Projet Open Food Fact

Advanced Machine Learning in NLP

YOUSAF Fajer

I. Importation de la base de données

La base de données de OpenFoodFacts a une taille supérieure à 4 Go d'où le fait que le téléchargement a pris du temps. De plus il fallait la charger sur notre IDE ce qui met encore plus de temps. Pour nous éviter de travailler sur toute la base de données,on la sépare en plusieurs échantillons de 50 000 lignes chacun ce qui va être plus simple d'utilisation et alléger notre programme.

Pour l'importation on a utilisé les librairies de base pandas et numpy.

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

Pour ce projet, nous allons travailler sur les 50 000 premières lignes. Le fichier est sous format CSV donc il est séparé en virgule et on ajoute la séparation par '\t' pour éviter les erreurs.

```
data=pd.read_csv('openfoodfacts_part1.csv',sep='\t' )
```

On va maintenant explorer la data, et déterminer les différentes variables, et leur nombre.

```
data.shape
```

Nombre de colonnes 187 et nombre de lignes 50000.

```
colnames  = data.columns.values
print(colnames)
```

On affiche le nom des colonnes, on remarque qu'il y a pas mal de colonnes liées avec le créateur du produit dans OpenFoodFacts et on sait que pour la suite ces données ne vont pas nous intéresser spécialement.

On va utiliser la fonction describe() de pandas afin d'afficher des informations sur les variables qui sont numériques. On a l'information sur le nombre de lignes remplies dans cette variable, la moyenne, l'écart-type, le minimum, le maximum et les différents quartiles.

Dans le tableau ci-dessous, on trouve les informations des premières variables.

	Unnamed: 0	created_t	last_modified_t	abbreviated_product_name	cities	allergens_en	serving_quantity	no_nutriments	additives_n	additives	...
count	50000.000000	5.000000e+04	5.000000e+04	0.0	0.0	0.0	37447.000000	0.0	38705.000000	0.0	...
mean	24999.500000	1.546501e+09	1.590064e+09	NaN	NaN	NaN	84.632737	NaN	2.872187	NaN	...
std	14433.901067	5.388986e+07	2.677677e+07	NaN	NaN	NaN	209.586509	NaN	3.551257	NaN	...
min	0.000000	1.335538e+09	1.416874e+09	NaN	NaN	NaN	0.000000	NaN	0.000000	NaN	...
25%	12499.750000	1.489076e+09	1.587584e+09	NaN	NaN	NaN	28.000000	NaN	0.000000	NaN	...
50%	24999.500000	1.574015e+09	1.587645e+09	NaN	NaN	NaN	42.000000	NaN	2.000000	NaN	...
75%	37499.250000	1.587665e+09	1.595084e+09	NaN	NaN	NaN	113.000000	NaN	4.000000	NaN	...
max	49999.000000	1.634083e+09	1.634083e+09	NaN	NaN	NaN	35371.000000	NaN	33.000000	NaN	...

8 rows × 128 columns

II. Nettoyage de la donnée

On remarque qu'il y a des colonnes qui sont complétement vides, ces derniers ne nous apportent aucune information pour la suite.

On va donc identifier les colonnes qui ont des données manquantes, lorsque dans ces colonnes il y a 60% des données manquantes on va considérer que la colonne n'est pas assez significative.

Pour pouvoir faire cela, on a utilisé la fonction (data.isnull().sum()) afin d'avoir le nombre de valeurs manquantes dans chaque colonne.

On va créer un data frame et stocker dedans cette information.

Pour la suite, on va créer un data frame où il y aura toutes les informations de notre data initiale mais on va retirer les colonnes qu'on a considéré comme inutile.

On a aussi retiré les informations liées aux créateurs et les informations qui sont sous forme URL car nous ne pourrons pas les exploiter.

	feature	valeur
0	Unnamed: 0	0
1	code	0
2	url	0
3	creator	0
4	created_t	0
...	...	...
182	choline_100g	49999
183	phylloquinone_100g	49855
184	beta-glucan_100g	50000
185	inositol_100g	49999
186	carnitine_100g	50000

187 rows × 2 columns

III. Identification des différents ingrédients

La première chose demandée pour ce projet est de définir et nettoyer le vocabulaire des ingrédients, identifier les erreurs possibles et les gérer. Et enfin proposer des solutions pour gérer/identifier les erreurs.

Pour ce faire, on va travailler sur la variable : 'ingredients_text' et on va créer un nouveau data frame où on trouve cette variable ainsi que le 'code' qui est la clé du tableau donc c'est une valeur unique pour chaque produit. On avait aussi ajouté la variable 'countries_tags' pour connaitre la langue avec laquelle ont été écrit les ingrédients.

	code	ingredients_text	countries_tags
0	00000000000000225	NaN	en:france
1	00000000000003429145	Leche semidesnatada, azucar 6.9% leche desnata...	en:spain
2	0000000000017	NaN	en:france
3	0000000000031	NaN	en:france
4	000000000003327986	NaN	en:spain
5	000000000004622327	NaN	en:spain
6	0000000000100	eau graines de téguments de moutarde vinaigre ...	en:france
7	00000000001111111111	NaN	en:france
8	0000000000123	NaN	en:france
9	0000000000178	NaN	en:france

On remarque qu'il y a des lignes vides ce qui signifie que les ingrédients ne sont pas définis et d'autre part ils ne sont pas tous indiqués dans la même langue.

On a donc décidé de traduire la variable ingrédient en français lorsque cela est possible. Pour la traduction, on a essayé de trouver des librairies en open source et dont l'utilisation est illimitée.

On a d'abord vu la librairie Translator, mais on a remarqué que celle-ci est limitée car on ne peut pas l'utiliser de manière illimitée donc avec nos 50 000 lignes de données l'utilisation de cette librairie est donc impossible.

On a ensuite vu TextBlob, le résultat n'était pas renvoyé en string mais en une variable textBlob donc cela ne va pas être possible de l'utiliser par la suite.
L'autre librairie utilisé est Goslate mais celle-ci aussi est limitée car on ne peut traduire qu'un nombre maximum de 500 caractères, de plus on ne peut pas l'utiliser de manière illimitée.

Pour notre travail, la meilleure librairie trouvée pour la traduction est Google Translator. Grâce à Google Translator, on n'a pas besoin d'indiquer la langue de départ car il détecte automatiquement la langue et on a décidé de tout traduire en français. On peut au maximum traduire 5000 caractères par chaine de caractères, il y a certaines lignes qui sont plus longues mais pour la suite nous allons les négliger.

```
traduction=[]

for i in range(0,50000):
    if ingredient['ingredients_text'].isnull()[i]==True:
        #print('vide')
        traduction.append("")
    if ingredient['ingredients_text'].isnull()[i]==False:
        if ingredient.iloc[i ,2]=="en:france":
            traduction.append(ingredient.iloc[i ,1])
        elif ingredient.iloc[i ,1][1] in ['0','1','2','3','4','5','6','7','8','9']:
            traduction.append("")
        else :
            translated = GoogleTranslator(source='auto', target='fr').translate(ingredient.iloc[i,1][:5000])
            #print(translated)
            traduction.append(translated)
```

On va initialiser une liste qu'on va remplir par des traductions de chaque ligne. On va effectuer la traduction lorsque la ligne n'est pas vide, et quand elle est remplie on vérifie si elle est déjà en français pour éviter une traduction inutile.

On remarque aussi des erreurs lorsque la ligne n'est composée que de chiffres, dans ce cas on n'effectue pas de traduction mais on remplit la liste d'un vide.

Pour la suite, on va séparer chaque ligne 'ingredient_text' en différent ingrédients et pour se faire on va utiliser la fonction split de la librairie Regex et on va mettre plusieurs séparateurs.

Avant d'utiliser la fonction on a essayé d'utiliser le Tokenize, mais on a vu que cela sépare la ligne en mots mais il existe des noms d'ingrédients composés.

Voici un exemple des résultats obtenus pour une ligne donnée avec les deux différentes techniques. Et le résultat qui nous intéresse le plus est celui qu'on retrouve avec la fonction split.

```
['Noix',
 'huile végétale,contient un ou plusieurs des éléments suivants ',
 'huile de canola',
 'huile de carthame',
 'huile de tournesol)',
 'sel',
 'extrait de romarin,']
```

```
['Noix',
 ',',
 'huile',
 'végétale',
 '(',
 'contient',
 'un',
 'ou',
 'plusieurs',
 'des',
 'éléments',
 'suivants',
 ':',
 'huile',
 'de',
 'canola',
 ',',
 'huile',
 'de',
 'carthame',
 ',',
 'huile',
 'de',
 'tournesol',
 ')',
 ',',
 'sel',
 ',',
 'extrait',
 'de',
 'romarin',
 '.']
```

Pour la suite, on souhaite avoir un dictionnaire d'ingrédients et pour faire cela on va créer une boucle qui va identifier les différents ingrédients et les inscrire dans une liste.

```
vocab_ingredient=[]

for i in range(50000):
    a=traduction[i].replace('(',',')
    a=a.replace(')',',')
    a=a.replace('[',',')
    a=a.replace(']',',')
    a=a.replace('.',',')
    a=a.replace(':',',')
    for j in range(len(re.split('; |, |•|: ', traduction[i]))):
        if re.split('; |, |•|: ', traduction[i])[j] not in vocab_ingredient :
            vocab_ingredient.append(re.split('; |, |•|: ', traduction[i])[j])
```

On a aussi essayé d'utiliser la technique de NLP Named Entity Recognition (NER), mais on a remarqué qu'elle ne permet pas d'identifier les catégories d'ingrédients mais elle permet plutôt de localiser et classer des entités nommées au sein d'un texte dans des catégories telles que des personnes, des organisations, des lieux, etc.

IV. Regroupement des produits

On va d'abord identifier les colonnes avec les informations nutritionnelles, et on les retrouve dans le descriptif des variables :

```
energy_100g energy-kj_100g energy-kcal_100g proteins_100g
casein_100g serum-proteins_100g nucleotides_100g carbohydrates_100g
sugars_100g sucrose_100g glucose_100g fructose_100g lactose_100g
maltose_100g maltodextrins_100g starch_100g polyols_100g fat_100g
saturated-fat_100g butyric-acid_100g caproic-acid_100g caprylic-
acid_100g capric-acid_100g lauric-acid_100g myristic-acid_100g
palmitic-acid_100g stearic-acid_100g arachidic-acid_100g behenic-
acid_100g lignoceric-acid_100g cerotic-acid_100g montanic-acid_100g
melissic-acid_100g monounsaturated-fat_100g polyunsaturated-fat_100g
omega-3-fat_100g alpha-linolenic-acid_100g eicosapentaenoic-
acid_100g docosahexaenoic-acid_100g omega-6-fat_100g linoleic-
acid_100g arachidonic-acid_100g gamma-linolenic-acid_100g dihomo-
gamma-linolenic-acid_100g omega-9-fat_100g oleic-acid_100g elaidic-
acid_100g gondoic-acid_100g mead-acid_100g erucic-acid_100g
nervonic-acid_100g trans-fat_100g cholesterol_100g fiber_100g
sodium_100g alcohol_100g vitamin-a_100g vitamin-d_100g vitamin-
e_100g vitamin-k_100g vitamin-c_100g vitamin-b1_100g vitamin-b2_100g
vitamin-pp_100g vitamin-b6_100g vitamin-b9_100g vitamin-b12_100g
biotin_100g pantothenic-acid_100g
silica_100g bicarbonate_100g potassium_100g chloride_100g
calcium_100g phosphorus_100g iron_100g magnesium_100g zinc_100g
copper_100g manganese_100g fluoride_100g selenium_100g chromium_100g
molybdenum_100g iodine_100g caffeine_100g taurine_100g ph_100g
vegetables-nuts_100g carbon-footprint_100g nutrition-score-fr_100g
nutrition-score-uk_100g
```

On avait observé au moment du nettoyage de la data qu'il y avait certaines colonnes non remplies donc nous les avions supprimées de notre nouveau data frame. Il reste donc 17 colonnes avec des données nutritionnelles :

```
|: Nutritionelle
```

	energy-kcal_100g	energy_100g	fat_100g	saturated-fat_100g	trans-fat_100g	cholesterol_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	1.40	0.90	NaN	NaN	9.80	9.80	NaN	2.70	0.1000	
2	375.0	1569.0	7.00	3.08	NaN	NaN	70.10	15.00	NaN	7.80	1.4000	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	163.9	685.8	1.90	1.00	NaN	NaN	NaN	NaN	NaN	15.30	1.1000	
...	...	...	...	...	...	...	...	...	...	...	...	
49995	220.0	920.0	5.00	1.00	0.0	0.010	38.00	2.00	4.0	8.00	1.4500	
49996	47.0	197.0	0.00	0.00	0.0	0.000	11.76	8.24	2.4	0.00	0.0000	
49997	47.0	197.0	0.00	0.00	0.0	0.000	11.76	7.06	3.5	0.00	0.0000	
49998	263.0	1100.0	10.53	5.26	0.0	0.026	26.32	3.51	3.5	12.28	1.3150	
49999	263.0	1100.0	10.53	5.26	0.0	0.026	29.82	1.75	3.5	12.28	1.2275	

50000 rows × 17 columns

Pour cette partie du projet, nous avons également besoin de faire apparaitre les catégories d'aliment, et donc nous avons créé un nouveau data Frame avec la variable 'main_category'. On remarque que les catégories ne sont pas toutes remplies et ne sont pas toutes de la même langue donc il a fallu effectuer une traduction des catégories et pour notre travail on a décidé de tout traduire en français.

	main_category
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
...	...
49995	en:pizzas
49996	en:fruits-based-foods
49997	en:fruits-based-foods
49998	en:mexican-dinner-mixes
49999	en:frozen-foods

On a essayé de recenser les différentes catégories de produit et on les a notés dans un dictionnaire puis dans un data frame. Au vu du traitement long, on a réduit notre jeu de données à 10 000 lignes. Et sur ces 10 000 lignes on voit qu'il y a 4351 produits dont les catégories ne sont pas remplies et il y a 859 catégories de produits en tout pour ces 10 000 produits. La catégorie qui vient en premier est « collation » si l'on ne compte pas les vides

```
:
```

	ProductName	Quantity
0		4351
174	collations	418
147	bonbons-chocolat	306
146	confiseries	290
358	Gâteaux	244
...	...	...
384	truites-fumées	1
387	boeufs-bourguignons	1
388	côtes de porc	1
389	merguez	1
858	fécule de maïs	1

859 rows × 2 columns

On a ensuite consolidé les deux Data Frame pour avoir sur le même data frame les informations nutritionnelles et les informations de la catégorie.

Après avoir réétudier la donnée on s'est rendu compte qu'il y avait une colonne plus efficace pour notre travail et c'est la colonne « main_category_en » et grâce à celle-ci on peut travailler sur toute notre dataset, soit les 50 000 lignes. On peut s'apercevoir que la catégorie la plus présente sur notre dataset sont les snacks

	ProductName	Quantity
0	NaN	13878
176	Snacks	3237
147	Confectioneries	1935
94	Sauces	1421
566	Frozen foods	1265
...	...	...
810	Pumpkins	1
817	Tahini	1
819	Canned carrots	1
832	Strawberry applesauces	1
1615	fr:nuggets-de-legumes	1

1616 rows × 2 columns

V. Idée de modèle pour Open Food Fact