**Projet Interpromo 2021**

## Groupe 8 : Innovation

# *Learning To Rank*


Cheffe : Flora Estermann
Cheffe adjointe : Katia Belaid
Chef qualité : Tanguy Dirat

Samba Seye
Célya Marcélo
Théo Saccareau
Damien Sonneville
Mélina Audiger
Jordi Mora Fernandez

# Sommaire

# 1. Introduction

Lorsque les utilisateurs consultent un moteur de recherche, ils s'attendent à ce que les résultats retournés soient pertinents et correspondent à leurs attentes.
En effet, d'un point de vue client, passer des heures à naviguer entre plusieurs pages de recherche dans l'espoir de trouver l'information ou le produit qui vous intéresse sur un site peut rapidement s'avérer frustrant, et résulte la plupart du temps à l'abandon de la recherche ou pire, à la consultation d'un site concurrent.

En considérant cette problématique, comment fournir aux utilisateurs un meilleur service en optimisant ces résultats de recherche selon les spécificités de leur requête ou bien en les personnalisant selon les caractéristiques de leur profil ?

C'est ici qu'intervient le learning-to-rank, cette méthode de Machine Learning permet de retourner non seulement des résultats pertinents, mais surtout classés par ordre de pertinence.
Intégrée aux moteurs de recherche, cette méthode présente des avantages très importants : augmentation significative du nombre de sessions de recherche par utilisateur, davantage de résultats consultés par recherche, ainsi qu'une réduction du nombre de recherches par session. Ces facteurs soulignent le fait que les utilisateurs sont en mesure de trouver ce qu'ils cherchent plus rapidement. Résultats pertinents, utilisateur content, donc tout le monde est gagnant !

Dans un premier temps, nous présenterons le principe du learning-to-rank et expliquerons les choix ayant été faits pour l'appliquer à la recherche d'informations.
Nous détaillerons ensuite les résultats obtenus avant de finir sur de potentielles pistes à explorer ou des caractéristiques qu'il pourrait être intéressant d'ajouter avec davantage de données.

# 2. État de l'art

## 2.1. Présentation de la méthode

### Le learning-to-rank, qu'est-ce que c'est ?

C'est un ensemble de techniques utilisant le Machine Learning supervisé pour résoudre des problèmes de classement.

Contrairement au ML traditionnel utilisé pour résoudre des problèmes de classification ou régression en considérant une seule instance à la fois, le LTR résout un problème de classement sur une liste d'items. Le but du LTR est de retourner un ordre optimal à partir des items de cette liste.

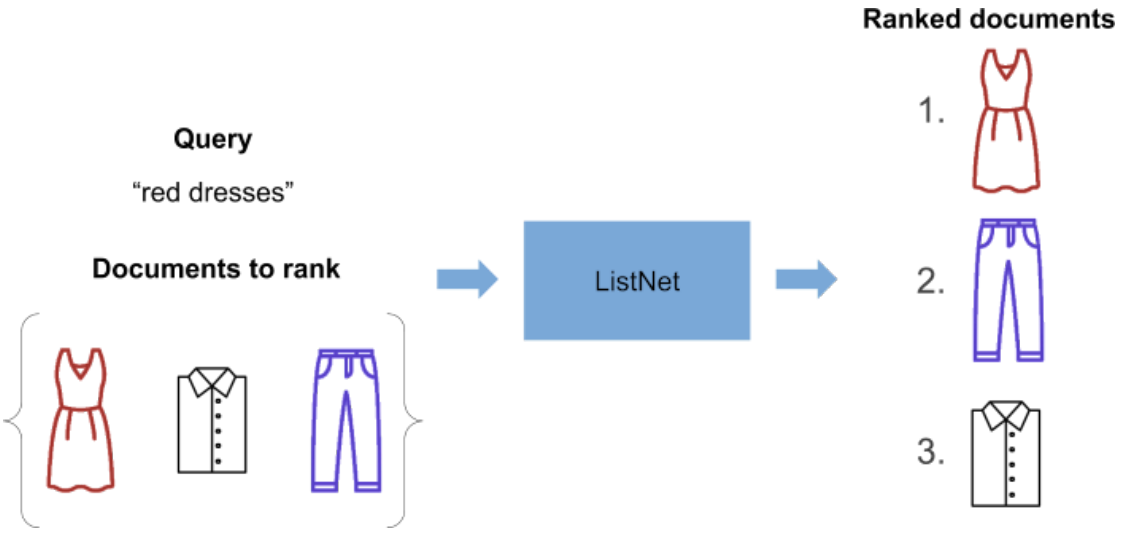


*Figure 1. Illustration de la méthode Learning-to-rank*

Cet exemple illustre très bien la différence avec le ML classique. Un simple modèle de classification retournerait un score de pertinence pour chacun des documents, et en sélectionnant un threshold adapté, permettrait de donner à l'utilisateur un ensemble de documents satisfaisant la requête.
Cependant, l'innovation du LTR appliqué à la recherche d'informations réside dans le fait qu'en prenant cette information sur les documents pertinents dans le contexte de la requête en entrée, et en y appliquant un modèle adapté, on peut ainsi obtenir en sortie une liste ordonnée de manière optimale.
Le LTR ne s'intéresse donc pas au score propre à chacun des documents, mais plutôt à l'ordre relatif qui les unit.

### Quels types de données ?

Les données d'entraînement pour un modèle de LTR sont généralement des listes de résultats de requête associés à un score de pertinence respectif à chacune des requêtes.

Les data scientists génère ce jeu d'entraînement en examinant les résultats et en décidant d'inclure ou exclure chacun des résultats au dataset.
Les données de cet ensemble constituent ainsi les labels de référence utilisés par le modèle pour faire ses prédictions. On appelle ce jeu de données annotées la « vérité terrain » et nous évaluons  nos prédictions en sélectionnant une mesure de distance pour évaluer l'écart entre les réponses du système et la vérité terrain.

### Plusieurs types d'approches possibles :

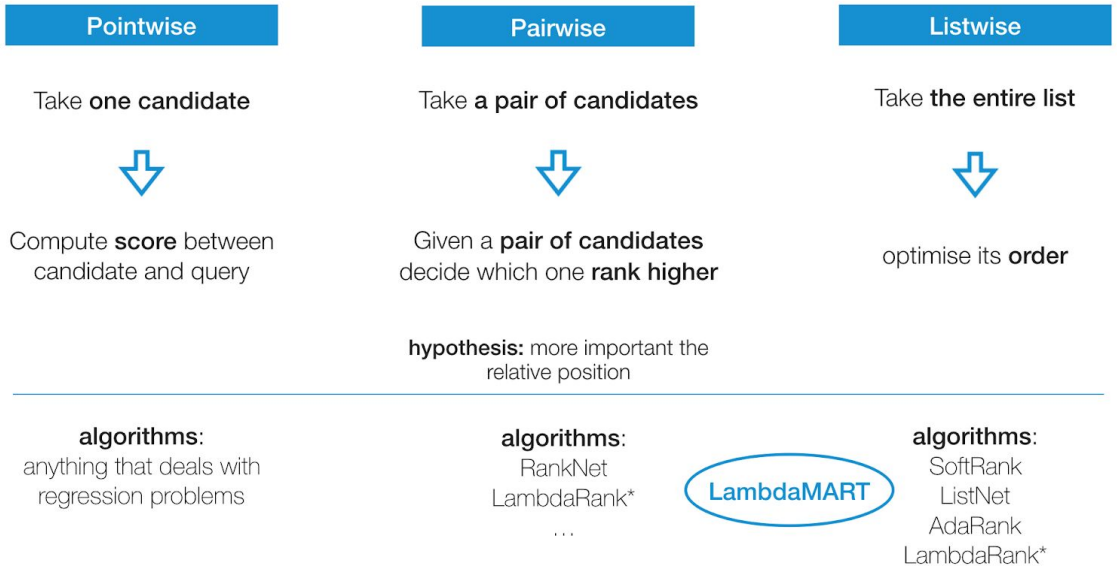


*Figure 2. Schéma des différents types d'approches possibles*

L'implémentation que nous avons choisie repose sur une approche pairwise et reproduit un algorithme de LambdaMART simplifié. Le choix du modèle sera justifié prochainement.

### Comment intégrer le retour utilisateur ?

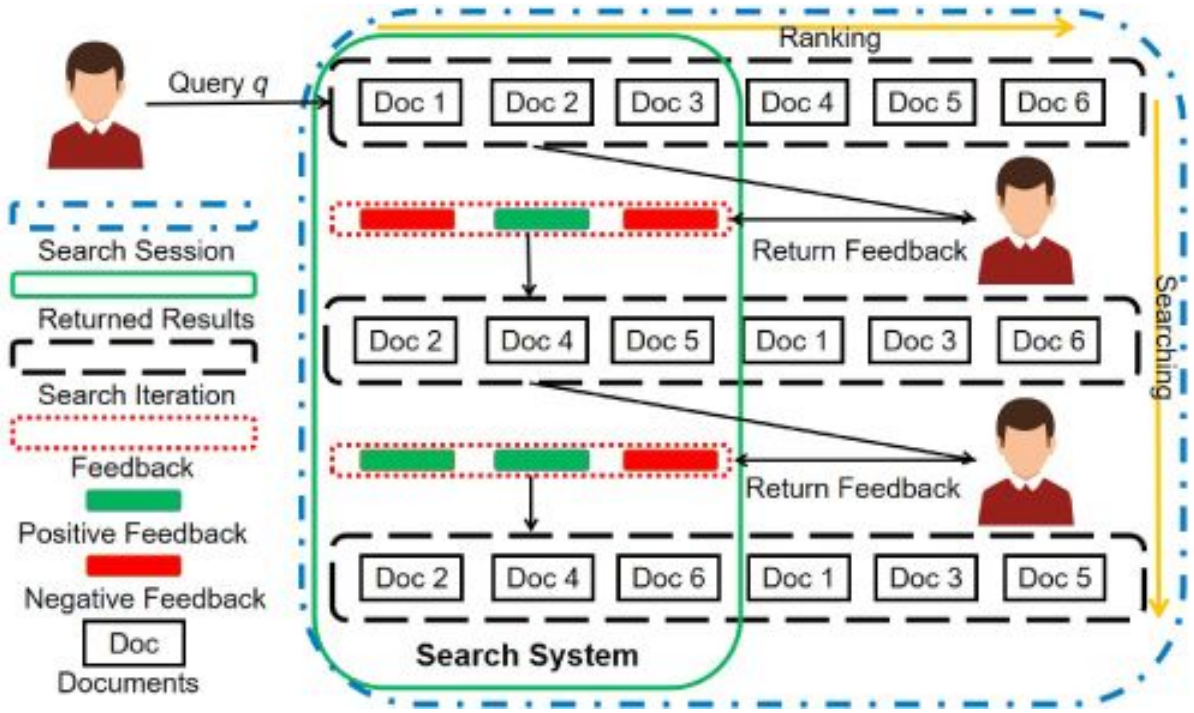


*Figure 3. Illustration du principe de retour utilisateur*

A terme, on souhaite également que le modèle soit en mesure de prendre en compte le retour des utilisateurs (Figure 3) et les caractéristiques de leur profil afin d'être en mesure de personnifier au maximum les résultats de leur recherche.

De plus, incorporer de nouvelles features au jeu de données devrait sûrement permettre d'améliorer la pertinence des résultats d'une requête également.

Nous présenterons plusieurs méthodes envisagées au cours du rapport.

## 2.2. Préparation du jeu d'entraînement

### 2.2.1. Simulation des jeux de requêtes

Nous avons simulé les requêtes à partir de deux jeux de données différents, la première simulation s'est faite sur les tags des données scrapées et la seconde s'est basée sur les mots clés des lexiques de gestion et d'innovation fournis par le client.

**Simulation à partir des tags :**

Pour cette première simulation nous avons procédé de 3 manières différentes.
Avant de commencer chacune de ces méthodes un nettoyage de données est effectué sur les tags.

Simulation naïve : Méthode très rapide où il suffit de récupérer une liste sans doublons de tous les tags. Une requête correspond à un tag.

Simulation Random : Une fois tous les tags récupérés, nous créons une liste de mots sans doublons avec laquelle nous faisons un tirage de N mots aléatoire et sans remise pour la création de nos requêtes.

Simulation avec le poids : Nous calculons le poids de chaque mot en fonction de son nombre d'apparitions dans les tags. Nous effectuons par la suite un tirage de N mots sur tous les mots en fonction de leur poids, tirage sans remise pour une requête donnée (pas de doublons dans une même requête). Une fois qu'un mot est apparu 10 fois dans les requêtes on supprime ce mot, on calcule le nouveau poids de chaque mot, on tire à nouveau N mot et on recommence ce processus jusqu'à obtenir le nombre de requête désiré (une requête = un tirage), ou bien jusqu'à avoir utilisé chaque mot 10 fois.

**Simulation à partir des mots clés :**

Une fois la liste de mots clés récupérée et nettoyée, nous effectuons un tirage aléatoire et sans remise de N mots pour la création de nos requêtes. On choisit un seuil maximum d'apparition d'un mot dans les requêtes, une fois ce seuil dépassé on supprime le mot de la liste pour les prochains tirages. On continue les tirages jusqu'à dépasser le seuil pour chaque mot.

Dans le but d'améliorer nos modèles nous avons créé une liste de requêtes à partir de deux jeux de données scrapées, simulation avec le poids sur les tags, ainsi que sur deux jeux de données de mots clés.

### 2.2.2 Simulation des tables qrels

La table Qrels nous permet d'identifier les documents pertinents associés à chaque requête. Elle permet, de plus, de connaître le degré de pertinence de chaque document identifié comme pertinent.
Pour implémenter la table Qrels, nous procédons en quatre étapes.

La première étape est l'indexation des documents présents dans nos données. Cette étape consiste à identifier le contenu de chaque document au moyen d'un vocabulaire. L'indexation permet de faciliter la recherche de contenu dans une collection.

Par la suite, on détermine les requêtes non pertinentes dans notre analyse en appliquant une fonction de ranking. La fonction permet d'estimer la pertinence des documents par rapport à une requête de recherche donnée. Les fonctions de ranking (classement) sont des fonctions utilisées par les moteurs de recherche. On choisit de supprimer les requêtes qui n'ont pas de documents retournés pertinents. On peut aussi établir un seuil correspondant à un nombre minimal de documents pertinents pour chaque requête, afin de ne récupérer que les requêtes les plus représentatives pour notre jeu de données.

Nous appliquons ensuite une fonction de ranking (BM25) pour construire une première partie de la table Qrels.

La dernière étape dans l'implémentation de la table est l'ajout des labels, qui nous indiquent le degré de pertinence des documents par rapport aux requêtes. A ce stade, nous avons procédé par simulation aléatoire de ces labels (0: non pertinent, 1: peu pertinent, 2: pertinent, 3: très pertinent).
Néanmoins, l'idée est que ces labels soient définis en se basant sur l'avis des utilisateurs. En regardant les résultats de sa requête, l'utilisateur a la liberté de noter chaque article retourné comme étant pertinent ou non, ou de le mettre en favori. Le nombre de clics sur les articles peut aussi être pris en compte afin d'évaluer les degrés de pertinence.

# 2.3. Architecture d'un pipeline LTR et choix des modèles optimaux

Lorsque les lexiques de gestion et innovation fournis par le client ont été intégrés et que les requêtes ont été simulées, il devient possible de formuler un voire plusieurs LTR pipelines.
Conceptuellement, le LTR se résume en 3 phases :

1. Pour chaque requête, identifier un set de documents correspondants grâce à des modèles de récupération.
2. Élaborer un pipeline LTR et le compléter en associant de manière complexe les modèles de récupération permettant ainsi d'optimiser les scores de pertinence obtenus ( association de modèles adaptés ).
3. Définir un ou plusieurs modèles d'apprentissage automatique - aux paramètres préalablement définis - et les appliquer au pipeline LTR afin de re-calculer les scores de pertinence sur le set de documents identifiés au 1. et donc obtenir de meilleurs scores.

### Évaluation des performances des modèles de recherche d'information et de fouille de textes disponibles :

Dans le domaine de la recherche d'information (RI), il existe un grand nombre de modèles de recherche d'information pour la détection de similarités entre les documents textuels d'une collection donnée.

PyTerrier, la librairie que nous utilisons, met à la disposition des utilisateurs des pipelines pré-définis de recherche et récupération d'information, et rend également possible leur évaluation et amélioration. Nous les avons comparés sur les données du client + les données issues de nos simulations et nous avons exploité les résultats de ces évaluations afin de récupérer les modèles les plus pertinents pour la construction du pipeline LTR.
Nous avons récupéré les 10 meilleurs modèles évalués selon les métriques MAP et NDCG.

**Remarque importante :** L'ensemble des résultats affichés pour les 3 types de simulation de requête ont été produits sur notre version initiale de la table Qrels (générée par simulation également.)
C'est la raison pour laquelle les résultats sont si élevés et nettement moins représentatifs de ce qu'on pourrait obtenir en réalité avec des documents pertinents annotés par un humain.
Par la suite, ce biais dans la simulation a été corrigé mais les résultats correspondants n'apparaîtront qu'à la fin dans la partie *Résultats*.

| name | map | ndcg |
|---|---|---|
| BM25 | 0.995926 | 0.997450 |
| TF_IDF | 0.991885 | 0.996568 |
| InL2 | 0.990938 | 0.996350 |
| DLH | 0.985519 | 0.995075 |
| DFReeKLIM | 0.980638 | 0.993931 |
| PL2 | 0.973615 | 0.992058 |
| DLH13 | 0.969720 | 0.991246 |
| DPH | 0.965123 | 0.989868 |
| DFRee | 0.936096 | 0.982018 |
| In_expB2 | 0.947054 | 0.980769 |

*Figure 4. Scores pour les requêtes issues de la simulation naïve à partir des tags*

Les modèles les plus performants pour ce jeu de requête sont les suivants : BM25, TF-IDF et InL2.

| name | map | ndcg |
|---|---|---|
| BM25 | 0.983420 | 0.988746 |
| DPH | 0.977866 | 0.987485 |
| TF_IDF | 0.975721 | 0.986737 |
| PL2 | 0.971109 | 0.986395 |
| DLH13 | 0.961043 | 0.984018 |
| DLH | 0.959969 | 0.983548 |
| In_expC2 | 0.956316 | 0.983438 |
| InL2 | 0.955129 | 0.982454 |
| IFB2 | 0.956029 | 0.982354 |
| In_expB2 | 0.952424 | 0.981399 |

*Figure 5. Scores pour les requêtes issues de la simulation random à partir des tags*

Cette fois-ci, les modèles les plus performants sont BM25, DPH et TF-IDF.

| name | map | ndcg |
|---|---|---|
| BM25 | 0.999541 | 0.999905 |
| TF_IDF | 0.997229 | 0.999421 |
| InL2 | 0.994735 | 0.998872 |
| DLH | 0.982132 | 0.995899 |
| PL2 | 0.980654 | 0.995596 |
| DFReeKLIM | 0.976987 | 0.994623 |
| DLH13 | 0.968978 | 0.992750 |
| DPH | 0.963643 | 0.991125 |
| IFB2 | 0.965775 | 0.990928 |
| In_expB2 | 0.964705 | 0.990686 |

*Figure 6. Scores pour les requêtes issues de la simulation avec le poids à partir des tags*

Pour ce dernier jeu de requête, on retient finalement les trois premiers modèles : BM25, InL2 et TF-IDF. Et ce sont d'ailleurs ces derniers qui seront combinés pour construire la pipeline LTR finale utilisée pour produire nos résultats finaux. En effet, nous avons constaté que le jeu de requêtes basé sur les tags pondérés (utilisé pour l'entraînement des modèles) était bien plus représentatif que les deux autres jeux de requêtes simulés.

# 3. Résultats

## 3.1. Démarches d'évaluation d'un système de recherche d'informations (SRI)

Après avoir sélectionné puis entraîné plusieurs modèles, la librairie Pyterrier nous permet de les comparer afin de déterminer celui qui offre les meilleurs résultats. Pour cela, nous avons choisi d'utiliser deux métriques différentes : MAP et NDCG. A noter, qu'il en existe d'autres mais elles sont plus rarement employées, nous nous sommes donc basés uniquement sur les métriques par défaut qui présentent l'avantage de plutôt bien se compléter.

### Mesures d'évaluations avec prise en compte de l'ordre :

Pour évaluer un SRI, on peut faire le choix de ne pas prendre en compte l'ordre des documents retournés. Dans ce cas, on utilise le rappel et la précision.
En revanche, si on souhaite prendre en compte ce facteur, on emploie les métriques MAP et NDCG. Ainsi, dans cette partie, la connaissance du rang auquel un document a été retourné par le système devient primordial.

Mean Average Precision (MAP) : Le calcul de la mesure agrégée pour une requête est relativement simple. Cette mesure est très utilisée car elle offre un bon compromis entre rappel en haut de liste et rappel total.

Normalized Discounted Cumulative Gain (NDCG) : La pertinence peut ne pas être binaire (Pertinent / Non Pertinent) mais graduelle. Par exemple, on peut disposer de l'échelle de pertinence suivante : Parfait (3), Très pertinent (2), Pertinent (1), Non Pertinent (0). Or les mesures classiques telles que MAP, se basent uniquement sur la binarité d'un document, contrairement au NDCG.

Pour obtenir les valeurs optimales du NDCG, on se base sur un système idéal qui renverrait donc en premier les documents les plus pertinents, puis les documents un peu moins pertinents et enfin les documents non pertinents.

→ Ces deux mesures sont donc les plus couramment utilisées. La NDCG présente l'avantage de prendre en compte la pertinence graduelle des documents. Étant donné dans nos simulations nous avons fait en sorte d'obtenir des labels de pertinences allant de 0 à 3, il est pertinent d'utiliser la NDCG en complément de la MAP (qui pour rappel ne distingue que les documents pertinents/non pertinents).

## 3.2. Scores obtenus

### Choix du pipeline LTR :

PyTerrier propose de combiner les modèles retenus de recherche d'information afin de construire un pipeline LTR adapté à la demande et aux résultats boostés.
La phase initiale (phase 0) de ce pipeline reçoit le modèle le plus performant de notre système de récupération de documents après indexation : le BM25. La première grande phase de l'élaboration du pipeline reçoit ainsi une association de modèles s'appliquant comme des "features" pour chaque document traité.

Pour cela, nous avons utilisé les opérateurs implémentés dans PyTerrier :

>> opérateur utilisé pour composer le pipeline (~then), ordonner les modèles
** opérateur représentant l'union des résultats des modèles associés

Le pipeline de départ est défini par la combinaison suivante (phase 1) :
BM25 >> ( TF_IDF ** **x** ) avec **x** un des deux modèles retenus (InL2 ou DPH).

On applique ensuite à cette pipeline un modèle d'apprentissage (phase 2) afin de prédire les scores définis par les métriques MAP et NDCG.

Nous avons ainsi pu comparer les performances du pipeline LTR que nous avions construit avec celles du modèle BM25 appliqué seul (*BM25 baseline*).

Pour **x** = InL2, on obtient les scores suivants :

| name | MAP | NDCG |
|---|---|---|
| BM25 Baseline | 0.023738 | 0.291419 |
| LTR | 0.022023 | 0.286325 |

La performance du InL2 est souvent davantage significative pour des requêtes courtes et pré-traitées. Or comme les requêtes des utilisateurs en entrée varient par leur taille et la manière dont nous les avons construites par simulation (mots clés uniquement ou phrases entières), il n'est pas étonnant que InL2 ne performe pas aussi bien que la baseline BM25.

Pour **x** = DPH, on obtient les scores suivants :

| name | MAP | NDCG |
|---|---|---|
| BM25 Baseline | 0.023738 | 0.291419 |
| LTR | 0.022247 | 0.288149 |

Ces résultats sont un peu meilleurs que ceux du InL2 mais sans que ce soit significatif, et ne dépassant pas non plus les résultats de la baseline BM25.

→ Cette incohérence vient probablement du fait que la majorité de notre travail repose sur des simulations (requêtes, tables Qrels et données d'utilisateurs) qui approximent difficilement la vérité terrain nécessaire au bon entraînement des modèles.
Par conséquent, notre enchaînement de modèles sur la pipeline ne fait que propager cet aléa issu de la simulation, et induit ainsi beaucoup de bruit que nous n'aurions pas eu avec de vraies données.

### Choix du modèle d'apprentissage automatique :

Il reste une dernière étape (phase 3) pour la construction du pipeline complexe LTR : assister l'expérience LTR avec les modèles d'apprentissage automatique.

Ici, nous avons reproduit un algorithme de LambdaMART à l'aide des modèles XGBoost et LightGBM.

Pour **x** = InL2, on obtient les scores MAP et NDCG suivant :

| name | MAP | NDCG |
|---|---|---|
| BM25 Baseline | 0.023738 | 0.291419 |
| LambdaMART(XGBoost) | 0.021916 | 0.294337 |
| LambdaMART (LightGBM) | 0.022208 | 0.286804 |

Pour **x** = DPH, on obtient les scores MAP et NDCG suivant :

| name | MAP | NDCG |
|---|---|---|
| BM25 Baseline | 0.023738 | 0.291419 |
| LambdaMART(XGBoost) | 0.021753 | 0.289957 |
| LambdaMART (LightGBM) | 0.024042 | 0.291908 |

Pour la métrique MAP, les scores sont légèrement meilleurs avec l'application de l'algorithme LightGBM. En effet, la documentation de PyTerrier stipule qu'associé à une combinaison fonctionnelle et pertinente de modèles de recherche d'information dans le pipeline LTR, LightGBM serait en général plus efficace que XGBoost. Nos résultats semblent confirmer cette intuition et nous décidons de retenir le choix d'utiliser l'union des résultats du DPH avec TF-IDF.
On observe de même que le score NDCG est légèrement amélioré avec le LightGBM, et sachant que le NDCG présente l'avantage de prendre en compte la pertinence graduelle des documents, cela s'avère plutôt intéressant.

→ Au vu des résultats obtenus, il est difficile d'affirmer qu'un modèle surpasse les autres.
Afin de valider une combinaison de modèles complète (phases 0 à 3), il faudrait reproduire l'ensemble de ces tests à partir d'un jeu d'entraînement disposant d'une vérité terrain annotée de manière concrète (et non simulée par l'absence de données).

# 4. Discussion

## 4.1. Méthode state-of-the-art sur la modification de requêtes

Afin d'améliorer nos résultats pour la partie LTR, PyTerrier propose des fonctions sur la réécriture de requête et l'expansion de requête.

### Réécriture de requête automatisée :

La réécriture de requête se réfère au changement de la formulation des requêtes afin d'améliorer l'efficacité des modèles de recherche d'information et de retourner des résultats plus pertinents. Il existe deux types de réécriture de requête :

- Q to Q : réécrire la requête en ajoutant/enlevant des termes basé sur le WordNet/Word2Vect ;

- R to Q : réécrire la requête en utilisant le set de documents associant suite à l'application des modèles de recherche d'information du LTR.

Plus le pipeline LTR est complexe, plus le risque que des soucis de fonctionnement du code apparaissent (si celui-ci est édité en amont ou que les données sources sont modifiées).
De plus, la documentation de la librairie n'explique pas comment récupérer ces requêtes modifiées. L'objectif de l'application du système de modification automatique de requête aurait été de récupérer ces nouvelles requêtes afin d'améliorer les performances du LTR. Cela n'a pas pu être fait.
Enfin, le système de réécriture étant déjà paramétré, cela ne nous permet pas de pouvoir améliorer les paramètres à partir des données du client.

### Expansion de requête :

Les approches classiques citées précédemment sont toutes basées sur l'utilisation de mot-clés. En réalité, un mot peut avoir plusieurs sens, et un sens peut être exprimé par des mots différents. Afin de traiter la correspondance entre les mots-clés et les sens, on propose généralement de faire les deux traitements suivants:

- considérer des mots (ou termes) reliés pour étendre la requête ;
- considérer une désambiguïsation des mots afin de reconnaître le sens dénoté.

## 4.2. Caractéristiques liées aux utilisateurs

L'un des axes d'amélioration majeurs de nos modèles demeure l'intégration de nouvelles caractéristiques pertinentes afin d'affiner au mieux les résultats de recherche retournés à chacun des utilisateurs.
En récoltant des informations sur les différents types d'utilisateurs parcourant l'interface et en analysant leur comportement, on serait réellement en mesure de leur proposer une expérience individuelle et adaptée à chacun d'entre eux, dans le sens où la combinaison de résultats qu'ils obtiendraient serait unique en fonction des signaux de leur profil, ainsi que de leur historique.

Par exemple, un employé du département de comptabilité n'aurait pas nécessairement le même ordre de priorité de recommandation qu'un employé travaillant dans le pôle ingénierie. Ainsi en construisant une base de requêtes et de feedbacks suffisamment importante, on pourrait s'appuyer sur toutes ces données afin de produire des clusterings de profils et axer une partie de la recommandation sur ces critères-là également.

## 4.3. Intégration à l'interface web du groupe G7

L'un des objectifs de notre projet était d'intégrer notre moteur de recherche à l'interface web du groupe G7. Initialement, cette interface disposait d'une première version de moteur de recherche très élémentaire : l'utilisateur devait saisir une requête dans la barre de recherche, et le moteur renvoyait simplement les articles dont le titre contenait la chaîne de caractères saisis.

Nous avons donc récupéré le template du fichier python initial du groupe 7 et y avons ajouté le code correspondant à l'implémentation de notre propre système de recherche d'informations.
Ce fichier largement commenté expliquait comment intégrer le module que nous avions réalisé à leur propre interface web.

Malheureusement, faute de temps sur la fin du projet et en considérant les problèmes de serveur rencontrés, le moteur de recherche n'a pas eu l'occasion d'être déployé sur le serveur lors de la phase finale et reste donc suspendu en phase de production.

De plus, l'un des obstacles à envisager lors de la mise en place du système global automatique pourrait être la fréquence de rafraîchissement des instances utiles au bon fonctionnement de notre moteur de recherche (indexation des nouveaux documents, fréquence d'entraînement des modèles, sauvegarde des données produites etc).

## Conclusion :

Dans l’ensemble, on soutient vraiment l’idée que toutes ces informations obtenues via le moteur de recherche pourraient vraiment avoir une valeur concrète parce qu’elles nous permettraient de mieux comprendre l’utilisateur, et donc potentiellement de remodeler les services qu’on lui propose à travers l’intégralité de la chaîne de production, de la phase de scrapping jusqu’à son interaction avec l’interface web.