
# Catégorisez automatiquement des questions

# Table des matières


1. Introduction ..... 3
2. Visualisation des données ..... 3
3. Nettoyage – Analyse ..... 4
   3.1. Title & Body ..... 4
   3.1.1. Balises HTML ..... 4
   3.1.2. Caractères spéciaux ..... 5
   3.1.3. Convertir le texte en minuscules ..... 5
   3.1.4. Enlever les stopwords ..... 5
   3.1.5. Stemming ..... 5
   3.1.6. Résultat ..... 5
   3.2. Tags ..... 6
   3.2.1. Nettoyage ..... 6
   3.2.2. Cohérence entre les tags et la question posée (titre et corps) ..... 7
4. Words occurrence ..... 8
   4.1. Words occurrence - Tags ..... 8
   4.2. Words occurrence - Title & Body ..... 8
5. Réduction dimensionnelle ..... 9
6. Méthode non supervisée ..... 10
   6.1. Latent Dirichlet Allocation (LDA) ..... 11
   6.2. Non-Negative Matrix Factorization (NMF) ..... 11
   6.3. Méthode d'évaluation propre ..... 11
   6.4. Résultats ..... 11
   6.5. Modélisation ..... 13
7. Méthode supervisée ..... 14
   7.1. Comparaison des modèles ..... 14
   7.2. Optimisation ..... **Erreur ! Signet non défini.**
8. Modèle final ..... **Erreur ! Signet non défini.**
9. Evaluation ..... **Erreur ! Signet non défini.**
10. Remerciements ..... 16

# 1. Introduction

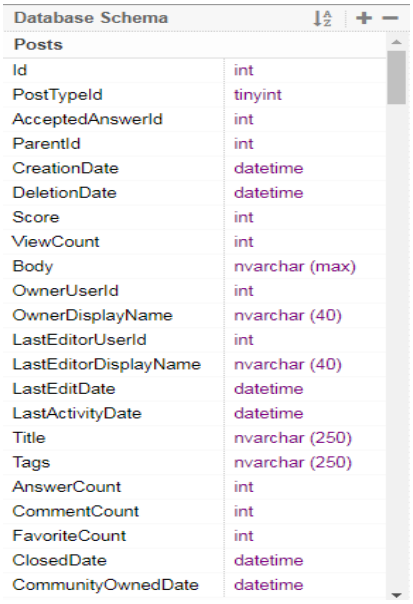
Poser une question sur Stack Overflow est très simple, plus dur est de trouver les bons tags permettant de catégoriser votre question et ainsi de pouvoir la retrouver plus facilement par la suite. Le but de ce projet est de construire un système de suggestion de tags utilisant le strict minimum en entrée, à savoir le titre et le corps de la question que vous posez.

L'idée est donc de récupérer dans un premier temps des données réelles, disponibles sur Stack Exchange, de les analyser, de construire un modèle permettant de générer des tags, et enfin de les inclure dans une IHM dédiée que l'on mettra à disposition.

# 2. Visualisation des données

Tout d'abord la récupération des données, elle se fait sur le site : https://data.stackexchange.com/stackoverflow/query/new

De nombreux champs sont disponibles comme on peut le voir ci-contre :



Database Schema

| Posts | |
|---|---|
| Id | int |
| PostTypeId | tinyint |
| AcceptedAnswerId | int |
| ParentId | int |
| CreationDate | datetime |
| DeletionDate | datetime |
| Score | int |
| ViewCount | int |
| Body | nvarchar (max) |
| OwnerUserId | int |
| OwnerDisplayName | nvarchar (40) |
| LastEditorUserId | int |
| LastEditorDisplayName | nvarchar (40) |
| LastEditDate | datetime |
| LastActivityDate | datetime |
| Title | nvarchar (250) |
| Tags | nvarchar (250) |
| AnswerCount | int |
| CommentCount | int |
| FavoriteCount | int |
| ClosedDate | datetime |
| CommunityOwnedDate | datetime |

Nous ne prendrons que ce qui nous intéresse pour le projet, c'est-à-dire l'identifiant, le corps de la question, le titre et les tags associés en nous restreignant aux seules questions :

SELECT Id, Body, Title, Tags FROM posts WHERE PosttypeId = 1 and Id < 500000

| | Id | Body | Title | Tags |
|---|---|---|---|---|
| 0 | 4 | <p>I want to use a track-bar to change a form'... | While applying opacity to a form, should we us... | <c#><winforms><type-conversion><decimal><opacity> |
| 1 | 6 | <p>I have an absolutely positioned <code>div</... | Percentage width child element in absolutely p... | <html><css><css3><internet-explorer-7> |
| 2 | 9 | <p>Given a <code>DateTime</code> representing ... | How do I calculate someone's age in C#? | <c#><.net><datetime> |
| 3 | 11 | <p>Given a specific <code>DateTime</code> valu... | Calculate relative time in C# | <c#><datetime><time><datediff><relative-time-s... |
| 4 | 13 | <p>Is there any standard way for a Web Server ... | Determine a User's Timezone | <javascript><html><browser><timezone><timezone... |

La contrainte sur l'id est liée au serveur, la récupération des données se fera donc en plusieurs fois.

# 3. Nettoyage – Analyse

Dans cette partie nous allons voir comment transformer notre texte de départ pour ne garder que ce semble être indispensable à sa caractérisation. Le but étant au final de se retrouver avec une matrice uniquement numérique pour que les statistiques fassent leur travail derrière.

On va commencer par normaliser les données, se débarrasser de tout ce qui ne semble d'aucune utilité, balises, mots de liaison, caractères spéciaux… Puis on va opérer une tokenisation

## 3.1. Title & Body

Regardons un exemple de donnée qu'on recevra en entrée de notre IHM :

*print(dataraw.Title[0])*

*print(dataraw.Body[0])*

*While applying opacity to a form, should we use a decimal or a double value?*

* I want to use a track-bar to change a form's opacity. *

* This is my code: *

*<pre><code>decimal trans = trackBar1.Value / 5000;*

*this.Opacity = trans;*

*</code></pre>*

* When I build the application, it gives the following error: *

*<blockquote>*

* Cannot implicitly convert type <code>'decimal'</code> to <code>'double'</code>. *

*</blockquote>*

* 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. *

D'après l'exemple ci-dessus on voit qu'il va falloir :

- Enlever les balises HTML
- Enlever les caractères spéciaux
- Convertir le texte en minuscules
- Enlever les stopwords
- Extraire la racine des mots

Voyons cela de plus près.

### 3.1.1. Balises HTML

Cette partie est classique et déjà développée, je vais donc utiliser la librairie **BeautifulSoup** qui me renvoie un text sans les balises HTML.

### 3.1.2. Caractères spéciaux

Alors là attention à ce qu'on met dans les caractères spéciaux car on pourrait enlever de l'information importante comme par exemple le # de c#. En effet, si on enlève le #, le c tout seul soit ne restera pas non en tant que lettre seule, soit fusionnera avec le c de c++ qui se retrouvera seul aussi.

Je crois aussi qu'il faut garder les chiffres qui peuvent donner des indication d'année ou de numéro de version important pour les tags, par exemple la version de SQL Server utilisée, de .Net, ou de n'importe quel logiciel ou langage informatique.

Je vais donc utiliser simplement une expression régulière pour ne garder que les caractères alphanumériques, le + et le #.

### 3.1.3. Convertir le texte en minuscules

De manière évidente, tout le texte doit avoir la même casse pour ne pas générer de doublons de mots.

A ce moment on peut aussi changer notre variable texte en un tableau de mots en utilisant l'espace comme séparateur.

### 3.1.4. Enlever les stopwords

Les stopwords sont les mots courant n'apportant pas d'information, comme les articles déterminants par exemple. La librairie NLTK permet de les enlever facilement.

*from nltk.corpus import stopwords*

*mystops = set(stopwords.words("english"))*

### 3.1.5. Stemming

Toujours dans l'idée de conserver au final le moins de dimensions possible, on essaie de grouper encore les mots en ne gardant que leur racine, encore avec la librairie NLTK :

*nltk.PorterStemmer()*

*meaningful_words = [porter.stem(w) for w in meaningful_words]*

### 3.1.6. Résultat

Voilà ce donne notre variable après nettoyage des éléments inutiles :

> *appli opac form use decim doubl valu want use track bar chang form opac code decim tran trackbar1 valu 5000 opac tran build applic give follow error cannot implicitli convert type decim doubl tri use tran doubl control work code work fine past vb net project*

Je pense qu'on va pouvoir utiliser nos données en tant que bag of words à partir de ce résultat.

Il n'est pas utile ici de considérer des groupes de mots (n-gram) pour avoir une notion de contexte car on sait déjà qu'on est dans un cadre informatique, avec des questions techniques. Il suffit juste de faire ressortir des mots clés de la question posée, et comme on va le voir par la suite les tags font déjà très

souvent partie du texte de l'utilisateur, la réponse est dans la question comme on dit. On peut donc se contenter d'un bag of words duquel on fera ressortir les tags appropriés.

## 3.2. Tags

### 3.2.1. Nettoyage

Dans un premier temps on peut :

- Enlever les balises '<' et '>'
- Garder les caractères spéciaux car ils font partie des tags
- Convertir le texte en minuscules

Pour les stop words je voudrais vérifier l'intérêt, j'affiche donc la liste de ceux qu'on trouve dans les tags :

- Against, any, between, can, d, each, having, ll, m out, this, was, where

Affichons les tags concernés pour voir :

```
<perl><hash><iteration><each>
<php><sql><where>
<d>
<d><popularity>
<editor><d>
<input><d><tango>
<java><constructor><methods><call><this>
<serial-port><microcontroller><can><can-bus>
<windows><installation><d>
<java><c++><c><d>
<linux><d><powerpc><tango>
<sql><sql-server-2005><foreign-keys><between>
<arrays><d>
<jquery><radio-button><this>
<d>
<mysql><full-text-search><against>
<perl><hash><each><while-loop>
<c#><generics><attributes><constraints><where>
<c#><.net><casting><ref><out>
```

Effectivement il y a des stop words dans les tags qu'on peut enlever. Peut-être que « Where » serait utile à garder si par exemple il y des questions concernant la méthode ou le mot clé SQL « Where ». On pourra affiner cela par la suite, continuons le nettoyage.

On peut donc enlever les stopwords, mais attention il y a des occurrences qui se retrouverons sans tag et qu'on va devoir faire disparaitre de nos tests, on peut considérer que ce sont des outliers, et comme le nombre et la nature des échantillons à notre disposition sont largement suffisants il n'y a aucun souci à se débarrasser de quelques données pour entrainer notre modèle.

```
Il y a 15090 tags différents, voilà les 20 premiers classés alphabétiquement :

['.bash-profile',
 '.doc',
 '.emf',
 '.htaccess',
 '.htpasswd',
 '.net',
 '.net-1.0',
 '.net-1.1',
 '.net-2.0',
 '.net-3.0',
 '.net-3.5',
 '.net-4.0',
 '.net-assembly',
 '.net-attributes',
 '.net-client-profile',
 '.net-core',
 '.net-framework-source',
 '.net-framework-version',
 '.net-internals',
 '.net-micro-framework']
```

### 3.2.2. Cohérence entre les tags et la question posée (titre et corps)

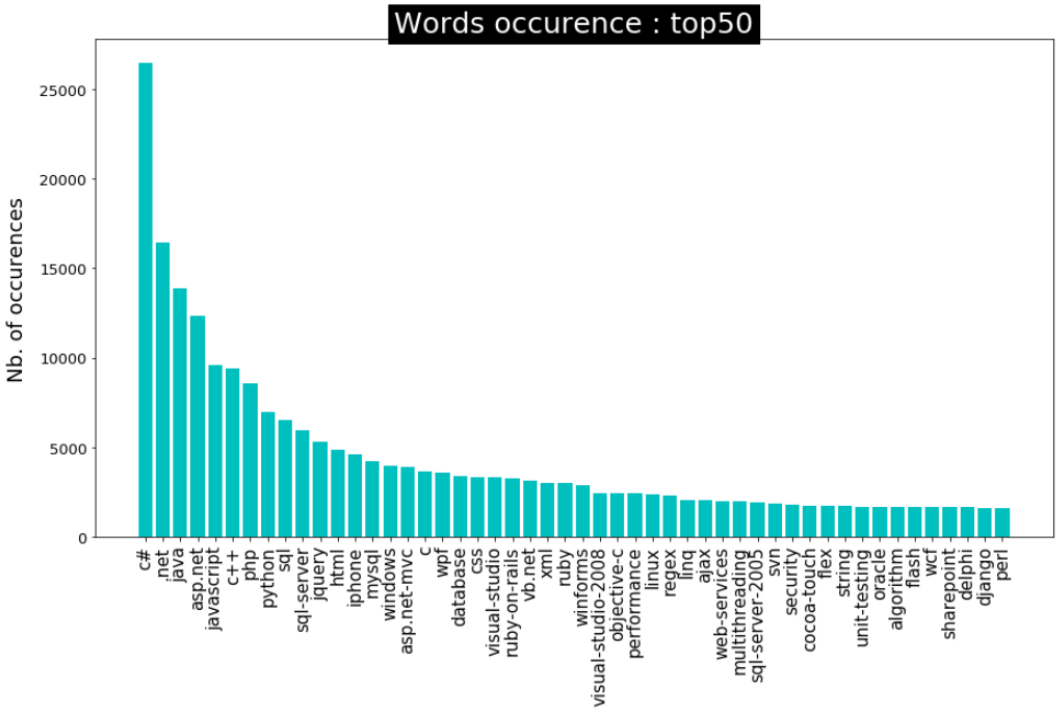
A l'aide d'une simple boucle, je vais maintenant regarder si on retrouve les tags dans le texte (corps + titre).

En analysant la répartition des tags dans les questions on voit que pour environ 90% des cas, au moins un des tags se retrouve dans la question, ce qui permet d'être optimiste quant à la possibilité qu'un algorithme obtienne de bons résultats sur ce jeu de données.

On constate même que dans presque un tiers des cas, tous les tags se retrouvent dans le texte, ce qui est encore plus motivant.

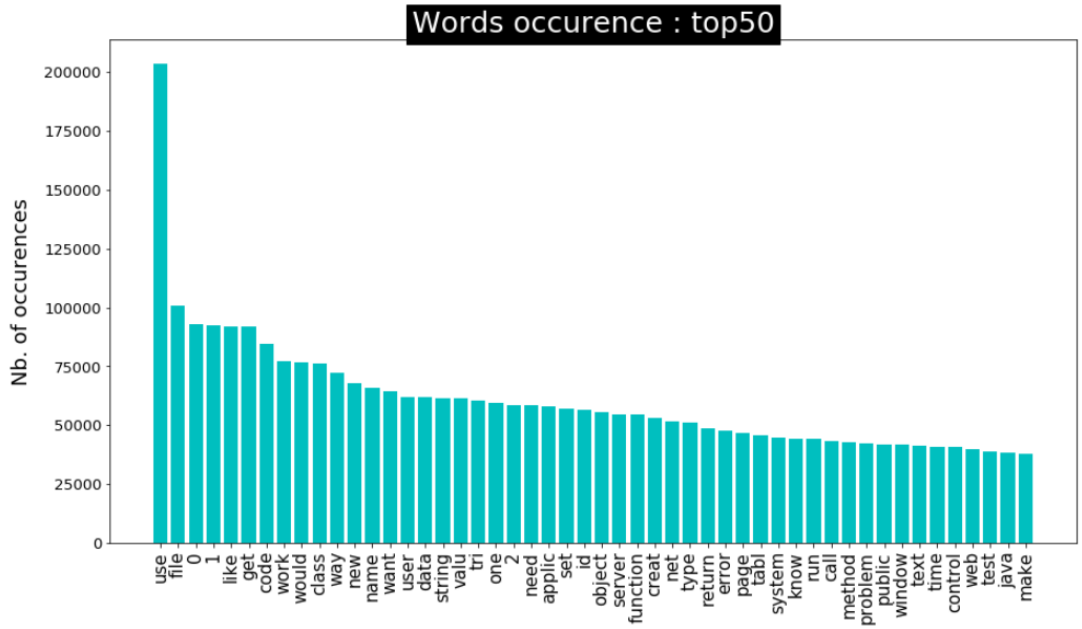
# 4. Words occurrence

## 4.1. Words occurrence - Tags




Le point important maintenant est qu'il va falloir choisir un nombre de tags à garder pour notre modèle supervisé. En effet, on va faire de la classification multi labels, et la liste des labels possibles doit être connue à l'avance.

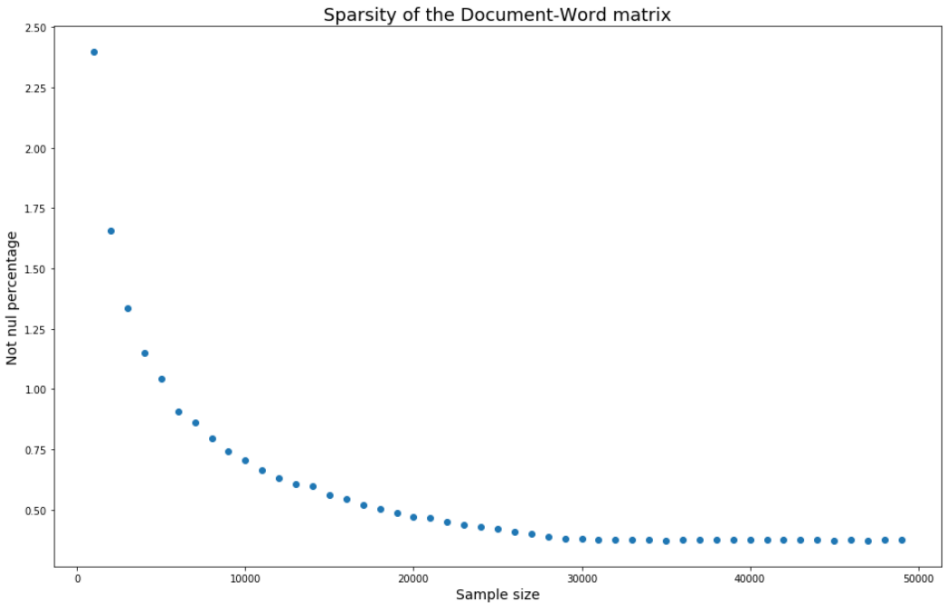
## 4.2. Words occurrence - Title & Body



On voit que les mots les plus utilisés ne sont pas forcément pertinents, on pourrait faire un nettoyage à ce niveau ici mais le paramètre max_df de la vectorisation que l'on va utiliser par la suite devrait s'en charger.

## 5. Réduction dimensionnelle

La transformation du texte en vecteurs numériques introduit un nombre de dimensions très important pour au final avoir une matrice avec beaucoup de zéros, sparse. Le graphique suivant montre l'évolution du taux de remplissage en fonction du nombre d'échantillons.



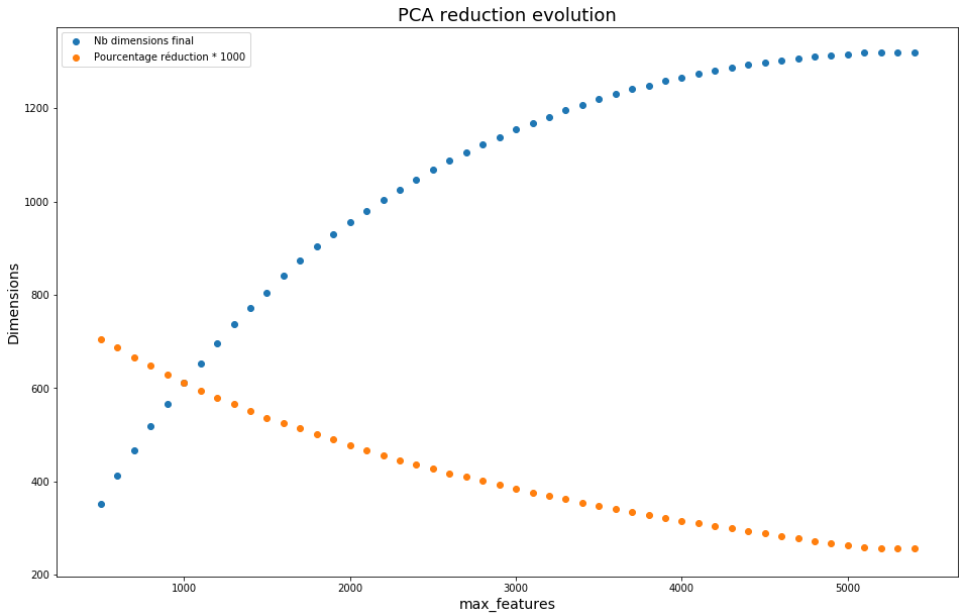

Très vite le pourcentage de valeurs non nulles passe à moins de 1%, pour se stabiliser 0.37%.

La courbe est la même si on fait varier max_feature, le paramètre de CountVectorizer. En allant de 1000 à 20000 max features, la densité minimale est atteinte autour de 10000.

Avec une telle quantité de zéros dans notre matrice on peut se dire qu'une réduction dimensionnelle va nous être bien utile. Essayons donc une analyse en composante principale :

```
PCA done in 611.445s.
Original number of features: 7887
Reduced number of features: 1162 pour 95% de variance
```

Le graphique suivant montre l'évolution du nombre de dimensions final en fonction du nombre de features max utilisé, ainsi que l'évolution du pourcentage de réduction dimensionnel pour un échantillon de 10000 questions. On voit que la PCA tend vers un nombre de dimension maximal d'environ 1300, qui correspond à une réduction d'environ 25% du nombre de dimensions.




Les mêmes tests avec TSVD donne exactement les même résultats.

## 6. Méthode non supervisée

A l'aide de méthodes non supervisée nous allons maintenant extraire de nos données des catégories. En fait on va créer des clusters de questions, mais combien ? Afin d'estimer l'efficacité du nombre de clusters choisi je vais créer une méthode permettant de donner une note à un estimateur donné. Cette note va dépendre du nombre de clusters, du nombre de mots représentant chaque cluster et de la liste complète des tags de notre jeu de données.

Pour chacun des sujets défini par le modèle, on va regarder ses mots clés, les mots les plus représentatifs du topic et noter si ces mots clés font partie de la liste des tags existants. Plus un cluster est représenté par une liste de tags, plus il est légitime en tant que groupe de questions. Le pourcentage total de mots clés étant aussi des tags est alors notre indicateurs de l'intérêt à porter à la clusterisation étudiée.

Attention les mots issus du texte ont été racinisés (stemming) donc on ne va peut-être pas les retrouver dans les tags dont on a gardé les mots originaux. Si la méthode ne donne pas de résultat il suffira de créer une liste de tags racinisés pour comparer.

On peut alors faire varier à volonté :

1. Les paramètres d'entrée de la vectorisation :
   - Max_df
   - Min_df
   - Max_features
2. Les paramètres de la modélisation :
   - N_components
   - Loss (pour la NMF)
3. Et Le paramètre d'affichage, le nombre de top words représentant un cluster.

## 6.1. Latent Dirichlet Allocation (LDA)

Très adapté aux matrices éparses comme ici pour de l'analyse de texte, la LDA va s'appliquer sur la matrice terme-fréquence.

CountVectorizer + LatentDirichletAllocation

## 6.2. Non-Negative Matrix Factorization (NMF)

On utilise maintenant le tf-idf qui prends en compte l'apparition d'un mot relativement à ses apparition dans les autres occurrences.

TfidfVectorizer + NMF

## 6.3. Méthode d'évaluation propre

Pour chacun des sujets que l'on va récupérer de notre modèle, une liste de mots clés, classés par ordre d'importance peut être récupérée.

On peut ensuite comparer cette liste à la liste des tags existants, et noter en fonction du nombre de mots se retrouvant dans les deux listes.

## 6.4. Résultats

Nos deux modèles, LDA et NMF peuvent être comparés à travers certains paramètres communs entre les deux vectorizers :

P_min : 5, 10, 20

P_max : 0.5, 0.8, 0.9

Max_features :1000, 10000, 100000

Et pour NMF on peut aussi tester différentes fonctions de perte: frobenius et kullback-leibler

Commençons avec un nombre 15 topics et 10 mots clés, puis avec 50 topics et 5 mots clés. Ci-dessous un exemple des résultats obtenus :

```
MIN
Note=0.72 (179 matches sur 250 possibles - LDA max=0.8 - min=5 - feat=10000)
Note=0.75 (187 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=10000)
Note=0.72 (180 matches sur 250 possibles - LDA max=0.8 - min=10 - feat=10000)
Note=0.76 (191 matches sur 250 possibles - NMF max=0.8 - min=10 - feat=10000)
Note=0.72 (181 matches sur 250 possibles - LDA max=0.8 - min=20 - feat=10000)
Note=0.77 (193 matches sur 250 possibles - NMF max=0.8 - min=20 - feat=10000)
MAX
Note=0.73 (183 matches sur 250 possibles - LDA max=0.5 - min=5 - feat=10000)
Note=0.79 (197 matches sur 250 possibles - NMF max=0.5 - min=5 - feat=10000)
Note=0.72 (179 matches sur 250 possibles - LDA max=0.8 - min=5 - feat=10000)
Note=0.75 (187 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=10000)
Note=0.72 (179 matches sur 250 possibles - LDA max=0.9 - min=5 - feat=10000)
Note=0.75 (187 matches sur 250 possibles - NMF max=0.9 - min=5 - feat=10000)
FEATURES
Note=0.74 (186 matches sur 250 possibles - LDA max=0.8 - min=5 - feat=1000)
Note=0.78 (195 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=1000)
Note=0.72 (179 matches sur 250 possibles - LDA max=0.8 - min=5 - feat=10000)
Note=0.75 (187 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=10000)
Note=0.74 (184 matches sur 250 possibles - LDA max=0.8 - min=5 - feat=100000)
Note=0.78 (196 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=100000)
NMF loss
Note=0.75 (187 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=10000)
Note=0.73 (183 matches sur 250 possibles - NMF max=0.8 - min=5 - feat=10000)
done in 8891.444s.
```

On a alors un modèle optimal pour 50 topics et 5mot clés. La fonction de perte par défaut (frobenius) m'a toujours donné un meilleur résultat. On peut continuer d'optimiser notre modèle en utilisant pour finir un GridSearch et jouer avec les autres paramètres.

NMF(max_df=0.5, min_df=5, max_features=10000, loss='frobenius')

Voyons les 20 premiers sujets qui ressortent:

```
Topic 0  (4 matchs) : code like way work know
Topic 1  (3 matchs) : valu properti type return set
Topic 2  (3 matchs) : tabl column databas row sql
Topic 3  (4 matchs) : button click event jqueri dialog
Topic 4  (4 matchs) : file directori folder upload path
Topic 5  (5 matchs) : asp net mvc web site
Topic 6  (4 matchs) : string charact public convert return
Topic 7  (2 matchs) : class public interfac privat properti
Topic 8  (5 matchs) : project visual studio 2008 build
Topic 9  (5 matchs) : server sql connect error 2005
Topic 10  (2 matchs) : imag png jpg background img
Topic 11  (4 matchs) : servic web wcf client soap
Topic 12  (4 matchs) : test unit run mock nunit
Topic 13  (3 matchs) : net vb framework ado entiti
Topic 14  (5 matchs) : field html valid input type
Topic 15  (4 matchs) : object properti type serial collect
Topic 16  (5 matchs) : list sort collect item element
Topic 17  (4 matchs) : page load html aspx web
Topic 18  (3 matchs) : java eclips jar org librari
Topic 19  (5 matchs) : window com http url open
Topic 20  (4 matchs) : control wpf custom action usercontrol
```

Les sujets trouvés sont très intéressants, on distingue déjà des catégories types, comme les bases de données (topic 3), de l'IHM (topic 4), de la gestion d'images (topic 10), des tests (topic 12)…

Et la répartition des premières lignes de nos données :

```
doc 0, topic 23, convert progress databas charact datatyp c# string...
doc 1, topic 49, way trigger server event jqueri asp net mvc situat...
doc 2, topic 14, dojo xhtml valid possibl make dojo javascript widg...
doc 3, topic 26, custom net authent membership profil provid portle...
doc 4, topic 49, determin total number initprogress event come prel...
doc 5, topic 18, use font resourc graphic languag java 1 4 follow i...
doc 6, topic 4, write output place consol new python write script ...
doc 7, topic 13, free code coverag tool net person project need fre...
doc 8, topic 0, possibl convert vba c# modul block code vba run ac...
doc 9, topic 47, share type wcf use vs 2008 work tri share dto data...
```

Et pour boucler la boucle on peut aussi regarder la répartition des questions dans les différents sujets en comptant la population des topics :

| | Topic Num | Num Documents |
|---|---|---|
| **0** | 0 | 30905 |
| **1** | 4 | 8305 |
| **2** | 1 | 7259 |
| **3** | 2 | 6794 |
| **4** | 6 | 5455 |
| **5** | 7 | 5326 |
| **6** | 5 | 4658 |

…

| | | |
|---|---|---|
| **44** | 36 | 2091 |
| **45** | 49 | 2081 |
| **46** | 39 | 2003 |
| **47** | 46 | 1726 |
| **48** | 44 | 1681 |
| **49** | 47 | 1581 |

On remarque qu'un sujet se détache et accapare la grande majorité des questions. En même temps les topics les moins peuplés restent assez remplis pour être significatifs, ce qui est bon signe quant à la pertinence du découpage.

## 6.5. Modélisation

Maintenant que les données sont reliées à un topic, ce dernier peut devenir notre target et il est aisé de créer un modèle qui assigne un topic à une question :

Utilisons un LabelEncoder car nous sommes maintenant dans une étude de classification simple, on encode notre nouvelle target puis on crée un jeu de test et un jeu d'entrainement. On vectorize à nouveau nos données puis on applique un GridSearch avec par exemple un modèle à vaste marge qui fonctionne plutôt bien normalement avec ce genre de données nombreuse et plutôt éparses.

On peut se dire qu'on a catégorisé nos données en clusters significatifs, ce qui est parfait lorsqu'on fait par exemple de la segmentation clientèle. L'inconvénient de cette méthode est qu'on a un nombre de catégories fixé à l'avance, ce qui est peu évolutif et ne laisse plus trop de choix à l'utilisateur dans la liste des tags qu'il peut accoler à sa question, et pire, la liste des tags proposés pour un topic donné sera toujours la même, à moins de randomiser sur un groupe de mots clés associés au topic plus large que le nombre de tags demandés mais bon, je pense qu'on peut faire mieux.

Par la suite nous allons voir ce que peut donner une approche purement supervisée.

# 7. Méthode supervisée

Dans le cas des méthodes supervisées nous allons utiliser des classifications multilabel. Ainsi, la target, c'est-à-dire l'ensemble des tags possibles doit bien être identifiée (impossible d'en ajouter une fois le model entrainé). Il est donc important de déterminer un nombre de tags maximum n à utiliser, on sélectionne alors les n premiers tags dans la liste des tags les plus utilisés.

La target peut ainsi être binarisée par un MultiLabelBinarizer intialisé avec la liste des n tags les plus utilisés.

Maintenant que les labels de la cible sont connus, il faut nettoyer les données afin de ne garder que les tags correspondants. Evidemment certaines lignes se retrouvent alors sans tag, dépendant de n le nombre de labels choisis. Le but étant de toujours proposer des tags on peut se débarrasser de ces lignes dans le jeu d'entrainement, cela n'est pas un problème si leur pourcentage est faible (moins de 5% maximum me semble raisonnable).

Préparons maintenant un Pipeline avec :

- CountVectorizer
- TfidfTransformer
- OneVsRestClassifier

Ce Pipeline associé à un GridSearch va permettre de comparer différentes possibilités de préparation des données, on peut ainsi tester plusieurs paramétrages de CountVectorizer (min_df, max_df, max_features, ngram_range), et de TfidfTransformer (use_idf, norm).

Le tout étant de plus soumis à une cross validation.

On utilise la méthode du One vs Rest (OneVsRestClassifier) pour sélectionner les meilleurs classifieurs binaires pour chacun des labels de la target (chaque label est séparé au mieux de l'union des labels restants).

## 7.1. Comparaison des modèles

Comparons les différents algorithmes testés selon plusieurs aspects :la durée du calcul, la taille du modèle sauvegardé et le score. Mais l'accuracy telle qu'elle existe ne tient pas compte du fait que certaines occurrences possèdent moins de tags que ce qui est demandé, la note alors va être minorée. Pour corriger cela j'introduis un scoring personnel qui met la note maximal dans le cas où justement un seul tag existe et la prédiction contient ce tag parmi les 5 meilleurs proposés, peu importe les 4 autres tags.

Ne pouvant me permettre de faire des dizaines de tests avec 150k de questions et une liste de 1000 tags, je lance une série de tests avec moins de dimensions pour voir l'évolution des différents algorithmes. Il faut aussi penser à poster le tout sur Heroku, et donc je vais m'en tenir à un modèle de

moins de 100Mo. Je voudrais aussi pouvoir rejouer mon modèle plusieurs fois de façon à optimiser ses hyper paramètres.

Donc rapidement je m'aperçoit que je vais devoir abandonner les ExtraTrees et les RandomForest car leur taille est fonction de la taille de l'échantillon, et qu'ils font plus de 100Mo dès 10k de questions.

Ensuite je vais éliminer le GradientBoosting et l'AdaBoostClassifier car leur temps d'exécution dépassent plusieurs heures avec aussi 10k de questions.

Pour un échantillon de 10k questions, et une liste de 40 tags possibles :

| | **Temps de calcul (s)** | **Taille du modèle** | **Score (Accuracy)** |
|---|---|---|---|
| RandomForestClassifier | 360 | 15Mo | 0,351 |
| LinearSVC | 2650 | 2Mo | 0,374 |
| ExtraTreesClassifier | 660 | 13Mo | 0,295 |
| SGDClassifier | 60 | 2Mo | 0,352 |
| AdaBoostClassifier | 6600 | 1.8Mo | 0,296 |
| GradientBoostingClassifier | 62000 | 6Mo | 0,321 |
| ExtraTreesClassifier | 330 | 32Mo | 0,275 |

Au passage j'ai entrainé un SVC avec 40k d'échantillons pour comparer la tokenisation en unigram et en bigram :

Unigram : accuracy=0.194 en 700s pour 34Mo

Bigram : accuracy=0.205 en 1000s pour 270Mo

La taille étant importante pour moi étant donnée la mise en ligne, et le gain faible je vais rester en unigram. On pouvait se douter, maintenant on a la preuve qu'ici la prise en compte du contexte n'apporte pas grand-chose.

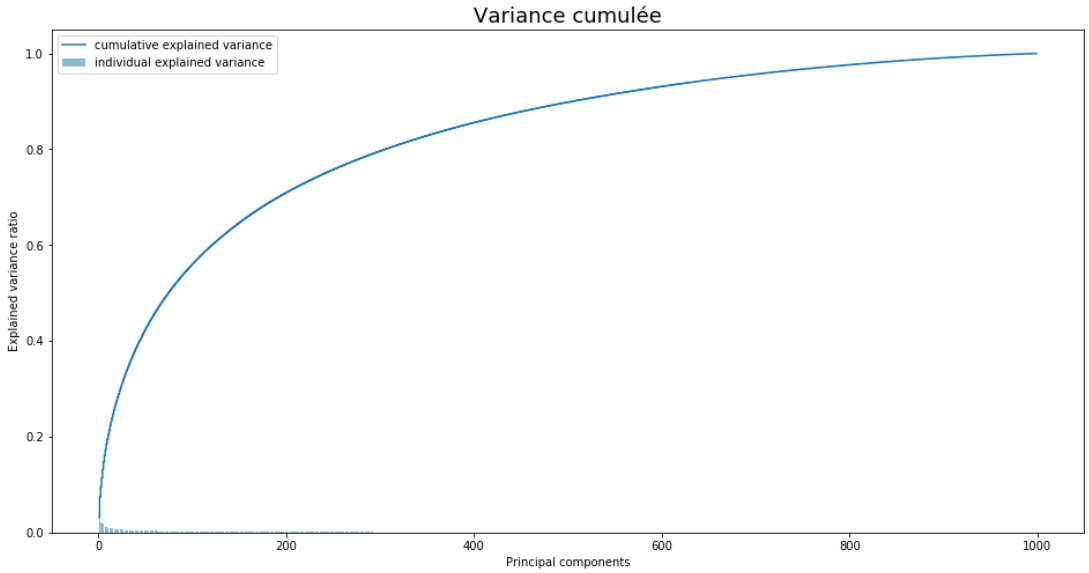
Pour le projet il me fallait un algorithme qui ne soit ni trop gourmand en temps d'entrainement, ni trop gourmand en taille de fichier sauvegardé mais avec quand même un score raisonnable. J'ai donc choisi un modèle à vaste marge, le LinearSVC.

A la suite de quoi j'ai cherché à optimiser ses paramètres à l'aide d'un GridSearch.

## 7.2. Réduction dimensionnelle

PCA et Truncated SVD donnent les même résultats, si ce n'est que PCA est plus rapide :

```
TSVD done in 99.539s.
Nombre de dimensions original : 1000
506  dimensions pour 90 % de variance
```




Au passage le même test avec la matrice TF-IDF confirme ce qu'intuitivement on pouvait imaginer : les 90% d'explication de variance nécessitent 721 dimensions contre 506 ici avec la matrice TF. La comparaison des fréquences d'apparition avec les autres questions dans le but de minimiser l'importance de mots très répandus n'est pas pertinente dans le cas de recherche de tags sur de petits documents (les questions).

## 7.3. Optimisation du modèle final

Après avoir testé plusieurs valeurs des paramètres de la vectorisation (max_df, min_df, max_feature, norme) passons maintenant aux hyper paramètres du LinearSVC à l'aide d'un GridSearch.

Les résultats permettent l'optimisation suivante :

Penalty = l2

Loss = squared_hinge

Dual =

Tol = 1e-6

C = 1

Max_iter = 1000

Le modèle final ainsi obtenu peut maintenant être intégré dans une API web. On utilise aussi le MultiLabelBinarizer qu'on a entrainé avec les tags sélectionnés et sauvegardé, ainsi que la méthode de nettoyage du texte de la question.

# 8. Remerciements

Je voudrais terminer ce rapport en remerciant chaleureusement mon mentor Mohammed Sedki pour ses conseils et sa patience sur ce projet qui ne s'est pas déroulé exactement comme prévu, le stage du projet 8 s'étant invité au beau milieu de l'agenda ☺, un grand merci à lui donc.