Catégorisation automatique des questions de StackOverflow

# StackOverflow

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

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

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 données

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. 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.

# Problématique

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 proposée

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 données

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.

## Lexique

* Document : un document est un individu du jeu de données : en l’occurrence, une question composée d’un titre et d’un corps.
* Corpus : le corpus est l’ensemble des documents contenus dans le jeu de données

## Description du jeu de données

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 devront être tokenizés.

# Prétraitement des données

L’objectif du prétraitement des données textuelles du corpus est d’extraire des données brutes les informations les plus essentielles de chaque document. 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 programmer - 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*).

Diagram

Description automatically generated

*Schéma du pipeline de prétraitement*

# Exploration des données

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

Text

Description automatically generated

Chart, bar chart

Description automatically generated

*Nuage de mots et fréquences des premiers termes de X*

A picture containing text

Description automatically generated

Chart, funnel chart

Description automatically generated

*Nuage de mots et fréquences des premiers termes de y*

# Traitement du jeu de données

Afin d’entrainer les modèles supervisés, et d’évaluer les prédictions des modèles supervisés, et non-supervisés, nous avons effectué des transformations supplémentaires sur le 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 détecter la présence des *tags* dans *y.*
* *Réduction de dimensionnalité*: la *vectorization de X* et la *binarization* ayant généré des matrices *numpy sparse*, nous appliquons des *TruncatedSVM* (l’équivalent d’un PCA) afin d’effectuer les réductions de dimensionnalité. Nous avons *fit* ces *transformers* en atteignant un pourcentage d’inertie (*explained variance ratio*) de 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.

A picture containing chart

Description automatically generated

*Éboulis des valeurs propres du TruncatedSVM appliqué sur y*

Diagram

Description automatically generated

*Schéma du pipeline de traitement*

Nous disposons d’une puissance de calcul limitée, pour cette raison nous choisissons 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*.

# Modélisation

Afin de générer des prédictions de suggestions de tags, dans le cadre du *machine learning supervisé*, la problématique se traduit par donc par de la *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 :

* SVM (*Support Vector Machine*) : les séparateurs à large marge optimisent la distance entre les individus d’un jeu de données afin de les classifier,
* SGD (*gradient stochastique*) : le *SGD* optimise la classification du *SVM* grâce à l’application de l’algorithme du *gradient stochastique*,
* KNN (*K Nearest Neighbors*) : le *KNN* rapproche les individus du jeu données ayant des caractéristiques similaires,
* Random Forest : la *Random Forest* crée un arbre décisionnel afin de catégoriser les individus,
* LDA (*Latent Dirichlet Allocation,* *machine learning non supervisé*) : attribue de manière probabiliste des thèmes 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 générerons 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* sera gratuit et ses performances limitées).

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

* Recall pondéré :
* Couverture (*Tags\_hit*) : le ratio moyen du nombre de tags *y\_true* prédits correctement pour chaque *document* du *corpus* de test,
* 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, les suffixes*« \_5 »* et *« \_select »* correspondent aux caractéristiques suivantes :

* *\_5* : les prédictions sélectionnées sont les 5 qui présentent les probabilités les plus élevées,
* *\_select* : les prédictions sélectionnées sont d’abord les termes présents dans le document (avec les probabilités les plus élevées), puis, si le nombre de prédictions sélectionné n’atteint pas 5, les prédictions sont complétées par celles qui présentent les probabilités les plus élevées.

Le traitement que nous avons réalisé demeurant relativement simple, et la dimensionnalité des données relativement élevée, les métriques calculées ne permettront pas d’établir entièrement la supériorité d’un modèle par rapport à un autre. Pour cette raison, en addition 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 permettra d’évaluer qualitativement les prédictions des différents algorithmes.

## Tableau récapitulatif des performances

Table

Description automatically generated

## Tableau comparatif des résultats

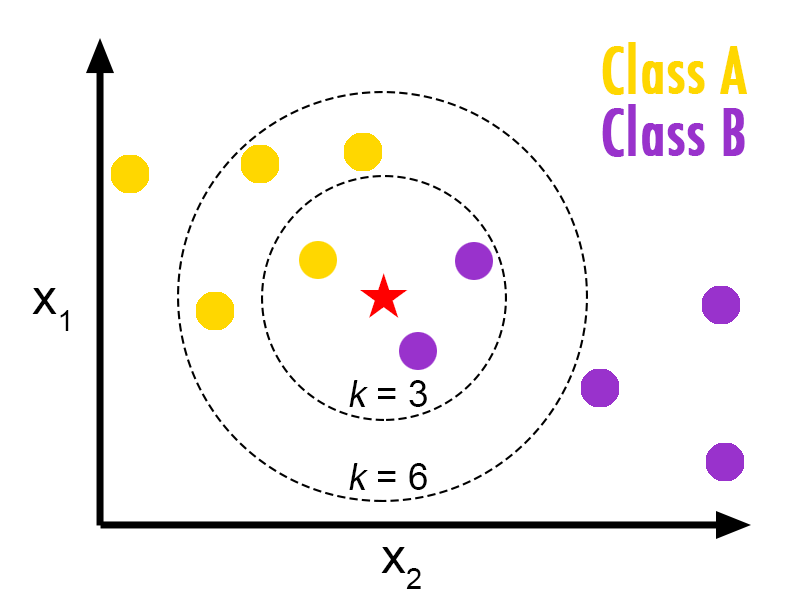
Table

Description automatically generated

## Modèle selectionné

Nous avons selectionné l’algorithme du *K Nearest Neighbors* avec la stratégie *\_select* pour sélectionner les prédictions. Ce modèle présente une performance correcte, puisqu’il permet de prédire correctement plus de la moitié des tags filtrés et plus de 40% des tags non filtrés (*y*), tout en restant très compact en terme d’espace disque (moins de 100 *mb*).

# K Nearest Neighbors



*Schéma de la logique du KNN, source :* *https://medium.com/machine-learning-researcher/k-nearest-neighbors-in-machine-learning-e794014abd2a*

Décrit comme du *Lazy Learning*, puisqu’il utilise l’ensemble du *jeu de données* pour fournir de nouvelles prédictions,

# Conclusion et pistes d’amélioration

Ne disposant pas d’une puissance de calcul élevée, nous avons dû limiter la dimensionalité 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 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, que nous ne l’avons pas explorée.

(<https://data.stackexchange.com/stackoverflow/query/new>)