Catégorisez automatiquement des questions

Synthèse
Traitements et Modélisation

Table of Contents

Catherine LE
20211012

1 - Traitements

1.1 – Récupération de la donnée

Avec les requêtes SQL sur https://data.stackexchange.com/stackoverflow/query/new, il est possible de récupérer en format .csv les 5 dataframes contenant les noms de colonnes souhaitées tels que `Title` et `Tags`.

Les tables des fichiers `Posts.csv, PostsWithDeleted.csv`, `SuggestedEdits.csv` permettent de récupérer ces informations. La table du fichier `TagSynonyms.csv` contient une colonne `TargetTagName` correspondant aux tags approuvés pas les utilisateurs. La table du fichier `Tags.csv` contient des tags n'apparaissant qu'une seule fois.

Les données sont filtrées avec :

- `AnswerCount` > 10
- `FavoriteCount` > 200
- `ViewCount` > 5000 vues
- `Score` > 15

Ces seuils sont choisis en fonction de la distribution de ces colonnes.

La juxtaposition des tables des cvs `Posts.csv, PostsWithDeleted.csv, SuggestedEdits.csv` donne un dataframe de dimension `(2015, 2)`.

Dataframe des données récupérées

	Title	Tags
3	How do I calculate someone's age based on a Da...	<c#><.net><datetime>
4	Calculate relative time in C#	<c#><datetime><time><datediff><relative-time-s...
108	Versioning SQL Server database	<sql-server><database><svn><version-control>
175	How do you make sure email you send programmat...	<email><email-spam>
222	The definitive guide to form-based website aut...	<security><http><authentication><language-agno...

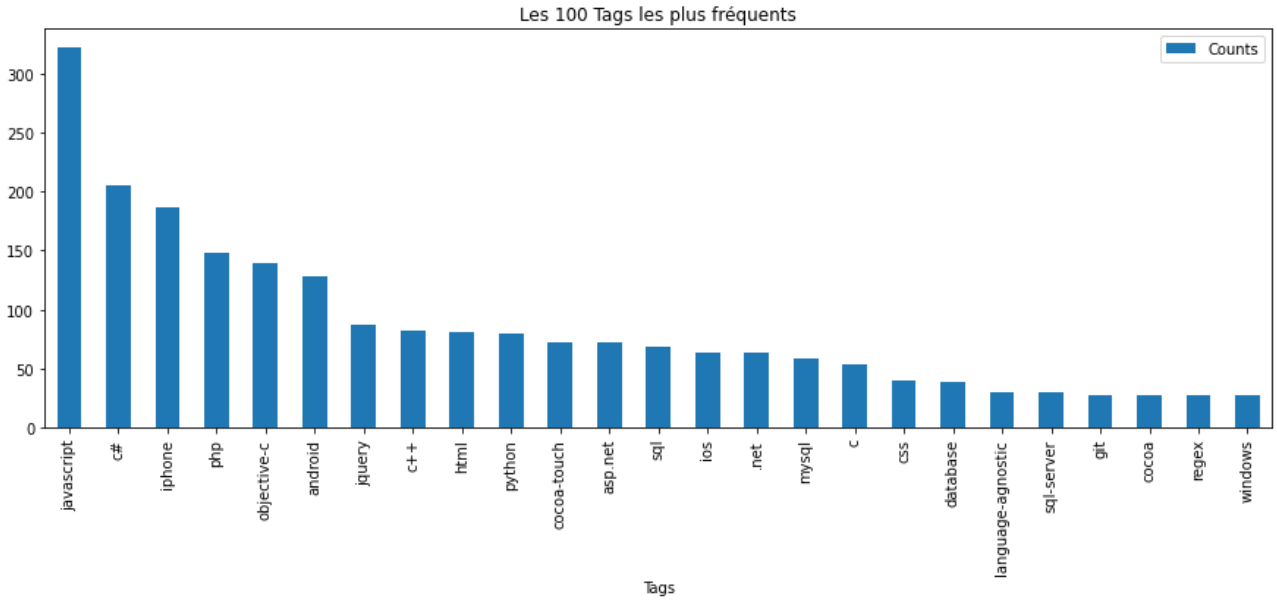
1.2 – Nettoyage des Tags

Suite à cette récupération les balises ont perdu leurs lettres en minuscule permettant de les identifier. Il est donc impossible d'utiliser BeautifulSoup.

Pour nettoyer la colonne Tags, la fonction `liste_tags` enlève les balises avec une regex, splite la chaîne de caractères en listes de mots. Sur cette liste, on récupère les 100 tags les plus fréquents afin de les utiliser comme variables *y* binaires dans les modèles.

Les tags "java" sont remplacés pour ne donner qu'un seul tag "javascript".

Les 25 Tags les plus fréquents



Tags nettoyés

	Title	Tags_List
0	How do I calculate someone's age based on a Da...	[c#, .net, datetime]
1	Calculate relative time in C#	[c#, datetime]
2	Versioning SQL Server database	[sql-server, database, svn, version-control]
3	How do you make sure email you send programmat...	[email]
4	The definitive guide to form-based website aut...	[http, language-agnostic]

MultiLabelBinarizer de sklearn transforme les tags de la colonne Tags_List en dummies. On obtient alors un dataframe de dimension `(2015,101)`.

1.3 – Nettoyage des Titles

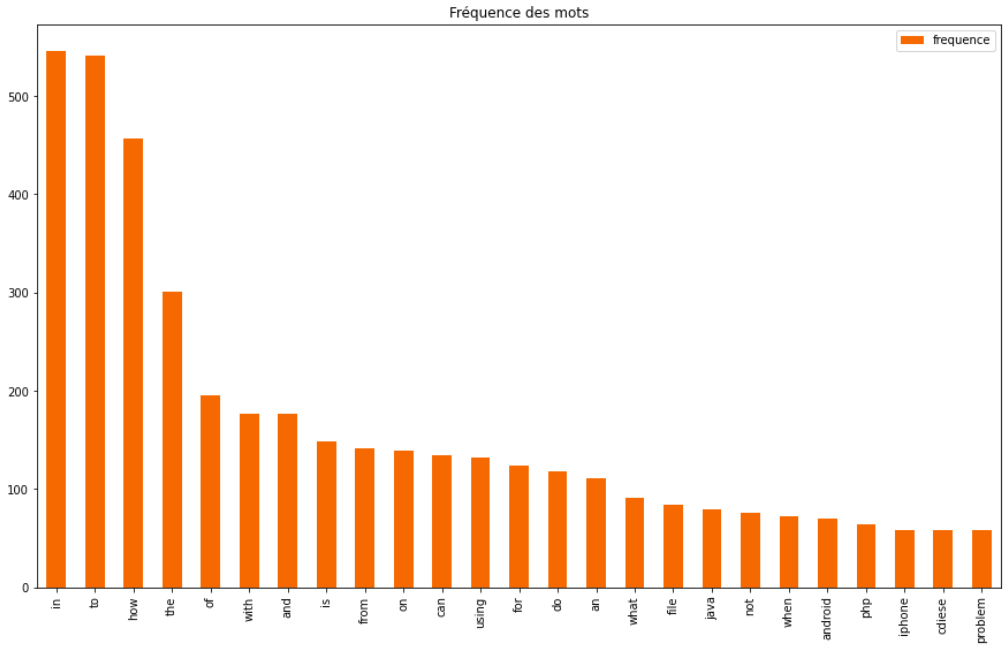
- **Tokenization** : la tokénisation transforme une phrase en une liste de mots. Il est possible de supprimer à cette étape les ponctuations, et les caractères spéciaux, de transformer les lettres majuscules en minuscules.

Phrases tokénisées

```
In [45]: #data tokenized = converts to list of words
         data_words
```

```
Out[45]: [['how',
           'do',
           'calculate',
           'someone',
           'age',
           'based',
           'on',
           'datetime',
           'type',
           'birthday'],
          ['calculate', 'relative', 'time', 'in', 'cdiese'],
          ['what', 'are', 'some', 'good', 'net', 'profilers'],
          ['what',
           'is',
           'the',
           'difference',
           'between',
           'string',
           'and',
```

-



Stopwords: ce sont les mots qui n'ont pas de signification particulière tels que les mots de liaison, les pronoms, les déterminants ect. Pour l'anglais, la librairie stopwords permet de les recenser.

Grâce à cette librairie, nous retenons que les mots non présents de cette bibliothèque.

```
In [59]: from nltk.corpus import stopwords
         nltk.corpus.stopwords.words('english')

Out[59]: ['i',
          'me',
          'my',
          'myself',
          'we',
          'our',
          'ours',
          'ourselves',
          'you',
          "you're",
          "you've",
          "you'll",
          "you'd",
          'your',
          'yours',
          'yourself',
          'yourselves',
          'he',
          'him',
```

- **Lemmatization** : cette étape permet de rendre le mot à sa forme la plus courte tout en gardant son sens. Cela a l'avantage de supprimer les mots qui se répètent rendant la matrice de vectorisation plus dense avec moins de colonnes pour ainsi générer de meilleurs topics. Il existe plusieurs librairies permettant de le faire: **PorterStemmer et SnowballStemmer**.

PorteStemmer tronque beaucoup trop les mots, cela peut faire perdre le sens du mot. **SnowballStemmer** donne de bons résultats mais il ne permet pas de supprimer les adverbes et les pronoms. Le package de Spacy avec sa fonction **lemmatization** permet de filtrer les mots selon la nature des mots.

Titles lemmatisés

```
In [70]: # Select ['ADJ'] to have some words such as cdiese et cplusplus
         data_lemmatized = lemmatization(data_words, allowed_postags=['NOUN', 'VERB','ADJ']) #OK
         data_lemmatized[11]

Out[70]: 'difference generic cdiese java template cplusplus'
```

Une données sont prêtes à l'emploi. Les prochaines étapes sont:

1- GridSearchCV sur les paramètres des vectorizers

2- Calcul des performances des modèles selon leurs types de vectorizers

puis sélection du meilleur modèle.

3- GridSearchCV du modèle et son vectorizer

4- Création ou prédiction des Tags et calcul des scores

2 – Modélisation

2.1 - GridSearchCV sur les paramètres des vectorizers

Définitions

- **CountVectorizer**: le nombre de colonnes est égal à la taille du vocabulaire trouvé.

Exemple:

```
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.get_feature_names())

['and', 'document', 'first', 'is', 'one', 'second', 'the',
'third', 'this']

>>> print(X.toarray())

                          [[0 1 1 1 0 0 1 0 1]
                           [0 2 0 1 0 1 1 0 1]
                           [1 0 0 1 1 0 1 1 1]
                           [0 1 1 1 0 0 1 0 1]]
```

- **TfidfVectorizer**: équivalent à CountVectorizer suivit par TfidfTransformer.

- **TfidfTransformer** :

Tf = fréquence des termes
tf-idf = fréquence des termes x fréquence inverse des documents

```
                          Term Frequency
 = (Nbre d'apparition du terme dans le doc) / (Total nbre de terme
                          dans le docunent)

                      Inverse Document Frequency
 = log(Nbre total de doc /  Nbre total de doc contenant le terme t)
```

Il s'agit d'un schéma de pondération des termes. Cela réduit l'impact des mots qui apparaissent très fréquemment dans un corpus donné et qui sont donc empiriquement moins informatifs que les mots apparaissant dans une faible proportion du corpus d'apprentissage.

- **TruncatedSVD**: ce transformateur effectue une réduction linéaire de la dimension au moyen de la décomposition de la valeur singulière (SVD) tronquée (en mathématiques, factorisation des matrices rectangulaires réelles ou complexes).

- **word2vec**: C'est un réseau de neurones simple avec peu de couches utilisé pour produire la représentation du mot. Cette représentation garde le sens des mots grâce aux calculs de cosinus entre deux vecteurs.

Définitions des paramètres:

ngram_range: paramètre du count vectorizer ngram_range de (1, 1) signifie uniquement des unigramme, (1, 2) signifie des unigramme et des bigramme, et (2, 2) signifie uniquement des bigramme.

Exemple: `ngram_range=(1, 2)`

```
['00',
 '10',
 '10 10',
 '10 feet',
 '10 for',
 '10 grade',
 '10 minutes',
 '10 on',
 '10 out',
 '10 plus']
```

max_features: les *n* features les plus importants ordonnés par la fréquence des termes dans le corpus.

n_components: nombre de colonnes désirées et inférieur aux nombres de colonnes initial.
vector_size : dimension du vecteur
use_idf : active la re-pondération de l' idf

Exemple de GridSearchCV sur les vectorizers

In [7]:
```
#GridSearch for tfidf transformers, ngram_range, max_features


estimator = LatentDirichletAllocation(
        batch_size=128, # n docs in each learning iter
        doc_topic_prior=None,
        evaluate_every=-1, # compute perplexity every n iters, default: -1
        learning_decay=0.7,
        learning_method='online', #To change: to have the best parametres,
        learning_offset=10.0,
        max_doc_update_iter=100,
        max_iter=10,
        mean_change_tol=0.001,
        #n_topics à 20 ..then use GridSearch to find n_topics
        n_components=20,  # Number of topics to change
        n_jobs=-1,        # Use all available CPUs
        perp_tol=0.1,
    # Attention ...
        random_state=42, # Random state to change
        topic_word_prior=None,
        total_samples=1000000.0,
        verbose=0)

pipeline = Pipeline([
                    ("transformer", Pipeline([
                                        ("vect", CountVectorizer()),
                                       ("tfidf", TfidfTransformer())
                     ("clf", estimator)])

parameters = {
"transformer__vect__max_features": [50000,5000, 3000, 1000],
"transformer__vect__ngram_range": ((1,1),(1,2)),
"transformer__tfidf__use_idf": (True, False)
}


#Using RandomForestClassifier:
model1 = GridSearchCV(pipeline, param_grid = parameters, scoring = None)
model1.fit(X_train["Title"])

# Best Model
best_model1 = model1.best_estimator_# Model Parameters

print("Best Model's Params: ", model1.best_params_)# Log Likelihood Score
#out: {'learning_decay': 0.7, 'n_components': 10}
print("Best log-likelihood score: ", model1.best_score_)# mean accuracy
```

```
/usr/lib/python3/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarn
ing: `should_run_async` will not call `transform_cell` automatically in t
he future. Please pass the result to `transformed_cell` argument and any
exception that happen during thetransform in `preprocessing_exc_tuple` in
IPython 7.17 and above.
  and should_run_async(code)
```

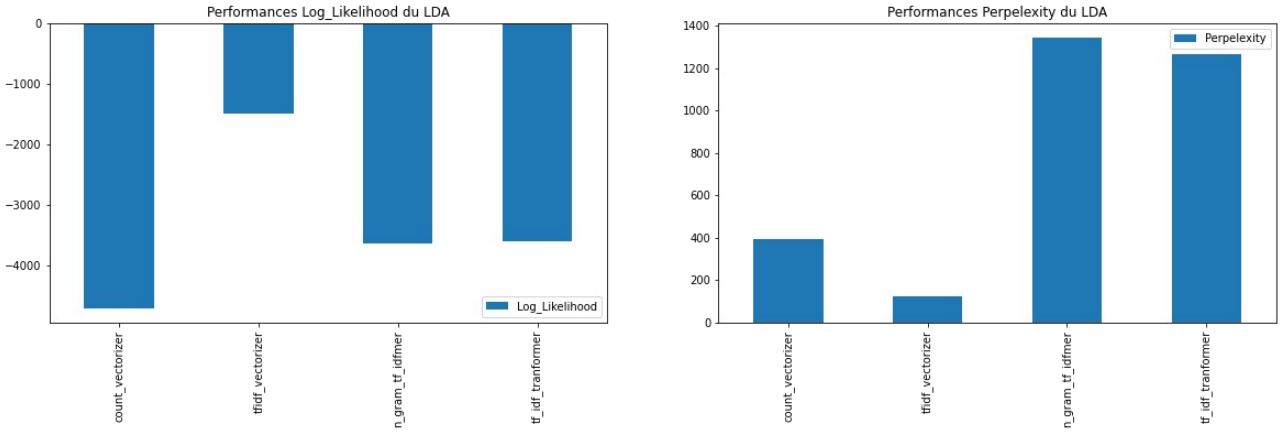
```
Best Model's Params:  {'transformer__tfidf__use_idf': False, 'transformer
__vect__max_features': 1000, 'transformer__vect__ngram_range': (1, 2)}
Best log-likelihood score:  -6665.79526336714
```

2.2 - Calcul des performances des modèles selon leurs types de vectorizers

Une boucle permet d'entraîner les modèles selon leurs différents types de vectorizers.

```
########################
#Initialize dict results
dict_Likelihood = {}
dict_Perplexity = {}

#list of models
models = [lda_model]

#list of methods
methods = []
methods.append(('count_vectorizer', count_vectorizer))
methods.append(('tfidf_vectorizer', tfidf_vectorizer))
methods.append(('n_gram_tf_idfmer', pipeline1_vec))
methods.append(('tf_idf_tranformer', pipeline2_vec))

for model in models:
    for name, vectorizer in methods :
        vectorizer.fit(X_train["Title"].values.astype('U'))
        feat_train = vectorizer.transform(X_train["Title"].values.astype
        feat_test= vectorizer.transform(X_test["Title"].values.astype('U


        output = model.fit(feat_train)
        dict_Likelihood[name] =model.score(feat_test)
        dict_Perplexity[name] =model.perplexity(feat_test)


#print(lda_model.get_params())
print("dict_Likelihood:", dict_Likelihood)
print("--------------------------------------")
print("dict_Perplexity:",dict_Perplexity)
```

Exemple de la LDA



A noter que pour la LDA, les vectorizers Word2vec et TruncatedSVD (avec Count vectorizer et Count + tfidf Transformer) ne peuvent fonctionner car ils produisent des matrices contenant des valeurs négatives. Ainsi, 4 vectorizers au total sont comparés au lieu de 7 vectorizers issus des approches supervisés.

2.3 – GridSearchCV du modèle choisi avec son vectorizer

Le GridSearchCV sera appliqué pour les approches suivantes:

- Approche non supervisée: LDA

- Approche combinée: OneVsRest SVC sur LDA

- Approches supervisées : OneVsRest SVC

- Approche supervisée : MultiOutput MultiNB

– Approche non supervisée: LDA LatentDirichletAllocation

On initialise **n_components** = 20, puis la GridSearchCV déterminera le nombre optimal de topics.

learning_decay est un paramètre qui contrôle le taux d'apprentissage dans la méthode d'apprentissage *on line*. La valeur doit être fixée entre 0,5 et 1,0 pour garantir une convergence asymptotique.

learning_method = *online* pour online variational Bayes algorithm.

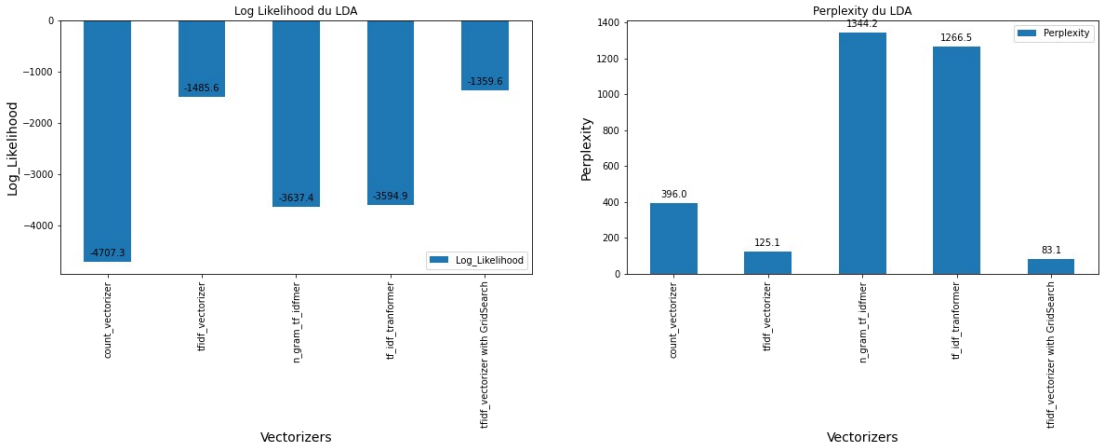
Mesures de performances:

- Perplexity
- Log Likelihood Score

Un modèle avec une log-vraisemblance plus élevée et une perplexité plus faible est considéré comme bon.

```
Perplexité
= exp[-1. * log-vraisemblance par mot]
```

Performance de la LDA après GridSearchCV



– Approche combinée : OnevsRest SVC sur les tags de la LDA

LDA en 10 topics avec 16 Tags

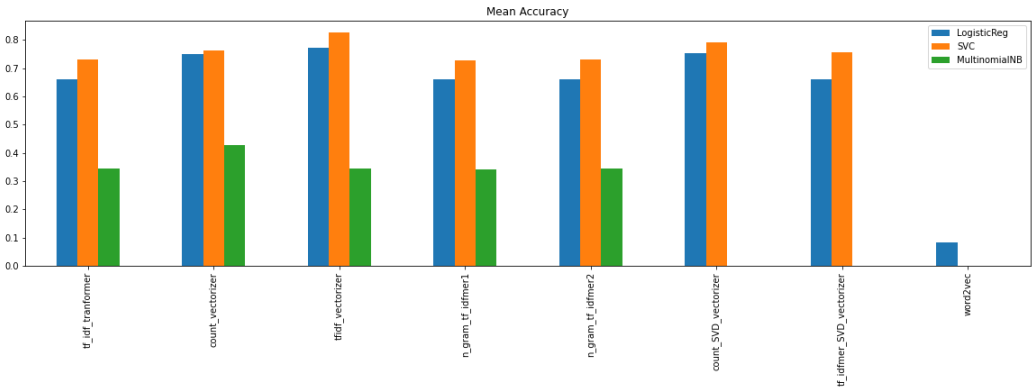
	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Topics
Topic 0	application	array	list	web	get	string	use	page	error	view	[arrays, string, error, web-development, web-s...
Topic 1	number	datum	javascript	find	cplusplus	add	base	difference	use	user	[javascript, c++]
Topic 2	image	value	string	base	form	database	php	cplusplus	change	function	[image, string, database, php, c++]
Topic 3	net	view	server	form	good	use	create	file	function	database	[.net, database]
Topic 4	file	android	create	cdiese	database	difference	table	variable	web	use	[android, c#, database, web-development, web-s...
Topic 5	iphone	app	page	work	change	good	use	file	add	get	[iphone]
Topic 6	code	function	table	array	use	list	user	string	add	database	[arrays, string, database]
Topic 7	get	error	php	make	user	file	android	web	use	change	[error, php, android, web-development, web-ser...
Topic 8	text	object	array	android	form	function	class	change	image	cplusplus	[arrays, android, class, image, c++]
Topic 9	use	problem	class	window	find	file	net	object	error	create	[class, windows, .net, error]

Puis, on entraîne et on calcul les performances de chaque modèles selon leurs type de vectorizers. Les 3 modèles testés sont :

- OneVsRestClassifier(LogisticRegression)

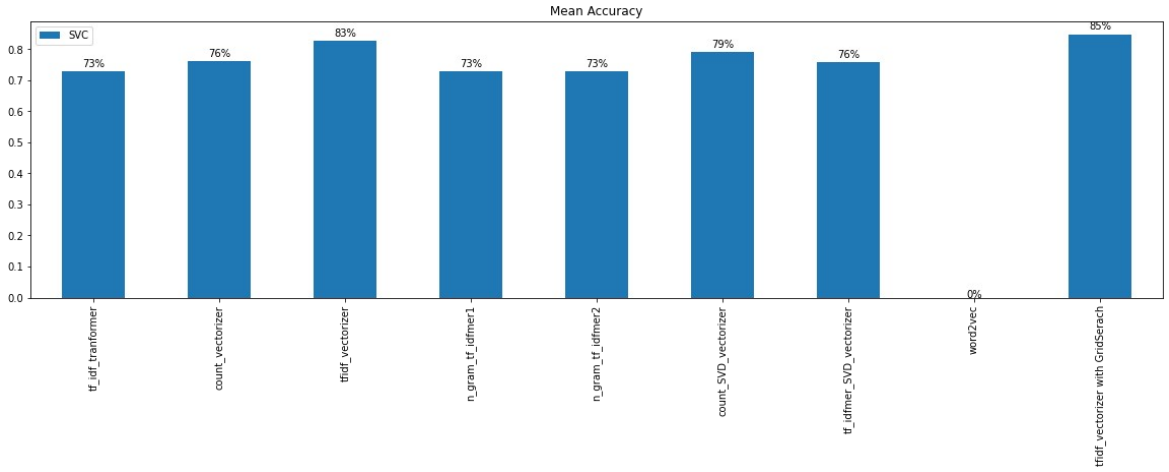
- OneVsRestClassifier(SVC)

- OneVsRestClassifier(MultinomialNB)



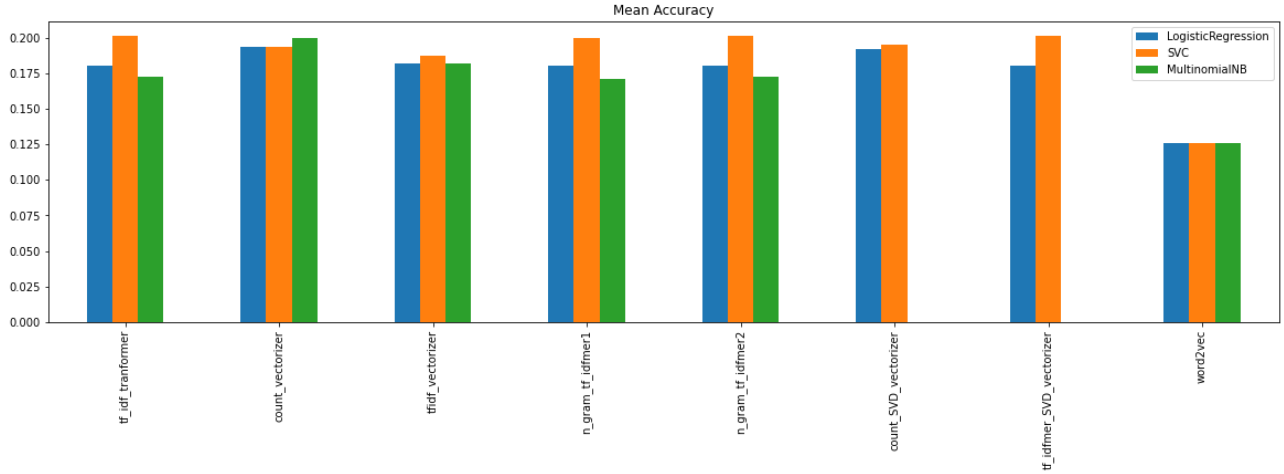
> modèle choisi: SVC avec `tfidf_vectorizer`

Pour la **GridSearchCV** du SVC, le paramètre à optimiser est `C`. C'est un paramètre de régularisation. La force de la régularisation est inversement proportionnelle à `C`. Il doit être strictement positif.



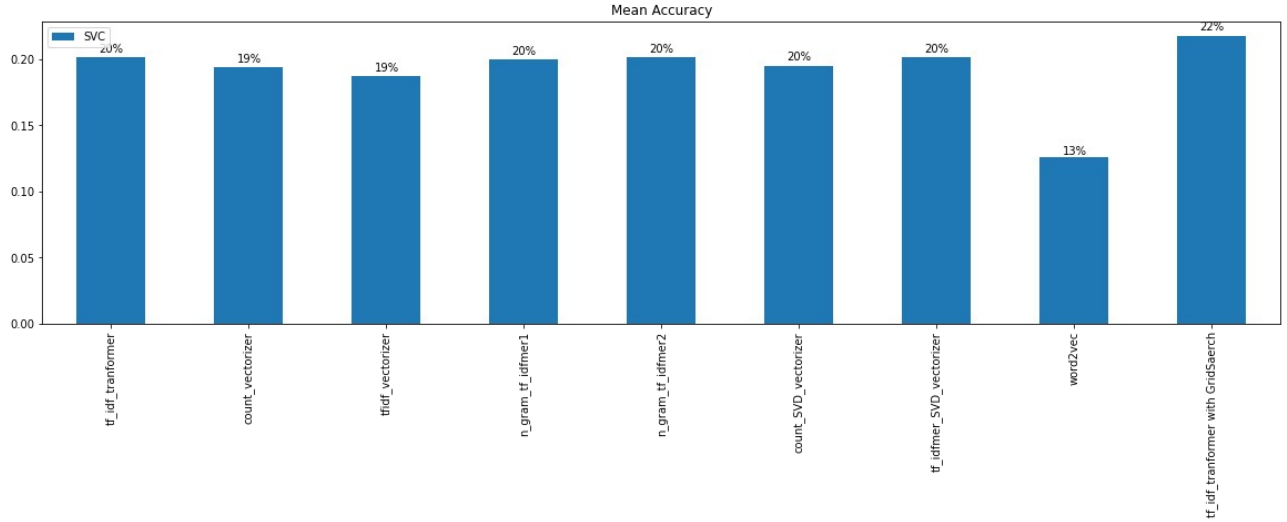
– Approche supervisée : OnevsRest SVC sur les tags initiaux

On répète le même procédé mais cette fois-ci avec les tags existants.



> SVC, le `tf_idf_tranformer`

Idem, pour la GridSearchCV du SVC, le paramètre à optimiser est `C`.



Avec la GridSearch, la Mean Accuracy est passée de 20% à 22% avec le `tf_idf_tranformer`.

– Approche supervisée : MultiOutput MultinomialNB

Toujours dans la prédiction sur les tags existants.

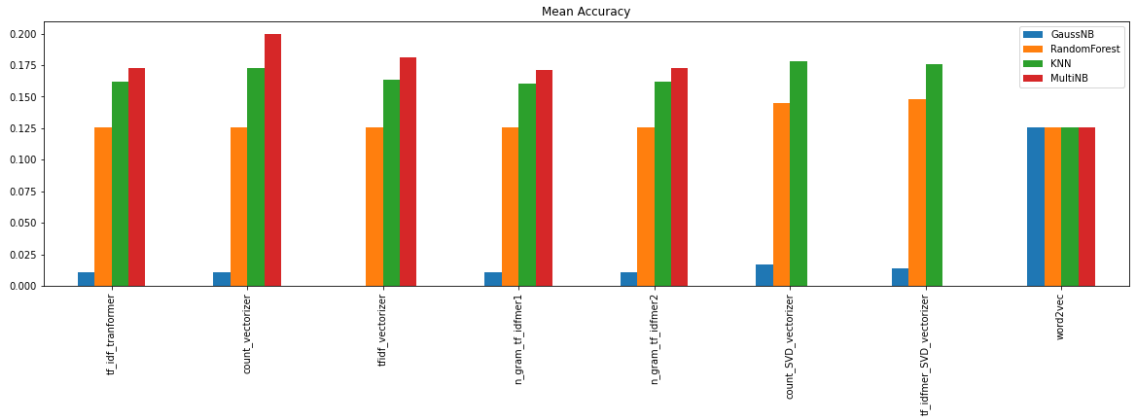
Les 4 modèles testés sont :

Le **KNN** classe les nouveaux individus par un vote majoritaire de ses k voisins. La classe est mesurée par une fonction de distance parmi ses K plus proches voisins.

Random Forest : pour classer un nouvel objet sur la base de ses attributs, chaque arbre donne une classification et on dit que l'arbre "vote" pour cette classe. La forêt choisit la classification ayant le plus de votes (sur tous les arbres de la forêt).

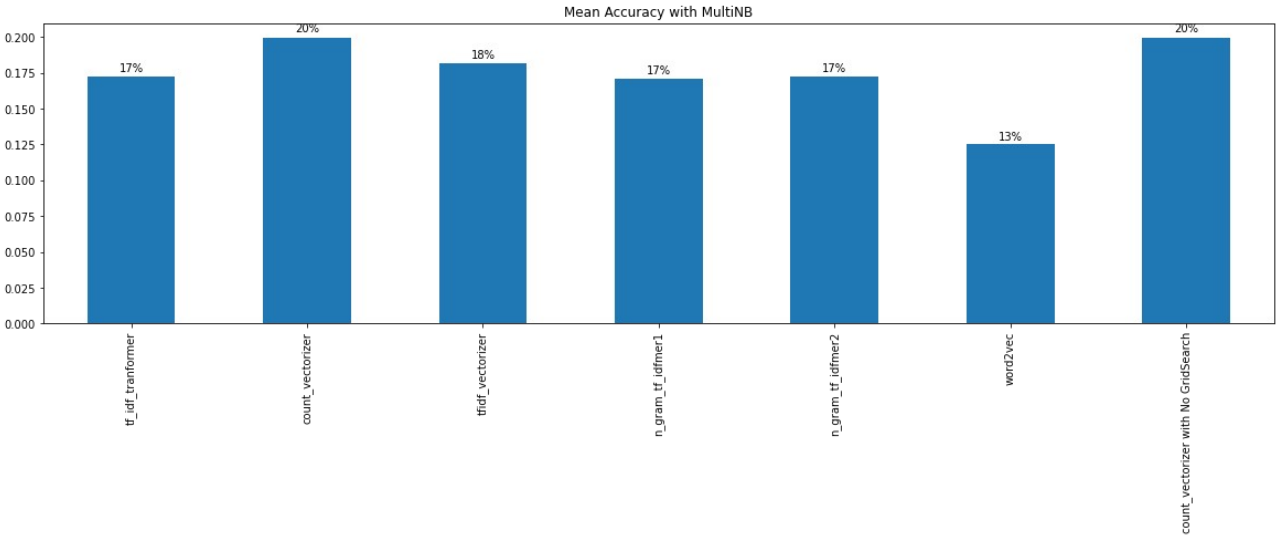
Gaussian Naïve Bayes: il s'agit d'une technique de classification basée sur le théorème de Bayes avec une hypothèse d'indépendance entre les prédicteurs.

MultinomialNB : il fonctionne de manière similaire au classificateur GaussianNB, mais les caractéristiques sont supposées être distribuées de manière multinomiale. En pratique, cela signifie que ce classificateur est généralement utilisé lorsque nous disposons de données discrètes.



> MultiNB avec `count_vectorizer`.

Le MultinomialNB ne possède pas d'hyper paramètres donc pas de GridSearch pour le **MultinomialNB.**



2.4 - Prédiction des Tags et calcul des scores

Nos modèles sont tunés et entraînés, on mesure leurs performances comme la **Mean Accuracy** ci-dessus, ensuite on mesure le **F1 Score** et le **Jaccard Score** une fois que les prédictions sur y_test sont faites.

Rappels

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Mean Accuracy : précision moyenne c'est-à-dire la part des "True Positive" sur la classe prédites "Positive".

Recall : Rappel c'est-à-dire la part des "True Positive" sur la classe réelle "Positive".

Score F1 : le score F1 peut être interprété comme une moyenne pondérée de la précision et du rappel, où un score F1 atteint sa meilleure valeur à 1 et son pire score à 0.

```
F1 = 2 * (précision * rappel) / (précision + rappel)
```

Score du coefficient de similarité de Jaccard : L'indice de Jaccard, ou coefficient de similarité de Jaccard, défini comme la taille de l'intersection divisée par la taille de l'union de deux ensembles d'étiquettes, il est utilisé pour comparer l'ensemble y_pred à y_true.

jaccard_score fonctionne comme precision_recall_f score s'appliquant nativement aux cibles binaires, aux multi labels et multi classes par l'utilisation de la moyenne.

La dernière étape consiste à comparer les 3 types de scores à savoir

Mean Accuracy,

F1 Score et

Jaccard score

pour les approches suivantes :

Approche combinée OneVsRest SVC sur la LDA

Approche supervisée OneVsRest SVC

Approche supervisée Multi-outputs MultinomialNB

Cette comparaison permettra de choisir le meilleur modèle pour la mise en production qui est l' approche combinée OneVsRest SVC sur la LDA.

L' approche non supervisée avec la LDA ne peut être comparée à aucun de ces 3 modèles. Elle permet uniquement de générer des tags de façon non supervisée.