



CATÉGORISATION DE QUESTIONS

Comparaison de différents algorithmes

Résumé

Ce rapport présente les différentes étapes qui ont été nécessaires à la réalisation d'une API de catégorisation de questions. Il se concentre sur les étapes d'obtention des données, du nettoyage ainsi que du choix de l'algorithme qui allait sous-tendre cette API. Une comparaison entre différents modèles a été faite pour pouvoir choisir celui qui semblait le plus pertinent pour l'application à mettre en place.

Cécile Guillot
cecile.gltslmcs@protonmail.com

Table des matières

Acquisition des données	2
Analyse exploratoire du corpus	2
Analyse des données numériques	2
Critères de sélection des posts	2
Preprocessing des posts restants	3
Visualisation des mots présents dans le corpus	4
Visualisation de la composition des posts	4
Les titres	4
Le corps de texte (ou body)	5
Les tags	5
Modélisation : Comparaison de différentes modélisations	6
Les modèles non-supervisés : Topic modelling	6
Non Negative Matrix Factorization (NMF)	6
Latent Dirichlet Allocation (LDA)	7
Evaluation du topic modeling	9
Conclusion	9
Les modèles supervisés : du Machine Learning « classique » au Deep Learning	9
La méthode d'évaluation	9
Les méthodes de Machine Learning	10
Algorithme de machine learning pour le multilabel	10
Deep Learning et plongement (embedding) de mots	10
Présentation de l'algorithme MLkNN	10
Tableau de synthèse	- 1 -
Références	1
Annexe 1 : Exemple de requête SQL	5
Annexe 2 : Code d'automatisation pour la création d'un CSV unique	6
Annexe 3 : Fonctions de pre-processing des données textuelles	7
Annexe 4 : Hiérarchie des topics selon le LDA	8
Annexe 5 : Architecture du réseau de neurones LSTM après optimisation bayésienne	9
Annexe 6 : Endpoint de l'API déployé	10
Annexe 7 : Repo GitHub hébergeant l'application	11

Le traitement naturel du langage est un champ très large qui regroupe différentes tâches. On retrouve des applications en lien avec l'analyse de sentiments ou encore la classification. La difficulté du travail avec des données textuelles est le fait qu'elles sont non structurées contrairement à des données tabulaires. Les étapes de preprocessing seront donc plus importantes pour rendre ces données compréhensibles par la machine afin d'utiliser des méthodes de Machine Learning dessus.

Dans ce rapport, plusieurs étapes vont être présentées. Il s'agit des étapes réalisées pour arriver à la création d'un algorithme qui permet de catégoriser des questions. La première partie porte sur l'acquisition des données, elle sera suivie par une partie d'analyse exploratoire du corpus obtenu avec une part importante de preprocessing. Enfin, une dernière partie sur les comparaisons de plusieurs algorithmes de Machine Learning sera présentée pour montrer la démarche réalisée pour sélectionner l'algorithme le plus pertinent dans la tâche que l'on cherche à réaliser.

Acquisition des données

L'acquisition des données a été réalisée à l'aide de l'outil de collecte du site de Q&A StackOverFlow (<https://data.stackexchange.com/stackoverflow/query/new>). La récupération des données s'est faite par l'intermédiaire de requêtes SQL. Un exemple des requêtes utilisées est présent dans l'annexe 1 de ce rapport. Il existait une limite de 50 000 posts par requête. Le choix a été fait de télécharger 100 000 posts pour chaque mois entre 2018 et 2021. Les posts les plus récents dataient de la fin du mois d'août 2021. Ensuite, 100 000 posts par an ont été téléchargés pour la période entre août 2008 (date de création du site) et 2018. L'intérêt de ce choix était d'obtenir une diversité des sujets de posts plus importante. En effet, il s'avère qu'en fonction du temps les questions des utilisateurs changent. Par exemple, dans les années 2010 beaucoup de questions sur Java étaient posées et plus récemment, ce sont des questions sur les langages de script comme Python ou JavaScript qui dominent. Une fois toutes nos données téléchargées au format csv, un script Python a permis de les ouvrir un à un pour former un seul fichier csv contenant plus de 2,5 millions de posts. Ce script est présenté dans l'annexe 2 de ce rapport.

Analyse exploratoire du corpus

Le corpus obtenu a été analysé de deux façons. Au début, une analyse des données numériques a été effectuée dans le but de voir la manière dont se distribuaient les posts. Cette analyse des données numériques a permis de déterminer des critères de choix pour sélectionner les posts les plus pertinents pour construire notre modèle. Après cette analyse et cette sélection des posts, les données textuelles ont été traitées afin de pouvoir les analyser avec des techniques propres au traitement naturel du langage.

Analyse des données numériques

Les données numériques sont peu nombreuses dans notre jeu de données. On dispose d'informations sur l'identification du post (Id du post, Id de la réponse acceptée, Id du créateur), de la date de création du post, du score attribué, du nombre de vues, de commentaires et de réponses.

Une analyse des données manquantes a montré l'absence d'identifiants de réponses acceptées sur près de la moitié des posts (43%). Ces posts ont été éliminés d'office. Pour éviter de se retrouver avec des données en doubles, un retrait des posts avec un identifiant déjà présent a été effectué. Après cette première sélection, il restait 1,4 million de posts.

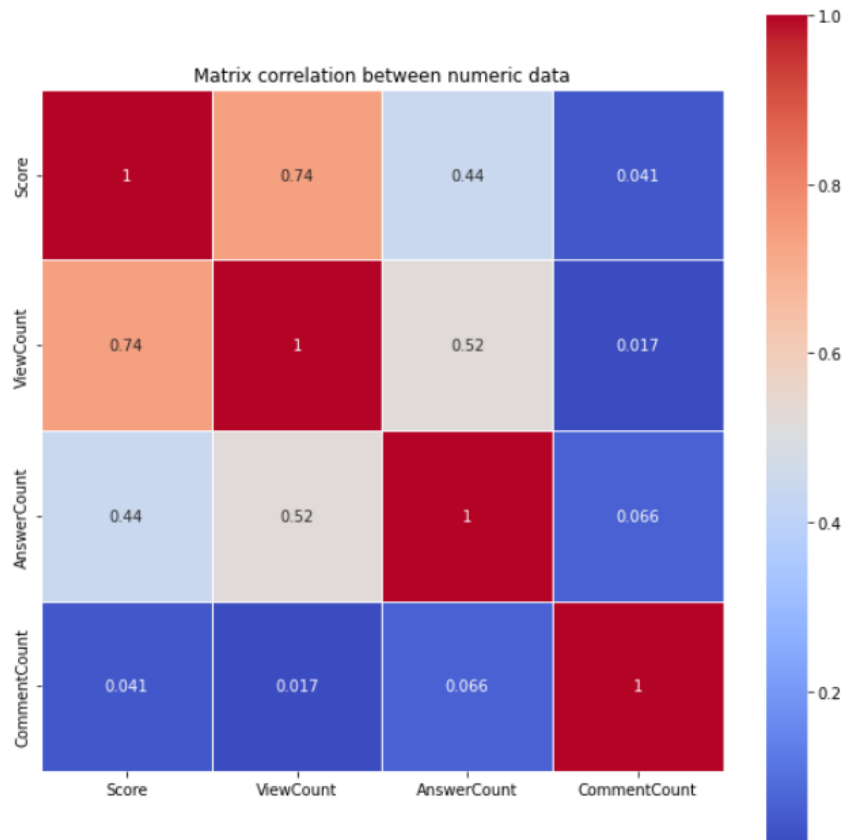
Pour affiner cette sélection, d'autres critères ont été mis en place.

Critères de sélection des posts

Les critères de sélection sont les suivants :

- Avoir une réponse acceptée
- Avoir au minimum 2 vues
- Avoir au minimum 2 commentaires
- Avoir au minimum un score de 2

Ces critères de sélection ont permis de réduire le nombre de posts à environ 165 000. Après avoir réalisé cette sélection, une analyse exploratoire sur les données quantitatives a été réalisée. Elle a montré une grande disparité en termes de nombre de vues, de commentaires et de scores sur nos posts ainsi qu'une forte corrélation entre le nombre de vues et le score attribué à un post.



Matrice de corrélations sur les données numériques des posts StackOverFlow

Preprocessing des posts restants

L'autre partie la plus importante dans cette étape concerne le nettoyage des données textuelles. En effet, les données textuelles sont des données non structurées qui nécessitent un prétraitement spécifique avant de pouvoir y appliquer des modèles de Machine Learning. Plusieurs fonctions pour permettre d'automatiser le prétraitement du texte ont été créées. Elles sont présentées dans l'annexe 3 de ce rapport. On va cependant détailler les différentes étapes qui ont été réalisées.

	Id	AcceptedAnswerId	CreationDate	Score	ViewCount	Body	OwnerUserId	Title	Tags	AnswerCount	CommentCount
0	415160	NaN	2009-01-06 02:00:17	7	6374	<p>What is the best method for creating an <a ...	51886.0	Best method of Instantiating an XMLHttpRequest...	<javascript><ajax><cross-browser><xmlhttprequest>	9	1

Exemple de données présentes dans le jeu de données créé

- On commence par **enlever** les **marqueurs typographiques du langage HTML**. L'observation des premières lignes du jeu de données montre que des marqueurs du langage HTML sont présents autour des corps de textes des posts.
- Les **caractères accentués** ont été **retirés**. On les retrouve dans certaines langues comme le français. Cependant vu que l'on travaille avec des posts rédigés en anglais, il ne doit pas avoir la présence de ces caractères accentués. Par la même occasion, toutes nos

chaînes de caractères subissent une transformation de leur encodage pour correspondre au standard **UTF-8**.

- Les **formes contractées** sont étendues. La langue anglaise possède la particularité d'avoir des formes contractées. Ces formes vont donc être étendues dans le but de ne pas les considérer comme des mots différents. Ainsi, *I'm* sera étendue en *I am*.
- Les **caractères spéciaux** sont **enlevés**.
- La **lemmatisation**, dans notre cas, est réalisée. Il s'agit d'une étape qui permet de ne conserver que la racine (le *lemma*) d'un mot. Ainsi les verbes sont tous mis sous la forme, les noms apparaissent sous la même forme. Il existe une autre opération semblable qui s'appelle la *stemmatisation*. On va chercher à conserver le mot sans ses affixes. Cependant, l'inconvénient de cette méthode est la perte de sens du mot¹.
- Les **stopwords** sont **retirés**. Il s'agit de mots communs dans la langue que l'on étudie qui n'apportent pas une information supplémentaire (liste des stopwords de la langue anglaise : <https://gist.github.com/sebleier/554280>). Pour la langue anglaise, il va s'agir de mots comme : *I, the, a, an, etc..*

La fonction qui permet de réaliser ces différentes étapes a été appliquée aux titres ainsi qu'au corps de texte de chaque post. Concernant les tags, le choix a été fait de n'enlever que les caractères qui entourent chaque tag. En effet, le retrait de caractères spéciaux comme des « + » ou des « # » peut entraîner une modification de la signification du tag. Par exemple, il existe une différence entre C, C++ et C#.

Visualisation des mots présents dans le corpus

Une fois ce nettoyage terminé, on va pouvoir passer à l'étape de visualisation des mots composant le titre, le corps de texte et les tags¹⁻³.

Visualisation de la composition des posts

Les titres, les corps de texte et les tags ont tous subi deux types de traitement différents pour pouvoir évaluer la composition de ces derniers. Les méthodes choisies pour chacun étaient Bag-Of-Word et Tf-IDF.

Le « Bag-Of-Word » (BoW ou sac de mots en français) est un dénombrement du nombre de mots présents dans le corpus de textes. À chaque fois que l'on va le voir apparaître, on va l'ajouter au compteur de mots. Ainsi on disposera d'une liste de mots (ou tokens) avec le nombre de fois où ils apparaissent dans le corpus de textes. Cependant, cette méthode a un inconvénient, elle ne prend pas en compte la rareté/fréquence du mot dans le corpus de textes⁴.

Pour pouvoir contrer les effets de la rareté/fréquence d'un mot, on va utiliser une autre méthode de dénombrement : le TF-idf (*term frequency-inverse document frequency*). Cette méthode permet de calculer une fréquence qui détermine l'importance de la présence d'un mot dans un corpus. Il y aura donc une pondération pour chaque occurrence de mots présents dans notre corpus. Cette méthode est plus précise que le Bag-Of-Word⁵.

$$w_{x,y} = tf_{x,y} \times \log \left(\frac{N}{df_x} \right)$$

Formule de Tf-IDF

Avec :

- $w_{x,y}$ le score de pertinence d'un terme x quelconques dans un document y (TF-IDF)
- $TF_{x,y}$ la fréquence du terme x dans le document y
- df_x le nombre de documents contenant x ;
- N le nombre total de documents.

L'analyse de la composition des posts est divisée en trois : le titre, le corps de textes et les tags.

Les titres

Une des représentations les plus présentes pour décrire les mots présents dans un corpus de texte est le nuage de mots. Voici le nuage de mots obtenus pour les titres.

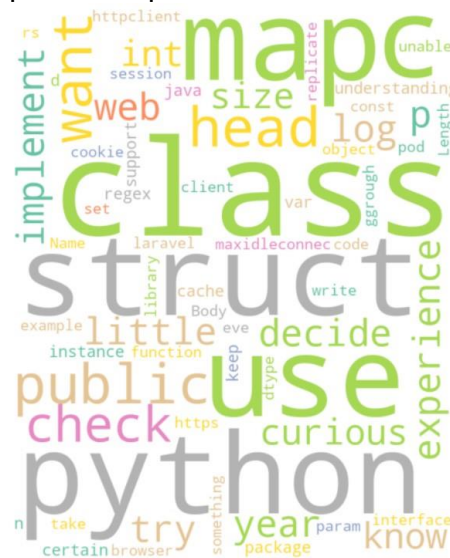


Nuage de mots des mots des titres des posts

Ce nuage de mots a été réalisé à partir des 1000 premiers mots les plus présents dans les titres. On distingue que certains éléments ressortent et donnent de bonnes indications sur le contenu des posts. *Python*, *Java*, *Laravel* et *Regex* semblent être les points sur lesquels les utilisateurs posent le plus de questions.

Le corps de texte (ou body)

Un nuage de mots a été réalisé pour le corps de texte.



Nuage de mots des mots des corps de textes des posts

Les résultats concernant le corps de texte sont plus hétérogènes. Cela s'explique par la présence de détails plus importants pour obtenir une réponse à un problème. On y trouve des verbes comme *want*, *know* ou *decide* qui peuvent donner une orientation sur les posts : on a bien affaire à des gens qui veulent faire quelque chose en partant d'une connaissance. On retrouve aussi des mots plus spécifiques au domaine informatique tels que *Python*, *Package*, *Library* ou *Laravel*. Certains mots peuvent aussi être des mots-clés de certains langages (*get*, *try*, *head*, etc.)

Les tags

Enfin, un nuage de mots a été réalisé dans le but de visualiser les tags les plus présents dans notre jeu de données.



Les tags les plus représentés vont dans le sens de ce que l'on a retrouvé dans les titres. On retrouve les mêmes mots comme *Python*, *Java* ou encore *Laravel*. Cela confirme que les titres contiennent des informations sur le contenu du post qui vont permettre un meilleur référencement. Il y a aussi d'autres mots comme *Language*, *cookies* ou *logging* qui vont être en lien avec des sujets plus généralistes.

Modélisation : Comparaison de différentes modélisations

Dans cette partie, une comparaison de différents types de modélisation va être faite. A l'issue de cette comparaison, le modèle le plus performant sera choisi pour être déployer dans une application. Le but étant de créer une API permettant de générer des suggestions de tags.

La première étape dans la modélisation est l'utilisation de **modèle non-supervisé** dans le but de faire de la réduction de dimensions. Ensuite, on passera à des **modèles supervisés** afin d'obtenir une solution déployable et pouvant **réaliser** de **nouvelles prédictions**.

Le problème auquel nous avons affaire est un problème de **classification multi-label**. Il faudra donc choisir des **métriques d'évaluation adapté** à ce type de problème.

Les modèles non-supervisés : Topic modelling

Le topic modelling fait partie des modèles non-supervisés^{6,7}. Il va aussi permettre de faire de la réduction de dimensions. Les algorithmes de topic modelling sont plus adaptés à cette tâche que les réductions de dimensions classiques comme PCA ou t-SNE. En analysant le corpus de textes, il va essayer de définir un nombre de thèmes (ou topic) défini au préalable. Pour cette partie, deux algorithmes ont été testés. Le critère d'évaluation pour ces modèles est le **score de cohérence**.

Non Negative Matrix Factorization (NMF)

Le modèle NMF est le premier qui a été testé. Cet algorithme décompose des vecteurs de grande dimension en vecteurs de plus petites dimensions.

Les vecteurs obtenus sont donc non négatifs. A

$$\begin{matrix} W \\ \left[\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \right] \end{matrix} \times \begin{matrix} H \\ \left[\begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline & & & & & \\ \hline & & & & & \\ \hline \end{array} \right] \end{matrix} \approx \begin{matrix} V \\ \left[\begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline & & & & & \\ \hline & & & & & \\ \hline \end{array} \right] \end{matrix}$$

Illustration du fonction de l'algorithme NMF

A partir du corpus de textes (V), une décomposition en deux autres matrices W et H va être faite : W correspondant aux topics et H aux poids/coefficients de chacun de ces topics⁸.

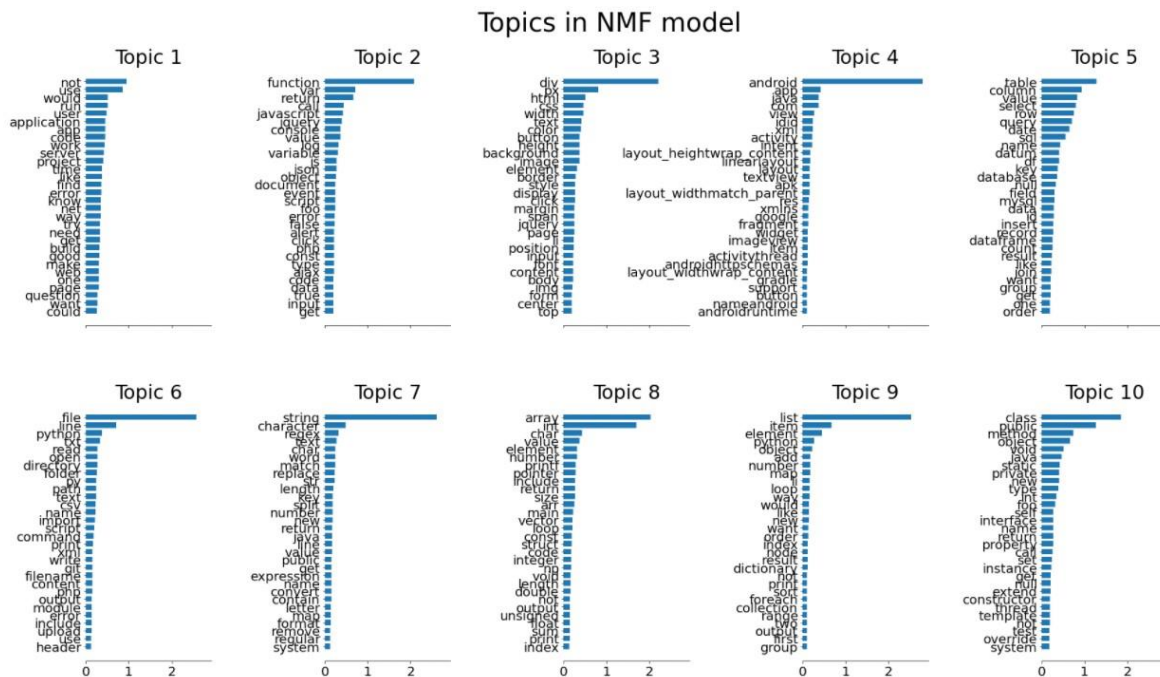
L'algorithme va donc modifier les valeurs de W et H pour s'approcher le plus possible des valeurs du corpus de texte original V.

NMF

```
NMF(alpha=0.1, n_components=10, random_state=42)
```

Paramètre de l'algorithme NMF utilisé

Dans notre cas, le nombre de topics a arbitrairement été fixé à 10. Le paramètre α correspond à la régularisation que l'on impose à notre modèle. Avant de réaliser cet algorithme, notre corpus a été vectorisé à l'aide de la méthode Tf-Idf. Voici les résultats obtenus à l'issue de la réalisation de cet algorithme.



Topic Modelling obtenu à l'issue de l'algorithme non supervisé NMF

Grâce à cette première réduction de dimension, on remarque que l'on peut dégager certains thèmes présents dans notre corpus. Le topic 1 correspond à des posts plutôt généraux, le topic 2 a du langage de script pour le web (*Javascript*), le topic 3 a du langage de markup pour le web, le topic 4 a du développement d'application mobile, le topic 5 a de la base de données, le topic 6 a du langage de script (Python), etc.

Latent Dirichlet Allocation (LDA)

L'autre méthode de topic modelling utilisée est le Latent Dirichlet Allocation^{9,10}. Il s'agit d'une méthode probabiliste qui va chercher à déterminer la probabilité d'appartenance d'un texte à un topic en fonction des mots présents dans le texte.

Cette technique va être répétée sur tous les textes du corpus pour en dégager des thèmes.

$$P(z_i = j | \mathbf{z}_{-i}, \mathbf{w}_i, \mathbf{d}_i, \cdot) \propto \frac{C_{w,j}^{WT} + \beta}{\sum_{w=1}^W C_{w,j}^{WT} + W\beta} \frac{C_{d,j}^{DT} + \alpha}{\sum_{t=1}^T C_{d,t}^{DT} + T\alpha}$$

Probability of word \mathbf{w} under topic \mathbf{t}

Probability of topic for a word in a document

Probability of topic \mathbf{t} in document \mathbf{d}

Formule de l'algorithme LDA (source : <https://thinkinfi.com/latent-dirichlet-allocation-explained/>)

Avant de réaliser le LDA, une vectorisation du texte a été fait à l'aide de la méthode Bag-Of-Words. Pour créer ce modèle, une recherche des hyperparamètres les plus pertinents a été réalisée à l'aide d'une GridSearch.

```
GridSearchCV
GridSearchCV(cv=5,
             estimator=LatentDirichletAllocation(batch_size=400,
                                                  random_state=42),
             param_grid={'learning_decay': [0.5, 0.7, 0.9],
                         'learning_method': ['batch', 'online'],
                         'n_components': [8, 9, 10]},
             verbose=2)

LatentDirichletAllocation
LatentDirichletAllocation(batch_size=400, random_state=42)
```

Paramètres recherchés à l'aide de la méthode GridSearchCV

Les paramètres qui ont été testés sont le nombre de topics, la méthode d'apprentissage et un paramètre de contrôle du taux d'apprentissage dans le cas d'une méthode d'apprentissage en direct (*online*). A l'issue de cette recherche, l'algorithme était appliqué sur notre corpus de texte. Voici les paramètres retenus pour le LDA : Best Model's Params: {'learning_decay': 0.9, 'learning_method': 'online', 'n_components': 8}. Les topics obtenus montrent une hiérarchie dans la distribution des thèmes.

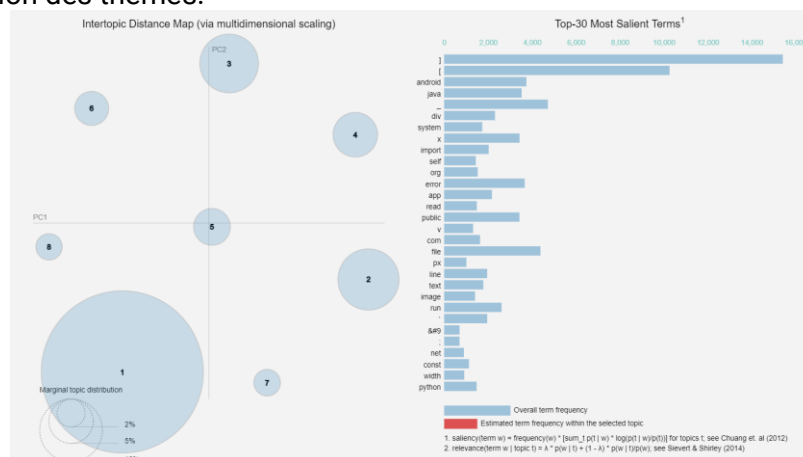


Illustration de la visualisation obtenue après LDA avec l'outil pyLDAvis

On peut les classer comme ceci :

- Topic 1 : Top topic - Generalities
- Topic 2 : Web Language/Front-End
- Topic 3 : Programming Language/Portable Language (Java, Android)
- Topic 4 : Script Language (Python, Javascript)
- Topic 5 : OS (Windows, errors)
- Topic 6 : Web Language/Specific functions

- Topic 7 : Exemples
- Topic 8 : Application utilities (geometry, physics)

Pour plus de détails, une représentation interactive est présente dans le notebook P05_02_notebooktest.

Evaluation du topic modeling

Pour comparer ces deux méthodes de réduction de dimension, le score de cohérence a été calculé pour chacune. La cohérence est une approche quantitative qui va calculer la vraisemblance conditionnelle entre les mots présents dans nos différents topics. Contrairement à la perplexité, il prend en compte la sémantique.

$$\text{Coherence} = \sum_{i < j} \text{score}(w_i, w_j)$$

Formule de calcul de la cohérence pour évaluer le topic modeling

Les résultats obtenus sont les suivants :

- Cohérence NMF : 0.414
- Cohérence LDA : 0.416

Les deux algorithmes se montrent performant dans la réalisation d'une dimension de réduction. Cependant, le LDA peut prédire l'appartenance d'un nouveau texte à un topic déjà défini sans avoir besoin de le réentraîner ce que ne peut pas faire le NMF.

Si on devait se baser sur une unique méthode de réduction de dimension, il serait donc intéressant de conserver le LDA.

Conclusion

L'utilisation d'algorithme de topic modeling a permis de voir les thèmes qui se dégagent du corpus de texte à notre disposition. Si on se base sur les résultats de l'algorithme LDA, on peut distinguer 8 thèmes qui vont créer une hiérarchie allant d'un thème très général en passant par des langages avec des applications et des exemples. Cette hiérarchie est présentée dans l'annexe 4 de ce rapport.

Les modèles supervisés : du Machine Learning « classique » au Deep Learning

La méthode d'évaluation

Avant de présenter les différents modèles choisis, on va définir les méthodes d'évaluation qui ont été sélectionnées. La classification que l'on doit effectuer est une classification multi-label, c'est-à-dire qu'un post pouvait se voir attribuer un ou plusieurs tags en fonction de son contenu. Il faut donc adapter l'évaluation pour qu'elle corresponde à ce type de tâche^{11,12}.

Nom de la mesure	Formule	Définition
Hamming Loss	$\text{Hamming Loss} = \frac{1}{nL} \sum_{i=1}^n \sum_{j=1}^L [I(y_i^{(j)} \neq \hat{y}_i^{(j)})]$ <p>Where, $n \Rightarrow$ Number of training examples $y_i^{(j)} \Rightarrow$ true labels for the ith training example and jth class $\hat{y}_i^{(j)} \Rightarrow$ predicted labels for the ith training example and jth class</p>	Ratio de mauvaise classification
Hamming Score (ou LBA)	1 – Hamming Loss	Précision du modèle en moyennant chaque catégorie
Subset Accuracy	$A = \frac{1}{n} \sum_{i=1}^n \frac{ Y_i \cap Z_i }{ Y_i \cup Z_i }$	Précision du modèle toute catégorie confondue sans pondération
Score F1 micro	$\text{Micro-F-Score} = 2 \cdot \frac{\text{Micro-Precision} \cdot \text{Micro-Recall}}{\text{Micro-Precision} + \text{Micro-Recall}}$	Balance entre précision et sensibilité, pondération selon les classes. Prend en compte les déséquilibres de classes

Tableau récapitulatif des méthodes de scoring utilisé

Lors de la réalisation d'optimisation des hyperparamètres, la méthode de scoring se basait sur le score F1 micro.

Les méthodes de Machine Learning

Les premiers essais ont été réalisés avec deux modèles de machine learning « classique » : un classifieur de la famille des Naïve Bayes (**NBMultinomial**) et un **support vecteur machine**. Dans les deux cas, les modèles ont été paramétrés de manière à utiliser une méthode *OneVsRest*. Dans cette configuration, on va entraîner autant de classifieur qu'il y a de tags. Pour des raisons de contraintes matérielles, seul le classifieur Naïve Bayes a eu le droit à une recherche d'hyperparamètres. Cependant, on remarque que sans optimisation le modèle de support vecteur machine obtient de meilleurs scores.

Algorithme de machine learning pour le multilabel

Suite à cet essai, un modèle spécialisé dans la classification multi-label a été sélectionné et entraîné. Il s'agit de **MLkNN** qui est une adaptation du classifieur kNN habituel. Cet algorithme sera celui déployé dans notre application et sera expliqué plus en détails dans une autre partie.

Deep Learning et plongement (embedding) de mots

Enfin, des essais de modélisation via des modèles de Deep Learning ont été tentés. La méthode utilisée combinait un plongement de mots via GloVe^{13,14} à un modèle de réseaux de neurones récurrents : LSTM^{15,16}. Il s'avère que ce modèle n'est pas le plus adapté à la tâche que l'on cherche à effectuer.

Présentation de l'algorithme MLkNN

L'algorithme **MLkNN** (pour **MultiLabel K Nearest Neighbors**) est une adaptation de l'algorithme kNN classique.

Initialement, la méthode des k plus proches voisins est l'un des algorithmes les plus intuitifs et les plus simples du Machine Learning. Il peut être utilisé pour des problèmes de classification comme des problèmes de régression. Il va s'agir de sélectionner un nombre k, de calculer la distance qui sépare le point à classifier par rapport aux autres points. Ensuite, on va sélectionner les k voisins les plus proches en fonction de la distance calculée. Puis on va compter le nombre de points dans chaque catégorie. Enfin, on va attribuer la catégorie la plus présente à notre point non classifié en fonction de ces k voisins.

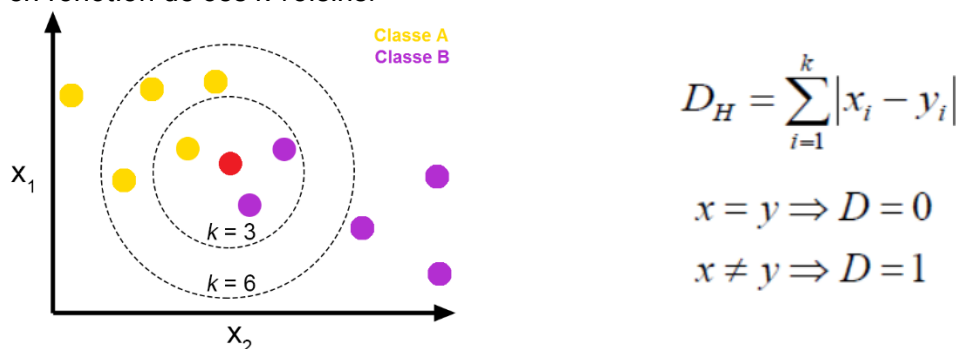


Illustration du principe de l'algorithme de kNN dans une classification classique (source : <https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d>) + formule de la distance de Hamming

Dans sa version adaptée au multilabel, cet algorithme va se baser sur les informations statistiques des points présents autour de lui pour maximiser la vraisemblance de la distribution a priori de nos paramètres. Cette méthode est plus efficace que les méthodes d'adaptation ou encore des méthodes de classifieurs de chaînes¹⁷.

A l'aide des paramètres par défaut de ce modèle, on obtient des scores supérieurs aux autres modèles. Pour pouvoir déployer un modèle plus performant, plusieurs paramètres ont été testés. On s'est intéressé au paramètre k qui a été testés pour des valeurs allant de 6 à 11 et au paramètre s. Le paramètre s correspond à une variable de lissage. Deux valeurs ont été testées : 0.7 et 1.0.

Tableau de synthèse

Nom du modèle	Famille du modèle	Description du modèle	Preprocessing	Recherche utilisé	Hyperparam.	F1-Score
Naïve Bayes Multinomial ¹⁸	Machine Learning – méthode probabiliste	Modèle probabiliste qui calcule la probabilité qu'un texte appartienne à une classe (ici un tag)	Vectorisation avec Tf-IDF + Ajout de la méthode <i>OneVsRest</i> pour adapter l'algorithme au multilabel	Grille de recherche avec validation croisée (GridSearchCV ¹⁹)	$\alpha = 0.3$	0.046
Support Vecteur Machine ^{20,21}	Machine Learning – méthode linéaire (et non-linéaire avec noyau)	Modèle calculant un hyperplan s'appuyant sur des points (supports) pour classer des individus non labellisés	Vectorisation avec Tf-IDF + Ajout de la méthode <i>OneVsRest</i> pour adapter l'algorithme au multilabel	-	Par défaut	0.21
MLkNN ¹⁷	Machine Learning adaptée au multilabel	Adaptation du kNN. cf paragraphe précédent	Vectorisation avec Tf-IDF	Grille de recherche avec validation croisée	k = 7 s = 0.7	0.27
Long Short Term Memory networks ^{15,16}	Deep Learning – Réseau de neurones récurrents	Réseau de neurones récurrents qui prend en compte le maintien de l'information à long terme ou non. Particulièrement efficace dans le cas de séries temporelles	Embedding avec GloVe ^{13,14}	Optimisation bayésienne	352 neurones (cf Annexe 5)	0.0013

Tableau récapitulatif des méthodes utilisées pour créer notre modèle de catégorisation de questions

Références

1. Stemming and lemmatization. <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>.
2. Sarkar, D. (DJ). A Practitioner's Guide to Natural Language Processing (Part I) — Processing & Understanding Text. *Medium* <https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72> (2018).
3. Essential Text Pre-processing Techniques for NLP! *Analytics Vidhya* <https://www.analyticsvidhya.com/blog/2021/09/essential-text-pre-processing-techniques-for-nlp/> (2021).
4. Brownlee, J. A Gentle Introduction to the Bag-of-Words Model. *Machine Learning Mastery* <https://machinelearningmastery.com/gentle-introduction-bag-words-model/> (2017).
5. Scott, W. TF-IDF for Document Ranking from scratch in python on real world dataset. *Medium* <https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089> (2021).
6. Topic extraction with Non-negative Matrix Factorization and Latent Dirichlet Allocation. *scikit-learn* https://scikit-learn/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.html.
7. Beliga, S., Meštrovi, A. & Martin, S. An Overview of Graph-Based Keyword Extraction Methods and Approaches. 39, 20 (2015).
8. Salgado, R. Topic Modeling Articles with NMF. *Medium* <https://towardsdatascience.com/topic-modeling-articles-with-nmf-8c6b2a227a45> (2020).

9. Ganegedara, T. Intuitive Guide to Latent Dirichlet Allocation. *Medium*
<https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158> (2020).
10. Kulshrestha, R. Latent Dirichlet Allocation(LDA). *Medium*
<https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2> (2020).
11. Jadhav, P. A Survey of Evaluation Metrics for MultiLabel Classification. *Medium*
<https://medium.datadriveninvestor.com/a-survey-of-evaluation-metrics-for-multilabel-classification-bb16e8cd41cd> (2021).
12. Pahwa, R. Micro-Macro Precision, Recall and F-Score. *Medium*
<https://medium.com/@ramit.singh.pahwa/micro-macro-precision-recall-and-f-score-44439de1a044> (2017).
13. GloVe: Global Vectors for Word Representation.
<https://nlp.stanford.edu/projects/glove/>.
14. Pennington, J., Socher, R. & Manning, C. Glove: Global Vectors for Word Representation. in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* 1532–1543 (Association for Computational Linguistics, 2014). doi:10.3115/v1/D14-1162.
15. Understanding LSTM Networks -- colah's blog.
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
16. Dive into Deep Learning — Dive into Deep Learning 0.17.0 documentation.
<https://d2l.ai/index.html>.
17. Zhang, M.-L. & Zhou, Z.-H. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition* 40, 2038–2048 (2007).
18. Naive Bayes text classification. <https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html>.

19. `sklearn.model_selection.GridSearchCV`. *scikit-learn* https://scikit-learn/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.
20. Platt, J. C. Probabilistic Outputs for Support Vector Machine and Comparisons to Regularized Likelihood Methods. *Microsoft Research* (1999).
- 21.1.4. Support Vector Machines. *scikit-learn* <https://scikit-learn/stable/modules/svm.html>.

Annexes

Annexe 1 : Exemple de requête SQL

```
-- Queries about Posts

-- August 2021
SELECT Id, AcceptedAnswerId, CreationDate, Score,
       ViewCount, Body, OwnerUserId, Title, Tags, AnswerCount, CommentCount
FROM Posts
WHERE ViewCount > 0
      AND AnswerCount > 0
      AND CommentCount > 0
      AND CreationDate BETWEEN '2021-08-01' AND '2021-08-29'
ORDER BY CreationDate DESC;

-- July 2021
SELECT Id, AcceptedAnswerId, CreationDate, Score,
       ViewCount, Body, OwnerUserId, Title, Tags, AnswerCount, CommentCount
FROM Posts
WHERE ViewCount > 0
      AND AnswerCount > 0
      AND CommentCount > 0
      AND CreationDate BETWEEN '2021-07-01' AND '2021-07-31'
ORDER BY CreationDate DESC;
```

Annexe 2 : Code d'automatisation pour la création d'un CSV unique

```
import glob, os
import pandas as pd

os.chdir("datasets")
extension = 'csv'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
#combine all files in the list
combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames])
#export to csv
combined_csv.to_csv("combined_csv.csv", index=False, encoding='utf-8')
```

Annexe 3 : Fonctions de pre-processing des données textuelles

```
import spacy
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize.toktok import ToktokTokenizer
import re
from bs4 import BeautifulSoup
from contractions import CONTRACTION_MAP
import unicodedata
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, hamming_loss

"""Functions are found in this article
https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72"""

#nlp = spacy.load('en_core_web_sm')
#nlp_vec = spacy.load('en_vecs', parse = True, tag=True, #entity=True)
tokenizer = ToktokTokenizer()
stopword_list = nltk.corpus.stopwords.words('english')
stopword_list.remove('no')
stopword_list.remove('not')

def strip_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text()
    return stripped_text

def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text

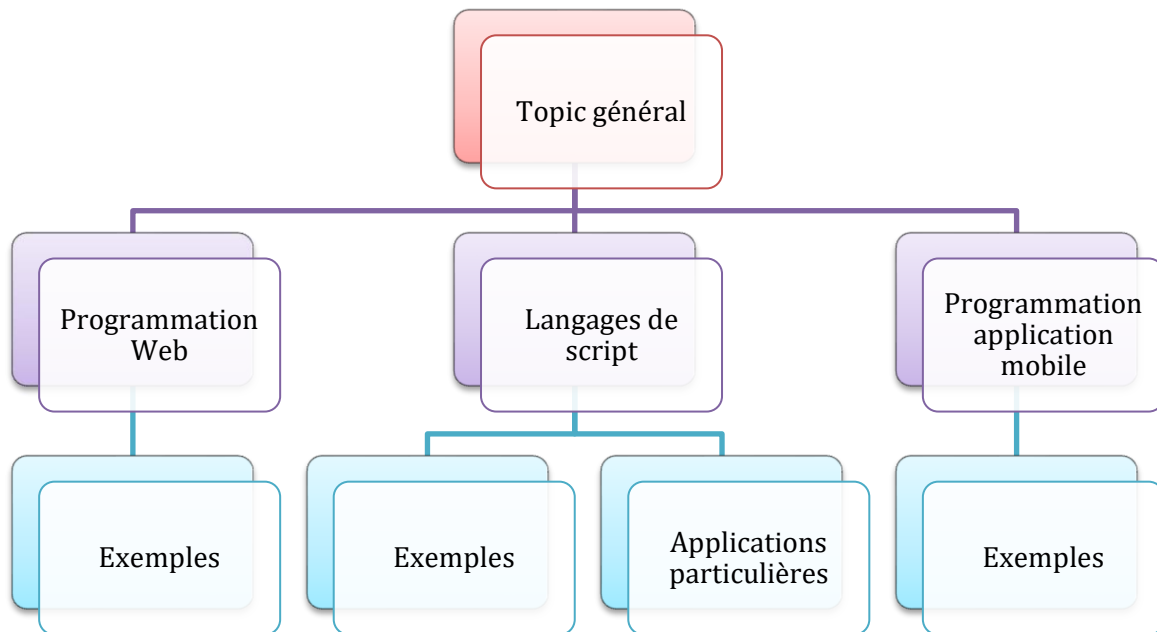
def expand_contractions(text, contraction_mapping=CONTRACTION_MAP):

    contractions_pattern = re.compile('{}'.format('|'.join(contraction_mapping.keys()))),
    flags=re.IGNORECASE|re.DOTALL)

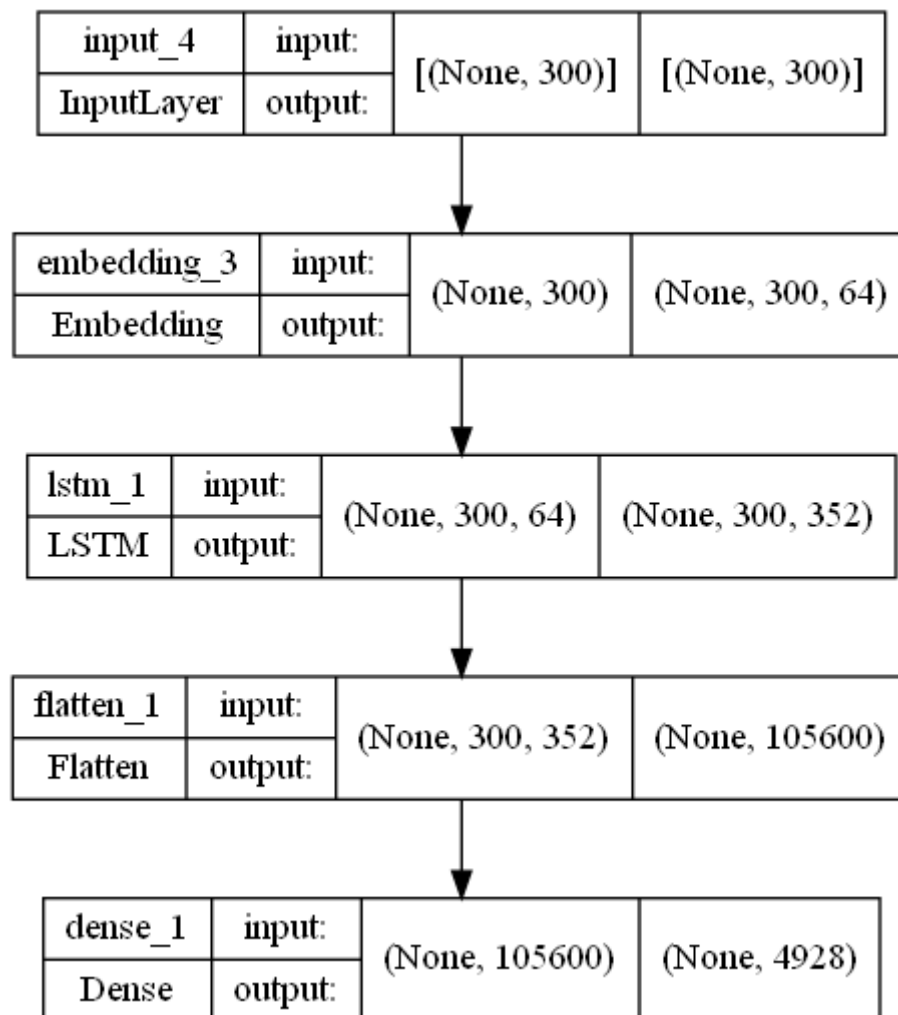
    def expand_match(contraction):
        match = contraction.group(0)
        first_char = match[0]
        expanded_contraction = contraction_mapping.get(match)\
            if contraction_mapping.get(match)\
            else contraction_mapping.get(match.lower())

        expanded_contraction = first_char+expanded_contraction[1:]
        return expanded_contraction
```

Annexe 4 : Hiérarchie des topics selon le LDA

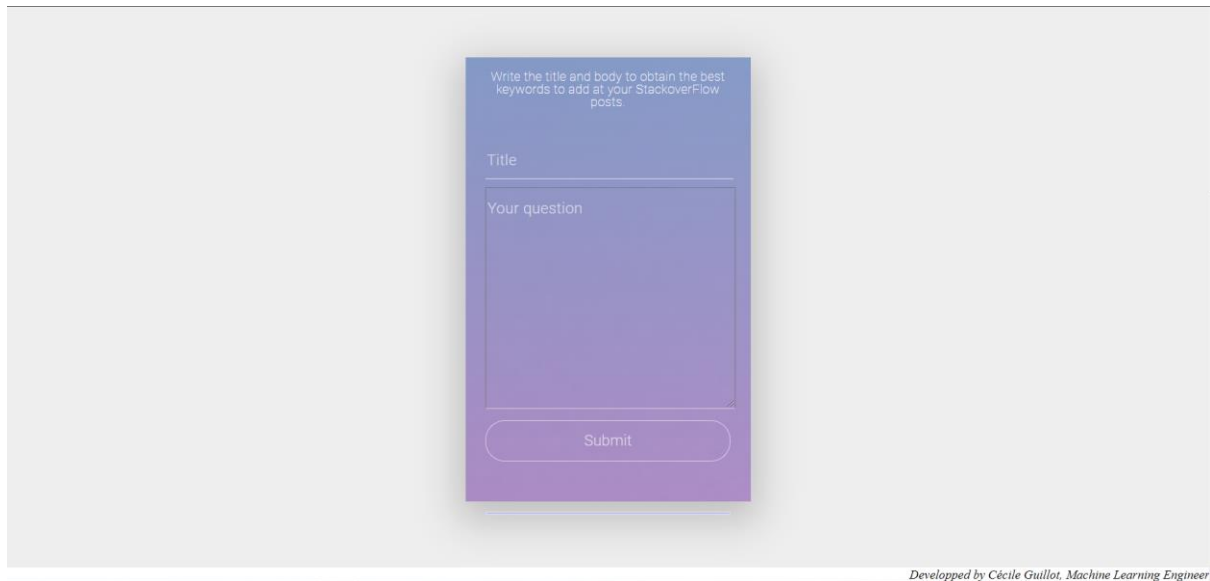


Annexe 5 : Architecture du réseau de neurones LSTM après optimisation bayésienne



Annexe 6 : Endpoint de l'API déployé

<https://tag-generator-stackoverflow.herokuapp.com/>



The image shows a web application interface for generating Stack Overflow tags. It features a purple gradient card centered on a light gray background. The card contains the following elements:

- Instructional text: "Write the title and body to obtain the best keywords to add at your Stackoverflow posts."
- A "Title" label above a text input field.
- A "Your question" label above a larger text area for the question body.
- A "Submit" button at the bottom of the card.

At the bottom right of the page, below the card, is the text: *Developped by Cécile Guillot, Machine Learning Engineer*.

Annexe 7 : Repo GitHub hébergeant l'application

https://github.com/Sylvariane/categorisation_de_questions/tree/app-deploiement