


# Recommandation de documents pour moteur de recherche Coveo

Philippe Blais  
Philippe Blouin-Leclerc  
William Bourget  
Stéphane Caron  
Samuel Lévesque


21 décembre 2018

## Résumé

Faire ressortir les points saillants de l'article en un seul paragraphe question de titiller le lecteur.

L'ensemble du code et des documents qui ont servi à la résolution de cette problématique et à l'écriture de l'article se trouvent dans le répertoire  du projet.

## 1 Présentation du problème et état de l'art

Dans le cadre de ce projet proposé par la compagnie Coveo , nous devons utiliser un historique de requêtes faites par des utilisateurs afin de développer un modèle de recommandation de documents. L'objectif du modèle est de proposer une série de 5 documents d'intérêt en fonction de la recherche qui est faite par l'utilisateur et de certaines autres caractéristiques.

Il existe plusieurs méthodes pour bâtir des systèmes de recommandation. Parmi ces méthodes, il en existe deux qui sont fréquemment utilisées en pratique. Ces méthodes sont le filtrage collaboratif et les systèmes basés sur le contenu. La première consiste à calculer des associations entre des items (documents) ou des utilisateurs pour ainsi utiliser cette information pour faire une recommandation. Cette méthode a l'avantage d'être relativement simple à implémenter. La deuxième méthode consiste à utiliser les attributs des items ou des utilisateurs pour prédire les documents d'intérêt. Cette méthode a l'avantage d'apprendre des liens entre certains attributs pour raffiner la qualité des recommandations.

Dans ces différentes approches décrites, le modèle commence par extraire de l'information des documents cible et peut par la suite se définir une mesure

de distance entre une requête et chacun des documents pour déterminer quel serait la meilleure correspondance requête-document. Dans notre cas, nous n'avons pas accès au contenu détaillé des documents que l'on souhaite prédire, mais seulement à un jeu restreint de caractéristiques comme la source du document, son auteur et son titre.

Nous avons également le choix d'attaquer la problématique comme un problème de régression ou comme un problème de classification. Dans le premier cas, il faut attribuer un score à chacun des documents et ainsi recommander ceux dont le score est le plus élevé. Dans le deuxième cas, il faut tenter de prédire directement une classe, qui correspond à un document en particulier.

Nous avons décidé d'utiliser un système basé sur le contenu dans un contexte de classification. Étant donné que nous avons beaucoup d'utilisateurs différents (environ 600) et beaucoup de documents différents (environ 6000), la méthode basée sur le filtrage collaboratif nous apparaissait moins efficace. De plus, certains utilisateurs sont inconnus par le système, ce qui rend plus complexe la tâche d'utiliser cette information.

Le reste du document est structuré de cette façon : la section 2 décrit notre approche de manière plus détaillée, la section 3 présente notre méthodologie expérimentale, la section 4 présente les résultats expérimentaux, la section 5 fait l'analyse de ces résultats. Finalement, la section 6 présente les différents constats et leçons que nous avons tirés de ce projet.

## 2 Approche proposée

Selon le livre *Introduction to Information Retrieval* de (Schutze, Manning, & Raghavan, ), une approche standard en recherche d'information est de se servir

du contenu des documents pour créer un jeu d'attributs pour chaque document disponible. Ces attributs sont ensuite utilisés pour faire l'apprentissage d'un algorithme prédictif. Cela correspond à l'approche basée sur le contenu que nous avons choisie.

On peut également utiliser cette approche pour créer des attributs pour chaque de nos requêtes. Une fois que nous avons deux ensembles d'attributs (requêtes et documents), nous pouvons définir une mesure de similarité qui permettra d'associer une requête à un document. En ayant de bons attributs, il est possible de penser que la mesure de similarité permettra de trouver des associations vers des documents pertinents pour l'utilisateur.

Dans notre cas, nous n'avons pas accès au contenu des documents. Ainsi, au lieu de créer deux ensembles d'attributs, nous avons créé des attributs pour les requêtes seulement. Ensuite, nous allons utiliser ces attributs pour entraîner un modèle supervisé où les classes à prédire sont les différents documents possibles.

Pour bâtir ce modèle, nous pouvons séparer le travail en 2 grandes étapes : prétraitement des données et modélisation. La première étape consiste essentiellement à créer les attributs, alors que la deuxième consiste à entraîner des modèles en utilisant ces attributs.

Pour ce qui est de la création d'attributs, nous pensons qu'une grande partie de l'information prédictive réside dans le contenu de la requête directement. Pour utiliser cette information, nous allons tester plusieurs méthodes de vectorisation de texte pour transformer ces requêtes textuelles en information numérique. De plus, afin de mieux capter le contexte autour des mots dans les requêtes, nous allons utiliser les plongements de mots, décrits dans le chapitre 6.8 du livre *Speech and Language Processing* de (Jurafsky & Martin, ). Pour faciliter l'apprentissage, nous allons également normaliser nos requêtes. Plus de détails sont donnés sur ces éléments dans la prochaine section.

Pour ce qui est de la partie modélisation, on souhaite utiliser les représentations numériques de nos requêtes pour comparer différents modèles d'apprentissage automatique et optimiser leurs hyperparamètres pour augmenter le pouvoir prédictif de notre modèle.

Finalement, puisqu'on s'attaque ici à un problème de classification ayant un très grand nombre de classes, nous allons également tenter de faire un modèle basé sur de l'apprentissage non-supervisé. Pour ce faire, nous allons créer des attributs de nos documents pour ainsi regrouper certains d'eux et réduire le nombre de classes possibles. On utiliserait par la suite ces classes

aggrégées pour entraîner un modèle de classification (qui serait cette fois-ci supervisé) retournant plutôt le groupe de documents duquel on choisirait les 5 plus pertinents (fréquents).

### 3 Méthodologie expérimentale

Avant même de créer les attributs, il était nécessaire de faire la jonction entre les différents jeux de données pour entre autres indiquer au modèle quelle requête a mené à quel document. D'abord, voici les différents jeux de données (avec le nombre d'observations) à notre disposition :

| Type            | E     | V     | T    |
|-----------------|-------|-------|------|
| <i>searches</i> | 52133 | 14895 | 7448 |
| <i>clicks</i>   | 24491 | 6920  | ??   |

La colonne E correspond à l'ensemble d'entraînement, la colonne V à l'ensemble de validation et la colonne T correspond à l'ensemble test. Pour ce dernier, nous avons accès aux recherches seulement et non aux clicks. Comme on peut le voir, il y a plus de recherches que de clicks. Cela veut dire que certaines recherches ont mené vers aucun click. Une même recherche peut également avoir mené à plusieurs clicks. En bref, il existe plusieurs façons de joindre ces recherches avec ces clicks. Afin de nous simplifier la vie, nous avons retiré les recherches qui n'ont pas mené vers un click. De plus, nous avons également gardé seulement le dernier click, en se basant sur le moment du click, pour une même recherche. Notre raisonnement derrière est que le dernier click est probablement le "bon" et que les précédents ajoutent du bruit au modèle. En somme, voici le nombre d'observations que nous avons pour les ensembles d'entraînement et de validation après ces traitements :

| Type                   | E     | V    |
|------------------------|-------|------|
| <i>searches/clicks</i> | 18571 | 6920 |

#### 3.1 Prétraitements des données

Maintenant que nous avons des observations avec des variables explicatives et des étiquettes, la prochaine étape consiste à effectuer certains prétraitements sur ce jeu de données. Nous avons fait plusieurs combinaisons de pré-traitements de données. Premièrement, nous avons sélectionné les variables que nous voulions inclure dans notre modèle prédictif, c'est à dire : `query_expression`, `search_results`, `user_country` et `user_language`.

Une imputation sur les données manquantes a aussi été effectuée. Lorsqu'une donnée était manquante,

nous avons pris la valeur moyenne de cette variable pour en faire l'imputation.

En ce qui concerne les transformations sur les variables, trois type transformations ont été effectué. Premièrement, une transformation des variables catégorielles vers des valeurs numériques. Deuxièmement, une normalisation de la variable `query_expression` est effectuée. Finalement, cette même variable est vectorisée dans le but d'être traité par notre modèle prédictif.

Nous avons testés deux scénarios pour la normalisation des textes de recherches. Nous avons intégré dans notre *pipeline* la normalisation de *Stemmer*, c'est-à-dire de couper les suffixes de chaque mots. Nous avons aussi laissé la possibilité de ne pas faire de normalisation. Pour la vectorisation des textes de recherches, nous avons intégré quatre types de vectorisation dans notre *pipeline*. Nous avons utilisé *Count-Vectorizer* de la librairie Python *sklearn*. Cette vectorisation compte l'occurrence des mots dans chaque textes de recherches. Nous avons aussi testé en mettant l'option binaire, qui fait la distinction entre la présence du mot ou non, au lieu de faire le compte. La méthode *tf-idf* est également testée. Cette méthode ressemble à *CountVectorizer*, mais il y a une pondération qui tient compte de la présence générale du mot dans l'ensemble des documents. Le poids,  $w_{i,j}$ , pour chaque mot de chaque document est calculé de cette façon :

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Le  $tf_{i,j}$  représente le nombre d'occurrences du mot  $i$  dans la phrase  $j$ ,  $df_i$  représente le nombre d'occurrences du mot  $i$  dans l'ensemble des phrases et  $N$  représente le nombre de phrases au total. Finalement, nous avons aussi testé avec *Word2Vec* de la librairie Python *gensim*. ICI SAM

### 3.2 Modélisation

Une fois que nous avons un jeu de données interprétable par un algorithme d'apprentissage supervisé, la prochaine étape consiste à tester différents type d'algorithme et aussi différentes combinaisons d'hyperparamètres. Voici les différents algorithmes que nous avons testé :

- Classifieur  $k$ -PPV
- Perceptron multicouche

Pour le classifieur  $k$ -PPV nous avons considéré deux hyperparamètres différents. En premier lieu, nous évidemment testé plusieurs valeurs pour le nombre de voisins ( $k$ ). Nous avons testé les valeurs suivantes :

1, 3, 8, 11, 15, 25 et 50. Ensuite, nous avons également testé différentes fonction de poids : **uniform** et **distance**.

Pour le perceptron multicouche, nous avons seulement testé la fonction d'activation *ReLU*. Cependant, nous avons testé plusieurs topographies de réseaux : 1 couche cachée avec 100 neurones, 2 couches cachées avec 100 neurones chacune et 3 couches cachées avec également 100 neurones par couche.

Pour attaquer de façon claire le problème de l'optimisation des pré-traitements et de la sélection des hyperparamètres, nous avons utilisé le module *pipeline* de la librairie Python *sklearn*. Ce dernier nous permet de définir nos différentes étapes de pré-traitement par des classes dont les paramètres sont modifiables. En procédant ainsi, on peut faire une recherche en grille non seulement sur les hyperparamètres de nos modèles, mais aussi sur les différentes prétraitements possibles.

Pour évaluer la performance de notre modèle et ainsi choisir la configuration optimale, il faut une mesure de performance. Coveo a défini ce qui constitue une bonne prédiction. Chaque document cliqué par un utilisateur après avoir effectué une recherche est considéré comme pertinent. Ce sont donc les cibles que le modèle doit prédire pour une recherche donnée. Une bonne prédiction doit prédire un ensemble d'au plus 5 documents parmi lesquels on doit retrouver au moins 1 document pertinent. Cette condition est formellement donnée par la fonction de perte suivante, qui indique qu'une perte de 1 résulte de l'absence de documents communs entre l'ensemble de 5 documents renvoyé par notre modèle de prédiction, et l'ensemble des documents cliqués pour la recherche en question :

$$l(y_i, \hat{y}_i) := \begin{cases} 1 & \text{si } y_i \cap \hat{y}_i = \emptyset; \\ 0 & \text{autrement.} \end{cases}$$

## 4 Résultats expérimentaux

Présentation des résultats avec tableaux, figures et tests statistiques. On n'analyse rien ici, on ne fait que montrer ce que nous avons obtenue avec l'approche décrite plus haut.

## 5 Analyse des résultats

Faire du gros blabla sale sur les résultats. Pourquoi notre score n'est pas si élevé que ça, comment on aurait pu améliorer l'efficacité des embeddings. Techniques qui fonctionnent le mieux et avantages/in-

convénients des différentes techniques en production (temps d’entraînement, mémoire, etc.)

## 6 Conclusion

Ouverture philosophique, constats du projet et apprentissages

## Références

- Ai, Q., Bi, K., Guo, J., Croft, W. B. (2018). Learning a deep listwise context model for ranking refinement.  
doi: 10.1145/3209978.3209985
- Alpaydin, E. (2010). *Introduction to machine learning* (2nd éd.). The MIT Press.

- Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep learning*. MIT Press. (<http://www.deeplearningbook.org>)
- Hastie, T., Tibshirani, R., Friedman, J. (2001). *The elements of statistical learning*. New York, NY, USA : Springer New York Inc.
- Jurafsky, D., Martin, J. H. (2014). *Speech and language processing* (Vol. 3). Pearson London.
- Pang, L., Lan, Y., Guo, J., Xu, J., Xu, J., Cheng, X. (2017). DeepRank : A new deep architecture for relevance ranking in information retrieval.  
doi: 10.1145/3132847.3132914
- Schütze, H., Manning, C. D., Raghavan, P. (2008). *Introduction to information retrieval* (Vol. 39). Cambridge University Press.