aux comportements utilisateurs** : hypothèses fortes sur la distribution des comportements

Nouvelle métrique : RPP **Recall-Paired Preference**

- Résolution des problèmes des métriques traditionnelles
- Modélisation des préférences **sous-population d'utilisateurs**
- Agrégation : pondération égale à tous les niveaux de rappel

Métriques. Pseudo-populations

Métriques

- **Rang** : f_i : rang du ième document pertinent du sytème π. $f_i < f_i' \rightarrow \pi > \pi'$
- **Préférences** : $I(\pi, \pi') = \mathbb{E}_k[sgn(P@k(\pi) - P@k(\pi'))]$
- Expérimentation en ligne. k : seuil de différence des clics.

Pseudo-Populations d'utilisateurs

- Différents types d'utilisateur en fonction du niveau de rappel
- U_i : pseudo-population intéressé par exactement i éléments pertinents $p(i) = P(u \in U_i)$
- $\mathbb{E}_i[\mu_i(\pi)] = \sum_i^m p(i)\mu_i(\pi)$
- On peut étendre aux seuils de pertinence et aux pertinences par thématiques

Recall-Paired Preference

SCIENCES SORBONNE UNIVERSITÉ

Formule et variantes

- $RPP(\pi, \pi') = \mathbb{E}_i[\text{sgn}(f'_i - f_i)] = \sum_{i=1}^{m} p(i) \times \text{sgn}(f'_i - f_i)$
- **RPP gardué** :
  $RPP(\pi, \pi') = \sum_{\lambda \in \Lambda} \sum_{i=1}^{m} p(i, \lambda) \times \text{sgn}(f'_{i,\lambda} - f_{i,\lambda})$
- **RPP catégorielle** :
  $\text{ST-RPP}(\pi, \pi') = \sum_{t \in T} \sum_{i=1}^{m} p(i, t) \times \text{sgn}(f'_{i,t} - f_{i,t})$

- $RPP(\pi, \pi') \in [-1, +1]$
- $RPP(\pi, \pi') = -RPP(\pi', \pi)$
- $RPP(\pi, \pi') > 0 \rightarrow \pi > \pi'$

Protocole d'expérimentation

Protocole d'expérimentation

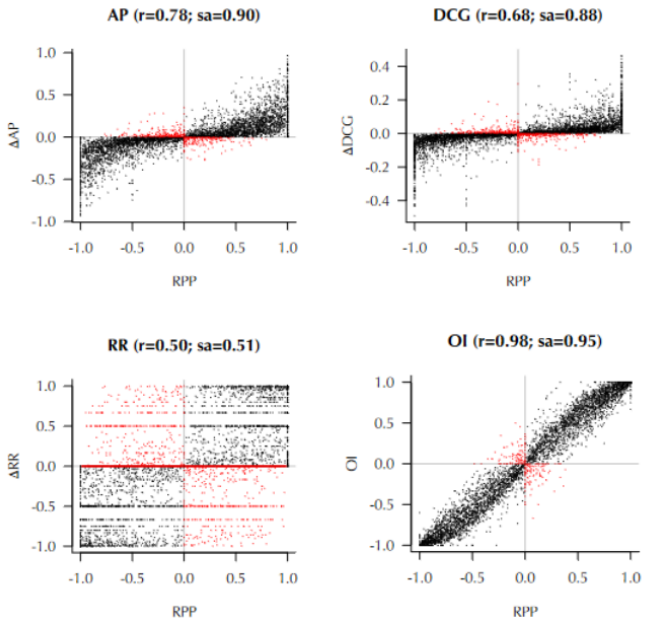
Expérimentations

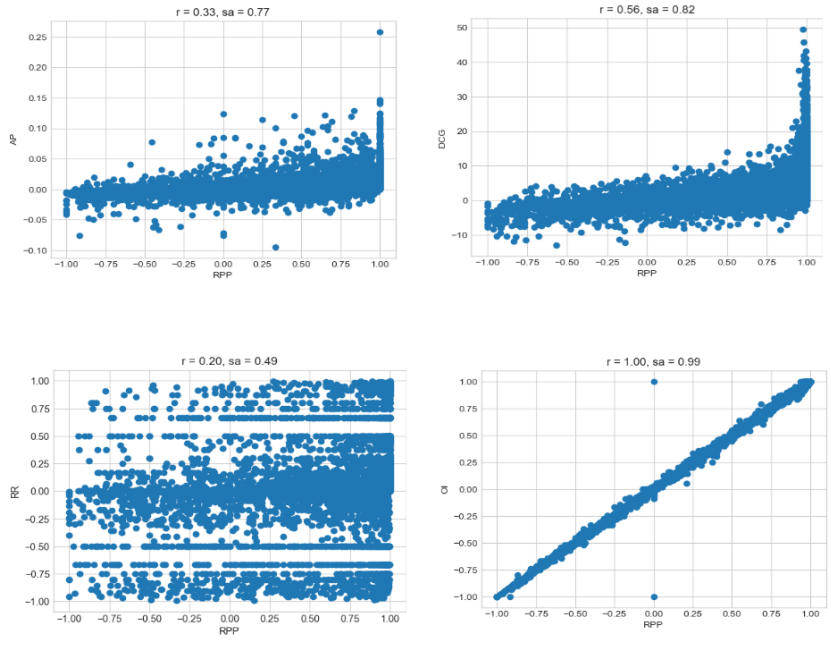
- Comment RPP est-elle corrélé aux autres métriques ?
- RPP est-elle robuste aux données incompltètes ?
- RPP est-elle efficace pour différencier les classements des systèmes entre elles ?

Metriques désagrégées

Métrique	$\Delta\mu_i$
AP	$\mathrm{i}\left(\frac{1}{f_i}-\frac{1}{f_i'}\right)$
NDCG	$\frac{1}{\log_2(f_i+1)}-\frac{1}{\log_2(f_i'+1)}$
RR	$\left(\frac{1}{f_i}-\frac{1}{f_i'}\right),(i=1)$
RPP	$\mathrm{sgn}(\mathrm{f'}_i-f_i)$

Corrélation de RPP avec les métriques existantes

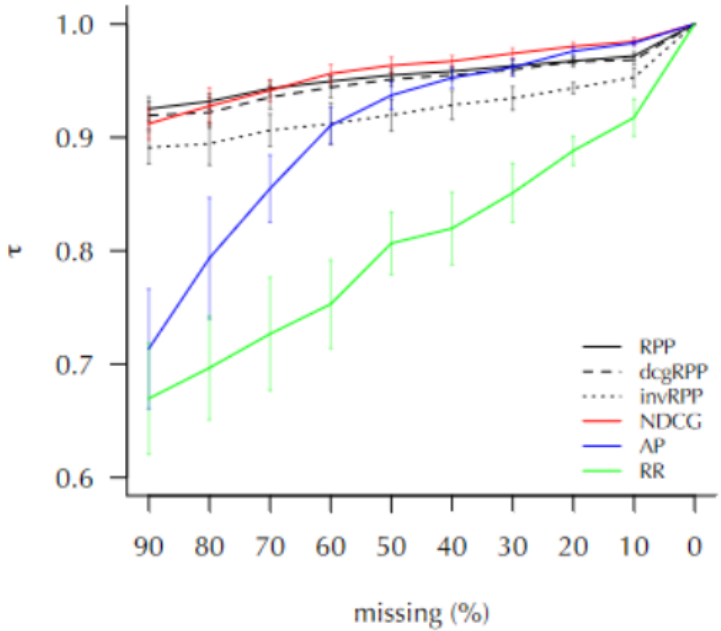


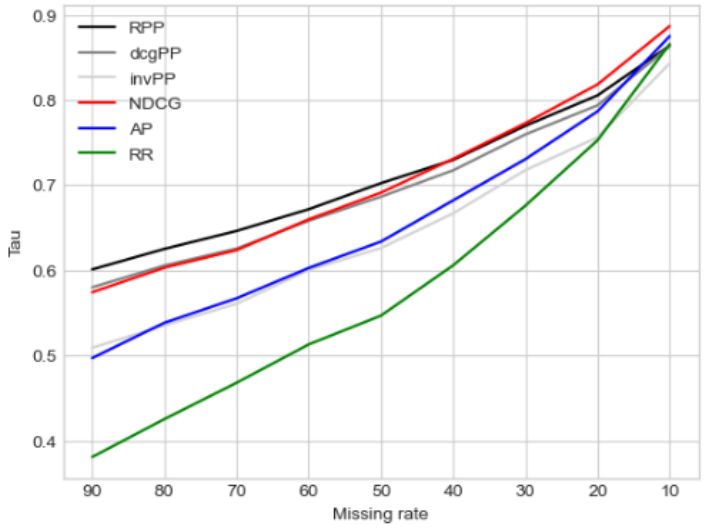


Métriques vs. RPP pour la même paire d'exécutions.

Corrélation de Pearson : mesure de relation linéaire entre variables.
Accord de signe : concordance d'évaluation entre évaluateurs.

Robustesse aux données incomplètes





Comparaison des classements avec labels manquants et labels complets.

Corrélation de Kendall τ : évalue la similarité ou l'accord entre deux classements.

Pouvoir discriminatif

Méthode de Sakai : évaluation du pouvoir discriminant.

p-value $< 0{,}05$ dans le test de Student : faible probabilité de différences aléatoires entre les échantillons observés.
$\rightarrow$ Suggère une réelle différence statistiquement significative entre des groupes comparés.

Métrique	Expérimental	Référence
RPP	94.28	96.19
DCGPP	93.80	95.71
INVPP	92.38	96.19
NDCG	90.47	94.29
AP	83.80	91.43
RR	84.76	85.71

Code

Recall-Paired Preference

SCIENCES SORBONNE UNIVERSITÉ

```python
def RPP(ranking1, ranking2, p=None):
    if p is None:
        m = min(len(ranking1), len(ranking2))
        p = np.ones(m)/m
    return (p * np.sign(ranking2[:len(p)] - ranking1[:len(p)])).sum()
```

Figure 1 – RPP - code

Comparaion entre métriques

SCIENCES SORBONNE UNIVERSITÉ

```
def compare(predictions1, predictions2, queries):
    rpp_all, dcgpp_all, invpp_all, ndcg_all, dcg_all, ap_all, rr_all, asl_all, oi_all = [], [], [

    for query_id in queries:
        rankings1 = np.array(predictions1[query_id][predictions1[query_id]["bin"] == 1]['rank'])
        rankings2 = np.array(predictions2[query_id][predictions2[query_id]["bin"] == 1]['rank'])
        if len(rankings1) == 0:
            rankings1 = np.append(rankings1, len(predictions1[query_id]))
        if len(rankings2) == 0:
            rankings2 = np.append(rankings2, len(predictions2[query_id]))

        relevant1 = np.array(predictions1[query_id]['rel'])
        relevant2 = np.array(predictions2[query_id]['rel'])

        rpp_all.append( RPP(rankings1, rankings2) )
        dcgpp_all.append( DCGPP(rankings1, rankings2) )
        invpp_all.append( INVPP(rankings1, rankings2) )
        ndcg_all.append( delta_NDCG(relevant1, relevant2) )
        dcg_all.append( delta_DCG(relevant1, relevant2) )
        ap_all.append( delta_AP(rankings1, rankings2) )
        rr_all.append( delta_RR(rankings1, rankings2) )
        asl_all.append(delta_ASL(rankings1, rankings2) )
        oi_all.append(OI(rankings1, rankings2) )

    return  rpp_all, dcgpp_all, invpp_all, ndcg_all, dcg_all ,ap_all, rr_all, asl_all, oi_all
```

Figure 2 – Comparaison - code

Robustesse des métriques

SCIENCES SORBONNE UNIVERSITÉ

```
def robustess(predictions1,
              predictions2,
              queries,
              docs,
              rpp_all,
              dcgpp_all,
              invpp_all,
              ndcg_all,
              ap_all,
              rr_all,
              doc_col_name='doc_id'):

    rpp_tau, dcgpp_tau, invpp_tau, ndcg_tau, ap_tau, rr_tau = [], [], [], [], [], []

    for missing_rate in range(90, 0, -10):
        rpp, dcgpp, invpp, ndcg, ap, rr = [], [], [], [], [], []
        for query_id in queries:
            preds1_df = predictions1[query_id]
            preds2_df = predictions2[query_id]
            n_docs = missing_rate * len(docs) // 100
            missing_docs = np.random.choice(docs, n_docs)
            missing_preds1_df = preds1_df.drop(preds1_df[preds1_df[doc_col_name].isin(missing_docs)].index)
            missing_preds1_df["rank"] = missing_preds1_df["score"].rank(ascending=False).apply(int)
            rankings1 = np.array(missing_preds1_df[missing_preds1_df["bin"] == 1]['rank'])
            relevant1 = np.array(missing_preds1_df['rel'])
            missing_preds2_df = preds2_df.drop(preds2_df[preds2_df[doc_col_name].isin(missing_docs)].index)
            missing_preds2_df["rank"] = missing_preds2_df["score"].rank(ascending=False).apply(int)
            rankings2 = np.array(missing_preds2_df[missing_preds2_df["bin"] == 1]['rank'])
            relevant2 = np.array(missing_preds2_df['rel'])
            if len(rankings1) == 0: rankings1 = np.append(rankings1, len(missing_preds1_df))
            if len(rankings2) == 0: rankings2 = np.append(rankings2, len(missing_preds2_df))
            rpp.append( RPP(rankings1, rankings2) )
            dcgpp.append( DCGPP(rankings1, rankings2) )
            invpp.append( INVPP(rankings1, rankings2) )
            ndcg.append( delta_NDCG(relevant1, relevant2) )
            ap.append( delta_AP(rankings1, rankings2) )
            rr.append( delta_RR(rankings1, rankings2) )

        rpp_tau.append( kendalltau(rpp, rpp_all)[0] )
        dcgpp_tau.append( kendalltau(dcgpp, dcgpp_all)[0] )
        invpp_tau.append( kendalltau(invpp, invpp_all)[0] )
        ndcg_tau.append( kendalltau(ndcg, ndcg_all)[0] )
        ap_tau.append( kendalltau(ap, ap_all)[0] )
        rr_tau.append( kendalltau(rr, rr_all)[0] )

    return rpp_tau, dcgpp_tau, invpp_tau, ndcg_tau, ap_tau, rr_tau
```

Pouvoir discriminant

```python
def power(all_runs, queries, test_fn=t_test_with_bonferroni_correction):
    N = len(all_runs)
    rpp_pvalue, dcgpp_pvalue, invpp_pvalue, ndcg_pvalue, ap_pvalue, rr_pvalue = [], [], [], [], [], []
    for (i, j) in list(itertools.combinations(range(N), 2)):
        rpp_all, dcgpp_all, invpp_all, ndcg_all, _ ,ap_all, rr_all, _, _ = compare(
            all_runs[i], all_runs[j], queries
        )
        rpp_pvalue.append( test_fn(rpp_all) )
        dcgpp_pvalue.append( test_fn(dcgpp_all) )
        invpp_pvalue.append( test_fn(invpp_all) )
        ndcg_pvalue.append( test_fn(ndcg_all) )
        ap_pvalue.append( test_fn(ap_all) )
        rr_pvalue.append( test_fn(rr_all) )

    pvalue = .05
    rpp_ratio = np.where(np.array(rpp_pvalue) < pvalue, 1, 0).mean() * 100
    dcgpp_ratio = np.where(np.array(dcgpp_pvalue) < pvalue, 1, 0).mean() * 100
    invpp_ratio = np.where(np.array(invpp_pvalue) < pvalue, 1, 0).mean() * 100
    ndcg_ratio = np.where(np.array(ndcg_pvalue) < pvalue, 1, 0).mean() * 100
    ap_ratio = np.where(np.array(ap_pvalue) < pvalue, 1, 0).mean() * 100
    rr_ratio = np.where(np.array(rr_pvalue) < pvalue, 1, 0).mean() * 100

    return rpp_ratio, dcgpp_ratio, invpp_ratio, ndcg_ratio, ap_ratio, rr_ratio
```

Figure 4 – Pouvoir discriminant - code