11/2021 | ESILV A5 - DIA 3
CELIE Kévin - CHEICK ISMAIL Mariyam

Link to GitHub : https://github.com/m-cheicki/NLP_Project

Advanced Machine Learning for NLP and Text Processing

Project 1 : OpenFoodFacts

OpenFoodFacts can be considerate of a wikipedia for food ! The goal of OpenFoodFacts is to share with everyone a maximum of informations on food products. It contains more than 800000 products but maybe all products are not perfectly described... Mainly, for a product, we can find the list of ingredients, nutrition facts and food categories.

Project structure

- README.md : report
- openfoodfacts_qst1.py : script to run the first question using `python openfoodfacts_qst1.py` in the terminal
- CELIE_CHEICKISMAIL_OpenFoodFacts_part1.ipynb : for part 1
- CELIE_CHEICKISMAIL_OpenFoodFacts_part2.ipynb : for part 2

To run locally : be sure to have the dataset (4,5Go) on your folder :

```
PATH = ./datasets/openfoodfacts.csv
```

Also, if you run the script, you have to split the dataset into samples (we did it into 40 samples but you can set your range in the script). Before running the script, please ensure that you have a folder named v2, like so (to save the cleaned dataset):

```
./datasets/v2/
```

There is a great chance that the script doesn't work on your machine as it uses lots of resources. It did worked once or twice in ours but does not anymore so even if it works on your, maybe one of our processing step will stop it, as we couldn't test it in the whole dataset. (We have tested on partial dataset and seemed to work).

Define and clean the vocabulary of ingredients

Define and clean the vocabulary of ingredients, do you find some mistakes ? How do you manage them ? Propose solutions to manage/identify errors.

Our code implementation for this question are in the CELIE_CHEICKISMAIL_OpenFoodFacts_part1.ipynb notebook in this repository.

First, this dataset is composed of almost 2 millions of lines and 187 columns. Here is the documentation of the dataset.

***Note**: Note that this documentation is not up-to-date, some columns are not present in the docs but are in the dataset. Also, there is some incoherence we have faced. For instance, in the docs it is said that the CSV containing the dataset is encoded in UTF-8 but when we have imported it we found some encoding issues (we can't say if this is from the data itself before being exported into CSV file, or during the export of the dataset).*

Cleaning dataset

Before answering this question some preprocessing steps have been done.

Importing 2M lines can be quite fat but having some processing steps on all these lines can be very fastidious. So, we have splitted the initial dataset into 40 samples.

Also, lots of columns were either empty or almost empty. Thus, for the purpose of this project, some columns/data weren't necesssary.

So by default, we have decided to delete these columns as we are not going to use them :

```
['Unnamed: 0', 'url', 'code', 'creator', 'created_t', 'created_datetime', 'last_modified_t','last_modified_datetime', 'abbreviated_p
```

Then, we droped columns that were filled with less than 20% of the whole dataset.

In the meantime, we also wanted columns to be mandatory :

```
'product_name', 'categories_tags', 'ingredients_text', 'additives_tags', 'nutriscore_score', 'nutriscore_grade', 'nova_group', 'pnns
```

These columns represent : - Product name - List of ingredients - Food Additives - Nutriscore (score and grade) - Nova group - PNNS groups (large and sub-categories), Food Categories and Main categories - Nutrition Facts (the most common in our packages, at least)

After deleting columns, we wanted to explore deeper our dataset and we have found out that some encoding issues were present. To correct them, we have replaced manually some problematic characters.
During our tests, we have checked the encoding row per row of the dataset and we found that we had `UTF-8` as mentionned in the docs but also `latin1`, `ascii`, `iso-8859-1`, `cp1252` ... which can explain weird and special characters.

We have also seen that in the ingredient list, some information contains quantities, percentages or additves (and others digit information) which was problematic for the next steps, so we have deleted them at the same time. By doing so, we could say that there is loss of information but additives have been drawn up in the `additives_tags` column, some quantities (grams and percentages) are sometimes mentionned in the corresponding column (+ we will see that we are not going to use them as we have nutrition facts for 100g).

TODO: During the next steps we found some other things that needs to be cleaned: - Some duplicates are present. We have tried to drop them but it didn't work well for the moment. - Some people put incoherent values for nutrition facts : we have a temporary fix for it but it is not ideal. - Some ingredients are not ingredients : URL/comments (`'pls look at pic'`)

Detect languages

As OpenFoodFacts is a project developed by thousands of volunteers accross the world, we can find products comming from different countries, which also means in different languages. Even though we have taken the France link to have French products only (as we thought at first) we found out that some products are in different languages like english, spanish, portugese, italian...

In order to know better our dataset and filter afterwards, we wanted to detect automatically the language of each product. To do so, we have used different packages to test them and finally chose `langdetect` .

In our tests, it took 2h15 to detect the language of almost 2M entries.

Note: It is not perfectly accurate but it is doing pretty well its job. Some issues occured in this step because some ingredients started with digits or were links and you cannot predict a language on numbers and URL (yes, some people found it funny to put some links in the ingredient links from TikTok, which unfortunately does not work :'(, or from commercials).

For the further steps, we have taken two approaches : - Either take only those in english as it is easier to check spelling mistakes - Or translate every row that are not in english and make the dataset unilingual

These two approaches have their own pros and cons : - First approach : - Main pro : unilingual (English) and its vocabulary/dictionnaries are well developped - Main con : this means a big loss of our dataset as only 15% of the dataset is in English - Second approach : - Main pro : unilingual and without loss of many entries of our dataset - Main cons : - Finding APIs/packages that does the trick (some have character limitation) - Accuracy of the translation ? - Time of processing (tried in a little sample, by calculus we found that we needed almost 10 days non-stop to translate every row, which we can't afford to do)

In our case, we decided to go for the first option : keep only those in English. *The code for the translate part is in comment.*

Handling mistakes

For this part, we have taken three different approaches : - Use NLTK's corpus vocabulary - Use SpellChecker, a python package - A manual approach

NLTK corpus vocabulary

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

Observations :

This method is quite simple to use. We retrieve words from the NLTK's corpora and we compare them to the words we have. But by doing so, we have got some unexpected behaviors : - even though we put a condition saying that if the element we want to check is already in the vocabulary, you don't have to check the spelling (if the word exists, this word is already correctly spelled), sometimes it checks : `green ==> green` . Hopefully, it gives the same results which is nice. - it doesn't recognise some words like `vinegar` . In our tests, we had `vingar ==> dingar` which means *The giant honeybee, Apis dorsata, native to South and South-East Asia, which is noted for aggressive defence of its nests, which typically hang from trees, cliffs, and buildings*. It can be understandable as there is only one letter difference `v ==> d`

There are also other issues but these are because of the dataset, for instance : - `containslessthanof` should be `contains less than of` - Some words are correctly spelled but the correction is not pertinent : `all-purpose ==> apurpose` , `hours ==> cours` - Even though we were very specific by filtering by language, keeping only those in English, some French words appeared and the vocabulary couldn't check the spelling correctly : `farine ==> parine`

Although, there are issues, some words were correctly spelled like `monfat ==> nonfat` , `northn ==> north` ...

```
 vanillan ==> vanilla
monfat ==> nonfat
lychs ==> lycus
trate ==> irate
carrangnan ==> caranna
lutr ==> lut
varying ==> warding
exct ==> exact
diglcid ==> diacid
farine ==> parine
un-ch ==> nunch
hucklry ==> hackery
aglycid ==> glycid
ard ==> dard
salisry ==> salish
dipotass ==> potass
knal ==> knap
goats' ==> goaty
butyloctyl ==> tylostyle
ocesses ==> acestes
hydroylz ==> hydrol
aflavors ==> flavory
dipot ==> divot
mst ==> tst
northn ==> north
containslessthanof ==> curtainless
vingar ==> dingar
aminos ==> minos
...
starcn ==> starch
isugar ==> sugar
cocoapowd ==> cocowood
ithout ==> without
phophate ==> phosphate
```

We couldn't run this method longer, it takes too much time to process. And the results were not as good as wanted.

SpellChecker

Pure Python Spell Checking based on Peter Norvig's blog post on setting up a simple spell checking algorithm.

It uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are more likely the correct results.

Observations :

This method is as easy to use as the first one. First, we check if the word is misspelled. If it is, we use the correction function that proposes multiple choices ordered by descending probabilities. For our tests, we have taken the first choice as it is the highest probability to be the correct spelling we want.

As expected, we found some issues/incoherence due to the dataset: - `exct ==> exact` instead of `extract`
- `farine ==> marine` from a french word

And sometimes, it says that the initial word is misspelled although it is not : `dipotass ==> dipotass` , `carrangnan ==> carrangnan` . It is not problematic but it takes some additional process time.

But globally, it does pretty well its job :

```
 vanillan ==> vanilla
monfat ==> nonfat
lychs ==> lochs
trate ==> trade
carrangnan ==> carrangnan
lutr ==> lute
exct ==> exact
diglcid ==> diploid
farine ==> marine
un-ch ==> bunch
hucklry ==> hickory
aglycid ==> aglycid
```

salisry ==> satisfy
dipotass ==> dipotass
knal ==> anal
goats' ==> goats
butyloctyl ==> butyloctyl
ocesses ==> dresses
hydroylz ==> hydroxyl
aflavors ==> flavors
dipot ==> depot
mst ==> must
northn ==> north
containslessthanof ==> containslessthanof
vingar ==> vinegar
aminos ==> amino
lgian ==> lian
spaning ==> sparing
comtry ==> country
swai ==> swap
frk ==> fry
uals ==> pals
stl-cut ==> stl-cut
zn ==> in
onctrate ==> nitrate
granulat ==> granular
arul ==> aru
acontains ==> contains
ngre ==> ogre
amountisving ==> amountisving
jarlsrg ==> jarlsrg
isotyrate ==> isotyrate
lithin-anulsifi ==> lithin-anulsifi
sorae ==> sore
cocao ==> cocoa
cholula ==> cholla
lactlate ==> lactate
rirose ==> hirose
xanthai ==> bantha
sodiumphosphat ==> sodiumphosphat
alumina ==> alumna
anch ==> inch
monocalcium ==> monocalcium
convtiona ==> convtiona
currart ==> currant
hci ==> hi
cormn ==> corn
gattigte ==> gattigte
sike ==> like
cantre ==> centre
ouality ==> quality
powd ==> pod
lilhin ==> within
thc-arm ==> thc-arm
goonlijk ==> goonlijk
ylowfin ==> yellowfin
suf ==> sun
hagisutiyfat ==> hagisutiyfat
xanthang ==> anthing
canfeine ==> caffeine
uil ==> oil
chflour ==> colour
corn-ack ==> corn-ack
cyclohasiloxane ==> cyclohasiloxane
spr ==> sir
sning ==> sing
agt ==> at

```
...
```

About the process time, this method is way faster than the first one. Also more accurate, we can say that this method better than the first one.

Manual approach

For this approach, we have decided to correct misspelled word using word frequency. The more the word is present the more it has to be the correct spelling.

For instance :

```
'alchol': 1,
...
'alcohol': 448,
'alcoholic': 6,
```

We can correct `alchol` and `alcoholic` by `alcohol`.

Another example is :

```
'annaito': 1,
'annat': 2,
'annato': 20,
'annatt': 1,
'annatto': 2626,
```

We can correct all these words by `annatto`.

Observations :

This method is quite simple to put in place : you put all word in a lit that you sort by alphabetical order and then you count all occurences. But it has its limits. To easily compare and have better results, misspelled letter should be in the end of the word. For example, if we want to correct `acontains` by `contains` we can't as `a` and `c` are not the same letter. The task becomes too fastidious as the dataset is very big. Or, we have to calculate distance (Euclidian, Levenshtein, Jaccard, Longest Common Substring or Hamming for example) word by word and take the minimum distance and replace it by the word that is more present. We just have to take in account that some words can be present only once and cannot be calculated/spelled correctly if it is not already.

To go further

There are some steps that we haven't tried but we could have :

- Lemmatization : consists in finding the root of inflected verbs and reducing plural and/or feminine words to the masculine singular form.
- Stemming : consists in keeping only the root of the word

We have decided to test stemming. By doing so, we can regroup words by their roots and then correct their spelling. For example, `alcohol` and `alcoholic` can be grouped in `alcohol`

Reminder: Our code implementation for this question are in the CELIE_CHEICKISMAIL_OpenFoodFacts_part1.ipynb notebook in this repository.

Clustering approaches

Our code implementation for this question are in the CELIE_CHEICKISMAIL_OpenFoodFacts_part1.ipynb notebook in this repository.

Based on nutrition facts and/or food categories, propose clustering approaches and a visualisation of some categories of products. Find outliers (a product very different from others of the same group). It exists products very similars in terms of nutrition facts but very different in terms of categories or ingredients ?

Our approach

The approach we had here is to find similarities and differences between products from a same category. The question we asked ourselves was : **Is there common ingredients and nutritions facts that explains that category?**

To do so, we have selected our columns (nutrition facts):

```
"energy-kcal_100g",
"energy_100g",
"fat_100g",
"saturated-fat_100g",
"carbohydrates_100g",
"sugars_100g",
"fiber_100g",
"proteins_100g",
"salt_100g",
"sodium_100g"
```

We have first started with `Cereals and potatoes` and we compared with `sugary snacks` (these are the two main categories of our dataset after cleaning).

Cereals and potatoes

As we want a custering approach, we wanted to try a K-Means at first.

We have more than two dimensions and we want to study all our features together so we first did a PCA (Principal Component Analysis) in order to reduce our dimensions we a minimum loss of information.

Our PCA permit to have two dimensions with :

```
Explained variation per principal component: [0.99633303 0.00126498]
Cumulative variance explained by 2 principal components: 99.76%
```

We also fixed to have 5 clusters (at first it has determined 8 clusters but they were hard to interpret). We have found some common traits from products that are in the same cluster but sometimes these traits can be found in another cluster : some products have high energy and bad nutriscore and these products are in the same cluster ; some products have high energy and high sugar levels and are in the same cluster, but we can find products that fill conditions to enter in a specific cluster but it is in another one... (to compare we have taken to extremes of each cluster for each nutrifact)

We have observed that `energy` was the column that permit to make clusters but the other columns were a little bit off, which was problematic. So we decided to apply log function on these values to reduce magnitude between different nutrifacts as we want them to be equally important.

By doing so, the results were not the same as the second try. Results are graphically more like what we have expected. But it is still hard to understand how the cluster are made altough we can see resemblance between nutrifacts on a same cluster. After looking at ingredients, it seems like there no similarities or are negligeable between ingredients and nutrifacts of a same cluster.

Sugary snacks

We did the same step as explained in the `Cereals and potatoes` part. Results were better but sometimes the same product is at the same time in two different clusters, which is impossible for K-Means in our case. So we trying another model with only 4 clusters instead of 5 could be interesting.

Another model

We wanted to try other model like DBSCAN but the results weren't good enough to be interpretable.

To go further

Another clustering approach that can be interesting is to find similarities and differences between ingredients and nutrifacts of all products and try to cluster them and see if we can determine categories like `Beverages` or `Sugary snacks` for example.

Difficulties encountered

- Dataset cleaning is not totally complete, some values are too weird to be treated this way
- Some values of the dataset are empty. This makes the analysis harder to interpret
- We had tried several approach like finding the nutriscore / nutrigrade thanks to nutrifacts, clustering by categories in order to find similarities between products... but the results were as good as wanted
- We have tried different algorithms with different parameters but they weren't interpretable.

Reminder: Our code implementation for this question are in the CELIE_CHEICKISMAIL_OpenFoodFacts_part2.ipynb notebook in this repository.

Enhance OpenFoodFacts

Based on your expertize on this dataset, propose and describe a model (no code required) that would be interesting to enhance the OpenFoodFacts project.

To enhance OpenFoodFacts we can propose these approaches, due the difficulties encountered.

The model itself

The OpenFoodFacts database contains columns that are not filled or with very few information that is usually not interesting like `abbreviated_product_name`, `generic_name`, `cities` ...

Also, some columns have redundant information like `categories`, `categories_tags`, `catgories_en` / `countries`, `countries_tags`, `countries_en`, etc.

It would be interesting to have less columns but with information without redundance.

Encoding

As described above in the first part, we face encoding errors. To ensure the encoding is the same in all CSV, it would be great to insert new entries with the correct encoding.

Unilingual

During our study we had to keep only those in English to have an homogeneous dataset and have better results espacially for word spelling mistakes. As it is a worldwide database, it could be interesting to have a translation of the product/ingredients in English, in order to have an uniligual information of the food diversity across the world.

Limit duplicates

Make sure that a product does appear more than once in the dataset, by indexing code and product name for example.

Control entered information

Barcode

The barcode can give lots of interesting information like the country from where it comes from, the manufacturer number, the item number. We can use barcode to fill the country column for instance.

Useful links : - (en) http://www.computalabel.com/aboutupc.htm - (en) http://www.computalabel.com/aboutean.htm - (en) http://www.computalabel.com/about128.htm - (fr) https://www.bioalaune.com/fr/actualite-bio/36197/comment-dechiffrer-code-barres-etiquettes

Nutrition facts

As we have seen above, some values didn't make any sense. This could be limited by the form when you try to insert a new product. For 100g of product, you will not be able to put 500g of sugar, or put higher quantity of sugar than carbohydrates.

Also, as it is a mandatory information on every food packages, making them necessary permits to have better knowledge about the product.

Ingredients

Limit special characters. URL should not be able to be put in this field. Also, to avoid things like `pls look at pic`, it would be great to have a system that verify if the input is really ingredients and not comments like this.

Ingredients and Nutrition facts

If the user decide to put an image of the list of ingredients and the nutrition facts table, it would be a good idea to put some computer vision with convolutional neural networks (CV with CNN) in order to fill the ingredients and the nutrition facts field with less effort and probably with less mistakes. Of course, to do so, the image has to be readable and the user can correct some mistakes. By doing so, we limit weird inputs on those fields.

Unlinked products

It would be nice to delete rows were the product doesn't exists anymore. Links returns empty results which is not a good user experience and can be difficult to find why a product is problematic if it doesn't exist anymore.