



## **Projet 5 : Catégoriser automatiquement des questions**



## Plan du projet :

Projet 5 : Catégoriser automatiquement des questions.....	1
Plan du projet : .....	2
I.    Le contexte du projet .....	3
II.   Les données du projet .....	3
III.  Les préparations et explorations des données .....	4
1.  Récupération des données : .....	4
2.  Le traitement des données : .....	4
3.  Exploration des données : .....	5
4.  Vectorisation des données : .....	6
IV.   L'approche non-supervisée .....	7
1.  Le modèle LDA (Latent Dirichlet Analysis) : .....	7
2.  Le modèle NMF (Non-negative Matrix factorisation) : .....	8
3.  Les performances de LDA et NMF : .....	8
V.    L'approche supervisée.....	9
1.  Les modèles : .....	9
2.  Les mesures de performance : .....	9
VI.   Le Web-API de test : .....	11
VII.  Conclusion : .....	12

## I. Le contexte du projet

Dans le cadre de ma formation en « Ingénieur Machine Learning », je dois travailler sur le projet d'un système de suggestion de tag pour Stackoverflow.

StackOverflow est un site de questions-réponses liées au développement informatique. Les tags nous permettent de retrouver facilement la réponse à une question posée, pourtant ce n'est pas évident de trouver des tags associés à notre question.

Donc il était intéressant de développer une application qui suggérerait automatiquement plusieurs tags pertinents à partir d'une question posée.

## II. Les données du projet

Les données à exploiter sont des données réelles qui proviennent d'une interface d'exportation mise en place par StackOverflow : « [stackexchangeexplorer](#) » .

La liste des tables disponibles :

- Posts	- PostsWithDeleted
- Users	- PostTags
- Comments	- PostTypes
- Badges	- ReviewRejectionReasons
- CloseAsOffTopicReasonTypes	- ReviewTaskResults
- CloseReasonTypes	- ReviewTaskResultTypes
- FlagTypes	- ReviewTasks
- PendingFlags	- ReviewTaskStates
- PostFeedback	- ReviewTaskTypes
- PostHistory	- SuggestedEdits
- PostHistoryTypes	- SuggestedEditVotes
- PostLinks	- Tags
- PostNotices	- TagSynonyms
- PostNoticeTypes	- Votes
	- VoteTypes

Il y a 29 tables dans lesquelles se trouvent les informations suivantes :

- Les données des utilisateurs,
- Les sujets évoqués
- Les notations des questions posées et leurs réponses
- Les notations des utilisateurs selon la pertinence de leurs réponses
- Les historiques de chaque échange
- ...

Mais ce qui nous intéresse pour ce projet sera la table « Posts ».

Dans « Posts », se trouve le titre, l'objet, le score de pertinence pour la question et le nombre de réponse à la question.

### III. Les préparations et explorations des données

#### 1. Récupération des données :

Pour récupérer les données, voici la requête utilisée :

```
1 SELECT Id, Body, Title, Tags, Score, FavoriteCount, AnswerCount
2
3 FROM posts pst
4
5 WHERE pst.FavoriteCount > 50
6 AND pst.Score > 100
7 AND pst.Body is not null
8 AND pst.Tags is not null
9 AND pst.Title is not null
10 AND pst.AnswerCount > 0
11
12 ORDER BY Id ASC
```

Avec un résultat : 19 759 lignes sur 4 colonnes

#### 2. Le traitement des données :

- Le dictionnaire des mots :

Le traitement des données consiste à :

- Fusionner les données de « Body » et « Title » dans une table « BodyTitle »
- Appliquer « BeautifulSoup » pour extraire les informations dans les codes html
- Appliquer des nettoyages pour retirer les chiffres, la ponctuation, les caractères spéciaux et les « stopwords »
- Appliquer la « tokenization » et la « lemmatisation » pour ne garder que le sens des mots utilisés pour éviter les redondances.

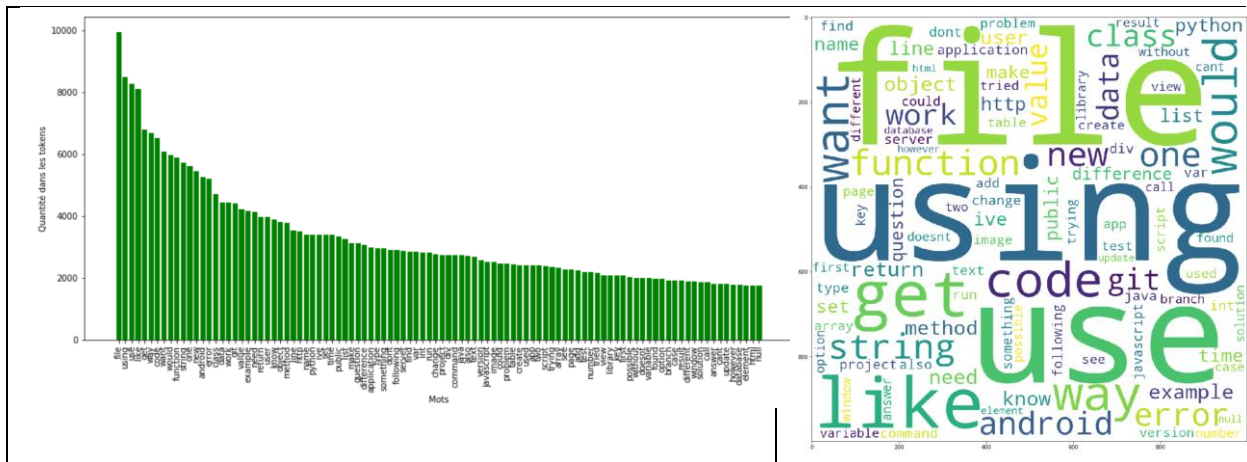
Les mots issus de ce traitement serviront de dictionnaire à notre modèle par la suite.

Le résultat de ce traitement nous donnera :

- 70 204 mots sur les 19 757 questions traitées

### 3. Exploration des données :

La représentation des 100 mots les plus utilisés :

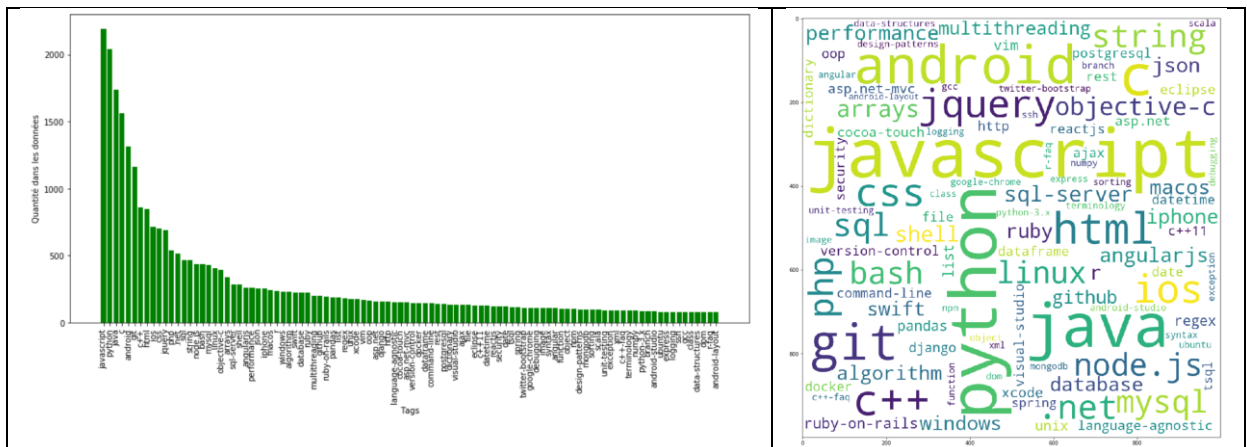


- Les tags :

Les tags sont sous forme de liste, pas besoin de faire des prétraitements pour les récupérer. Voici les résultats :

- 6 795 tags sur les 19 759 questions

La représentation des 100 tags les plus utilisés :



Etant donné que les 100 tags les plus utilisés couvrent plus de 98 % des questions, nous allons faire notre modélisation sur la base des top100.

Donc nous allons garder que les lignes qui contiennent au moins un des tags dans top 100 :

Cette suppression faite, nous allons passer de (19759, 8) à (19748, 8)

#### 4. Vectorisation des données :

Une fois les données (corpus et tags) bien nettoyées, il faudra les transformer en vecteurs, c'est l'étape de vectorisation, la vectorisation est nécessaire pour pouvoir faire des calculs et donc la modélisation.

Le corpus représente les données en entrée de notre modèle (=X),

On appliquera la vectorisation des « Bag Of Words » sur les top1000 des mots pour avoir les résultats suivants :

Out[28]:

	file	using	use	like	get	way	code	want	would	function	string	one	new	android
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	1	0	0	0
3	0	1	0	0	0	1	1	1	0	0	0	1	1	0
4	0	0	0	0	1	0	0	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2099	1	1	0	1	0	1	1	1	0	0	0	0	0	0
2100	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2101	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2102	1	1	0	1	0	0	0	0	0	0	0	0	0	0
2103	0	0	0	0	0	0	1	0	0	1	1	0	1	1

Le tag sera la valeur à prédire (=y).

on appliquera le «sklearn.preprocessing.MultiLabelBinarizer » et ne garder que les top 100 des tags pour avoir le résultat suivant :

Out[29]:

	javascript	python	java	c	android	git	c++	html	ios	css	jquery	php	.net	sql	stri
0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2099	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2100	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2101	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Puis nous allons séparer les données en 80% d'entraînement et 20 % de test :

```
# Split train and test :
X_train, X_test,
y_train, y_test = train_test_split(X_data,
                                    y_data,
                                    test_size=0.2,
                                    random_state=1)
```

#### IV. L'approche non-supervisée

Dans cette partie, nous allons faire de modélisation de « sujets(Topics) », il s'agit de créer des clusters des mots en fonctions de leur contenu et leur ressemblance.

Nous verrons deux approches, la **LDA** (Latent Dirichlet Analysis) et le **NMF**(Non-negative Matrix factorisation) .

Et pour mesurer la similarité entre la prédiction des modèles et les vrais tags, nous allons utiliser la « cosine\_similarity », elle permet de quantifier la ressemblance entre deux vecteurs (vecteurs des tags de prédictions et vecteurs des vrais tags)

##### 1. Le modèle LDA (Latent Dirichlet Analysis) :

C'est un modèle probabiliste génératif avec les hypothèses suivantes :

- Chaque question du corpus est un ensemble de mots sans ordre (*bag-of-words*) ;
- Chaque question m aborde un certain nombre de topic dans différentes proportions qui lui sont propres  $p(\theta_m)$  ;
- Chaque mot possède une distribution associée à chaque topic  $p(\phi_k)$  . On peut ainsi représenter chaque topic par une probabilité sur chaque mot.
- $z_n$  représente le topic du mot  $w_n$

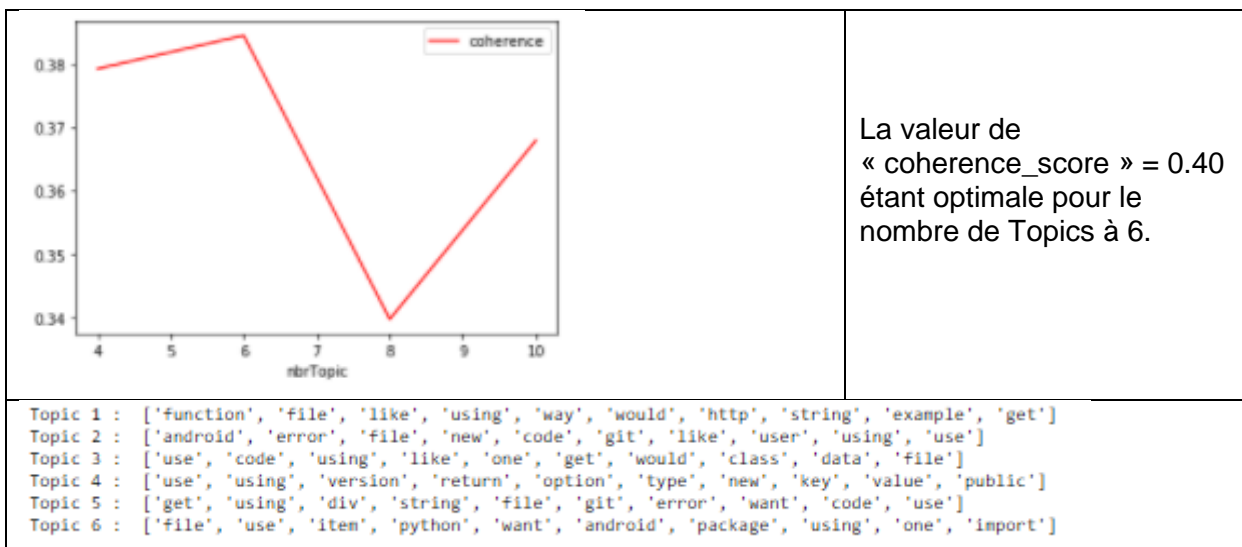
Les variables (X, Y) sont déjà définies précédemment, donc la librairie

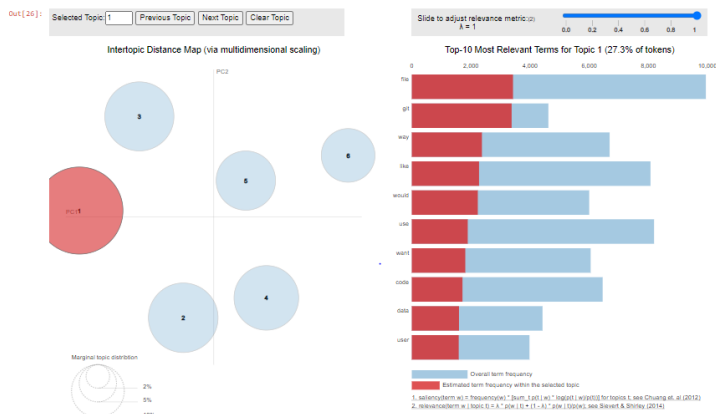
« sklearn.decomposition.LatentDirichletAllocation » ou

«gensim.models.Ldamulticore.LdaMulticore» permettra d'extraire les informations sur les topics.

##### Les topics :

Pour déterminer le nombre de topics optimal, j'ai utilisé le calcul de « coherence\_score » en fonction de nombre de topics, dont voici le résultat :





Sur le côté gauche, la zone de chaque cercle représente l'importance du topic par rapport au corpus.

Comme il y a 6 topics, nous avons donc 6 cercles.

La distance entre le centre des cercles indique la similitude entre les sujets.

Sur le côté droit, l'histogramme de chaque topic montre les 10 mots les plus pertinents. Par exemple, dans le topic 1, les mots les plus pertinents sont file, git, like, ...

## 2. Le modèle NMF (Non-negative Matrix factorisation) :

En alternative à LDA, le modèle NMF permet aussi de faire une modélisation de sujet automatique non supervisée. J'ai utilisé la valeur optimale de topics vu sur LDA, c'est-à-dire 6.

### Les topics :

```
Les Topics dans NMF model (Frobenius norm):
Topic #0: function use code like class using would way object get
Topic #1: file directory line command folder python path script using open
Topic #2: git branch commit repository master remote change push commits merge
Topic #3: string convert array character value list int java public return
Topic #4: difference whats use one java two vs. explain used please
Topic #5: table database sql column mysql row select query server key
```

Ici , on peut facilement déduire visuellement par exemple que :

- le topic 2 parle de gestion de versionning de projet
- le topic 5 parle de base des données

## 3. Les performances de LDA et NMF :

La cosine\_similarity : c'est une valeur comprise dans l'intervalle [-1,1],

-1 indiquera des vecteurs opposés,

0 des vecteurs indépendants (orthogonaux)

1 des vecteurs similaires

```
DummyClassifier score : 0.004810126582278481
Cosine_similarity score LDA : 0.016174398111016484
Cosine_similarity score NMF : 0.01630386829837769
```

Nous constatons que les scores des cosine\_similarity sont supérieurs à celui de DummyClassifier mais ils sont très proche de 0  
Donc les vecteurs sont indépendants, il faudra tester d'autres approche de prédiction



## V. L'approche supervisée

A la différence de l'approche non-supervisée qui crée des clusters des mots, l'objectif ici est de déterminer la liste des tags qui correspond à une phrase en entrée dans l'algorithme.

### 1. Les modèles :

Plusieurs modèles ont été testé pour les comparer et choisir le plus performant :

LogisticRegression : C'est un modèle de régression binomiale. Elle constitue un cas particulier de modèle linéaire généralisé.

SGDClassifier : C'est un modèle de classification de descente de gradient(itérative)

MultinomialNB : C'est un modèle de classification de Naïve Bayes

LinearSVC : le modèle « Linear Support Vector Classifier » applique une fonction de noyau linéaire pour effectuer la classification et fonctionne bien avec un grand nombre d'échantillons. Si nous le comparons avec le modèle SVC, le SVC linéaire a des paramètres supplémentaires tels que la normalisation de pénalité qui applique «L1» ou «L2» et la fonction de perte. Noyau rbf(par défaut)

Perceptron : C'est un modèle de classifieur binaire (c'est-à-dire séparant deux classes). Il a été inventé en [1957](#) par [Frank Rosenblatt](#)

PassiveAggressiveClassifier : C'est un modèle d'algorithme d'apprentissage incrémental, Le concept de base est que le classificateur ajuste son vecteur de poids pour chaque échantillon d'apprentissage mal classé qu'il reçoit, en essayant de le corriger.

RandomForest : C'est un modèle qui repose sur l'apprentissage par arbre de décision

KNN : C'est la méthode de classification des  $k$  plus proches voisins

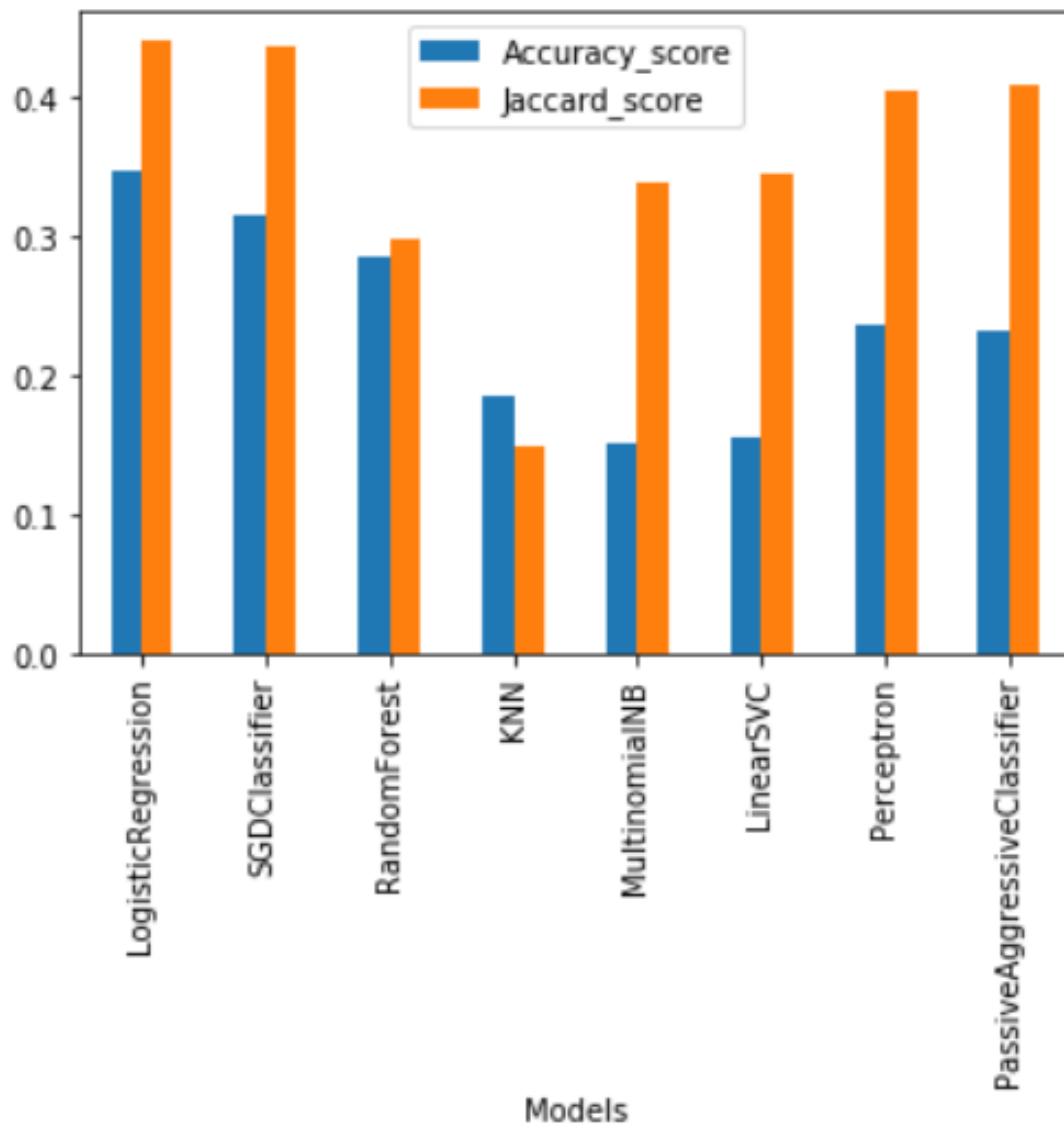
### 2. Les mesures de performance :

Pour comparer les résultats, deux metrics « Jaccard\_Score » et « accuracy\_score » ont été utilisé .

« Jaccard\_Score » : sert à comparer la similarité et la diversité entre les échantillons. On l'appelle ainsi d'après le botaniste suisse Paul Jaccard qui l'a mis en place.

« Accuracy\_score » : sert à calculer la précision d'une prédiction, c'est le rapport entre les observations correctement prédites et les observations totales

Voici les résultats des scores sur les différents modèles :



Donc on peut dire que le modèle de « LogisticRegression » est le plus performant parmi les 8 testés.

## VI. Le Web-API de test :

Pour tester la modélisation, une interface graphique est mise en place grâce à « Streamlit ».

« Streamlit » est un Framework open-source Python qui permet de faciliter l'interaction des utilisateurs avec les modèles et les données dans Python.

Le formulaire est accessible sur le lien suivant : <http://localhost:8501> ou <http://192.168.1.86:8501>

Mais avant, il faut :

- récupérer les projet sur git (<https://github.com/leandre12/suggestion-tag/>)
- démarrer le serveur en suivant la procédure décrite dans le document « **LancementFormulaire.pdf** » qui se trouve dans le même répertoire que celui-ci.

Voici la page de test de notre formulaire :

The screenshot shows a web browser window with the address bar set to 'localhost:8501'. The page title is 'formAPI - Streamlit'. On the left, there is a sidebar with the text 'Léandre ANDRIANIINA : Projet de suggestion de tag pour stackoverflow'. The main area contains a form with the label 'Saisissez votre question :', an empty text input field, and a button labeled 'Recherche de tags'.

En saisissant le texte suivant : « Mes langages préférés sont : javascript, python , C#, PHP », on obtiendra le résultat :

This screenshot shows the same web interface after a query has been submitted. The text input field now contains 'Mes langages préférés sont : javascript, python , C#, PHP'. Below the input field, the text 'voici la liste des tags : ['python', 'javascript']' is displayed. The 'Recherche de tags' button remains visible.

Donc l'algorithme propose deux tags : Python et javascript pour cette phrase.

## VII. Conclusion :

Pour conclure l'étude sur ce projet, la base des données à disposition était très riches et complètes mais par manque des temps pour approfondir le sujet, on doit se limiter à n'utiliser que la table « Posts ».

Le modèle de « RegressionLogistics » est le plus performant parmi tous les modèles testés.

Une interface graphique a été mise en place pour faciliter le test de la modélisation.

Cependant il y a des axes d'améliorations qui semblent importantes, comme l'importance de prendre en compte les dates d'enregistrements des « Posts » (questions) parce que si une question était très bien notée mais date de plus de 10 ans, elle ne doit pas avoir la même importance qu'une question récente, et les réponses technique peut être obsolète dans le temps.

Il est aussi possible d'améliorer le temps de réactivité du formulaire en chargeant toutes les données nécessaires à l'ouverture de la page.