# Rapport Projet 6 : Catégorisez automatiquement des questions

Zakaria MESSAI

# Table des matières

# Introduction

Nous allons dans cette étude proposer de catégoriser des questions issues du site stack-overflow. Stack Overflow est un site d'entraide permettant à ses utilisateurs d'échanger sur des problématiques rencontrées dans le développement informatique. Les questions portent sur les problèmes de programmation des utilisateurs et d'autres utilisateurs peuvent répondre à leurs questions. Un système de votes a également été mise en place de telle sorte que les utilisateurs peuvent se rendre compte des meilleurs questions et des meilleurs réponses données sur le site.

Stack Overflow a été créé en 2008 et est rapidement devenu une référence en la matière. Le but de notre étude est de pouvoir proposer un taguage des questions pour des utilisateurs amateurs pour faire en sorte de leur simplifier l'utilisation du site. Le taguage est déjà présent sur le site de telle sorte que l'on pourra s'aider ou non de ce taguage.

Nous allons dans une première partie essayer de créer des variables qui vont nous permettre dans une seconde partie de pouvoir catégoriser ce texte : chaque catégorie correspondra à un tag. Cette catégorisation se fera soit à l'aide des tags que nous avons soit sans. Il s'agira ensuite de choisir quelle méthode semble la plus pertinente.

## 1. Préparation des données

### 1. Les Données

Pour retirer les données du site, il existe un outil d'export des données présent sur le site stackexchange.explorer. Il s'agit sur ce site d'écrire une requête sql pour pouvoir retirer les questions du site.

Nous avons choisi de ne retirer que les questions (et pas les réponses) avec plus de 3 votes positifs et nous les avons requêtées par paquet d'approximativement 50 000 questions pour éviter des temps de requêtes trop long. Pour ces questions nous avons retiré le corps en texte de la question, le titre et les tags correspondants.

Voici un exemple de requête :

```
SELECT Title, Body, Tags
From Posts
WHERE PostTypeId = 1
AND Id < 500000
AND Score >= 3
```

Nous avons retiré en tout prêt de 200 000 questions ce qui nous semble suffisant pour l'étude que nous allons mener.

## 2. Traitement des données

Nous avons ici à faire à des données textuelles. Il s'agit donc de pouvoir « nettoyer » le texte de toutes les informations qui ne nous intéressent pas et qui ne nous aideront pas à catégorisez les questions.

Nous allons computer les opérations suivantes à notre texte en utilisant le module NLTK :

- Enlever les balises HTML
- Enlever les caractères qui ne sont pas des lettres
- Convertir tous les caractères en minuscule
- Prendre la racine de chaque mot (*stemming*)
- Enlever les mots récurrents du langage (*stopwords* en anglais)

Un exemple :

| | |
|---|---|
| Texte original | `While applying opacity to a form, should we use a decimal or a double value?`<br>`<p>I want to use a track-bar to change a form's opacity.</p>`<br><br>`<p>This is my code:</p>`<br><br>`<pre><code>decimal trans = trackBar1.Value / 5000;`<br>`this.Opacity = trans;`<br>`</code></pre>`<br><br>`<p>When I build the application, it gives the following error:</p>`<br><br>`<blockquote>`<br>`  <p>Cannot implicitly convert type <code>'decimal'</code> to <code>'double'</code>.</p>`<br>`</blockquote>`<br><br>`<p>I tried using <code>trans</code> and <code>double</code> but then the control doesn't work. This code worked fine in a past VB.NET project.</p>` |
| Texte « nettoyé » | appli opac form use decim doubl valu want use track bar chang form opac thi code decim tran trackbar valu thi opac tran build applic give follow error cannot implicitli convert type decim doubl tri use tran doubl control work thi code work fine past vb net project |

## 3. Analyse du corpus

Après n'avoir gardé que les mots qui nous intéressaient, nous décidons de faire une petite analyse des mots de notre corpus.

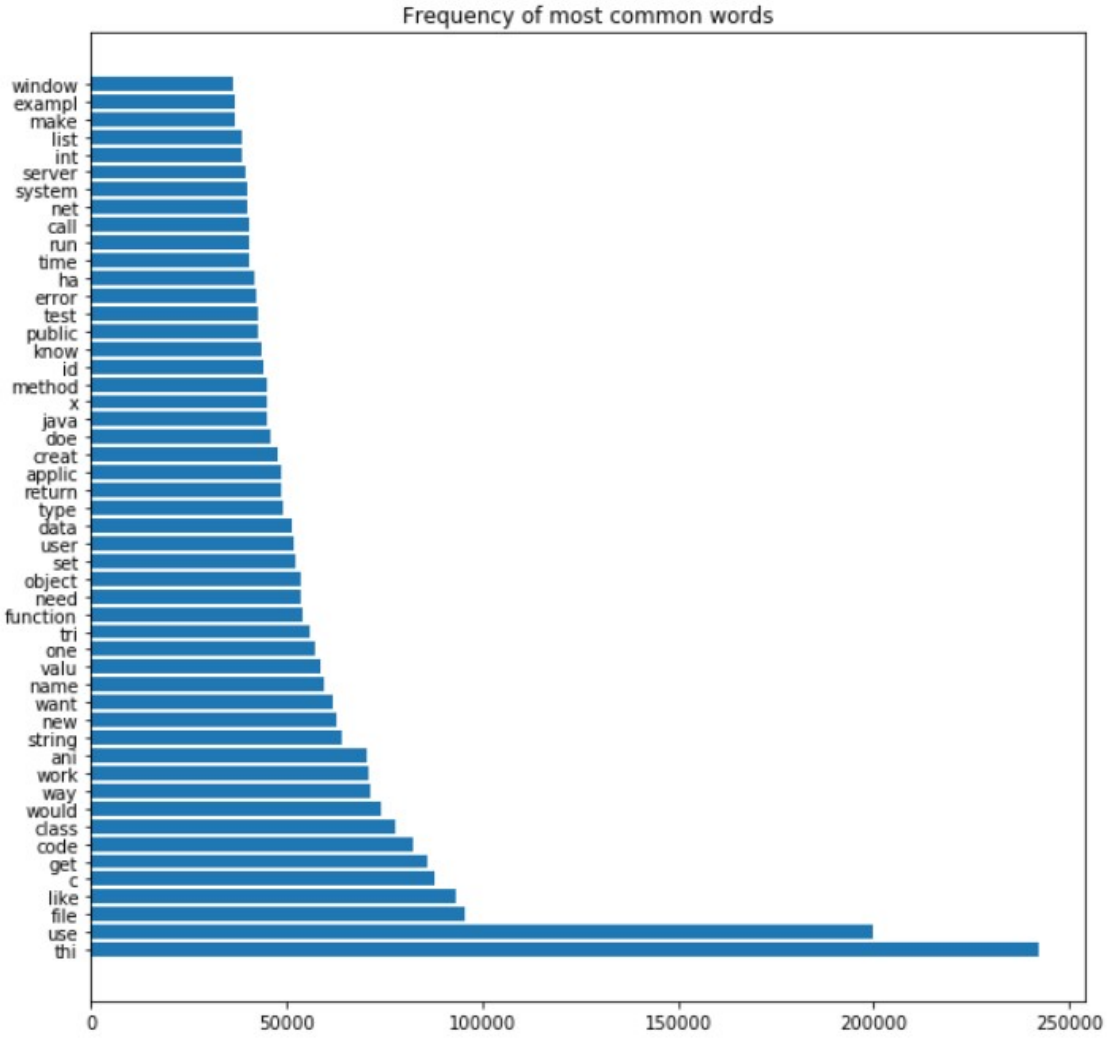
Nous remarquons tout d'abord que notre corpus est constitué de **209178 mots différents**. On ne pourra garder tous ces mots pour notre catégorisation sous peine d'engendrer des calculs beaucoup trop longs.

Nous allons nous intéresser aux mots qui reviennent le plus fréquemment puisque c'est les mots qui seront utilisés le plus fréquemment qu'on estime porter le plus d'informations pour notre taguage. Les mots comme : java, database, sql, server, network…

Nous avons donc essayé d'observer ces mots :

```
['thi' 'use' 'file' 'like' 'c' 'get' 'code' 'class' 'would' 'way' 'work'
 'ani' 'string' 'new' 'want' 'name' 'valu' 'one' 'tri' 'function' 'need'
 'object' 'set' 'user' 'data' 'type' 'return' 'applic' 'creat' 'doe' 'java'
 'x' 'method' 'id' 'know' 'public' 'test' 'error' 'ha' 'time' 'run' 'call'
 'net' 'system' 'server' 'int' 'list' 'make' 'exampl' 'window']
```

LES 50 MOTS LES PLUS FREQUENTS DE NOTRE CORPUS




DISTRIBUTION DES 50 MOTS LES PLUS UTILISEES DE NOTRE CORPUS

Pour savoir combien de mots il nous faudra retenir nous avons simplement observé ce tableau de mots en essayant de trouver une valeur optimale à laquelle nous arrêter :

```
['far' 'correct' 'standard' 'group' 'ask' 'folder' 'machin' 'thought'
 'assembl' 'locat' 'insid' 'block' 'last' 'context' 'must' 'miss' 'record'
 'statement' 'tell' 'place' 'someon' 'loop' 'via' 'comment' 'microsoft'
 'replac' 'featur' 'avail' 'ie' 'td' 'v' 'initi' 'std' 'height' 'either'
 'cach' 'practic' 'linux' 'note' 'approach' 'task' 'learn' 'password'
 'figur' 'background' 'non' 'caus' 'parent' 'consol' 'config']
```

LES 350 A 400 MOTS LES PLUS FREQUENTS DE NOTRE CORPUS

```
['gui' 'commun' 'play' 'regular' 'txt' 'combin' 'frame' 'exit' 'uniqu'
 'primari' 'track' 'variou' 'optim' 'deleg' 'purpos' 'regist' 'produc'
 'hit' 'past' 'reflect' 'vb' 'mac' 'author' 'layer' 'lang' 'split' 'later'
 'val' 'usual' 'obvious' 'cooki' 'rel' 'io' 'sun' 'mail' 'abstract' 'guid'
 'often' 'tutori' 'maven' 'mention' 'hi' 'queue' 'q' 'delphi' 'associ'
 'easili' 'nhibern' 'gcc' 'master']
```

LES 650 A 700 MOTS LES PLUS FREQUENTS DE NOTRE CORPUS

```
['cursor' 'cross' 'advantag' 'describ' 'crash' 'silverlight' 'bottom'
 'multi' 'margin' 'zero' 'graph' 'inner' 'leav' 'high' 'obviou' 'third'
 'xs' 'sp' 'cpu' 'hold' 'translat' 'assert' 'launch' 'xp' 'navig' 'advic'
 'unabl' 'outsid' 'embed' 'sound' 'accomplish' 'interact' 'parser' 'inject'
 'resiz' 'mous' 'ptr' 'focu' 'align' 'parti' 'bash' 'mock' 'uri' 'pair'
 'zip' 'recurs' 'math' 'typeof' 'evalu' 'graphic']
```

LES 950 A 1000 MOTS LES PLUS FREQUENTS DE NOTRE CORPUS

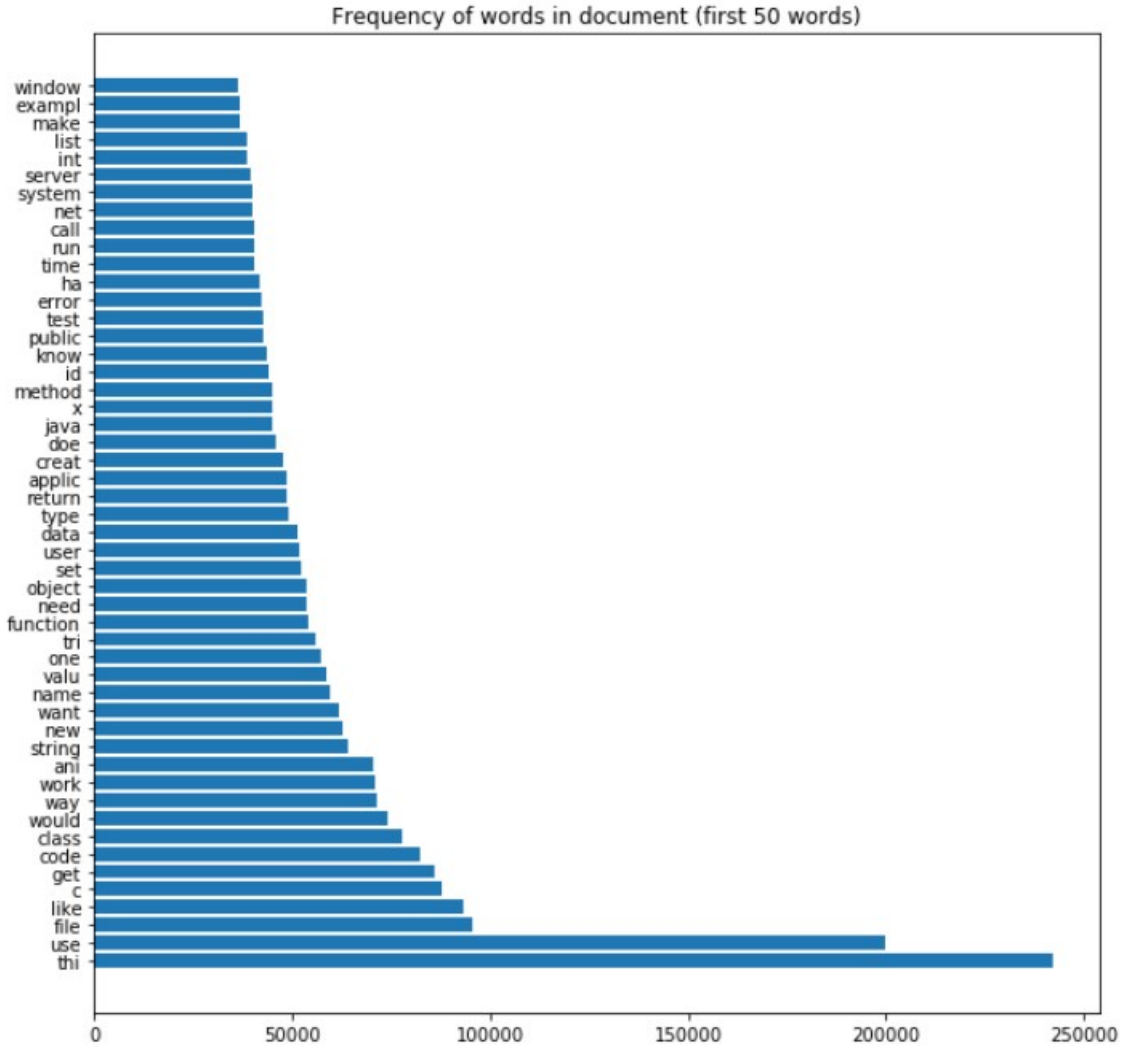
Nous avons décidé après ces quelques observations simples de ne garder que les 400 mots les plus fréquents pour construire notre dictionnaire. On garde un nombre petit pour pouvoir simplifier les calculs.

On va donc entrainer nos algorithmes sur un dataset avec 400 features ce qui est déjà très volumineux.

Nous décidons alors de créer notre tableau avec une représentation tf c'est-à-dire en ne prenant en compte que la fréquence des mots. On utilise pour ça la librairie CountVectorizer.

## 4. Représentation Tf-idf

Nous avons également essayé d'observer la fréquence à laquelle les mots apparaissaient dans nos documents la *document frequency.*




DISTRIBUTION DES 50 MOTS LES PLUS PRESENTS DANS LES DOCUMENTS DE NOTRE CORPUS

On se rend compte que les distributions entre fréquence par document et fréquence se ressemblent beaucoup. Ce pourquoi nous décidons de nous concentrer simplement sur la fréquence des mots sans essayer de la pondérer par l'inverse de la fréquence dans les documents.

Nous créons tout de même à cette étape une matrice de représentation tf-idf pour pouvoir également tester nos algorithmes dessus et pour pouvoir tester l'assomption que l'on a faite sur ces deux représentations.

## 2. Catégorisation des questions

Nous allons dans cette partie tester des méthodes de taguages de nos questions Stack OverFlow. Nous allons tout d'abord tester des méthodes de taguages non supervisées c'est-à-dire sans entrainer d'algorithmes sur le taguage déjà existant. Ensuite nous évaluerons les méthodes de taguage supervisées.

Il nous restera à choisir la méthode de taguage qui nous semblera la plus pertinente.

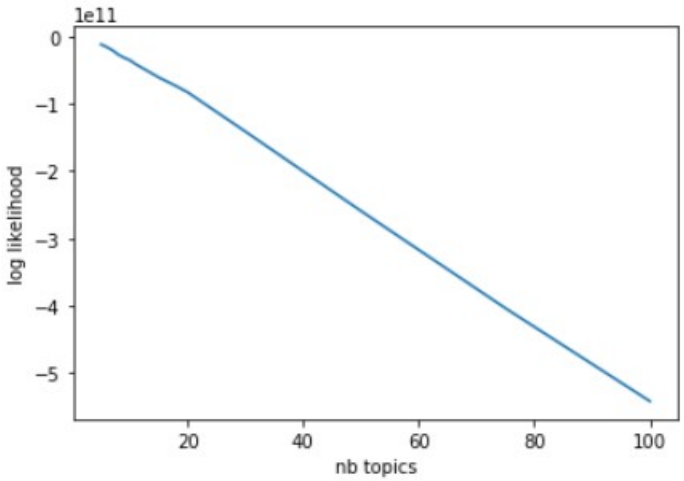
## 1. Apprentissage non supervisée

Pour l'apprentissage non supervisée nous allons réutiliser tout d'abord la matrice créé dans la partie 1 de type Bag of Words puis nous testerons les méthodes via la matrice tf-idf.

### a. Allocation Latente de Dirichlet

Nous allons utiliser l'algorithme appelé Allocation Latente de Dirichlet (LDA en anglais). Cet algorithme va nous permettre de trouver les n sujets les plus importants dans nos documents. Ces sujets sont décrits par la probabilité des mots de nos dictionnaires pour chacun des sujets.

Pour déterminer le nombre de sujets n nous allons utiliser une grille de validation (GridSearch). Le facteur de performance de cet algorithme est le maximum de vraisemblances.




LOGARITHME DE LA FONCTION DE VRAISEMBLANCE PAR NOMBRE DE SUJETS POUR L'ALGORITHME LDA

Comme nous cherchons le maximum, la courbe ci-dessus nous indique que le nombre n optimal de sujets est 5.

Nous allons essayer d'observer les 5 sujets prédits par cet algorithme. Ces 5 sujets sont prédits par la probabilité de chaque mot du corpus. Nous avons décidé, pour avoir une idée du sujet de regarder les dix mots les plus probables pour chaque sujet :

| | |
|---|---|
| Sujet 1 | 'thi' 'class' 'string' 'object' 'return' 'public' 'new' 'int' 'function' 'method' |
| Sujet 2 | 'file' 'java' 'error' 'system' 'server' 'run' 'window' 'thread' 'servic' 'script' |

| | |
|---|---|
| Sujet 3 | 'use' 'thi' 'would' 'like' 'ani' ' code' 'way' 'one' 'work' 'time' |
| Sujet 4 | 'id' 'tabl' 'name' 'thi' 'select' 'valu' 'key' 'data' 'sql' 'databas ' |
| Sujet 5 | 'user' 'thi' 'page' 'use' 'control ' 'html' 'form' 'http' 'view' 'net ' |

10 MOTS CLASSES LES PLUS PROBABLES POUR CHACUN DES 5 SUJETS PREDITS PAR LA LDA

Nous pouvons plus ou moins résumer maintenant chacun de ces 5 sujets :

1 – language orienté objet

2- réseaux

3- pas de sujets précis

4- database

5- front end

Ces sujets sont censés nous donner la base pour pouvoir décrire près de 200 000 documents ce qui est très peu en termes de précision des sujets.

On peut par ailleurs essayer d'observer si les sujets sont correctement prédits pour chacun des documents. L'algorithme LDA nous donne la probabilté de chaque sujet pour chacun des documents, on peut donc essayer d'observer sur quelques documents les sujets abordés.

Exemple 1 :

| | |
|---|---|
| Titre | Percentage width child element in absolutely positioned paren t on Internet Explorer 7 |
| Question | I have an absolutely positioned <code>div</code> containin g several children, one of which is a relatively positioned < code>div</code>. When I use a percentage-based width< /strong> on the child <code>div</code>, it collapses to '0' w idth on <a href="http://en.wikipedia.org/wiki/Internet_Explor er_7" rel="noreferrer">Internet Explorer 7</a>, but not on Firefox or Safari. <br> If I use pixel width, it works. If the pa rent is relatively positioned, the percentage width on the ch ild works. <br><ol><br><li>Is there something I'm missing here?</li><br><li>Is there an easy fix for this besides the pixel-based width on the<br>child?</li><br><li>Is there an area of the CSS specification that covers thi s?</li><br></ol> |

| | |
|---|---|
| Probabilités des sujets | [ 0.00534982 0.00529059 0.00543081 0.97859047 0.0053383 ] |

Nous avons ici une question qui porte plutôt sur du front end, html, css. On s'attend donc à ce que ce soit le sujet 5 qui est la plus grande probabilité or on voit que l'algorithme à classer ce sujet dans la catégorie database.

Exemple 2 :

| | |
|---|---|
| Titre | While applying opacity to a form, should we use a decimal or a double value? |
| Question | While applying opacity to a form, should we use a decimal or a double value?<br> I want to use a track-bar to change a form's opacity. <br> This is my code: <br><pre><code>decimal trans = trackBar1.Value / 5000;<br>this.Opacity = trans;<br></code></pre><br> When I build the application, it gives the following error : <br><blockquote><br> Cannot implicitly convert type <code>'decimal'</code> to <code>'double'</code>. <br></blockquote><br> I tried using <code>trans</code> and <code>double</code> b ut then the control doesn't work. This code worked fine in a past VB.NET project. |
| Probabilités des sujets | 0.53665479, 0.00585954, 0.20443982, 0.00580943, 0.2472364 |

On voit ici que les sujets abordées sont plutôt orienté objet et front end ce qui est un résultat pas excellent mais pas mauvais non plus étant donné que la question est plutôt orienté front end.

En conclusion si on prend 5 sujets, les sujets sont très peu précis pour décrire les documents et de plus ils peuvent ne pas prédire correctement chacune de nos questions.

Nous allons tout de même essayer de voir si en augmentant le nombre de sujets, on peut avoir des résultats un peu plus satisfaisant.

On va tester le nombre de sujets égal à 15.

Voici les différents sujets observés :

| Sujet | 10 mots les plus probables |
|---|---|
| 1 | 'thi' 'use' 'like' 'way' 'would' 'time' 'one' 'want' 'object' ' foo' |
| 2 | 'page' 'control' 'html' 'view' 'javascript' 'asp' 'jqueri' 'sty le' 'load'<br>'thi' |

| | |
|---|---|
| 3 | 'use' 'thi' 'java' 'like' 'ani' 'would' 'code' 'know' 'doe' 'work' |
| 4 | 'valu' 'id' 'tabl' 'select' 'name' 'key' 'thi' 'properti' 'queri' 'column' |
| 5 | 'file' 'line' 'script' 'div' 'thi' 'command' 'path' 'use' 'directori' 'width’ |
| 6 | 'user' 'form' 'php' 'app' 'button' 'custom' 'input' 'browser' 'click' 'action' |
| 7 | 'name' 'type' 'code' 'url' 'includ' 'post' 'output' 'compil' 'search' 'print' |
| 8 | 'server' 'data' 'web' 'applic' 'servic' 'connect' 'use' 'databas' 'client' 'net' |
| 9 | 'python' 'xml' 'model' 'import' 'templat' 'node' 'context' 'td' 'parent' 'pattern' |
| 10 | 'text' 'string' 'array' 'imag' 'size' 'format' 'convert' 'tag' 'byte' 'charact' |
| 11 | 'test' 'project' 'build' 'version' 'sourc' 'instal' 'develop' 'android' 'visual' 'tool' |
| 12 | 'http' 'com' 'request' 'org' 'memori' 'lib' 'www' 'rubi' 'microsoft' 'respons' |
| 13 | 'class' 'public' 'return' 'int' 'new' 'object' 'method' 'string' 'thi' 'function' |
| 14 | 'error' 'system' 'window' 'except' 'thi' 'messag' 'log' 'run' 'set' 'tri' |
| 15 | 'list' 'item' 'element' 'number' 'self' 'sort' 'address' 'count''collect' 'add' |

Sans vouloir préciser à quoi correspondent tous les sujets, on se rend compte que nos sujets restent très vagues. On va donc avoir du mal à décrire nos 200 000 questions avec la base de ses sujets quand bien même tous les sujets seraient correctement attribués aux questions.

Nous n’allons donc pas retenir cette méthode.

On avait préparé une matrice de représentation de nos documents de type tf-idf malgré que celle-ci on l’a remarqué n’apporte pas beaucoup plus d’informations que la matrice tf. Nous allons tout de même nous en servir pour tester un autre algorithme de prédiction de sujets.

### b. Negative Matrix Factorization

On ne peut pas utiliser l’algorithme LDA pour une représentation tf-idf. Nous allons plutôt nous servir de l’algorithme Negative Matrix Factorization (NMF). Cet algorithme fonctionne de la même manière que le précédent dans le sens où il va produire des sujets qui seront composés de la probabilité de chaque mot de notre dictionnaire et chaque document aura une probabilité assigné à chacun de ces sujets.

On choisit  de prendre 10 sujets pour voir si les résultats diffèrent beaucoup des résultats observés avec la LDA.

Voici les sujets :

| Sujet | 10 mots les plus probables |
|---|---|
| 1 | use thi would code like ani know t ime one way |
| 2 | valu int list function array retur n type thi item error |
| 3 | file directori line folder path op en command read xml project |
| 4 | tabl sql databas queri column sele ct id row data server |
| 5 | imag text html page div javascript jqueri button css form |
| 6 | class public object method static name properti new void type |
| 7 | string charact convert format publ ic text char new thi match |
| 8 | test unit run code write framework fail project build method |
| 9 | java org eclips thread librari jar util com apach applic |
| 10 | net applic asp web window server s ervic user control mvc |

On retrouve approximativement les mêmes sujets que ceux abordées avec une representation tf. Comme on a déjà fait l'observation que cette matrice tf-idf n'apportait pas beaucoup plus d'informations que la matric tf, cela ne nous étonne pas.

On arrive donc rapidement à la conclusion que cette méthode non plus n'est pas satisfaisante pour pouvoir taguer nos questions.

On a laissé ici de côté tous les tags fournis par le site qui est une véritable mine d'informations sur lesquelles certains algorithmes vont pouvoir apprendre. On va donc s'en servir et essayer d'avoir des résultats plus satisfaisants que dans cette partie.

## 2. Apprentissage supervisée

Nous allons donc nous servir des tags mais au préalable nous allons cleaner et étudier ces données qu'on a laissé de côté jusqu'à présent.

### a. Préparation des tags

Voici des exemples de tags que nous renvoient le site stackexchange.

| |
|---|
| <c#><winforms><type-conversion><decimal><opacity> |
| <html><css><css3><internet-explorer-7> |
| <c#><datetime><time><datediff><relative-time-s... |
| <c#><.net><datetime> |
| <javascript><html><browser><timezone><timezone... |

Après avoir cleaner ces tags on s'intéresse aux nombres totaux de tags différents : il y en a approximativement 8000.

On ne va pas s'intéresser à tous les tags d'abord parce que un trop grand nombre de tags poserait des problèmes à nos algorithmes lors de l'apprentissage : il nous faudrait un volume de données d'apprentissage beaucoup plus grand que nos 200 000 questions et les temps de calculs seraient énormes mais également parce que notre but ici est de proposer un taguage pour des utilisateurs novices : qui ne sont pas intéressés pas un taguage de leurs questions trop précis. Ces utilisateurs sont potentiellement uniquement intéressés par un taguage assez général de leurs questions. Il s'agit de montrer la marche à suivre et non pas de faire le travail à leurs places.

Nous avons donc décidé de ne retenir que les 50 tags les plus communs.

### b. Les algorithmes testés

Nous avons testé quelques algorithmes. Il faut comprendre ici que notre sortie (les tags) est une sortie multiple : il peut y avoir plusieurs tags pour une seule question et il n'y a pas de priorité entre les tags.

Notre première idée était d'entrainer une **forêt aléatoire** en prenant comme donnée d'entrainement chacun des tags comme une sortie pour chaque question (répéter autant qu'il y a de tags).

Données d'entrainement :

| | |
|---|---|
| Question 1 | Tag 1 de la question 1 |
| Question 1 | Tag 2 de la question 1 |
| Question 2 | Tag 1 de la question 2 |
| Question 3 | Tag 1 de la question 3 |
| Question 3 | Tag 2 de la question 3 |
| Question 3 | Tag 3 de la question 3 |

L'idée était ici d'utiliser le fait que les forêts aléatoires sont bâtis sur des arbres randomisés aux niveaux de leurs variables et au niveau de l'échantillon d'entrainement pour que l'algorithme apprennent les sorties multiples.

Après des recherches il apparait que les forêts aléatoires sont tout à fait capable de prédire des sorties multiples et c'est ce que nous avons utilisés par la suite.

Nous avons également entrainé un algorithme de **régression logistique**. Pour cet algorithme nous avons utilisés un outil qui s'appelle BinaryRelevance. Cet outil va nous permettre d'entrainer une régression logistique sur chacun des tags et d'ensuite combiner les résultats de chacune de ces 50 régressions.

Nous avons testé deux autres algorithmes qui supportent les sorties multiples et qui sont le ExtraTreesClassifier et l'algorithme des k voisins.

### c. Résultats des algorithmes

Pour évaluer les résultats nous allons utiliser l'outil accuracy_score qui permet simplement de compter le nombre d'erreurs commises par l'algorithme vis-à-vis du nombre de tags correctement prédits.

| Algorithme | Accuracy Score |
|---|---|
| Random Forest | 0.30 |
| ExtraTreesClassifier | 0.21 |
| KNeirestNeighbor | 0.17 |
| LogisticRegression | 0.27 |

On voit des scores assez bas, on verra cependant par la suite que toutes les erreurs ne sont pas à compter.

On va essayer d'observer les prédictions de l'algorithme avec le moins d'erreur le random forest. Pour cela nous allons observer les tags prédits en les comparants aux tags réels.

| Prédiction | Réalité |
|---|---|
| | `delphi` |
| `sql-server` | `sql-server` |
| | `asp.net-mvc, c#` |
| `mysql` | `mysql` |
| `java` | `database` |
| `python` | `python` |

On voit déjà sur ces premiers exemples différents types d'erreurs apparaitre : les erreurs de type mauvais tags et les erreurs de types aucun tag.

Nous allons observer plus précisément les erreurs de taguage de notre algorithme.

| Question (preprocessed) | réalité | prédiction |
|---|---|---|
| `calcul someon age c given datetim repres person birthday calcul age year` | `'asp.net-mvc', 'c#'` | |
| `determin user timezon ani standard way web server abl determin user timezon within web page perhap http header part user agent string` | `database` | java |
| `multipl submit button html form let say creat wizard html form one button goe back one goe forward sinc back button appear first markup press enter use button submit form exampl form put cursor thi field press enter input type text name field thi button submit input type submit name prev valu previou page thi button want submit input type submit name next valu next page form would like get decid button use submit form user press enter way press enter wizard move next page previou use tabindex thi` | `c` | |

| | | |
|---|---|---|
| page collect linq page collect linq given start index count | 'ruby', 'ruby-on-rails' | |
| get subclips aptana work newest releas subvers version subclips current avail aptana automat plugin manag doe work newest version subvers see subclips websit howev eclips ad new remot updat site updat manag tri instal told need mylyn much search found mylyn ad anoth new remot updat site updat manag tri instal told need org eclips ui equival look configur detail aptana look like built eclips doe anyon know way upgrad version eclips aptana built way get subclips work veri newest version subvers know thi necessarili program question hope ok sinc highli relev program experi | '.net', 'c#' | |
| flat file databas best practic around creat flat file databas structur php lot matur php flat file framework see attempt implement sql like queri syntax top purpos case would use databas point ani eleg trick get good perform featur small code overhead one would want take thi problem first place | 'mysql', 'php', 'sql' | 'mysql' |

On voit ici differents types d'erreurs: les tags vides, les tags faux et dans le dernier exemple un tag prédit juste sachant qu'on en attendait trois. Dans cet exemple précis, on va considérer que notre taguage est juste : on a réussi à prédire au moins un tag et celui-ci est juste.

Nous allons donc observer pour trois des algorithmes (les plus performants) les résultats vis-à-vis de ces erreurs.

| Tags | Vrai | Faux | Inexistant |
|---|---|---|---|
| Random Forest | 46% | 12% | 42% |
| Extra Trees | 33% | 4% | 62% |
| Logistic Regression | 44% | 15% | 40% |

### d. Amélioration – Optimisation

Dans un premier temps, on s'est posé la question de savoir si la longueur de nos questions avait un rapport avec la capacité à prédire ou non un tag. En effet, on peut s'attendre à ce que lorsqu'il n'y a pas beaucoup de mots dans la question, notre algorithme n'arrive pas à prédire correctement les tags associés.

Nous avons donc regardé le nombre de mots en moyenne dans la matrice tf pour observer une différence entre les questions avec de bons tags, aucun tag ou des tags faux.

| Tags | Vrai | Faux | Inexistant |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Moyenne des mots de la question | 43.9 | 40.6 | 46 |

Nous n'arrivons pas à la conclusion attendue : il y a un peu plus de mots pour les questions où nous ne sommes pas arrivé à prédire un tag.

Nous en concluons que la raison principale de nos erreurs est due à la sélection des mots. Nous avions dans la première partie simplement sélectionné les 400 mots les plus fréquents dans notre dictionnaire. Pour être un peu plus performant nous supposons qu'il faut choisir avec plus de précaution les mots qui constituent notre dictionnaire (à la main par exemple).

Pour notre algorithme final nous avons décidé d'utiliser trois  des algorithmes précédemment présentés : extra trees, random forest et logistic regression. Nous allons les utiliser d'une manière un peu spéciale qui est spécifique à notre problématique.

**Solution**: Lorsque nous voudrons proposer un taguage nous allons tout d'abord observer si l'algorithme extra trees (taux d'erreurs le plus faible) nous renvoie un tag si ce n'est pas le cas, nous allons observer si le random forest nous renvoie un tag, sinon nous utiliserons la régression logistique. De cette manière on diminue un peu le fait de ne retourner aucun tag et nous garderons une erreur la plus faible possible.

Nous avons testé cet algorithme et il nous donne les résultats suivants :

| Tags | Vrai | Faux | Inexistant |
|---|---|---|---|
| Notre algorithme | 54% | 16% | 30% |

Nous avons donc amélioré nos résultats de près de 10% au niveau des tags vrais ce qui est un très bon résultat. On a maintenant 54% des questions qui vont trouver un tag.

## Conclusion

Nous avons dans cette étude essayé de prédire des tags à partir d'un corpus de questions stackoverflows avec leurs titres. Nous avons tout d'abord testé des méthodes non supervisés qui se sont avérés être des méthodes peu précises pour prédire des tags. Nous nous sommes alors tourner vers un taguage supervisé qui lui apporte des résultats bien meilleurs.

Après avoir testé quelques algorithmes de classification supervisé à sorties multiples, nous avons essayé d'élaborer une méthode pour améliorer nos résultats. Compte tenu du problème en question qui est d'essayer d'aider les utilisateurs à taguer leurs questions nous avons décidé que ce que nous voulions éviter était principalement les tags faux et les retours nuls.

Nous avons alors décidé de faire tourner les algorithmes les uns après les autres en faisant priorité aux algorithmes qui affichaient le moins d'erreur. Nous nous contentons donc de récupérer le résultat d'un algorithme extra-trees, puis d'une régression logistique puis d'une forêt aléatoire.

Nous obtenons un résultat satisfaisant: plus de 50% des tags prédits sont bons, 16% sont faux et le reste n'est pas prédit. Il reste cependant de nombreuses améliorations sur lesquelles on pourra potentiellement travailler pour améliorer les résultats comme:

- choisir un dictionnaire plus pertinent en sélectionnant les mots à la main parmi les mots avec la plus grande occurrence pour obtenir de meilleurs features et du coup améliorer le résultat de nos algorithmes.
- tester l'augmentation de la taille de nos échantillons d'entrainement et de tests pour savoir si on peut avoir de meilleurs résultats avec un plus gros volume de données (on avait ici près de 200 000 questions ce qui est déjà très volumineux vis à vis de la puissance de calcul à disposition: 8Go de RAM)
- augmenter le nombre d'algorithmes en essayant de garder des algorithmes qui ont approximativement les mêmes résultats que le random forest ou mieux ce qui nous permettrait comme on l'a vu de combiner les résultats de manière intelligente et ainsi encore améliorer nos résultats.
- on pourrait également s'intéresser aux tags prédits faux: on a fait ici que compter les tags qui sont faux mais on pourrait essayer d'évaluer à quel point la prédiction est fausse: le tag "C" est une erreur moins importante si le tag à prédire était "C++" que si on avait à prédire le tag "database". On pourrait essayer d'introduire une notion de distance entre les tags pour mieux apprécier l'erreur que l'on commet réellement.
- Il est également intéressant d'observer à l'apport de l'information du titre. En effet, il y a certainement plus d'informations à récupérer dans le titre que dans le corps de la question. Peut-être qu'en donnant plus d'importances aux mots du titre qu'aux mots de la question, on arriverait à un résultat amélioré.

Ces améliorations n'ont pas été mise en place mais nous avons tout de même essayé d'apporter une solution satisfaisante à ce problème de taguage des questions stack overflows.