Projet 5 : Catégorisation automatiquement des questions

Formation IML 2020

Plan de présentation :

i. Le contexte du projet
ii. Les données du projet
iii. Les préparations et explorations des données
iv. L'approche non-supervisée
v. L'approche supervisée
vi. Le Web API de test

Le contexte du projet :

- StackOverflow:
  - Créé en 2008, plus de 50 M de visiteurs uniques par mois
  - Questions-réponses sur des programmations informatiques
  - Votes des utilisateurs définissent la qualité et pertinence
  - Recherche des sujets par tags

- Objectifs :
  - Aider les nouveaux utilisateurs
  - suggestion des tags associés à une question
  - Créer une interface graphique pour tester

Les données :

- Les données authentiques
- 29 tables qui contiennent :

➢ Les informations utilisateurs (nom, adresse, score…)

➢ Les informations questions ou Posts(titre, contenu, score, nb réponses, tags, dates, …)

➢ Les historiques

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

Préparations des données :

Récupération des données

- Utiliser la table « Posts »
- Score supérieur à 100
- FavoriteCount supérieur à 50
- Au moins une réponse

```
SELECT Id, Body,Title,Tags,Score, FavoriteCount,AnswerCount

FROM posts pst

WHERE pst.FavoriteCount > 50
AND pst.Score > 100
AND pst.Body is not null
AND pst.Tags is not null
AND pst.Title is not null
AND pst.AnswerCount > 0
```

Résultat : (19 759, 4)

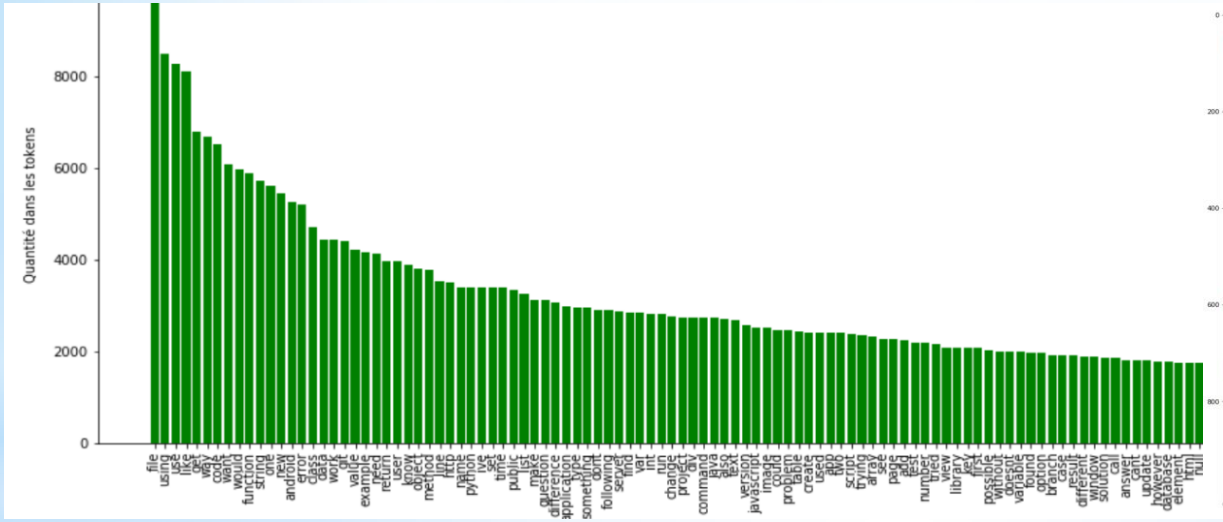
Traitement des données

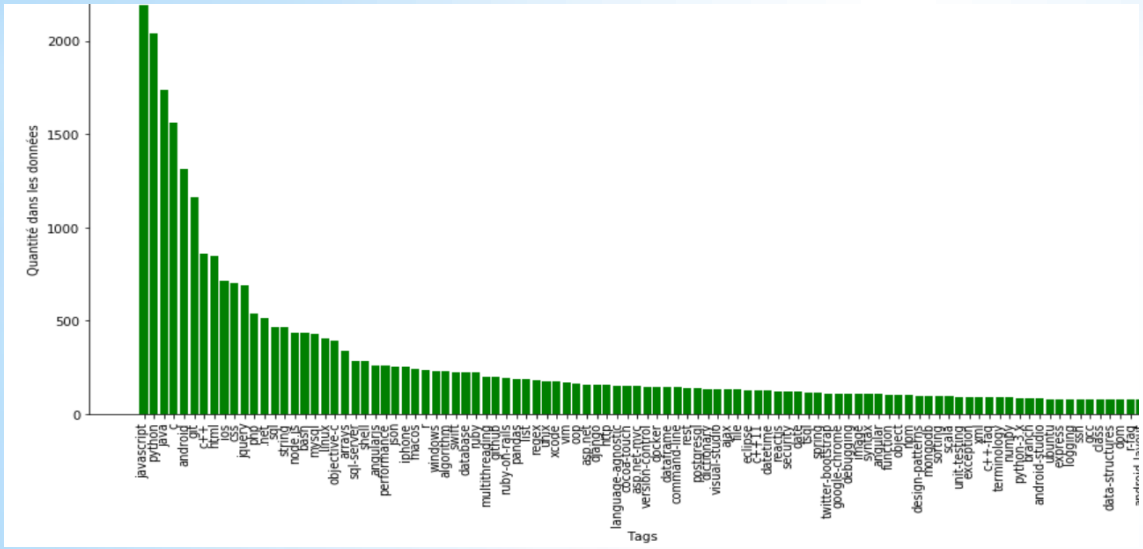
- Fusionner « Body » et « Title »
- BeautifulSoup pour extraire des données dans html
- Nettoyages : chiffres, ponctuations, car speciaux, stopwords
- Tokenisation et lemmatisation

- 70 204 mots
- 6 791 tags

Explorations des données :

- Les 100 mots et tags les plus utilisés : (histogramme, wordcloud)





Explorations des données :

Nettoyage :

- Ne garder que les top1000 des mots
- Ne garder que les questions contenant au mois un tag dans top100 : (19 759, 8) → (19 748, 8)

Vectorisation des données pour modélisations :

- En entrée (=X)de notre modèle, on a les mots (corpus) vectorisé en « Bag Of Words »

	file	using	use	like	get	way	code	want	would	function	string
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	1
3	0	1	0	0	0	1	1	1	0	0	0
4	0	0	0	0	1	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
2099	1	1	0	1	0	1	1	1	0	0	0

- En sortie (=y), ou valeur prédite, on a les top 100 des tags :

	javascript	python	java	c	android	git	c++	html	ios	css	jquery	php
0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1

- Split en 80% d'entrainements et 20% de tests
- Cas d'une classification multi-étiquette ou multi-label

Modélisation : Approche non-supervisée

Créer automatiquement des thèmes(topics) en fonctions de contenu et ressemblance des mots(Trending topics)

Le modèle LDA (Latent Derilicht Analysis) :

C'est un modèle probabiliste avec hypothèses :

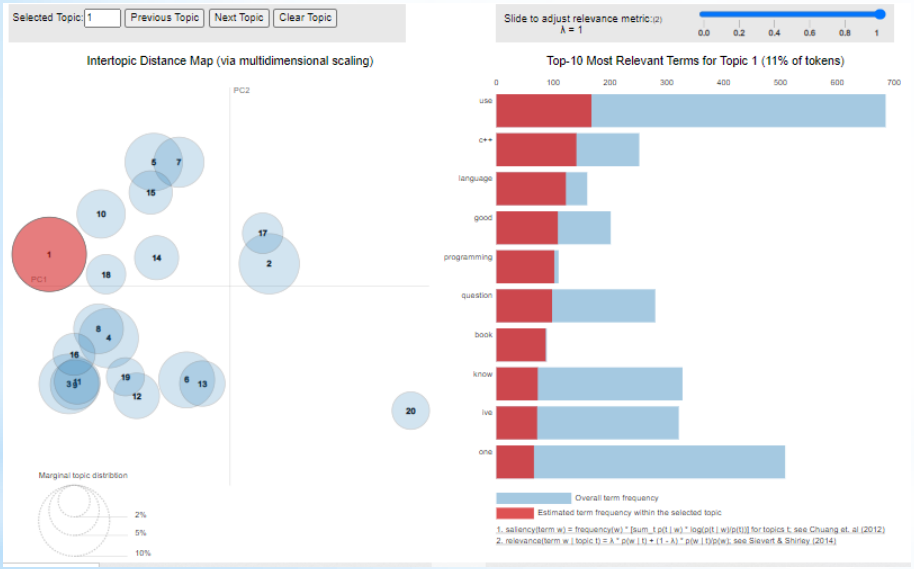
- Un document est formé d'ensemble de mots(BOW)
- Chaque document aborde un certain nbr de topic dans différentes proportions
- Chaque mot a sa proportion de probabilité d'être liée à un topic
- Donc on peut représenter chaque topic par une probabilité sur chaque mot

LDA nous permet d'extraire ces probabilités entre topics et mots

Topics optimal avec le modèle HDP: 20

```
Les topics du modele LDA :
Topic #0: number service window range .net python find output print way
Topic #1: int import return eclipse like node field form set class
Topic #2: code best statement mysql whats sql like result practice dont
Topic #3: use c++ language good programming question book know ive one
Topic #4: feature file hidden view copy internet explorer set use one
Topic #5: class method java static use object one interface function using
Topic #6: file database one utf using used memory content data output
Topic #7: http public variable foo type void method static example class
Topic #8: git branch merge commit repository database change svn file container
Topic #9: python list file using module way want like function add
Topic #10: test unit file class abstract testing directory constructor make using
Topic #11: sql query table procedure join date index stored currency server
Topic #12: visual studio expression project regular ball asp.net system solution control
Topic #13: file command script line way window using like bash get
Topic #14: div jquery element javascript html using page like script form
Topic #15: string function object way table javascript value return column get
Topic #16: error image event text way get add remove input function
Topic #17: value exception type string java one public example code dont
Topic #18: difference use array one would memory data whats process pattern
Topic #19: application web way user app would using need iphone thread
```

Représentation graphique avec PyLDAVis :



La taille du cercle -> l'importance du topic dans le document
Distance entre les cercles -> similitude
Plot en rouge -> proportion des mots dans le topic

Modélisation : Approche non-supervisée

Le modèle NMF (Non-Negative Matrix Factorization) :

```
Les Topics dans NMF model (Frobenius norm):
Topic #0: code use would like way using one ive know application
Topic #1: file directory line open path header batch xml using filename
Topic #2: string convert like way comparison int char character str remove
Topic #3: table database sql mysql column server row name select query
Topic #4: git branch commit repository merge svn change remote add conflict
Topic #5: function return var call jquery variable something parameter way int
Topic #6: difference whats one i++ use vs. main performance loop explain
Topic #7: python module print directory source using window platform output use
Topic #8: class method static public private interface abstract void use object
Topic #9: list item generic use order range name person search linked
Topic #10: div image jquery element img center id= html height width
Topic #11: value key dictionary column int convert option variable integer sort
Topic #12: script bash shell command directory exit tag path line unix
Topic #13: java getting object implementation interface library using equivalent write different
Topic #14: javascript object property var way check json key obj oriented
Topic #15: array element byte php way linked delete int new find
Topic #16: date datetime time current day get return format month range
Topic #17: test unit testing code tool coverage framework objective app print
Topic #18: feature hidden trick operator tip ruby know known question sure
Topic #19: c++ int pointer std book variable static template struct unsigned
```

Mesures de performance :

« **Jacard_smilarity** », permet de quantifier la ressemblance entre deux ensembles (tags de prédictions et tags des vrais valeurs)

valeur comprise entre [0,1]

```
Jacard similarity LDA :  0.1407035175879397
Jacard similarity NMF :  0.1619718309859155
```

Modélisation : Approche supervisée

Déterminer la liste des tags qui correspond à une phrase en entrée

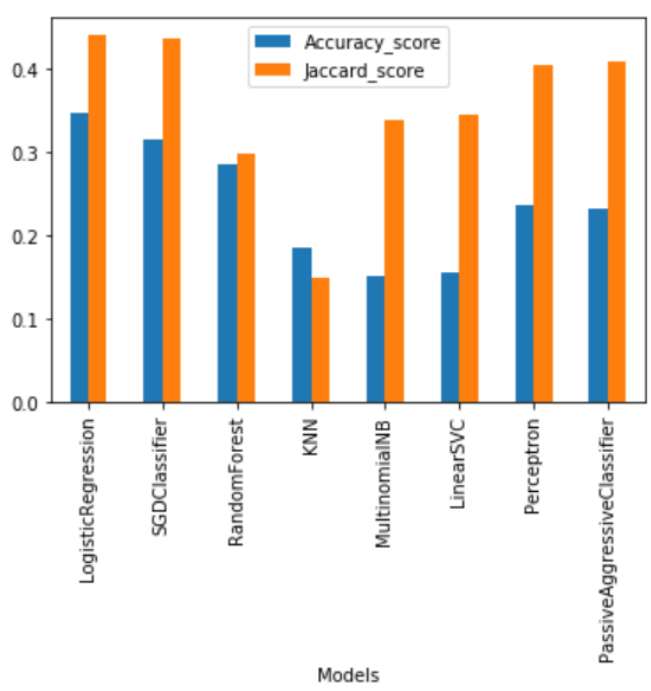
Plusieurs modèles testés

Metrics :

Jaccard_score : mesure de similarité et diversité entre les échantillons

Accuracy_score : mesure de précision de prédiction

Résultats :



Temps d'entrainement (s) :

LogisticsRegression :	2,25 mn
SGDBClassifier :	0,28 mn
RandomForest :	1,60 mn
KNN :	13,4 mn
MultinomialNB :	0,10 mn
LinearSVC :	9,08 mn
Perceptron:	1,08 mn
PassiveAgressiveClassifier :	1,40mn

LogisticRegression a un meilleur score(0,45)

C'est un modèle de régression binomiale, c'est un cas particulier de modèle linéaire généralisé

Modèle à utiliser dans le WebAPI

Test Web API

API developpé avec Flask utilisant le modèle « LogisticRegression »

Intérface en HTML

hébergement sur PythonAnywhere(http://leandre12.pythonanywhere.com/)

Phrase de test : « Mes langages préférés sont : javascript, python, c#, PHP»

Résultat : ['javascript', 'python']

Conclusion :

- BDD complète avec beaucoup de possibilités d'analyse
- 3 données essentielles dans Posts : Title, Body, Tags
- Modèle de LogisticRegression le plus performent
- Axes d'améliorations possibles :
    - Pris en compte date d'enregistrement
    - Amélioration du temps de réactivité du formulaire

Merci pour votre attention