CATEGORISATION AUTOMATIQUE DES QUESTIONS DE STACKOVERFLOW

Février 2022

Thibaud GROSJEAN
Thibaud.grosjean@gmail.com

SOMMAIRE

Introduction

Crée en 2008 par Jeff Atwood, *StackOverflow* est une plateforme (site web) d'entraide dédiée à la programmation informatique. Communautaire, la plateforme permet à ses utilisateurs de poser des questions techniques sur les diverses technologies de programmation et d'y répondre.

Similaire au système mis en place par *Reddit* (plateforme d'actualité communautaire), le vote des utilisateurs permet de mettre en avant les questions et les réponses les plus qualitatives. Un système de réputation permet par ailleurs aux utilisateurs d'être récompensés pour leurs contributions grâce à des badges.

Plébiscitée par plus de 14 millions d'utilisateurs inscris, qu'ils soient des programmeurs confirmés ou débutants, le site recense plus de 20 millions de questions et plus de 30 millions de réponses.

QUESTION : DEFINITION

Tout utilisateur enregistré peut poser une question sur *StackOverflow* par le biais de l'interface. Pour cela, il dispose de trois boîtes de texte à compléter :

- Le titre de la question,
- Le corps de la question, qu'il peut mettre en forme grâce à un éditeur de texte et dans lequel il a la possibilité d'inclure du code,
- Un à cinq *tags* associés à la question.

StackOverflow a mis en place des didacticiels afin d'inciter les utilisateurs à écrire leurs questions de la manière la plus pertinente possible.

TAGS : DEFINITION

Le *tag*, ou étiquette (en français) est un mot-clé, une métadonnée (une donnée qui décrit les caractéristiques d'une donnée). Dans le cadre de *StackOverflow*, les *tags* permettent aux utilisateurs de décrire le contenu de leurs questions afin de les référencer correctement.

Les *tags* permettent ainsi à la plateforme de catégoriser les questions en fonction des sujets (technologies) abordés. Cela permet aux autres utilisateurs de trouver efficacement les questions qui les intéressent.

LES DONNEES

Les données de *StackOverflow* sont disponibles librement, tout utilisateur, même non enregistré, peut effectuer des recherches sur la plateforme, lire les questions et leurs réponses.

De plus, il est possible d'obtenir des extractions de la base de données *SQL* de *StackOverflow* sous la forme de *fichiers tabulaires* (*.csv*). La base de données (dont la table *Posts* – qui référence les questions – est en lecture seule) est requêtable par le biais de l'outil *StackExchange data explorer* qui génère des fichiers qui comportent jusqu'à 50.000 entrées. Un schéma de la base de données est mis à disposition.

PROBLEMATIQUE

Actuellement, les tags associés aux questions sont choisis par les utilisateurs eux-mêmes, et *StackOverflow* propose un système d'auto-complétion (taper les premiers caractères alphanumériques permet d'obtenir une liste des tags les plus utilisés).

En revanche, la plateforme ne dispose pas d'un système de suggestion de *tags*, un système qui suggérerait des *tags* de manière automatique, en fonction du contenu de la question.

SOLUTION PROPOSEE

Pour répondre à cette problématique, nous allons mettre en place un système de recommandation de *tags*. Pour se faire, nous avons :

- Extrait les données afin de constituer un jeu de données,
- Appliqué un prétraitement des données textuelles,
- Exploré le jeu de données afin de nous l'approprier,
- Appliqué une transformation des données afin d'entraîner les modèles,
- Entrainé des modèles de *machine learning*,
- Comparé et sélectionné le modèle le plus approprié.

Pour la modélisation, nous mettrons en œuvre des approches supervisées, et non supervisées. Nous mettrons un *endpoint* d'API à disposition afin de permettre au public d'obtenir des prédictions (suggestions de *tags*).

EXTRACTION DES DONNEES

Nous avons utilisé l'outil *StackExchange* data explorer pour extraire des questions de *StackOverflow* afin d'entraîner des modèles de *machine learning*. La requête utilisée est la suivante :

```
SELECT TOP(50000) Id, Title, Body, Tags
From Posts
WHERE PostTypeId = 1
AND LEN(Tags) - LEN(REPLACE(Tags, '<','')) >= 5
AND Score >= 5
ORDER BY CreationDate
```

Afin d'obtenir des questions qualitatives, nous avons sélectionné des questions avec un score minimal de 5, et qui comportent 5 *tags* afin d'optimiser l'entrainement de nos modèles. Nous avons obtenu un fichier composé de 50.000 entrées.

DESCRIPTION DU JEU DE DONNEES

Nous disposons de 3 variables brutes :

- Le titre (*Title*) : il est constitué du texte brut entré par l'utilisateur,
- Le corps de la question (*Body*) : il inclut du code *html* de mise en forme, ce qui demandera un traitement particulier avant de pouvoir être traité avec des outils classiques de NLP (Natural Language Processing).
- Les tags pour chaque ligne (individu) du jeu de données, les tags contiennent des séparateurs (*« < » et « > »*) et nous avons dû appliquer une *tokenization*.

PRETRAITEMENT DES DONNEES

L'objectif du prétraitement des données textuelles du *corpus* (ensemble des *documents* contenus dans le jeu de données) est d'extraire des données brutes les informations les plus essentielles de chaque *document (*individu du jeu de données : en l'occurrence, une question composée d'un titre et d'un corps). Il convient dès lors de conceptualiser ce qu'est une question de *Stack Overflow*.

Exemple de question présente dans le jeu de données : *« How to decode HTML entities in Rails 3? »*

Nous faisons l'hypothèse qu'une question de *StackOverflow* vise à trouver une solution à un problème de programmation. Par conséquent, les termes les plus essentiels d'une question devraient se trouver en ses verbes - qui traduisent l'action que l'utilisateur souhaite réaliser programmatiquement - et ses noms, et noms propres qui traduisent les outils, *technologies* utilisées. Par conséquent, nous sélectionnerons ces types de termes durant le prétraitement.

Enfin, il convient dès à présent de différencier les données qui serviront à prédire (*X* – le titre et corps de la question) et les données que nous souhaitons prédire (*y* – les *tags*). Le partitionnement des données en set d'entraînement et de test se fera ultérieurement au prétraitement des données, en revanche, le prétraitement sera différencié pour *X* et *y*.

Pour *X*, nous avons appliqué les transformations suivantes à l'aide de la librairie *NLTK* :

- *Extraction du texte du corps de la question* : pour le corps de la question, qui intègre du code *html*, nous avons procédé à l'extraction du texte à l'aide de la librairie *BeautifulSoup*, de plus, nous avons supprimé les citations qui servent d'exemples de code (présentes entre les balises *html <code>*).
- *Tokenization* : la *tokenization* permet de découper un document en « *tokens* », une liste de termes, afin de faciliter leur traitement ultérieur. Nous avons utilisé une *tokenization* simple, qui se base sur les espaces pour séparer chaque terme,
- *Lemmatization* : la *lemmatization* permet d'extraire la racine des différents termes afin, entre autres, de diminuer la *dimensionnalité* des données.
- *Stop words* : ce sont des mots les plus communs, avec un sens non significatif, nous les avons supprimés.
- *POS Tagging* : le *POS (part-of-speech) tagging* ou étiquetage morpho-syntaxique permet de détecter d'obtenir des informations grammaticales pour chaque terme extrait. Nous avons décidé de ne garder que les verbes, noms et noms propres, afin de respecter notre hypothèse.
- *Concatenation du titre et du corps de la question* : afin de constituer X, nous avons concaténé le titre et le corps de la question.

Pour la cible *y*, nous avons simplement appliqué une *tokenization*. Nous n'avons pas appliqué de *lemmatization*, afin de générer des *tags* non tronqués avec les modèles de *machine learning*. L'ensemble de ces transformations sont génériques et ne posent donc pas de risque de *fuite de données* (*data leak*).

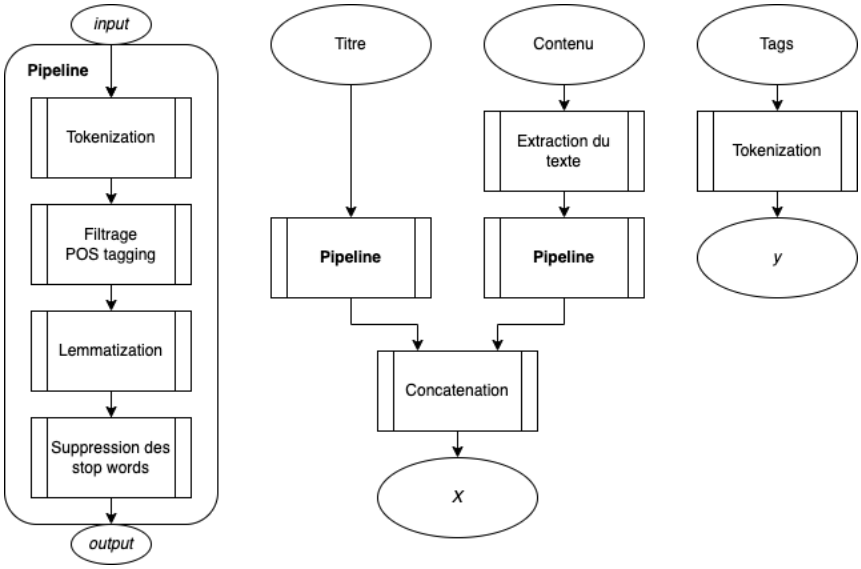
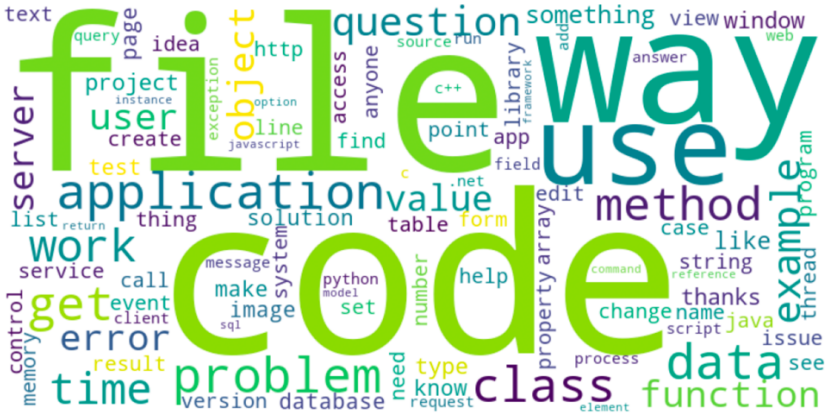


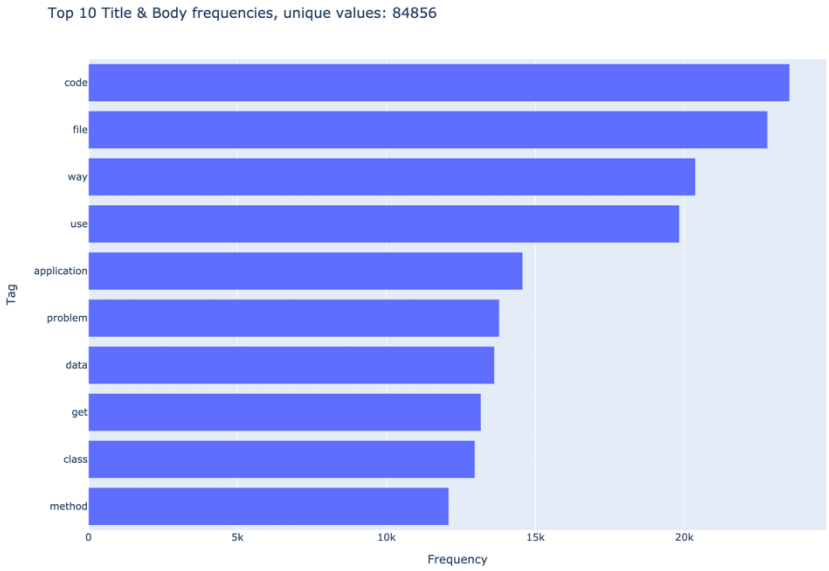
Schéma du pipeline de prétraitement

EXPLORATION DES DONNEES

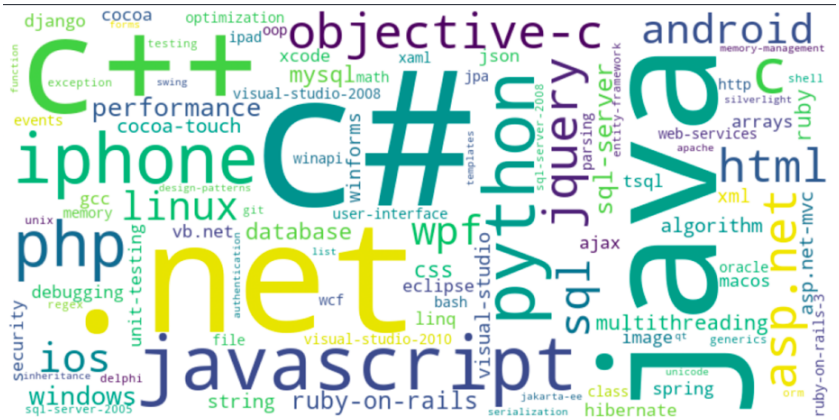
Les données ayant été nettoyées, nous avons calculé les fréquences des termes présents dans les documents du *corpus* (la jointure des titres et des contenus) ainsi que dans les *tags*.



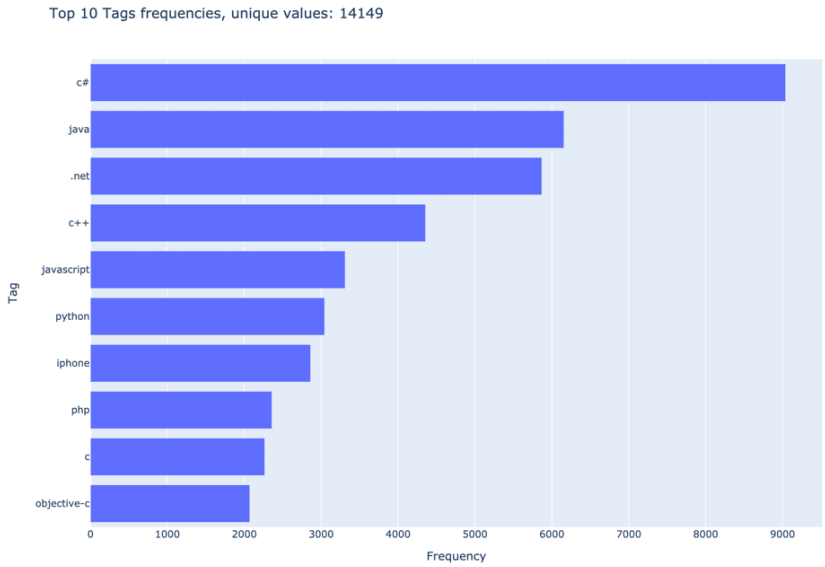
Nuage des premiers termes de X



Fréquences des premiers termes de X



Nuage des premiers termes de y



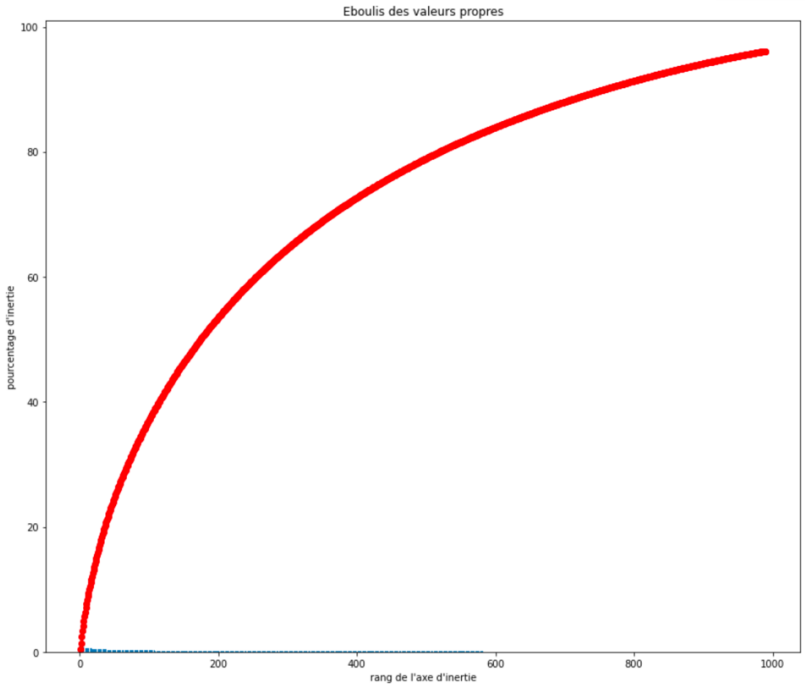
Fréquences des premiers termes de y

TRAITEMENT DU JEU DE DONNEES

Afin d'entrainer les modèles supervisés, et d'évaluer les prédictions des modèles supervisés, et non-supervisés, nous avons effectué des transformations supplémentaires du jeu de données. Pour se prémunir de la *fuite de données*, un partitionnement des données a été effectué et les données du set d'entraînement ont servi de référence pour effectuer les transformations sur le set de test, notamment durant l'utilisation des *transformers* de la librairie *Scikit-Learn*.

- *Split (partitionnement) des données* : 80% des données sont placées dans un set d'entraînement et 20% dans le set de validation.
- *Filtrage des termes les plus fréquents* : afin de limiter la *dimensionnalité* et de limiter les temps d'entraînement, nous avons décidé de filtrer les termes et *tags* les plus fréquents. La constitution des termes et tags les plus fréquents ont réalisée sur le *set d'entraînement*, avant d'être utilisé pour filtrer les données du *set de test*,
- *Vectorisation de X* : nous avons appliqué un *TfidVectorizer* afin de générer des sacs de mots (*bag of words*) et de calculer leurs importances dans l'ensemble du corpus grâce à la fréquence inverse de document (*inverse document frequency*).
- *Binarization de y* : nous avons appliqué un *MultiLabelBinarizer* afin de générer une matric des *tags* dans *y.*
- *Réduction de dimensionnalité* : la *vectorization de X* ayant généré une matrice *numpy sparse*, nous avons appliqué un *TruncatedSVD* (l'équivalent d'un *PCA*) afin d'effectuer la *réduction de dimensionnalité*. Nous avons délibérément over*fit* le *TruncatedSVD* en atteignant un pourcentage d'inertie (*explained variance ratio*) de 96% afin de perdre un minimum d'information (on considère généralement qu'une *réduction de dimensionalité* est *fit* à 80%).

Le *pipeline* appliqué sur *y* sera utilisé pour inverser les transformations sur les prédictions des modèles afin d'obtenir des résultats lisibles par l'homme.



Éboulis des valeurs propres du TruncatedSVM appliqué sur X

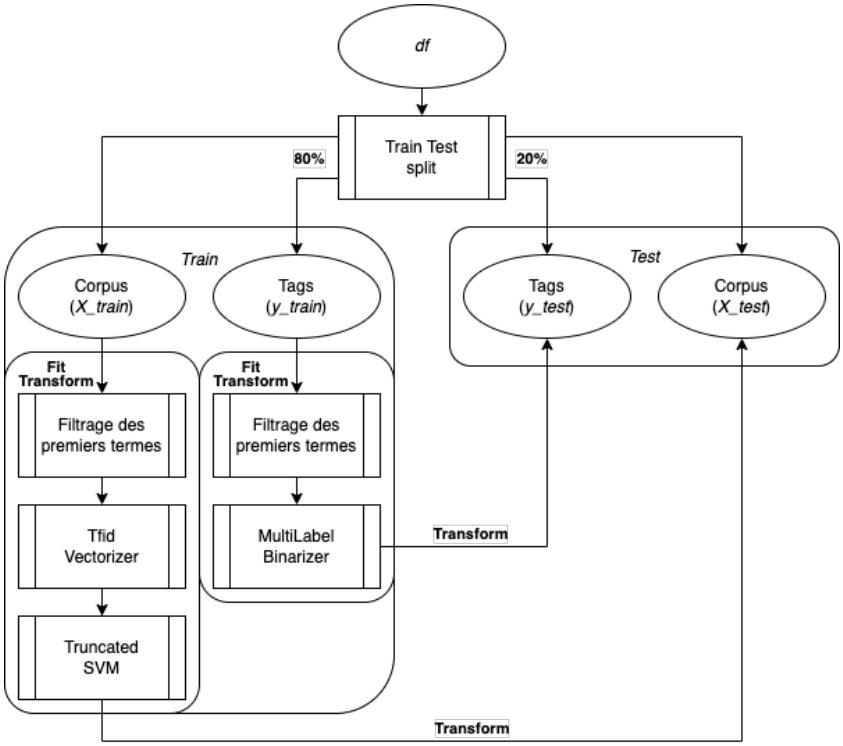


Schéma du pipeline de traitement

Disposant d'une puissance de calcul limitée, nous avons choisi de générer un *set X* composé de 1250 variables, réduites à 990 (variance expliquée du *TruncatedVSD* : 96%), et un *set y* composé de 250 *tags*.

MODELISATION

Générer des prédictions de suggestions de *tags*, dans le cadre du *machine learning supervisé*, traduit une problématique *classification multiple* : il s'agit de catégoriser chaque *document* par une ou plusieurs *classes* : les *tags*. De plus, comme nous avons pu le voir précédemment, les classes sont déséquilibrées, ce qui demande une attention particulière pour la sélection des métriques.

Nous avons testé plusieurs algorithmes :

- *Régression Logistique* : la régression logistique est une régression linéaire qui utilise la fonction logistique comme fonction de lien,
- *SVM* : le séparateur à large marge (*Support Vector Machine*) optimisent la distance entre les *individus* d'un jeu de données afin de les classifier,
- *SGD* : le *gradient stochastique* (*Schocastic Gradient Descent*) optimise la classification du *SVM* grâce à l'application de l'algorithme du *gradient stochastique*,
- *KNN* : l'algorythme des *plus proches voisins* (*K Nearest Neighbors*) rapproche les individus du jeu données ayant des caractéristiques similaires,
- *RFC* : la *forêt aléatoire* (*Random Forest*) crée un arbre décisionnel afin de classifier les individus,
- *LDA* (*machine learning non supervisé*) : la *Latent Dirichlet Allocation* attribue de manière probabiliste des *thèmes* (topics) en fonction de la présence de *termes* dans chaque *document*. Nous avons utilisé l'algorithme de la librairie *gensim*.

Afin de générer des prédictions multiples avec des classificateurs simples (tels que la *Régression Logistique*, le *SVM,* le *SGD*), nous avons utilisé la technique du *One vs Rest* (ou *One vs* All) grâce à la méthode éponyme de *de Scikit-Learn.* Cette approche permet d'entraîner un *classificateur binaire* pour

chaque *tag* du jeu de données afin de constituer un modèle pour prédire l'ensemble des *tags*. Le *KNN* et la *Random Forest* offrent des sorties multiples et ne requièrent donc pas l'application de cette stratégie. Dans le cadre du *machine learning non supervisé*, nous avons généré ce qui s'apparente à une *clusterisation* des différents documents afin de déceler les termes qui différencient chaque document du *corpus*.

ÉVALUATION

Le modèle sélectionné devra présenter un compromis entre performance, rapidité d'exécution et un poids minime, étant donné que nous avons pour objectif de proposer un *endpoint* d'API disponible en ligne (le service *cloud* est gratuit et ses performances limitées).

Afin d'évaluer les différents modèles, nous avons calculé les métriques suivantes :

- *Jaccard* (*samples*) : le ratio entre l'intersection et l'union des prédictions, il permet d'estimer la similarité entre les valeurs réelles et les prédictions,
- *Recall* (*pondéré*) : le *recall* permet d'optimiser le ratio de *vrais positifs* du modèle, il s'agit du ratio des *vrais positifs* dans l'ensemble des individus positifs du modèle. Étant face à des classes déséquilibrées, nous utilisons le moyennage pondéré (*weighted*),
- *Targets_hit* : le ratio moyen de *tags cibles* prédits par le modèle (jusqu'à 5 *prédictions*),
- *Tags_hit* : le ratio moyen du nombre de tags *y_true* prédits correctement pour chaque *document* du *corpus* de test,
- *Fill* : le ratio entre le nombre de *tags filtrés*, et le nombre de prédictions fournies par le modèle,
- Temps de prédiction : le temps nécessaire pour réaliser une inférence moyenne de *X_test*,
- Taille du modèle en *mb* : la taille du modèle en mégabytes.

Lorsque la modélisation a rendu possible la génération des probabilités associées à chaque prédiction (utilisation de la méthode *predict_proba de Scikit-Learn*), nous avons filtré les probabilités prédites, afin de générer un total de *5 tags,* en abaissant le seuil par défaut de *0.5*. Dans le tableau récapitulatif des résultats, le suffixe *« _5 »* correspond à la caractéristique suivante : les prédictions sélectionnées sont les 5 qui présentent les probabilités les plus élevées,

En complément d'un tableau récapitulatif des performances, nous avons généré un tableau comparatif des résultats où figure les prédictions des différents modèles pour des *échantillons* sélectionnés aléatoirement. Ce tableau nous permet d'évaluer qualitativement les prédictions des différents algorithmes.

TABLEAU RECAPITULATIF DES PERFORMANCES

Algorirthme	Jaccard	Recall	Inference_ti	Model_size_	Targets_hit	Tags_hit	Fill
LDA					0.16499	0.139719	1.184823
LogR	0.251044	0.269541	0.000459	2	0.26994	0.227495	0.344097
SVM	0.36818	0.411815	0.000462	2	0.412766	0.347575	0.560452
SGD	0.379367	0.434134	0.000492	2	0.43493	0.366413	0.619432
KNN	0.147182	0.153386	0.002602	99	0.153567	0.129459	0.201919
KNN_5	0.325345	0.539083	0.002542	99	0.4723	0.3983	1.39035
Random_Forest	0.021931	0.021844	0.00116	5037	0.022144	0.018437	0.026023
Random_Forest_5	0.265594	0.438883	0.001293	5037	0.439479	0.370421	1.259759
Random_Forest_HE	0.23456	0.245607	0.000825	5085	0.246834	0.207295	0.309811
Random_Forest_HE_5	0.392175	0.58733	0.001139	5085	0.588477	0.495711	1.225852

Nous avons constaté que sans stratégie de selection des prédictions, par défaut, les algorithmes ne permettaient pas de générer un nombre de prédictions exploitable (Fill). De plus, nous avons relevé que la *réduction dimensionnelle* était affectait les performances de la *RandomForest,* nous avons donc *fit* un *RandomForest* « high-end » (suffixe HE) sur des données non réduites.

TABLEAU COMPARATIF DES RESULTATS

Tags_target	LDA	LogR	SVM	SGD
pointers,struct,variables,c	[vector, c++, inheritance, function, object]	(c++, c, pointers)	(c++, c, pointers, struct)	(c++, c, pointers, struct)
linq,concurrency,transactions,linq-to-sql,c#	[sql, linq, database, performance, null]	(c#, linq)	(c#, sql, linq, linq-to-sql)	(c#, sql, linq, linq-to-sql)
macos,xcode,debugging,c	[build, xcode, java, performance, configuration]	()	(xcode,)	(xcode,)
linux,qt,c++,windows	[time, qt, javascript, java, winforms]	(c++, qt)	(c++, qt)	(c++, qt)
linux,shell,macos,windows	[process, .net, time, difference, program]	(linux, shell)	(linux, shell)	(linux, unix, shell, process)
Tags_target	**KNN**	**KNN_5**	**Random_Forest**	**Random_Forest_5**
pointers,struct,variables,c	(c++, c, pointers)	(c#, c++, c, arrays, pointers, struct)	()	(c#, .net, c++, c, pointers)
linq,concurrency,transactions,linq-to-sql,c#	(c#,)	(c#, .net, sql, sql-server, linq-to-sql)	()	(c#, .net, sql, sql-server, linq, tsql)
macos,xcode,debugging,c	()	(c#, .net, c++, xcode, unix, command-line)	()	(c#, c++, iphone, c, objective-c)
linux,qt,c++,windows	(c++, qt)	(c++, windows, visual-studio, user-interface, qt)	()	(c#, java, .net, c++, iphone, c, objective-c)
linux,shell,macos,windows	(shell,)	(c++, linux, unix, shell, process)	()	(c#, .net, c++, c, shell)

MODELE SELECTIONNE

Nous avons selectionné l'algorithme du *K Nearest Neighbors* (*nombre de plus proches voisins : 25, distance :* euclidienne) avec la stratégie *_5* pour sélectionner les prédictions. Ce modèle présente une performance correcte, puisqu'il permet de prédire correctement près de la moitié des tags filtrés et près de 40% des tags non filtrés, tout en restant très compact en terme d'espace disque (moins de 100 *mb*). La *RandomForest* affiche les meilleures performances, avec 60% de tags filtrés prédits, mais la taille du modèle est relativement élevée qui pose des problèmes pour le déploiement de l'*API*.

K NEAREST NEIGHBORS

Décrit comme du *Lazy Learning*, puisqu'il utilise l'ensemble du *jeu de données* pour fournir de nouvelles prédictions, l'algorithme du *K Nearest Neighbors* catégorise les individus d'un jeu de données en se basant sur les similitudes des individus grâce à un calcul de distance.

Pour un ensemble de données *D*, une fonction de définition de distance (*euclidienne*, dans notre cas), un nombre entier *K* on prédit pour une observation *X* la variable de sortie *y* en calculant la distance entre *X* et l'ensemble des observations de *D*. On retient les *K* plus proches voisins, dans le cadre d'une classification, on retient le *mode* de *y*.

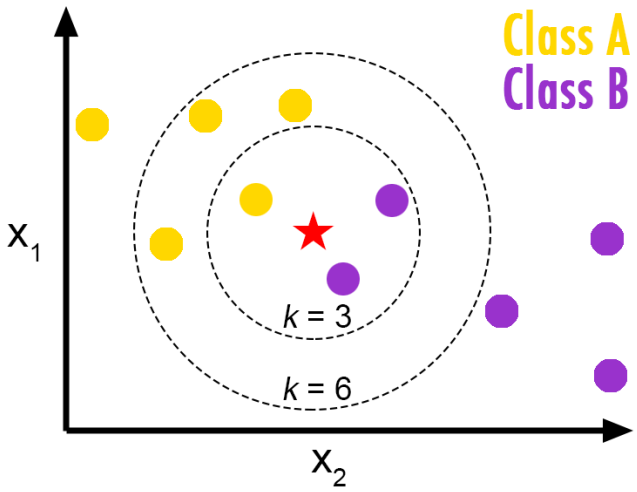


Schéma de la logique du KNN

En l'occurrence, nous utilisons la *distance euclidienne*, la racine carrée de la somme des carrées entre les coordonées (*x* et *y*) de deux points, calculée par la formule suivante :

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

CONCLUSION ET PISTES D'AMELIORATION

Ne disposant pas d'une puissance de calcul importante, nous avons dû limiter la dimensionnalité des données au maximum. Certains modèles, tel que la *RandomForest* ont requis des temps d'entrainement beaucoup plus élevé que les autres et la taille de fichier du modèle généré semble disproportionnée pour répondre à note problématique.

Afin d'obtenir et de déployer des modèles plus performants, la location de machines spécialisées (*Cloud Computing*) serait nécessaire. L'une des pistes pour répondre à cette problématique serait la mise en place d'un *neuronet,* beaucoup plus léger. On pourra aussi envisager le transfert learning grâce à des modèles tels que *Bert*. Bien sûr, de nombreuses améliorations peuvent être apportées à ce projet, l'extension du *jeu de données*, un traitement des données plus avancé (avec la constitution d'un *dictionnaire* assemblé à la main).