# Projet 5 : Catégorisez automatiquement des questions

Claire Gayral

Août 2021

## Contents

**1 Analyse exploratoire** **2**
1.1 Choix des tags . . . 2
1.2 Prétraitements de textes . . . 3

**2 Prédiction de tags** **5**
2.1 Modélisation non supervisée . . . 5
2.1.1 La NMF . . . 5
2.1.2 La LDA . . . 6
2.1.3 Comparaison des modèles sur l'ensemble test . . . 7
2.2 Modélisation supervisée . . . 7
2.2.1 Présentation des modèles utilisés . . . 7
2.2.2 Les métriques . . . 8
2.2.3 Résultats sur l'ensemble test . . . 9

# Introduction

Stack Overflow, ce site utilisé presque quodiennement par la plupart des personnes amenées à écrire du code, n'est pas si simple à utiliser pour les novices d'un outil informatique, puisqu'il s'appuie sur un système de tags, issus d'une sémantique propre. L'idée du projet 5 de la formation "Ingénieur Machine Learning" d'OpenClassroom est justement de proposer une solution de suggestion de tags à partir de la question. Du nettoyage des données aux différents modèles d'apprentissage, en passant par les méthodes de réduction propres au text-mining, ce document présente le travail effectué pour répondre à cette problématique.

# 1 Analyse exploratoire

La première étape, quelque soit le modèle, va être de nettoyer les posts, et d'en extraire les mots pertinents. Pour l'approche supervisée, les tags formeront les classes à prédire, et pour l'approche non supervisée, les labels permettront de comparer les projections (les mots les plus importants pour chaque topics) et les tags proposés dans le poste.

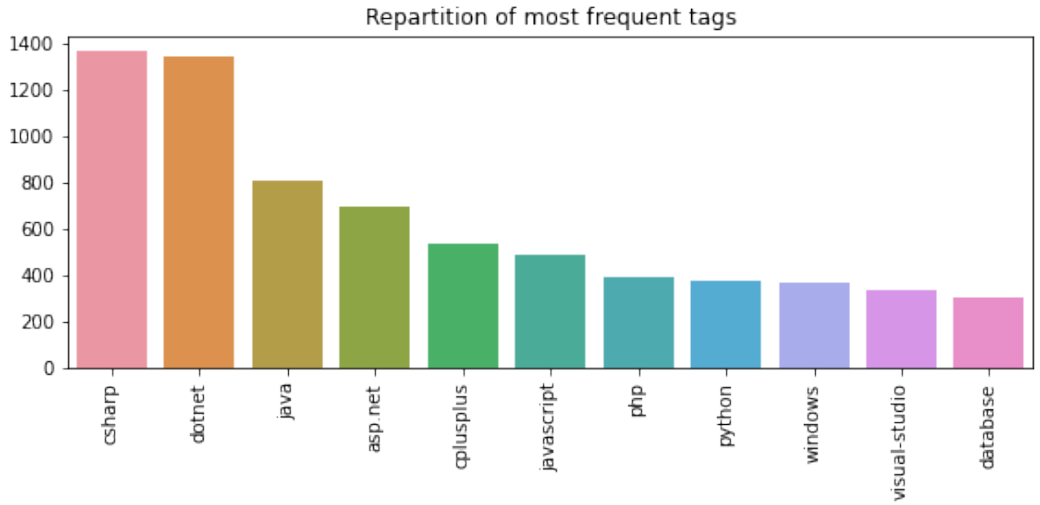
## 1.1 Choix des tags

Pour la modélisation supervisée, il est nécessaire de disposer des tags de sortie. Cependant, dans les posts de Stack Overflow, il y a plusieurs tags pour chaque sujet. Comme la classification multi-classe peut vite s'avérer complexe (problème de dimension, interprétabilité), il faut réduire la complexité des tags en entrée.

Un premier pré-traitement permet de limiter la perte d'information. Par exemple, les tags faisant références à différentes versions du même langage sont homogénéisés afin de limiter la perte d'information. Par ailleurs, les tags qui n'apparaissent que dans un post sont trop peu informatif, et ne seront pas conservés.

### Choix d'une variable de sortie univariée

Pour une classification univariée, le choix le plus simple est de prendre le tag qui apparait le plus souvent. C'est ainsi que la classification supervisée univariée fera référence au tags "csharp"

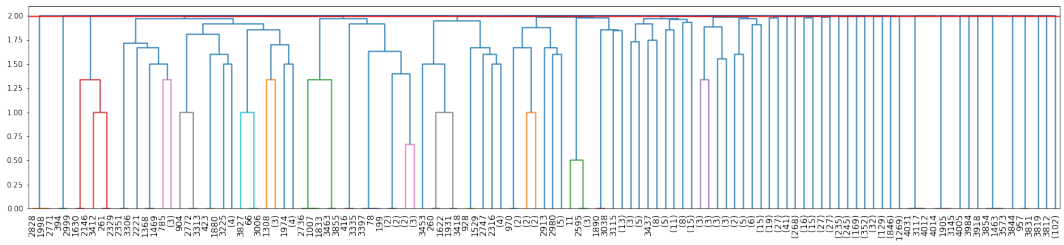



### Réduction de dimension

Une première projection sur des axes choisis (NMF, PCA ou SVD) permet de vérifier s'il existe une structure algébrique qui permet de caractériserr la répartition des tags dans les posts. Ces méthodes de projection permettraient de contruire des méta-tags comme pondération sur les tags à partir des composantes. Cependant, même avec une régularisation Lasso, chaque dimension est soit très fortement tirée par un ou deux tags, soit avec une composition de beaucoup de tags différents.

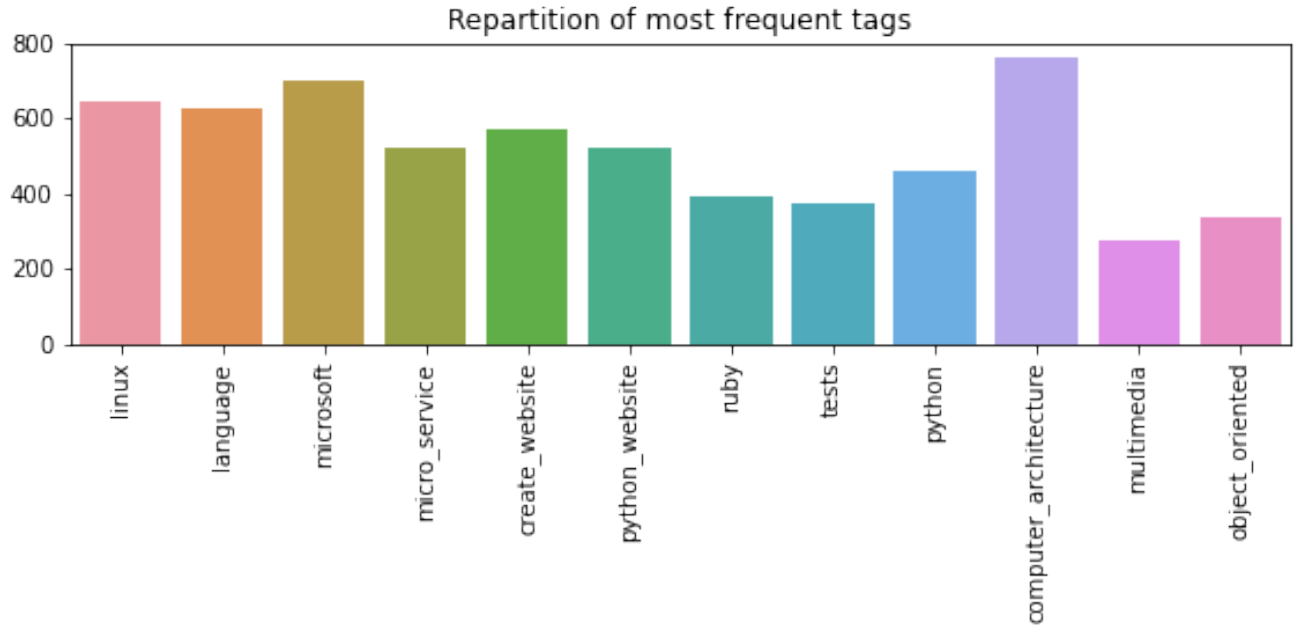
Ensuite, une réduction de dimension par regroupement en communauté permet de regrouper certains tags qui apparaissent dans plusieurs posts.

Un clustering hierarchique est alors plus pertinente : cette approche permet de choisir à postériori où tronquer l'arbre, pour obtenir suffisament peu de cluster, et garder des clusters qui ont un sens, et qui sont suffisamment représentés dans les données pour apprendre le lien avec le texte des posts.



Par exemple, le cluster numéro 46 de la classification donnée par le tronquage de la ligne rouge (à 1.99) regroupe les termes suivants, faisant tous référence à la création de site web avec python :
`active-directory, android-emulator, apache2, app-config, asp.net-membership, assert, authentication, authorization, certificate, code-snippets, distribution-list, django, django-authentication, django-model django-templates, django-urls, django-users, dojo, e-commerce, firebug, firephp, forms-authentication, http-authentication, internet-explorer, intranet, javascript-debugger, ...`

Du clustering, j'extrait 12 clusters que mes connaissances informatiques permettaient d'identifier :
`linux, language, microsoft, micro_service, create_website, python_website, ruby, tests, python, computer_a multimedia, object_oriented`




## 1.2 Prétraitements de textes

Dans un premier temps, il fallait choisir une façon de représenter les mots, et en extraire leur sémantique. Pour cela, j'ai effectué un prétraitement contenant :

Les différentes étapes sont illustrées sur la phrase suivante :
`<p>How do I forcefully unload a <code>ByteArray</code> from memory using ActionScript 3?</p>`

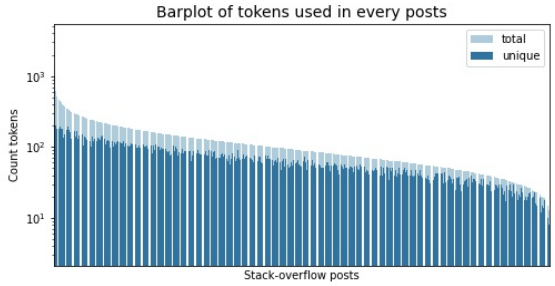
1. Changement de format et suppression de la ponctuation, filtre pour regrouper les différentes versions des langages de programmation
   `how, do, i, forcefully, unload, a, bytearray, from, memory, using, actionscript'`

2. Retrait des mots de transition ou très présents (stop words)
   `forcefully, unload, bytearray, memory, actionscript`

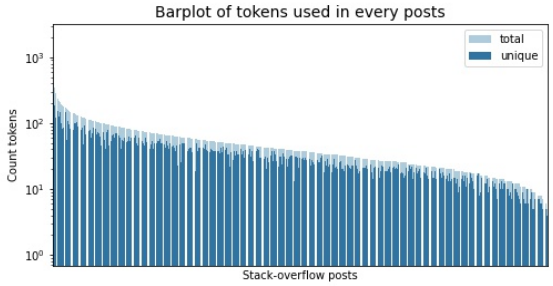
3. Lemmatisation
   `forc, unload, bytearray, memori, actionscript`

Ces traitements permettent de réduire progressivement la dimension des données textuelles, en ne conservant que les informations pertinentes. L'évolution de la répartition des mots peut être suivie par différentes statistiques telles que le nombre d'apparition des mots dans les postes :

1. Répartition des mots dans le corpus original :



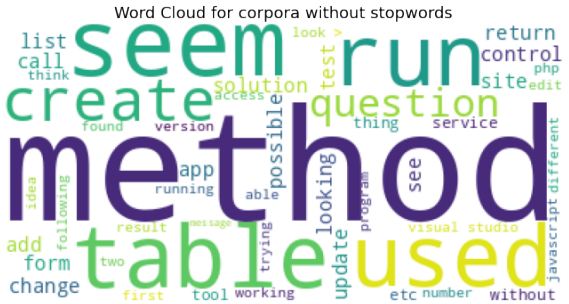

2. Répartition des mots après les pré-traitements :




Ces différents prétraitements donnent accès à une liste de pseudo-mots appelés tokens. Ces ensembles de tokens peuvent être représentés de différente façon selon les modèles statistiques pour lesquellent ils vont être utilisés.

- bag of words (BOW) : C'est la représentation naturelle, consistant en une matrice réprésentant le nombre d'occurence de chaque mots dans les différents postes. Commes il y a beaucoup de mots différents et que très peu de mots sont appelés dans chaque postes, cette matrice est très creuse.
- tfidf : Une pondération des différents tokens prenant en compte leur rareté
- word2vec : D'autres représentations existent et sont plus complexes, s'appuyant sur des modèles de NLP déjà entraînés.

Les mots peuvent aussi être représentés graphiquement, notamment à l'aide de nuages de mots comme suit :


Ces illustrations permettent par exemple de voir l'importance de retirer les mots de liaisons tels que "S" ou "need" ou "one".

Le corpus sans stopwords, et lemmatisé servira de données d'entrainement pour la suite. Pour rappel, l'objectif de ce projet est de proposé des tags de façon automatique. Pour cela, deux approches différentes sont utilisées. La première est non supervisée : on cherche des tags sans apriori, c'est-à-dire sans utiliser les tags liés aux publications. A l'inverse, la deuxième approche apprend le lien entre la distribution des tags et la distribution des mots dans les publications.

# 2 Prédiction de tags

Nous allons donc proposer et tester différents modèles pour prédire les tags, puis les comparer.

## 2.1 Modélisation non supervisée

Une première approche consiste à projeter la distribution des mots dans des espaces réduits, que l'on appellera "topic". Les phrases alors projetées seront une combinaison de ces topics : plus le poids du topic est grand, plus la phrase est dans ce thème. Comparer les différentes projections, la métrique utilisée est le MSE, c'est-à-dire l'erreur moyenne au carrée entre la table orginale et sa projection dans l'espace réduit. Ce critère, à minimiser, permet dans un premier temps de choisir les meilleurs paramètres pour chaque modèles, puis de choisir le modèle le plus adapté.

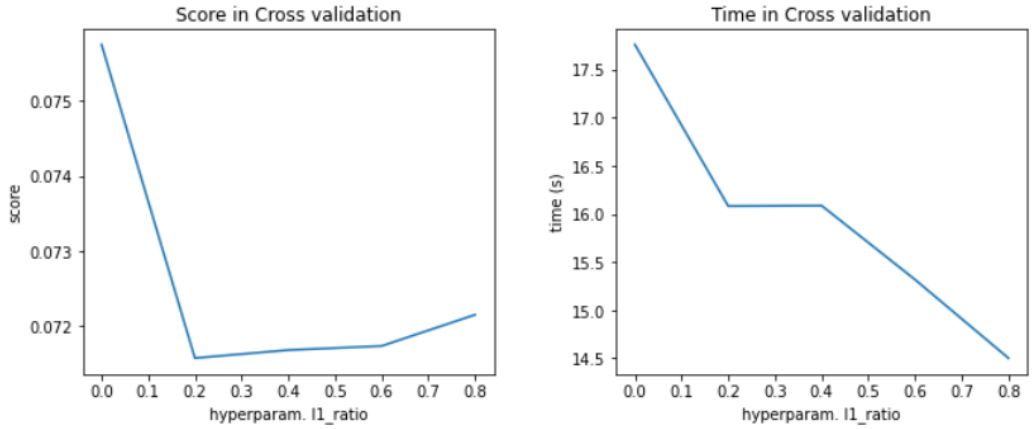
### 2.1.1 La NMF

Comme pour la projection non supervisée des tags, comme tous les coefficients sont positifs, on peut faire appel à la "Non-Negative Matrix Factorization" (NMF) pour projeter les tokens dans des espaces sémantiques. Cette méthode de projection est une décomposition de la matrice des mots en BOW. Il s'agit de trouver les variables latentes (topics) qui expliquent au mieux la structure de covariance entre les variables. Contrairement à la PCA, qui maximise la variance dans l'espace projeté, la NNF n'a pas de contrainte d'orthonormalité.

Pour cette partie, le pré-traitement du corpus est relancé avec le module de sklearn, afin d'accéder facilement aux features, comme proposé dans cet exemple de sklearn.

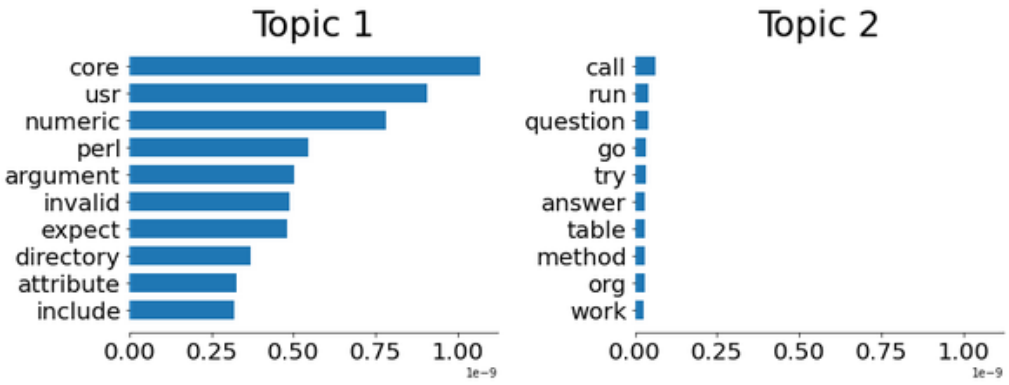
Le choix des hyper-paramètres est fait par une pseudo validation croisée. L'ensemble d'entrainement est divisé en différents folds. Le modèle est alors appris sur un sous-ensemble de l'entrainement initial, et validé sur la partie laissée intacte. On ne peut pas vraiment parler de validation croisée car il n'y a pas de sortie visée, la métrique sur l'ensemble de validation ne fait pas référence à une sortie mais à la similitude, mesurée par le MSE, entre la validation et sa projection dans le sous-modèle.

Ainsi, les paramètres optimaux sont choisis par la lecture graphique. Par exemple, dans le ratio de pénalisation l1 est optimal à 0.2 :




Pour donner un exemple plus détaillé, pour la NMF sur ce corpus, le meilleur modèles est choisi avec 20 composantes et une régularisation lasso de 0.2. Ces 20 composantes ne sont pas le meilleur paramètre, puisque le score chutte encore après 500 composantes, néansmoins, il devient trop fastidieux d'explorer autant de topics.

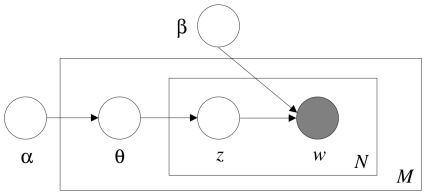
Bien que pratique à manier, le MSE donne peu d'information quant à la pertinence de la projection. Il est alors intéressant de regarder le poids des principaux mots pour les topics.



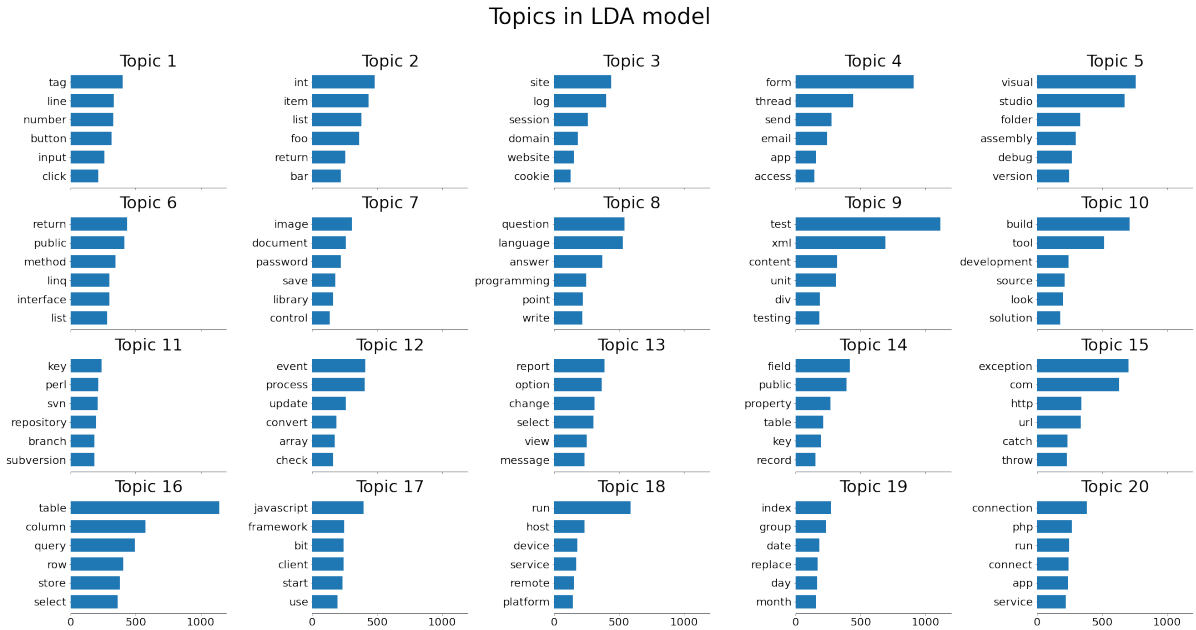

### 2.1.2 La LDA

Une autre méthode de projection basée sur un modèle génératif, la "Latent Dirichlet Allocation" est un modèle bayésien couramment utilisé pour l'extraction de topics.

Le modèle graphique probabiliste de la LDA est le suivant Cette figure est issue de l'article orginale de la LDA. Le consulter pour le détail du sens de chaque variable latente.

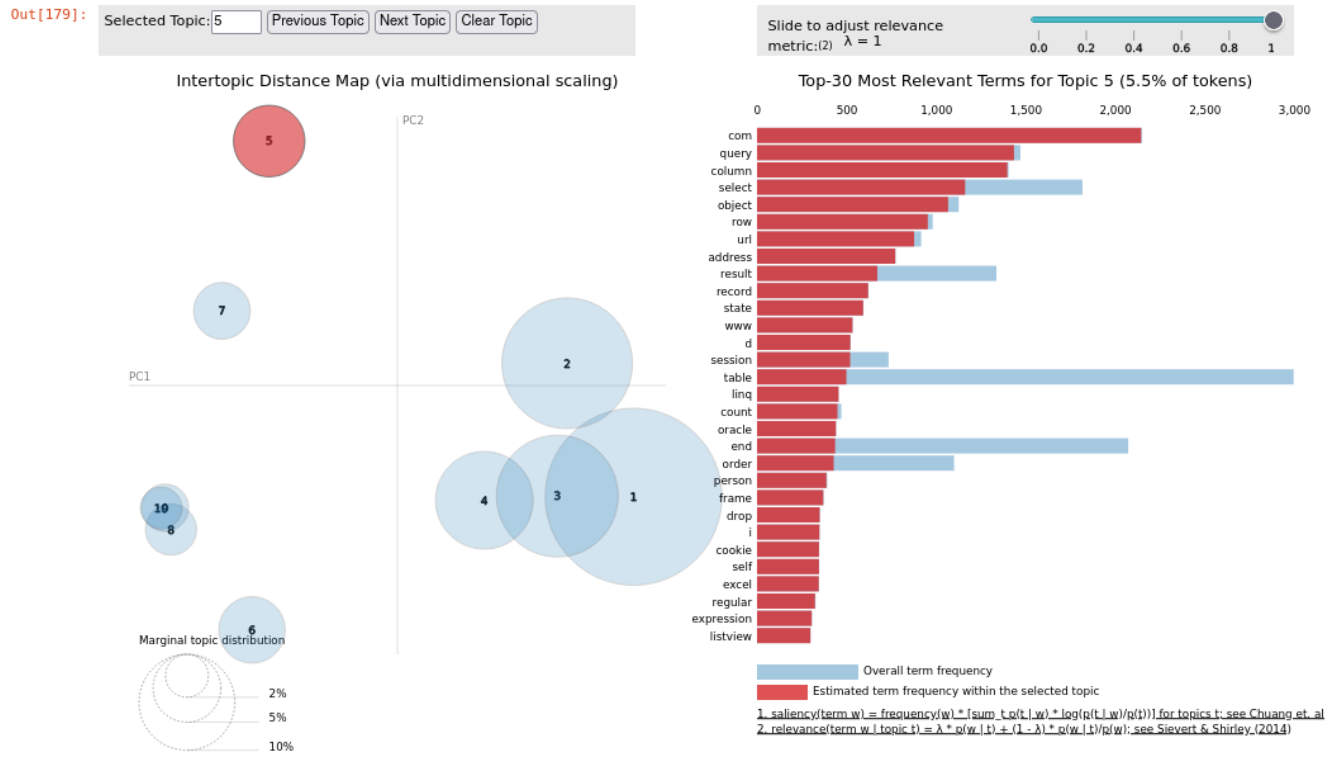



Avec la même méthodologie que celle présentée pour la NMF, les paramètres du modèle sont fixés à learning_decay = 0.9, learning_offset = 100, n_components = 2000, max_iter = 5. Bien que ce soit le modèle le plus fidèle aux données, il est fastidieux de trouver un nom pour chaque topics lorsqu'il y en a 2000. Pour aller au bout du projet, j'ai donc réduit le nombre d'axes de projection, à 20 classes, les autres paramètres de la LDA sont adaptés à cette contrainte. Les 20 classes sont alors résumées comme suit :




Ainsi, la prédiction de tags à partir de cette table implique une labellisation des topics à partir de leur sémantique, comme le topic 1 semble référer à l'UI, le deuxième aux manipulations de base,...

Il est possible d'explorer les différentes sémantiques à l'aide de l'outil de visualisation pyLDAviz, donc l'interface est la suivante :




La visualisation ne correspond pas au même modèle de LDA que celui proposé dans la décomposition des topics, car le premier modèle est sklearn, le deuxième gensim.

### 2.1.3 Comparaison des modèles sur l'ensemble test

L'objectif était de prédire les tags les probables pour les publication de l'ensemble de test. On peut au moins prédire les topics pour chaque post, et en constituer un tag.

## 2.2 Modélisation supervisée

Ensuite, des modèles supervisés sont entrainés pour apprendre le lien entre la distribution des mots dans les postes, et les tags associés. Un premier travail consiste à apprendre les relations entre la distribution des mots et le tag en question. Un deuxième axe de travail cherche une classification en plusieurs classes, à partir des méta-tags construits dans la première partie.

### 2.2.1 Présentation des modèles utilisés

Les 3 modèles suivants sont couramment utilisé pour la classification de texte.

**Naive Bayes**

La classique classification de Bayes Naive suppose que les variables sont indépendantes et renversent le théorème de Bayes pour prédire la classe la plus probable conditionnée aux mots observés. Ces classificateurs naïfs sont très courants pour la classification de texte.

Ces modèles assez simples n'ont que deux paramètres à fixer. Il s'agit d'une part de la loi de probabilité des mots. Je testerai avec une Bernoulli Naive Bayes, une Complement Naive Bayes, et Mutlinomial Naive Bayes. Le deuxième paramètre, alpha, un paramètre de régularisation Pour chaque modèle, le choix du meilleur alpha est fait par validation croisée. L'accuracy du meilleur model est de presque 90 sur l'ensemble de test.

**Gradient Boosting**

Le Gradient Boosting est une méthode de recherche de minimum pour une perte donnée, pour laquelle la descente de gradient est faite sur plusieurs arbres décisionnels, pour éviter de tomber dans des minima locaux. Ainsi,

cette méthode dépend énormément de la métrique choisie. Pour la classification binaire, on peut par exemple tester la perte logistique, ou exponentielle, l'accuracy ou encore l'AUC. Je resterai basique en comparant les pertes logisitique et exponentielle.

Perte logistique :

$$L_{\text{logistic}} : v \to K\log(1+e^{-v})$$

nb : pour "$K = \frac{1}{\log(2)}$" on parle de perte logistique
et pour "$K = 1$" de perte de classification cross-entropique

Perte exponentielle :

$$L_{\text{exp}} : v \to e^{-v}$$

C'est la perte utilisée notamment dans l'algorithme AdaBoost.

**Random Forest**

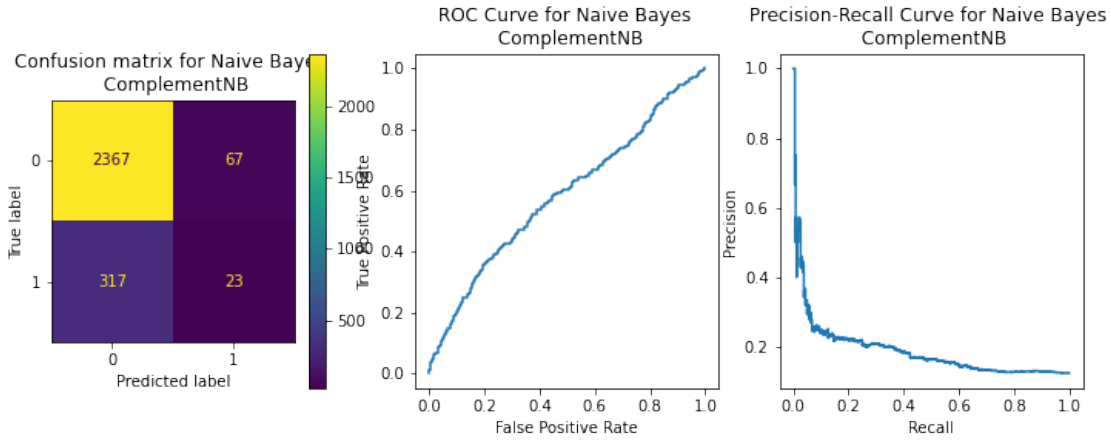
Les forêts aléatoires - Randoms Forest en anglais - sont des méthodes de classification pour lesquelles plusieurs arbres décisionnels dits faibles sont calculés, à partir d'un sous-ensemble de l'entrainement. Le modèle de classification final est alors obtenu par un vote à la majorité des classifieurs. Comme pour les autres méthodes, les paramètres de ce modèles sont sélectionnés par validation croisée.

### 2.2.2 Les métriques

Pour sélectionner et comparer les modèles, on utilisera 3 métriques différentes.

En premier, l'accuracy est définie comme le quotien entre le nombre de prédiction correcte et le nombre de prédiction totale. L'accuracy est donc à maximiser, sachant que la valeur maximale est de 1, atteinte lorsque les classes produites sur l'ensemble de teste coincindent exactement avec les classes réelles. Une classification aléatoire aurait alors une accuracy égale à $\frac{1}{2}$. Néanmoins, les modèles lancés sont feignants : ils prédisent la classe dominante systématiquement. Ainsi, l'accuracy semble bonne : comme la classe la plus représentée apparaît plus souvent, le prédicteur se trompe moins qu'un aléatoire en bernoulli(0.5). Donc cette métrique ne sera pas la métrique la plus pertinente pour cette classification, car les tags ne sont pas répartis de façon uniforme.

Un autre ensemble de métrique découle de la matrice de confusion. C'est le cas de l'Area Under Curve (AUC), qui est l'aire sous la courbe ROC. Elle indique la probabilité qu'une instance positive choisie au hasard soit classée plus haut qu'une instance négative choisie au hasard. Lorsque l'on fait une classification multi-classe et multi-label, cette métrique est à prendre avec des pincettes car les labels ne sont pas répartis de façon homogène entre les postes. A creuser : AUC pour pour callibrer un modèle de classification




Enfin, la perte logistique est une métrique à minimiser. Elle revient à minimiser la divergence de Kullback, c'est-à-dire une mesure de similarité entre la distribution prédite et la loi théorique modélisée. Elle est ainsi aussi appelée cross-classification entropy ou log-loss. Pour une classification binaire, s'écrit comme :

$$L(y,p) = -(y\log(p) + (1-y)\log(1-p))$$

### 2.2.3 Résultats sur l'ensemble test

**Classification binaire**

Dans cette partie, le tag a été choisi comme le tag le plus présent dans les postes, il s'agit du langage "csharp". Les modèles suivants cherchent à produire une classification des données en "fait référence à csharp" et "ne fait pas référence à csharp". Il serait alors possible de produire un classifieur par tag, et garder au final comme prédiction les tags les plus probables pour chaque publication.

Deux modèles ont été testés pour cette tâche : les modèles Naive Bayes, et le gradient boosting.

Résultats de la classification binaire Naive Bayes (NB) :

| Loi | $\alpha$ | Accuracy | log-loss | AUC | Temps |
|---|---|---|---|---|---|
| Bernoulli | 100 | 0.876 | 4.27 | 0.49 | 0.12s |
| Complement | 125 | 0.862 | 4.78 | 0.52 | 0.05s |
| Multinomial | 600 | 0.877 | 4.25 | 0.5 | 0.05s |

Résultats de la classification binaire Gradient Boosting :

| Loss | n estimators | learning rate | max depth | Accuracy | log-loss | AUC | Temps |
|---|---|---|---|---|---|---|---|
| Deviance | 100 | 0.01 | 3 | 0.88 | 4.16 | 0.524 | 13.9s |
| Exponential | 100 | 0.2 | 3 | 0.878 | 4.22 | 0.501 | 13.5s |

**Classification multi-classe**

Deux modélisations ont été testées pour la classification multi-classe. La première est donnée par le clustering sur les table effectué dans l'exploration des données, la deuxième est constituée des tags qui apparaissent le plus souvent.

*Classifieur multi-label ou uni-label ?*

Prédire label par label peut s'avérer fastidieux, et demande une grande quantité de postes, pour que tous les labels soient correctement représentés. Ainsi, une classification multi-classe peut être utilisée pour avoir un seul modèle pour tous les tags. Il y a deux prédictions possibles pour la classification multi-classe :

- soit on sépare les postes en catégories et on obtient une classe par poste dans la prédiction. On parle de classification multi-classe uni-label. Bien que certain modèles existent sous cette forme, le gros problème est que les classes ne sont pas séparées dans notre ensemble de postes : certain postes ont plusieurs tags. Ainsi, soit on prend le tags qui apparait en premier dans la table, ou on prend un des tags au hasard, ou encore, vu que les modèles ont tendance à être lazy, on peut aussi prendre le tags qui apparait le moins souvent (ce qui tendrait à réduire la disparité de répartition des tags).
- soit on prédit plusieurs tags par poste, ou du moins une probabilité pour chaque tag qui est ensuite seuillée. Cela revient exactement à lancer une classification binaire pour chaque poste, en ajustant le seuil pour laquelle la probabilité d'être dans la "bonne classe" est modifiée. Cela s'appelle la classification multi-label.

*Résultats de la classification*

| Modèle | Accuracy | log-loss | AUC | Temps |
|---|---|---|---|---|
| Complement NB | 0.615 | 2.33 | ovo = 0.569, ovr = 0.574 | 0.25s |
| Arbre de décision - gini | 0.615 | 3.1 | 0.71 | 1.14s |
| Arbre de décision - entropie | 0.609 | 3.07 | 0.73 | 1.05s |
| Gradient Boosting | 0.685 | 1.137 | ovo = 0.842, ovr = 0.81 | 3.5 min |

Les paramètres respectifs sont précisés dans le notebook.

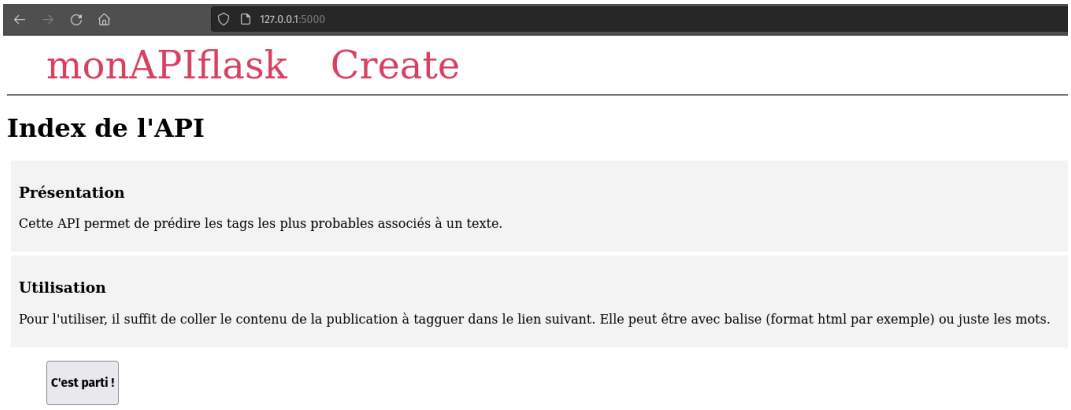
*Multi-label*

Excepté les arbres de décisions, il n'est pas possible d'utiliser les méthodes ci-dessus pour prédire du multi-label (c'est-à-dire plusieurs classes à prédire - multiclasse, et chaque poste peut avoir plusieurs labels). La méthode

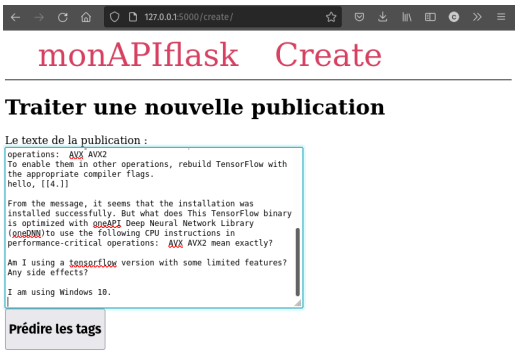
la plus simple serait alors de lancer une méthode binaire pour chaque label. L'idée serait alors de faire une première classification avec ces tags regroupé puis de relancer une sous-classification parmis ces tags. En effet, en retirant les postes qui parlent de choses très différentes, on peut vite réduire la dimension des données d'entraînement, et obtenir des classifieurs plus pertinents.

## Conclusion

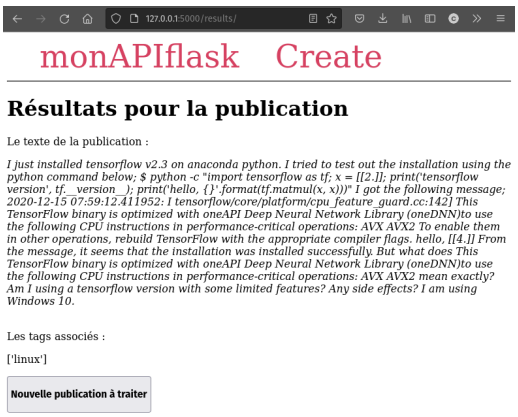
Le meilleur modèle, celui choisi pour la mise en production est la classification multi-classe donnée par le clustering hierarchique et pour lequel l'unique label attribué à chaque poste est pris au hasard parmis les tags. Ce modèle est enregistré, et peut-être appelé soit en console python, soit dans l'API suivante :




Le formulaire




Les résultats




Pistes d'amélioration

- Tester l'influence des différents pré-traitements et représentations des mots
- D'autres modèles pré-entrainés et sûrement beaucoup plus performants existents. BERT en est un exemple. Cela aurait été intéressant d'explorer cette méthode.
- Faire deux niveaux de tags : un premier global (les méta-tags, c'est-à-dire la classification de l'API), et une sous-classification par méta-tag pour avoir les tags finaux. Utiliser la LDA sur le corpus comme tags première projection, puis relancer une classification avec tous les tags
- D'autres pré-traitements possibles : un tag est dans le text, directement l'extraire et changer de composante

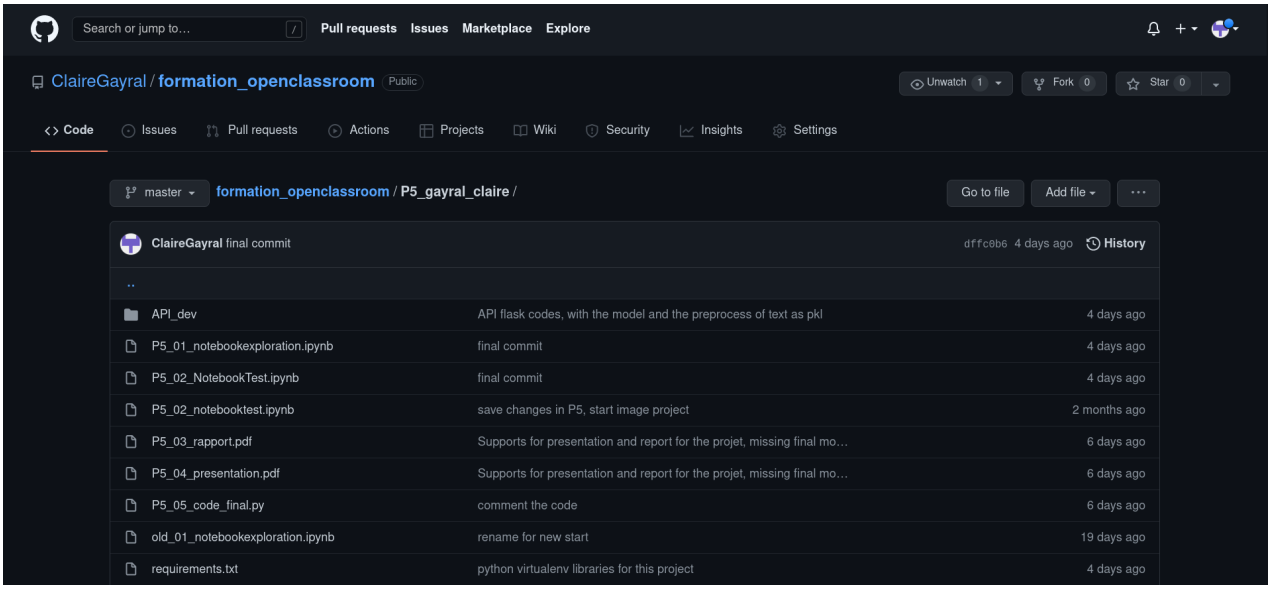
# Bibliographie


- Pré-traitements du corpus :
  - Une référence tès complète sur le NLP
  - Le cours d'OC sur le traitement des données textuelles
  - Deux bibliothèque principales python : gensim et scikit-learn
- Représentation des mots :
  - Cours d'OC sur la représentation des mots
  - Word 2 vect
- Extraction de mots-clé (non supervisé) - généralités :
  - Article overview
  - Slide de présentation séminaire sur "Topic Models" Thibaut THONET
- NMF :
  - La page wikipedia
  - Le cours d'OpenClassRooms sur les méthodes non supervisées
  - La doc de scikit-learn et un exemple d'utilisation pour l'extraction de topics
- LDA :
  - La page wikipedia
  - Un article sur l'extraction de topic avec la LDA, un autre article
  - https://github.com/blei-lab/onlineldavb
- Métriques pour la classification :
  - Détail des différentes pertes de classification sur wikipédia
  - Logistic Regression sur les Machine Learning Crash Course de Google. Sur la même ressource, il y a aussi un cours sur l'accuracy, les courbes ROC, ...
  - un récap très synthétique des différentes pertes
  - Une référence de Jason Brownlee qui détaille l'entropie croisée et une autre sur la régression logistique
- Naive Bayes:
  - Une explication illustrée
  - Un article similaire - des idées à la programmation, sur la prédiction de lien sémantique
  - Une référence en français
  - La documentation de sklearn est détaillée à ce sujet
- Gradient Boosting
  - Présentation de l'algorithme
  - Un article de Jason Brownlee
  - La documentation de sklearn
- Random Forest
  -

- API flask
  - création d'un template html de base
  - Boîte d'entrée en multi-ligne
  - Documentation flack : prise en main, et page d'accueil de la documentation
  - Le cours d'OC pour créer des dashboard en flask
  - de digital ocean, trois ressources utiles : un tuto flask de base, un article pour créer un htlm de base et comment créer une API python avec flask

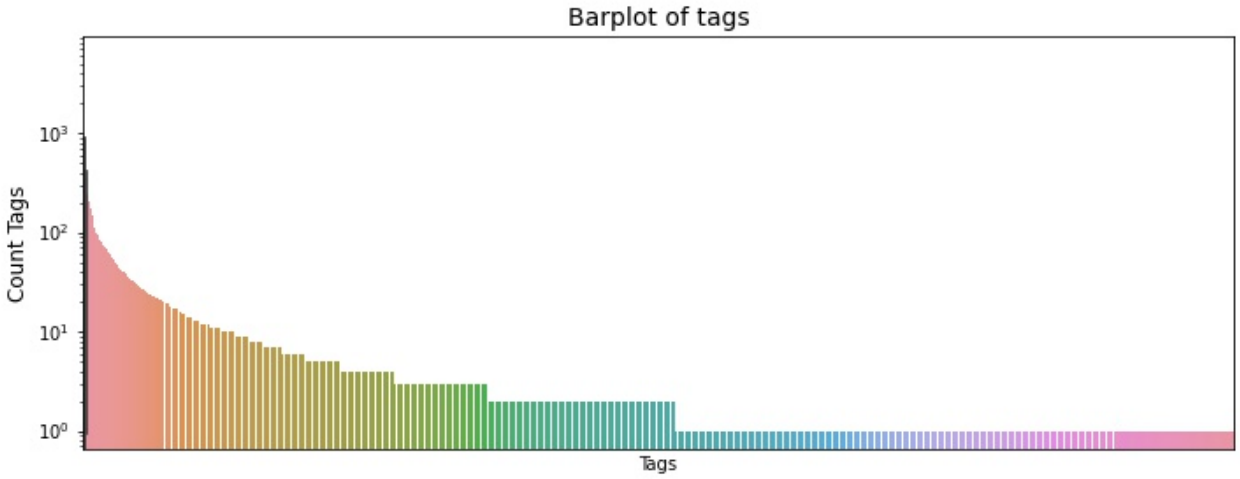
# Suivi de version

L'intégralité du projet est suivi dans le répertoire github en lien.

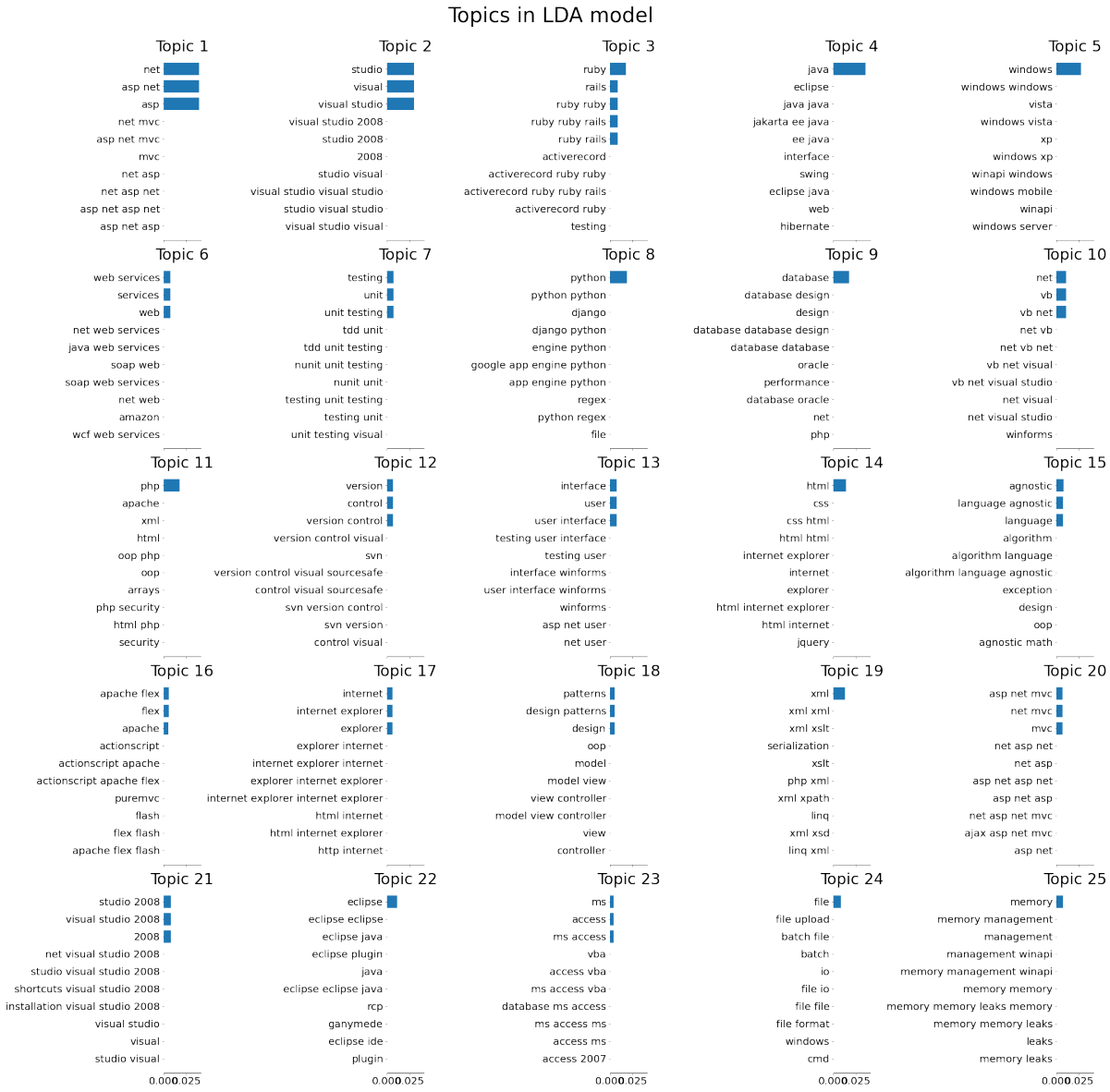


# Annexe

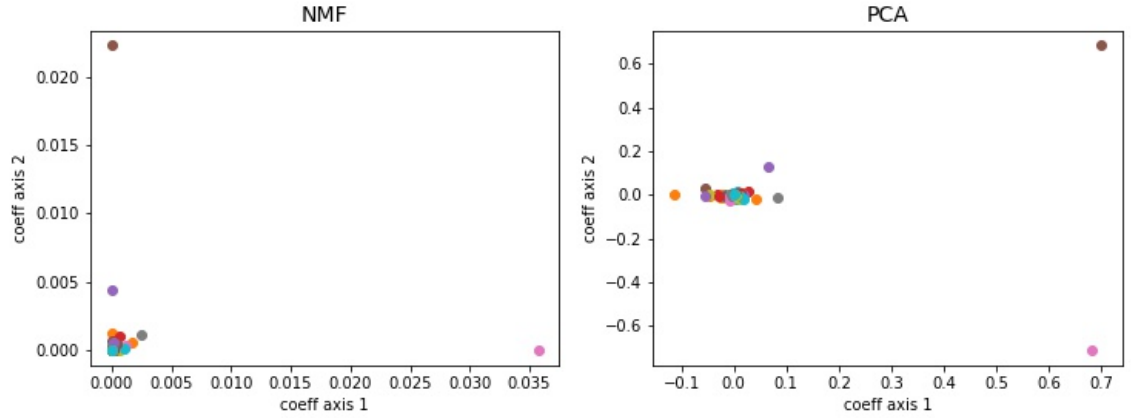
Distribution des tags :



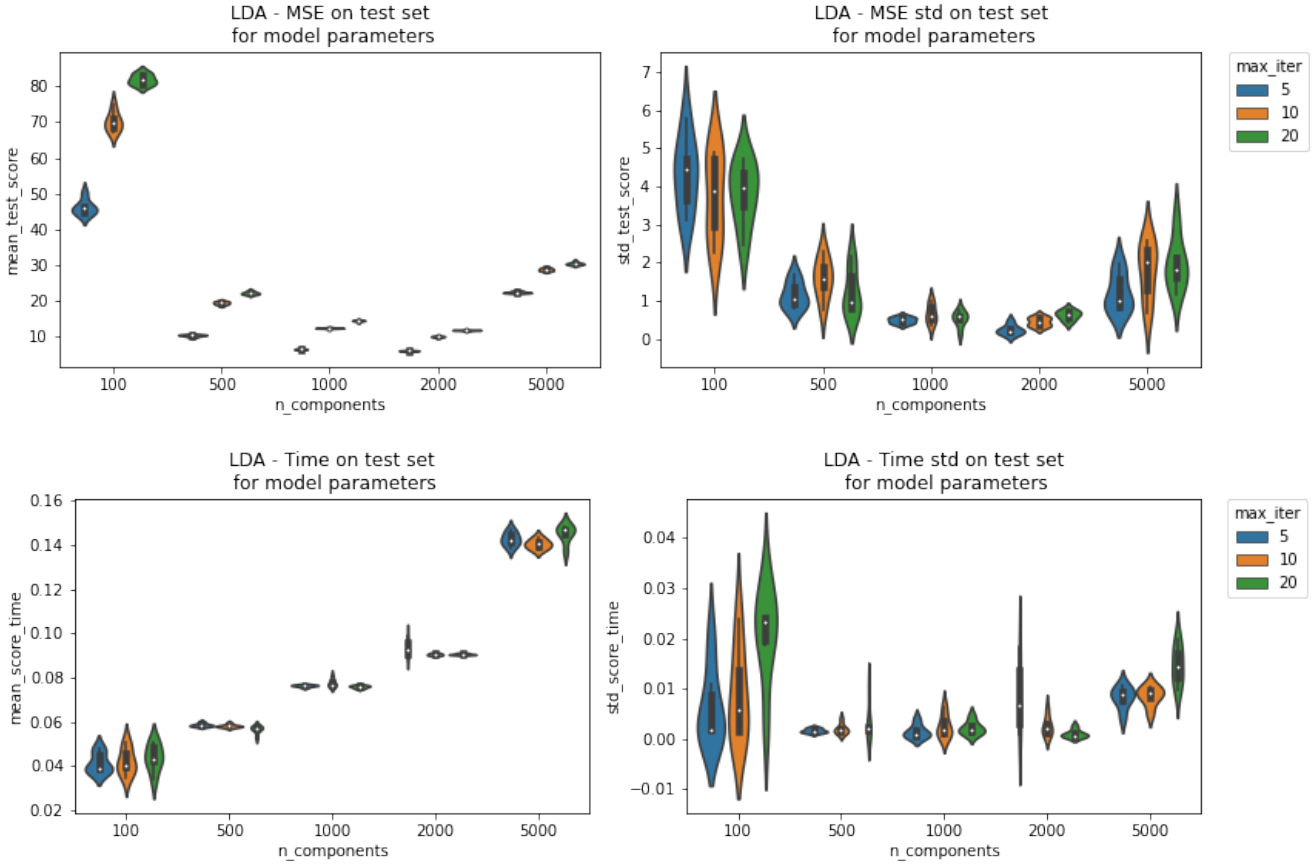

Topics pour la NMF sur les tags :



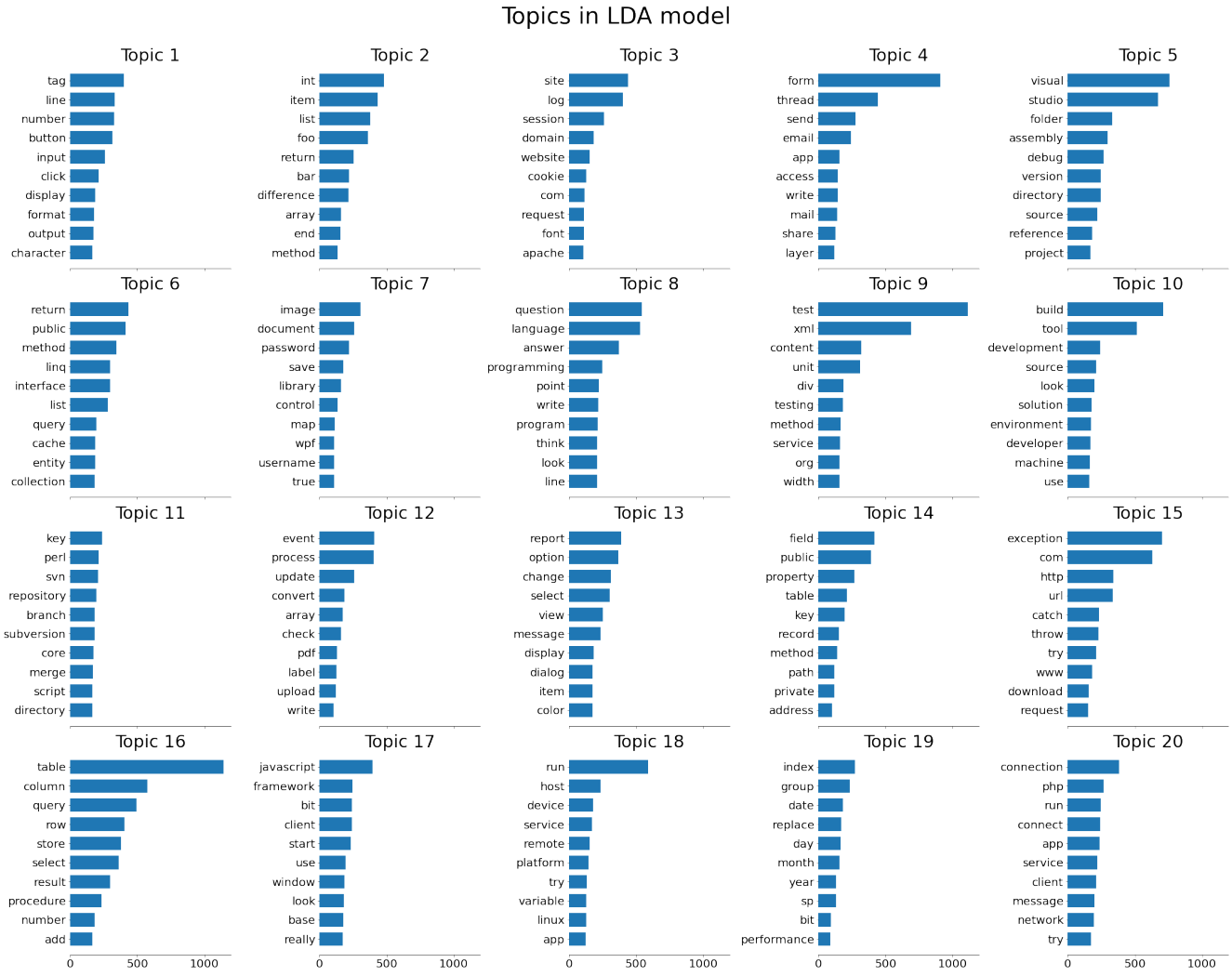
Projection des tags dans 2 dimensions :



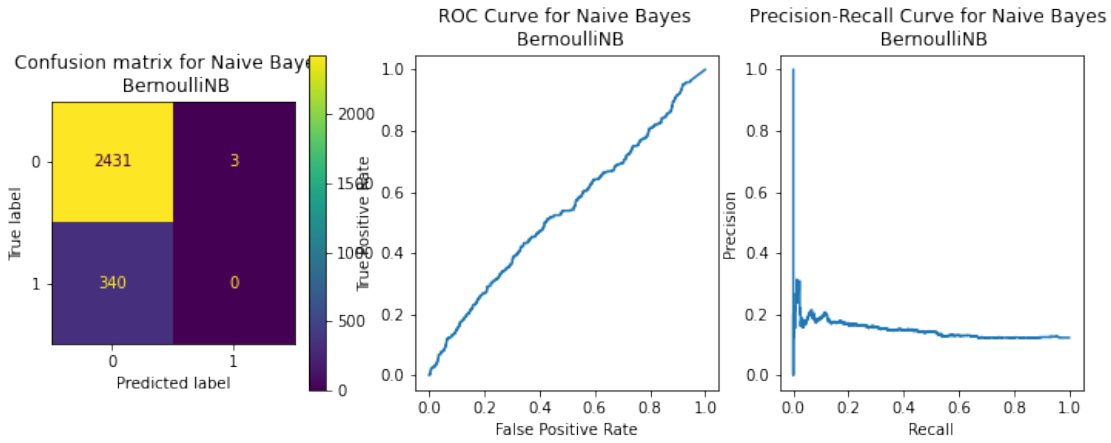

Choix des hyper-paramètres par Cross Validation (CV) pour la LDA sur le corpus :

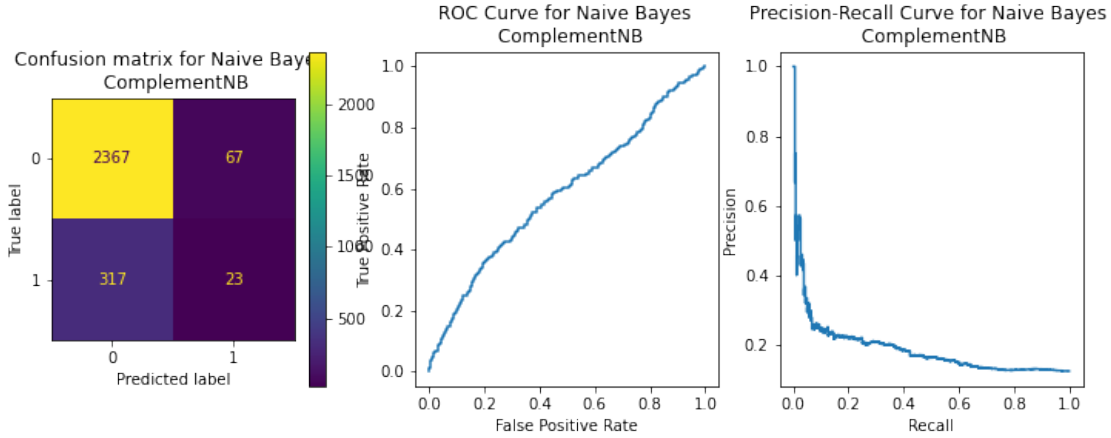


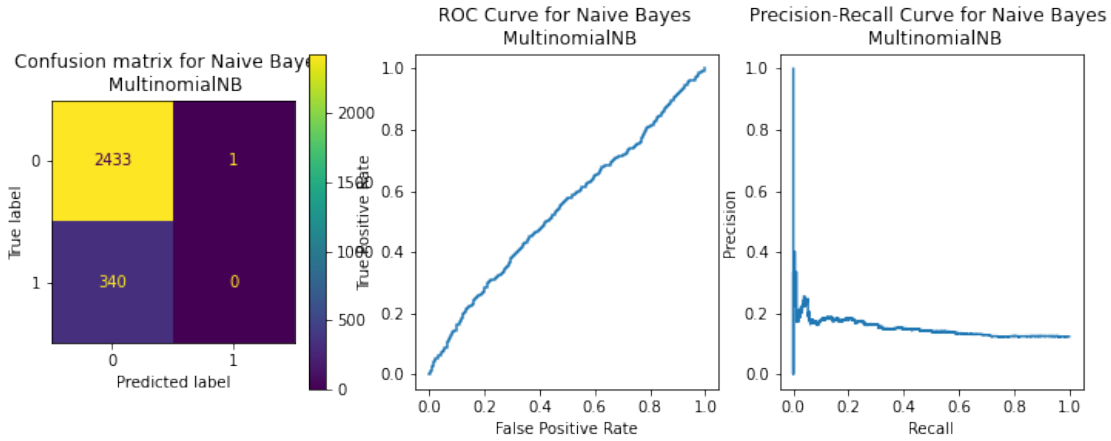

Projection LDA du corpus :

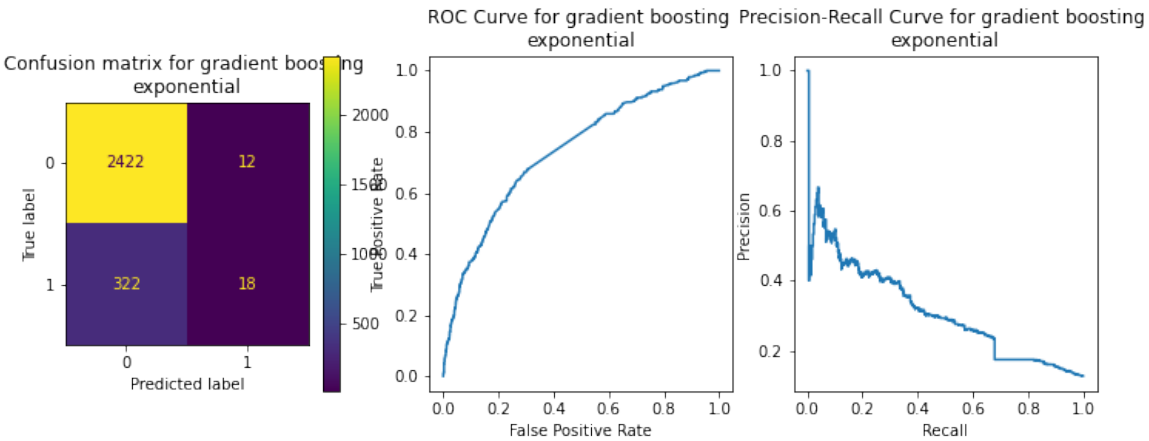


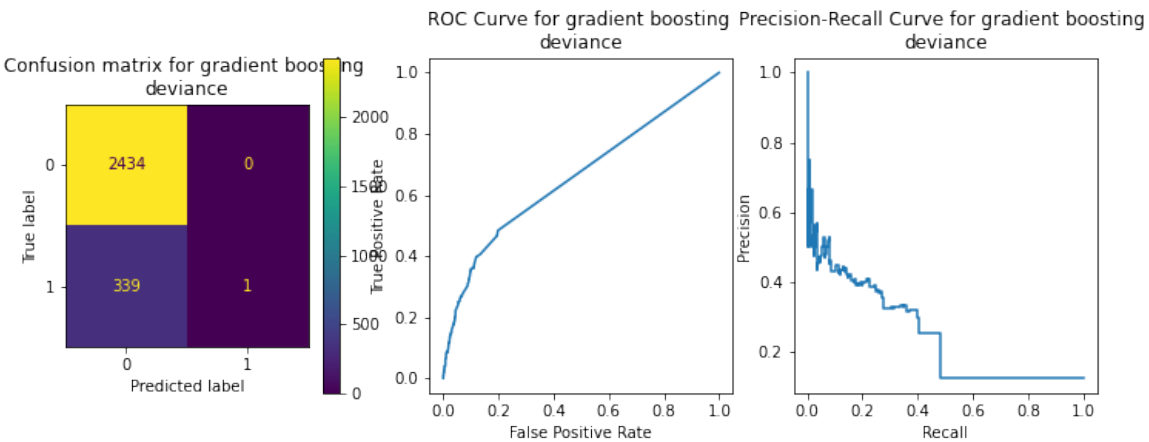

Les courbes issues de la matrice de confusion pour la classification binaire :



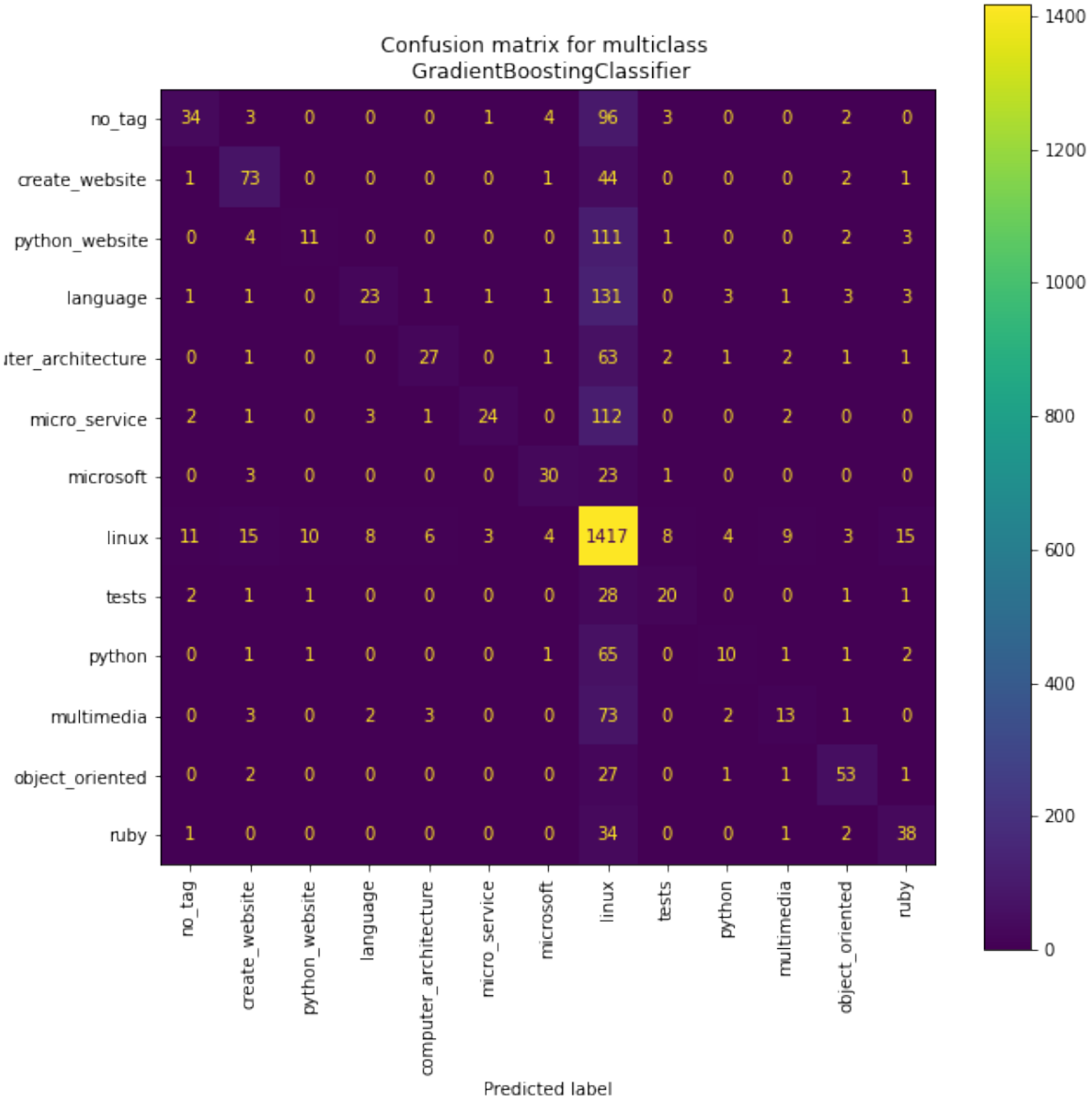

Confusion matrix for Naive Baye
ComplementNB
2367
67
317
23
True label
Predicted label
ROC Curve for Naive Bayes
ComplementNB
True Positive Rate
False Positive Rate
Precision-Recall Curve for Naive Bayes
ComplementNB
Precision
Recall


Confusion matrix for Naive Baye
MultinomialNB
2433
1
340
0
True label
Predicted label
ROC Curve for Naive Bayes
MultinomialNB
True Positive Rate
False Positive Rate
Precision-Recall Curve for Naive Bayes
MultinomialNB
Precision
Recall


Confusion matrix for gradient boos ng
exponential
2422
12
322
18
True label
Predicted label
ROC Curve for gradient boosting
exponential
True Positive Rate
False Positive Rate
Precision-Recall Curve for gradient boosting
exponential
Precision
Recall

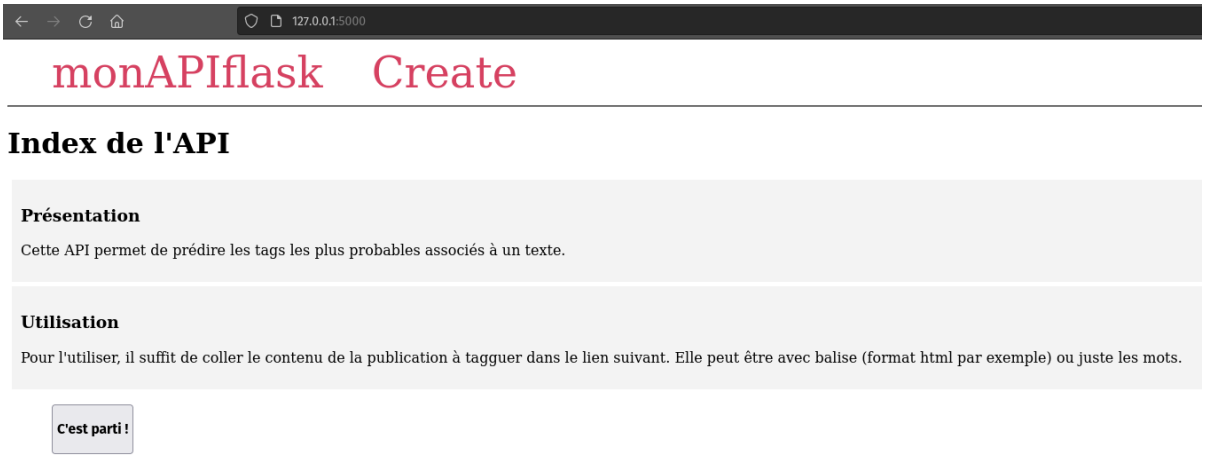

Confusion matrix for gradient boos ng
deviance
2434
0
339
1
True label
Predicted label
ROC Curve for gradient boosting
deviance
True Positive Rate
False Positive Rate
Precision-Recall Curve for gradient boosting
deviance
Precision
Recall

Matrice de contingence, gradient boosting multi-class avec un label pris au hasard parmis les tags de chaque poste :

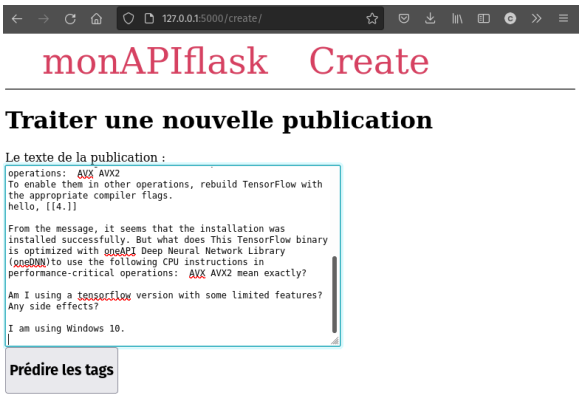


L’API produite avec flask :

La page d’accueil




Formulaire




Résultat