Rapport d'analyse de Stack Overflow avec des méthodes de data mining

Cuzin-Rambaud Valentin et Marrel Pierre-Emmanuel

Abstract Dans ce rapport, nous explorons un jeu de donnée, extrait du forum Stack Overflow, à l'aide de plusieurs méthodes de Data mining. Nous utilisons différentes méthodes sur les systèmes de recommandation (omniprésents sur le web), comme KNN, SVD et NMF. Nous utilisons ces méthodes dans le but de recommander des *tags* à l'utilisateur. Le jeu de donnée possède une date de création. Nous analyserons si des paternes peuvent être détectés, et si des anomalies sont présentes. De plus les questions possèdent plusieurs *tags*, nous allons essayer de créer un graphique orienté avec le traitement des paternes fréquents sur les *tags*. Afin de comprendre quelles sont les associations possibles de *tags* entre eux. Enfin, avec le Word Embedding, nous regardons la similarité entre les questions à partir du titre de celle-ci uniquement.

Introduction

Nos données sont composées de trois parties : les questions, les réponses et les *tags*. Les *tags* comme les réponses réfèrent aux questions directement. Chaque question ou réponse est datée et un score est associé à celle-ci. Quelques données sont manquantes, mais en raison de la taille de nos données, nous décidons de supprimer les informations plutôt que les imputer. En effet, il y a 1 264 216 questions, 2 014 516 de réponses et 37 034 *tags* différents. Pour des raisons de limitation, que ce soit la mémoire ou le temps de calcul, nous avons souvent sélectionné qu'un échantillon de nos données, la sélection de données se fait par le score des questions ou des réponses.

Recommandation de tag

Chaque question est labellisée par un ensemble de *tags*, et on remarque que les utilisateurs préfère répondre à un certain type de questions. On peut faire de la recommandation de *tag* pour les utilisateurs, afin de les orienter vers les questions qu'il pourrait les intéresser.

Prétraitement de la donnée

Premièrement, nous avons pré-traiter la donnée, en raison de sa taille conséquente. Nous avons sélectionné un sous-ensemble pertinent pour la recommandation de *tag*. nous avons

sélectionné les 0.001% de *tag* les plus utilisés du forum, cela représente 38 *tags*. Nous avons ensuite sélectionné les réponses des 0.1% d'utilisateurs les plus actifs pour un total de 34 446 réponses, cela fait 1338 utilisateurs. Enfin, nous avons créé une matrice Utilisateurs par *tag* donc 1338 lignes et 38 en colonnes. Cette matrice est remplie par la proportion du nombre de réponses dans le domaine de ce tag. Pour calculer ce score de participation, pour chaque utilisateur, nous avons compté le nombre de réponses associé par *tag*, puis nous avons fait la moyenne par utilisateurs de façon à ce que la somme des scores d'un individu soit égale à un. par exemple voici un échantillon de l'utilisateur cinq 1 :

c	c++	c#	python	ruby	ruby-on-rails	sql	sql-server	xml	r
0.06	0.0	0.2	0.13	0.06	0.06	0.06	0.06	0.0	0.0

TABLE 1. *Échantillon de l'utilisateur 5 dans la matrice Utilisateur/Tag*

Factorisation de Matrices

Nous commençons notre analyse par la méthode NMF. Le principe est de trouver deux matrices de telle sorte à ce que leur produit scalaire donne une matrice de recommandation. Ainsi sur le même précédent échantillons, nous obtenons ceci 2:

c	c++	c#	python	ruby	ruby-on-rails	sql	sql-server	xml	r
0.06	0.0	0.2	0.13	0.05	0.08	0.08	0.03	0.01	0.0

TABLE 2. *Échantillon de l'utilisateur 5 dans la matrice de recommandation produite par NMF*

Par la suite, nous avons évalué trois méthodes de recommandation : KNN, SVD, NMF, choisit pour leur efficacité et leurs popularités. KNN est une méthode basé utilisateur. SVD contrairement à NMF n'impose pas la non-négativité, SVD décompose en trois matrices, SVD est utilisé dans une variété d'autre problème comme la réduction de dimension, analyse de corrélation, etc. Pour évaluer nos algorithmes, nous utilisons la méthode de validation croisée. Pour chaque algorithme, nous avons mesuré sur cinq partitions de la donnée la RSME et la MAE. voici nos résultats 3:

TABLE 3. *Évaluation sur 5 partitions de RSME et MAE pour 3 algos de recommandation*

ALGO	MÉTHODE	PART 1	PART 2	PART 3	PART 4	PART 5	MEAN	STD
SVD	RSME	0.9320	0.9313	0.9315	0.9328	0.9297	0.9315	0.0010
	MAE	0.9309	0.9302	0.9304	0.9317	0.9285	0.9304	0.0011
NMF	RSME	0.9312	0.9320	0.9313	0.9309	0.9320	0.9315	0.0004
	MAE	0.9301	0.9308	0.9303	0.9297	0.9309	0.9304	0.0005
KNN	RSME	0.9316	0.9311	0.9312	0.9315	0.9319	0.9315	0.0003
	MAE	0.9305	0.9300	0.9301	0.9304	0.9308	0.9304	0.0003

Comme on peut le voir sur ce tableau 3, les trois algos produisent des performances similaires. L'*accuracy* moyenne est de **0.93** ce qui est très correcte. Nous pourrions faire une recherche exhaustive d'hyper-paramètre d'un algorithme pour espérer avoir de meilleures performances. Cependant, nous avons déjà suffisamment de bonne performance sur notre jeu de donnée. Ces bonnes performances sont dues en parties au prétraitement de la donnée qui est qualitative avec les utilisateurs qui participe le plus sur les *tags* les plus fréquents, cela réduit le problème des utilisateurs qui ne participe que peu, ou des *tags* peu utilisés.

Calcul sur les séries de temps

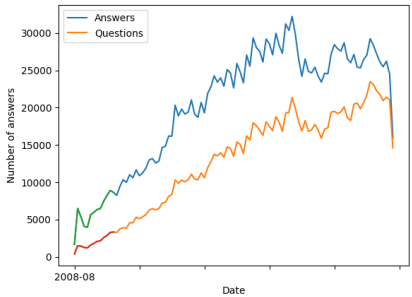
Nos données étant datées, nous pouvons analyser de potentielles tendances dans la fréquence de publications. Ainsi que trouver de potentielles anomalies

Sélection d'une période

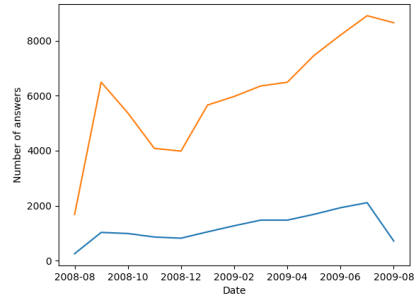
Nous avons sélectionné une période allant du 15 septembre 2008 au 15 septembre 2009, en évitant le premier mois de collecte (août 2008) pour limiter les biais liés à des données potentiellement incomplètes. En effet, nous pouvons voir 1 un volume de données faible pour le premier mois. Cette période d'un an offre un échantillon représentatif pour une première analyse des tendances et relations entre questions et réponse.

Détection d'anomalies

Un premier but est de trouver des anomalies dans la publication, en suivant la tendance et la saisonnalité. Pour calculer les compositions périodiques de publication, nous avons cherché un cycle. Pour ce faire, nous avons utilisé un calcul d'*acf* en suivant les indications de Drelczuk, n.d. De manière logique, nous attendons un cycle de sept jours qui correspond à une semaine.

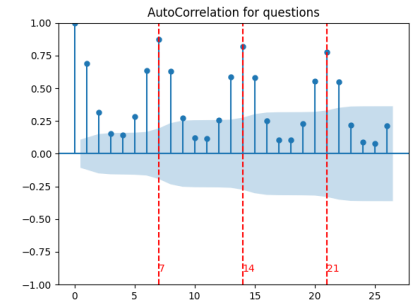


Nos données entière. En vert et rouge notre sélection

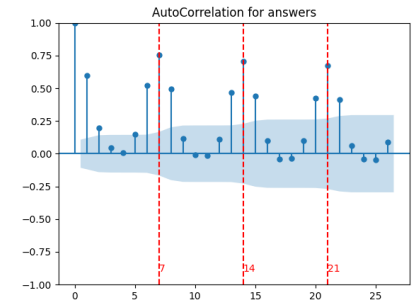


Notre séletion de données. En plus le mois 2008-08.

FIGURE 1. *Visualisation de nos données, et de la sélection temporelle*



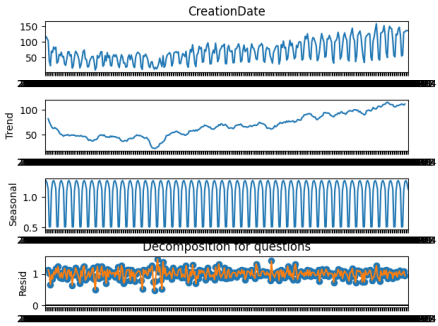
L'acf des questions
En rouge les trois meilleurs acf, lag de 0 exclue



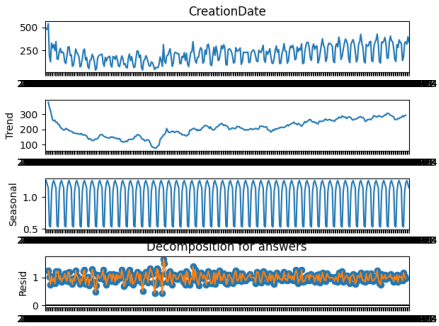
L'acf des réponses
En rouge les trois meilleurs acf, lag de 0 exclue

FIGURE 2. *Graphique de l'acf de la publication de question et réponse de Stack Overflow pour une certaine période (2008/09/15 – 2009/09/15)*

Sur nos graphiques 2, le cycle d'une semaine est confirmé. Nous voyons que l'*acf* est le meilleur tous les sept jours. Maintenant que nous avons notre cycle, nous pouvons décomposer notre période en tendance, saisonnalité et résiduels.



Decomposition sur les questions



Decomposition sur les réponses

FIGURE 3. *Décomposition en tendance, saisonnalité et résiduels de la publication sur la période de 2008-09-15 au 2009-08-15*

Ce graphique 3 montre une tendance, une saisonnalité et des résiduels sur notre période d'un an. Nous pouvons maintenant analyser les résiduels pour trouver les anomalies. Afin de détecter une anomalie, nous avons déterminé un seuil. Avant de trouver ce seuil, nous en avons testé plusieurs. Celui choisi permet d'avoir une cohérence dans les anomalies trouvée (ni trop peu, ni trop).

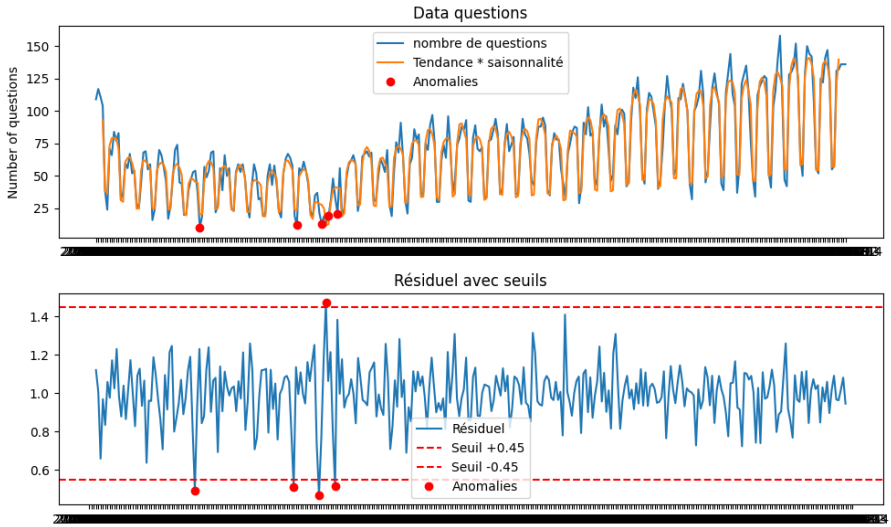
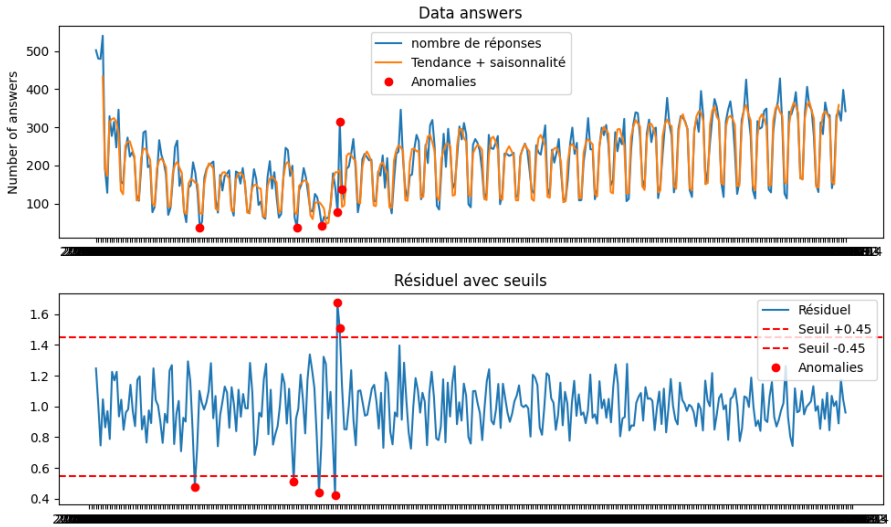


FIGURE 4. *Décomposition de la publication des questions sur la période de 2008-09-15 au 2009-08-15*



La première courbe montre les vraies données par rapport à la tendance multipliée par la saisonnalité. La seconde courbe indique les résiduels ainsi que le seuil que nous avons choisi pour déterminer une anomalie ou non. Ainsi, nous avons pu mettre en rouge les points déterminés comme anormales.

FIGURE 5. *Décomposition de la publication des réponses sur la période de 2008-09-15 au 2009-08-15*

Ces graphiques 4, 5 illustre pour, respectivement, les questions et réponses les anomalies détectées avec les résiduels. Nous pouvons étudier ces anomalies pour trouver des explications.

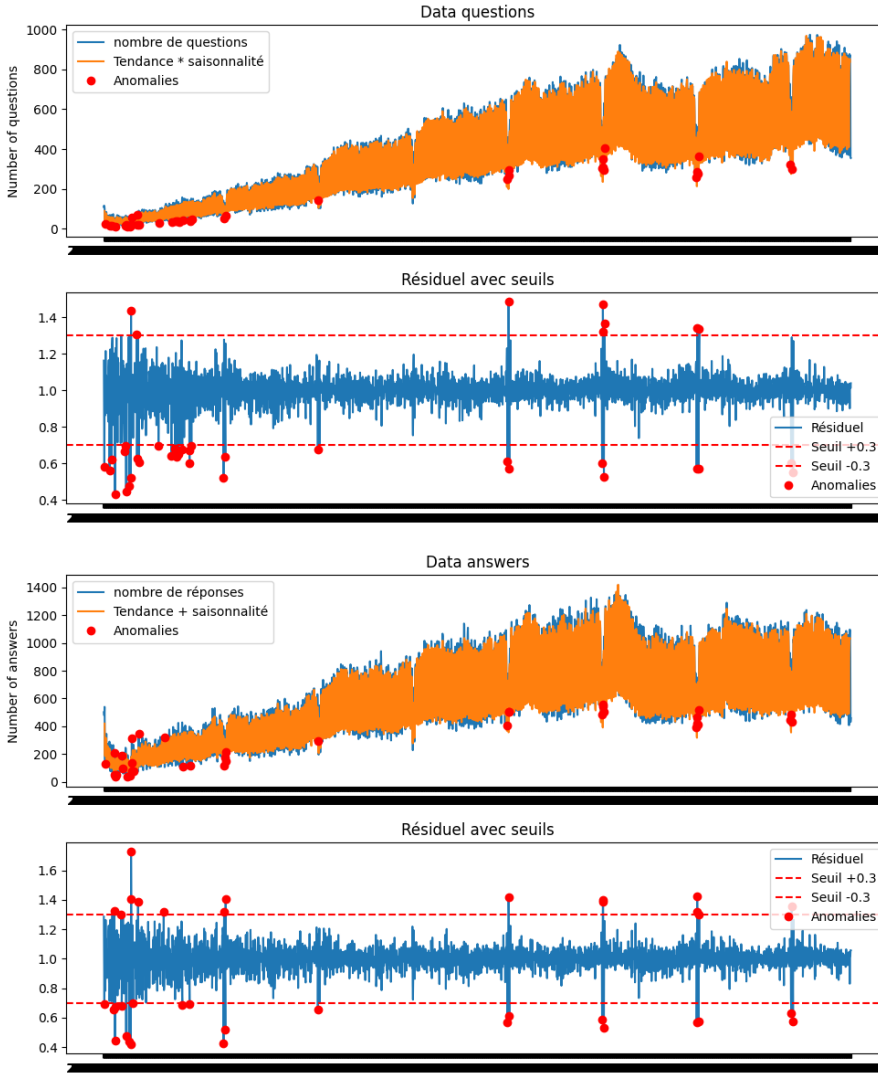
Jour anormal	Questions	Réponses	Jour férié
2008-11-01	Moins de publications	Moins de publications	Non
2008-12-14	Moins de publications	Moins de publications	Non
2008-12-25	Moins de publications	Moins de publications	**Oui**
2009-01-01	Moins de publications	Moins de publications	**Oui**
2009-01-02	**Donnée normale**	**Plus** de publications	Non
2009-01-03	**Donnée normale**	Moins de publications	Non

Ce tableau montre les anomalies par rapport à un certain jour.
Il est indiqué lorsqu'il s'agit d'un jour fériés.

TABLE 4. *Jour anormal trouvé*

Ce tableau 4 récapitule les jours trouvés comme anormales. Dans la plupart des cas, il s'agit d'un taux plus bas que celui attendu. Les jours anormaux sont présents pour les questions et réponse (sauf pour les 2-3 janvier). Nous remarquons aussi que deux des anomalies se trouve être le jour de Noël et celui du Nouvel An. Les fins de semaines, grâce à la décomposition en saisonnalité, ne sont pas détectés comme anormales. Les autres mois (autre que décembre et novembre) ne présentent pas d'anomalie.

Les jours correspondent à de potentielles vacances. Sauf le 2008-11-01, qui est étonnant. Nous avons donc cherché si certains jours étaient anormaux pour les autres années. Ou s'il s'agit d'une coïncidence. Nous avons donc répété les étapes jusqu'à l'analyse des décompositions en tendance, saisonnalité et résiduels.



Sur ces graphiques, nous apercevons des piques régulés de résiduels.
Ces piques surviennent de manière périodique, tous les ans.
Le seuil est diminué à 0.3.

FIGURE 6. *Décomposition de la publication sur la période de 2008-09-15 au 2009-08-15*

Nous avons diminué le seuil pour avoir plus de jour férié. Ces graphiques 6 montrent que les résiduels varient en fonction des années. Les années les plus instables sont celle de 2008 - 2009, qui correspondent aussi aux années avec les moins de données. L'analyse des piques d'anomalie demanderait une étude comme celle de Hochenbaum, Vallis, and Kejariwal 2017, qui permet de trouver des anomalies locales ou globaux. Dans notre cas, nous avons décidé d'ignorer les anomalies exceptionnelles des années 2008 et 2009, et la non-présence d'anomalie pour l'année 2011.

mois-jour	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**
12-14	2	0	0	0	0	0	0	0	None
12-25	2	2	0	0	2	2	2	2	None
12-27	0	**Réponse**	0	0	0	0	**Réponse**	**Réponse**	None
12-28	0	0	0	0	0	2	2	0	None
12-29	0	0	0	0	2	2	0	0	None
01-1	None	2	2	0	0	2	2	2	2
01-2	None	2	**Réponse**	0	0	0	0	0	0
01-3	None	**Réponse**	0	0	0	0	0	0	0

Ce tableau montre les jours férié par rapport aux années.
2 : Questions et Réponses ont une anomalie.
0 : Ni les réponses ni les questions n'ont d'anomalie.
None : Pas de données

TABLE 5. *Tableau récapitulatif des anomalies*

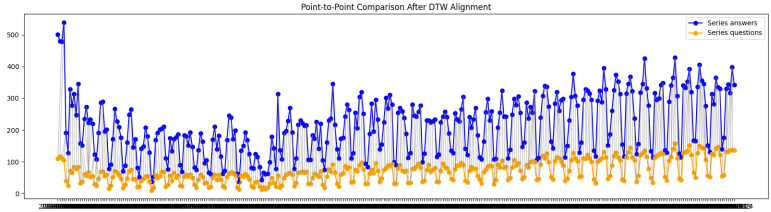
De manière plus générale, si nous comparons à la période de l'année, nous remarquons que les diminutions visibles sur le total des données se situent de manière cyclique à Noël. L'année 2011 ne montre pas d'anomalies, nous apercevons tout de même un pique de résiduel, mais moins conséquent.

Cluster avec *time warping*

La visualisation globale de publication semble indiquer qu'il n'y a pas de décalage entre la publication de questions et de réponse. Dans un premier temps, nous nous assurons que c'est bien le cas. Après, nous cherchons à voir s'il existe un décalage entre la publication de *tags*. Par exemple, si le *tag* c++ est en décalage temporel avec le *tag* shell.

Données globals

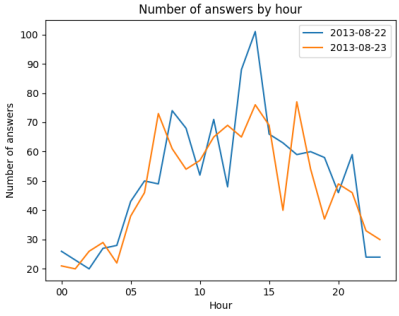
Nous avons calculé la courbe de *time warping*, en suivant les indications du site de Stent, n.d. et de la librairie "Dynamic Time Warping (DTW)", n.d. Il y a beaucoup plus de question que de réponses. Pour comparer les deux courbes, nous avons centré réduit les données.



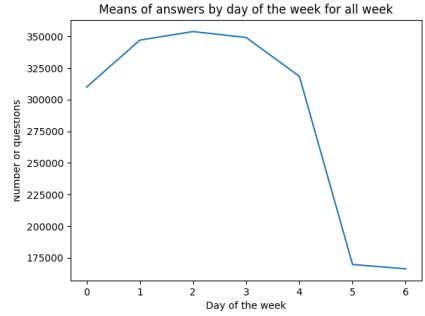
Time warping entre les quesitons et réponse après réallignement

FIGURE 7. *Décomposition de la publication sur la période de 2008-09-15 au 2009-08-10*

Comme illustré à la figure 7, il n'existe pas de décalage temporel significatif entre les publications de questions et de réponses, indiquant une absence d'influence directe entre ces deux types de publications. Les publications suivent un cycle lié aux jours de travail, avec une diminution notable durant le week-end (figure 8).



Pour deux jour, le nombre de réponse par heure
.

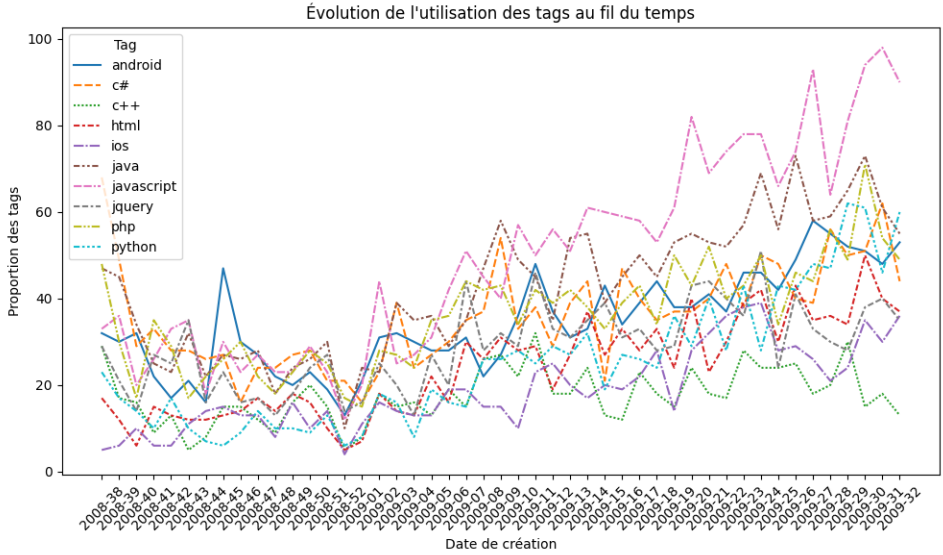


Le nombre moyen de réponses par jour.
0 : Lundi ... 6 : dimanche

FIGURE 8. *Décomposition de la publication sur la période de 2008-09-15 au 2009-08-10*

Données avec *tags*

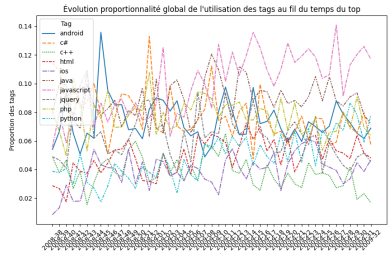
L'objectif est d'analyser si certaines tendances de *tags* en influencent d'autres. La période sélectionnée s'étend du 15 septembre 2008 au 10 août 2009 (exclu), car les *tags* des questions ne sont pas disponibles après cette date. Une première visualisation montre un manque de données, avec des jours où aucun *tag* du top 10 n'est associé à une question. Pour pallier ce problème, les questions ont été regroupées par semaine, ce qui permet de lisser les variations et d'assurer une meilleure représentativité. La période débute un dimanche (15 septembre 2008) et se termine un lundi (10 août 2009) exclus, garantissant ainsi des semaines complètes pour l'analyse.



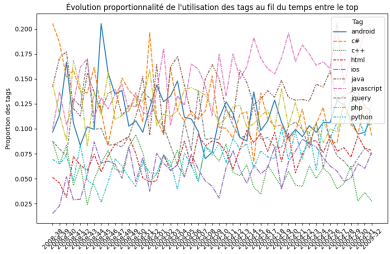
Seulement les 10 meilleurs *tags*. En absice l'année - le nombre de semaine.

FIGURE 9. *L'évolution de l'utilisation des tags par semaines.*

En plus d'analyser les données brutes 9, nous pouvons analyser la proportionnalité des *tags*, pour voir les tendances de *tags* tendances. Nous avons divisé les résultats par le nombre total de questions.



Tags proportionel au total des questions



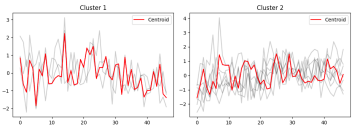
Les questions qui ne possède pas un *tag* dans le top 10 n'est pas pris en compte

FIGURE 10. *Visualisation de la proportionnalité des tags en fonction des semaines*

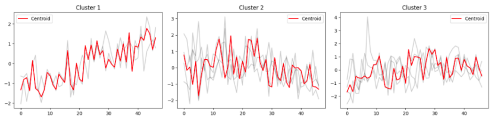
Nous avons calculé de deux manières l'influence des *tags*, il n'y a pas de grande différence entre les deux méthodes. Par la suite, nous allons garder la proportion globale.

Pour calculer le *time warping* entre les *tags*, les questions ont été regroupées par *tag*. Le

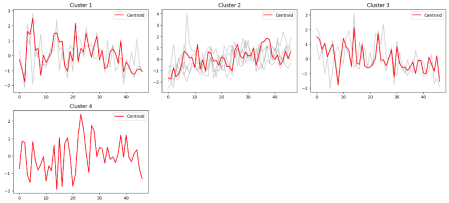
time warping ainsi obtenu offre une nouvelle mesure pour intégrer la dimension temporelle dans l'analyse (cf. annexe 22).



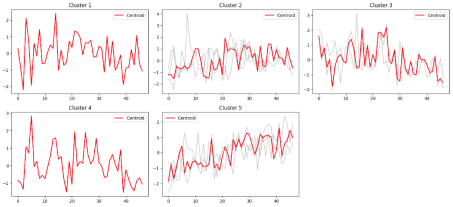
Claculs avec 2 clusters.
Cluster 1 : c#, c++, php
Cluster 2 : android, html, ios, java, javascript, jquery, python



Calculs avec 3 clusters.
Cluster 1 : javascript, python
Cluster 2 : c#, c++, jquery, php
Cluster 3 : android, html, ios, java



Claculs avec 4 clusters.
Cluster 1 : c#, c++, jquery
Cluster 2 : android, html, java
CLuster 3 : ios, javascript, python
Cluster 4 : php
.



Calculs avec 5 clusters.
Cluster 1 : php
Cluster 2 : android, html, java
Cluster 3 : c++, c#
Cluster 4 : jquery
Cluster 5 : ios, javascript, python

FIGURE 11. *Visualisation des clusters trouvée avec Kmeans et prise en compte de time warping*

Nous trouvons des clusters 11, mais leur centroïde ne sont pas énormément différents. Les *tags* regroupés ne semblent pas complétement incohérents, le c# et le c++ étant des langages liés. Android et iOS faisant référence aux téléphones mobiles. Nous devons tout de même noter que certain lié ne semble pas pertinent (Exemple le python et le JavaScript). Comme seconde méthode pour trouver des clusters, nous allons nous intéresser aux patterns fréquents, et analyser de potentiels clusters résultants.

Pattern fréquent

Nous avons une large sélection de *tags* ainsi que de questions les utilisant. Nos données sont donc propices à la recherche de pattern fréquent. Nous avons suivi les indications de Tuts, n.d., pour appliquer un *one-hot encodage* suivit de l'utilisation de la librairie Raschka, n.d. pour utiliser l'algorithme apriori puis trouver les règles associés. Nous avons adapté un minimum support pour les *items*. Ce minimum a été défini, le plus petit que notre matériel de calcul pouvait accepter. A noté que modifier ce minimum permet d'obtenir des résultats, et des clusters différents.

Les résultats présentés dans le tableau 6 mettent en évidence que certaines associations

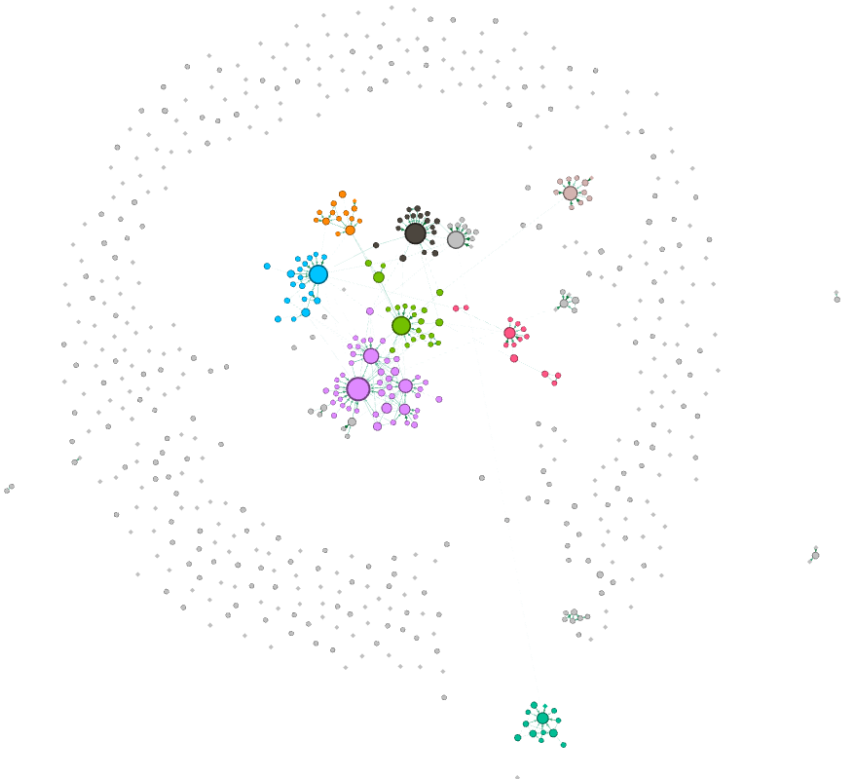
Support	**Itemsets**
0.102421	(javascript)
0.088688	(java)
0.077479	(c#)
0.076153	(php)
0.070420	(android)
⋮	⋮
0.001027	(c#, windows-phone-8)
0.001027	(vb.net, c#)
0.001027	(batch-file, windows)
0.001027	(c#, asp.net, .net)
0.001027	(webview)

618 groupements de *tags* trouvés, la plupart sont des *tags* seuls

TABLE 6. *Résultat des supports pour les items.*

de tags peuvent offrir des *insights* intéressants. En appliquant des règles d'association, nous avons calculé plusieurs métriques clés, telles que : confidence, lift, leverage, conviction, et Zhang's metric.

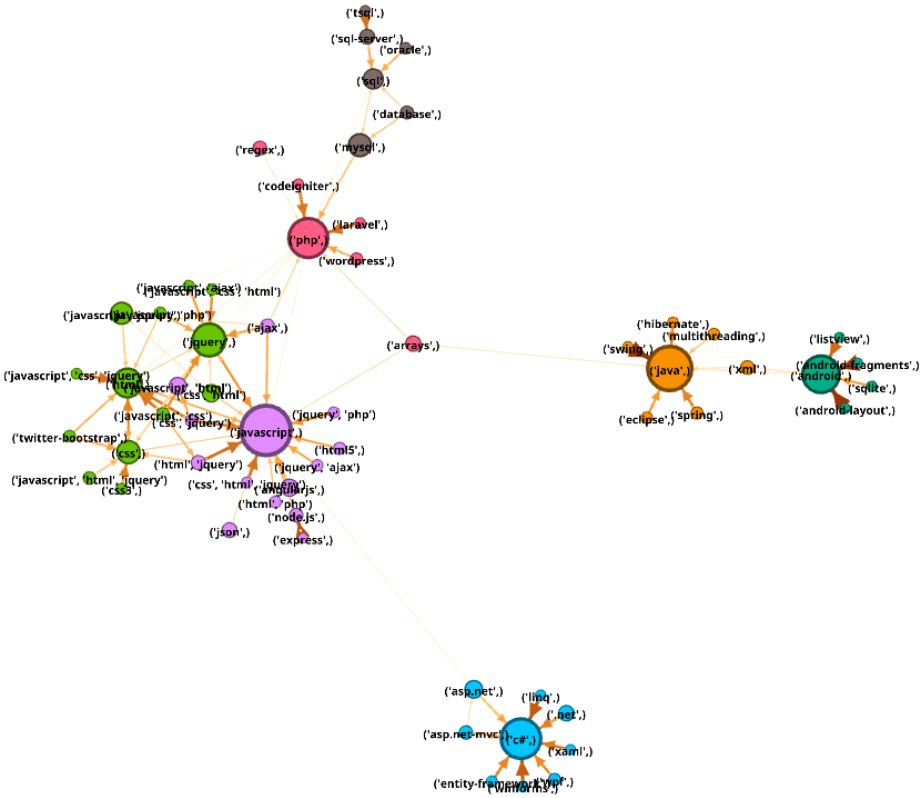
Pour la visualisation des graphes, nous avons choisi d'utiliser la métrique support comme taille des nœuds afin de refléter leur importance relative. Les liens dirigés entre les nœuds sont représentés avec une couleur classée en fonction de la confidence, qui illustre l'influence d'un nœud sur un autre. Après avoir comparé la pertinence des différentes métriques, nous avons retenu cette approche, car elle permet une bonne interprétation des relations entre les nœuds.



Graphique des *tags* les liens représente l'association de groupe de *tags*

FIGURE 12. *Graphique des tags ou groupe de tags*

Comme l'illustre le graphique 12, de nombreux tags apparaissent de manière isolée, sans connexions avec d'autres. Nous observons également la formation de sept groupes connexe distincts de *tags*, chacun possédant au moins un lien interne. Parmi ceux-ci, un groupe de taille notable se distingue, suggérant une plus grande densité de connexions dans le réseau. Les couleurs des nœuds représentent différents clusters calculés avec GEPHI et la métrique "Détection de communauté $\rightarrow$ modularité".

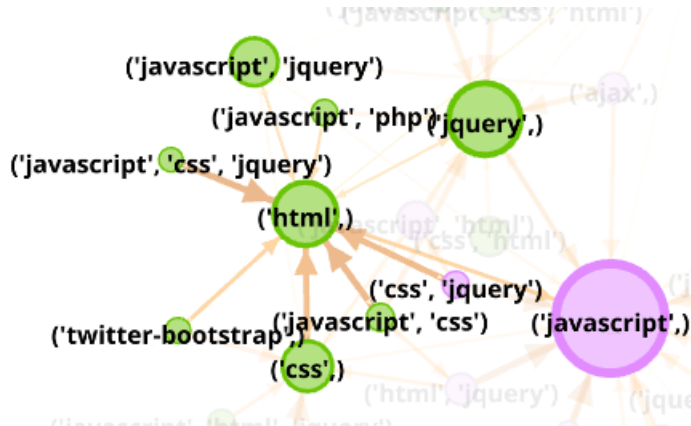


Le groupe connexe le plus importants.

FIGURE 13. *Sous graphique n°1*

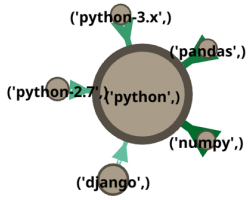
Nous pouvons voir plusieurs informations dans ce sous graphique 13. La première est la construction de communauté qui semble pertinent (pour voir en détail les différents clusters, cf. annexe 21). La deuxième montre que des groupes de *tags*, comme le JavaScript et le CSS, peuvent influencer un autre tag, ici le HTML. Alors que seul le JavaScript n'a pas d'influence sur le *tag* HTML 14.

Nous regarder la pertinence des autres groupe connexe.

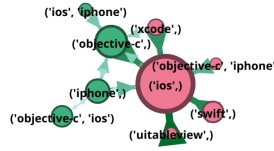


Il y a un lien de HTML vers JavaScript, mais pas de JavaScript vers HTML.

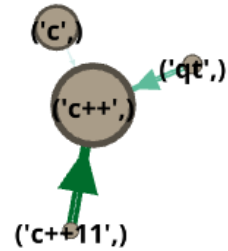
FIGURE 14. *Zoom sur le tag HTML du sous graphique n°1*



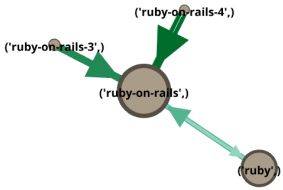
Cluster avec les *tags* en rapport avec python. Un seul cluster détecté.



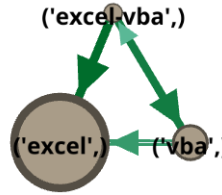
Deux clusters détecté, objectif-C influence iOS et inversement.



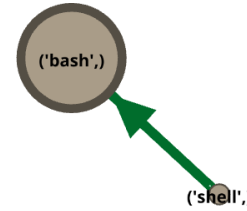
Groupe avec c++, le *tag* c++ est souvent présent. c++11 est souvent présent avec le c++



Tous les tags font référence à ruby-on-rail. Nous pouvons remarquer que la version plus récente implique plus le ruby-on-rail de base.



L'association dans le cluster semble logique, on peut noter qu'exel n'implique pas beaucoup vba. L'inverse étant faux, vba implique souvent excel.



Les questions de bash sont populaires, si une question parle de shell, alors elle est aussi asosicer a bash

FIGURE 15. *Autres sous graphs*

Contrairement à notre première analyse de clusters, Python n'apparaît pas comme étant directement lié à JavaScript. De manière générale, les clusters obtenus 15 semblent plus cohérents, et l'analyse couvre un plus grand nombre de *tags*.

Il apparaît que la majorité des *tags* sont indépendants les uns des autres. Les langages de programmation étroitement liés — que ce soit en raison de versions différentes ou d'une association fréquente dans les pratiques de développement — tendent à être présents ensemble dans les mêmes questions, montrant ainsi des communautés.

Word Embedding

Notre jeu de donnée comporte une quantité non négligeable de texte que ce soit à travers les questions (titres + corps de questions) et les réponses. Les méthodes de *word embedding* permettent de comprendre notre jeu de donnée, à travers la sémantique des mots. La visualisation des vecteurs de mots permet de comparer les concepts clés associés, tandis que le rapprochement entre les questions basé sur leur similarité permet de visualiser les connexions entre les questions à travers un graphe.

Prétraitement de la donnée

Notre modèle Word2Vec s'entraine sur une liste de texte comprenant toutes les questions, toutes les réponses et tous les titres de questions pour un total de 208 734 mots différents. Nous filtrons les questions et réponses ayant un score supérieur ou égale à 10 pour éviter d'apprendre sur du texte jugé non pertinent par la communauté. Les questions et les réponses sont sous forme HTML, nous faisons donc un premier clean, en supprimant toutes les balises, les ’
n’ et les liens href en gardant bien le texte du lien. Nous avons ensuite appliqué un premier calcul de la métrique tf-idf (permettant de connaître l'importance d'un mot), et nous avons trouvé énormément de mot non interprétable, en voici une liste non exhaustive :

- ’â³’
- ”½wï”
- ’½í²¼’
- ’0000000000000000000000000000000000000000000001401’
- ’ù¾’

La proportion de mot présent qu'une fois représente 50% du dataset. Ainsi, nous avons décidé d'enlever les mots qui ne sont pas assez utilisés. Vous pouvez remarquer aussi la présence de nombre, qui ne permettront pas de faire apprendre notre modèle, nous les enlevons aussi. Pour Finir notre traitement, nous enlevons les stops words, car trop commun pour apporter un contexte au mot étudier.

Création des modèles Word2Vec

En nous appuyant de la bibliothèque Řehůřek, n.d., nous avons entraîné trois modèles différents. Les paramètres communs à ses trois modèles sont les paramètres par défaut

à l'exception du nombre d'époques augmenté de 5 à 10. Le premier modèle s'entraine sur notre corpus nettoyé. Nous l'avons paramétré pour utiliser l'algorithme Skipgram qui est théoriquement plus efficace pour les tâches de similarités entre mots que CBOW. Le deuxième s'entraine sur le corpus non nettoyé, à part l'HTML, et conserve l'algorithme Skipgram. Le troisième est pour la visualisation, alors nous fixons la taille du vecteur à 2, et cette fois-ci utilisons l'algorithme CBOW.

Notre premier test doit montrer l'importance du prétraitement de la donnée et se compare à un modèle pré-entrainé sur les news de googles. Ce test démontre l'importance d'utiliser un modèle spécialisé dans nos données, car les résultats divergent avec le modèle pré-entrainé. Par exemple, voici les mots les plus similaires du mot 'java' 7:

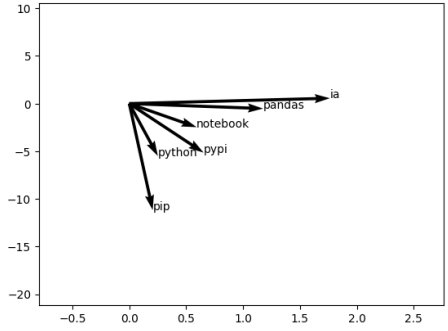
GOOGLES NEWS	coffee	o joe	chai latte	joe	espresso
	0.65	0.63	0.58	0.57	0.56
CLEAN MODEL	defineclass	jdk	doprivileged	util	invokenative
	0.72	0.72	0.69	0.67	0.67
RAW MODEL	defineclass	util	lang	jdk	urlclassloader
	0.74	0.70	0.69	0.69	0.68

TABLE 7. *les 5 mots plus proches du mot java, par différents modèles*

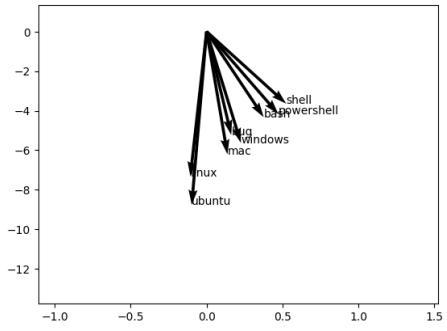
Comme vous pouvez le voir, le modèle pré-entraîné ne ressort pas des mots similaires pertinents avec notre contexte par rapport au deux autres modèles. L'analyse entre les deux autres modèles ne permet pas de trouver une distinction forte pour ce seul test, c'est pour cela que nous les réévaluons par la suite.

Visualisation des vecteurs

En utilisant notre modèle avec un vecteur de taille deux, nous pouvons facilement visualiser les vecteurs de mots dans un plan 16 :



visualisations des vecteurs de mots



visualisations des vecteurs de mots

FIGURE 16. *2 exemples de visualisations des vecteurs de mots*

Le premier exemple met en évidence que le modèle associe étroitement la bibliothèque *pandas* avec le concept d'*intelligence artificielle (IA)*, tout en éloignant le gestionnaire de paquets Python, *pip*, de ces notions. Le terme *notebook* se situe quant à lui entre *python* et *IA*, constituant ainsi un intermédiaire cohérent entre ces deux concepts. En revanche, la position du vecteur correspondant à *pypi* soulève des interrogations : bien qu'étroitement lié à *python*, il apparaît plus proche du mot *notebook* que de *pip*, malgré la forte corrélation entre *pip* et *pypi*.

Le deuxième exemple nous montre deux types de vecteurs proches. Le premier groupe en haut à droite concernent les programmes shell et dériver. Parmi ce groupe, nous retrouvons les mots *shell, powershell* et *bash*. Le deuxième groupe concerne les systèmes d'exploitation *linux, ubuntu, mac, windows* et le vecteur du mot *bug* qui semble coincé entre mac et Windows. Cela peut révéler que les utilisateurs de Stack Overflow ont plus souvent des bugs sur *mac* ou windows que sur *linux*. A noté que selon le score tf-idf, windows est plus cité que linux qui est plus cité que mac. ubuntu est très correctement placé à proximité de linux.

Malgré les informations pertinentes que l'on peut tirer de ces visualisations, notre modèle perd de l'information en raison de la taille petite du vecteur. Un vecteur de mot de taille 100 à 300 permet de capturer le contexte lié pendant l'apprentissage contrairement au vecteur de taille 2 qui ne peut synthétiser tout le contexte.

Similarité entre questions

Nous proposons un modèle doc2vec qui s'appuie sur un modèle Word2Vec donné et les scores tf-idf pour encoder des courtes phrases, et ainsi calculer leur similarité. Pour encoder une phrase, notre modèle fait la somme des vecteurs de mot qui la compose pondérer par les scores tf-idf, ainsi mettant plus d'importances sur les mots moins communs. La similarité entre deux phrases est calculée avec la similarité cosinus.

Retour sur notre contexte, nous voulons trouver la similarité entre des titres de questions afin de voir si la sémantique seule permet de rapprocher des questions entre elles. Voici un test de comparaison de quelques questions avec 2 modèles, le raw et le clean 8 :

Cet exemple permet de montrer l'importance du nettoyage de la donnée. Nous commenterons la troisième phrase où le modèle clean rapproche mieux les deux titres alors qu'ils ne contiennent pas de mot en commun, et la quatrième phrase qui où le modèle raw les rapproche plus fortement, alors que les deux questions n'ont rien à voir (l'un parle de web, l'autre de git).

Nous avons réalisé un modèle doc2vec basé sur le modèle word2vec de visualisation, pour obtenir ce graphique 17 sur les 500 questions les mieux notées par la communauté, colorer par le tag de la question le plus utilisé.

Comme nous pouvons le voir, nous ne pouvons pas tirer de groupe de question directement avec ce graphique. C'est pourquoi une approche en graphe permettrait de conserver une taille de vecteur à 100.

titre de question	model clean	model raw
iframes considered 'bad practice'? Make iframe automatically adjust height according contents using scrollbar?	0.57	0.55
use StringBuilder java Java - Convert integer string	0.65	0.66
Need minimal Django file upload example common uses Python decorators?	0.61	0.57
does Ajax response like 'for (;;); json data ' mean? replace master branch git, entirely, branch?	0.36	0.41

TABLE 8. *Exemple de phrases, avec leur similarité calculé par 2 modèles*

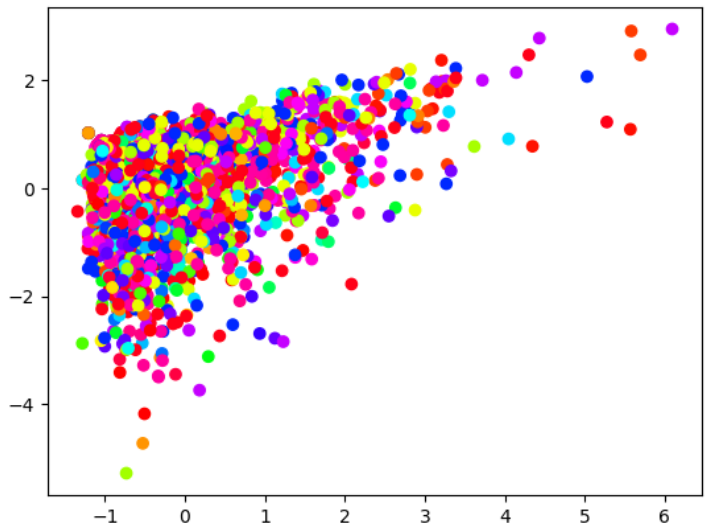


FIGURE 17. *nuage de point, 500 questions, colorer par tag*

Graphe de similarité de question

Pour construire ce graphe, nous définitions un nœud comme le titre d'une question. les attributs de ce nœud sont :

- le tag le plus utilisé
- le score de la communauté
- le titre

Il existe une arête entre 2 nœuds s'il y a 70 % de similarité, nous faisons cela en calculant une matrice d'adjacence. Nous supprimons les nœuds isolés. Vous trouverez à la figure 18 une première visualisation avec matplotlib, avec une coloration par tag.

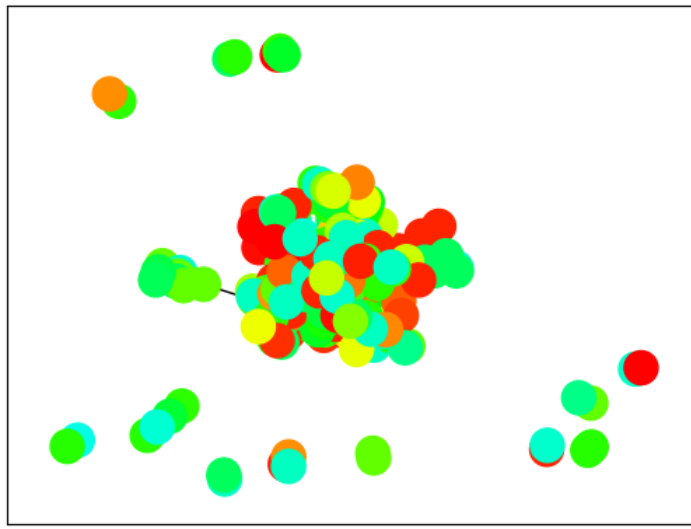


FIGURE 18. *Graphe des titres de questions, avec matplotlib, coloration par tag*

En raison de la faible interprétation des résultats, nous continuerons l'analyse sous l'outil Gephi 19.

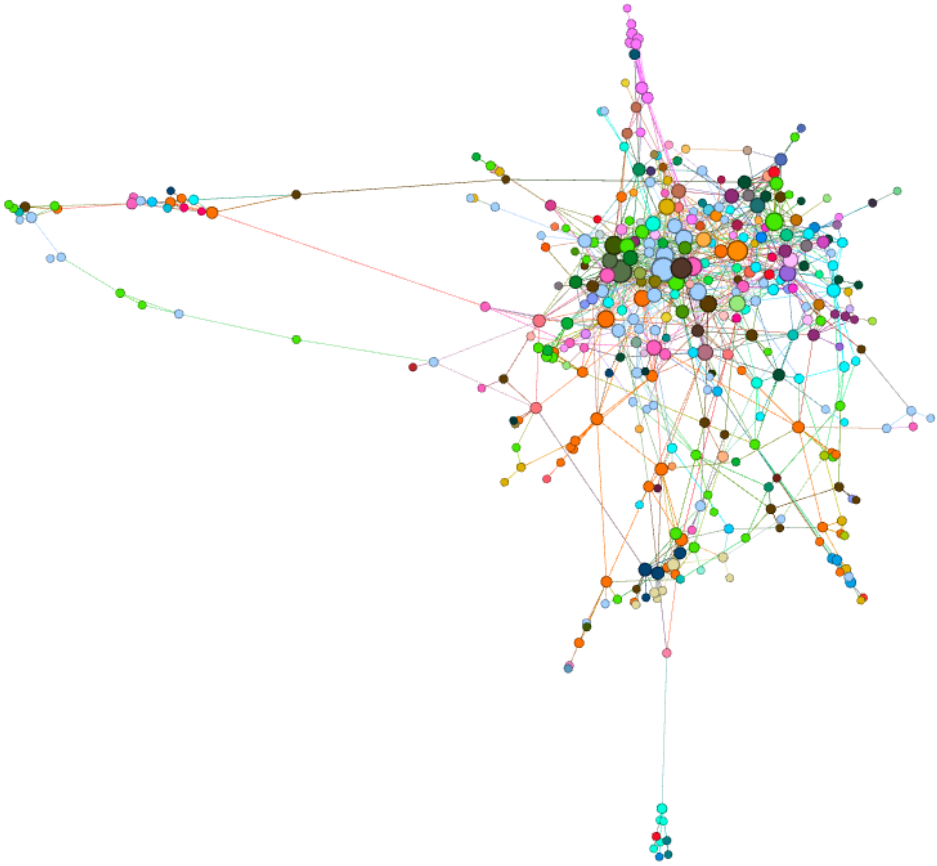
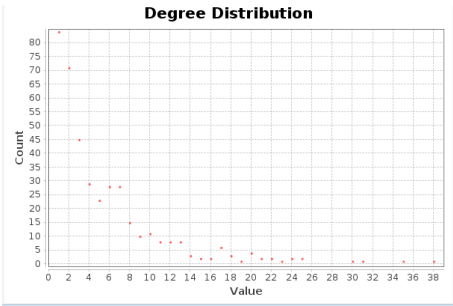


FIGURE 19. *Graphe des titres avec Gephi, coloration par tag, taille des nœuds pondéré par le degré, forceAtlas 2 pour la disposition*

Nous remarquons que les nœuds se lie bien souvent avec d'autre nœud aillant le même tag, c'est par exemple le cas pour les nœuds au nord de la figure qui sont tous violets : ils appartiennent au tag git.

Nous pouvons voir que beaucoup de nœuds sont surconnecté avec trop de nœud 20. Le degré moyen du graphe est de 5.63, ce qui semble correcte, mais certain nœud se connecte avec plus d'une vingtaine d'autres nœuds, ce qui est trop.

Nous pouvons corréler cette information avec une coloration par la métrique du clustering coefficient, qui nous montre que les groupes de nœud qui peuvent former de bons clusters, ne sont pas au centre du graphe, mais plus excentré.



Répartition des degrés du Graphe des titres

Clustering coefficient sur le Graphe des titres

FIGURE 20. *Répartition des degrés et Coefficient de Clustering pour le graphe des titres*

Conclusion

Pour Conclure, nous avons vu que les méthodes de data mining permettent d'extraire de l'information de multiple façon, que ce soit par des systèmes de recommandation, des visualisations temporels, des patterns fréquents, ou du word embedding.

Supplementary Material

(This is dummy text) Supplementary material for this research note is available at <https://doi.org/10.1017/Sxxxxxxxxx>.

References

"Dynamic Time Warping (DTW)". n.d. Accessed: 2024-11-16. Available at <https://dtaidistance.readthedocs.io/en/latest/usage/dtw.html>.

Drelczuk, Krzysztof. n.d. "ACF (autocorrelation function) — simple explanation with Python example". Accessed: 2024-11-16. Available at <https://medium.com/@krzysztofdrelczuk/acf-autocorrelation-function-simple-explanation-with-python-example-492484c32711>.

Stent, Mark. n.d. "Dynamic Time Warping". Accessed: 2024-11-16. Available at <https://medium.com/@markstent/dynamic-time-warping-a8c5027defb6>.

Hochenbaum, Jordan, Owen S. Vallis, and Arun Kejariwal. 2017. Automatic Anomaly Detection in the Cloud Via Statistical Learning. 13 pages, 12 figures, *arXiv preprint arXiv:1704.07706,* arXiv: 1704.07706 [cs.LG]. Available at <https://doi.org/10.48550/arXiv.1704.07706>.

Raschka, Sebastian. n.d. "mlxtend". Accessed 17 November 2024. Available at <https://rasbt.github.io/mlxtend/>.

Řehůřek, Radim. n.d. "Radim Řehůřek: Machine learning consulting". Accessed 17 November 2024. Available at <https://radimrehurek.com/>.

Tuts, Coding. n.d. "python code for Frequent pattern mining". Accessed 17 November 2024. Available at <https://medium.com/@codingTuts/python-code-for-frequent-pattern-mining-45ae6fb167bb>.

Annexe

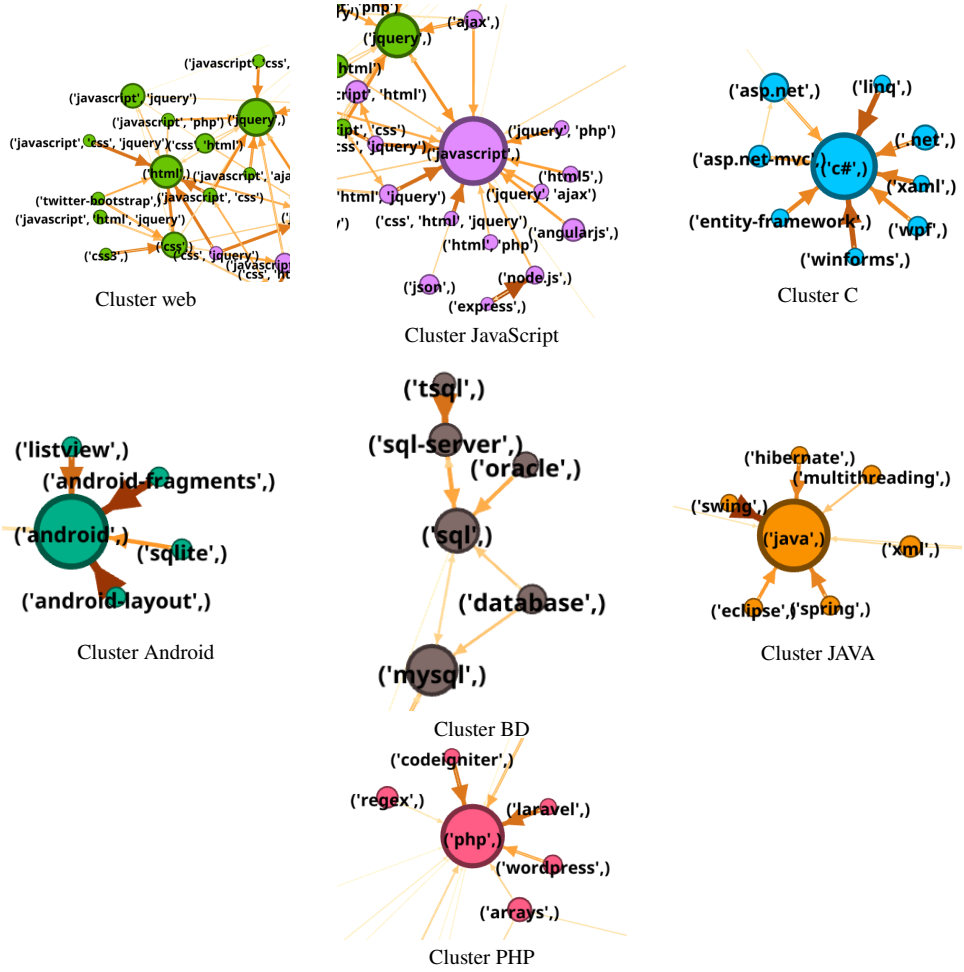
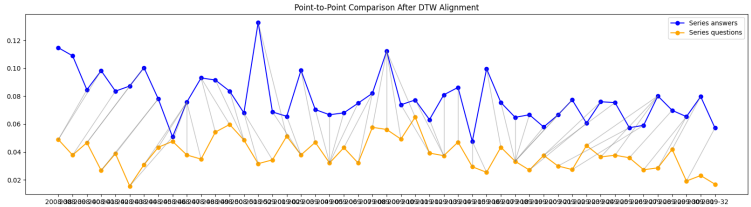
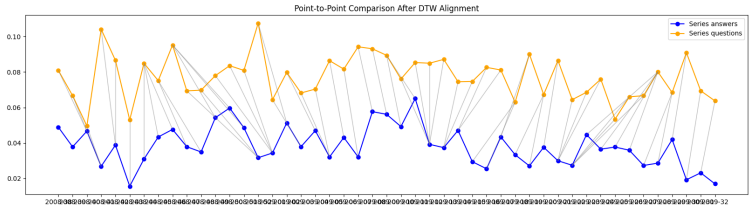


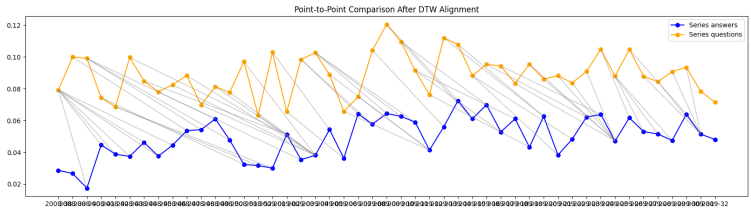
FIGURE 21. *All cluster in subgraph 01*



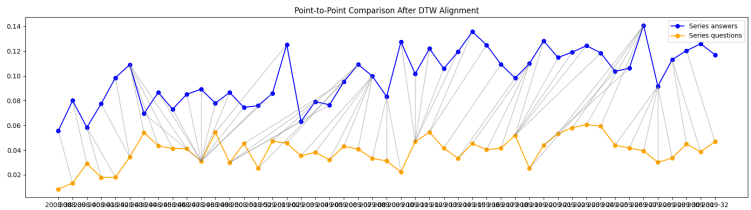
Time Warping entre c# et c++



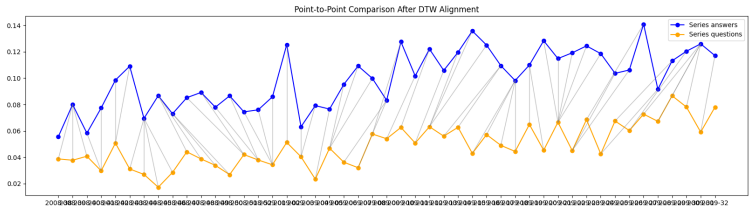
Time warping entre c++ et PHP



Time warping entre HTML et JAVA



Time warping entre JavaScript et iOS



Time warping entre JavaScript et Python

FIGURE 22. *Time Warping avec de bons résultats*