

Extraction des listes d'ingrédients depuis les fiches techniques de
produits alimentaires

Pierre MASSÉ

Juin 2020

Résumé

La gestion de l'information produit est devenue un enjeu de société majeur ces dernières années. Les scandales sanitaires récents ont déclenché une prise de conscience collective des consommateurs, en parallèle de la mise en place de réglementations de plus en plus contraignantes pour l'ensemble des acteurs de la filière [3][4].

La construction et le déploiement d'outils mettant en oeuvre les principes du Machine Learning appliqués au traitement du langage permettrait d'aider les opérationnels de la gestion de l'information à interpréter plus efficacement les documents mis à disposition par les fournisseurs du groupe. Le présent rapport détaille la mise en place d'un outil permettant d'extraire les listes d'ingrédients des fiches techniques transmises par les fabricants des produits.

Dans une première partie, on explicitera les objectifs et le cas d'usage retenu. Nous analyserons ensuite les données disponibles, et la manière de les exploiter. La troisième partie sera consacrée à la construction du modèle à proprement parler, et la dernière partie portera un regard critique sur les résultats obtenus et ouvrira la réflexion sur les travaux à venir.

TABLE DES MATIÈRES

I Les objectifs	5
1 Les cas d'usage	5
1.1 Objectifs : Qualité et productivité	5
1.2 La préalimentation d'information	5
1.3 Le contrôle des informations transmises	6
1.3.1 Le contrôle à la saisie fournisseur	6
1.3.2 L'aide aux vérifications Pomona	6
1.3.3 Les contrôles en masse asynchrones	8
2 Présentation rapide des données relatives au cas d'usage	8
2.1 Les listes d'ingrédients	8
2.2 Pièces jointes	9
2.2.1 Fiches techniques fournisseur	10
2.2.2 Étiquettes produit	11
2.3 Récapitulatif de la complétude des données	12
2.3.1 Complétude des listes d'ingrédients	12
2.3.2 Complétude des pièces jointes	13
2.4 Les données « manuellement étiquetées »	13
2.4.1 Pour répondre à quel besoin ?	13
2.4.2 Mode de constitution de l'échantillon	13
2.4.3 Méthodologie de l'étiquetage manuel	14
2.4.4 Règles de gestion pour l'étiquetage manuel	14
2.4.5 Confrontation avec le contenu du PIM	14
3 Le choix du cas d'usage	15
3.1 Étiquettes contre fiches techniques	15
3.1.1 La représentation dominante des fiches techniques	15
3.1.2 L'extraction de données textuelles	15
3.2 Impossibilité de produire des templates pour l'exploitation de ces documents	17
3.3 Quelles données ?	18
3.3.1 Les principales données et leur mode de présentation dans les fiches techniques	18
3.3.2 L'identification d'une liste d'ingrédient par son contenu	18
3.4 Quels produits ?	18
3.5 Conclusion quant au choix du cas d'usage	19
II Analyse des données textuelles	20
4 Caractéristiques générales du jeu de données	20
4.1 Distributions des longueurs des textes	20
5 Analyse textuelle	22
5.1 Mots de chacun des corpus	22
5.2 Répartitions des mots dans et hors des listes d'ingrédients	22
5.3 Représentations graphiques des textes et mots	24
5.3.1 Vectorisation des mots	24
5.3.2 Vectorisation des textes	26

III Construction du modèle	30
6 Prototypage	30
6.1 Premier prototype : modèle simple « ouvert »	30
6.1.1 Principes généraux	30
6.1.2 Entraînement	31
6.1.3 Prédiction	32
6.1.4 Illustration des résultats obtenus	33
6.2 Second prototype : industrialisation	35
6.2.1 Chargement des données manuellement étiquetées	36
6.2.2 Découpage des textes en blocs	37
6.2.3 Entrainement et prédiction	38
6.2.4 Illustration des prédictions obtenues	38
7 Mesure de la performance	41
7.1 Accuracy	41
7.1.1 Approche naïve	41
7.1.2 Avec du « text-postprocessing »	43
7.2 Fonctions de « similarité » spécifiques	45
7.2.1 Similarité basée sur la distance de Levenshtein	45
7.2.2 Similarité basée sur la distance de Damerau-Levenshtein	46
7.2.3 Similarité de Jaro	46
7.2.4 Similarité de Jaro-Wrinkler	46
7.2.5 Évaluation de ces similarités sur la ground truth	46
7.2.6 Décision sur la métrique à utiliser et illustration	47
7.2.7 Performance en fonction de la longueur de la liste d'ingrédients	47
8 Ajustement des paramètres	49
8.1 Description des paramètres ajustables	49
8.1.1 Text-preprocessing	49
8.1.2 Découpage du texte des documents en blocs	49
8.1.3 Vectorisation des textes	50
8.1.4 Calcul de similarité	50
8.2 Ajustements et résultats	53
8.2.1 Ajustement du text-preprocessing	53
8.2.2 Tuning du découpage du texte des documents	54
8.2.3 Comparatif des fonctions de similarité	54
8.2.4 Comparaison des méthodes de vectorisation	57
8.2.5 Utilisation d'embeddings	62
8.3 Évaluation finale de la performance	62
IV Travaux subséquents	65
9 Extension des fonctionnalités offertes	65
9.1 Amélioration de la performance du modèle	65
9.1.1 Découpage du contenu des documents en blocs de textes	65
9.1.2 Amélioration de l'identification par similarité	66
9.1.3 Filtrage des similarités trop basses	66
9.2 Extension du périmètre couvert avec la méthode « textuelle »	67
9.2.1 Prise en compte de nouveaux types de pièces jointes	67
9.2.2 Utilisation d'outil d'OCR pour les pdf non structurés	67
9.2.3 Évaluation de la performances sur d'autres familles de produits	67
9.2.4 Récupération des dénominations réglementaires	68
9.3 Changement de méthode	68

9.3.1	Mise en place d'outil de spatialisation des textes	68
9.3.2	Aides pour le contrôle de cohérence	68
10	Déploiement opérationnel de l'outil	69
10.1	Mise en place d'une organisation projet	69
10.1.1	Sponsor et Client	69
10.1.2	Choix techniques et technologiques	69
10.1.3	Développement microservice	70
10.2	Maintien en condition opérationnelle	71
10.2.1	Mise à niveau	71
10.2.2	Intégration et déploiement continu	71
10.2.3	Monitoring de la performance du modèle	71
10.3	Conclusion	72
V	Annexes	73
A	Figures, tableaux et bibliographie	73
B	Contexte métier : le groupe, information produit et généralités sur les données	77
B.1	Description du groupe	77
B.1.1	Le métier du Groupe Pomona	77
B.1.2	La décentralisation	78
B.2	La gestion de l'information produit	82
B.2.1	L'information produit	82
B.2.2	Le processus	88
B.2.3	Les outils informatiques associés	93
C	Le périmètre produit	98
C.1	Les produits non-alimentaires	98
C.2	Accessibilité de la donnée en fonction des branches	99
C.3	Analyses quantitatives	99
C.3.1	Comparatifs entre les branches	99
C.3.2	Les grands types de produits	100
D	Les données utilisables, issues du PIM	101
D.1	Données structurées	104
D.1.1	Description des données structurées	104
D.1.2	Analyse de ces données structurées	105
D.2	Données non structurées	111
D.2.1	Les libellés	111
D.3	Analyse qualitative des données	111
D.3.1	Évaluation de la qualité des données	111
D.3.2	Types de pdf possédés	112
E	Exemples de pièces jointes et ground truth	112
E.1	Fiches techniques	112
E.1.1	Fiche technique sel Cerebos	113
E.1.2	Fiche technique olives Valtonia	114
E.1.3	Fiche technique Panna Cotta Nestlé	115
E.1.4	Fiche technique confiture Andros	117
E.1.5	Fiche technique ciboulette La case aux épices	119
E.1.6	Fiche technique poivron El Arenal	121
E.1.7	Fiche technique mélange trappeur Terre Exotique	124
E.2	Étiquettes produit	125

E.2.1	Étiquette curry Grain d'ailleurs	125
E.2.2	Étiquette madeleines Saint Michel	126
E.2.3	Étiquette lentilles Soufflet	127
E.2.4	Étiquette pannacotta Nestlé	128
E.2.5	Étiquette sauce soja Kikkoman	129
E.2.6	Étiquette mélange trappeur Terre Exotique	130
E.3	Étiquetage manuel des données	131
E.3.1	Règles de gestion pour l'étiquetage	131
F	Résultats du modèle retenu	135
G	Les notebooks de ce projet	140
G.1	Génération de l'échantillon de données manuellement étiquetées	141
G.2	Analyse des données textuelles	147
G.3	Modèle « ouvert »	168
G.4	Modèle basé sur les données manuellement étiquetées	174
G.5	Mesure de la performance	180
G.6	Ajustement des paramètres du modèle	190
G.7	Analyse quantitative	218
G.8	Analyse des données du PIM	226
H	Le code des différents modules	244
H.1	Gestion du fichier de configuration - Module conf	244
H.2	Extraction des données du PIM - Module pimapi	245
H.3	Conversion des pièces jointes en textes - Module pimpdf	248
H.4	Transformateurs et estimateurs spécifiques - Module pimest	250

Première partie

LES OBJECTIFS

L'objet de ce mémoire est de présenter si et comment le machine learning pourrait appuyer le Groupe Pomona dans la gestion des informations produit qu'il commercialise. Le groupe et sa gestion de l'information produit au travers du système PIM sont présentés en annexe B page 77.

Chapitre 1

LES CAS D'USAGE

1.1 Objectifs : Qualité et productivité

La gestion de l'information produit est un processus lourd, avec des étapes de contrôle redondantes visant à assurer la qualité de la donnée (cf. annexe B.2.2 page 91), en particulier la cohérence des informations transmises (données saisies) avec le contenu des pièces jointes (documents pdf) mises à disposition. Un traitement automatique des pièces jointes permettrait de décharger les personnes qui interviennent dans le processus (fournisseurs, acheteurs, gestionnaires de référentiels, ingénieurs qualité) et de garantir une meilleure pertinence de l'information produit.

1.2 La préalimentation d'information

Une des manières de gagner en productivité et en qualité serait :

- de modifier l'écran de saisie des données par le fournisseur, pour que le chargement des pièces jointes se fasse en premier (ce n'est pas le cas aujourd'hui)
- d'interpréter le contenu des pièces jointes dès qu'elles sont chargées, pour alimenter les autres champs de saisie
- permettre au fournisseur de compléter et corriger ces informations avant de les soumettre à Pomona

Cependant, il est illusoire d'automatiser totalement la saisie et d'imaginer se passer d'une saisie complémentaire par le fournisseur. La mise en place de la GDSN (cf. section B.2.3 page 96), qui permet pourtant de faire transiter par EDI les données produit de manière standardisée et efficace, n'a pas permis de s'affranchir de cette tâche. Sous réserve d'avoir un outil suffisamment fiable, on pourrait faire gagner du temps aux fournisseurs et améliorer la qualité de l'information produit. Néanmoins, plusieurs freins existent à la mise en place

- cela n'a d'intérêt que si le système est capable de produire de l'information structurée avec une fiabilité élevée (par exemple, 80% de données correctes serait un minimum)
- cela entre en concurrence directe avec le système GDSN et pose la question de la manière de gérer les conflits entre les informations issues de ce réseau et celles extraites des pièces jointes
- la valeur ajoutée revient essentiellement aux fournisseurs, et pas au groupe

1.3 Le contrôle des informations transmises

La détection plus tôt et de manière plus fiable des erreurs sur les données représente des avantages multiples :

- la qualité des données s'en trouve évidemment améliorée
- le processus est plus court en temps, en évitant les aller-retours
- on décharge l'ensemble des acteurs, en limitant le nombre d'interventions de chacun, ainsi que la charge administrative de synchronisation de leurs actions

Les différentes étapes possibles pour lesquelles effectuer un contrôle sont présentées à la FIGURE 1 page 7.

1.3.1 Le contrôle à la saisie fournisseur

Si on déclenche un contrôle au moment où le fournisseur soumet les données et qu'on l'alerte en cas d'incohérence, on peut dès le début du processus éviter une erreur. L'avantage d'avoir une identification des problèmes aussi tôt est qu'on économise immédiatement au moins une étape de contrôle des données.

1.3.2 L'aide aux vérifications Pomona

Lors des contrôles inclus au sein du processus, on pourrait mettre en évidence les incohérences détectées entre pièces jointes et données à contrôler. Cela permettrait de fiabiliser les étapes de contrôle, et éviter qu'une erreur soit détectée tardivement (par les gestionnaires de référentiel ou les ingénieurs qualité). C'est d'autant plus intéressant que la première étape de contrôle est faite par les acheteurs. Dans la mesure où c'est une tâche administrative qui n'est pas dans leur cœur de métier, elle est souvent effectuée rapidement et est peu efficace.

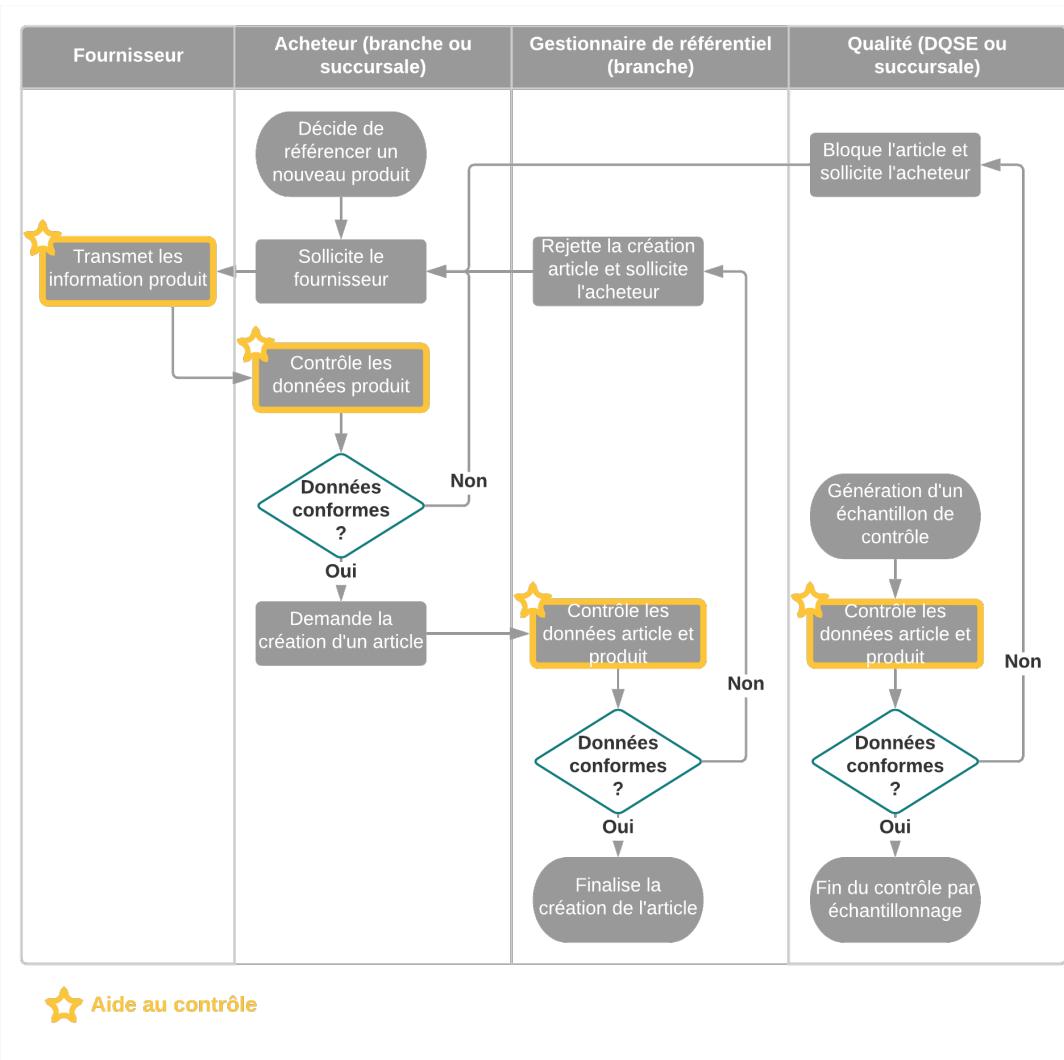


FIGURE 1 – Étapes du processus article pouvant être améliorées

1.3.3 Les contrôles en masse asynchrones

Enfin, il pourrait être pertinent de faire tourner de manière asynchrone des contrôles de qualité de données sur l'ensemble de la base. Il pourrait sembler inutile d'effectuer cette tâche alors qu'on a une validation des données par l'application (contrôles « informatiques ») et par des contrôleurs humains. Néanmoins, il arrive que des changements de règles de gestion informatiques ou métier soit mis en place, sans que les données soient ensuite remises à jour. Par exemple, les mentions des traces d'allergènes (cf. le détail sur les allergènes à l'annexe B.2.1 page 84) dans les ingrédients était par le passé retirées des listes d'ingrédients, alors qu'elle sont aujourd'hui incluses. Mais aucun chantier de remise à niveau de l'historique n'a été entrepris au moment du changement de règle, d'où des données qui ne sont pas alignées avec les règles de gestion.

Chapitre 2

PRÉSENTATION RAPIDE DES DONNÉES

RELATIVES AU CAS D'USAGE

2.1 Les listes d'ingrédients

Une information produit primordiale est la liste d'ingrédients. La construction des listes d'ingrédients doit suivre les règles suivantes, même si l'application n'est pas toujours parfaitement respectée :

- elle doit détailler l'ensemble des ingrédients, y compris les additifs et les arômes
- elle doit être triée par ordre d'importance pondérale décroissante (i.e. les ingrédients les plus représentatifs en poids doivent être cités en premier)
- la quantité de certains ingrédients (en pourcentage de la masse) par exemple ceux mis en valeur sur l'étiquetage ou dans la dénomination de vente (ex. gâteau aux fraises, pizza au jambon)

Même s'il ne s'agit pas d'une exigence réglementaire, le Groupe Pomona demande à ses fournisseurs de ne pas distinguer les ingrédients par phase comme cela se fait parfois. Cela signifie, par exemple, séparer une partie de la composition du produit (la pâte de la garniture pour une tarte, la sauce et les raviolis, ...). De telles pratiques peuvent parfois induire le consommateur en erreur, comme par exemple dans la liste d'ingrédients suivante (s'applique à des chips de légumes) :

Légumes 64% (betterave, panais, carottes, patates douces), huile de tournesol, sel marin.

Sans l'artifice d'avoir regroupé les légumes en une seule phase, le premier ingrédient de la liste aurait pu être l'huile de tournesol, qui est un ingrédient moins valorisant pour un consommateur de chips de légumes.

Quelques exemples de listes d'ingrédients sont présentées à la TABLE 1 page 9.

Désignation produit fournisseur	Liste d'ingrédients
HARICOTS BLANCS À LA TOMATE	Haricots blancs (UE et non UE), eau, sel, tomate concentré, antioxydant : acide citrique.
POIVRE VERT DÉSHYDRATÉ	100% poivre vert
Crème de marron de l'Ardèche en boîte 500 g CLEMENT FAUGIER	Châtaignes 50%, sucre, marrons glacés, sirop de glucose, eau, extrait naturel de vanille
Velouté de poireaux pommes de terre hyposodé en sachet 800 g NEFF MADA	Pomme de terre 32 %, amidon modifié de pomme de terre, féculle de pomme de terre, maltodextrine de blé, poireau 8 %, arômes naturels, sucre, oignon, antiagglomérant : E551 "nano", plantes aromatiques, curcuma
Poivre Kampot rouge en pot 50 g TERRE EXOTIQUE	Poivre de Kampot rouge 100%

TABLE 1 – Exemples de listes d'ingrédients

Les contraintes ci-dessus s'appliquant aux listes d'ingrédients font qu'en général, il s'agit d'une énumération d'ingrédients, sans doublon.

2.2 Pièces jointes

Dans le PIM, les pièces jointes - intéressantes au titre de l'information produit - gérées au niveau du produit sont les suivantes :

- les fiches techniques fournisseur
- les étiquettes produits

D'autres types de pièces jointes sont gérées, mais elle ne seront pas décrites dans le détail (car non pertinentes pour le cas d'usage présenté) :

les certificats des labels : ce sont des documents pdf produits par les organismes de certification attestant qu'un produit peut porter un label. Ils sont mis à disposition par les fournisseurs

les images produit : ce sont des visuels des produits (photographies ou images construites) qui visent à être utilisées dans des catalogues ou sur les sites de vente en ligne

les fiches logistiques : ce sont des fichiers qui viennent compléter les informations de la fiche technique lorsque cette dernière ne porte pas les informations relatives à la hiérarchie logistique.

les fiches techniques et argumentaires Pomona : ce sont des documents pdf produits par le PIM, stockées sur les articles, qui reprennent des informations produit et article (cf. annexe B.2.2 page 88 sur la distinction entre produit et article). Elles sont transmises aux clients ou utilisés par les commerciaux Pomona. Elles permettent d'avoir une présentation uniforme de l'ensemble de l'assortiment (les fiches techniques fournisseur ont des formats très variables)

2.2.1 Fiches techniques fournisseur

Généralités sur les fiches techniques

Une fiche technique fournisseur est un document, d'une à une dizaine de pages, qui reprend l'essentiel des informations techniques à propos du produit. Elles portent globalement l'ensemble des informations produit telles que présentées à l'annexe B.2.1 page 84. C'est le document le plus complet vis-à-vis des informations produit, il porte en général des informations complémentaires à toutes les informations présentes sur l'emballage du produit (les étiquettes). En général, une fiche technique ne porte d'information que pour un unique produit, mais dans le cas d'assortiments, les informations peuvent être relatives à plusieurs d'entre eux (cf. la fiche technique pour l'assortiment de confitures, présentée en annexe E.1.4 page 117). Les fiches techniques fournisseur sont des pièces jointes qui sont collectées par la branche ÉpiSaveurs depuis la mise en place du logiciel de gestion historique GIP (cf. la description de ce système en annexe B.2.3 page 93), et il s'agit d'une donnée obligatoire dans le PIM également.

Le format des fiches techniques

Dans le PIM, ces pièces jointes sont collectées et stockées sous forme de pdf (les autres formats de fichier ne sont pas autorisés). Sauf de rares exceptions, il s'agit de fichiers issus de logiciels de traitement de texte, au format A4 portrait. Ce sont des documents techniques, et sont donc en général très structurés, avec des paragraphes, sous-paragraphes, tableaux, ... Comme cela se constate aisément sur les exemples présentés en annexe, même si les informations portées sont sensiblement toujours les mêmes, les formats de ces documents sont extrêmement variables. Il n'y a aucun standard réglementaire ou normatif relatif à la construction de ces fiches, chaque fournisseur constitue donc son propre format. On a donc un foisonnement de formats différents.

Focus sur le mode de présentation de quelques informations

Les généralités relatives aux informations produit sont présentées en annexe B.2.1 page 84. Dans les fiches techniques, ces informations sont en général présentées de la manière suivante :

Liste d'ingrédients : elle est quasiment toujours présentée sous forme de texte (identique à la liste d'ingrédients telle qu'affichée sur l'emballage du produit), mais elle peut également être présentée sous forme de tableau. Cf. la fiche technique du poivron présentée en annexe E.1.6 page 121 (celle-ci ne porte pas de liste d'ingrédients sous forme de texte)

Allergènes : il peuvent être (cf. le détail sur les allergènes en annexe B.2.1 page 84) :

- soit mis en évidence dans la liste d'ingrédients via la police (gras, souligné, majuscules)
- soit listés hors de la liste d'ingrédient, sous forme de texte. Cf. la fiche technique de la préparation pour panna cotta présentée en annexe E.1.3 page 115
- soit listés sous forme d'un tableau reprenant l'ensemble des allergènes réglementaires, et le niveau de présence associé (présence, contamination croisée ou absence). Cf. la fiche technique de la ciboulette

présentée en annexe E.1.5 page 119

Données nutritionnelles : les données nutritionnelles sont en général présentées sous forme de tableau dans les fiches techniques. Les informations nutritionnelles sont données pour 100 grammes (ou 100 millilitres), ce qui est réglementaire, mais également parfois pour une portion. La taille de la portion est définie arbitrairement par le fournisseur. Cf. la fiche technique de la préparation pour panna cotta E.1.3 page 115.

Données logistiques : les informations relatives à la hiérarchie logistiques se présentent souvent sous forme de tableaux. L'interprétation de ces tableaux est généralement complexe et nécessite un peu de réflexion.

2.2.2 Étiquettes produit

Généralités sur les étiquettes

L'étiquette produit est la partie de l'emballage du produit qui porte les informations produit. Selon la technologie de l'emballage - très liée au fait que le produit est plutôt brut ou plutôt transformé - il peut s'agir :

- d'une photo de l'étiquette collante apposée sur l'extérieur de l'emballage du produit (cf. exemple des étiquettes de lentilles E.2.3 page 127 ou de sauce soja E.2.5 page 129)
- d'un applat de l'emballage, ou son bon-à-tirer, qui est le document qui est ensuite envoyé aux chaînes de production pour impression (cf. exemple du bon à tirer pour la préparation pour panna cotta E.2.4 page 128)
- ou de tout autre visuel montrant une partie indissociable de l'emballage physique du produit (ex : un photo de la face de l'emballage qui porte les informations produit, cf. l'étiquette des madeleines E.2.2 page 126)

Pour résumer, l'étiquette est une pièce jointe qui représente l'information produit qui « voyage avec le produit ». La cohérence entre les données de l'étiquette et l'information produit transmises aux clients est un enjeu majeur, dans la mesure où ce sont les informations portées par le produit physique qui sont réputées correctes. La collecte systématique des étiquettes produit est une nouveauté arrivée avec la mise en place du PIM pour ÉpiSaveurs (mai 2019). En théorie, les étiquettes mises à disposition par les fournisseurs devraient systématiquement porter les informations réglementaires. Or, du fait de la relative nouveauté de cette collecte, les pièces jointes transmises ne sont pas toujours conformes (cf. l'étiquette des lentilles en annexe E.2.3 page 127, qui ne porte aucune information nutritionnelle ou de composition).

Le format des étiquettes

Comme les fiches techniques, ces pièces jointes sont collectées et stockées sous forme de pdf (les autres formats de fichier ne sont pas autorisés). Du fait que les natures mêmes de ces pièces jointes sont diverses

(cf. paragraphe précédent), il n'existe pas de format prédominant.

Focus sur le mode de présentation de quelques informations

Les étiquettes portent moins d'information que les fiches techniques. Par exemple, elles ne portent pas d'information sur la hiérarchie logistique, les données administratives (tel que le taux de TVA ou la nomenclature douanière), les codes d'identification (hormis le GTIN qui est présent sur le code à barre), ... Les durées de vie ne sont pas mentionnées : sur l'emballage d'un produit seule la date limite apparaît, et elle dépend du lot de production. En règle générale, hormis quelques allégations volontairement affichées par l'industriel, l'étiquette ne porte que les informations réglementaires. Les informations positionnées sur l'étiquette se présentent de la manière suivante :

Liste d'ingrédients : elle est toujours présentée sous forme d'un texte énumérant les ingrédients (cf. la section 2.1 page 8 qui détaille le contenu d'une liste d'ingrédients)

Allergènes : ils sont uniquement mis en évidence dans la liste d'ingrédients, par l'utilisation d'une police spécifique (gras, souligné, majuscule, ...). Cela peut se constater par exemple sur l'étiquette de madeleines en annexe E.2.2 page 126 ou celle de la sauce soja en annexe E.2.5 page 129

Données nutritionnelles : les données nutritionnelles se présentent souvent sous forme d'un tableau, toujours avec les valeurs pour 100 grammes ou 100 millilitres, et parfois pour une portion. Il peut arriver, plutôt pour les produits peu transformés, que ces données soient simplement écrites sous forme de texte tel que

Énergie : 1101 kJ/260 kcal, Glucides : 57 g, Sucres : 52 g, Protéines : 4,3 g, Sel : 7,1 g

2.3 Récapitulatif de la complétude des données

Dans cette section, les indicateurs présentés se basent sur les notions « En qualité » et « Hors qualité », qui sont explicitées en annexe D.1.2 page 105.

2.3.1 Complétude des listes d'ingrédients

L'analyse de la complétude des listes d'ingrédients dans le PIM est décrite à la TABLE 2 page 12.

	En qualité			Hors qualité			Total		
	cpt	sur	%	cpt	sur	%	cpt	sur	%
Epicerie	3051	3051	100%	5317	5742	92%	8368	8793	95%
Boissons	356	356	100%	504	551	91%	860	907	94%
Alcools	254	254	100%	259	353	73%	513	607	84%
Hygiène	0	783	0%	0	1743	0%	0	2526	0%
Chimie	0	138	0%	3	329	0%	3	467	0%
Total	3661	4582	79%	6083	8718	69%	9744	13300	73%

TABLE 2 – Analyse volumétrique des pièces jointes

On peut déduire de cette analyse :

- que les produits d'hygiène et de chimie, n'étant pas des produits alimentaires, ne portent pas de listes d'ingrédients
- que les listes d'ingrédients sont systématiquement renseignées sur les produits « En qualité »
- que le taux de complétude reste quand même globalement bon sur les produits « Hors qualité »

2.3.2 Complétude des pièces jointes

Si on analyse le niveau de renseignement des données pour les étiquettes et les fiches techniques, on obtient la TABLE 3 page 13.

	En qualité						Hors qualité						Total					
	Fiche technique		Etiquette				Fiche technique		Etiquette				Fiche technique		Etiquette			
	cpt	sur	%	cpt	sur	%	cpt	sur	%	cpt	sur	%	cpt	sur	%	cpt	sur	%
Epicerie	3007	3007	100%	2996	3007	99%	4791	5755	83%	1921	5755	33%	7798	8762	88%	4917	8762	56%
Boissons	355	355	100%	352	355	99%	458	551	83%	184	551	33%	813	906	89%	536	906	59%
Alcools	253	253	100%	253	253	100%	296	354	83%	74	354	20%	549	607	90%	327	607	53%
Hygiène	768	768	100%	688	768	89%	1509	1733	87%	420	1733	24%	2277	2501	91%	1108	2501	44%
Chimie	130	130	100%	130	130	100%	310	329	94%	127	329	38%	440	459	95%	257	459	55%
Total	4513	4513	100%	4419	4513	97%	7364	8722	84%	2726	8722	31%	11877	13235	89%	7145	13235	53%

TABLE 3 – Analyse volumétrique des pièces jointes

On en tire comme conclusions que :

- le taux de collecte de ces documents est quasiment parfait pour les produits dits « En qualité » (cf. les définitions données à la section D.1.2 page 105)
- la volumétrie d'étiquettes sur les produits « Hors qualité » est bien plus faible, de l'ordre de 30%

2.4 Les données « manuellement étiquetées »

2.4.1 Pour répondre à quel besoin ?

La qualité des données actuellement présentes dans le système fait que la cohérence entre les listes d'ingrédients du PIM et celles présentes dans les fiches techniques n'est pas assurée. Comme dans tout modèle de machine learning se basant sur des données, il est indispensable ici d'avoir des données dont on est sûrs de la qualité. Il a donc été décidé d'étiqueter manuellement un échantillon représentatif de fiches techniques, en leur faisant correspondre leurs listes d'ingrédients.

2.4.2 Mode de constitution de l'échantillon

Le code pour la constitution de l'échantillon est présenté dans le notebook en annexe G.1 page 141.

Les règles de gestion adoptées pour la constitution de cet échantillon sont les suivantes :

- On constitue un échantillon de 500 fiches techniques étiquetées
- On se limite aux produits d'Épicerie et de Boisson non alcoolisée, car ce sont les produits pour lesquels la réglementation impose d'afficher une liste d'ingrédients aux consommateurs

- On stratifie cet échantillon par type de produit (Épicerie vs. Boisson non alcoolisée)
- On se limite à des produits qui possèdent une fiche technique

Il aurait pu être intéressant de se limiter aux produits « En qualité » (cf. les définitions données en annexe D.1.2 page 106 sur les statuts des produits), mais l'étiquetage manuel avait été fait avant l'analyse de ces statuts. La constitution de cet échantillon s'est traduite par la génération d'un fichier csv de 500 lignes, avec 3 colonnes :

- l'uid du produit de l'échantillon
- la désignation produit fournisseur
- une colonne vide, visant à accueillir les listes d'ingrédients lors de l'étiquetage manuel

2.4.3 Méthodologie de l'étiquetage manuel

Cette activité s'est faite simplement, dans un environnement Windows :

- les pièces jointes ont été téléchargées localement, dans des dossiers portant l'uid du produit associé
- le csv contenant les uid et les désignation produit fournisseur a été ouvert dans le tableur Microsoft Excel
- en prenant chaque uid séquentiellement, il était relativement rapide de rechercher localement la fiche technique correspondante dans l'explorateur de fichier et de l'ouvrir
- le plus souvent, il était possible de copier/coller la liste d'ingrédient dans le tableur Excel
- le fichier a ensuite été à nouveau sauvegardé au format csv, afin de pouvoir être chargé dans pandas

Une passe de nettoyage des caractères spéciaux issus des copier/coller a ensuite été faite dans excel.

2.4.4 Règles de gestion pour l'étiquetage manuel

Malgré le fait que l'exercice semble à priori peu complexe, en pratique un nombre assez élevé de cas particuliers ont nécessité de prendre des décisions, parfois arbitraires. En général, il s'agit de décisions à prendre lorsque des mentions qui ne sont pas des ingrédients au sens strict sont incluses dans le texte des ingrédients. Le principe de base qui a été retenu est d'essayer de coller au plus à ce qui est attendu dans le PIM. Par exemple, dans le PIM on demande de retirer les préfixes du type « Ingrédients : » car ces mentions sont reprises telles sur les sites internet, ce qui peut se traduire par l'affichage de « Ingrédients : Ingrédients : ... ». L'ensemble des règles sont documentées à l'annexe E.3.1 page 131.

2.4.5 Confrontation avec le contenu du PIM

Il est possible de comparer les liste d'ingrédients du PIM, et ceux des données étiquetées manuellement. Si on prend uniquement en compte les égalités strictes, on a un niveau de cohérence exactement à 10% (50 produits sur les 500 du périmètre). Des exemples de liste d'ingrédients en écart sont présentées à la TABLE 4 page 16. Il apparaît clairement que l'essentiel des écarts sur cet échantillon sont dus à des ajustements de

forme (mise en majuscule des allergènes, retrait de parenthèses, retours à la ligne, ...).

Chapitre 3

LE CHOIX DU CAS D'USAGE

3.1 Étiquettes contre fiches techniques

3.1.1 La représentation dominante des fiches techniques

Les pièces jointes utilisées lors des contrôles métier pour la vérification des données produit sont les fiches techniques fournisseur et les étiquettes. Or, les étiquettes sont moins représentées que les fiches techniques dans le PIM (cf. TABLE 3 page 13).

3.1.2 L'extraction de données textuelles

Comme cela a été décrit à la section 2.2.2 page 11 Les étiquettes sont des pièces jointes au format pdf, mais qui peuvent être générées de multiples façons :

- issues d'un logiciel graphique pour la constitution du applat du packaging
- construites à partir d'une ou plusieurs photos insérées dans outil de traitement de texte
- ...

Les textes présents dans ces pdf se présentent donc parfois sous forme de textes extractibles par des bibliothèques de pdf mining, soit sous forme d'images qui nécessitent alors des fonctionnalités d'OCR (Optical Character Recognition [20]). Le taux de pièces jointes avec des textes extractibles est présenté à la TABLE 5 page 17; pour mémoire seul un tiers des étiquettes ont un contenu exploitable par les outils de pdfmining. De plus, l'orientation des textes peut parfois faire l'objet d'une rotation (cf. l'étiquette des lentilles en annexe E.2.3 page 127), voire n'être pas toujours la même selon les faces de l'emballage (cf. l'étiquette de la panna cotta, en annexe E.2.4 page 128). Ces contraintes, bien que surmontables, imposent des travaux qui n'auraient pas d'intérêt dans le cadre de ce mémoire.

On va donc dans un premier temps travailler avec les fiches techniques plutôt qu'avec les étiquettes

Ingrédients du PIM	Ingrédients de la ground truth
Céréales (farine de FROMENT (27,5%), céréales complètes (15,1%) (farine complète de FROMENT (15%), farine complète de SEIGLE, farine complète d'ORGE) sucre graisses végétales (palme, palmiste, colza) poudre de cacao maigre (5,1%) amidon de BLE LAIT écrémé en poudre (équivalent lait 16%) amidon de pomme de terre poudres à lever (carbonates de sodium et d'ammonium, diphosphates) LACTOSE et protéines de LAIT sel arôme émulsifiant : lécithine de tournesol agent de traitement de la farine : E223 (SULFITES).	Céréales [farine de froment (27.5%), céréales complètes (15.1%) (farine complète de froment (15 %), farine complète de seigle , farine complète d'orge)] sucre graisses végétales (palme, palmiste, colza) poudre de cacao maigre (5.1%) amidon de blé lait écrémé en poudre (équivalent lait 16%) amidon de pomme de terre poudres à lever (carbonates de sodium et d'ammonium, diphosphates) lactose et protéines de lait sel arôme émulsifiant : lécithine de tournesol agent de traitement de la farine : E223 (sulfites) Peut contenir des traces de fruits à coque, de soja, d'oeuf et de sésame
cheddar fondu(44%) (eau, LACTOSERUM, cheddar(5%) (LAIT, ferment, acidifiant (sulfates de sodium (E514)), enzymes), LAIT écrémé, maltodextrine, antioxydant (phosphate de sodium (E339)), CREME, épaississants (octényle succinate d'amidon sodique (E1450), gomme xanthane (E415)), acidifiant (sulfates de sodium (E514))), eau, huile de palme, épaississants (amidon modifié de tapioca, amidon modifié de maïs), maltodextrine, acidifiants (sulfates de sodium (E514), citrate de sodium (E331), acide phosphorique (E338), acide lactique (E270)), arôme, antioxydant (phosphate de sodium (E339)), colorants (béta carotène (E160a), annatto (E160b), extrait de paprika (E160c)), émulsifiant (stéaroyl-2-lactylate de sodium (E481), mono et diglycérides d'acides gras alimentaires (E471))	cheddar fondu (44%) (eau, lactosérum, cheddar (5%) (lait, ferment, acidifiant (sulfates de sodium (E514)), enzymes), lait écrémé, maltodextrine, antioxydant (phosphate de sodium (E339)), crème, épaississants (octényle succinate d'amidon sodique (E1450), gomme xanthane (E415)), acidifiant (sulfates de sodium (E514))), eau, huile de palme, épaississants (amidon modifié de tapioca, amidon modifié de maïs), maltodextrine, acidifiants (sulfates de sodium (E514), citrate de sodium (E331), acide phosphorique (E338), acide lactique (E270)), arôme, antioxydant (phosphate de sodium (E339)), colorants (béta carotène (E160a), annatto (E160b), extrait de paprika (E160c)), émulsifiants (stéaroyl-2-lactylate de sodium (E481), mono et diglycérides d'acides gras alimentaires (E471))
Pommes* (80%), framboises* (20%)	pommes* (80%), framboises* (20%) *Ingrédient issu de l'agriculture biologique
Eau, graines de MOUTARDE, vinaigre d'alcool, sel, vin blanc (contient SULFITES), sucre, épices, acidifiant (acide citrique), conservateur (DISULFITE de potassium)	Eau, graines de moutarde, vinaigre d'alcool, sel, vin blanc (contient sulfites), sucre, épices, acidifiant (acide citrique), conservateur (disulfite de potassium)
Pâte à tartiner aux NOISETTES et au cacao 81,5 % (sucre, huile de palme, NOISETTES 13%, LAIT écrémé en poudre 8,7%, cacao maigre 7,4%, émulsifiants : lécithines [SOJA] , vanilline), farine de FROMENT 16%, levure de bière, extrait de malt d'ORGE, sel, LAIT écrémé en poudre, émulsifiants : lécithines [SOJA] , protéines de FROMENT, protéines de LAIT, eau.	pâte à tartiner aux noisettes et au cacao 81,5 % (sucre, huile de palme, noisettes 13%, lait écrémé en poudre 8,7%, cacao maigre 7,4%, émulsifiants : lécithines [soja]; vanilline), farine de froment 16%, levure de bière, extrait de malt d'orge, sel, lait écrémé en poudre, émulsifiants : lécithines [soja]; protéines de froment, protéines de lait, eau. Le chocolat utilisé est un chocolat pur beurre de cacao.
Sirop de glucose, sucre, amidons transformés, acidifiants : acide citrique, acide malique, correcteurs d'acidité : citrate tricalcique, malate acide de sodium, agent d'enrobage : cire de carnauba, arôme, concentrés de fruits et de plantes : carthame, citron, colorants : carmins, bleu patenté V, lutéine, sirop de sucre inverti.	sirop de glucose ; sucre ; amidons transformés ; acidifiants : acide citrique, acide malique ; correcteurs d'acidité : citrate tricalcique, malate acide de sodium ; agent d'enrobage : cire de carnauba ; arôme ; concentrés de fruits et de plantes : carthame, citron ; colorants : carmins, bleu patenté V, lutéine ; sirop de sucre inverti.
Huile de colza, jaunes d'OEUFS, purée de tomates, eau, vinaigre, câpres (2.8 %) (câpres, eau, sel), cornichons (2.8 %) (cornichons, vinaigre, sel), sucre, sel, amidon modifié, acidifiant : glucono-delta-lactone, conservateur : sorbate de potassium, antioxydant : E385, épice, arôme naturel aneth.	Huile de colza (70.45 %), jaunes d'oeufs (6.2 %), purée de tomates, eau, vinaigre, câpres (2.8 %) (câpres, eau, sel), cornichons (2.8 %) (cornichons, vinaigre, sel), sucre, sel, amidon modifié, acidifiant : glucono-delta-lactone, conservateur : sorbate de potassium, antioxydant : E385 (74 mg/kg), épice, arôme naturel aneth.
Lentilles, eau, sel, arôme naturel (CELERI)	Lentilles, eau, sel, arôme naturel (céleri)
Ingrédients : Sucre, pâte de cacao, NOISETTES (23%) , beurre de cacao, BEURRE concentré, émulsifiant (lécithine de SOJA), arôme, LAIT écrémé en poudre. Cacao : 46% minimum dans le chocolat noir. PEUT CONTENIR AUTRES FRUITS À COQUE.\t\t\t	Sucre, pâte de cacao, NOISETTES (23%) , beurre de cacao, BEURRE concentré, émulsifiant (lécithine de SOJA), arôme, LAIT écrémé en poudre. Cacao : 46% minimum dans le chocolat noir. PEUT CONTENIR AUTRES FRUITS À COQUE.
Huile de colza, eau, vinaigre d'alcool, jaunes d'OEUFS, sucre, estragon (2,5%), échalote (2,4%), sel, cerfeuil, jus de citron à base de concentré, piment de Cayenne, arômes, amidon modifié, colorant : bêta-carotène, conservateur : sorbate de potassium.	Huile de colza, eau, vinaigre d'alcool, jaunes d'oeufs, sucre, estragon (2,5%), échalote (2,4%), sel, cerfeuil, jus de citron à base de concentré, piment de Cayenne, arômes, amidon modifié, colorant : bêta-carotène, conservateur : sorbate de potassium.Ce produit peut contenir des traces de moutarde.

TABLE 4 – Exemples d'écart entre les données étiquetées et celles du PIM

Documents vides			
	Nombre de documents vides	Nombre total de documents	Taux de vides
Etiquettes	214	315	67%
Fiches techniques	22	500	4%

TABLE 5 – Pièces jointes dont les textes ne sont pas extractibles

3.2 Impossibilité de produire des templates pour l'exploitation de ces documents

Chaque fournisseur décide du format des fiches techniques qu'il souhaite produire. On a donc beaucoup de formats différents, à peu de choses près on a un format par fournisseur. On pourrait imaginer développer un outil qui sur la base d'un template permet d'extraire de manière fiable le contenu de documents qui ont toujours le même format (cela se fait par exemple pour les commandes clients reçues par mail). L'établissement d'un « traducteur » de documents de ce type représente une charge d'une dizaine de jours de travail par format de fichier. Le Pareto des produits par fournisseur présenté à la FIGURE 2 page 17 montre qu'il ne serait pas économiquement pertinent de construire ce genre d'outil (10 templates ne permettraient de couvrir que 3000 produits, et le gain marginal ne cesserait de diminuer ensuite). On n'approche pas du seuil de rentabilité requis. En comparaison, les outils de parsing des commandes mail permettent de traiter plus de 20 000 postes de commandes par jour, avec moins de 5 templates.

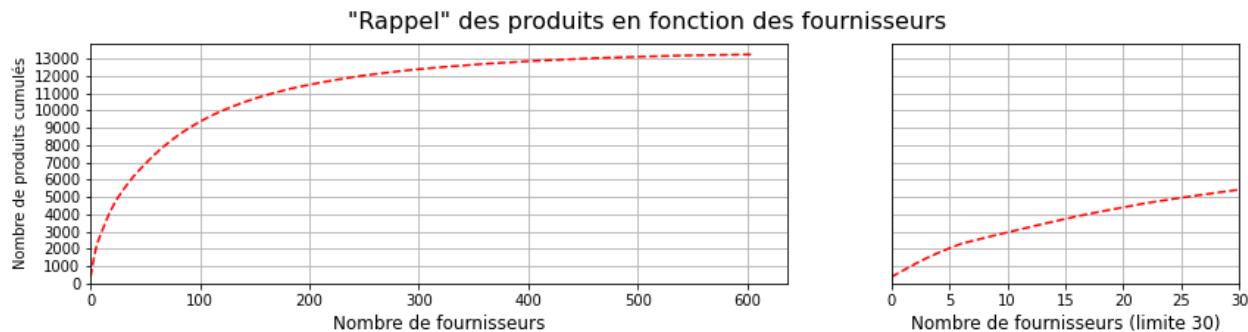


FIGURE 2 – Le Pareto des produits en fonction des fournisseur

Une approche traditionnelle d'extraction de données depuis des documents pdf ne pourra pas être appliquée pour ce cas d'usage

3.3 Quelles données ?

3.3.1 Les principales données et leur mode de présentation dans les fiches techniques

Les données que l'on souhaite récupérer sont globalement de 4 types :

- données de composition : listes d'ingrédients
- données de composition : allergènes
- données nutritionnelles
- données logistiques

Comme vu dans la section 2.2.1 page 10, parmi les données présentes dans les fiches techniques, seule la liste d'ingrédients se présente quasiment toujours sous la forme d'un texte brut (par opposition à un affichage au sein d'un tableau). La représentation des données nutritionnelles se fait régulièrement sous forme de tableau, les données logistiques également (cf. les exemples de fiches techniques présentées en annexe, à partir de la section E.1.1 page 113).

3.3.2 L'identification d'une liste d'ingrédient par son contenu

Il est possible de dire si un texte est une liste d'ingrédients, juste en lisant ce texte. Il s'agit souvent d'une énumération d'ingrédients individuels (cf. la section 2.1 page 8). Même sortie de son contexte, il est en général faisable pour un humain d'identifier comme telle une liste d'ingrédients. Par exemple :

Huile de colza (70.45 %), jaunes d'oeufs (6.2 %), purée de tomates, eau, vinaigre, câpres (2.8 %) (câpres, eau, sel), cornichons (2.8 %) (cornichons, vinaigre, sel), sucre, sel, amidon modifié, acidifiant : lucono-delta-lactone, conservateur : sorbate de potassium, antioxydant : E385 (74mg/kg), épice, arôme naturel aneth

est de toute évidence une liste d'ingrédients.

À l'inverse, les autres grands types d'information nécessitent du contexte dans le document. Par exemple, « 14g » peut être une quantité de glucides, de lipides, le poids d'une pièce unitaire, ...

L'identification de listes d'ingrédients pourrait se faire sans avoir à construire de fonctionnalités s'appuyant sur le contexte au sein du document

3.4 Quels produits ?

Pour des raisons d'accessibilité de la donnée, on travaillera sur les produits de la branche ÉpiSaveurs. Il est de toute façon prévu de déployer le PIM sur l'ensemble des autres branches du groupe, on pourra adresser leurs produits à ce moment-là. La notion d'ingrédients n'a pas de sens pour les produits d'hygiène (cf. section C.1 page 98 sur les produits non-alimentaires), mais existe pour les produits de chimie. Comme les

ingrédients des produits alimentaires sont en général différents des produits de chimie, on se limitera d'abord aux premiers. Enfin, la réglementation étant moins stricte pour les boissons alcoolisées sur le sujet spécifique de l'étiquetage de la composition, on ne les prendra pas non plus en compte dans ce mémoire.

3.5 Conclusion quant au choix du cas d'usage

Au vu des différentes contraintes listées dans cette partie, on s'attachera à extraire les listes d'ingrédients des produits alimentaires de la branche EpiSaveurs (épicerie et boissons non alcoolisées) depuis les fiches techniques fournisseur, en se basant sur le contenu textuel de ces documents.

Deuxième partie

ANALYSE DES DONNÉES TEXTUELLES

Chapitre 4 CARACTÉRISTIQUES GÉNÉRALES DU JEU DE DONNÉES

Dans toute cette analyse, on se basera sur les données manuellement étiquetées (ou ground truth), échantillon de 500 fiches techniques avec les listes d'ingrédients associées. Le code ayant servi à effectuer ces analyses est présenté dans le notebook en annexe G.2 page 147.

4.1 Distributions des longueurs des textes

L'analyse des longueurs des textes montre que les listes d'ingrédients sont en moyenne longues de 400 caractères, et le contenu des documents de 4000 caractères (cf. TABLE 6 page 20). La distribution des longueurs de listes d'ingrédients et des longueurs de textes est présentée à la FIGURE 3 page 21, et leur représentation de l'une en fonction de l'autre à la FIGURE 4 page 21. Il apparaît qu'il n'y a pas de corrélation entre la longueur de la liste d'ingrédients et la longueur du contenu du document ($r^2 = 0.139$).

	text count	mean	std	min	25%	50%	75%	max
Texte des documents	500.0	3937.630	3280.860732	0.0	2173.75	3247.5	5034.25	37322.0
Listes d'ingrédients	500.0	199.686	457.044723	0.0	43.00	122.0	250.75	7963.0

TABLE 6 – Longueur des textes dans le dataset

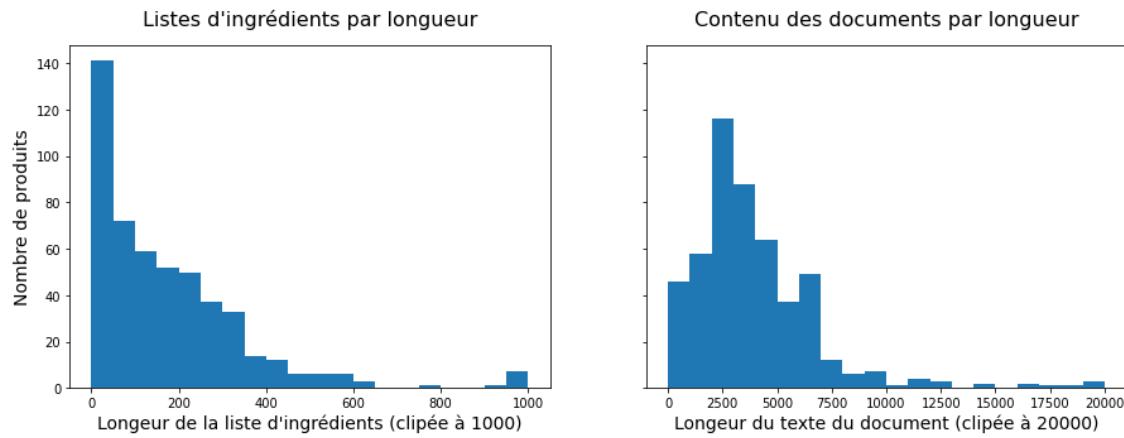


FIGURE 3 – Distribution des longueurs de textes

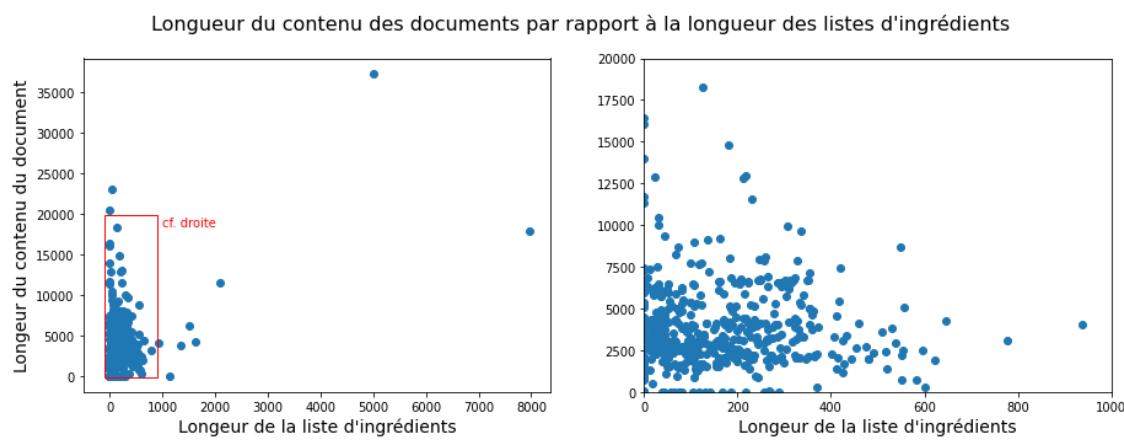


FIGURE 4 – Corrélation entre longueurs des textes

Chapitre 5

ANALYSE TEXTUELLE

5.1 Mots de chacun des corpus

Pour l'analyse des mots de chacun des corpus (listes d'ingrédients et contenu des documents), on fait d'abord un preprocessing de ces textes bruts :

- mise en minuscule de ces textes
- retrait des accents
- retrait des stopwords suivants : 'pas', 'le', 'en', 'pour', 'ou', 'ce', 'de', 'dans', 'du', 'and', 'un', 'sur', 'et', 'of', 'est', 'par', 'la', 'les', 'dont', 'au', 'des', 'que'
- split des textes à chaque whitespace (espace, tabulation, retour à la ligne, ...)

La liste de ces stopwords a été manuellement établie à partir des mots les plus fréquents de chacun des vocabulaires. Elle est commune aux deux corpus. Un descriptif des vocabulaires est présenté à la TABLE 7 page 23. Un rapide parcours de ces exemples montre que l'utilisation des mots dans ces deux corpus (ingrédients et contenu des documents) est différente. On s'attend à pouvoir différencier automatiquement ces types de textes. Le vocabulaire des ingrédients est, comme on pouvait s'y attendre, entièrement inclus dans le vocabulaire du contenu des fiches techniques.

5.2 Répartitions des mots dans et hors des listes d'ingrédients

On calcule la « document frequency » des mots du corpus des ingrédients dans les listes d'ingrédients. Elle s'étend simplement au corpus du contenu des documents en la mettant à 0 pour les mots qui n'appartiennent pas au vocabulaire des ingrédients. Cela permet de calculer un score absolu s_i pour chaque mot i via la formule suivante :

$$s_i = \log_2(1 + c_i)$$

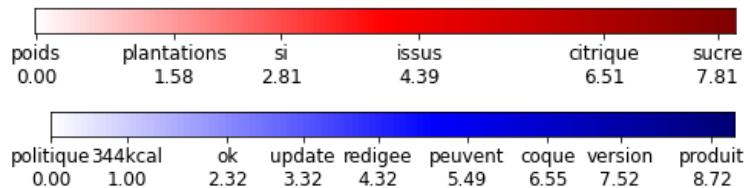
où c_i est le nombre de listes d'ingrédients dans lequel le mot i apparaît (proportionnel à la document frequency).

Le score absolu vaut donc 0 si le mot n'est pas présent dans le vocabulaire des ingrédients, 1 s'il est présent une fois, et progresse de manière logarithmique en fonction du taux de présence du mots dans le corpus des ingrédients (c'est une « smooth document frequency »). On peut, de manière similaire, calculer un score pour l'ensemble des mots, relatif à la présence dans le contenu des documents. Attention, dans la mesure où la composition est normalement portée par la fiche technique, en théorie les mots de la liste d'ingrédients sont

	Ingrédients	Contenu des documents
Longueur du vocabulaire	1324	15465
Top 20 mots	sucré : 363 occurrences	produit : 2198 occurrences
	acide : 255 occurrences	non : 2164 occurrences
	sel : 240 occurrences	produits : 1508 occurrences
	sirop : 180 occurrences	10 : 1404 occurrences
	eau : 178 occurrences	kg : 1290 occurrences
	poudre : 176 occurrences	base : 1119 occurrences
	arome : 175 occurrences	sel : 1041 occurrences
	lait : 171 occurrences	poids : 1021 occurrences
	ble : 171 occurrences	100 : 971 occurrences
	huile : 146 occurrences	palette : 933 occurrences
	critique : 132 occurrences	ingredients : 917 occurrences
	farine : 128 occurrences	date : 904 occurrences
	amidon : 125 occurrences	sucré : 841 occurrences
	glucose : 124 occurrences	12 : 803 occurrences
	cacao : 122 occurrences	code : 801 occurrences
	extrait : 117 occurrences	lait : 733 occurrences
	acidifiant : 114 occurrences	absence : 697 occurrences
	aromes : 98 occurrences	fiche : 688 occurrences
	soja : 96 occurrences	the : 658 occurrences
	concentre : 92 occurrences	france : 657 occurrences

TABLE 7 – Caractéristiques des vocabulaires

contenus dans le texte du document. Pour prendre en compte ce point, avant de calculer le score par rapport au contenu des documents, on soustrait les comptes des mots de la liste d'ingrédients des comptes de mots du contenu du document. Ne sont considérés comme apparaissant dans le contenu du document que les mots dont la différence est strictement positive. Une illustration de ces scores est donnée à la FIGURE ?? page 23.



En rouge sont présentés les scores des mots pour les listes d'ingrédients. En bleu les scores pour le reste des documents.

FIGURE 5 – Exemples de scores absolus de mots

La TABLE 7 page 23 montre que certains mots sont répandus à la fois au sein des listes d'ingrédients, mais également dans le corps des documents (ex : « sel », « sucre », « lait », ...). L'identification des listes d'ingrédients devrait être plus efficace si on prend en compte la fréquence relative des mots dans chacun des corpus :

- la présence du mot « sucre », bien qu'il soit très souvent présent dans les listes d'ingrédients, n'est pas forcément un bon indicateur que le bloc de texte considéré est une liste d'ingrédients

- le mot « poids » devrait pénaliser fortement un bloc de texte car il n'est pas présent dans les listes d'ingrédients, mais fréquemment à l'extérieur

On peut calculer un score relatif s'_i pour chaque mot i via la formule suivante :

$$s'_i = \log_2\left(\frac{1 + c_i^{ingred}}{1 + c_i^{docs}}\right) = \log_2(1 + c_i^{ingred}) - \log_2(1 + c_i^{docs}) = s_i^{ingred} - s_i^{docs}$$

où c_i^{corpus} est le nombre de documents du corpus *corpus* dans lequel le mot i apparaît, et s_i^{corpus} est le score absolu du mot i dans le corpus *corpus* tel que défini précédemment. Cette manière de calculer le score :

- fait qu'un mot autant présent (au sens de la document frequency) dans les deux corpus a un score de 0
- un mot 2 fois plus présent dans le corpus des ingrédients que dans le corpus des documents aura un score de 1
- un mot plus présent dans le corpus des documents que dans le corpus des ingrédients aura un score négatif
- un mot plus présent dans le corpus des ingrédients que dans le corpus des documents aura un score positif
- peut fonctionner parce que la taille des corpus est équilibrée (il faudrait passer en document frequency en cas de déséquilibre)

Une illustration de ces scores est présentée à la FIGURE 6 page 24.

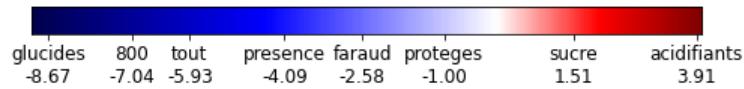
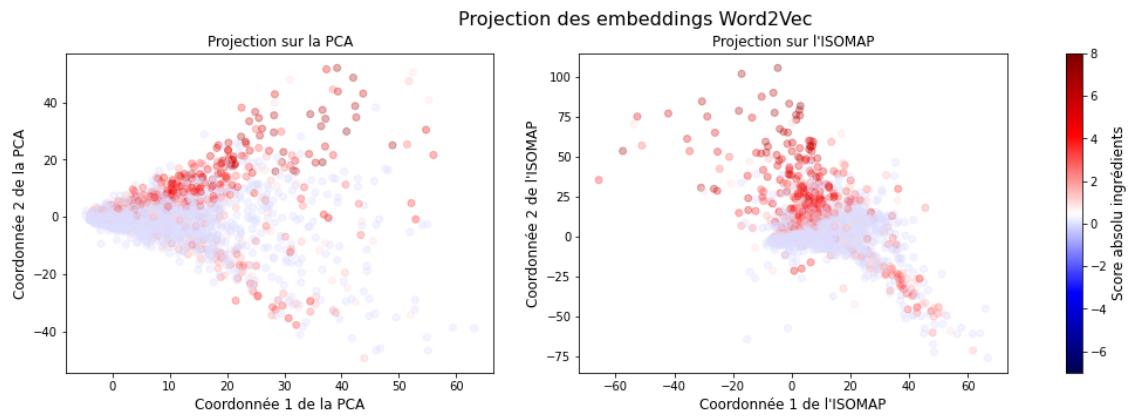


FIGURE 6 – Exemples de scores relatifs de mots

5.3 Représentations graphiques des textes et mots

5.3.1 Vectorisation des mots

On calcule les embeddings des mots via l'algorithme Word2Vec, sur le corpus du contenu des documents. Si l'on applique deux méthodes de réduction de la dimensionnalité sur ces embeddings (PCA et ISOMAP), on peut en faire une représentation graphique. Le résultat de cette projection est présenté à la FIGURE 7 page 25. On voit clairement que les embeddings des mots issus du vocabulaire des ingrédients sont localisés dans les mêmes régions. Le calcul des embeddings via l'algorithme Word2Vec semble donc avoir un intérêt pour l'identification des listes d'ingrédients.



Remarque : ci-dessus, la colormap a très légèrement été décalée pour que les mots ayant un score de 0 soient visibles (gris/bleu clair) - scores absolus

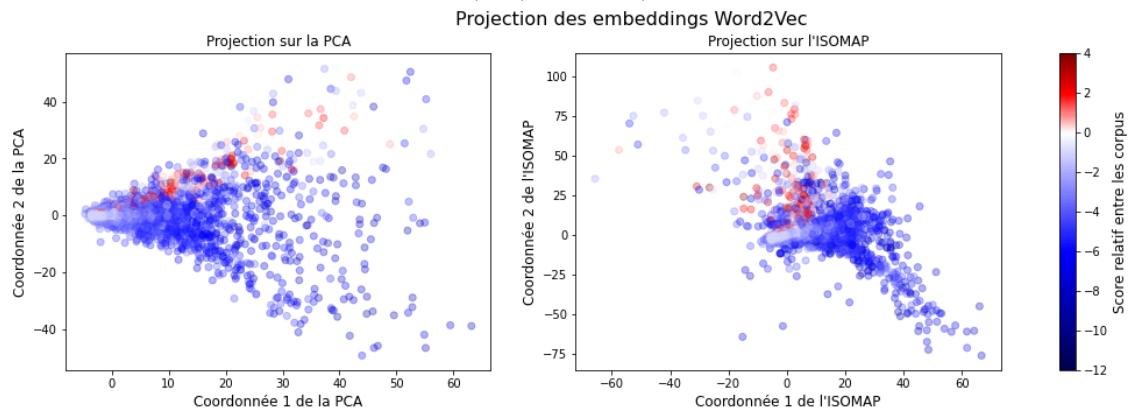


FIGURE 7 – Projection des embeddings Word2Vec

5.3.2 Vectorisation des textes

La vectorisation des textes va se faire sur la base de la matrice des comptes ou des fréquences de mots. On commence par effectuer une PCA (après standardisation) sur les comptes de mots de chacun des textes : listes d'ingrédients ou contenu des documents. Les résultats sont présentés à la FIGURE 8 page 26. Les corpus

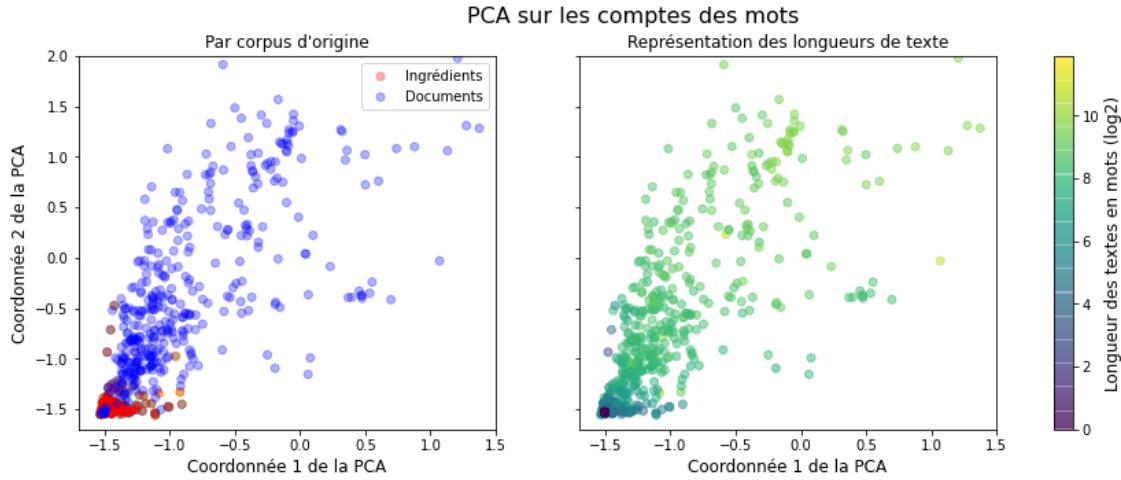


FIGURE 8 – Projection des comptes : PCA

sont bien séparés. Une explication pour cette séparation assez franche est la suivante : les vecteurs du corpus des contenus de documents ont en général une norme plus élevée que celle des ingrédients. La dispersion est donc plus grande pour les contenus des documents que pour les ingrédients.

On peut appliquer une transformation similaire, mais cette fois en utilisant une Truncated SVD (cf. FIGURE 9 page 26). Là encore, les deux corpus sont différemment localisés. La même explication que pour

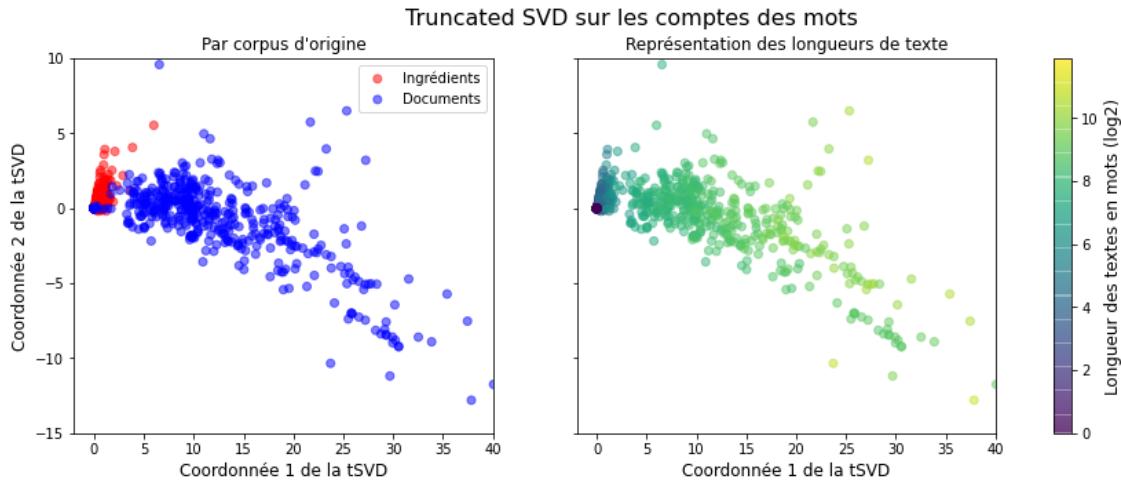


FIGURE 9 – Projection des comptes : Truncated SVD

les résultats de la PCA pourrait tenir pour expliquer cette répartition différenciée.

On peut changer d'espace de départ, en se basant sur les fréquences (« term-frequencies ») d'apparition des mots dans les textes. Cela permet de contourner le problème des longueurs différentes des textes. L'application d'une PCA est présentée à la FIGURE 10 page 27. Les textes correspondant aux ingrédients

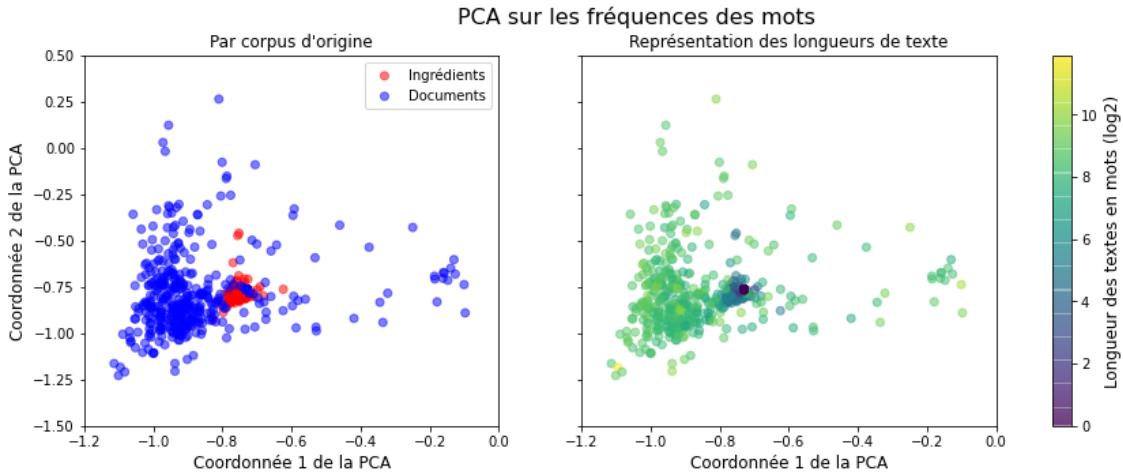


FIGURE 10 – Projection des fréquences : PCA

sont à nouveau localisés dans la même région, bien que cette fois les normes des vecteurs de l'espace de départ soient comparables entre les deux corpus. Une explication serait que les nombreux « trous » dans les vecteurs correspondants aux ingrédients (pour rappel, le vocabulaire des ingrédients est plus de 10 fois plus petit que le vocabulaire des documents, cf. TABLE 7 page 23) les cantonnent à un partie limitée de l'espace de départ. D'où la similarité dans l'espace de la PCA.

Enfin, si on applique une Truncated SVD sur les fréquences de mots, on observe à nouveau une séparation franche des corpus. Le résultat est présenté à la FIGURE 11 page 28. Cette fois, c'est le corpus des documents qui occupe une région limitée de la représentation. On peut expliquer ce point par le fait que du fait que les listes d'ingrédients sont courtes, les « term frequencies » sont élevées (le diviseur étant relativement plus petit). Les vecteurs des listes d'ingrédients contribuent donc à des valeurs singulières importantes, et leurs projections dans les espaces correspondant sont plus longues.

On observe de plus deux « moustaches » distinctes (proches des axes de cette projection, cf. FIGURE 12 page 28). Si on compare les textes de ces deux groupes de points, on s'aperçoit que des tendances nettes se dégagent (cf. TABLE 8 page 29) :

- comme anticipé, il s'agit de listes d'ingrédients relativement courtes (d'où leur norme importante dans cet espace)
- la moustache de droite reprend des listes d'ingrédients correspondant principalement à du thon ou des mono-légumes en boîte (des conserves)
- la moustache du haut correspond essentiellement à des pâtes ou des mono-produits d'épicerie (épices,

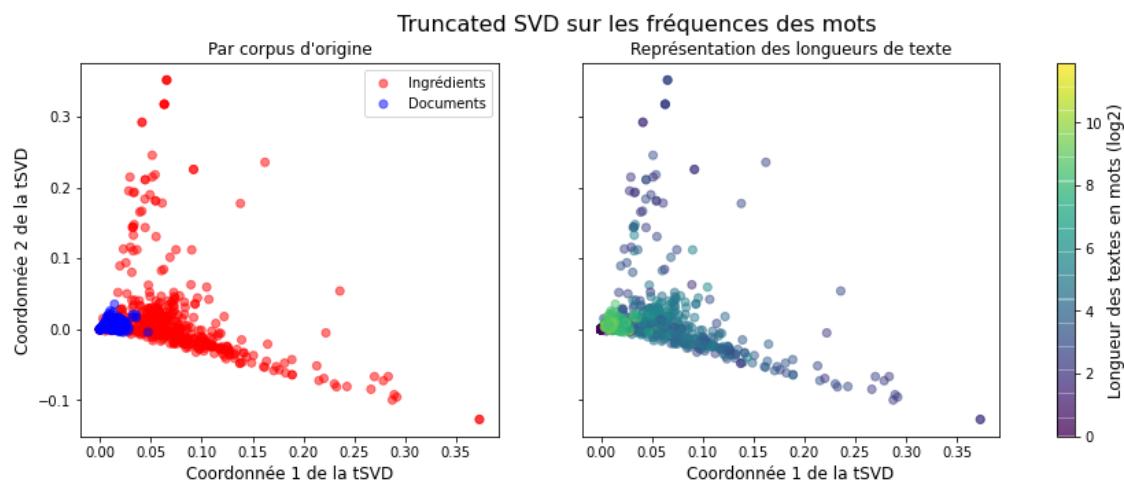


FIGURE 11 – Projection des fréquences : Truncated SVD

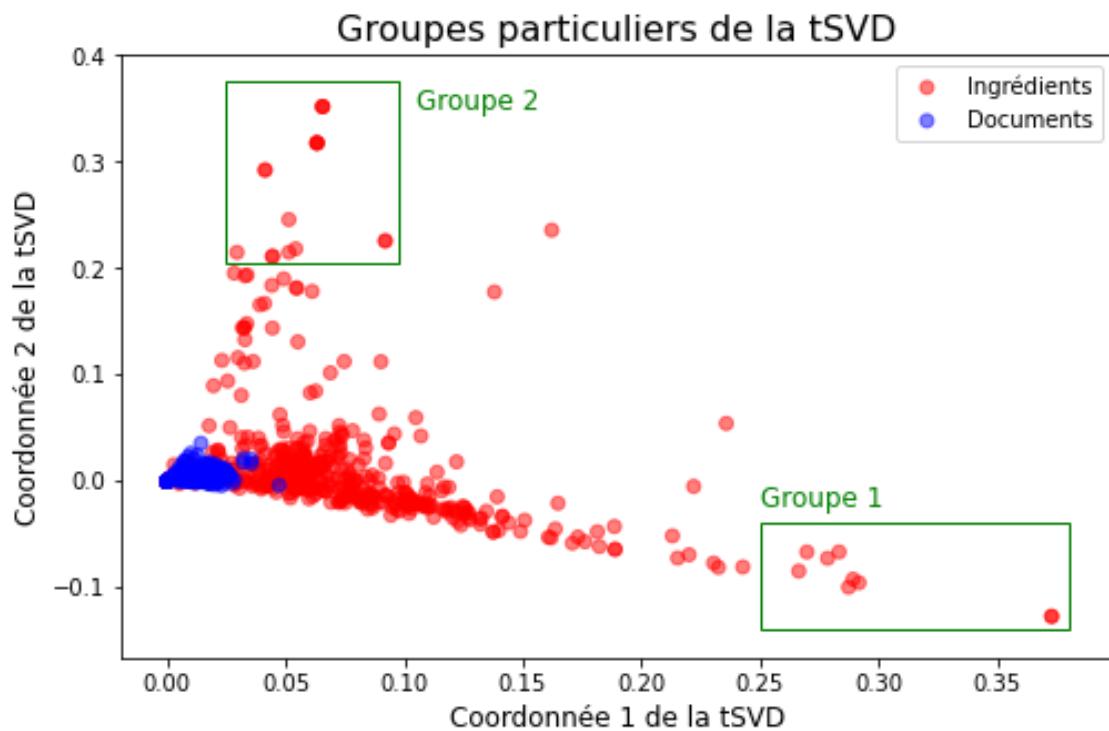


FIGURE 12 – Groupes particuliers dans la tSVD des fréquences

farine, thé, café)

	Texte	x	y
Groupe 1	Thon, eau, sel	0.373079	-0.127052
Groupe 1	Pois chiches, eau, sel.	0.266423	-0.084501
Groupe 1	Haricots beurre, eau, sel	0.283409	-0.066561
Groupe 1	Câpres, eau, vinaigre, sel	0.291819	-0.095479
Groupe 1	Thon albacore, eau, sel	0.287423	-0.099627
Groupe 1	Eau, haricots verts, sel.	0.278588	-0.072474
Groupe 1	Pommes de terre, eau, sel.	0.289291	-0.092162
Groupe 1	Carottes rondelles, eau, sel, sucre	0.269849	-0.066600
Groupe 1	Thon, eau, sel	0.373079	-0.127052
Groupe 2	- 100% Semoule de blé dur de qualité supérieure	0.065551	0.351800
Groupe 2	100% haricots blancs	0.051373	0.214921
Groupe 2	100% Semoule de blé dur de qualité supérieure	0.065551	0.351800
Groupe 2	Semoule de blé dur*	0.054211	0.218192
	*issu de l'agriculture biologique		
Groupe 2	Tilleul (100%).	0.041323	0.292199
Groupe 2	100% semoule de blé dur de qualité supérieure	0.065551	0.351800
Groupe 2	Farine de blé T45	0.092002	0.225689
Groupe 2	- Semoule de blé dur de qualité supérieure	0.044446	0.211188
	- 30% oeufs frais		
Groupe 2	- 100% Semoule de blé dur de qualité courante	0.051337	0.245649
	- Contient du gluten		
Groupe 2	SEMOULE DE BLE' DUR de qualité supérieure	0.063201	0.317653
Groupe 2	100% Arabica	0.041324	0.292222
Groupe 2	agar-agar 100%	0.029466	0.214845
Groupe 2	SEMOULE DE BLE' DUR de qualité supérieure	0.063201	0.317653
Groupe 2	SEMOULE DE BLE' DUR de qualité supérieure	0.063201	0.317653
Groupe 2	SEMOULE DE BLE' DUR de qualité supérieure	0.063201	0.317653
Groupe 2	- Semoule de blé dur de qualité supérieure	0.044446	0.211188
	- 30% oeufs frais		
Groupe 2	Farine de blé T45	0.092002	0.225689

TABLE 8 – Illustration des listes d'ingrédients des groupes particuliers

Troisième partie

CONSTRUCTION DU MODÈLE

Chapitre 6

PROTOTYPAGE

6.1 Premier prototype : modèle simple « ouvert »

6.1.1 Principes généraux

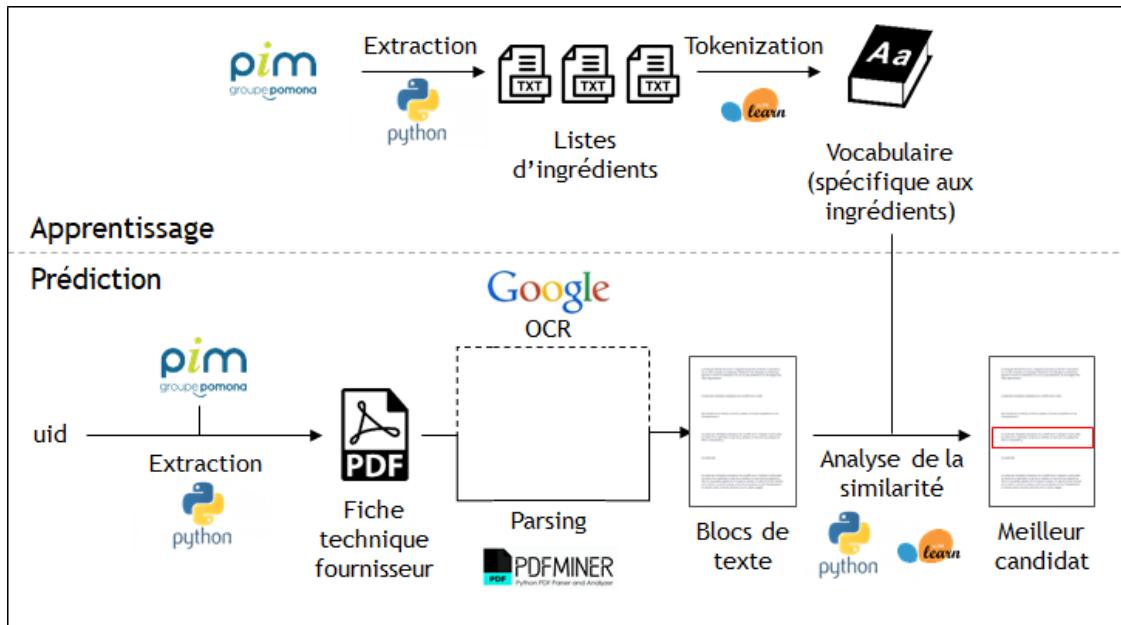


FIGURE 13 – Schéma de principe du « modèle ouvert »

Le fonctionnement global de ce premier modèle (présenté à la FIGURE 13 page 30) ne respecte pas les principes du machine learning. Il permet juste d'éprouver la méthode pressentie, ainsi que de se faire une idée de l'efficacité d'un modèle de ce type. En effet, même si on utilise des fonctionnalités d'extraction de

features depuis des textes classiques dans des modèles de machine learning, il manque une partie de mesure de la performance, indispensable pour pouvoir évaluer et améliorer la pertinence du modèle. Les illustrations de ce chapitre sont issues du notebook présenté en annexe G.3 page 168, et le code des classes utilisées (IngredientExtractor et PIMIngredientExtractor) est inclus dans le module pimest, en annexe H.4 page 250.

La classe PIMIngredientExtractor est juste un habillage du CountVectorizer de la bibliothèque scikit-learn et du Requester construit dans le module pimapi, permettant d'aller récupérer les données du PIM. Ce modèle met en application les principes du « Bag Of Words » [12], à savoir que :

- un vocabulaire est établi à partir de l'ensemble des différents mots du corpus de listes d'ingrédients
- chaque liste d'ingrédients est transformée en un vecteur de nombres entiers qui a la même longueur que le vocabulaire, ou chaque entier est le compte du nombre d'occurrence du mot dans le vocabulaire

Ce modèle n'utilise pas les données étiquetées manuellement (présentées à la section 2.4 page 13), mais se base simplement sur les listes d'ingrédients du PIM. L'hypothèse qui est faite avec ce modèle est la suivante : bien que non parfaitement en qualité (i.e. exactement identiques au contenu des pièces jointes, cf. la comparaison entre la ground truth et le contenu du PIM en section 2.4.5 page 14), les listes d'ingrédients du PIM sont des textes dont le contenu est très similaire à une liste d'ingrédients. Un survol rapide de quelques exemples montre que cette hypothèse semble vérifiée (cf. TABLE 1 page 9).

6.1.2 Entrainement

Périmètre du set d'entraînement

Pour l'entraînement de ce modèle, on va uniquement se limiter aux produits d'épicerie ou de boissons non-alcoolisées. En effet, ce sont pour ces produits que la réglementation impose d'afficher en clair la composition aux consommateurs. On se limitera aussi aux produits qui portent une liste d'ingrédients, et qui sont « En qualité » (cf. les définitions données à la section D.1.2 page 105 sur les statuts des produits). Sur les 13 000+ produits présents dans le PIM, environ 3 400 font partie du périmètre et 9 800 en sont exclus.

Constitution du vocabulaire

La constitution du vocabulaire se fait de manière très basique :

- les listes d'ingrédients sont mises en minuscules
- les mots sont ensuite séparés aux whitespaces (espaces, retours à la ligne, tabulations, ...) et marques de ponctuation

Pour parvenir à ce résultat, on applique simplement la méthode fit de la classe CountVectorizer de la bibliothèque scikit-learn. Aucune des autres fonctionnalités standard (retrait des accents, gestion des stopwords, prise en compte des ngrams) n'a été utilisée. Lors de cet apprentissage, le vocabulaire obtenu a une longueur de 2 500 mots environ, et les mots les plus fréquents sont (ces valeurs sont susceptibles de changer à la marge par rapport au notebook en annexe, avec les changements potentiels sur les données au sein du PIM) :

- de : 11419 occurrences
- sucre : 2057 occurrences
- sel : 1669 occurrences
- eau : 1288 occurrences
- acide : 1241 occurrences
- lait : 1215 occurrences
- huile : 1214 occurrences
- poudre : 1100 occurrences
- en : 962 occurrences
- arôme : 938 occurrences

Les stopwords 'de' et 'en' apparaissent dans les mots les plus fréquents.

6.1.3 Prédiction

Parsing des fiches techniques

Les fiches techniques sont extraites sous formes de binaires depuis de le PIM via des appels API. On utilise ensuite la bibliothèque PDFMiner.six pour récupérer le texte de la fiche technique sous forme d'un long string. Cette étape fait appel à la méthode statique PDFDecoder.content du module pimpdf, présenté en annexe H.3 page 248. On découpe ensuite ce long string en une liste de strings plus court, en considérant comme séparateur la présence de deux retours à la ligne consécutifs. Ce choix de séparateur a été fait car les textes produits par PDFMiner.six portent des retours à la ligne à la fin de chaque ligne typographiée, même si la phrase continue sur la ligne suivante. Cette étape fournit pour chaque fiche technique le texte contenu sous forme d'une liste de strings (les blocs de texte de la FIGURE 13 page 30).

Sélection du meilleur candidat

Principe L'identification du meilleur candidat parmi les blocs de texte se fait de la manière suivante :

- on calcule une similarité avec le vocabulaire des ingrédients pour chacun des blocs de texte
- on retourne le bloc de texte avec la similarité la plus élevée

La formule de calcul de la similarité utilisée dans ce modèle a été trouvée par chance, mais elle donne des résultats plutôt encourageants. D'autres modes de calcul de la similarité seront présentés dans la suite de ce rapport.

La formule de calcul de la similarité est :

$$\frac{\text{Nombre de mots du candidat qui sont des ingrédients}}{\text{Norme euclidienne du vecteur du candidat}}$$

Attention, la norme du vecteur du candidat s'entend *indépendamment du vocabulaire des ingrédients*, c'est à dire que tous les mots comptent, y compris ceux n'appartenant pas au vocabulaire.

Exemple Si on illustre par un exemple fictif : imaginons que le vocabulaire des ingrédients soit composé des mots « *eau* », « *sucré* » et « *farine* ». Le bloc de texte candidat est « *Sucré et farine sont biologiques. La farine est équitable.* » Si on vectorise ce bloc de texte *sur son propre vocabulaire*, on obtient le vecteur suivant :

sucre	et	farine	sont	biologiques	la	est	équitable
1	1	2	1	1	1	1	1

TABLE 9 – Exemple de vectorisation d'un texte

Sa norme euclidienne [19] se calcule de la manière suivante (en notant x_i le compte des mots) :

$$\sqrt{\sum_i x_i^2} = \sqrt{1 + 1 + 4 + 1 + 1 + 1 + 1 + 1} \approx 3.317$$

Le bloc contient 9 mots, dont 3 sont des ingrédients (« *farine* » est mentionné deux fois). La similarité pour cet exemple vaut donc :

$$\frac{3}{3.317} \approx 0.905$$

Étant donné son mode de calcul, cette similarité peut tout à fait être supérieure à 1.

6.1.4 Illustration des résultats obtenus

Périmètre du test

Pour éviter de surestimer la performance du modèle, on le fait tourner sur des produits n'ont pas fait partie du set d'entraînement. Comme ce modèle ne permet de toute façon pas de fournir de résultats qui permettent de mesurer la performance, on ne le fera tourner que sur un échantillon réduit de produits, soit 5 d'entre eux.

Résultats

Les résultats obtenus sont les suivants :

```
Fetching data from PIM for uid d9b233a6-b455-4af6-afb4-623f1f7f62a6...
Done
-----
Ingredient list from PIM is :

Ingrédients: Huile de tournesol, oignon, curry (11,2%) (ail, coriandre, curcuma, gingembre, paprika, poivre, cumin, poivre de Cayenne, fenouil, cardamome, noix de muscade, cannelle, clous de girofle, safran), pomme, sel, exhausteur de goût (glutamate de sodium), sucre, huile de colza totalement hydrogénée, extrait de levure, ail.

Downloading content of technical datasheet file...
Done!
-----
Parsing content of technical datasheet file...
Done!
-----
Ingredient list extracted from technical datasheet:

Liste d'ingrédients : Huile de tournesol, oignon, curry (11,2%) (ail, coriandre, curcuma, gingembre, paprika, poivre, cumin,
```

poivre de Cayenne, fenouil, cardamome, noix de muscade, canelle, clous de girofle, safran), pomme, sel, exhausteur de goût (glutamate de sodium), sucre, huile de colza totalement hydrogénée, extrait de levure, ail

```
=====
```

```
=====
```

Fetching data from PIM for uid 5666235b-9e78-44f2-8e0e-1de53f88fe04...

Done

```
=====
```

Ingredient list from PIM is :

Ingrédients : Sucre, Gomme base, Sirop de glucose, Arômes, Humectant (E422), Antioxydant (E321), Colorant (E141).

```
=====
```

Downloading content of technical datasheet file...

Done!

```
=====
```

Parsing content of technical datasheet file...

Done!

```
=====
```

Ingredient list extracted from technical datasheet:

INGRÉDIENTS : Sucre, Gomme base, Sirop de glucose, Arômes, Humectant (E422), Antioxydant (E321), Colorant (E141).

```
=====
```

```
=====
```

Fetching data from PIM for uid 6e976147-adeb-4d2d-925a-cb7c58c111a2...

Done

```
=====
```

Ingredient list from PIM is :

Maltodextrine, amidon de maïs, sel, farine de BLE, colorant : caramel ordinaire ; arômes (BLE,CELERI), huile de palme, épaississant : gomme guar ; oignon, féculé de pomme de terre, extrait de levure, jus de cuisson de viande de boeuf (0,9%), acidifiant : acide citrique ; extrait de vin blanc, extraits d'ail, de thym et de poivre. Peut contenir : LAIT, OEUFS.

```
=====
```

Downloading content of technical datasheet file...

Done!

```
=====
```

Parsing content of technical datasheet file...

Done!

```
=====
```

Ingredient list extracted from technical datasheet:

Maltodextrine, amidon de maïs, sel, farine de blé, colorant : caramel ordinaire ; arômes (blé,céleri), huile de palme, épaississant : gomme guar ; oignon, féculé de pomme de terre, extrait de levure, jus de cuisson de viande de bœuf (0,9%), acidifiant : acide citrique ; extrait de vin blanc, extraits d'ail, de thym et de poivre. Peut contenir : lait, œufs.

```
=====
```

```
=====
```

Fetching data from PIM for uid db449562-d16d-4f72-b7a5-c0d487bc8206...

Done

```
=====
```

Ingredient list from PIM is :

Huile d'ARACHIDE

```
=====
```

Downloading content of technical datasheet file...

Done!

```
=====
```

Parsing content of technical datasheet file...

Done!

```
=====
```

Ingredient list extracted from technical datasheet:

*Selon le règlement UE n°1259-2011 / In accordance with regulation UE n°1259-2011

CARACTÉRISTIQUES MICROBIOLOGIQUES

L'huile étant un milieu anhydre, tout développement bactérien est impossible (cf. ouvrage de référence dans ce domaine "La qualité microbiologique des aliments" CNERMA-CNRS coordonné par Jean-louis Jouve).

ORIGINES/ ORIGIN

- Amérique du Sud majoritairement
- Afrique de l'Ouest

AUTRES INFORMATIONS

```
-----  
=====  
=====  
Fetching data from PIM for uid f2af54a2-6820-4f1b-99e7-d6e64642bdf3...  
Done  
-----
```

```
Ingredient list from PIM is :
```

```
None  
-----
```

```
Downloading content of technical datasheet file...  
Done!  
-----
```

```
Parsing content of technical datasheet file...  
Done!  
-----
```

```
Ingredient list extracted from technical datasheet:
```

```
mini 16,56 ( - 10 % )  
-----  
=====
```

Pistes d'améliorations identifiées

En plus de la mesure de la performance, qui est indispensable avant de pouvoir procéder à des ajustements, les pistes identifiées à sont les suivantes :

- Faire un découpage plus élaboré du texte des pièces jointes en blocs, potentiellement avec des expressions régulières
- Essayer d'autres manières de vectoriser les textes (utiliser un comptage de mots binaire : présence/absence, calculer l'inverse document frequency, ...)
- Essayer une autre manière de calculer la similarité
- Utiliser des fonctions plus élaborées de text preprocessing (retrait des stopwords, ...)

6.2 Second prototype : industrialisation

L'objectif de ce second prototype est de permettre est d'industrialiser un peu le code afin :

- d'être en mesure inspecter les résultats obtenus et pouvoir donner des explications sur la réponse donnée par le modèle
- de pouvoir utiliser des fonctionnalités de mesure de la performance du modèle
- d'être en mesure de faire varier les paramètres du modèle en vue de l'optimiser

Comme présenté à la section 2.4.5 page 14 relative à la comparaison entre les données du PIM et celles

récupérées lors de l'étiquetage, il y a un grand nombre d'écart. Or, si on entraîne le modèle et qu'on mesure sa performance sur des données de mauvaise qualité, on aura de mauvais résultats. Ce second modèle se basera sur les données manuellement étiquetées, afin d'avoir une base de comparaison dans la mesure de la performance. Le fonctionnement de ce modèle est présentés à la FIGURE 14 page 36. La méthodologie utilisée à cette partie est présentée dans le notebook « Modèle basé sur les données manuellement étiquetées » en annexe G.4 page 174. Les différents transformateurs et estimateurs spécifiques sont définis dans le module pimest, inclu en annexe H.4 page 250.

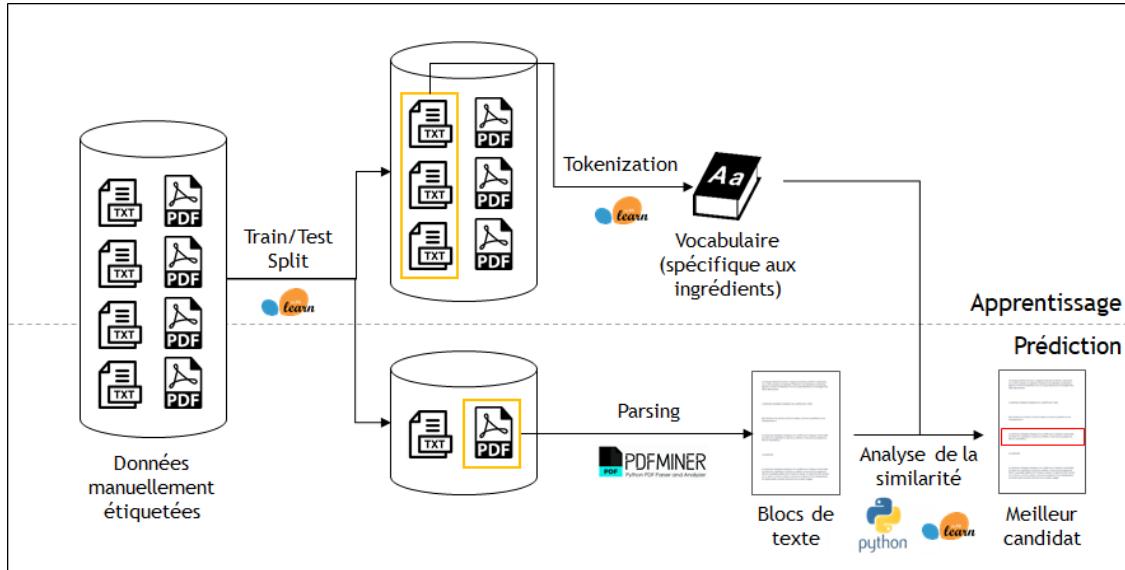


FIGURE 14 – Schéma de principe du modèle basé sur les données étiquetées

6.2.1 Chargement des données manuellement étiquetées

La toute première étape est la constitution d'un dataframe contenant :

- les uid pour indexer les produits
- les listes d'ingrédients manuellement étiquetées depuis les fiches techniques
- le contenu de chacune des fiches techniques au format texte

On commence par charger les données du fichier csv contenant les uid et les listes d'ingrédients. Ensuite, un pipeline scikit learn d'acquisition des données est lancé. Il s'agit de 4 transformateurs en série, qui effectuent les travaux suivants :

- construction du chemin pointant vers les fiches techniques (sur la base des uid)
- construction d'une feature contenant les données des fichiers, en binaire
- construction du texte complet de la fiche technique (en se basant sur la library pdfminer.six)

Le résultat du lancement de ce pipeline est présenté à la TABLE 10 page 37.

text

NESCAFÉ® SPÉCIAL FILTRE\n\nDose individuelle de 2 g\nTechnologie micro-grains\nCODE EAN (UC)\n\n3033710076017\nDENOMINATION LEGALE DU PRODUIT\nDESCRIPTION DU PRODUIT\nCafé instantané et café torréfié moulu.\nUne dominante Arabica pour l'arôme et une pointe de Robusta pour le ncorsé, associés à une torréfaction légère pour un café équilibré et peu \namer.\nSachet dose pour une tasse.\n\nDOSAGE PRECONISÉ\n\nMODE OPERATOIRE\nPour obtenir\n1 café Court (DA)\n1 café Long (DA)\nEau\n7\n12\nncl\nncl\nNESCAFÉ®\n\nSPÉCIAL FILTRE\nn2\nn2\nng\nng\nA reconstituer avec de l'eau. \nTempérature de l'eau : 75°C\nPour une qualité optimale, utilisez de l'eau filtrée.\nIngrédients : Café instantané, café torréfié moulu (3%).\nINGRÉDIENTS\n\nPROFIL GUSTATIF\n\nIntensité\nConditionné sous atmosphère protectrice.\n\nENGAGEMENT QUALITÉ\n- NESTLÉ a un système de management de la qualité, le NMS (NESTLÉ \nManagement System), en cohérence avec les systèmes ISO 9001 ...

LENTILLES BLONDES 4mm\nRéférence PQG007-3.22.1\nVersion\nDate d'application :\nPage 1/2\nG\nn15/10/2019\nPrésentation\nCaractéristiques\nnques\nnphysico-\nnchimiques\nnDéfinition\nnOrigine\nnDénomination\nnlégale\nLentilles de couleur brun clair. Elles sont de forme biconvexe et npossèdent une peau assez épaisse. Leur diamètre est compris entre 4mm et 5mm\nnChine, Canada, France, Italie, USA, Turquie\nnLentilles blondes\nnProcess\nnNettoyage, épierrage, triages\nnConservation\nn36 mois à fabri de la chaleur et de l'humidité\nnCritères d'analyses\nnMoyenne/Tolérance\nnMéthodes\nnHumidité\nnMatières minérales végétales\nnGraines\nnImpropres\nnBrisées\nnGermées\nnCalibre 4-5 mm\nn11,5% / 16%max\nn0,05% / 1%max\nn0,15% / 0,5%max\nn0,5% / 1%max\nn0,4% / 1%max\nn0,05% / 1%max\nn95% / 90%min\nnNF V03707\nnMicrobiologie\nnIl n'existe pas de réglementation concernant les exigences microbiologiques npour ce produit.\n\nPesticides\nnMét...

FICHE TECHNIQUE\n\nPRODUIT FINI\nn000100\nnPurée de Poire Sans Sucres Ajoutés\nnDate d'application : 05/05/2014\nnPage : 1/2\nnCoupelles Aluminium 120 x 95 g\nnDéfinition\nnCe produit est une purée de fruits obtenue à partir des parties comestibles des fruits (après broyage et sans nconcentration notable).\nCe produit est sans sucres ajoutés : il contient uniquement les sucres naturellement présents dans les fruits.\nLa purée présente une texture homogène et légèrement granuleuse.\nLa stabilité du produit est obtenue par pasteurisation et dosage à chaud.\nAspects nutritionnels\nnDésignation et liste des ingrédients\nnValeurs nutritionnelles (pour 100 g)\n\nDésignation légale :\n\nnPurée de Poires sans sucres ajoutés *\n* Contient les sucres naturellement présents dans les fruits\nnListe des ingrédients :\n\nPoire 99,9%, antioxydant : acide ascorbique.\n\nMatières grasses\nnEnergie\nn65 kcal\nn273 kJ\nndont acides gras saturés\nnGlucides\nnFibres alimentaires\nPro...

TABLE 10 – Exemples du contenu de fiches techniques au format texte (tronqués)

6.2.2 Découpage des textes en blocs

Le second travail est le découpage des textes en blocs. Cette étape étant peu coûteuse en temps de calcul, elle est réalisée sur l'ensemble du dataset avant le split entre set d'entraînement et set de test. Dans un premier temps, on va simplement effectuer ce découpage en coupant le texte lorsque deux retours à la ligne successifs sont détectés, comme cela avait été fait pour le premier prototype. Il est maintenant possible d'inspecter les résultats de cette étape, un exemple de découpage est présenté ci-dessous.

30/12/19	/ (Latin name)	tés (ail, oignon, poivron rouge), sel de mer, sucre d'érable, arôme d'érable naturel, huile végétale (canola)
Date d'impression :	TEEPTRAPPEUR	Sugar, black pepper, coriander, dehydrated vegetables (garlic, onion, red bell pepper), sea salt, maple sugar, natural maple aroma, vegetable oil (canola)
Remarque :	X	
Les informations contenues dans cette fiche technique sont données de bonne foi, en l'état actuel de nos connaissances, et selon les indications communiquées par le producteur ou le fournisseur. Il appartient au client de vérifier la conformité de la marchandise par rapport à l'usage qu'il en fait.	3760063322262	
Création :	Poids net	
12/06/12	Poids brut	
12 rue René Cassin 37390 NOTRE DAME	Origine	
Tél : 02 47 85 55 00 Fax : 02 47 41 33 32	/ net weight	
FICHE TECHNIQUE	/ gross weight	
Mélange du trappeur, 70 g Trapper blend, 70g	/ Origin	
Code article KEREX Nom latin (si disponible) / EAN Code Code barre	0,07 Kilogramme 0,125 Kilogramme CANADA	
/ KEREX Code	/ General information	
	Informations générales DLUO conseillée / "Best before date" recommandé	
	Nomenclature douanière / Customs code	
	Conditions idéales de stockage	
	/ Conditions of storage	
	Ingrédients :	
	Conserver dans un endroit frais et sec Store in a cool dry place	
	5 ans / 5 years 0910999900	
	Sucre, poivre noir, coriandre, légumes déshydra-	

In accordance with Reg 396/2005 /CE.	/ Mustarde	Salmonelles
Conforme au règlement 1881/2006 /CE	/ Sésame	Levures
In accordance with Reg 1881/2006 /CE..	/ Sulphites	Moisissures
Gluten	/ Lupin	Aflatoxine Total
Crustacés	/ Shellfish	Aflatoxine B1
Oeufs	Absence	/
Poisson	Absence	Total plat count (APC)
Soja	Absence	E. Coli
Lait	Absence	/
Fruits à coque - Arachides	Absence	/ Salmonella
Céleri	Absence	/ Yeasts
Moutarde	Absence	/ Moulds
Sésame	Absence	/ Total aflatoxin
Sulfites	Absence	B1 aflatoxin
Lupin	Absence	/
Mollusques	Absence	NF V05-051 < 6 000 000 / g
/ Gluten	Absence	NF V08-053 < 10 / g
/ Crustaceans	Absence	NF V08-052 Absence dans 25g
/ Eggs	Caractères microbiologiques	NF V08-059 < 10 000 / g
/ Fish	/ Microbiological characteristics	NF V08-059 < 10 000 / g
/ Soy		Kit Enzymatique < 10 ppb
/ Milk		Kit Enzymatique < 5 ppb
/ Peanuts and Treenuts	Microorganismes aérobies 30 °C	
/ Celery and celeriac	Escherichia coli	

On constate que le découpage n'est pas idéal, cf. la fiche technique de ce produit, présentée en annexe E.1.7 page 124. Les séparations des cellules des tableaux de cette fiche ne sont pas prises en compte, et on a des blocs trop étendus.

6.2.3 Entrainement et prédition

Dans la mesure où l'on possède assez peu de données, on va conserver un échantillon assez important dans le jeu d'entraînement : 400 produits (soit 80% des données disponibles). Comme cela a été présenté ci-dessus, l'entraînement ne consiste qu'en la détermination du vocabulaire des ingrédients à partir des textes manuellement étiquetés. On fait tourner de la même manière que sur le modèle dit « ouvert », à savoir qu'on ne préprocesse que peu les données avant d'appliquer le CountVectorizer. Sur le set d'entraînement retenu, on a un vocabulaire d'ingrédients représentant 1204 mots.

La prédition du meilleur candidat se fait de la même manière que dans le cadre du prototype précédent (cf. section 6.1.3 page 32) : on compare la norme du vecteur du bloc avec le nombre de mots appartenant au vocabulaire des ingrédients. La représentation des textes sous forme de vecteur se fait simplement via les comptes de chacun des mots du texte.

6.2.4 Illustration des prédictions obtenues

Un échantillon des prédictions obtenues est présenté dans la TABLE 11 page 39. Pour éviter d'avoir des listes d'ingrédients prédites prenant trop de place dans cette table, celles dont la longueur dépasse 500 caractères ont été filtrées avant génération de cet échantillon. Les résultats présentés à cette table sont donc vraisemblablement biaisés, dans la mesure où les très longues listes prédites doivent avoir plus de chance d'être erronées. Les grandes tendances qui se dégagent à l'analyse de cette liste sont les suivantes :

- globalement, les résultats sont bons. On retrouve régulièrement des morceaux de texte qui sont similaires à la liste cible
- une erreur qui revient régulièrement est le fait que le découpage en blocs est parfois imparfait, on sélectionne « trop large »

- à l'inverse, le modèle n'a pas retiré des listes d'ingrédients prédites des mentions qui ont été retirées lors de l'étiquetage manuel (cf. les règles d'annotation présentées en annexe E.3.1 page 131) : les préfixes de type « Liste d'ingrédients : », les allégations telles que « Teneur totale en sucres : 60g pour 100g »
- ...
- le modèle semble plus performant lorsque la liste d'ingrédients réelle est longue. On le vérifiera dans le chapitre relatif à la mesure de la performance du modèle

Un mot sur les cas où la liste d'ingrédients cible ou prédite sont vides :

- Les listes d'ingrédients cible sont vides lorsque la pièce jointe ne mentionnait pas de liste d'ingrédients. Cela peut arriver, et les produits concernés n'ont pas été sortis de l'échantillon. Il est important de pouvoir aussi mesurer les faux positifs, qui sont nombreux avec cette technique de choix systématique du meilleur candidat
- Les listes d'ingrédient prédites sont vides lorsque l'outil de parsing des pdf (pdfminer.six) n'a extrait aucun texte. C'est le cas quand la pièce jointe était un document imprimé qui a été scanné. Le texte n'est présent que sous forme d'image (cf. la fiche technique du sel en annexe E.1.1 page 113)

TABLE 11 – Extrait des résultats de la prédiction

Liste d'ingrédients cible	Liste d'ingrédients prédite
sucre*, LAIT en poudre*, beurre de cacao*, pâte de cacao*, émulsifiant : lécithine de tournesol (E322), extrait de vanille* * matière première issue de l'agriculture biologique cacao : 27% minimum	Liste des Ingrédients : sucre*, LAIT en poudre*, beurre de cacao*, pâte de cacao*, émulsifiant : lécithine de tournesol (E322), extrait de vanille* * matière première issue de l'agriculture biologique
<rien>	<rien>
Amidon de maïs* - Lait écrémé* - Sel - Fécule de pomme de terre* - Tomate* - Oignon* - Arômes naturels - Poivres* 3 % (poivre vert*, poivre blanc*, poivre noir*) - Huile de tournesol* - Extrait de levure* - Sucre caramélisé* - Ail* - Maltodextrine de maïs*. * issus de l'agriculture biologique	Amidon de maïs* - Lait écrémé* - Sel - Fécule de pomme de terre* - Tomate* - Oignon* - Arômes naturels - Poivres* 3 % (poivre vert*, poivre blanc*, poivre noir*) - Huile de tournesol* - Extrait de levure* - Sucre caramélisé* - Ail* - Maltodextrine de maïs*. * issus de l'agriculture biologique
semoule de blé dur supérieure et de l'eau	Ingrediënts : semoule de blé dur supérieure et de l'eau
<rien>	Boisson gazeuse aromatisée au jus de fruit à base de concentré S.Pellegrino Orange 33 cl (Aranciata) S.Pellegrino Citron 33 cl (Limonata) Le plaisir des fruits à l'italienne
Eau, maltodextrine, sel, arômes, sucre, arôme naturel de citronnelle, amidon modifié, ail en poudre, épices (com-bava, curcuma), extraits d'épices (gingembre, poivre), stabilisant (gomme xanthane).	Eau, maltodextrine, sel, arômes, sucre, arôme naturel de citronnelle, amidon modifié, ail en poudre, épices (com-bava, curcuma), extraits d'épices (gingembre, poivre), stabilisant (gomme xanthane).
Sucre, cacao maigre en poudre (beurre de cacao : 11% minimum), arôme vanille. Cacao : 32% minimum	Sucre, cacao maigre en poudre (beurre de cacao : 11% minimum), arôme vanille.
Sucre ; sirop de glucose ; graisse de palme ; humectant : sirop de sorbitol ; gélatine ; acidifiant : acide citrique ; arôme.	<rien>
Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.	Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.

Continued on next page

Liste d'ingrédients cible	Liste d'ingrédients prédicté
Purée de tomates mi réduite (64%), sucre, vinaigre, amidon modifié, sel, acidifiant : acide citrique	Liste des ingrédients : Purée de tomates mi réduite (64%), sucre, vinaigre, amidon modifié, sel, acidifiant : acide citrique
<rien>	Boisson gazeuse aromatisée au jus de fruit à base de concentré S.Pellegrino Orange 33 cl (Aranciata) S.Pellegrino Citron 33 cl (Limonata) Le plaisir des fruits à l'italienne
Sirop de glucose-fructose, framboises 35%, sucre, gélifiant : pectines, acidifiant : acide citrique.	Liste ingrédients : Sirop de glucose-fructose, framboises 35%, sucre, gélifiant : pectines, acidifiant : acide citrique. Préparée avec 35g de fruits pour 100g de produit fini. Teneur totale en sucres : 60g pour 100g.
Café instantané, café torréfié moulu (3%).	- NESTLÉ a un système de management de la qualité, le NMS (NESTLÉ Management System), en cohérence avec les systèmes ISO 9001 et ISO 22000. - Etiquetage conforme à la réglementation en vigueur sur les OGM - Ce produit ne contient pas d'ingrédients ionisés. - Certifications usines : ISO 9001, FSSC 22000, ISO 14001 et OHSAS 18001. - Agrément sanitaire : site non soumis à agrément sanitaire
sucre, pâte de cacao, beurre de cacao, cacao maigre en poudre, émulsifiant : lécithine de tournesol (E322), arôme vanille cacao : 50% minimum	Liste des Ingrédients : sucre, pâte de cacao, beurre de cacao, cacao maigre en poudre, émulsifiant : lécithine de tournesol (E322), arôme vanille
Eau, huile de tournesol, beurre 9,5 %, jaune d'oeuf 6 %, amidon modifié, sel, jus de citron concentré, amidon de maïs, épaississants (gomme guar, gomme xanthane), protéines de pois, sucre, arôme naturel, curcuma, extrait de paprika.	Eau, huile de tournesol, beurre 9,5 %, jaune d'oeuf 6 %, amidon modifié, sel, jus de citron concentré, amidon de maïs, épaississants (gomme guar, gomme xanthane), protéines de pois, sucre, arôme naturel, curcuma, extrait de paprika.
Piment rouge fort équeuté* (85%), cumin, ail moulu (3%), eau, arôme naturel d'ail, sel, sorbate de potassium. (* présence de sulfites)	A) Ingrédients : Piment rouge fort équeuté* (85%), cumin, ail moulu (3%), eau, arôme naturel d'ail, sel, sorbate de potassium. (* présence de sulfites) B) Origines des ingrédients : - Piments : Maroc - Ail : chine - Sel, arôme, épices, sorbate de potassium : Europe I) A) Critères Physico-chimiques :
Sucre, amidon de maïs, arôme vanille	B) Critères microbiologiques : ajouter le produit à la préparation avec les autres ingrédients de la recette et mélanger pour homogénéiser
Salicornes de culture, eau, sel, acide citrique	Se consomment en légumes d'accompagnement avec toutes préparations de poissons ou crustacés ou aussi avec des viandes blanches

Continued on next page

Liste d'ingrédients cible	Liste d'ingrédients prédicté
cèpes 70% (Boletus edulis et respective famille), huile de tournesol, blanquette 5% (Tuber borchii Vitt.), oignon, beurre, sel, protéines du lait, farine de riz, amidon de maïs, dextrose, extrait de levure, épices, arôme truffée, arômes naturels, antioxydant : acide l-ascorbique (E 300).	CODE DU PRODUIT : NOME DU PRODUIT : FORMAT : BARCODE (EAN13) : NOMENCLATURE COMBINÉE :
	Ingrédients : cèpes 70% (Boletus edulis et respective famille), huile de tournesol, blanquette 5% (Tuber borchii Vitt.), oignon, beurre, sel, protéines du lait, farine de riz, amidon de maïs, dextrose, extrait de levure, épices, arôme truffée, arômes naturels, antioxydant : acide l-ascorbique (E 300).
	Allergène
Eau, haricots verts, sel.	Égoutter, ne pas rincer. Faire sauter 3 minutes avec de la matière grasse.

Chapitre 7

MESURE DE LA PERFORMANCE

Comme vu aux chapitre précédents, il est indispensable de mesurer la performance de nos modèles. On le fera sur la base du modèle précédent, se basant sur les données manuellement étiquetées. Le principe est présenté à la FIGURE 15 page 42.

7.1 Accuracy

La métrique qui tombe le plus sous le sens est l'accuracy (on utilisera le terme anglais pour éviter les confusions avec la notion de « precision » telle qu'elle est utilisée par exemple dans le f1-score). On mesure simplement la proportion de prédictions qui sont égales à la ground truth. La méthodologie utilisée est détaillée dans le notebook « Mesure de la performance » présenté en annexe G.5 page 180.

7.1.1 Approche naïve

Description de cette approche

L'approche « naïve » consiste simplement à mesurer la proportion de textes prédits strictement égaux à la ground truth. Or, comme cela a été vu précédemment :

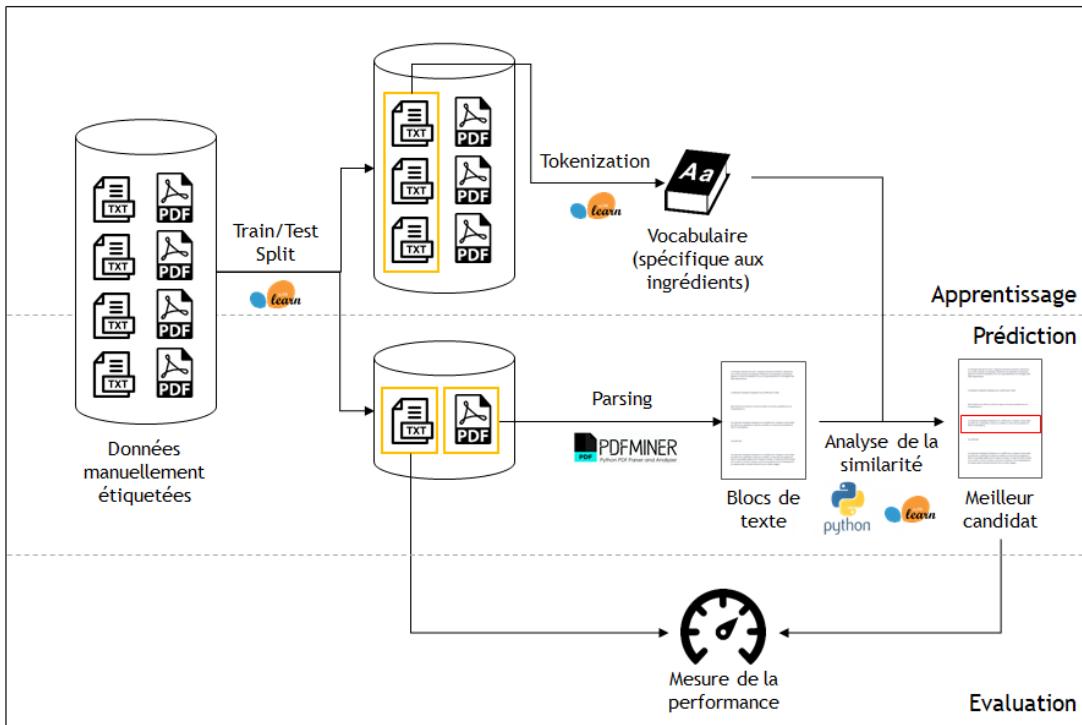


FIGURE 15 – Illustration de la méthodologie de mesure de la performance

- les règles d'étiquetage manuel, détaillées à l'annexe E.3.1 page 131, montrent que des transformations sont parfois appliquées au contenu des listes d'ingrédients des pièces jointes avant d'établir la ground truth
- les textes à comparer sont longs (jusqu'à quelques centaines de caractères), cf. TABLE 11 page 39
- la mise en forme, en particulier les retours à la ligne, ne sont pas positionnés aux mêmes endroits. Dans le parsing des documents pdf, lorsque le texte revient à la ligne après avoir atteint le bord de la page, l'outil positionne un retour à la ligne. Ce comportement n'a pas été reproduit dans l'établissement de la ground truth
- le découpage en blocs de texte, de manière simple, produit des textes qui ne sont pas toujours le reflet du contenu spatialisé de ce document (cf. l'exemple donné à la section 6.2.2 page 37, et la fiche technique associée en annexe E.1.7 page 124)

On s'attend donc à avoir une « accuracy naïve » faible.

Les résultats obtenus

Comme présenté dans notebook « Analyse de la performance » (cf. annexe G.5 page 180), les résultats obtenus sont conformes à l'attendu. L'accuracy est très faible : 1% (1 échantillon sur 100 produits) lorsqu'on la mesure sur l'échantillon de test après entraînement sur l'échantillon d'entraînement. Le seul produit pour

lequel la liste d'ingrédients a été correctement identifiée porte les ingrédients suivants.

Sirop de glucose, sucre, eau, stabilisants (E440i, E440ii, E415), acidifiants (E330, E450i), conservateur (E202).

Si on effectue une cross-validation sur l'ensemble des données étiquetées, en appliquant un découpage en 10 folds, on obtient une accuracy moyenne de $1.80\% \pm 1.89\%$.

Nécessité d'améliorer cette métrique

On a vu que l'accuracy calculée de manière naïve porte un jugement sévère sur la performance du modèle. Par exemple, à la troisième ligne de la TABLE 11 page 39, on voit bien que la liste d'ingrédients prédite est identique à la ground truth, si ce n'est qu'les retours à la ligne ne sont pas positionnés exactement au même endroit. Comme on souhaite pouvoir ajuster le modèle, il est nécessaire d'avoir une métrique de mesure de la performance qui soit plus précise.

7.1.2 Avec du « text-postprocessing »

Le principe

Afin de pallier ces problèmes de mise en forme de texte, on assouplit un peu les contraintes par rapport à l'égalité stricte. En effectuant un traitement de text processing, à la fois sur la ground truth et les résultats du modèle, on va comparer des textes un peu plus « standardisés ». Les traitements effectués sont les suivants :

- On passe le texte en minuscules
- On retire la ponctuation
- On remplace tous les « whitespaces » (retours à la ligne, espaces multiples, tabulations, ...) par des espaces simples
- On retire les accents

L'ensemble de ces transformations sont faites en utilisant les fonctionnalités proposées par le CountVectorizer de la bibliothèque scikit-learn (cf. le notebook en annexe G.5 page 180 et le module pimest inclus en annexe H.4 page 250).

Les résultats

Si on évalue cette nouvelle accuracy avec le text processing, sur l'échantillon d'entraînement après entraînement sur l'échantillon d'entraînement, on obtient une accuracy de 14% (14 listes d'ingrédients correctement prédites sur 100). Ces 14 listes d'ingrédients sont présentées à la TABLE 12 page 44.

De la même manière que précédemment, si on fait une cross-validation sur l'ensemble des données manuellement étiquetées, on obtient une accuracy de $16.60\% \pm 3.35\%$.

Liste d'ingrédients cible	Liste d'ingrédients prédicté
Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.	Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.
Edulcorants sorbitol, isomalt, sirop de maltitol, aspartame, mannitol, sel d'aspartame-acesulfame, acesulfame-k, sucralose; gomme base (contient de la lecithine de SOJA), aromes, épaississant gomme arabique, humectant glycerol, colorant E171, agent d'enrobage cire de carnauba, colorant E163, antioxygène BHA. Contient une source de PHENYLALANINE.	Edulcorants sorbitol, isomalt, sirop de maltitol, aspartame, mannitol, sel d'aspartame-acesulfame, acesulfame-k, sucralose; gomme base (contient de la lecithine de SOJA), aromes, épaississant gomme arabique, humectant glycerol, colorant E171, agent d'enrobage cire de carnauba, colorant E163, antioxygène BHA. Contient une source de PHENYLALANINE.
mini poivrons jaunes, eau, sucre, sel, affermissant chlorure de calcium : E509, acidifiant : acide citrique, vinaigre, antioxydant : vitamine C E330	mini poivrons jaunes, eau, sucre, sel, affermissant chlorure de calcium : E509, acidifiant : acide citrique, vinaigre, antioxydant : vitamine C E330
Farine de BLE, huile de colza non hydrogénée, OEUFS de poules élevées en plein air (21%), sucre, stabilisant : glycérol, sirop de glucose-fructose, émulsifiant : mono- et diglycérides d'acides gras, poudres à lever : diphosphates et carbonates de sodium (BLE), féculle, sel, arôme.	Farine de BLE, huile de colza non hydrogénée, OEUFS de poules élevées en plein air (21%), sucre, stabilisant : glycérol, sirop de glucose-fructose, émulsifiant : mono- et diglycérides d'acides gras, poudres à lever : diphosphates et carbonates de sodium (BLE), féculle, sel, arôme.
Pommes de terre 59,5 % - Céleris 40 % - Amidon de maïs - Sirop de glucose de maïs - Huile de colza - Emulsifiants : E322, E471 - Stabilisant : E450i - Curcuma - Conservateur : E223 - Antioxydant : E304 - Acidifiant : E330.	Pommes de terre 59,5 % - Céleris 40 % - Amidon de maïs - Sirop de glucose de maïs - Huile de colza - Emulsifiants : E322, E471 - Stabilisant : E450i - Curcuma - Conservateur : E223 - Antioxydant : E304 - Acidifiant : E330.
<rien>	<rien>
Amidon de maïs* - Lait écrémé* - Sel - Fécule de pomme de terre* - Tomate* - Oignon* - Arômes naturels - Poivres* 3 % (poivre vert*, poivre blanc*, poivre noir*) - Huile de tournesol* - Extrait de levure* - Sucre caramélisé* - Ail* - Maltodextrine de maïs*. * issus de l'agriculture biologique	Amidon de maïs* - Lait écrémé* - Sel - Fécule de pomme de terre* - Tomate* - Oignon* - Arômes naturels - Poivres* 3 % (poivre vert*, poivre blanc*, poivre noir*) - Huile de tournesol* - Extrait de levure* - Sucre caramélisé* - Ail* - Maltodextrine de maïs*. * issus de l'agriculture biologique
Farine de FROMENT, poudre de LACTOSERUM, sucre, poudre d'OEUF entier, poudres à lever : (E450, E500), matière grasse LAITIERE, sel. Peut contenir des traces de : soja, fruits à coques, lupin.	Farine de FROMENT, poudre de LACTOSERUM, sucre, poudre d'OEUF entier, poudres à lever : (E450, E500), matière grasse LAITIERE, sel. Peut contenir des traces de : soja, fruits à coques, lupin.
Eau, maltodextrine, sel, arômes, sucre, arôme naturel de citronnelle, amidon modifié, ail en poudre, épices (com-bava, curcuma), extraits d'épices (gingembre, poivre), stabilisant (gomme xanthane).	Eau, maltodextrine, sel, arômes, sucre, arôme naturel de citronnelle, amidon modifié, ail en poudre, épices (com-bava, curcuma), extraits d'épices (gingembre, poivre), stabilisant (gomme xanthane).
OEUFS, farine de BLE, sucre, amidon de BLE, stabilisants : sorbitols- glycérol, cacao maigre en poudre (3,5%), émulsifiants : E472b - E477, poudres à lever : E450 - E500, sirop de glucose, LAIT écrémé en poudre, sel, conservateur : E202, épaississant : E410, arôme.	OEUFS, farine de BLE, sucre, amidon de BLE, stabilisants : sorbitols- glycérol, cacao maigre en poudre (3,5%), émulsifiants : E472b - E477, poudres à lever : E450 - E500, sirop de glucose, LAIT écrémé en poudre, sel, conservateur : E202, épaississant : E410, arôme.
Sirop de glucose, sucre, eau, stabilisants (E440i, E440ii, E415), acidifiants (E330, E450i), conservateur (E202).	Sirop de glucose, sucre, eau, stabilisants (E440i, E440ii, E415), acidifiants (E330, E450i), conservateur (E202).
Flageolets verts. Jus : eau, sel, affermissant : chlorure de calcium (E509)	Flageolets verts. Jus : eau, sel, affermissant : chlorure de calcium (E509)
Carottes, eau, sucre, sel, vinaigre d'alcool, acidifiant : acide lactique. Présence fortuite de CELERI	Carottes, eau, sucre, sel, vinaigre d'alcool, acidifiant : acide lactique. Présence fortuite de CELERI
Légumes 43,2 % (pomme de terre, oignon, carotte, tomate, poireau) - Amidon modifié de pomme de terre - Extrait de levure - Sirop de glucose de maïs - Huile de colza - Sucre - Arôme naturel - Ail - Curcuma.	Légumes 43,2 % (pomme de terre, oignon, carotte, tomate, poireau) - Amidon modifié de pomme de terre - Extrait de levure - Sirop de glucose de maïs - Huile de colza - Sucre - Arôme naturel - Ail - Curcuma.

TABLE 12 – Prédictions identifiées comme correctes après postprocessing

Les limites de cette métrique

Cette métrique est déjà plus intéressante que l'approche naïve, mais elle a quand même un défaut majeur : elle a une vision encore trop binaire des résultats. En effet, que le texte soit identique à un préfixe près (cf. la quatrième ligne de la TABLE 11 page 39), ou qu'il n'ait rien à voir (d'autres exemples sont présents dans cette même table), elle considérera la prédiction comme erronée. Or, il est important d'identifier les cas où le modèle s'est complètement trompé par rapport à ceux où il a quand même identifié le bon bloc contenant les ingrédients.

7.2 Fonctions de « similarité » spécifiques

On peut définir des fonctions de similarité, qui permettent d'être plus fin qu'une simple évaluation binaire du résultat du modèle. Cela permet de prendre plus finement en compte les cas où le modèle a identifié un bloc très similaire à la ground truth. On va pour cela s'appuyer sur diverses métriques permettant de mesurer l'écart entre des chaînes de caractères. À chaque fois, on définira une fonction de scoring qui sera une similarité. De plus, elles seront normées pour prendre leurs valeurs entre 0 (chaînes de caractères totalement différentes) et 1 (chaînes identiques), ce qui permettra d'exprimer ces métriques en pourcentage. Ces similarités seront calculées après application du text-postprocessing, comme décrit à la section 7.1.2 page 43. L'ensemble de ces similarités seront calculées sur les caractères.

7.2.1 Similarité basée sur la distance de Levenshtein

La distance de Levenshtein [17] entre deux chaînes de caractères est la distance d'édition pour passer de l'une à l'autre en prenant en compte les transformations suivantes :

- insertion d'un caractère
- suppression d'un caractère
- substitution d'un caractère par un autre

Cette distance possède les caractéristiques suivantes :

- elle peut se calculer entre deux chaînes de longueur différentes
- elle a pour minimum 0 (si et seulement si les deux chaînes sont identiques)
- a pour majorant la longueur de la plus longue des deux chaînes

Pour construire une fonction de similarité telle que présentée en introduction de cette section, on appliquera la transformation suivante :

$$sim_{lev}(s_1, s_2) = 1 - \frac{dist_{lev}(s_1, s_2)}{\max(\text{len}(s_1), \text{len}(s_2))}$$

en notant $dist_{lev}$ la distance de Levenshtein, $\text{len}(s)$ la longueur de la chaîne s , et s_1 et s_2 les chaînes de caractères à comparer. Par exemple, la distance entre les chaînes « rateaux » et « chameau » vaut 4 :

- substitution du « r » par un « c »

- insertion du « h »
- substitution du « t » par un « m »
- suppression du « x »

7.2.2 Similarité basée sur la distance de Damerau-Levenshtein

La distance de Damerau-Levenshtein [13] est une variante de la distance de Levenshtein. Elle calcule également une distance d'édition, mais en ajoutant une transformation possible : l'interversion de deux caractères successifs. Les transformations possibles pour cette transformation sont :

- insertion d'un caractère
- suppression d'un caractère
- substitution d'un caractère par un autre
- interversion de deux caractères successifs

Ses caractéristiques sont les mêmes que la distance de Levenshtein (minimum, maximum) ; et elle est toujours inférieure ou égale à la distance de Levenshtein. On convertit cette distance en similarité de la même manière que la distance de Levenshtein.

7.2.3 Similarité de Jaro

La similarité de Jaro [14] est une fonction permettant de mesurer une similarité entre 0 (chaînes complètement différentes) et 1 (chaînes parfaitement identiques). L'heuristique derrière cette similarité est de considérer qu'entre deux chaînes, un caractère est « correspondant » (« matching ») s'il est déplacé de moins de la moitié de la longueur de la plus longue des deux chaînes. Ensuite, la similarité est fonction croissante du nombre de caractères « correspondants » et décroissante du nombre de caractères « correspondants » mal placés. Sur des chaînes longues de quelques dizaines de caractères, la quasi-totalité des caractères seront « correspondants ». Elle est en général plutôt adaptée à des comparaison de chaînes courtes telles que des noms propres ou des mots de passe.

7.2.4 Similarité de Jaro-Winkler

La similarité de Jaro-Winkler [15] est une variante de la similarité de Jaro. Elle possède la même heuristique que la distance de Jaro, avec en plus comme caractéristique de donner un poids plus important aux 4 caractères formant le début de la chaîne de caractères. Cette métrique est encore moins adaptée au cas d'usage que la précédente.

7.2.5 Évaluation de ces similarités sur la ground truth

Les résultats de l'évaluation de chacune de ces similarités sont présentés à la TABLE 13 page 47. On constate que :

	train/test set	cross validation
Levenshtein	48.86%	49.79% +/-3.73%
Damerau-Levenshtein	48.86%	49.80% +/-3.72%
Jaro	63.56%	62.78% +/-3.28%
Jaro-Winkler	65.67%	64.40% +/-3.50%

TABLE 13 – Évaluation du modèle en utilisant les métriques de similarité

- les similarités de Levenshtein et Damerau-Levenshtein donnent des résultats identiques
- les similarités de Jaro et de Jaro-Winkler donnent des évaluations très généreuses de la performance du modèle, comme on pouvait s'y attendre sur la base de textes longs

7.2.6 Décision sur la métrique à utiliser et illustration

Les similarités de Jaro et Jaro-Winkler sont abandonnées car non pertinentes pour notre cas d'usage, qui se base sur des textes de plusieurs dizaines de caractères. Les similarités de Levenshtein et Damerau-Levenshtein donnant des résultats identiques, on choisit la distance de Levenshtein car son implémentation en C (bibliothèque python-Levenshtein) semble plus performante que celle en C++ de la bibliothèque Jellyfish.

Des exemples du début, du milieu et du bas du classement en termes de similarité de Levenshtein sur le jeu de test, après entraînement sur le jeu d'entraînement, sont présentés à la TABLE 14 page 48.

7.2.7 Performance en fonction de la longueur de la liste d'ingrédients

À la section 6.2.4 page 38, on avait mentionné le fait que le modèle semblait moins performant lorsque la liste d'ingrédients cible était courte. Maintenant que nous avons une fonctionnalité de mesure de la performance, il est possible de confirmer ou d'infirmer ce point. La représentation de ce lien est présentée en FIGURE 16 page 48. Il y a bien une corrélation positive entre longueur de la liste d'ingrédients et performance du modèle ($r^2 \approx 0.340$, après retrait des outliers). Le mode de production de ce graphe est présenté à la fin du notebook « Mesure de la performance » inclus en annexe G.5 page 180.

Listes d'ingrédients cibles	Listes d'ingrédients prédites	Lev	Dam-Lev	Jaro	Jaro-Win
Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.	Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.	100.00%	100.00%	100.00%	100.00%
mini poivrons jaunes, eau, sucre, sel, affermissant chlorure de calcium : E509, acidifiant : acide citrique, vinaigre, antioxydant : vitamine C E330	mini poivrons jaunes, eau, sucre, sel, affermissant chlorure de calcium : E509, acidifiant : acide citrique, vinaigre, antioxydant : vitamine C E330	100.00%	100.00%	100.00%	100.00%
Farine de BLE, huile de colza non hydrogénée, OEUFS de poules élevées en plein air (21%), sucre, stabilisant : glycérol, sirop de glucose-fructose, émulsifiant : mono- et diglycérides d'acides gras, poudres à lever : diphosphates et carbonates de sodium (BLE), féculé, sel, arôme.	Farine de BLE, huile de colza non hydrogénée, OEUFS de poules élevées en plein air (21%), sucre, stabilisant : glycérol, sirop de glucose-fructose, émulsifiant : mono- et diglycérides d'acides gras, poudres à lever : diphosphates et carbonates de sodium (BLE), féculé, sel, arôme.	100.00%	100.00%	100.00%	100.00%
Pommes de terre, eau, sel.	Anhydride sulfureux et sulfites en concentrations de plus de 10 mg/kg exprimé en SO2	15.48%	15.48%	48.29%	48.29%
AMANDES décortiquées	Valeur énergétique : Matières grasses : dont ac gras saturés : Glucides : dont sucres : Fibres alimentaires : Dietary fibre Protéines : Sel :	12.80%	12.80%	46.93%	46.93%
Poire 99,9%, antioxydant : acide ascorbique.	Ce produit est une purée de fruits obtenue à partir des parties comestibles des fruits (après broyage et sans concentration notable). Ce produit est sans sucres ajoutés : il contient uniquement les sucres naturellement présents dans les fruits. La purée présente une texture homogène et légèrement granuleuse.	9.67%	9.67%	43.73%	43.73%
100% haricots blancs bio	Les légumes secs sont mis en avant par tous les nutritionnistes pour leurs apports en protéines et fibres. Ils s'inscrivent parfaitement dans le PNNS. Les légumes secs bio bénéficient également d'une agriculture résonnée et donne des garanties en termes de développement durable.	6.62%	6.62%	35.47%	35.47%
Persil	Céréales contenant du gluten (à savoir blé, seigle, orge, avoine, épeautre, Kamut ou leurs souches hybrides) et Produits à base de ces Céréales.	4.55%	4.55%	50.76%	50.76%

TABLE 14 – Illustration de l'évaluation du modèle à l'aide des métriques de similarité

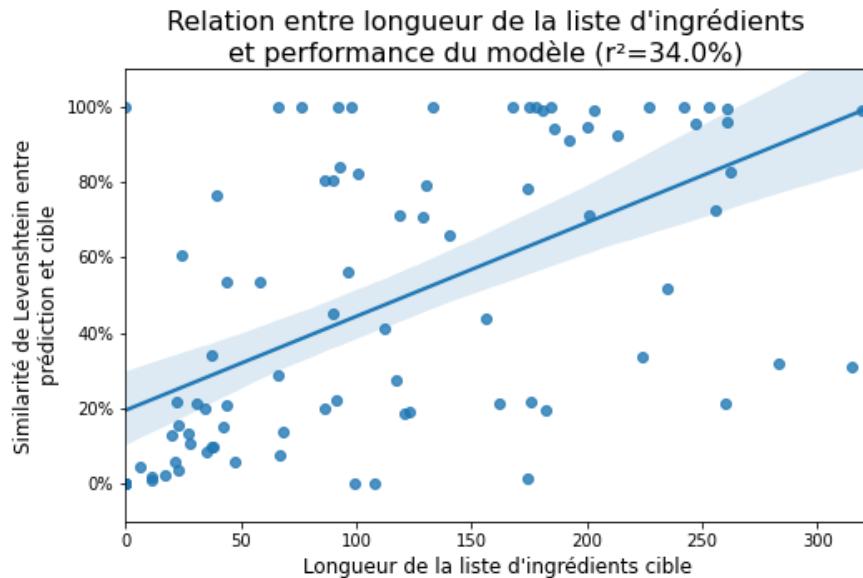


FIGURE 16 – Performance du modèle en fonction de la longueur des listes d'ingrédients

Chapitre 8

AJUSTEMENT DES PARAMÈTRES

8.1 Description des paramètres ajustables

Les paramètres ajustables sur le modèle présenté ci-dessus sont multiples (text-preprocessing, découpage des textes en blocs, calcul de similarité, ...). Maintenant qu'un pipeline équipé d'une fonctionnalité de mesure de la performance est disponible, on va pouvoir faire varier les paramètres et évaluer l'impact sur la performance du modèle.

8.1.1 Text-preprocessing

L'étape de text-preprocessing vise à nettoyer et standardiser le contenu des textes. Elle est appliquée aux textes avant entraînement et avant prédiction. Il est nécessaire de bien appliquer les mêmes règles à ces deux étapes. Parmi les fonctionnalités classiques de texte préprocessing, on va procéder de la manière suivante :

Mise en minuscule : On appliquera systématiquement la mise en minuscule. La casse est plutôt un facteur qui va pénaliser l'interprétation du texte, en contribuant à la création artificielle de mots synonymes.

Retrait des accents : L'impact du retrait des accents sera mesuré sur la performance du modèle. En effet, même s'il peut amener à la création d'homonymes, sur des textes en français inclus dans des pdf et issus de multiples sources cette fonctionnalité peut avoir une influence positive.

Gestion des stopwords : On vérifiera l'impact de retirer les stopwords (mots fréquents peu porteurs de sens) de l'ensemble des textes (listes d'ingrédients cible, contenu des documents) avant entraînement ou prédiction. On se basera sur une liste déterminée à partir des mots les plus fréquents dans le corpus des documents, qui ne portent pas de signification. Il est pressenti que ces stopwords risquent d'avoir un impact important sur le résultat de l'algorithme, en ajoutant dans l'ensemble des blocs candidats un grand nombre de mots étant considérés comme des ingrédients.

8.1.2 Découpage du texte des documents en blocs

On peut envisager d'autres méthodes de découpage des textes complets des documents en blocs candidats. Il avait été vu lors du prototypage que cette fonction pouvait influer fortement sur la qualité du résultat (il est inutile de chercher une bonne solution pour trouver le bon candidat si on détermine mal la liste de candidats). On éprouvera les techniques suivantes :

- comme en prototypage, en coupant dès que deux retours à la ligne successifs sont identifiés
- à chaque retour à la ligne

- via une expression régulière, on coupera dès qu'on trouve un ensemble de whitespaces (espaces, retour à la ligne, tabulation, ...) contenant au moins 2 retours à la ligne

8.1.3 Vectorisation des textes

Bag Of Words : Les textes seront représentées sous forme de vecteurs via l'approche « Bag Of Words ».

On évaluera les méthodes suivantes pour la transformation des textes en vecteurs :

- de manière binaire : la coordonnée est à 1 si le mot est présent au moins une fois dans le texte, 0 sinon
- en term-frequency (tf) : la coordonnée correspond à la fréquence d'apparition du mot dans le texte
- en tf-idf, l'inverse document frequency pouvant être calculée en référence au corpus du contenu des documents
- via une fonction de score absolu, tel que décrit dans la partie sur l'analyse des données, à la section 5.2 page 22
- via une fonction de score relatif (décrit dans la même section)

n-grams : On testera également la possibilité de prendre en compte des n-grams de mots (mots successifs) et les résultats que cela amène (à la fois dans l'entraînement et dans la prédiction).

Embeddings des mots : Enfin, on procédera aux calculs d'embeddings de mots, en appliquant les méthodes suivantes :

- truncated SVD [16] : on approxime la matrice des textes vectorisés en calculant ses valeurs singulières et en n'en gardant que les 100 plus importantes (valeur choisie arbitrairement). Les embeddings des mots sont récupérable directements depuis le transformateur TruncatedSVD mis à disposition dans la bibliothèque scikit-learn.
- Word2Vec [21] : cette méthode de calcul des embeddings se base sur l'entraînement d'un réseau de neurones pour la prédiction d'un mot à partir des autres mots de son contexte (Continuous Bag Of Words, CBOW), ou de prédire le contexte à partir d'un mot (skip-gram). Les embeddings sont les poids de la couche intermédiaire du réseau de neurones après entraînement. On utilisera l'algorithme CBOW, avec un nombre de features de 100.

Cette manière de représenter les textes et les mots se traduit par une *réduction de la dimensionnalité*, qui passe du nombre de mots dans le vocabulaire (plus de 10 000 dans notre cas) au nombre de features conservées (on en gardera 100).

8.1.4 Calcul de similarité

Similarité cosinus

Cette méthode nécessite de représenter le vocabulaire des ingrédients sous la forme d'un vecteur, en faisant la moyenne des vecteurs de textes obtenus lors de la vectorisation des listes d'ingrédients. On calcule le cosinus de l'angle entre le vecteur du vocabulaire et celui du candidat. Plus ce cosinus est élevé, plus la

similarité est importante. Cette similarité n'a pas de paramètre, une fois que les vecteurs ont été calculés. Il est possible d'appliquer ce calcul de similarité y compris après application d'un embedding des vecteurs de texte (ex : Word2Vec, tSVD, ...). Une illustration de la similarité cosinus est proposée à la FIGURE 17 page 52. Enfin, ce mode de calcul de la similarité permet d'ajuster la direction du vecteur cible en fonction de divers critères (fonctions de scoring spécifiques, cf. section 8.1.3 page 50 sur la vectorisation des textes). La similarité cosinus est toujours comprise entre 0 (vecteurs orthogonaux) et 1 (vecteurs colinéaires), sauf dans le cas où on applique des coefficients négatifs à certains mots (cas de la fonction de scoring relatif, où des poids sont négatifs).

Par projection

Cette méthode nécessite uniquement de projeter le candidat sur le sous-espace généré par les mots du vocabulaire des ingrédients. On fait ensuite le rapport entre la norme du vecteur initial, à celle de sa projection. On peut envisager cette similarité un peu comme un calcul de l'angle entre le vecteur candidat et le sous-espace cible. Attention, si on utilise la même norme pour mesurer le vecteur initial et sa projection, on choisira toujours les candidats dont aucun mot n'est hors du vocabulaire (cf. l'illustration présentée à la FIGURE 18 page 53). Une manière de s'affranchir de ceci est choisir pour mesurer le vecteur initial une norme L_p et pour sa projection une norme L_q avec $p > q$. En effet, un résultat d'algèbre linéaire [10] montre que :

$$\forall x \in \mathbb{R}^n, p > q \Rightarrow \|x\|_p \leq \|x\|_q$$

Pour rappel, la norme L_p d'un vecteur x est définie par :

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

où les x_i sont les coordonnées de x . Ce calcul de similarité ne peut se faire que dans l'espace initial, où les vecteurs représentant les textes peuvent être projetés sur un espace généré par des mots. Si une réduction de la dimensionnalité a eu lieu, par exemple via le calcul d'embeddings, il n'est pas possible de déterminer un sous-espace généré par les mots du vocabulaire des ingrédients. De plus, contrairement au calcul de similarité cosinus, il n'est pas possible de pondérer les différents mots du vocabulaire des ingrédients (via un calcul de score spécifique, un tf-idf, ...). Ce mode de calcul est paramétrique, c'est à dire qu'il faut définir la norme de l'espace de départ et la norme dans le sous-espace de projection. Contrairement à la similarité cosinus, la similarité projection peut être supérieure à 1.

Le mode de calcul retenu dans les prototypes décrits précédemment (chapitre 6 page 30) est un calcul de similarité par projection utilisant les normes L_2 sur l'espace initial et L_1 sur le sous-espace de projection. La

formule de calcul était :

$$\frac{\text{Nombre de mots du candidat qui sont des ingrédients}}{\text{Norme euclidienne du vecteur du candidat}} = \frac{\|\text{Vecteur projeté}\|_1}{\|\text{Vecteur initial}\|_2}$$

où le vecteur initial est la matrice des textes avec les comptes de mots.

Illustration de ces calculs de similarité

On illustre ces deux manières de mesurer la similarité en prenant l'exemple suivant. Le vocabulaire des ingrédients est constitué des mots « eau » et « sucre », et le mot « eau » est 2 fois plus représenté que le mot « sucre ». Les textes candidats sont « sucre eau sucre » et « commercial commercial sucre ». Ces deux exemples sont présentés en FIGURE 17 page 52 et FIGURE 18 page 53. Sur ces figures, la similarité correspond aux angles matérialisés en vert. Si on récapitule les valeurs respectives des similarités, on obtient les résultats présentés à la TABLE 15 page 52

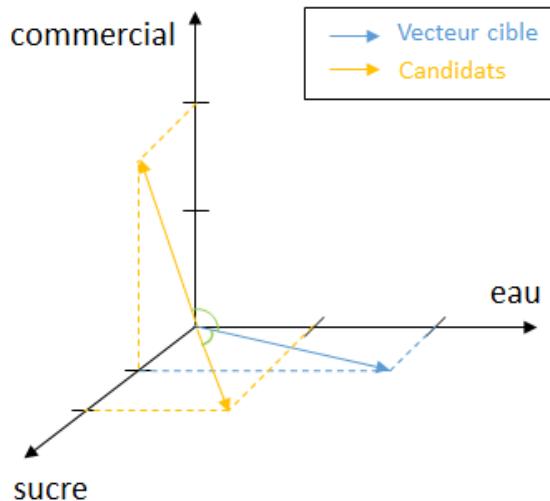


FIGURE 17 – Illustration de la similarité cosinus

Candidats	Cosinus	Projection L_1/L_1	Projection L_2/L_1	Projection L_3/L_1
sucre eau sucre	0,8	1,	1,34	1,44
commercial commercial sucre	0,2	0,33	0,45	0,48

TABLE 15 – Exemples de calculs de similarité

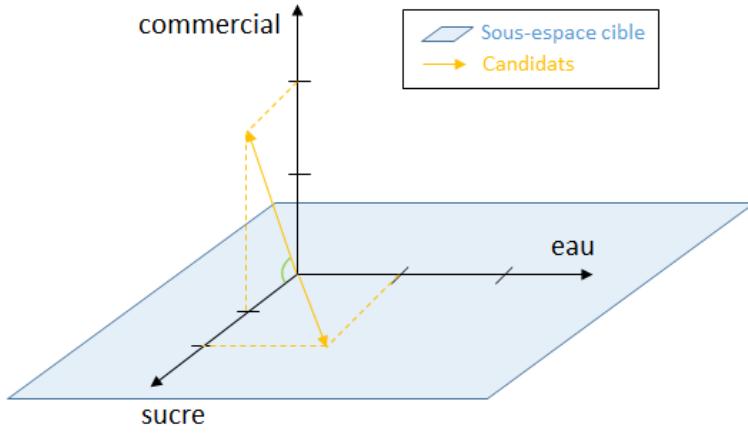


FIGURE 18 – Illustration de la similarité projection

8.2 Ajustements et résultats

On peut, dans l’optique d’améliorer la performance du modèle, ajuster certains paramètres et d’évaluer l’impact via une grid search. Pour l’ensemble des ajustements, on travaillera uniquement sur le set d’entraînement (400 échantillons) afin de se prémunir des effets de data leaking (qui amènerait à surestimer la performance lors de l’évaluation finale). On travaillera avec 8 folds de cross-validation, afin de rester sur des ordres de grandeurs comparables à ce qui avait été fait lors de l’évaluation du second prototype (10 folds sur 500 échantillons). Enfin, on effectuera une évaluation finale sur le set de test, après entraînement du modèle avec les meilleures performances sur le set d’entraînement. La sélection des meilleurs paramètres se fera sur la base de la similarité de Levenshtein (cf. chapitre 7 page 41), mais on illustrera également la performance du modèle retenu via l’accuracy avec text-postprocessing. L’ensemble des résultats présentés à cette section sont issus du notebook « Tuning du modèle » présenté en annexe G.6 page 190.

8.2.1 Ajustement du text-preprocessing

Les résultats de la grid search sur le text-preprocessing sont présentés à la FIGURE 19 page 55. Dans l’ensemble des représentations de ce chapitre, chaque point du swarmplot représente une cross validation de l’ensemble du set de test. Les folds individuels ne sont pas représentés, l’écart type pour un jeu de paramètres donné n’est donc pas représenté (le détail est présenté dans le notebook mentionné précédemment). Dans ce premier run, les paramètres explorés sont :

gestion des accents : conservation ou retrait des accents sur les textes

gestion des stopwords : conservation ou retrait des stopwords ('pas', 'le', 'en', 'pour', 'ou', 'ce', 'de', 'dans', 'du', 'and', 'un', 'sur', 'et', 'of', 'est', 'par', 'la', 'les', 'dont', 'au', 'des', 'que') des textes

constitution des ngrams : seulement les monogrammes, puis monogrammes et bigrammes, puis mo-

nogrammes et bigrammes et trigrammes

découpage des textes en blocs : comparatif des trois fonction présentées précédemment à la section 8.1.2 page 49

fonctions de similarité : identification du meilleur candidat par similarité cosinus, ou par projection L_2/L_1

soit un total de 72 jeux de paramètres testés. Il apparaît que :

- le retrait des stopwords a un impact fortement positif sur la performance de l'algorithme
- le retrait des accents a un impact positif mais relativement marginal
- l'écart-type sur la performance (similarité de Levenshtein) est très élevé au sein d'un jeu de paramètre testé : il se situe systématiquement aux alentours de 5-6% pour une valeur moyenne autour de 50%

8.2.2 Tuning du découpage du texte des documents

On se sert des résultats de la grid search présentée précédemment. Les résultats sont présentés à la FIGURE 20 page 56. On en déduit que la fonction qui présente les meilleurs résultats est la troisième fonction (via une expression régulière avec 2 retours à la ligne). L'écart n'est pas énorme avec la fonction 1 (1 à 2% sur l'accuracy moyenne), mais il est présent de manière consistante.

8.2.3 Comparatif des fonctions de similarité

Dans une seconde grid search, on va comparer les similarité cosinus et projection, en faisant varier les paramètres de la similarité projection (normes de l'espace complet et du sous-espace de projection). Les résultats sont présentés à la FIGURE 21 page 58. Ils sont issus d'une grid search lors de laquelle on a fixé :

- la tokenisation avec des monogrammes et des bigrammes
- le retrait des accents
- le retrait des stopwords
- la fonction de découpage en blocs numéro 3 (avec l'expression régulière)

et on a fait varier :

- les fonctions de similarité, cosinus ou projection
- les normes utilisées, pour les fonctions de projection
- la construction de la matrice des textes entre les comptes de mots, ou l'identification uniquement de la présence / absence d'un terme (« binary Bag Of Word »)

Dans la représentation qui en a été faite, on a regroupé ensemble les similarité projection en fonction de l'écart de valeur entre les normes dans l'espace initial et le sous-espace de projection (ex : pour la similarité projection L_3/L_1 cet écart vaut 2) ; En peut en tirer des conclusions intéressantes :

- les écarts-types au sein d'une cross-validation sont toujours importants (5-6% environ, sur la similarité de Levenshtein)

Comparaison des fonctions de preprocessing

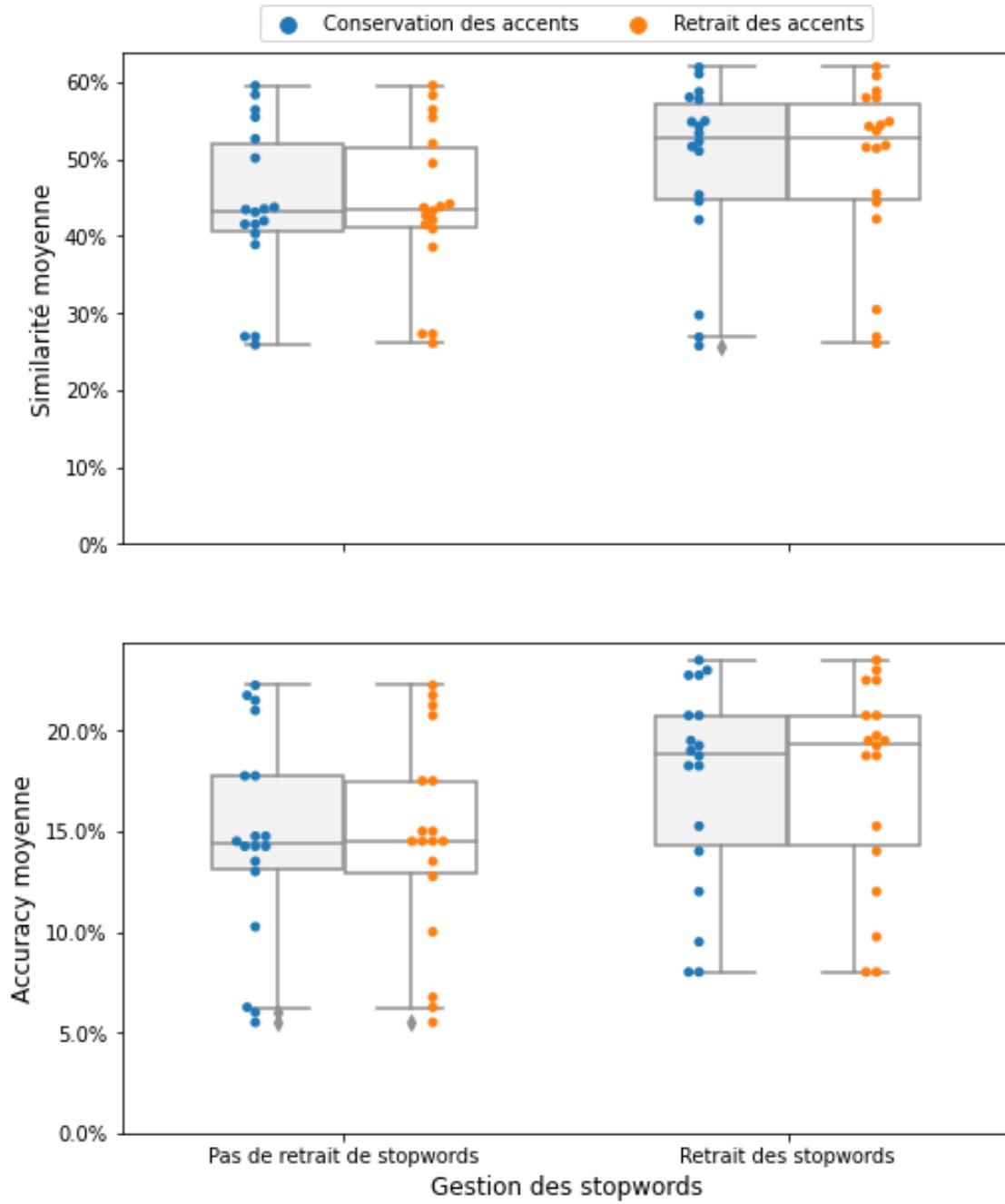


FIGURE 19 – Résultats du tuning du preprocessing

Comparaison des fonctions de découpage

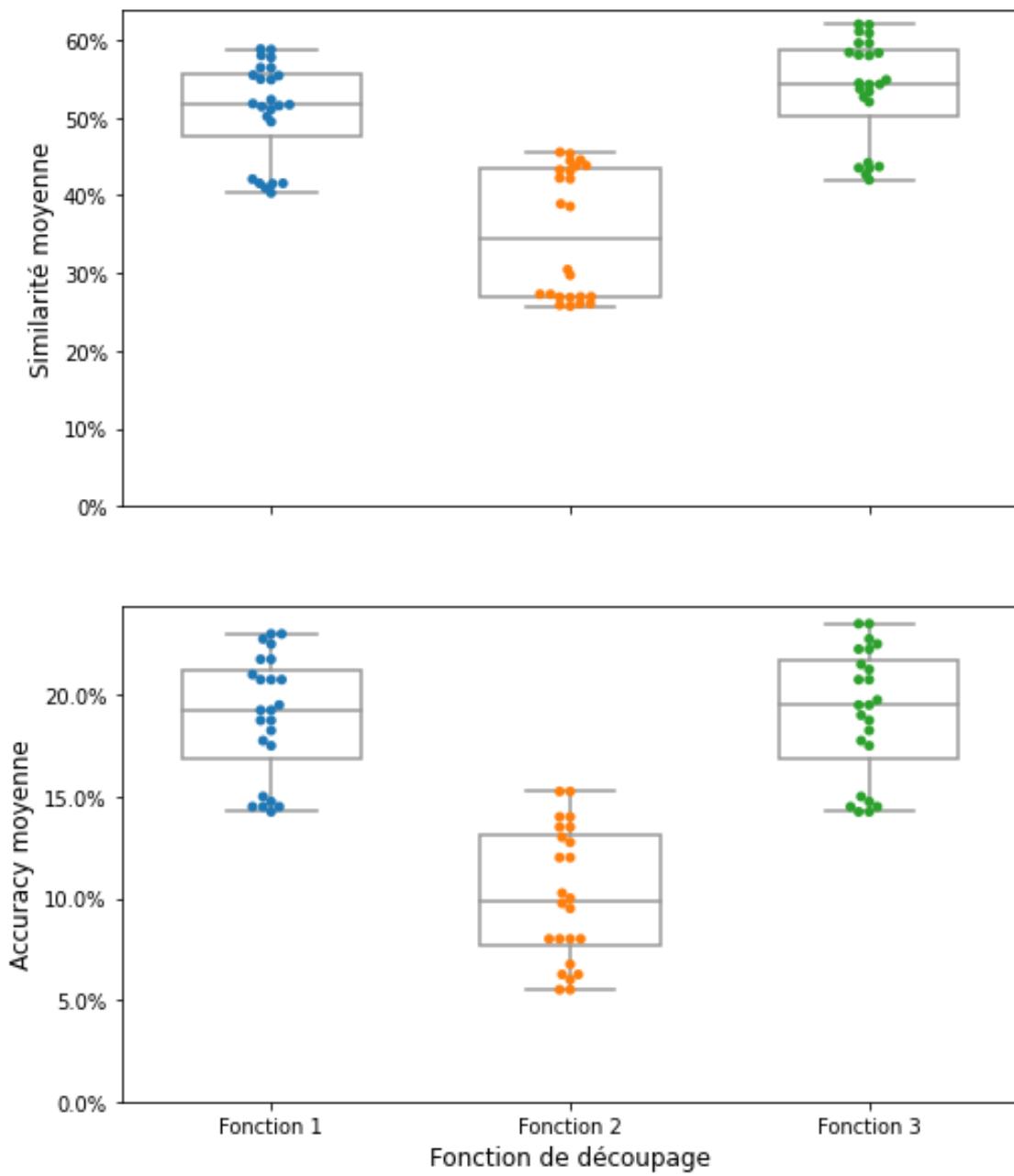


FIGURE 20 – Résultats du tuning du découpage des textes des documents

- la similarité par projection surperforme nettement la similarité cosinus
- comme on pouvait s'y attendre, les performances sont catastrophiques si on applique pour la similarité projection la même norme dans l'espace de départ et le sous-espace de projection (on choisira toujours un texte qui n'a aucun mot hors du vocabulaire des ingrédients, peu importe sa longueur)
- des tendances régulières se dessinent parmi les groupes avec un delta constant, bien visibles sur l'indicateur de performance en similarité :
 - les performances semblent suivre deux tendances polynomiales avec des coefficient d'ordre maximal négatif, différentes selon que la tokenisation est binaire ou non
 - la tokenisation binaire est systématiquement meilleure que la tokenisation par comptes
 - on atteint un maximum de performance aux alentours des projections L_4/L_3 , L_6/L_4 , L_8/L_5

8.2.4 Comparaison des méthodes de vectorisation

Inverse document frequency

On peut apprécier l'impact de la prise en compte de l'inverse document frequency lors de la vectorisation, sur la performance du modèle. Cette inverse document frequency se calcule par rapport à la présence des termes dans le corpus des textes des documents. Les résultats sont présentés à la FIGURE 22 page 59, sur la base d'une grid search faisant varier :

- le type de similarité : cosinus ou projection L_4/L_3
- l'utilisation ou non de l'inverse document frequency
- la prise en compte de ngrams, avec des ngrams de longueur de 1 à 5

L'impact est très différent selon qu'on identifie le meilleur candidat via le cosinus, plutôt que via la projection. La qualité des prédictions s'améliore en utilisant l'inverse document frequency couplée à la similarité cosinus, mais est très fortement dégradée dans le cas de la projection.

Prise en compte des n-grams

La vectorisation des textes peut également être ajustée, en ajoutant en tant que features des ngrams de longueur variable. L'impact de la prise en compte de ngrams de plus en plus long (de 1 à 5 mots) est illustré à la FIGURE 23 page 60.

On peut constater que :

- la prise en compte de ces ngrams a des effets mitigés dans le cas du cosinus
- elle est par contre bénéfique pour la projection, avec une saturation à partir des trigrammes

Fonctions de scoring spécifique

Les fonctions de scoring « absolu » et « relatif » ont été présentée dans le chapitre sur l'analyse des données, à la section 5.2 page 22. Elles permettent de calculer un score pour chaque mot, afin de quantifier

Comparaison des méthodes d'identification du meilleur candidat

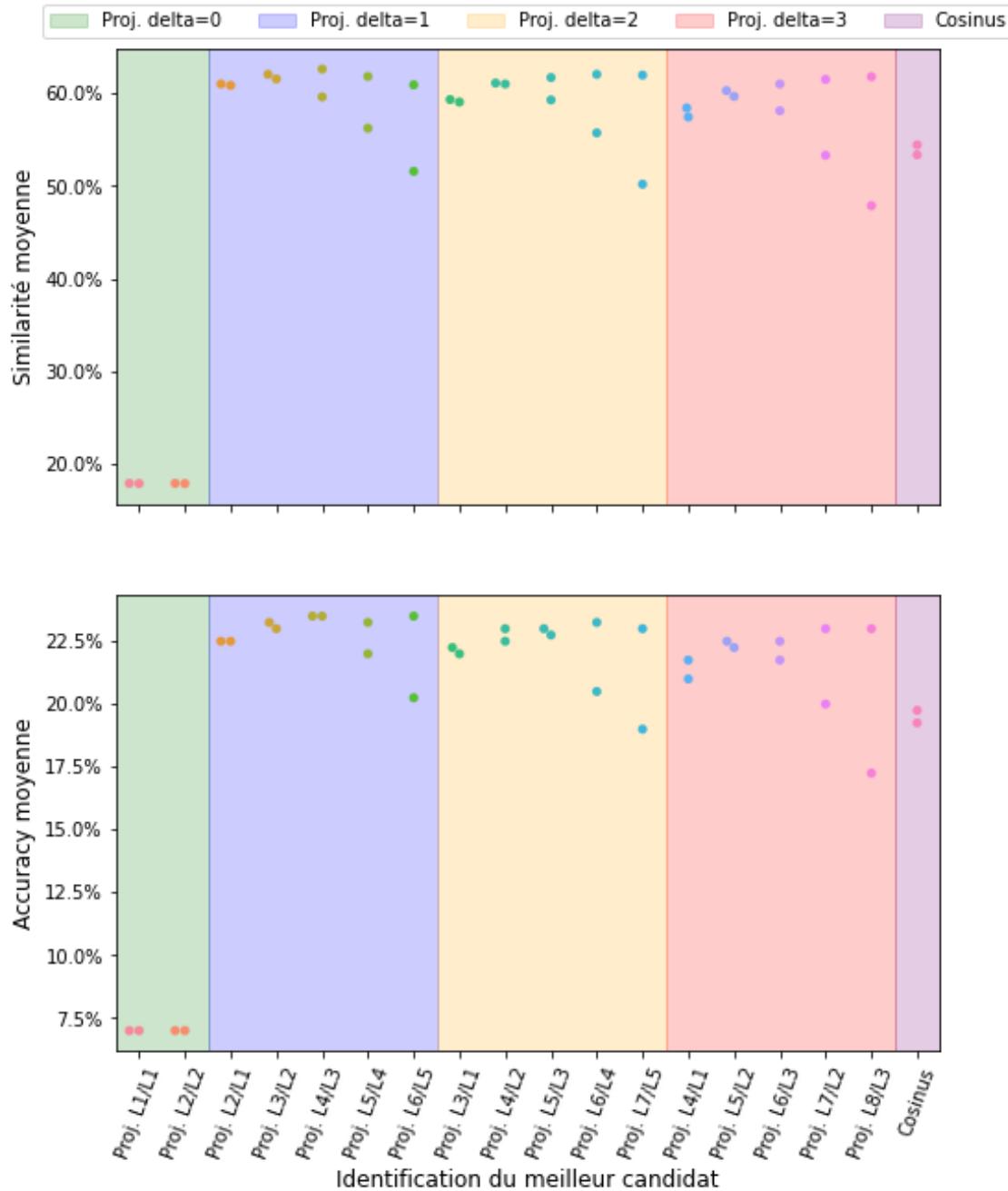


FIGURE 21 – Résultats du tuning des fonctions de similarité

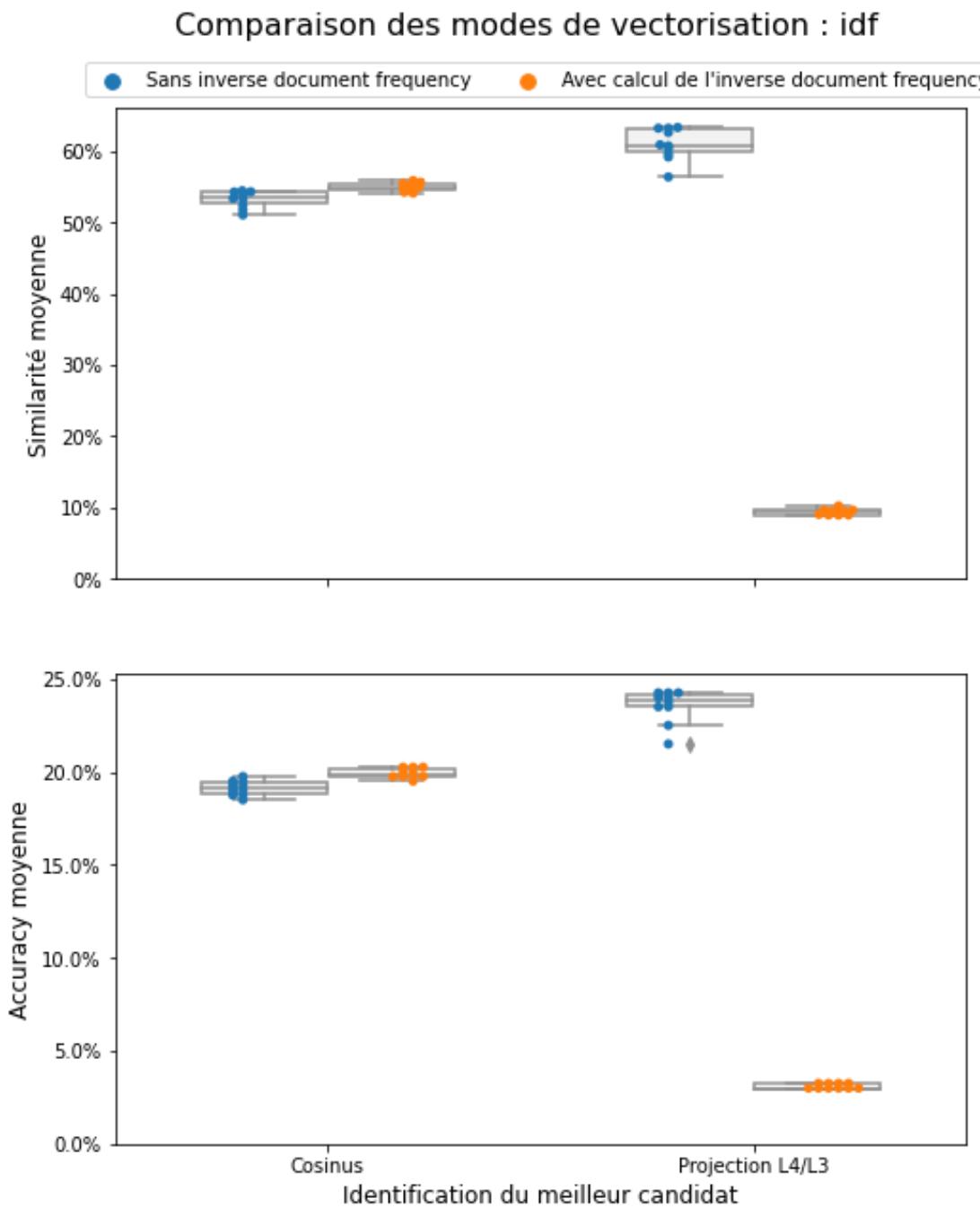


FIGURE 22 – Résultats du tuning de l'utilisation de l'inverse document frequency

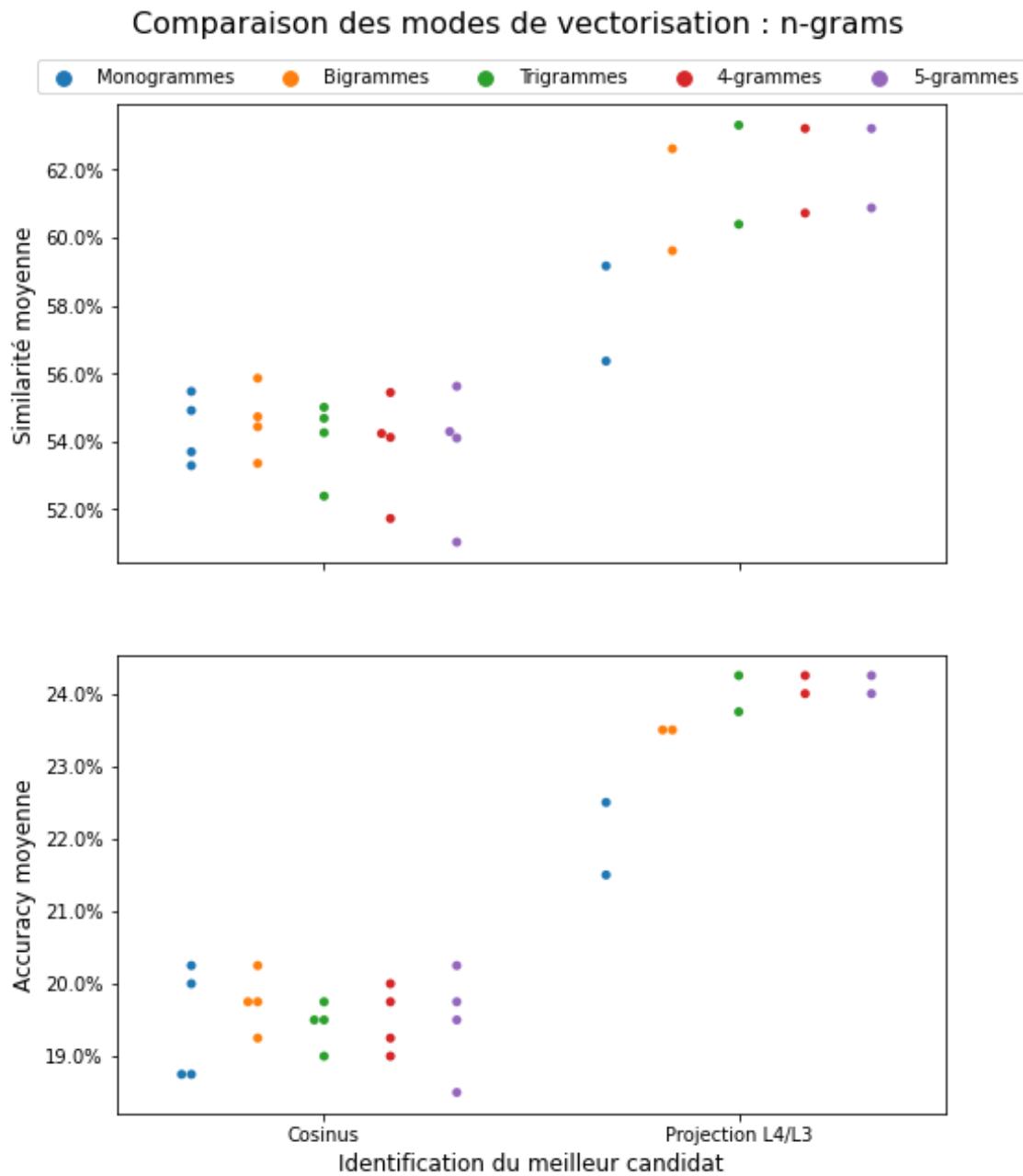


FIGURE 23 – Résultats du tuning de l'utilisation de l'inverse document frequency

son « affinité » pour les listes d'ingrédients. L'utilisation d'un score relatif, qui peut être négatif (i.e. le mot concerné est plus souvent présent dans le corps du texte des documents que dans les listes d'ingrédients), et donc pénaliser le candidat. C'est par exemple le cas du mot « sel », qui est évidemment un ingrédient mais qui est également très représenté par ailleurs dans le contenu des fiches techniques. L'utilisation de ces scores ne se peut se faire que pour la similarité cosinus, de la même manière qu'ilustré à la FIGURE 24 page 61 (qui montre la « négativité » du score du mot « sel »). Il est même possible que la similarité du candidat avec le vecteur cible soit négative.

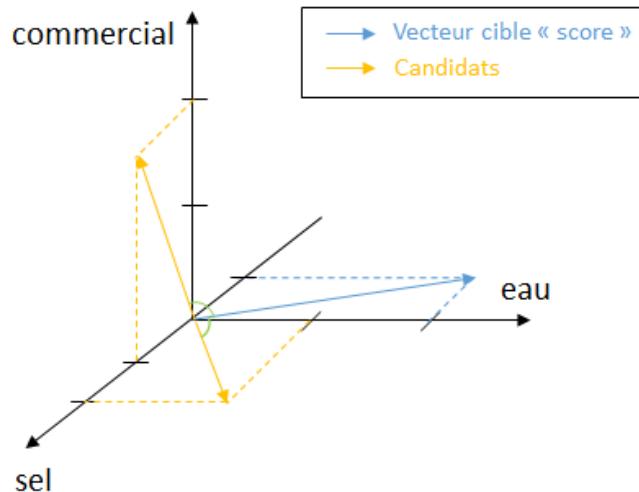


FIGURE 24 – Illustration de la similarité cosinus avec score relatif

L'impact des fonctions de scoring a été illustré sur la base d'une grid search s'appliquant uniquement au mode d'identification cosinus, sans n-grams, avec retrait des stopwords et des accents. Elle fait varier :

- la comptabilisation des mots de manière binaire ou via les comptes
- la détermination ou non de l'inverse document frequency
- les scores : sans, avec le score absolu ou avec le score relatif
- les embeddings de mots : sans, Word2Vec, truncated SVD

Les résultats sont présentés à la FIGURE 25 page 62. On constate que :

- l'utilisation du score relatif dégrade très fortement la performance du modèle
- l'utilisation de l'idf avec le score absolu dégrade légèrement la performance
- le seul impact positif à l'utilisation d'une fonction de scoring est que score absolu améliore marginalement la performance en l'absence d'idf

La cause de la très faible performance du score relatif semble être la suivante : comme illustré précédemment à la FIGURE 6 page 24, du fait de l'écart entre la taille des vocabulaires ingrédients et documents, les scores relatifs sont en moyenne négatifs. Par conséquent, plus ils sont longs, plus les vecteurs candidats ont en moyenne un produit scalaire négatif avec le vecteur cible. Ce qui signifie que le choix du meilleur candidat

revient à prendre en compte le vecteur le moins négatif, voire un bloc de texte vide ou constitué de mots inconnus à l'entraînement...

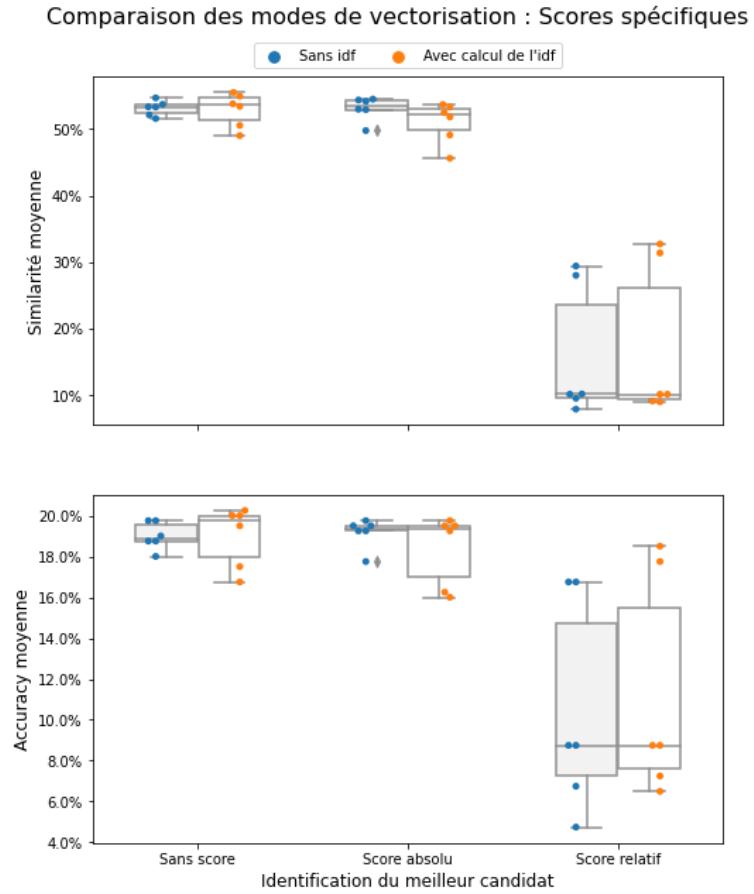


FIGURE 25 – Illustration de la similarité cosinus avec score relatif

8.2.5 Utilisation d'embeddings

L'utilisation d'embeddings a été évaluée via la même grid search que pour l'évaluation du scoring. Les résultats sont illustrés à la FIGURE 26 page 63. On constate que :

- les embeddings issus de la truncated SVD sous-performent par rapport aux autres méthodes (sans embeddings ou Word2vec)
- l'utilisation de Word2vec n'augmente globalement pas la performance du modèle

L'utilisation de ces embeddings, entraînés sur notre jeu de données uniquement, n'est pas concluant.

8.3 Évaluation finale de la performance

On peut tirer les enseignements suivants des travaux menés :

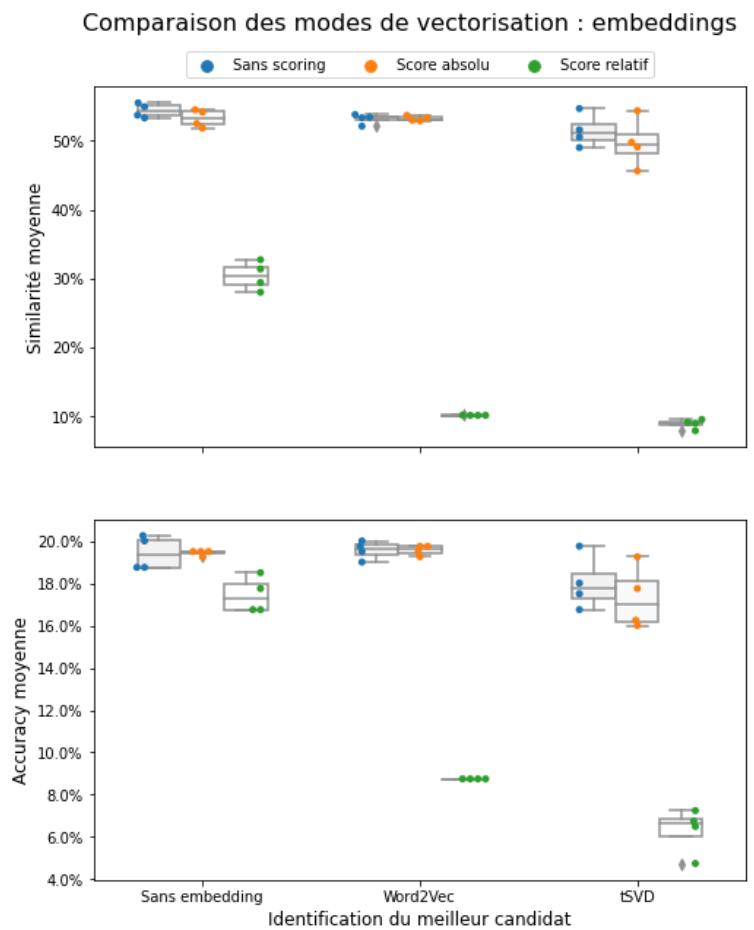


FIGURE 26 – Illustration de la similarité cosinus avec score relatif

- des gains rapides ont été obtenus en travaillant sur le preprocessing de base : stopwords, gestion des accents, découpage en blocs
- la similarité projection sur-performe nettement la similarité cosinus
- l'utilisations d'algorithmes plus complexes (embeddings et scoring) ne suffit pas à compenser cet écart de performance

Les paramètres ayant obtenu la meilleure performance ($63.31\% \pm 3.83\%$) sur la grid search sont :

- similarité projection, avec les normes L_4/L_3
- retrait des stopwords, des accents
- découpage en blocs via la fonction 3 (avec expression régulière)
- vectorisation binaire des textes, sans appliquer l'inverse document frequency
- prise en compte des monogrammes, bigrammes et trigrammes

Si on entraîne un modèle avec ces paramètres sur le set d'entraînement, et qu'on évalue sa performance sur le set de test (qui n'a pas été utilisé lors du tuning des paramètres), on obtient les résultats suivants :

- Similarité de Levenshtein moyenne : 67.18%
- Accuracy postprocessée : 27%

Ces résultats sont globalement bons, et illustrés en annexe à la TABLE 30 page 135. Pour rappel, la performance initiale d'un modèle avant ajustement des paramètres était de 48.86% (cf. TABLE 13 page 47). Il apparaît que les principales améliorations passeraient par la mise en place de fonctions de découpage plus précises, de fonctionnalités de text postprocessing pour éliminer les préfixes, et d'OCR pour prendre en compte les documents numérisés.

Quatrième partie

TRAVAUX SUBSÉQUENTS

Chapitre 9 EXTENSION DES FONCTIONNALITÉS OFFERTES

9.1 Amélioration de la performance du modèle

Même si les résultats se montrent désormais très encourageants, il reste encore quelques améliorations à apporter pour que les utilisateurs y trouvent un réel intérêt. Avec une accuracy à 27%, il y a encore trop de corrections manuelles à apporter. Pour chacune des pistes d'amélioration identifiées, il faudra évidemment procéder à une vérification de l'impact via une cross validation sur le set d'entraînement.

9.1.1 Découpage du contenu des documents en blocs de textes

L'essentiel des erreurs vient de soucis sur le découpage du texte en blocs. Cela dégrade à la fois la performance en termes de similarité et d'accuracy, mais pénalise également la fonctionnalité d'identification du meilleur candidat. Plusieurs pistes d'amélioration sont offertes pour améliorer ce point :

- analyser les cas d'erreurs, et complexifier un peu l'expression régulière servant à découper le texte des documents en blocs
- multiplier les candidats pour une même fiche technique : on pourrait appliquer plusieurs fonctions de découpage différentes sur le document, et proposer l'ensemble des candidats obtenus au sélecteur
- dans la même logique, la constitution de « ngrams de blocs » pourrait compenser les cas de mauvais découpages car trop fins, et multiplierait également le nombre de candidats proposés au sélecteur
- enfin, il faudrait se pencher sur les fonctionnalités de clustering des objets (caractères en mots, mots en lignes, lignes en paragraphes) proposées par les bibliothèques de mining de pdf. Les blocs candidats pourraient correspondre aux paragraphes identifiés par l'outil de mining.

De plus, il apparaît qu'une partie des écarts se situent en début ou en fin de texte (préfixes conservés alors qu'il aurait fallu les supprimer, mentions connexes non supprimées, ...) pourrait être corrigés s'il était possible de « tailler » le début et la fin du texte. On pourrait là encore procéder par apprentissage en déterminant :

- une liste de mentions qui améliorent la performance lorsqu'elles sont retirées
- d'autres critères (document frequency dans les textes des fiches techniques, document frequency dans les listes d'ingrédients, ...)

9.1.2 Amélioration de l'identification par similarité

Dans les résultats sur le set de test, il reste encore des blocs incorrectement identifiés alors que la liste d'ingrédients était bien présente par ailleurs. Même si l'écart entre similarité cosinus et projection est important, on suspecte qu'il existe des manières plus performantes d'ajuster la manière de représenter les textes. En particulier, l'utilisation du scoring relatif (cf. section 5.2 page 22) semblait prometteuse. Peut-être que tester une version où les scores relatifs sont calculés pour ne pas décaler les produits scalaires des candidats vers les valeurs négatives en moyenne pourrait être pertinente.

Une autre piste serait de faire tourner en parallèles plusieurs sélecteurs de blocs, et d'appliquer une méthode ensembliste en les faisant voter pour le meilleur candidat. La performance du modèle est bonne sur les produits dont les listes d'ingrédients sont relativement longues, mais moins pertinente pour les produits plus bruts. L'identification d'un modèle performant sur les listes d'ingrédients courtes permettrait de pallier ce point et il faudrait l'associer au modèle actuel.

9.1.3 Filtrage des similarités trop basses

Un irritant majeur pour les utilisateurs est d'avoir un résultat complètement impertinent. Il serait intéressant d'identifier les cas où le modèle a retourné un candidat qui ne correspond pas à une liste d'ingrédients. On pourrait ensuite filtrer la proposition, et afficher à l'utilisateur que le traitement a échoué et que le modèle n'a pas réussi à extraire la liste d'ingrédients du document soumis. Une telle fonctionnalité pourrait également améliorer la performance du modèle, en couvrant les cas où le document fourni ne comporte pas de liste d'ingrédients. Cela pourrait se faire :

- en fixant un seuil minimum de similarité (cosinus et/ou projection)
- en utilisant d'autres critères (longueur du candidat, document frequency moyenne des mots qui le composent, ...)

9.2 Extension du périmètre couvert avec la méthode « textuelle »

Avant de pouvoir mettre en production des fonctionnalités de types extraction de connaissance depuis des documents, il serait intéressant d'en étoffer le périmètre. L'approche présentée dans ce document consiste à interpréter uniquement le texte brut, sans avoir recours à des fonctionnalités de spatialisation au sein du document.

9.2.1 Prise en compte de nouveaux types de pièces jointes

Les étiquettes produit doivent porter l'ensemble des informations réglementaires (données nutritionnelles, composition, allergènes, ..., cf. section 2.2.2 page 11). Même si les étiquettes ont plus tendance à être des documents pdf dont le contenu textuel n'est pas extractible sans faire appel à des fonctionnalités d'OCR (cf. TABLE 5 page 17), un potentiel accessible existe déjà. La performance de l'identification des listes d'ingrédients devrait être d'ailleurs assez bonne sur les étiquettes, dans la mesure où il s'agit de documents bien moins verbeux.

9.2.2 Utilisation d'outil d'OCR pour les pdf non structurés

L'utilisation d'outils d'OCR pourrait venir compléter les fonctionnalités de pdfminer.six, en permettant l'extraction des textes depuis les documents non structurés. La suite du traitement pourrait rester parfaitement identique, en traitant uniquement le contenu textuel extrait, sans notion de spatialisation. De plus, cela complémenterait parfaitement l'extension des fonctionnalités aux étiquettes produits.

9.2.3 Évaluation de la performances sur d'autres familles de produits

Le modèle a été construit sur la base des données de la branche ÉpiSaveurs (cf. la description des branches du groupe, en annexe B.1.2 page 79). Cette branche ne commercialise que des produits stockés à température ambiante (pas de produit frais ou surgelés). Il serait intéressant de tester ce modèle sur les données des autres branches du groupe, dans l'optique de vérifier que les fonctionnalités pourraient leur être étendues. Un point à vérifier sur cette extension serait si la performance serait meilleure en construisant des modèles séparés, en fonction de la famille de produit, ou un modèle général qui tournerait sur l'ensemble des données.

Les produits de chimie (cf. annexe C.1 page 98), commercialisés par la branche ÉpiSaveurs, font également l'objet d'une déclaration de composition obligatoire sur l'emballage. On pourrait envisager de faire la même évaluation du modèle sur ces produits. Comme les « ingrédients » des produits de chimie sont différents de ceux des produits alimentaires, il faudrait faire un entraînement particulier sur leurs données.

9.2.4 Récupération des dénominations réglementaires

Il est un autre texte qui est normalement présent sur l'emballage et la fiche technique du produit : la dénomination réglementaire (cf. annexe D.2.1 page 111). Il est envisageable de constituer un modèle qui aurait en charge de récupérer cette information depuis les fiches techniques, ou les étiquettes produit. Comme ces désignations sont en général plus courtes que les liste d'ingrédients, il faudrait prévoir d'ajuster les paramètres du modèle, avec une méthode similaire à ce qui a été fait et présenté au chapitre 8 page 49 « Ajustement des paramètres ».

9.3 Changement de méthode

Dans le cadre d'un use case en lien avec l'extraction de connaissance sur les produits depuis des documents pdf, on pourrait également envisager d'autres méthodes. Cela rendrait accessible un grand nombre d'autres fonctionnalités, voire d'autres cas d'usage, en lien ou non avec la gestion de l'information produit.

9.3.1 Mise en place d'outil de spatialisation des textes

C'est clairement le sujet le plus prometteur à tester, mais également un des plus complexes à mettre en œuvre. L'idée derrière la spatialisation est de réussir à associer les informations présentes dans les documents entre elles, en fonction de leurs position dans le document. Il s'agit clairement d'un tout nouveau sujet, on passerait d'une problématique de traitement du langage à du traitement d'image. Le sujet est vaste, et il faudrait être en mesure :

- de détecter les tableaux présents dans les documents (voir par exemple la fiche technique des poivrons en annexe E.1.6 page 121) :
- puis de détecter les entêtes de lignes ou de colonnes, pour avoir une représentation à simple ou double entrée de l'information
- de construire les relations entre les niveaux hiérarchiques d'indexation dans ces tableaux (ex : Valeurs nutritionnelles > Énergie = 109 kJ)
- de comprendre la structure du document, et les liens entre les titres entre les contenus des différentes sections (cf. la fiche technique de la panna cotta, en annexe E.1.3 page 115)

Il s'agit d'un sujet ambitieux, mais qui aurait en plus l'avantage de pouvoir être étendu à la quasi totalité des cas métier où des documents sont interprétés. Dans le cas précis de la gestion de l'information produit, cela permettrait d'aller récupérer les données nutritionnelles par exemple (cf. annexe B.2.1 page 85).

9.3.2 Aides pour le contrôle de cohérence

Toujours dans l'optique d'améliorer la qualité des données des produits, on pourrait changer de sujet et passer à des contrôles de cohérence entre les données elles-mêmes. Par exemple, il est normalement possible

pour un utilisateur correctement formé de déterminer l'ensemble des allergènes du produit à la lecture de la liste d'ingrédients (sous réserve qu'elle mentionne bien également les contaminations croisées). On pourrait envisager de déterminer quels mots ou expressions parmi les listes d'ingrédients permettent d'identifier tel ou tel allergène réglementaire (cf. annexe B.2.1 page 84 pour plus de détail sur les allergènes).

Il serait également possible de déterminer, par des méthodes de classification, si un produit se qualifie pour une allégation nutritionnelle quelconque (ex : faible en sel), sur la base de ses données nutritionnelles, afin de contrôler si une allégation qui a été transmise par le fournisseur est cohérente.

Chapitre 10

DÉPLOIEMENT OPÉRATIONNEL DE

L'OUTIL

10.1 Mise en place d'une organisation projet

10.1.1 Sponsor et Client

La toute première étape pour le lancement d'un projet visant à mettre en œuvre ce modèle, serait d'identifier un sponsor capable de mobiliser une enveloppe budgétaire, et de prendre les décisions de niveau stratégique. Ensuite, il faudrait identifier quelques représentants métier (gestionnaires de référentiel, ingénieurs qualité, acheteurs, ...) qui seraient capable de se prononcer sur les règles de gestion et les décisions organisationnelles. Il est indispensable d'avoir ces deux niveaux engagés dans le projet dès son lancement.

La toute première décision attendu du sponsor serait le choix du cas d'usage, tel que décrit au chapitre 1 page 5 sur le cas d'usage. Souhaite-t-on mettre à disposition des fonctionnalité de préalimentation des informations produit ou bien faciliter et fiabiliser le contrôle ?

10.1.2 Choix techniques et technologiques

Les autres grandes décisions à prendre en compte en début de projet sont d'ordre techniques et technologiques. La première est de savoir dans quel langage le modèle de production serait développé. Poursuivre en python aurait l'avantage de pouvoir capitaliser sur les travaux déjà effectués, et de se baser sur les bibliothèques qui ont fait leurs preuves (essentiellement numpy et scikit-learn). De plus, il faudrait dès le début du projet définir la criticité de ces fonctionnalités, et prévoir dès le départ le mode de redondance et l'éventuel plan de reprise d'activité. Enfin, même si les travaux effectués ont montré que le modèle en l'état était peu

gourmand en mémoire et puissance de calcul, il faudrait extrapolier le dimensionnement nécessaire pour tenir la charge pour l'ensemble des branches du groupe.

10.1.3 Développement microservice

Tant la préalimentation, que l'aide au contrôle des données seraient implantables comme un service. Il suffirait pour cela de publier un service, qui fonctionnerait de la manière suivante (illustré à la FIGURE 27 page 70) :

- le PIM appelle le service, avec un message contenant l'uid du produit à contrôler ou préalimenter
- le serveur récupère du PIM les données nécessaires au contrôle ou à la préalimentation
- en retour, il renvoie au PIM soit l'état du contrôle (OK, erreur, avertissement, avec les précisions nécessaires), soit les données telles qu'elles doivent être préalimentées
- le PIM, sur la base de ce retour, affiche le résultat du contrôle ou bien alimente les données et les présente à l'utilisateur

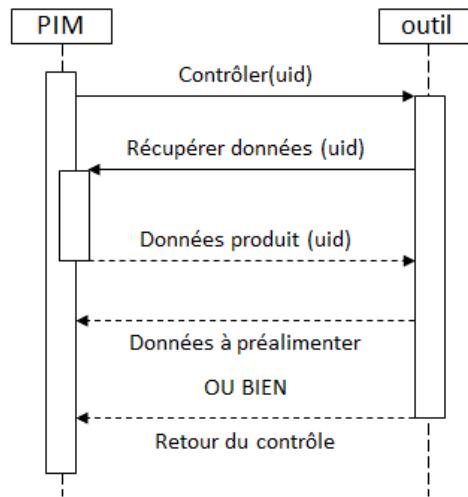


FIGURE 27 – Diagramme de séquence des échanges entre PIM et le service

Ce mode de fonctionnement nécessiterait des développements supplémentaires au niveau du PIM :

- appel du service lorsque les conditions sont réunies (ex : lancement du contrôle des données par l'acheteur)
- intégration des données retournées si le cas d'usage est la préalimentation
- affichage à l'utilisateur du résultat du contrôle si le cas d'usage est l'appui au contrôle

10.2 Maintien en condition opérationnelle

10.2.1 Mise à niveau

Si le projet devait se poursuivre en python, afin de disposer d'un outil restant maintenu au fil du temps, il est nécessaire d'industrialiser ce prototype. Pour cela, il faudrait :

- procéder au refactoring de certaines classes. Le requester porte trop de responsabilités (requêteage des données du PIM, conversion en DataFrame, conservation persistantes des données stockées localement, ...) et devrait être séparé en classes plus petites. Le calcul de similarité par projection devrait être réécrit, pour éviter d'utiliser deux TFIDFVectorizer, mais plutôt d'effectuer une projection par produit de matrices.
- mettre en place les fonctionnalités de typage statique
- poursuivre la rédaction de tests unitaires, avec une cible de couverture à 80%
- revoir ou réécrire l'ensemble des docstrings, et générer la documentation du code via des outils tels que Sphinx

10.2.2 Intégration et déploiement continu

En parallèle de l'écriture des tests et du passage au typage statique, il faudrait mettre en place un pipeline d'intégration et déploiement continu. Celui-ci permettrait d'identifier au plus tôt les problèmes en :

- procédant dès la production du code à la vérification du typage
- contrôlant la qualité de l'écriture du code (linting)
- faisant tourner les tests unitaires pour éviter les régressions

Le déploiement continu permettrait de mettre à disposition régulièrement et rapidement les améliorations et corrections aux utilisateurs

10.2.3 Monitoring de la performance du modèle

Un point crucial pour que le projet soit un succès est la surveillance de la performance du modèle. Il est possible que des dérives apparaissent, par exemple si au fil du temps les caractéristiques des données étaient modifiées (par exemple, de nouvelles manières de nommer les ingrédients apparaissaient, les règles de gestion changeaient, ...). Il faudrait être capable d'identifier ce phénomène, pour pouvoir prendre les actions correctrices nécessaires :

- changement des paramètres du modèle
- réentraînement sur des données plus récentes
- construction de nouveaux modèles pour gérer des domaines spécifiques

Un point à trancher est la manière de monitorer la performance. Une façon de faire serait de capturer les décisions prises par les utilisateurs après que le modèle a tourné. Par exemple, dans le cadre de l'aide au

contrôle, est-ce que l'utilisateur a décidé de passer outre l'avertissement qui lui a été remonté par le système ? Dans le cas de la préalimentation d'information, l'utilisateur a-t-il corrigé la valeur qui lui a été fournie ? Ces décisions devraient être enregistrées, puis analysées afin de s'assurer que la performance du modèle ne se dégrade pas au fil du temps.

10.3 Conclusion

Les résultats obtenus démontrent la faisabilité de la mise en place d'un outil d'extraction automatique de connaissance depuis des documents. Même s'il reste encore des travaux à mener avant de pouvoir déployer un outil apportant de la valeur au groupe, ce démonstrateur permet d'illustrer les fonctionnalités rendues accessibles par l'analyse de données et le machine learning. Cette ouverture, si elle est partagée avec le management du Groupe Pomona, peut permettre le lancement de nombreux projets innovants et porteurs de valeur.

Cinquième partie

ANNEXES

Annexe A FIGURES, TABLEAUX ET BIBLIOGRAPHIE

LISTE DES TABLEAUX

1	Exemples de listes d'ingrédients	9
2	Analyse volumétrique des pièces jointes	12
3	Analyse volumétrique des pièces jointes	13
4	Exemples d'écart entre les données étiquetées et celles du PIM	16
5	Pièces jointes dont les textes ne sont pas extractibles	17
6	Longueur des textes dans le dataset	20
7	Caractéristiques des vocabulaires	23
8	Illustration des listes d'ingrédients des groupes particuliers	29
9	Exemple de vectorisation d'un texte	33
10	Exemples du contenu de fiches techniques au format texte (tronqués)	37
11	Extrait des résultats de la prédiction	39
12	Prédictions identifiées comme correctes après postprocessing	44
13	Évaluation du modèle en utilisant les métriques de similarité	47
14	Illustration de l'évaluation du modèle à l'aide des métriques de similarité	48
15	Exemples de calculs de similarité	52

16	Exemple de tableau de données nutritionnelles	86
17	Volumétrie article par branche	100
18	Utilisation des variables catégorielles article au sein des branches RHD	103
19	Répartition des produits par statut	105
20	Répartition des produits par statut de migration	106
21	Répartition des produits par « qualité des données »	106
22	Nombre de produits par GTIN	107
23	Exemples de codes d'identification	108
24	Description des codes d'identification sur le dataframe	108
25	Exemples de dimensions	109
26	Description des dimensions sur le dataframe	109
27	Exemples de conservation	110
28	Description des conservations sur le dataframe	110
29	Exemples de libellés produit	111
30	Prédictions du meilleur modèle sur le set de test	135

TABLE DES FIGURES

1	Étapes du processus article pouvant être améliorées	7
2	Le Pareto des produits en fonction des fournisseur	17
3	Distribution des longueurs de textes	21
4	Corrélation entre longueurs des textes	21
5	Exemples de scores absolus de mots	23
6	Exemples de scores relatifs de mots	24
7	Projection des embeddings Word2Vec	25
8	Projection des comptes : PCA	26
9	Projection des comptes : Truncated SVD	26
10	Projection des fréquences : PCA	27
11	Projection des fréquences : Truncated SVD	28
12	Groupes particuliers dans la tSVD des fréquences	28
13	Schéma de principe du « modèle ouvert »	30
14	Schéma de principe du modèle basé sur les données étiquetées	36
15	Illustration de la méthodologie de mesure de la performance	42

16	Performance du modèle en fonction de la longueur des listes d'ingrédients	48
17	Illustration de la similarité cosinus	52
18	Illustration de la similarité projection	53
19	Résultats du tuning du preprocessing	55
20	Résultats du tuning du découpage des textes des documents	56
21	Résultats du tuning des fonctions de similarité	58
22	Résultats du tuning de l'utilisation de l'inverse document frequency	59
23	Résultats du tuning de l'utilisation de l'inverse document frequency	60
24	Illustration de la similarité cosinus avec score relatif	61
25	Illustration de la similarité cosinus avec score relatif	62
26	Illustration de la similarité cosinus avec score relatif	63
27	Diagramme de séquence des échanges entre PIM et le service	70
28	Les flux métier avec les partenaires commerciaux	78
29	La répartition de l'activité des branches	81
30	Le maillage régional de la branche ÉpiSaveurs	83
31	La distinction entre produit et article	90
32	Le processus de création article	92
33	Une capture d'écran du PIM	94
34	L'intégration du PIM au sein des systèmes du Groupe Pomona	95
35	Schéma de principe de la GDSN	97
36	L'uid d'un produit	97
37	Volumétrie article par branche	100
38	Recouvrements entre branches RHD	101
39	Répartition des articles en fonction des variable catégorielles	102
40	Répartition des produits par statut	105
41	Nombre de produits par GTIN	107
42	Distribution des fournisseurs par nombre de produits	107

BIBLIOGRAPHIE

- [1] Conseil de l'Union Européenne. Règlement n°1333/2008 sur les additifs alimentaires, dec 2008. <https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2008:354:0016:0033:FR:PDF>.
- [2] Conseil de l'Union Européenne. Règlement n°1907/2006 dit REACH, dec 2006. <https://bit.ly/2Jm05v9>.

- [3] Conseil de l'Union Européenne. Règlement n°1169/2011 dit INCO, nov 2011. https://www.senat.fr/europe/textes_europeens/ue0120.pdf.
- [4] Direction Générale de la Concurrence, de la Consommation et de la Répression des Fraudes. Étiquetage des denrées alimentaires : nouvelles règles européennes, jan 2015. <https://www.economie.gouv.fr/dgccrf/etiquetage-des-denrees-alimentaires-nouvelles-regles-europeennes>.
- [5] Direction Générales des Douanes et Droits Indirects. Notions essentielles sur la Déclaration d'Échanges de Biens. <https://www.douane.gouv.fr/notions-essentielles-sur-la-declaration-dechangers-de-biens>.
- [6] GS1. GDSN Trade Item Implementation Guide, nov 2019. https://www.gs1.org/docs/gdsn/tiig/3_1/GDSN_Trade_Item_Implementation_Guide.pdf.
- [7] GS1 France. Le réseau GDSN, le canal pour l'échange d'informations produits. <https://www.gs1.fr/Notre-offre/Le-reseau-GDSN-le-canal-pour-l-echange-d-informations-produits>.
- [8] GS1 Global. Global Data Synchronisation Network. <https://www.gs1.org/services/gdsn>.
- [9] GS1 Global. GS1 General Specifications. https://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf.
- [10] Jean-François Burnol. Normes Lp. <http://jf.burnol.free.fr/agregnormeslp.pdf>.
- [11] Groupe Pomona. Site institutionnel du groupe pomona. <https://www.groupe-pomona.fr/>.
- [12] Wikipedia. Bag-of-words model. https://en.wikipedia.org/wiki/Bag-of-words_model.
- [13] Wikipedia. Damerau–Levenshtein distance. https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance.
- [14] Wikipedia. Jaro Similarity. https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance#Jaro_Similarity.
- [15] Wikipedia. Jaro–Winkler Similarity. https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance#Jaro%E2%80%93Winkler_Similarity.
- [16] Wikipedia. Latent Semantic Analysis. https://en.wikipedia.org/wiki/Latent_semantic_analysis.
- [17] Wikipedia. Levenshtein distance. https://en.wikipedia.org/wiki/Levenshtein_distance.
- [18] Wikipedia. Liste des additifs alimentaires. https://fr.wikipedia.org/wiki/Liste_des_additifs_alimentaires.
- [19] Wikipedia. Norm (mathematics). [https://en.wikipedia.org/wiki/Norm_\(mathematics\)#Euclidean_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm).
- [20] Wikipedia. Optical character recognition. https://en.wikipedia.org/wiki/Optical_character_recognition.
- [21] Wikipedia. Word2vec. <https://en.wikipedia.org/wiki/Word2vec>.

Annexe B

CONTEXTE MÉTIER : LE GROUPE, INFORMATION PRODUIT ET GÉNÉRALITÉS SUR LES DONNÉES

L'objet de l'ensemble de cette annexe est de donner sur le Groupe Pomona et la gestion de l'information produit. des éclairages nécessaires à la compréhension du cas d'usage développé. Bien d'autres aspects sur la société, pourraient être mentionnés (ex : des indicateurs sur l'activité, l'histoire du groupe...) mais ils seront omis car non indispensables à la compréhension du sujet. Plus de détails sur le groupe sont accessibles sur le site web de la société[11].

B.1 Description du groupe

B.1.1 Le métier du Groupe Pomona

Le Groupe Pomona est une société de distribution livrée de produits alimentaires à destination des professionnels des métiers de bouche. L'activité du groupe consiste uniquement à acheter et revendre de la marchandise, à l'exclusion de toute activité de fabrication ou de transformation¹. Le groupe Pomona est une société de *distribution*. Elle ne possède d'ailleurs pas d'actif industriel (autre que des entrepôts logistiques) ni d'agrément pour transformer les marchandises.

Cette activité d'achat/vente se fait dans la majorité des cas sous le régime du *négoce*, à savoir que le groupe acquiert la propriété des marchandises qu'il commercialise avant de la céder à ses clients. L'autre régime est celui dit de la *prestation (logistique)*. Dans ce cas, par le jeux d'écritures comptables, la valorisation du stock disparaît des comptes du groupe. Néanmoins, indépendamment de cet aspect purement comptable, l'ensemble :

des flux de documents : commandes d'achat, factures fournisseur, commandes de vente, factures clients

des flux financiers : paiements fournisseur, paiements client

des flux physiques : réception et stockage, préparation et expédition

restent largement inchangés.

^{1.} de très rares cas de transformation existent (ex : mûrissement de fruits, filetage de poisson) mais sont extrêmement exceptionnels

Pour résumer, l'activité de l'ensemble des entités du groupe pourraient se résumer via le schéma présenté à la FIGURE 28 page 78

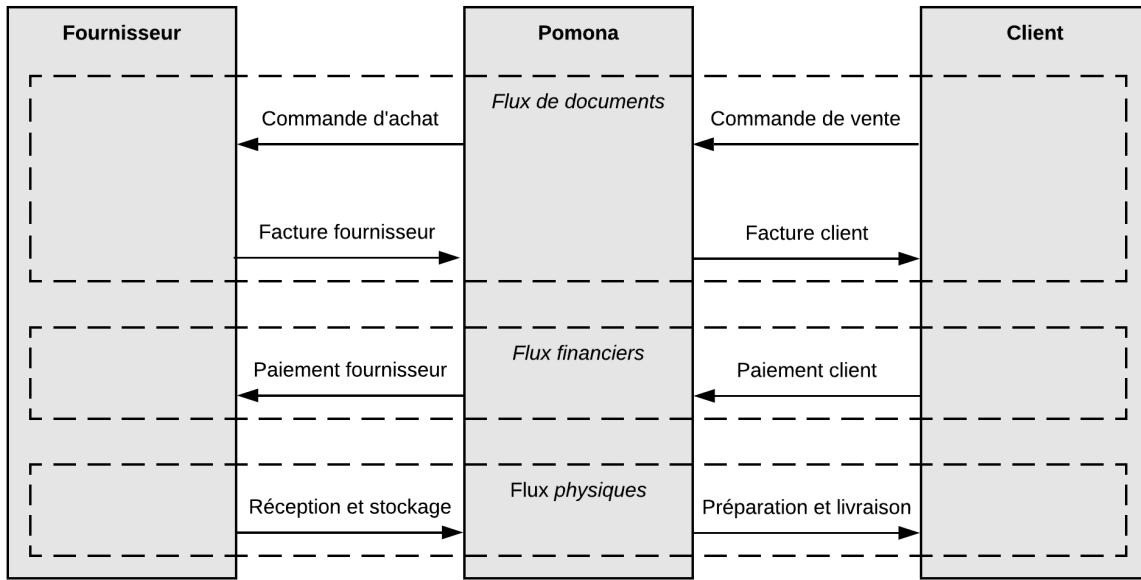


FIGURE 28 – Les flux métier avec les partenaires commerciaux

Le métier du groupe est d'être un grossiste, qui achète et revend des produits alimentaires² sans produire ou transformer quoi que ce soit.

B.1.2 La décentralisation

Le Groupe Pomona est un groupe fortement décentralisé, avec des organisations largement indépendantes les unes des autres.

Les Directions fonctionnelles

Pour des raisons évidentes de recherche de synergies ou de conformité réglementaires, certaines activités restent toutefois mutualisées à la maille du groupe. Il s'agit des organisations suivantes :

La Direction Administrative et Financière (DAF) : regroupe les équipes comptables groupe, l'audit interne et la consolidation financière

La Direction Qualité Sécurité et Environnement (DQSE) : est en charge de définir et contrôler l'application des standard de qualité

2. dans la grande majorité des cas, cf. *Les produits non-alimentaires* C.1 page 98

La Direction des Systèmes d'Information (DSI) : développe et maintient en condition opérationnelles les systèmes d'information du groupe

La Direction Technique et Logistique (DTL) : est en charge des projets immobiliers (entrepôts), des négociations avec les transporteurs et joue un rôle de conseil interne sur les sujets logistiques

La Direction des Ressources Humaines : se charge de l'ensemble des aspects en lien avec le recrutement, la paye et les sujets sociaux

La Direction Commerciale groupe (DCG) : définit une stratégie et des bonnes pratiques commerciales et marketing

Les clients du groupe

Afin de comprendre l'organisation du groupe, il est nécessaire de connaître la typologie de ses clients. Comme mentionné précédemment, le groupe s'adresse exclusivement aux professionnels des métiers de bouche. Aucune marchandise n'est vendue à des particuliers. Les principales typologies de clients sont les suivantes :

Les Sociétés de Restauration : elles exploitent les restaurants d'entreprise, certaines cantines d'établissements d'enseignement, les maisons de retraite, ...

Les Marchés Publics : regroupent les clients qui dépendent des collectivités (écoles, hôpitaux, prisons, ...)

La restauration commerciale : est l'ensemble des restaurants à vocation commerciale, qu'ils soient chainés (hippopotamus, O'Tacos, ...) ou indépendants (« le restaurant du coin »)

Les spécialistes : il s'agit des détaillants spécialisés qui s'adressent aux particuliers. Boulanger, pâtissier, bouchers, traiteurs, vente à emporter, ...

Les Grandes et Moyennes surfaces (la GMS) : sont les enseignes de la grande distribution. En général, l'accès à ces clients est compliqué par les règles mises en place par leurs centrales d'achat. Il représentent en général qu'un canal de vente d'opportunité.

Les trois premières de ces catégories représentent ce que l'on appelle la *Restauration Hors Domicile (RHD)* (ou parfois également la Restauration Hors Foyer, RHF).

Premier niveau de décentralisation : les branches

Le Groupe Pomona est divisé en branches, qui sont des unités opérationnelles indépendantes, et qui ont toute latitude pour gérer leurs stratégie et politique commerciales, la gestion de leurs achats, leur stratégie marketing, ... En particulier, les systèmes d'information ne sont pas identiques entre les différentes branches. Afin d'éviter de se concurrencer entre elles, leurs domaines d'activité respectifs ont été partitionnés par familles de produit commercialisés, segments client cibles et géographie.

Les branches RHD Les branches RHD s'adressent aux clients de la Restauration Hors Domicile (cf. section B.1.2 page 79) en France. Elles se répartissent ce marché en travaillant des gammes de produits distinctes. Il s'agit des branches historiques du groupe, qui représentent l'essentiel de son chiffre d'affaire. La répartition par produit est la suivante :

PassionFroid : spécialiste des *produits surgelés, de la viande fraîche et des produits laitiers*

ÉpiSaveurs : spécialiste des produits qui se conservent à température ambiante : *produits d'épicerie, conserves, boissons et consommables de cuisine non-alimentaires*

TerreAzur : spécialistes des *Fruits et Légumes frais, et Produits De la Mer frais*

La non-concurrence entre les branches est assurée par le fait qu'elles ne commercialisent pas les mêmes produits. Bien que nommées RHD, elles peuvent également vendre leurs produits à la grande distribution (GMS : Grandes et Moyennes Surfaces), mais généralement ces marchés sont verrouillés par les centrales d'achat des grandes enseignes. La branche TerreAzur arrive toutefois à prendre des parts de marché significatives sur ce segment. Les branches RHD utilisent le logiciel SAP comme système de gestion. La branche TerreAzur est en cours de déploiement, en 2020 environ les 2 tiers des succursales travaillent avec ce logiciel.

Les branches spécialistes Les branches spécialistes s'adressent aux clients dits spécialistes (cf. section B.1.2 page 79) en France. Elles sont en mesure de commercialiser tout type de produit pour répondre aux besoins de leurs clients. En particulier, elles peuvent tout à fait commercialiser certains produits qui sont également vendus par les branches RHD. Elles se répartissent la clientèle spécialiste de la manière suivante :

Délice et Crédit : s'adresse aux *Boulanger et Pâtissier*

Saveurs d'Antoine : s'adresse aux *Bouchers, Charcutiers et Traiteurs*

Relais d'Or : s'adresse à la *restauration commerciale indépendante*

Comme pour les branches RHD, ces branches peuvent lorsqu'elles en ont l'opportunité vendre leurs produits à la GMS.

L'étranger Bien que le Groupe Pomona soit une société dont l'essentiel de l'activité est faite sur le marché français, deux réseaux sont en cours de constitution sur des pays limitrophes. Ces branches sont susceptibles de travailler tout type de produit, à destination de tout type de client. Elles sont positionnées sur les marchés suivants :

Pomona Suisse : présente sur le marché Suisse

Pomona Iberia : présente sur le marché Espagnol

On peut synthétiser la répartition de l'activité par branche de la manière présentée à la FIGURE 29 page 81.

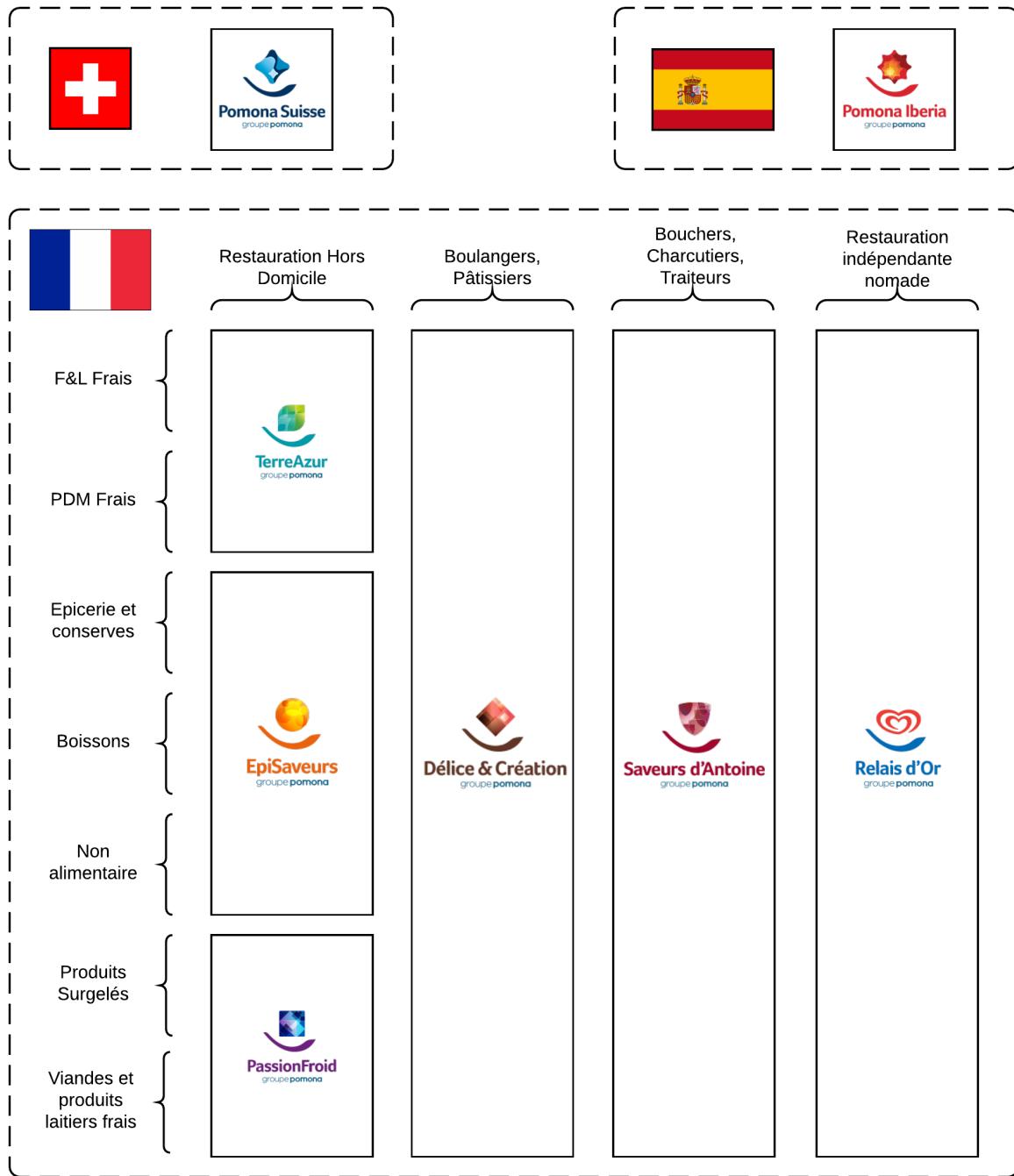


FIGURE 29 – La répartition de l'activité des branches

Le second niveau de décentralisation : les succursales

Chacune des branches est elle-même à son tour décentralisée en un réseau d'entrepôts régionaux : les succursales (parfois également appelées simplement « régions »). Ces succursales sont gérées comme des PME indépendantes, avec un directeur et un compte de résultat qui leur est propre. Si certaines négociations avec des fournisseurs ou des clients nationaux sont parfois menées par les branches, les succursales sont autonomes dans :

- la définition de leur assortiment, même si des contraintes s'appliquent
- la stratégie de développement commercial
- la négociation des prix d'achat
- la négociation des prix de vente
- la politique de rémunération de leurs employés

À ce titre, elles ont leurs propres équipes d'achat, leurs équipes commerciales (télévente et vente route), leurs équipes administratives et évidemment leurs équipes logistiques (essentiellement en entrepôt et les chauffeurs livreurs en charge des livraisons client).

Certaines activités restent de la responsabilité des équipes centrales des branches, comme : la négociation avec les clients ou les fournisseurs nationaux, la constitution de l'assortiment commun (les produits que toutes les succursales doivent détenir), la gestion des référentiels de données de base métier, ...

Un exemple de maillage régional est présenté en FIGURE 30 page 83, sachant que ce maillage régional est différent pour chacune des branches.

B.2 La gestion de l'information produit

B.2.1 L'information produit

Utilisations de l'information produit

Conformité réglementaire La gestion de l'information produit est essentiellement une contrainte réglementaire à statisfaire. Comme mentionné au préambule, la réglementation autour de l'information des consommateur s'est sans cesse complétée au cours des dernières années. Un des textes centraux est le règlement n°1169/2011 dit INCO (INformation COnsommateur)[3][4]. C'est ce règlement qui définit l'ensemble des informations qui doivent être étiquetées sur le produit (liste d'ingrédients, tableau de données nutritionnelles, ...), mais également affichée au client lors de commande en ligne sur les sites de e-commerce. Il s'agit principalement d'informations relatives à la sécurité alimentaire (ex : les allergènes) ou la santé (ex : informations nutritionnelles).

Attentes client Les consommateurs finaux (les « convives ») étant de plus en plus sensibles au contenu de leur assiette, les clients du groupe sont de plus en plus demandeurs d'informations relatives aux produits qu'ils



FIGURE 30 – Le maillage régional de la branche ÉpiSaveurs

commandent. Ils demandent donc régulièrement des informations qui vont au-delà de ce qui est normalement prévu par la réglementation.

De plus, sur certains marchés pour lesquels des contrats courant sur de longues périodes - jusqu'à un an - sont établis (les marchés publics sont très concernés), il n'y a pas d'échantillonnage des produits. La seule manière pour ces clients d'évaluer la qualité des produits est de se référer aux documents contenant les informations produit, fournis par les distributeurs.

Gestion Certaines informations relatives aux produits sont nécessaires pour des raisons de gestion administrative. Par exemple, la gestion des taxes sur les produits alimentaires est complexe :

- les taux de TVA sont variables en fonction du type de produit
- des taxes spécifiques s'appliquent aux produits contenant de l'huile ou de la farine
- des règlements particuliers s'appliquent aux alcools
- ...

D'autres informations, comme la nomenclature douanière, sont nécessaires pour effectuer les déclarations auprès des douanes européennes.

Un autre type d'information capital pour la gestion des flux d'achat et de vente sont les informations logistiques, qui définissent par exemple le nombre d'unités consommateur dans le colis, le nombre de colis sur une palette, ... Une gestion rigoureuse de ces informations est indispensable pour que les flux d'achat ou de vente soient correctement exécutés (que les quantités commandées soient les bonnes, que les montants facturés soient corrects, ...).

Des produits bruts aux produits transformés

Le niveau d'exigence en termes d'information produit est variable en fonction du niveau de transformation de ce produit. Par exemple, sur des fruits et légumes frais, à peu de choses près seul le pays d'origine doit être affiché au client. Sur une barre chocolatée, ou un plat cuisiné, il sera nécessaire d'afficher :

- une liste d'ingrédients (mettant en évidence les allergènes)
- un tableau de données nutritionnelles (protéines, glucides, ...)
- une dénomination réglementaire

Les grands types d'information

On se focalisera dans ce paragraphe sur les informations relatives aux *produits alimentaires*.

La composition La première grande famille de données réglementaires sont les données de composition. Elles détaillent quels sont les ingrédients qui sont mis en œuvre dans la fabrication des produits. Évidemment, la composition a en général plus de sens pour les produits transformés que pour les produits bruts. Elle peut prendre la forme d'un texte listant la liste des ingrédients (l'étiquetage de ce texte est en général obligatoire sur les emballages des produits), ou d'un tableau.

Les ingrédients incluent également les additifs. Il s'agit de substances ajoutées à la recette pour répondre à des fonctions particulières (colorant, exhausteur de goût, émulsifiant, ...). Elles ne représentent en général qu'un pourcentage en masse négligeable dans la composition totale du produit.

Le pourcentage en masse est parfois inclus sur certains ingrédients. La réglementation l'oblige dans certains cas, par exemple quand l'ingrédient en question est mentionné dans la dénomination du produit (pour une *tarte aux framboises*, la proportion de framboise doit être mentionnée dans la composition).

Enfin, un aspect à la fois réglementaire et particulièrement important est la présence d'allergènes dans la composition. Le règlement INCO[3][4] impose de mettre en évidence les allergènes relevant d'une des 14 catégories suivantes :

1. Céréales contenant du gluten, à savoir blé, seigle, orge, avoine, épeautre, kamut ou leurs souches hybrides, et produits à base de ces céréales
2. Crustacés et produits à base de crustacés
3. Œufs et produits à base d'œufs
4. Poissons et produits à base de poissons
5. Soja et produits à base de soja
6. Lait et produits à base de lait (y compris le lactose)
7. Fruits à coque, à savoir : amandes, noisettes, noix, noix de cajou, noix de pécan, noix du Brésil, pistaches, noix de Macadamia ou du Queensland, et produits à base de ces fruits
8. Céleri et produits à base de céleri
9. Moutarde et produits à base de moutarde
10. Graines de sésame et produits à base de graines de sésame
11. Anhydride sulfureux et sulfites
12. Lupin et produits à base de lupin
13. Mollusques et produits à base de mollusques

Il peut y avoir deux niveaux de présence d'un allergène dans un produit (au-delà de la simple absence) :

intentionnellement mis en œuvre : dans le cas où un ingrédient allergène fait volontairement partie de la recette. Ex : présence de moutarde dans un plat cuisiné.

contamination croisée : par exemple lorsque le produit fini est issu d'une chaîne de transformation qui traite un ingrédient allergène, mais que cet ingrédient ne fait pas partie de la recette. Ce cas de figure est en général mis en évidence par des mentions telles que "*Peut contenir des traces de soja*" ou bien "*Transformé dans un atelier processant également des fruits à coques et du sésame*".

Les informations nutritionnelles Une autre grande famille d'information produit sont les informations nutritionnelles. Elles détaillent la quantité des principaux nutriments contenus dans les produits. Certains

d'entre eux sont rendus obligatoires par le règlement INCO [3][4] cf. l'exemple de tableau à la TABLE 16 page 86, et d'autres sont optionnels, comme par exemple la quantité de fer, de calcium, ...

Informations nutritionnelles	Pour 100g	Pour un biscuit	% des AJR pour un biscuit
Énergie	1674 kJ 398 kcal	209 kJ 50 kcal	3 %
Protéines	3.0 g	1.0 g	3 %
Matières grasses	13.0 g	1.6 g	2 %
dont acides gras saturés	5.8 g	0.7 g	4 %
Glucides	66 g	8.2 g	3 %
dont sucres	48 g	6.1 g	7 %
Fibres alimentaires	2.5 g	0.3 g	
Protéines	3.3 g	0.4 g	1 %
Sel	0.41 g	0.05 g	1 %

TABLE 16 – Exemple de tableau de données nutritionnelles

La réglementation rend obligatoire de mentionner les informations nutritionnelles de cette table pour 100g, ou 100mL de produit (pour les boissons).

Les informations nutritionnelles peuvent également se présenter sous forme d'allégations, qui ont des définitions précises dans la réglementation. Ces allégations peuvent être : *sans sel, faible en sucres, riche en fibres, ...*

Les origines Du fait de la complexification des opérations de transformation et de la complexification des flux d'échanges de marchandises, l'origine des produits alimentaires est une notion qui n'est pas définie avec précision dans l'absolu. Il n'y a donc pas non plus de réglementation précise sur le sujet, si ce n'est que l'information produit doit toujours être présentée de manière loyale au consommateur. On peut se donner une règle simple pour définir l'origine d'un produit alimentaire : plus il est brut, plus va compter l'origine de ses ingrédients ; plus il est transformé, plus va compter le lieu de dernière transformation.

Par exemple, sur des morceaux piés de viande fraîche, on aura des origines multiples en fonction du pays de naissance, d'élevage ou d'abattage de la bête. Et à l'inverse, sur un steak haché cette information n'aura aucun sens dans la mesure où il est produit d'un assemblage de « minéraux » pouvant provenir de multiples pays. L'industriel pourra choisir de communiquer sur le fait que la viande a été transformée en steak dans telle usine par exemple.

Les données logistiques On appelle données logistiques essentiellement le plan de palettisation et de conditionnement du produit. Il s'agit de la définition de la « hiérarchie logistique » du produit. Cette hiérarchie se base d'abord sur la définition d'une « unité de base » qui est la plus petite unité légalement détaillable (i.e. qui porte l'ensemble des informations réglementaire pour sa commercialisation). Ces notions ont été standardisées par l'organisme international de standardisation GS1[6]. Deux exemples pour illustrer :

- pour un boîte de sachets de thé, l'unité de base est la boîte car les sachets de thé ne portent pas les informations nécessaires à leur commercialisation
- pour un paquet de barres chocolatées (comme celles qu'on peut trouver au détail en boulangerie), l'unité de base est la barre car elle porte l'ensemble des mentions réglementaires sur son emballage

La hiérarchie logistique est à la fois :

- la définition des niveaux successifs d'emballage des produits : combien d'unités de base dans un paquet, combien de paquets dans un carton, combien de cartons sur une palette, ...
- la définition du contenu de l'unité de base (ex : le nombre de sachets de thé, le nombre de doses dans une boîte d'aides culinaires, ...)

Les données logistiques concernent également les durées de vie du produit (qu'il s'agisse de Date Limite de Consommation ou Date de Durabilité Minimale) :

- la durée de vie totale à fin de production
- la durée de vie garantie à la livraison

Parfois, certaines contraintes d'approvisionnement peuvent être mentionnées :

- unités commandables (ex : on ne peut commander que des cartons complets)
- multiples de commande (ex : on ne peut commander les cartons que 10 par 10 pour des raisons de montage des palettes)
- minimum de commande (ex : il faut commander au minimum 30 cartons)

mais elles sont dépendantes d'un accord entre l'industriel et son client distributeur et ne sont donc pas à proprement parler des informations produit.

Les données administratives et financières Les données dites administratives et financières regroupent le reste des informations de gestion pour lesquelles il existe des contraintes réglementaires. Il s'agit :

- du taux de TVA du produit
- de sa nomenclature douanière et du pays d'origine au sens de la déclaration d'échange de biens^[5]
- de toute autre taxe applicable au produit

Les labels Afin de garantir des qualités spécifiques à certains produits, des organismes de certification ont mis en place des labels pouvant s'appliquer aux produits. En général, ils se basent sur des cahiers des charges et peuvent être assortis d'audits de certification ou de contrôle. Ils peuvent garantir des méthodes de production ou transformation, des lieux de production, des caractéristiques de leurs ingrédients, des pratiques commerciales équitables, ... Les types de labels les plus connus sont :

- les produits Biologiques
- les origines protégées (Appellation d'Origine Protégée, Indication Géographique Protégée, viandes de France, Bleu Blanc Cor, Régions UltraPériphériques d'Europe...)
- les pratiques commerciales équitables (Max Havelaar, ...)

- les modes de production respectueux de l'environnement (Aquaculture Stewardship Council, Marine Stewardship Council, Roundtable on Sustainable Palm Oil, Nordic Swan, ...)
- la qualité « générale » des produits (Label Rouge, ...)

Les données marketing Certaines données marketing font également partie de l'information produit. La plus évidente est la marque commerciale du produit, qui parfois définit totalement le produit. Par exemple, on sait ce qu'est un Snickers, de la Mousline, du Nutella, ... Les produits peuvent également porter d'autres allégations marketing, non réglementaires ou labelisantes : Élu produit de l'année, Vu à la télé, Issu de notre savoir-faire centenaire, ...

B.2.2 Le processus

Le fournisseur est propriétaire des informations produit

Comme présenté à la section B.1.1 page 77, le Groupe Pomona n'a pas d'activité de fabrication ou de transformation de marchandises. À ce titre, l'ensemble des données produits ne peuvent être déterminées que par les fournisseurs de ces produits. L'ensemble des entités du groupe s'appuient donc sur les données transmises par les industriels ou producteurs de marchandises.

Il peut arriver que certains produits soient achetés par Pomona à d'autres négociants non-producteurs. Dans ce cas, de la même manière que le groupe Pomona a la responsabilité de collecter puis transmettre les informations produit à ses clients, ces fournisseurs négociants doivent eux-même aller chercher l'information produit et la transmettre à Pomona.

Dans tous les cas, c'est le fournisseur qui est responsable de produire et de transmettre l'information produit aux entités du groupe Pomona.

La notion de produit et d'article

Un mot sur la modélisation des données adoptée est nécessaire pour comprendre les grandes lignes du processus. Comme il a été vu à la section B.1.2 page 79, certains produits sont susceptibles d'être commercialisés par plusieurs branches du groupe.

De plus, du fait que les systèmes d'information ne sont pas identiques entre les branches, certaines contraintes imposent parfois des différences de modélisation, des duplications volontaires de codes pour répondre à ces contraintes. Une illustration de ce point pour clarifier : la facturation client pour une canette de soda peut se faire au litre (permet de comparer les prix entre les différents conditionnements et les différentes marques) ou à la cannette (permet de se faire une idée du coût portion d'un produit). Or, la possibilité de pouvoir facturer un même article dans plusieurs unités différentes n'est pas une fonctionnalité offerte par tous les systèmes d'information. En particulier, ÉpiSaveurs peut gérer dans ce cas un unique article et le facturer dans l'unité de son choix en fonction des demandes des clients. Mais Délice et Création (qui possède

un système d'information différent) doit dupliquer cet article car une seule unité de facturation est possible pour un article donné.

Enfin, au-delà des contraintes liées au SI, certaines pratiques imposent de laisser aux branches une indépendance forte dans la gestion de leurs référentiels articles. Il faut savoir que commercialiser sous un même code article des produits qui sont similaires permet d'obtenir des gains de productivité à plusieurs niveaux :

- on économise des emplacements en entrepôt (un emplacement ne peut contenir qu'un article)
- on gagne du temps administratif dans la gestion des prix : le foisonnement d'articles impose de gérer plus de prix client
- ...

La contrepartie à adopter cette pratique est qu'il n'est alors plus possible de différencier ces produits similaires, par exemple pour leur appliquer des prix de vente distincts, ou bien offrir la possibilité à un vendeur de garantir au client la livraison d'un produit plutôt que l'autre. Néanmoins, en fonction de la clientèle adressée, certains produits pourront être considérés comme similaires, alors que pour d'autres ils ne seront pas interchangeables.

Cet exemple est détaillé dans la FIGURE 31 page 90.

On différencie donc les deux notions suivantes :

les produits : ils représentent une marchandise physique produite par un fournisseur. Ce sont les produits qui portent les *informations produit* décrites à la section B.2.1 page 84. Un produit ne peut appartenir qu'à un seul fournisseur. Le référentiel produit est unique pour l'ensemble du groupe.

les articles : ce sont les objets qui sont gérés par les branches dans leurs systèmes de gestion respectifs. Leurs attributs sont très liés au système d'information qui les porte. Chaque branche gère de manière autonome son référentiel article, incluant les liens qui sont faits entre produits et articles.

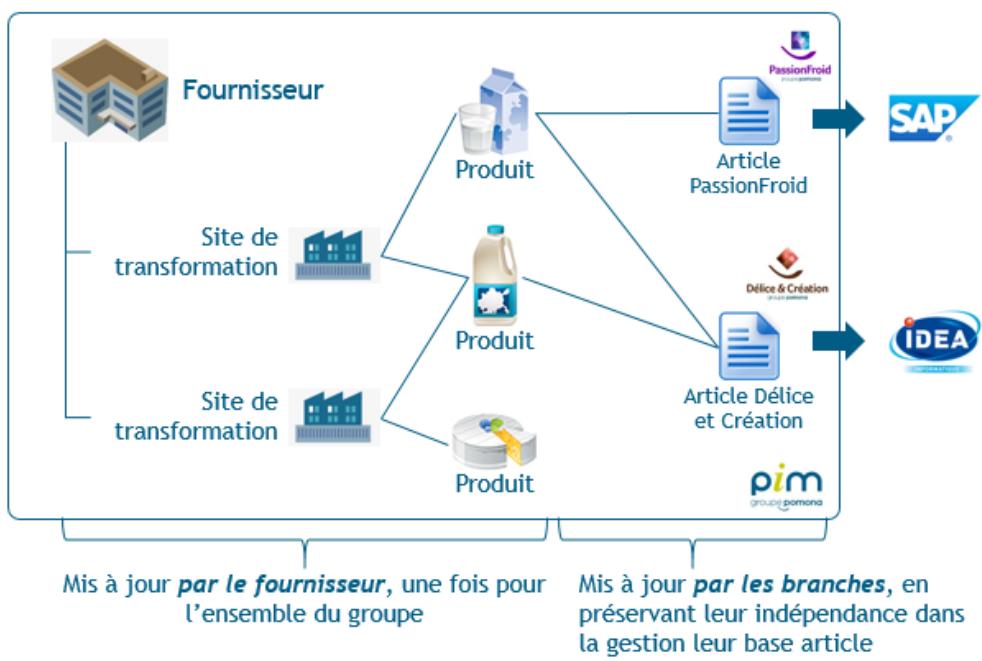
Cette modélisation permet de répondre à l'ensemble des contraintes présentées dans ce paragraphe.

Les acteurs

L'acheteur L'acheteur peut être en succursale, ou à la centrale d'achat de la branche (cf. B.1.2 page 79 et B.1.2 page 82). Il est responsable de l'assortiment, i.e. des articles qui sont commercialisés par sa branche, ou sa succursale. Il est à l'origine de la création d'un nouveau produit, dans la mesure où c'est lui qui va décider de le référencer ou pas. La gestion de l'information produit ne représente qu'une petite partie de l'ensemble de ses responsabilités (qui peuvent inclure négociations fournisseurs, gestion du sourcing, gestion des approvisionnements, prévisions d'évolution des prix, ...).

Le fournisseur Il a été vu que le Groupe Pomona - dans sa qualité de distributeur - n'est pas en mesure de déterminer seul les informations produit sur les marchandises qu'il commercialise. À ce titre, les équipes des fournisseurs sont en charge de transmettre les informations produit à Pomona. En général, ce sont deux types de profil qui effectuent cette tâche :

- les commerciaux (aux sens large, incluant les assistants commerciaux)



Dans cet exemple fictif, le type de conditionnement du lait n'a aucun impact sur les clients de la branche Délice et Création. Elle commercialise donc sous un même code article deux produits distincts.

Pour des raisons de contrainte de conservation, la branche PassionFroid a quant à elle choisi de ne commercialiser qu'un seul produit - en brique. Elle pourrait choisir d'ouvrir un nouveau code article pour le lait en bouteille (non représenté sur le schéma ci-dessus).

FIGURE 31 – La distinction entre produit et article

— les ingénieurs qualité

Le gestionnaire de référentiel - SEGER Le gestionnaire de référentiel travail au SErvice de GEstion des Référentiels (SEGER), qui sont des équipes positionnées au niveau des branches (cf. B.1.2 page 79). Ils sont responsables de la qualité des données dans les référentiels métier (articles, fournisseurs, clients, ...). La gestion des données de base dans les référentiels est leur mision principale.

L'ingénieur qualité L'ingénieur qualité travaille à la DQSE ou en succursale (cf. B.1.2 page 78 et B.1.2 page 82). Dans le processus de gestion de l'information produit, il est en charge de contrôler la qualité des données, mais également leur conformité réglementaire (ex : on ne peut qualifier un produit de « faible en sel » que s'il comporte moins de n grammes de sel pour 100 grammes de produit.) La gestion de l'information produit ne représente qu'une partie des responsabilités de l'ingénieur qualité.

Les contrôles

Comme cela a été vu dans la section B.2.1 page 82, il est nécessaire que les différentes entités du groupe soient en possession d'une information produit fiable. Or, avoir des données de qualité nécessite des efforts de la part des métiers, en particulier lorsque le processus n'est pas entièrement porté en interne dans la société. À ce titre, plusieurs étapes de contrôle ont été définies dans le processus de gestion du référentiel de données produit et article :

lorsque le fournisseur a saisi les données produit : la personne à l'origine de la demande de référencement (en général, un acheteur) doit contrôler la cohérence des données produit

lorsque le demandeur a demandé la création d'un article : le gestionnaire de référentiel valide à nouveau la cohérence des informations produit

après la création article, de manière asynchrone : le service qualité contrôle par échantillonnage les données d'une partie des produits et articles qui ont été modifiés pendant une période.

Le retour d'expérience montre que ces contrôles, loin d'être redondants, sont nécessaires pour avoir une qualité de données acceptable. Ces processus de contrôle sont décrits à la FIGURE 32 page 92.

Les contrôles effectués à chacune des étapes sont les suivants :

contrôle de la complétude des données : vérification que les données transmises comportent l'ensemble des données attendues

contrôle de cohérence entre les données : vérification que les informations transmises sont cohérentes entre elles (ex : un allergène présent dans la liste d'ingrédients du produit a bien été signalé comme allergène par ailleurs)

contrôle de la cohérence avec les pièces jointes : en plus de données structurées, les fournisseurs transmettent également des fichiers portant des informations produit (ex : l'étiquette produit ou le

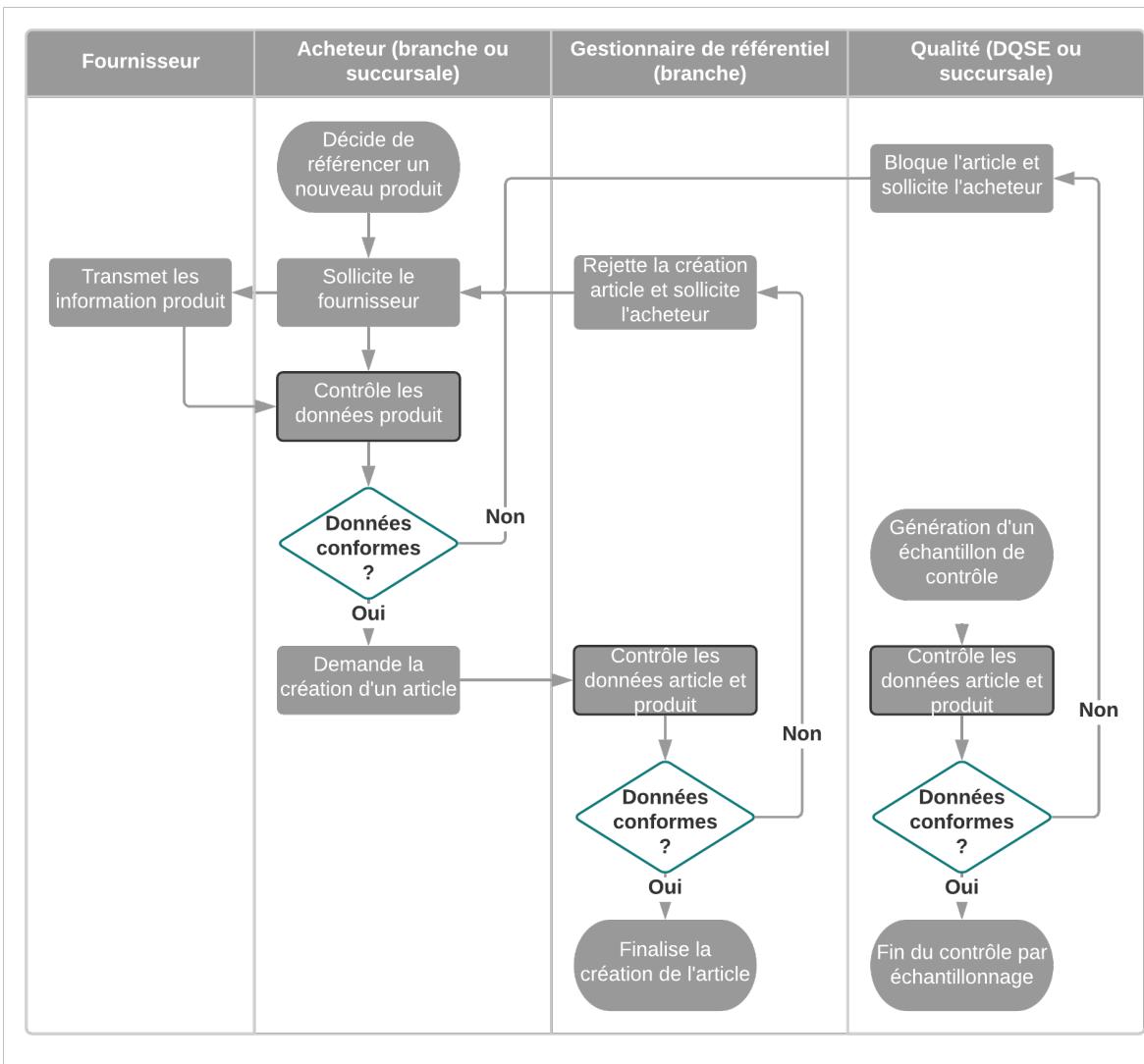


FIGURE 32 – Le processus de création article

visuel de l'emballage). Ces pièces jointes sont décrites à la section 2.2 page 9. La personne en charge du contrôle vérifie que les données transmises sont cohérentes avec ces documents.

B.2.3 Les outils informatiques associés

Comme vu dans la description des branches du groupe (voir section B.1.2 page 79), les outils informatiques ne sont pas tous les mêmes sur l'ensemble des branches. Ainsi, les outils utilisés pour la gestion de l'information produit ne sont pas les mêmes.

Les branches faiblement outillées

Les branches étrangères (Pomona Suisse, Pomona Iberia), spécialistes (Délice et Création, Saveurs d'Antoine) et la branche TerreAzur sont aujourd'hui faiblement outillées. Cela signifie que l'information produit est en général stockée uniquement sous la forme de fichiers (essentiellement les pièces jointes, décrites à la section 2.2 page 9). L'ensemble des échanges avec les fournisseurs se font par mail, et les articles sont créés directement dans les systèmes de gestion par les gestionnaires de référentiel. Les liens entre les articles et les informations produit ne sont pas matérialisés dans les systèmes informatiques.

Le GIP

Le GIP (Gestion de l'Information Produit) est utilisé sur la branche PassionFroid. C'est un système de gestion de l'information produit qui est maintenant obsolescent et en cours de remplacement. Il a toutefois le mérite de permettre le stockage dans une application des données et des pièces jointes relatives aux produits, avec la possibilité d'accéder aux informations produit à partir des identifiants des articles. C'est ce système qui a permis de pouvoir alimenter les sites de e-commerce PassionFroid et ÉpiSaveurs avec les informations produit. Il est toutefois ancien, et ne propose pas de fonctionnalité d'export en masse fiable. Il s'agit d'une application qui n'est pas ouverte aux utilisateurs externes au groupe, et les échanges avec les fournisseurs passent donc par des échanges de mails.

Le PIM

Le PIM (Product Information Management) est un système de gestion de l'information produit qui a été mis en production en mai 2019, pour la branche ÉpiSaveurs. Il a vocation à être déployé sur l'ensemble des branches du groupe. En terme de responsabilités, il vise à gérer la totalité des informations produit, mais également d'être maître sur les référentiels articles. Ce sera le système de référence pour tous les autres systèmes consommant de l'information produit, ou des données article.

Description générale de l'outil C'est un système qui porte l'ensemble du processus de gestion de l'information produit, tel que décrit à la FIGURE 32 page 92. Il est accessible aux fournisseurs du groupe, qui

viennent directement mettre à disposition les données et les pièces jointes. Les fonctionnalités caractéristiques de ce système par rapport à d'autres systèmes informatiques sont :

- la gestion de workflows (processus), qui permet d'orchestrer l'activité de l'ensemble des acteurs du système
- la possibilité de paramétriser un modèle de données relativement complet, avec des centaines de métadonnées sur les différents objets
- la présentation de l'interface utilisateur via le navigateur, qui permet de s'adresser simplement à des acteurs hors du groupe (pas de client lourd à installer)
- la gestion performante de pièces jointes (documents informatiques) en grandes quantité
- Une gestion de versions des objets robuste, permettant d'auditer l'historique ou de restaurer des données dans un état précédent

Cet outil porte entre autres les fonctionnalités de contrôle des informations, comme illustré à la FIGURE 33 page 94. L'intégration du PIM au sein des systèmes informatiques du Groupe Pomona est décrite dans la FIGURE 34 page 95.

The screenshot shows a web-based PIM application interface. At the top, there's a navigation bar with links like 'ESPACE FOURNISSEURS', 'SMILDE FOODS', 'ROMI HEALTHY BLUE HUILE DE FRITURE LONGUE DURÉE À LA GRASSE DE BOEUF EN BIB 15 L', and a date '2019-11-29_CONTROLE_QUALITE'. The main area is titled 'CONTROLES QUALITÉ' and shows a 'Section en cours de modification'. It displays nutritional information for a product, including a table of values per 100g and a list of ingredients. To the right, there's a summary section with a green bar labeled 'NUTRITION : OK' and a 'Déclaratif des données nutritionnelles' section. At the bottom, there are tabs for 'DONNEES LOGISTIQUES' and 'ALLERGENES', along with a 'Le taux d'acides gras franc dans le...' slider and buttons for 'Oui' and 'Non'.

Cette capture d'écran montre l'outil de contrôle des données produit. Sur la partie gauche, le contenu des pièces jointes est affiché (visuel de l'emballage en haut, fiche technique en bas), sur la droite les données qui ont été transmises par le fournisseur.

FIGURE 33 – Une capture d'écran du PIM

Le socle technologique (et un mot de vocabulaire) Ce logiciel a été construit sur la base de l'outil de gestion de contenu généraliste Nuxeo. Il s'agit d'un outil de GED (Gestion Électronique de Documents), et tous les objets (produits, fournisseurs, articles, ...) qu'il stocke sont nommés « documents ». Dans le cadre du PIM, ce qui est habituellement appelé « document » au sens d'un fichier informatique (document pdf,

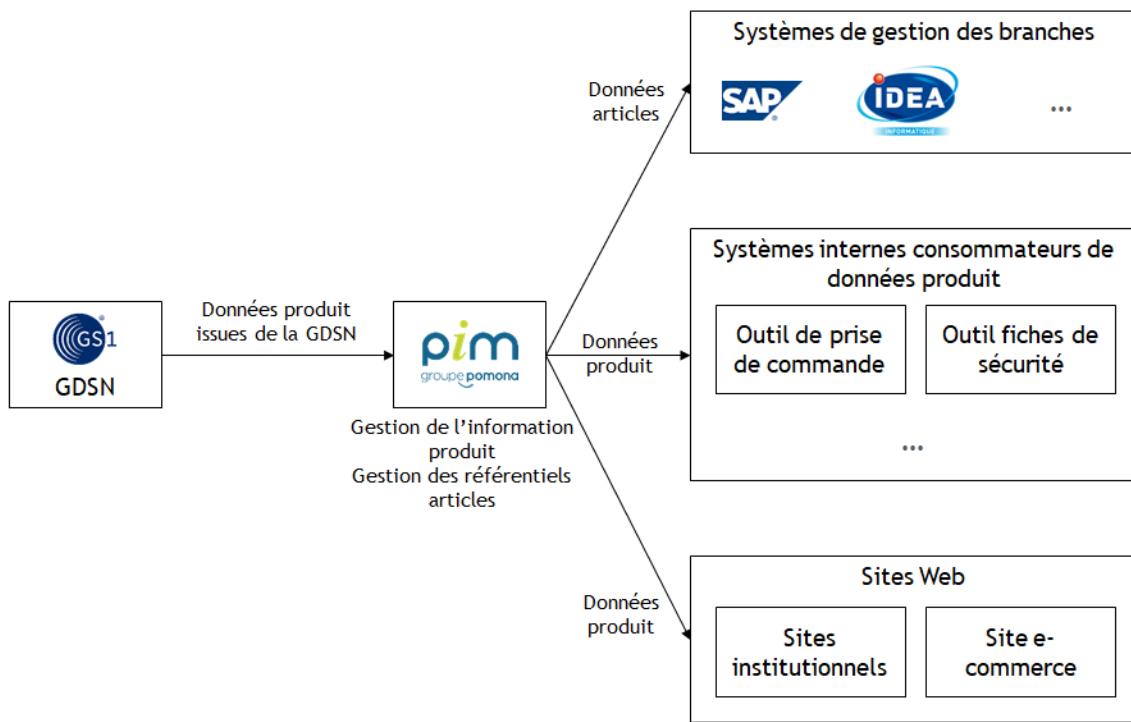


FIGURE 34 – L'intégration du PIM au sein des systèmes du Groupe Pomona

image png, vidéo, ...) est appelé « pièce jointe ». On utilisera ce vocabulaire dans le présent rapport.

Le logiciel Nuxeo est adapté par une société - Keendoo - qui pré-paramètre la solution généraliste Nuxeo pour en faire un outil spécialisé dans la gestion d'information pour les produits alimentaires.

La dernière « couche » de développement a été réalisé de manière spécifique par les équipes de développement du Groupe Pomona.

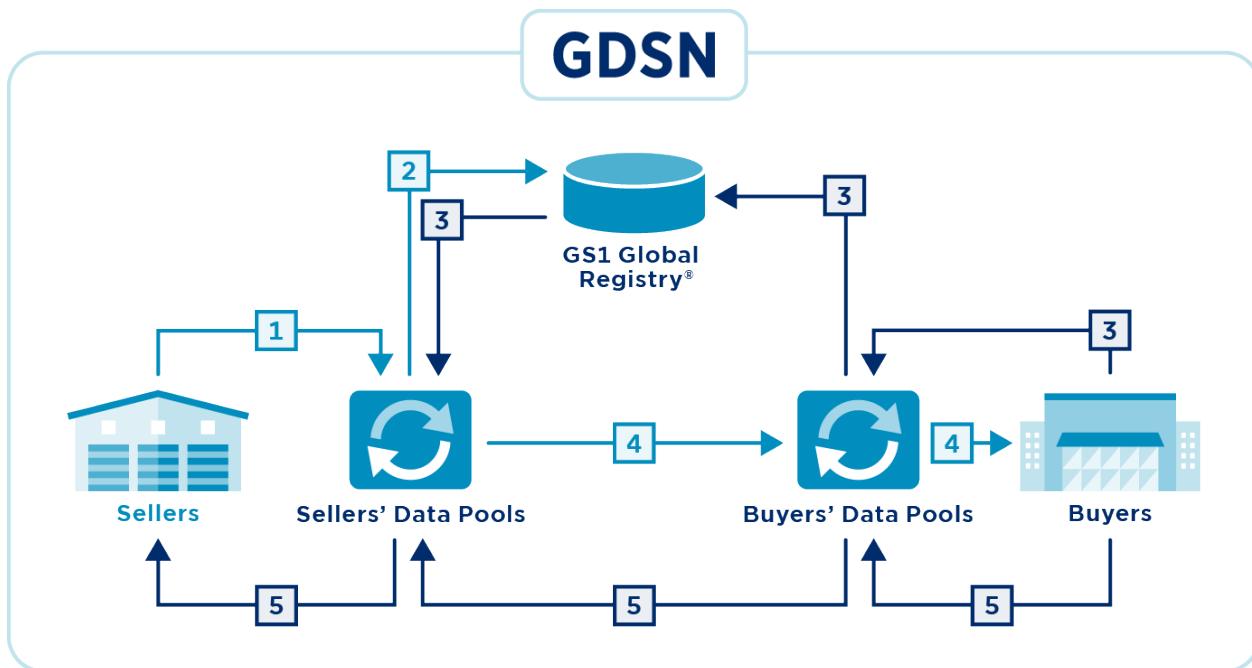
La base de données sous-jacente à l'application est une base NoSQL MongoDB.

La GDSN La GDSN (Global Data Synchronization Network) est un réseau d'échange de données produit entre industriels, distributeurs, restaurateurs, ... Ce réseau est exploité par des opérateurs privés, mais le format et la chorégraphie des échanges ont été standardisés par l'organisme de standardisation GS1. Son schéma de principe est décrit à la FIGURE 35 page 97. Sans rentrer dans le détail, au sein du Groupe Pomona l'utilisation qui en est faite est de récupérer les informations depuis ce réseau d'échange, afin de préalimenter les données produit pour les fournisseurs. Cette fonctionnalité permet de faire gagner du temps aux fournisseurs pour leur éviter une partie de ressaisie, mais également de limiter les erreurs. Toutefois, cette fonctionnalité ne permet pas à elle seule de garantir une parfaite qualité de données. Il s'agit uniquement d'un « tuyau », si les données en entrée ne sont pas correctes, elles ne seront pas correctes en sortie. Pour aller plus loin dans la compréhension de ce réseau, il est possible de consulter les ressources mises en ligne par GS1 [7][8].

L'identification des objets dans le PIM Un dernier point à connaître à propos du PIM, est la manière d'identifier l'ensemble des objets en son sein. Chaque objet géré (ainsi que toute version archivée) porte un identifiant unique, nommé *uid* qui est totalement univoque. Pour la suite, on se basera sur ces uid pour faire référence à des produits stockés dans le PIM. Une illustration est présentée à la FIGURE 36 page 97.

Les API Un des aspects intéressants du PIM pour l'exploitation en masse des données produit, est qu'il expose des API permettant d'aller requérir l'ensemble de son contenu. Cela concerne à la fois les données dites structurées, mais également les pièces jointes. Cela rend les données produit de la branche ÉpiSaveurs bien plus simplement accessibles que celles des autres branches.

La reprise de données initiale et la migration Un point à avoir également en tête est le mode d'alimentation initial du PIM pour ÉpiSaveurs et le processus dit d'enrichissement qui ont été décidés. Les données ont été transférées du système de gestion historique (le GIP, cf. section B.2.3 page 93) vers le PIM, sans transformation métier. Cela signifie qu'il n'y a eu ni correction, ni enrichissement de données lors du chargement initial. Il a été décidé de lancer ces travaux après le démarrage, directement dans le système PIM, en utilisant les fonctionnalités de sollicitation des fournisseurs qu'il propose. La manière de procéder est la suivante :



1. Loading of company data
2. Registering of company data
3. Subscription to seller's data pool
4. Publishing of company data
5. Confirmation receipt of company data

FIGURE 35 – Schéma de principe de la GDSN

The screenshot shows a web-based application for managing products. The URL in the browser bar is <https://produits.groupe-pomona.fr/nuxeo/ui/#!/doc/a2b33d4b-5c39-404c-aa65-a41e87bac7e9>. The page displays a product detail for "PIM'S ORANGE EN PAQUET 150 G LU". The product image is "PiM's Orange en paquet 150 g", the name is "PIM'S ORANGE EN PAQUET 150 G LU", and the ID is "PIMP-000005795". A green "VALIDE" button indicates the status. On the left, there is a sidebar with various icons and navigation links. At the bottom, there are three tabs: "ADMINISTRATION POMONA", "FICHES DE CONTRÔLES", and "IDENTIFICATION PRODUIT".

L'uid du produit affiché est mis en évidence dans l'url

FIGURE 36 – L'uid d'un produit

- les produits qui ont été créés lors de la reprise de données initiale sont envoyés aux fournisseurs pour qu'ils corrigent et complètent les informations produit
- le processus est ensuite le même que pour un nouveau référencement (en particulier, les contrôles et les renvois au fournisseur, tels que présenté à la FIGURE 32 page 92)
- lorsque la validation finale des gestionnaires de référentiels est effectuée, le produit est considéré comme « migré » et ses données sont réputées correctes

Les données du PIM étaient donc incomplètes et vraisemblablement incorrectes au démarrage, et en juin 2020, ces travaux sont toujours en cours. Il existe donc des produits dans le systèmes, pour lesquels la revue et la correction des données n'a pas encore eu lieu.

Annexe C

LE PÉRIMÈTRE PRODUIT

C.1 Les produits non-alimentaires

Un petit aparté est nécessaire concernant les produits non-alimentaires. Si l'essentiel des produits commercialisés par les branches du groupe sont des produits alimentaires, comme évoqué précédemment une partie de l'activité commerciale se fait tout de même autour de produits non-alimentaires. Ces produits restent malgré tout destinés exclusivement aux professionnels des métiers de bouche, et il s'agit de consommables (par opposition à des articles d'électroménager, de la vaisselle non-jetable, ...).

On distingue en général deux catégories de produits non-alimentaires :

- les produits dits « d'hygiène »
- les produits dits « de chimie »

Les produits de chimie regroupent les produits qui doivent faire l'objet d'une fiche de données de sécurité au sens du règlement Européen No 1907/2006 dit « REACH » (Registration, Evaluation, Authorisation and Restriction of Chemicals) [2].

On appelle produits d'hygiène tous les autres produits non-alimentaires. L'appellation « d'hygiène » est donc réductrice, dans la mesure où cette large famille regroupe les consommables de nettoyage (éponges, papiers absorbants, ...) mais également tout type d'autres consommables (serviettes de tables, gobelets en plastiques, pics à brochettes, boîtes de produits à emporter, ...).

La commercialisation de produits non-alimentaires existe au sein du groupe, mais on se focalisera pour la suite sur les produits alimentaires qui reste le cœur de métier.

C.2 Accessibilité de la donnée en fonction des branches

Comme vu à la section B.2.3 page 93, les systèmes d'information associés à la gestion de l'information produit offrent des niveaux d'accès hétérogènes à la donnée produit. Le récapitulatif par branche est le suivant :

ÉpiSaveurs : on peut simplement accéder à l'ensemble des données produit, structurées, non structurées (i.e. textes longs) et pièces jointes

PassionFroid : on a uniquement la possibilité d'exporter manuellement les données structurées articles depuis le système de gestion SAP. Elles permettent de produire quelques analyses quantitatives. Il est difficile de faire des exports en masse de l'outil de gestion de l'information produit GIP (cf. section B.2.3 page 93).

TerreAzur : idem PassionFroid, si ce n'est qu'en plus le système GIP n'est pas utilisé au sein de cette branche.

Délice et Création : le système d'information ne permet pas d'exporter les données et donc de produire des indicateurs détaillés. On peut toutefois avoir des informations quantitatives de la part des opérationnels.

Saveurs d'Antoine : idem Délice et Création

Pomona Suisse : la branche est en cours de structuration, et les référentiels articles ne sont pas partagés entre les succursales. Il n'est pas possible d'obtenir d'information quantitative sur ces données.

Pomona Iberia : idem Pomona Suisse

Pour les analyses quantitatives, on pourra se baser sur des extractions uniquement pour les branches RHD (ÉpiSaveurs, PassionFroid, TerreAzur). L'ensemble des analyses portant sur les branches RHD sont produites sur la base d'extractions de leur système de gestion SAP.

C.3 Analyses quantitatives

Les graphes de cette section ont été produits via le code présenté en annexe, au chapitre G.7 page 218. Les données pour les branches spécialistes (Délice et Création et Saveurs d'Antoine) sont issues d'informations fournies par le métier, hors système. Dans l'ensemble de cette section, on raisonnera à la maille *article* (cf. la définition article vs. produits, présentée à la section B.2.2 page 88).

C.3.1 Comparatifs entre les branches

En termes de volumétrie article (cf. FIGURE 37 page 100, c'est TerreAzur qui possède le référentiel le plus étendu (environ 62 000 articles de marchandises actifs). Cela s'explique par le fait que cette branche commercialise essentiellement des produits bruts, non-préemballés (ex : des cagettes de fruits ou de légumes).

Or, ces produits ne sont pas clairement identifiés, par exemple par un GTIN. Au démarrage de SAP pour cette branche, afin de limiter la charge sur les gestionnaires de référentiels, le parti a été pris de créer en avance de phase l'ensemble des articles susceptibles d'être commercialisés. Cela s'est traduit par la création d'un grand nombre d'articles, du fait de l'application « brutale » de la combinatoire des différents critères pouvant définir un produit. Un exemple (fictif) serait, sur les pommes :

- 8 variétés possibles (Gala, Golden, ...)
- 4 calibres possibles
- 6 conditionnements possibles (plateau 6kg, plateau 4,5kg, ...)
- 2 catégories (I, II)
- 8 origines (France, Espagne, ...)

ce qui donne un total de 3072 articles uniquement sur cette gamme de produits.

Viennent ensuite PassionFroid, et ÉpiSaveurs, qui sont les autres « grosses » branches historiques du Groupe.

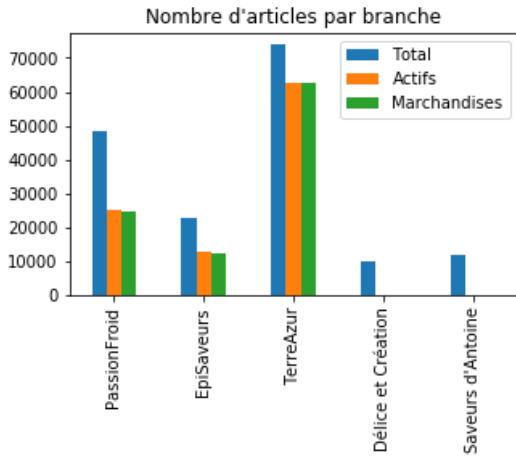


FIGURE 37 – Volumétrie article par branche

Une analyse du recouvrement des référentiels montre que dans l'ensemble, les branches ne travaillent pas les mêmes articles (cf. FIGURE 38 page 101). PassionFroid commercialise certains produits des branches ÉpiSaveurs et TerreAzur, mais cela s'explique par une petite entité luxembourgeoise qui travaille des produits de tout type de stockage. Une réserve toutefois par rapport à cette analyse de recouvrement produit : elle sous-estime vraisemblablement lesdits recouvrements, dans la mesure où la présence de doublons n'est pas prise en compte.

C.3.2 Les grands types de produits

Comme montré à la FIGURE 39 page 102, on voit bien (en plus du fait que les articles étaient peu partagés entre les branches) que :

- les deux types d'articles (négocie et presté) sont utilisés par les 3 branches

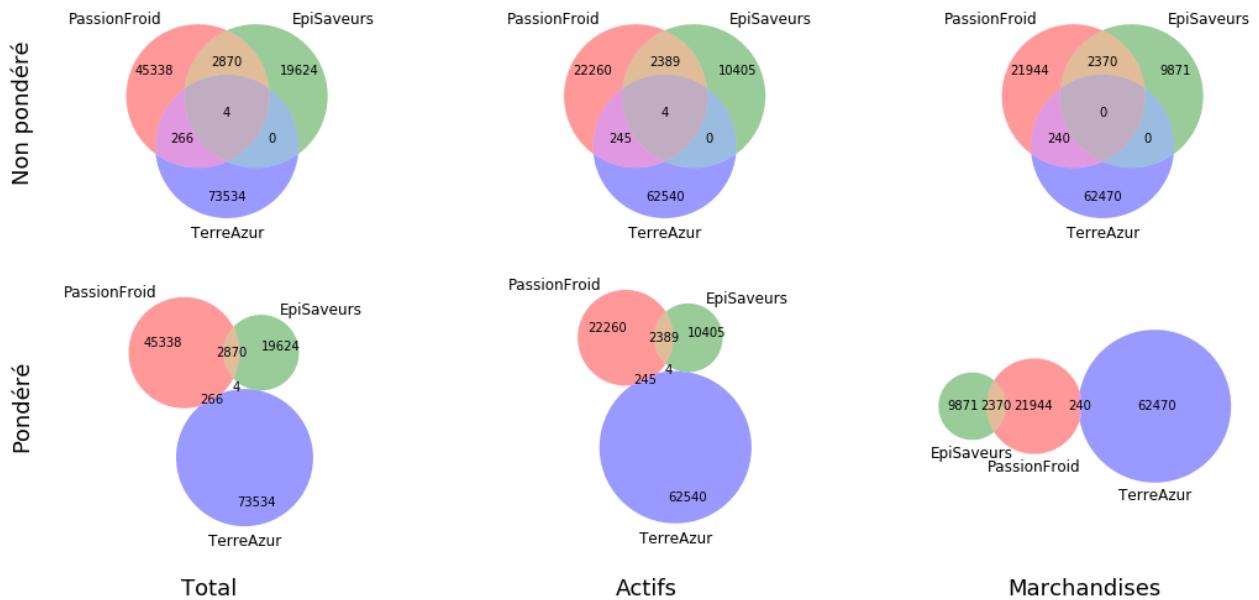


FIGURE 38 – Recouvrements entre branches RHD

- c'est tout de même PassionFroid qui fait l'utilisation majoritaire d'articles prestés
- les branches *ne partagent pas* l'utilisation des autres « catégories » (groupe de marchandises, conditions de stockage, hiérarchie produit, ...)

Les indicateurs sont également récapitulés dans la TABLE 18 page 103.

Possibilité d'extension : - ajouter la vision produit (via les FIA) - faire une sorte d'heatmap qui montre la forte corrélation entre les différentes variables catégorielles (et les définitions associées. Ex : ZELAB + SA = Saurisserie, etc...)

Annexe D

LES DONNÉES UTILISABLES, ISSUES DU PIM

Comme vu au chapitre C page 98, les données produit ne sont simplement accessibles que pour la branche ÉpiSaveurs. On se focalisera donc sur cette branche pour la suite de cette étude, ainsi que sur les produits alimentaires (en excluant donc les produits d'hygiène et de chimie, cf. section C.1 page 98). Contrairement au chapitre précédent, ici on travaillera à la maille *produit* (cf. la distinction produit vs. article section B.2.2 page 88). L'ensemble des

Répartition des articles selon les features catégorielles

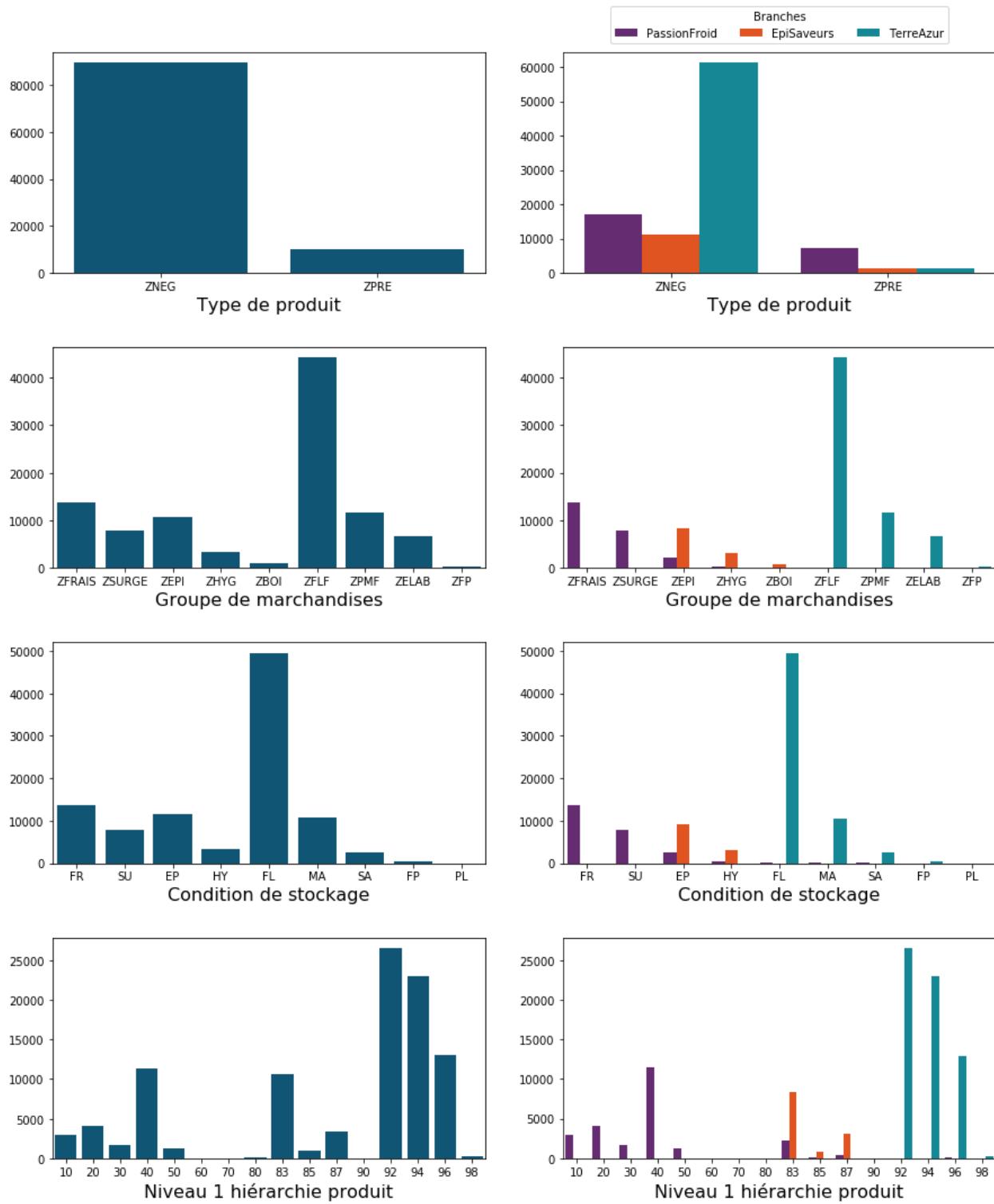


FIGURE 39 – Répartition des articles en fonction des variable catégorielles

Branche Type de produit	PassionFroid	EpiSaveurs	TerreAzur	Total
ZNEG - Article de négoce	17166	11048	61273	89487
ZPRE - Article de prestation	7388	1193	1437	10018
Branche Groupe de marchandises	PassionFroid	EpiSaveurs	TerreAzur	Total
ZSURGE - Surgelés	7756	-	-	7756
ZFRAIS - Frais	13785	6	4	13795
ZEPI - Epicerie	2298	8305	-	10603
ZBOI - Boissons	126	826	-	952
ZHYG - Hygiène	350	3078	-	3428
ZFLF - Fruits et Légumes	4	-	44133	44137
ZPMF - Produits de la mer	142	-	11594	11736
ZELAB - Produits élaborés	91	-	6644	6735
ZFP - Fleurs et plantes	-	-	297	297
ZAUTRE - Autres	2	26	38	66
Branche Condition de stockage	PassionFroid	EpiSaveurs	TerreAzur	Total
SU - Surgelés	7758	-	-	7758
FR - Frais	13781	6	3	13790
EP - Epicerie	2430	9155	-	11585
HY - Hygiène	344	3080	-	3424
FL - Fruits et légumes	78	-	49508	49586
MA - Marée	126	-	10501	10627
FP - Fleurs et plantes	-	-	286	286
SA - Saurisserie	34	-	2408	2442
PL - Publicié	2	-	1	3
Branche Niveau 1 hiérarchie produit	PassionFroid	EpiSaveurs	TerreAzur	Total
10 - Beurre, oeufs, fromage	3010	6	1	3017
20 - Elaborés	4150	2	6	4158
30 - Garnitures et fruits	1701	-	-	1701
40 - Produits carnés	11413	-	-	11413
50 - Produits de la mer	1214	-	2	1216
60 - Consommables	1	-	-	1
70 - Emballage	-	1	-	1
80 - Publicité sur le lieu de vente	34	25	37	96
83 - Epicerie	2306	8296	-	10602
85 - Liquides	135	836	-	971
87 - Hygiène et entretien	348	3075	-	3423
90 - Services	10	-	-	10
92 - Fruits	35	-	26543	26578
94 - Légumes	37	-	22929	22966
96 - Produits de la mer Frais	160	-	12891	13051
98 - Fleurs - plantes	-	-	301	301

TABLE 18 – Utilisation des variables catégorielles article au sein des branches RHD

tableaux et graphes de ce chapitre ont été produit via le notebook "Analyse des données du PIM", présenté en annexe G.8 page 226.

D.1 Données structurées

D.1.1 Description des données structurées

Les données dites structurées sont l'ensemble des données qui peuvent prendre leurs valeurs dans un domaine restreint. Par exemple, ce sont les données booléennes, les choix issus de listes déroulantes, les valeurs numériques... Les principales données structurées pour les produits alimentaires dans le PIM sont :

le code du produit : calculé par le système

le fournisseur : référence croisée vers le code du fournisseur

le type de produit : épicerie, boisson alcoolisée, hygiène, chimie, boisson non-alcoolisée

le GTIN du produit : identifiant numérique unique, utilisé entre autres pour l'étiquetage sous forme de code à barres [9]

le type d'unité de base : paquet, boîte, sachet, rouleau, bouteille, pot, ...

les poids : brut, net, net égoutté (pour les conserves)

le volume : pour les produits liquides

les durées de vie : le type (Date Limite de Consommation ou Date de Durabilité Minimale) et la durée (totale à fin de production, garantie à livraison)

les modes de conservation avant/après ouverture : à température ambiante, au réfrigérateur puis à consommer sous 2 jours, ...

les labels : le(s) label(s) s'appliquant au produit (cf. section B.2.1 page 87)

les régimes particuliers : Halal, Casher, Sans porc, Végétarien, Végétalien, ...

les caractéristiques spéciales : sans OGM, non traité par ionisation

la présence d'allergènes : le niveau de présence de chacun des 14 allergènes réglementés (cf. section B.2.1 page 84) : absence, présence ou traces

les matières grasses utilisées : palme, beurre, coco, tournesol, palmiste, ...

les additifs présents : les codes Exxx et les fonctions des additifs mis en oeuvre [1][18]

les données nutritionnelles obligatoires : pour 100g ou 100mL, valeur énergétique (en kJ et kcal), matières grasses, dont acides gras saturés, Glucides, dont sucres simples, Fibres, Protéines, simplement

les données nutritionnelles facultatives : vitamines, minéraux, omégas, ...

les allégations nutritionnelles : riche en, faible en, sans,... associé à un nutriment défini dans les 2 points précédents

le nutriscore : note allant de A à E, définie dans la loi Santé de janvier 2016

le taux de TVA : un des quatre taux définis dans la réglementation française

le code nomenclature douanière : code identifiant les marchandises défini par les douanes pour la Déclaration d'Échange de Biens [5]

le pays d'origine pour la DEB : le pays d'origine à déclarer dans la Déclaration d'Échange de Biens [5]

les informations logistiques : il s'agit du plan de conditionnement et de palettisation du produit. Elles regroupent les différents niveaux et les quantités pour passer de l'un à l'autre (ex : 3 boites dans un cartons, 64 cartons dans une palette), les poids et dimensions de ces niveaux logistiques, leurs GTIN,

...

D.1.2 Analyse de ces données structurées

Le statut des produits

Comme cela a été présenté à la FIGURE 32 page 92, le processus de gestion de l'information produit fait intervenir de multiples acteurs et peut donc être assez long. Un produit qui n'est pas arrivé au bout du processus et fait l'objet de contrôles et validations successives n'est pas réputé comme portant des informations correctes. Dans le PIM, la « localisation » du produit dans le processus est portée par son statut (c'est une simple donnée catégorielle). Seuls les produits au statut « Validé » sont considérés comme portant des données valides. La répartition des produits pour chacun des statuts est présenté à la FIGURE 40 page 105.

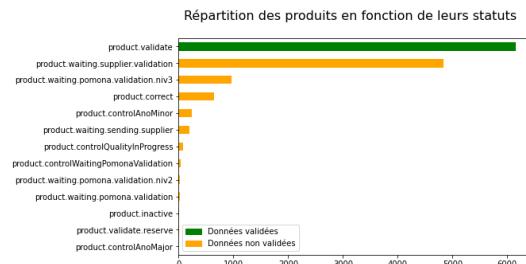


FIGURE 40 – Répartition des produits par statut

state
product.validate
product.waiting.supplier.validation
product.waiting.pomona.validation.niv3
product.correct
product.controlAnoMinor
product.waiting.sending.supplier
product.controlQualityInProgress
product.controlWaitingPomonaValidation
product.waiting.pomona.validation.niv2
product.waiting.pomona.validation
product.inactive
product.validate.reserve
product.controlAnoMajor

TABLE 19 – Répartition des produits par statut

De plus, comme cela a été présenté dans la section migration B.2.3 page 96, certains produits ont fait l'objet d'un contrôle et d'une correction après le démarrage, mais d'autres non. Il est nécessaire de prendre également cet aspect en compte si l'on souhaite avoir la vision des produits qui sont censés avoir des données correctes. La détermination du statut des produits vis-à-vis du processus de migration se fait au travers de facettes :

- les produits qui ont été créés lors de la reprise de données initiale portent une facette "beginningMigration"
- les produits créés dans le PIM après le démarrage (cas des nouveaux référencements) ne portent pas cette facette

- les produits issus de la migration (portant la facette "beginningMigration") sont envoyés aux fournisseurs pour qu'ils complètent les données
- une fois que le processus de validation de l'information produit est terminé (validation des gestionnaires de référentiels), une facette "endMigration" est apposée sur le produit

La répartition des produits fonction de leurs statuts de migration est présenté à la TABLE 20 page 106. Les produits portant des données correctes, du point de vue de la migration, sont :

- ceux qui ont été créés après le démarrage (donc qui ont été créés avec le processus et les règles de gestion cible)
- ceux qui ont été créés à la reprise, mais qui sont identifiés comme ayant terminé le processus de migration (portant la facette de fin)

	Facette fin de migration : Non	Facette fin de migration : Oui
Créé après le démarrage	1680	0
Créé au démarrage	7210	4345

TABLE 20 – Répartition des produits par statut de migration

Si on combine le statut des produits avec leurs statut de migration, on peut calculer un statut « Données valides » qui doit permettre d'identifier des produits avec des données de qualité. C'est ce statut qui sera utilisé dans les paragraphes suivants, lorsqu'on fera des analyses relatives à la qualité des données. On nommera ces statuts « En qualité » et « Hors qualité » pour la suite de ce document. La volumétrie des produits en fonction de chacun de ces statuts est présentée à la TABLE 21 page 106.

	Répartition produit par qualité
Hors qualité	8722
En qualité	4513

TABLE 21 – Répartition des produits par « qualité des données »

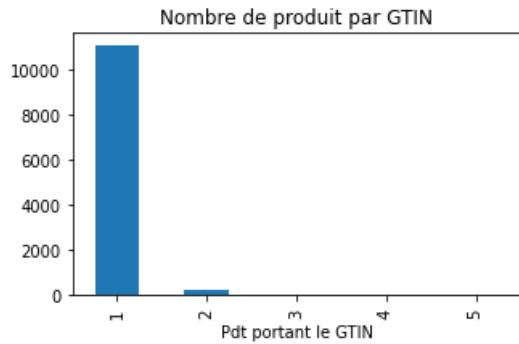
Données d'identification

Un exemple et une description statistique de ces données est présentées aux TABLE 23 page 108 et TABLE 24 page 108. On peut en tirer les conclusions suivantes :

- Les codes produits sont bien des identifiants uniques des produits : on a autant de valeurs uniques que de valeurs renseignées
- Une proportion importante de produits (91%) porte un GTIN (cf. [9]), ce qui reflète une « maturité » des filières produits avec lesquelles travaille ÉpiSaveurs.
- Ces GTIN sont également censés être des identifiants uniques des produits. Néanmoins, comme présenté à la FIGURE 41 page 107, il peut arriver que le même GTIN soit présent sur plusieurs produits. Quelques

illustrations et explications sont présentées en annexe, dans le notebook d'analyse des données.

- La distribution numérique des produits par fournisseur est importante. Il existe plus de 600 fournisseurs pour les 13000 produits, et l'essentiel des fournisseurs n'ont qu'un nombre très limité de produits. On peut avoir une vision à la FIGURE 42 page 107 ou à la FIGURE 2 page 17.



Pdt portant le GTIN	Nb de GTIN
1	11089
2	228
3	21
4	20
5	1

TABLE 22 – Nombre de produits par GTIN

FIGURE 41 – Nombre de produits par GTIN

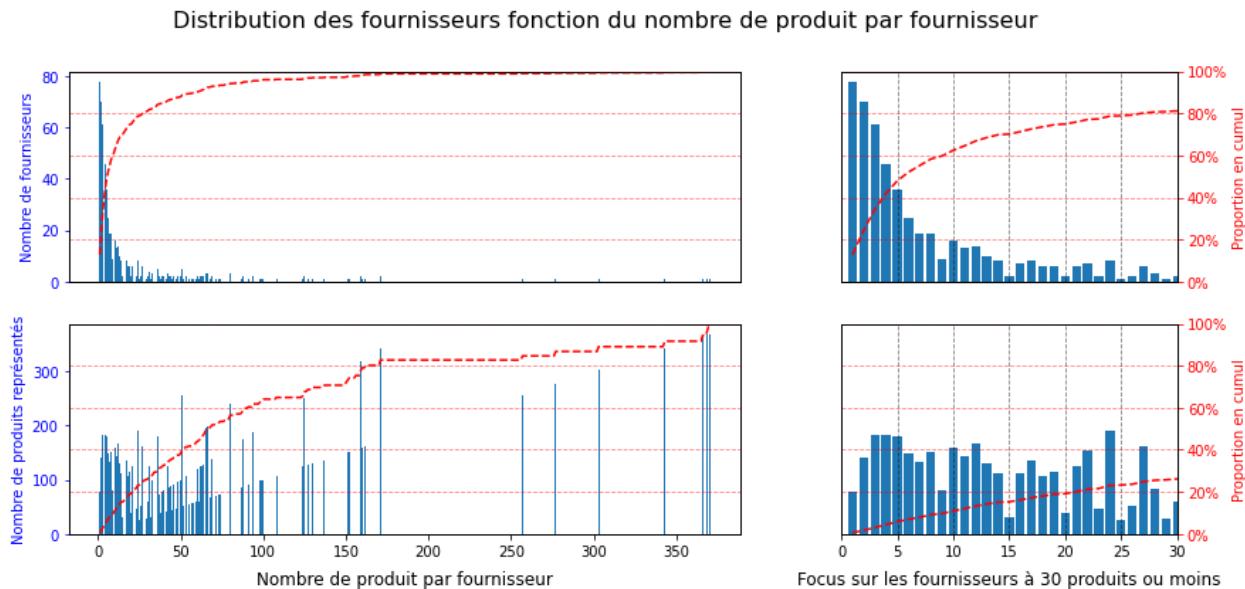


FIGURE 42 – Distribution des fournisseurs par nombre de produits

uid	Code produit	Code fournisseur	Type de produit	GTIN	data_ok
e24ff685-40da-4fc8-8454-ec70282552be	PIMP-00000009877	PIMF-0000000483	alcoholicDrink	3387210001261	True
a4e317c3-8fd6-441d-82fc-13f528ba137	PIMP-0000013133	PIMF-0000000283	hygiene	3342690134229	False
729883ee-8d83-45a3-a7c7-015b22473ac9	PIMP-0000007508	PIMF-0000000378	grocery	3077317520227	True
4312cd74-4708-4216-a56e-fbc404c5afa0	PIMP-0000012015	PIMF-0000000596	nonAlcoholicDrink	3230140003938	True
2581a202-7737-448f-9e2d-eb5a02472101	PIMP-0000008730	PIMF-0000000311		5449000227089	True

TABLE 23 – Exemples de codes d’identification

data_ok	Code produit	Code fournisseur	Type de produit	GTIN
Hors qualité	count	8722	8722	7663
	unique	8722	551	7445
	top	PIMP-0000008549	PIMF-0000000250	
En qualité	freq	1	313	
	count	4513	4513	67
	unique	4513	339	4382
	top	PIMP-0000008225	PIMF-0000000179	4062
	freq	1	305	285

TABLE 24 – Description des codes d’identification sur le dataframe

uid	Unité de base	Poids net	Poids brut	Poids net égoutté	Volume	data_ok
e24f6685-40da-4fc8-8454-ec70282552be	BIB	5.00	5.300	-	5.0	True
a4e317c3-f8d6-441d-82ff-13f5288ba137	SAC	125.00	0.130	-	-	False
729883ee-8d83-45a3-a7c7-015b22473ac9	BT.	0.50	0.825	-	-	True
4312cd74-4708-4216-a56e-fbc404c5afa0	SEA	1.10	1.242	-	-	True
2581a202-7737-448f-9e2d-eb5a02472101	BT.	0.51	0.530	-	0.5	True

TABLE 25 – Exemples de dimensions

data_ok	Unité de base	Poids net	Poids brut	Poids net égoutté	Volume
count	8722	8341.000	8341.000	728.000	866.000
unique	29	-	-	-	-
top	BTE	-	-	-	-
freq	2260	-	-	-	-
mean	-	3.616	3.304	1.567	5.330
std	-	73.871	52.238	1.057	43.624
min	-	0.000	0.000	0.000	0.000
25%	-	0.500	0.550	0.500	0.500
50%	-	1.000	1.149	1.600	0.900
75%	-	3.000	3.368	2.458	4.124
max	-	4900.000	4730.500	6.050	1000.000
count	4513	4513.000	4513.000	299.000	1053.000
unique	26	-	-	-	-
top	SAC	-	-	-	-
freq	1075	-	-	-	-
mean	-	2.174	2.394	1.246	9.810
std	-	3.127	3.324	1.273	98.786
min	-	0.000	0.000	0.000	0.000
25%	-	0.440	0.514	0.350	0.500
50%	-	1.000	1.070	0.560	0.975
75%	-	3.000	3.120	2.210	2.000
max	-	33.474	40.589	10.000	3100.000

TABLE 26 – Description des dimensions sur le dataframe

uid	Durée de vie totale	Durée minimale restante	Type de conservation	Conservation avant ouv.	Conservation après ouv.	Température	data_ok
e24f6685-40da-4fc8-8454-ec70282552be	-	-	AM	coolAndDryPlace	-	True	
a4e317c3-88d6-441d-82ff-13f5288ba137	-	-	AM	-	-	False	
729883ee-8d83-45a3-a7c7-015b22473ac9	-	-	AM	ambientTemperatureoolAndDryPlace	-	True	
4312cd74-4708-4216-a56e-fbc404c5afa0	365.0	243.0	AM	noConcerned	coldIfO3Months	-	True
2581a202-7737-448f-9e2d-eb5a02472101	180.0	120.0	AM	ambientTemperature notConcerned	-	True	

TABLE 27 – Exemples de conservation

data_ok	Durée de vie totale	Durée minimale restante	Type de conservation	Conservation avant ouv.	Conservation après ouv.	Température
count	5601.000	6068.000	8320	6369	6348	8
unique	-	-	2	7	18	8
top	-	-	AM	ambientTemperature	coolAndDryPlace	ambianté
freq	650.149	345.489	8299	5731	2781	1
mean	490.058	371.169	-	-	-	-
std	0.000	0.000	-	-	-	-
min	360.000	160.000	-	-	-	-
25%	540.000	270.000	-	-	-	-
50%	900.000	480.000	-	-	-	-
75%	9999.000	9999.000	-	-	-	-
max	3342.000	3408.000	4513	3614	3607	13
count	-	-	2	7	18	5
unique	-	-	AM	ambientTemperature	coolAndDryPlace	<10°C
top	-	-	4500	2532	1733	5
freq	655.031	362.071	-	-	-	-
mean	482.709	395.676	-	-	-	-
std	0.000	0.000	-	-	-	-
min	360.000	180.000	-	-	-	-
25%	540.000	360.000	-	-	-	-
50%	730.000	480.000	-	-	-	-
75%	9999.000	9999.000	-	-	-	-
max	-	-	-	-	-	-

TABLE 28 – Description des conservations sur le dataframe

D.2 Données non structurées

D.2.1 Les libellés

Plusieurs libellés permettent de faire référence à chaque produit, avec des usages distincts :

Libellé temporaire unité de besoin : il s'agit simplement d'un libellé qui est choisi par la personne à l'origine de la demande de référencement produit (souvent l'acheteur, cf. FIGURE 32 page 92), afin que le fournisseur comprenne sur quel produit porte le référencement.

Désignation du produit fournisseur : c'est le libellé qui est donné par le fournisseur afin d'identifier simplement son produit.

Code article interne fournisseur : c'est l'identifiant du produit, dans le système de gestion du fournisseur. Il s'agit généralement de codes, avec autant de formats distincts que de fournisseur venant contribuer.

Marque commerciale du produit : il s'agit du nom de la marque commerciale du produit.

Dénomination réglementaire : c'est le libellé qui doit décrire de manière neutre le produit, sans notion liée au marketing (telle que la marque, entre autres), et qui doit obligatoirement figurer sur l'emballage du produit.

Des exemples de libellés sont présentés à la TABLE 29 page 111.

Libellé temporaire	Désignation produit fournisseur	Code interne fournisseur	Marque commerciale	com-	Dénomination réglementaire
VIN BEAUJO NOUVEAU(75CLX6) D MOREL	BEAUJOLAIS NOUVEAU	BN 2019	DOMINIQUE MOREL		VIN AOP
COCKTAIL FRT NATUREL BTE 5/1X3 DUNE	COCKTAIL DE FRUITS A L'EAU - FORMAT 5/1	COCEODUR	DUNE Restau- ration	COCKTAIL FRUITS A L'EAU	DE
Sauce barbecue en coupelle 20 g GYMA	Sauce barbecue en coupelle 20 g GYMA	400937	GYMA		Sauce Barbecue
Confiture abricot en bocal 450 g VALADE	Confiture abricot en bocal 450 g VALADE	PF100006	VALADE EN CORREZE		Confiture d'abricots
Confiture extra de figue en bocal verre	BONNE MAMAN CONF FIGUES VIOLETTTE 370	20000929	BONNE MAN		Confiture Extra de Figues violettes.

TABLE 29 – Exemples de libellés produit

D.3 Analyse qualitative des données

D.3.1 Évaluation de la qualité des données

Montrer qu'un sondage basique fait que la qualité actuelle est perfectible

D.3.2 Types de pdf possédés

Une information intéressante est de faire le point sur la possibilité ou non d'extraire les textes des pdf possédés. En effet, cela permet d'en exporter le contenu avec plus de simplicité et de fiabilité qu'en ayant recours à des fonctionnalités d'OCR (Optical Caracter Recognition [20]). Pour obtenir cet indicateur, on a procédé de la manière suivante :

- extraction des fichiers pdf correspondant aux étiquettes et aux fiches techniques des produits appartenant à l'échantillon de données manuellement étiquetées (la ground truth, cf. la section à venir 2.4 page 13)
- application des traitements d'extractions de textes depuis les pdf sur l'ensemble de ces fichiers
- comptage de ceux parmi eux qui apparaissent « vides », c'est à dire dont le retour de ces fonctionnalités les affiche comme contenant uniquement des whitespaces (espaces, tabulation, retours à la ligne, « form feed caracters », ...)

L'application de cette méthode est présentée à la fin du notebook « Génération de l'échantillon de données manuellement étiquetées » présenté en annexe G.1 page 141. Les résultats de cette analyse sont présentés à la TABLE 5 page 17.

On en déduit :

- que la très grande majorité des fiches techniques ont un contenu textuel extractible (96%)
- que les étiquettes, en plus d'être moins représentées que les fiches techniques, ont pour les deux tiers d'entre elles un contenu qui n'est pas exploitables par les outils de pdfmining

Annexe E EXEMPLES DE PIÈCES JOINTES ET GROUND TRUTH

E.1 Fiches techniques

E.1.1 Fiche technique sel Cerebos



Fiche technique

CEREBOS® Sel Gros Alimentaire



Version 1.9

Page 1 / 1

date d'impression: 26.10.2010

No.-CAS:	7647-14-5	No.-EINECS:	231-598-3
Apparence:			produit blanc, cristallin
Analyses chimiques	Spécification	Typique	Méthodes
• Chlorure de sodium	> 99,8 %	99,9 %	ASTM 534-98
• Teneur en eau	< 0,1 %	0,02 %	ISO 2483
• Insolubles dans l'eau	< 0,01 %	0,005 %	ISO 2479
• Anti-agglomérant E 535	< 20 mg/kg		EuSalt AS 004
Granulométrie	Typique	Méthodes	
• > 3,15 mm	1 %	EN 1235	
• 1,00 - 3,15 mm	89 %		
• < 1,00 mm	10 %		
Propriétés physiques:		Méthodes	
• Masse volumique apparente	1.100 - 1.300 kg/m³	EN 1236	
Sur demande:			
• Iode (sous la forme de NaI) exprimé en I	15 - 20 mg/kg	EuSalt AS 002	
• Fluor (sous la forme KF)	250 mg/kg	EuSalt AS 017	
• Législation nationale: 212,5 - 287,5 mg F/kg			
Réglementation sur les denrées alimentaires, Impuretés et contaminants:			
Conforme au CODEX ALIMENTARIUS.			
Domaine d'application			
Sel gros de table ou de cuisine de qualité alimentaire pouvant être supplémenté en iode ou en iodure et fluor.			
Site de conditionnement			
Salines Cérébos et de Bayonne à Dombasle (France 54) et à Mouguerre (France 64) -			
Groupe esco			
Stockage			
Il est conseillé de ne pas gerber les palettes. Le CHLORURE DE SODIUM étant un produit hygroscopique, tout conditionnement ouvert devra être stocké à l'abri de l'humidité.			
Moyens de livraison			
• sur demande			

Les données précédentes résultent de nos contrôles qualité. Ces données ne dispensent pas l'utilisateur d'un contrôle à réception et ne sont pas forcément des garanties de vente. L'utilisateur est seul responsable du choix du produit en fonction de l'application souhaitée

esco - european salt company GmbH & Co.KG
 Headquarters • Landschaftstraße 1 • 30159 • Hannover • Allemagne • +49-(0)511-85030-0 ...-131
 esco benelux nv • Park Lane, Culliganlaan 2G bus 1 • B-1831 • Diegem • Belgique • +32-2711-0160 ...-0161
 esco france s.a.s • 49 Avenue Georges Pompidou • F-92693 • Levallois-Perret Cedex • France • +33(0)1.49.64.59.00 ...-1.49.64.59.10
 Vatel S.A. • Apartado 211-Sobralinho • P-2616-956 • Álverca • Portugal • +35-1219-5184-20 ...-39
 esco Espagne S.L. • Joan d'Austria, 39-47 • 08005 • Barcelona • Espagne • +34 (93) 2247238 ...+34 (93) 2214193
 esco Nordic AB • Drakegatan 10 • 401 23 • Göteborg • La Suède • +46-31 773 70-01 ...-02
 www.esco-salt.com Certification EN ISO 9001:2008

E.1.2 Fiche technique olives Valtonia

copram	FICHE TECHNIQUE	Date : 10/12/2019	V 1.2
	OLIVES NOIRES CONFITES DENOYAUTEES 4/4 VALTONIA		

INGREDIENTS :	Olives, eau, sel, stabilisateur de couleur : E579
---------------	---

MARQUE :	VALTONIA
BOITAGE :	4/4
ORIGINE :	Espagne
EAN 13 :	3061435001137
EAN colis :	3061435101134

CARACTERISTIQUES PHYSICO-CHIMIQUES ET ORGANOLEPTIQUES	
PH :	5.5 à 8.0
TAUX DE SEL :	2.0 à 3.5 %
Calibre :	30/33 unités / 100 g d'olives entières
ODEUR ET SAVEUR :	Franche et caractéristique
COULEUR :	Noire à robe de moine
PRÉSENCE DE NOYAUX :	1 noyau pour 100 fruits 2 fragments pour 100 fruits
DEFAUTS AUTRES CRITERES	Se référer au code des olives

CARACTERISTIQUES MICROBIOLOGIQUES	
Produit conforme à l'arrêté d'octobre 1997 relatif au contrôle de la stabilité des produits appétisés et assimilés (Norme AFNOR NF V08-401).	

VALEURS NUTRITIVES MOYENNES POUR 100 G DE PRODUIT EGOUTTE								
Kj	Kcal	Protéines	Matières grasses	Dont AGS	Glucides	Dont sucres	Sel	Fibres
515	125	0.5 g	13.0 g	2.2 g	0 g	0 g	2.0 g	3.0 g
INFORMATIONS COMPLEMENTAIRES								
Présence d'allergènes :	Non			Pesticides :	Conforme à la législation en vigueur.			
OGM et ionisation :	Absence			Métaux lourds :	Conforme à la législation en vigueur.			

CONDITIONNEMENT		
Contenance :	Poids net :	Poids net égoutté :
850 ml	800 g	360 g
Nombre de boîtes / carton :	Nombre de cartons /couche :	Nombre de couches / palettes :
6	12	12

CONSERVATION	
DLUO avant ouverture	3 ans. Stocker à température ambiante à l'abri de l'humidité
Après ouverture	au réfrigérateur dans un récipient alimentaire avec son liquide de couverture une semaine

Les informations contenues dans cette fiche sont celles dont nous disposons à la date de validation. Elles sont donc susceptibles d'être modifiées ultérieurement. Pour toute réclamation ou demande d'information merci de préciser impérativement le lot et/ou date de fabrication portés sur le couvercle.

Copram - Société de commercialisation des produits alimentaires du Maroc
 17, Quatrième Rue-Zone industrielle B.P. 168 – 13745 VITROLLES cedex – France
 RC Salon B 315 712/299 - Siret : 315 712 299 00017
 NAF : 511N - N° T.V.A. FR 03 315 712 299

E.1.3 Fiche technique Panna Cotta Nestlé



**Panna Cotta
NESTLÉ Docello®
Etui de 600 g (2 x 300g)
pour 50 portions**



CODE EAN
9002100034771

DESCRIPTION DU PRODUIT

Préparation en poudre pour Panna Cotta.

BÉNÉFICES CLÉS DU PRODUIT

Avec arôme naturel.
Facile à mettre en œuvre et à personnaliser.

INGRÉDIENTS

Sucre, dextrose, maltodextrine, stabilisants (E460, E450, E516, E401, E404), épaississant (E407), arôme naturel (**lait**).
Peut contenir : **fruits à coque, œuf, soja et gluten.**

ALLERGÈNES MAJEURS

Conformément aux réglementations vigueur :

- Allergènes présents dans la recette ou dans un de ses ingrédients : **lait**
- Allergènes potentiellement présents : **fruits à coque, œuf, soja et gluten.**

Merci de vérifier ces informations sur l'étiquette de votre produit, car seules celles-ci font foi.

ENGAGEMENT QUALITÉ

NESTLÉ a un système de management de la Qualité certifié par les normes ISO 9001 et FSSC 22000.
Etiquetage conforme à la réglementation en vigueur sur les OGM.
Ce produit ne contient pas d'ingrédients ionisés.

MODE D'EMPLOI

1. Porter à ébullition le mélange de lait et de crème liquide.
 2. Hors du feu, verser en pluie tout en fouettant la préparation pour Panna Cotta puis mélanger jusqu'à parfaite homogénéisation. Porter à nouveau le mélange à ébullition.
 3. Verser la préparation dans des ramequins, faire refroidir puis stocker en chambre froide (entre 0°C et +3°C). Servir frais.
- Suggestion: pour une Panna Cotta encore plus onctueuse vous pouvez réduire le dosage à 130g par litre de liquide.

DOSAGES

Produit déshydraté	Dosages			
	Base	Lait	Crème liquide	Nombre de portions (90g)
Panna Cotta	150 g	0,5 L	0,5 L	12
	300 g	1 L	1 L	25
	600 g	2 L	2 L	50

UTILISATION

Pour une texture plus souple et un rendement amélioré : un sachet (300 g) + 2 L de crème liquide + 1 L de lait = 36 portions (90 g).

Idéal dans la composition d'un café gourmand.

Facilement personnalisable, ce dessert se consomme nature ou parfumé (thé vert, cardamome, badiane, pulpe de fruits...), mais également accompagné de fruits frais, de Sauces Desserts NESTLE Docello®, de coulis de fruit...



Panna Cotta
NESTLÉ Docelio®
Etui de 600 g (2 x 300g)
pour 50 portions

Nestlé
docelio®

CODE EAN
9002100034771

DÉCLARATION NUTRITIONNELLE

	Pour 100 g	Par portion (90 g)*
Énergie	1563 kJ 368 kcal	811 kJ 195 kcal
Matières grasses	0 g	14 g
- dont acides gras saturés	0 g	9,0 g
Glucides	91 g	14 g
- dont sucres	86 g	14 g
Fibres alimentaires	2,1 g	0,3 g
Protéines	0 g	2,2 g
Sel	0,87 g	0,19 g

*préparée avec du lait demi-écrémé et de la crème liquide (34% de matières grasses)

Bénéfices nutritionnels

Ce produit entre dans la catégorie GEMRCN** des desserts à limiter à hauteur de 3/20 repas maximum.

3/20 repas maximum

**Groupe d'Étude des Marchés Restauration Collective et Nutrition

AVANTAGES ET BENEFICES DU PRODUIT

Crémeux et fondant, ce dessert est facile et rapide à mettre en œuvre.

CONSERVATION - STOCKAGE

Durabilité minimale : 24 mois

À conserver au sec et à l'abri de la chaleur dans son emballage d'origine. Bien refermer après chaque utilisation.

DONNÉES LOGISTIQUES

	Type UC / UD	Code EAN	Poids Net	Poids Brut	Dimensions (L x l x H) en mm
Unité consommateur (UC)	Boîte	9002100034771	600 g	691 g	80 x 185 x 200
Unité de distribution (UD)	Carton	9002100034788	3,6 kg	4,49 kg	388 x 253 x 216
Palette - Gerbabilité : OUI	Palette	7613033074578	129,6 kg	187 kg	1200 x 800 x 1014

Codes internes Nestlé			Code Douanier	Pays de production	Nbre UC par UD	Nbre UD par Couche	Couches par Palette	Nbre UD par Palette	Nbre UC par Palette
M74C/09	12202814	43825851	2106909855	Serbie	6	9	4	36	216

Nestlé France S.A.S. Noisiel 542 014 428 RCS MEAUX, NOISIEL © Reg. Trademark of Société des Produits Nestlé S.A.



Nestlé
PROFESSIONAL

Créateur de Solutions Culinaires & Boissons

Version du 05/07/2018

Les dernières mises à jour sont disponibles sur notre site internet www.nestleprofessional.fr

Page 2 / 2

E.1.4 Fiche technique confiture Andros



Route de Oinville
28 700 AUNEAU
Tél : + 33 2 37 33 16 33 - Fax : + 33 2 37 33 16 91



	05/01/2017
	-

DENOMINATION COMMERCIALE
Colis assorti de 60 bocaux de confitures et marmelade 30g Bonne Maman
15 Confiture Myrtilles et Cassis + 15 Confiture Fraises et Groseilles + 15 Confiture Abricots et Pêches + 15 Marmelade Oranges et Mandarines

DENOMINATION LEGALE DE VENTE
Confiture de Myrtilles et de Cassis - Confiture de Fraises et de Groseilles - Confiture d'Abricots et de Pêches - Marmelade d'Oranges et de Mandarines

CARACTERISTIQUES			
Variété(s) :	myrtille cassis, fraises groseilles, abricots pêches, oranges mandarines		
Conservation :	A conserver au frais après ouverture.	Date de durabilité minimale :	Voir sur unité de vente
Gencod :	3 60858 082942 3	Code article :	50082942
	Confiture Myrtilles et Cassis: 20079939		
Codes douaniers :	Confiture Fraises et Groseilles: 20079933		
	Confiture Abricots et Pêches: 20079939		
	Marmelade Oranges et Mandarines: 2007993110		
Confiture de myrtilles et de cassis			
Confiture de fraises et de groseilles			
Confiture d'abricots et de pêches			
Marmelade d'oranges douces et de mandarines			

ARGUMENTAIRE PRODUIT
Les confitures Bonne Maman duo, des alliances subtiles où le deuxième fruit vient rehausser la saveur et la gourmandise du premier fruit

VALEURS NUTRITIONNELLES MOYENNES													
		Myrtilles Cassis			Fraises Groseilles			Abricots Pêches			Oranges Mandarines		
		pour 100 g	pour 30 g	% AR*	pour 100 g	pour 30 g	% AR*	pour 100 g	pour 30 g	% AR*	pour 100 g	pour 30 g	% AR* par ration
ENERGIE	En kJoules (kJ) :	1016	305	4%	1018	305	4%	1031	309	4%	1022	307	4%
	En kcalories (kcal) :	239	72	4%	240	72	4%	243	73	4%	241	72	4%
Matières grasses (g) :		0,2	0	0%	0,3	0	0%	0,2	0	0%	0,1	0	0%
Dont Acides Gras saturés		0	0	0%	0	0	0%	0	0	0%	0	0	0%
Glucides (g) :		58	17	7%	58	17	7%	59	18	7%	59	18	7%
Dont sucres:		58	17	19%	58	17	19%	59	18	20%	59	18	20%
Fibres (g):		1,8	1	—	1,5	0	—	1,3	0	—	1	0	—
Protéines (g):		0,5	0	0,3%	0,5	0	1,0%	0,6	0	0,4%	0,4	0	0,2%
Sel (g):		0	0	0%	0	0	0%	0	0	0%	0	0	0

* Apports de Référence (Apports de référence pour un adulte type (8400kJ/2000kcal)). Ces valeurs et les portions peuvent varier selon l'âge, le sexe et l'activité physique

CARACTERISTIQUES PHYSICO - CHIMIQUES et ORGANOLEPTIQUES				
Parfums	Myrtilles Cassis	Fraises Groseilles	Abricots Pêches	Oranges Mandarines
Texture	onctueuse			gélifiée
Brix	60 ° brix			
Goût	Caractéristique des fruits			
Couleur	noire	rouge vif	orange	orange
pH	3,00 à 3,4	3,00 à 3,15	3,20 à 3,40	3,00 à 3,15

MENTIONS COMPLEMENTAIRES				
Allergène(s) obligatoire(s) présent(s) :	NON			
Allergène(s) obligatoire(s) pouvant être introduit(s) de manière non intentionnelle :				
Présence OGM, ionisation :				
Les produits ANDROS sont exempts :				
- d'Organisme Génétiquement Modifié et / ou d'ingrédients provenant d'Organisme Génétiquement Modifié				
- d'ingrédient ionisé y compris dans les matières premières et les ingrédients utilisés.				

RECOMMANDATIONS				
GEMRCN	—	Programme 1 fruit pour la récré		

CONDITIONNEMENT et PALETTISATION				
Bocal verre 44 ml / Capsule métallique TO 48 BM				
		Unité	COLIS	PALETTE
Format :	1 colis assorti de 60 pots de 30 g			
GENCOD :	3 60858 082942 3			03 60858 814419 1
Poids net (Kg) :	1,800			280,800
Poids brut (Kg) :	4,843			778,000
Largeur (mm) :	236			800
Longueur (mm) :	146			1200
Hauteur (mm) :	219			1459
Nombre de pièces :				156
Nombre de colis :				156
Nombre de couches :				6
Nombre de colis par couche :				26

COMMENTAIRES :				
En cas d'incohérence entre la fiche technique et l'emballage, veuillez noter que seul l'emballage fait foi.				

E.1.5 Fiche technique ciboulette La case aux épices

 <p>LA CASE AUX ÉPICES</p>	FICHE TECHNIQUE CIBOULETTE TUBULAIRE FLAPPERS		117801 Indice : f Date : 20/11/2014 Page : 1 / 2				
	DESCRIPTION DU PRODUIT						
 <p>CARTONS de 8 Flapper's</p>		<p>Liste des ingrédients</p> <p>Ciboulette</p> <p>Caractéristiques organoleptiques</p> <p>La Ciboulette tubulaire est constituée de feuilles séchées de l'Allium schoenoprasum.</p>					
<p>Dénomination légale</p> <p>Ciboulette tubulaire</p>							
<table border="1"> <thead> <tr> <th>Conservation : DDM (Date de Durabilité Minimale)</th> <th>Commentaires</th> </tr> </thead> <tbody> <tr> <td>DDM : 36 mois DDM minimum à réception : 18 mois</td> <td>Origine : Chine</td> </tr> </tbody> </table>				Conservation : DDM (Date de Durabilité Minimale)	Commentaires	DDM : 36 mois DDM minimum à réception : 18 mois	Origine : Chine
Conservation : DDM (Date de Durabilité Minimale)	Commentaires						
DDM : 36 mois DDM minimum à réception : 18 mois	Origine : Chine						
<p>Conditions de stockage</p> <p>A conserver dans un endroit frais et sec</p>							
CARACTERISTIQUES PHYSICO-CHIMIQUES		VALEURS NUTRITIONNELLES MOYENNES pour 100g					
Humidité : max 8 % aW : - Cendres totales : max 13 % Cendres ins. dans l'acide : max 2 % Huiles essentielles : - mL/100g		Valeur énergétique : - kcal soit - kJ Matières grasses : - g dont acides gras saturés : - g Glucides : - g dont sucres : - g Protéines : - g Sel : - g Fibres alimentaires : - g					
<p>Commentaires Physico-chimie</p>							
<p>CARACTERISTIQUES MICROBIOLOGIQUES</p>							
Flore totale / g Entérobactéries / g Coliformes totaux / g Escherichia coli / g Salmonella Staphylocoques à coagulase positive / g Listeria monocytogenes Anaérobies sulfito-réducteurs /g Clostridium perfringens /g Bacillus cereus /g Levures /g Moisiures /g		<p>Criteres</p> <p>-</p> <p>1 000</p> <p>-</p> <p>100</p> <p>absence dans 25g</p> <p>100</p> <p>absence dans 1g</p> <p>-</p> <p>1 000</p> <p>1 000</p> <p>-</p> <p>-</p>	<p>Tolérances</p> <p>-</p> <p>10 000</p> <p>-</p> <p>1 000</p> <p>-</p> <p>10 000</p> <p>100 /g</p> <p>-</p> <p>10 000</p> <p>5 000</p> <p>-</p> <p>-</p>				
<p>Commentaires Microbiologie</p>							
<p>Epicez, sauez, dosez !</p> 							



FICHE TECHNIQUE
CIBOULETTE TUBULAI RE FLAPPERS

117801
Indice : f
Date : 20/11/2014
Page : 2 / 2

CONDITIONNEMENT	Emballage primaire	Emballage secondaire	Emballage tertiaire	Palette
Poids net	60 g	0.48 kg	- kg	47.04 kg
Poids brut	123 g	1.184 kg	- kg	139.03 kg
Longueur	- mm	315 mm	- cm	120 cm
Largeur	- mm	160 mm	- cm	80 cm
Hauteur	215 mm	220 mm	- cm	175 cm
Materiel	-	Carton	-	-
Marquage	-	-	-	-
GENCOD	3344540027736	3344540026111	-	-
Nb. emballages primaires	-	8	-	-
Plan de palettisation	Nb colis par rangée : 14	Nb rangées par palette : 7	Nb total colis par palette :	98

ALLERGENES (selon la directive 2007/68/CE)

Présence

All1	Céréales contenant du gluten (à savoir blé, seigle, orge, avoine, épeautre, Kamut ou leurs souches hybrides) et Produits à base de ces Céréales.	Non
All2	Poissons et Produits à base de Poissons.	Non
All3	Crustacés et Produits à base de Crustacés.	Non
All4	Oeufs et Produits à base d'Oeufs.	Non
All5	Arachides et Produits à base d'Arachides.	Non
All6	Soja et Produits à base de Soja.	Non
All7	Lait et Produits à base de lait (y compris lactose)	Non
All8	Fruits à coque, à savoir amandes, noisettes, noix, noix de cajou, noix de pécan, noix du Brésil, pistaches, noix de macadamia et noix du Queensland et Produits à base de ces fruits.	Non
All9	Graines de sésame et Produit à base de graines de sésame.	Non
All10	Céleri et Produits à base de céleri.	Non
All11	Moutarde et Produits à base de moutarde.	Non
All12	Anhydride sulfureux et sulfites en concentrations de plus de 10 mg/kg ou 10 mg/l exprimées en SO2.	Non
All13	Lupin et Produits à base de lupin.	Non
All14	Mollusques et produits à base de mollusques.	Non

Commentaires Allergènes

ATTESTATIONS

Conformément à la réglementation européenne sur les OGM (règlements CE n°1829/2003 et 1830/2003), ce produit est un ingrédient conventionnel et ne nécessite donc pas d'étiquetage spécifique.

Conformément à la directive 1999/2/CE, ce produit n'est pas ionisé et ne contient aucun ingrédient soumis à ionisation.

Cette fiche technique est conforme au règlement UE 1169/2011.

CERTIFICATIONS



Une gamme complète de sauces et d'épices pour les professionnels de la restauration...



E.1.6 Fiche technique poivron El Arenal

CONSERVAS EL ARENAL S.L.	FICHES TECHNIQUES POIVRONS ROUGES ENTIERS	<i>Date création: 10/02/2015 Date révision: 10/02/2015 Version: 1</i>								
Nom du produit	POIVRONS ROUGES ENTIERS	FORMAT								
DESCRIPTION		<i>PRODUIT CONSISTANT DE POIVRONS ROUGES ENTIERS DE LA VARIETE CAPSICUM ANNUM L, PELÉS DEPOURVUS DU CALICE ET TIGE ET PRACTIQUEMENT DE GRAINES CONSERVÉS DANS UNE SAUMURE COMPOSÉE D'EAU, DE SEL, ET D'ACIDE CITRIQUE DANS DES BOÎTES HERMETIQUES ET PASTEURISÉES JUSQU'A OBTENIR LEUR STERILITÉ COMMERCIALE</i>								
INGRÉDIENTS		<table border="1"> <tr> <td>POIVRONS ROUGES</td><td>62,50%</td></tr> <tr> <td>EAU</td><td>36,96%</td></tr> <tr> <td>SEL</td><td>0,34%</td></tr> <tr> <td>ACIDE CITRIQUE</td><td>0,20%</td></tr> </table>	POIVRONS ROUGES	62,50%	EAU	36,96%	SEL	0,34%	ACIDE CITRIQUE	0,20%
POIVRONS ROUGES	62,50%									
EAU	36,96%									
SEL	0,34%									
ACIDE CITRIQUE	0,20%									
DURÉE DE CONSERVATION	<i>LA DATE LIMITE DE CONSOMMATION EST DETERMINÉE PAR LA MANIPULATION ET LES CONDITIONS DE TRANSPORT ET DE STOCKAGE. DANS DES CONDITIONS NORMALES (TEMPÉRATURE ENTRE 10-25 °C ET HUMIDITÉ INFÉRIEURE À 75 %) ELLE EST DE 36 MOIS</i>									
CONDITIONS DE STOCKAGE ET RECOMMANDATIONS D'UTILISATION	<i>Protéger de la lumière, et garder à température ambiante, une fois ouvert, garder au frais dans un récipient non métallique à une température entre 1 - 3° C, pendant une période maximum de 3 jour.</i>									
INFORMATION DIÉTÉTIQUE	<i>Produit sans conservant ni colorants. Produit apte pour les végétariens</i>									
SPECIFICATIONS DE FERMETURES	<table border="1"> <tr> <td>Pourcentage minimum d'embrochement</td><td>45%</td></tr> <tr> <td>Pourcentage minimum de compacité</td><td>80%</td></tr> </table>		Pourcentage minimum d'embrochement	45%	Pourcentage minimum de compacité	80%				
Pourcentage minimum d'embrochement	45%									
Pourcentage minimum de compacité	80%									

CONSERVAS EL ARENAL S.L.

**FICHES TECHNIQUES
POIVRONS ROUGES ENTIERS**

*Date création: 10/02/2015
Date révision: 10/02/2015
Version: 1*

DESTINATION

Consommation humaine en général

NORMES DE QUALITÉ	CONTENU	1/2 KG
	POIDS NET (g)	400
	POIDS ÉGOUTTÉ (g)	250
	ESPACE DE TÊTE (MM)	< 15
	VACUUM (CM Hg)	> 10
	PH	4.0 ± 0.3
	ODEUR	TIPIQUE
	COULEUR	BRILLANT ROUGE FONCÉ
	TEXTURE	TIPIQUE
	SAVEUR	TIPIQUE

VALEURS NUTRITIONNELLES
En accord avec le règlement CE 1169/2011

Energie	109 kJ
Energie	26 Kcal
Graisses	0,2 G
Dont saturées	0 G
Carbohydrates	4,5 G
Dont sucre	2 G
Fibre	1,5 G
Protéines	0,8 G
Sel	0,4 G

MINERAUX/VITAMINES
(valeur moyenne/100g poids égoutté)

SODIUM (mg)	160
CALCIUM mg)	20
FER (mg)	0,4
VIT. A (IU)	520
VIT.C (mg)	45

CONSERVAS EL ARENAL S.L.	FICHES TECHNIQUES POIVRONS ROUGES ENTIERS	<i>Date création: 10/02/2015 Date révision: 10/02/2015 Version: 1</i>
---------------------------------	--	---

PARAMÉTRES MICROBIOLOGIQUES	INCUBATION A 37°C PENDANT 7 JOURS (AFNOR V08-408)	SANS ALTERATION
------------------------------------	---	-----------------

INFORMATION ALLERGÉNIQUE	Liste des composants qui causent des allergies ou une hypersensibilité (CE Régulation 1169/2011. Annexe II)	<i>Présent dans le produit</i>	
		<i>oui / Non</i>	<i>Ingrédients</i>
	Céréales contenant du gluten et produits à base de céréales contenant du gluten (blé, maïs, seigle, orge, avoine, épeautre, Canut ou de leurs hybrides)	NON	
	Crustacés et fruits de mer	NON	
	Les oeuf et produits dérivés	NON	
	Les poissons et produits dérivés	NON	
	Cacahuète et produits dérivés	NON	
	Soja et produits à base de soja, à l'exception de l'huile et de graisse de soja entièrement raffinées	NON	
	Lait, produits laitiers et dérivés (inclus la lactose et les protéines de lait)	NON	
	Noix: amandes, noisettes, noix, noix de cajou, noix de pécan, noix du Brésil, noix de pistaches, noix de macadamia, et produits dérivés	NON	
	Céleri et produit dérivés	NON	
	Moutarde et produits dérivés	NON	
	Graines de sésame et produit dérivés	NON	
	Le dioxyde de soufre et sulfites en concentrations de plus de 10 mg / kg ou 10 mg / litre en termes de total SO	NON	
	Lupin et produits dérivés	NON	
	Molusques et produits dérivés	NON	

CONFORMITÉ	Ne contient pas et n'a pas été fabriqué avec des produits GMO
	Conforme à la législation pour les pesticides
	Conforme avec la législation de résidus de métaux
	Conforme avec la législation de traçabilité Règlement CE 178/2002) et HACCP's (Règlement CE 852/2004).

E.1.7 Fiche technique mélange trappeur Terre Exotique



Date d'impression : 30/12/19

Création : 12/06/12

Remarque :

Les informations contenues dans cette fiche technique sont données de bonne foi, en l'état actuel de nos connaissances, et selon les indications communiquées par le producteur ou le fournisseur. Il appartient au client de vérifier la conformité de la marchandise par rapport à l'usage qu'il en fait.

FICHE TECHNIQUE Mélange du trappeur, 70 g

Trapper blend, 70g

Code article KEREX / KEREX Code	TEEPTRAPPEUR	Poids net / net weight	0,07 Kilogramme
Nom latin (si disponible) / (Latin name)	X	Poids brut / gross weight	0,125 Kilogramme
Code barre / EAN Code	3760063322262	Origine / Origin	CANADA

Informations générales / General information

DLUO conseillé / "Best before date" recommended	5 ans / 5 years
Nomenclature douanière / Customs code	0910999900
Conditions idéales de stockage / Conditions of storage	Conserver dans un endroit frais et sec Store in a cool dry place
Ingrédients : / Ingredients	Sucre, poivre noir, coriandre, légumes déshydratés (ail, oignon, poivron rouge), sel de mer, sucre d'érythritol, arôme d'érythritol naturel, huile végétale (canola) Sugar, black pepper, coriander, dehydrated vegetables (garlic, onion, red bell pepper), sea salt, maple sugar, natural maple aroma, vegetable oil (canola)

Contaminants / Contaminating

Ionisation / Irradiation	Conformité à la directive 1999/2/CE (22/02/99) Produit non ionisé et ne contenant pas d'ingrédients ionisés. Not irradiated accordingly with the Reg 1999/2/CE (22/02/99).		
OGM / GMO	Free from GMO Ne contient pas d'OGM, est non soumis à l'étiquetage sur les OGM		
Pesticides/ Pesticides	Conforme à la directive 396/2005 /CE In accordance with Reg 396/2005 /CE.		
Métaux Lourds / Heavy Metals	Conforme au règlement 1881/2006 /CE In accordance with Reg 1881/2006 /CE..		
Allergènes et leurs dérivés (si présents) / Allergens (if existing)	Gluten	/ Gluten	Absence
	Crustacés	/ Crustaceans	Absence
	Oeufs	/ Eggs	Absence
	Poisson	/ Fish	Absence
	Soja	/ Soy	Absence
	Lait	/ Milk	Absence
	Fruits à coque - Arachides	/ Peanuts and Treenuts	Absence
	Céleri	/ Celery and celeriac	Absence
	Moutarde	/ Mustarde	Absence
	Sésame	/ Sésame	Absence
	Sulfites	/ Sulphites	Absence
	Lupin	/ Lupin	Absence
	Mollusques	/ Shellfish	Absence

Caractères microbiologiques / Microbiological characteristics

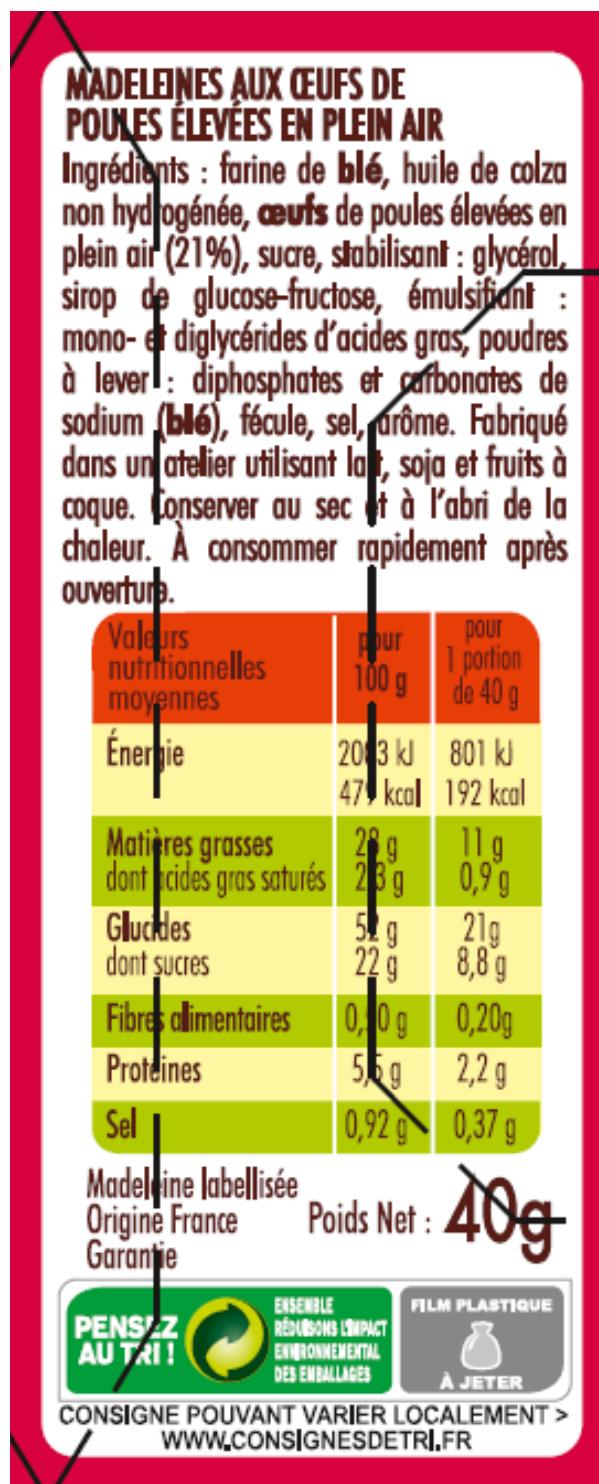
Microorganismes aérobies 30 °C	/ Total plat count (APC)	NF V05-051 < 6 000 000 / g
Escherichia coli	/ E. Coli	NF V08-053 < 10 / g
Salmonelles	/ Salmonella	NF V08-052 Absence dans 25g
Levures	/ Yeasts	NF V08-059 < 10 000 / g
Moisiures	/ Moulds	NF V08-059 < 10 000 / g
Aflatoxine Total	/ Total aflatoxin	Kit Enzymatique < 10 ppb
Aflatoxine B1	/ B1 aflatoxin	Kit Enzymatique < 5 ppb

E.2 Étiquettes produit

E.2.1 Étiquette curry Grain d'ailleurs



E.2.2 Étiquette madeleines Saint Michel

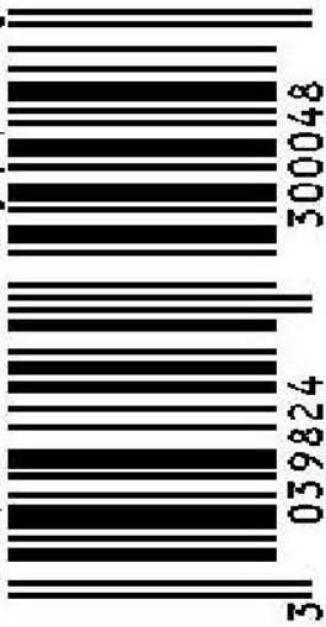


E.2.3 Étiquette lentilles Soufflet

**LENTELLER LINSEN 4mm
GREEN LENTILS 4mm
LINZEN 4mm**

Trace possible de céréales contenant du gluten - Possible traces of gluten containing cereals

Mögliche Spuren von Getreide, die Gluten enthalten - Mogelijk spoor van graangewassen met gluten



5 KG

VP

E.2.4 Étiquette pannacotta Nestlé



E.2.5 Étiquette sauce soja Kikkoman



E.2.6 Étiquette mélange trappeur Terre Exotique



E.3 Étiquetage manuel des données

E.3.1 Règles de gestion pour l'étiquetage

#	REGLES DE GESTION
REGLES GENERALES SUR L'INTERPRETATION DU DOCUMENT	
020	Si la liste d'ingrédients n'est pas présente sur le document, on laisse le champ vide. Ex : 21233a00-bc20-40fc-acb9-ee2e2321cac2
026	Si un document est visiblement corrompu (ex : ne contient rien), on le garde dans le jeu de test, avec aucune liste d'ingrédients en cible. Ex : 8e8bbc12-7e4b-4fff-a111-f7c8ae35129a
027	En règle générale, on ne récupère que ce qui est mentionné dans le bloc 'Ingrédients' ou 'Composition' de la fiche technique. Et ce, même si on aurait gardé le texte d'un autre bloc s'il avait été avec le texte de la liste d'ingrédients. Ex : Composition typique (Données inappropriées pour la demande des restitutions) pâte de cacao Tanzanie 69,5% ; sucre 21,5% ; beurre de cacao 9,0% ; vanille naturelle en poudre <1% Contact croisé d'allergènes possible au cours du process Peut contenir : Lait => on ne garde que : pâte de cacao Tanzanie 69,5% ; sucre 21,5% ; beurre de cacao 9,0% ; vanille naturelle en poudre <1% Ingrédients: SEMOULE DE BLE' DUR de qualité supérieure Origine des matières premières Blé cultivé en UE et Hors UE, moulu en Italie => on ne garde que : SEMOULE DE BLE' DUR de qualité supérieure
028	Si la fiche ne contient qu'un tableau récapitulatif des ingrédients, et pas la liste des ingrédients telle qu'elle doit être imprimée sur l'étiquette, on ne récupère rien. Ex : bc48d583-edf0-460e-a536-58f347ba6823
032	Si la liste d'ingrédients ne concerne qu'un seul composant et qu'elle est préfixée par le nom du produit, on ne garde que la liste d'ingrédients en tant que telle. Ex : CARACTERISTIQUES TECHNIQUES Ingrédients Sauce tomate, fumée et pimentée :Vinaigre d'alcool, eau, purée de tomates concentré (17%), sucre, purée de datte, pâte de piment Chipotle (5%, purée de tomates, piments Jalapeno fumé, eau, oignon, vinaigre d'acétol, sucre, poivres, sel, ail, épices, persil, extrait d'épice), amidon modifié, sel, jus de pomme concentré, poudre de Chili, arôme de fumée, épice, poudre d'ail, poudre d'oignon, conservateur (sorbate de potassium), extrait d'épice. On ne garde que : Vinaigre d'alcool, eau, purée de tomates concentré (17%), ...
039	Si la liste d'ingrédient mentionne une interprétation de la liste (un synonyme plus parlant), on ne reprend pas cette mention. Ex : 100% tourteau de cacao maigre = poudre de cacao pure Peut contenir des traces d'arachide et de lait devient : 100% tourteau de cacao maigre Peut contenir des traces d'arachide et de lait
LANGUES	
029	Si la fiche technique est en anglais, on ne récupère rien. Ex : 45aaafdde-9ca7-42ab-973f-70d6fa402e17
037	Si la liste d'ingrédients est présente en plusieurs langues, on ne garde que le français. Y compris si les textes sont "mélangés". Ex : Ingrédients - Ingredients: semoule de blé dur de qualité supérieure, issu de la filière Alpina durum wheat premium semolina from Alpina durum wheat channel Peut contenir des traces d'œufs - May contain traces of eggs devient : semoule de blé dur de qualité supérieure, issu de la filière Alpina Peut contenir des traces d'œufs
MISE EN PAGE ET FORMAT	
001	On conserve les retours à la ligne qui correspondent à un retour chariot volontaire. On garde sur une unique ligne les longues suites de mots qui sont visuellement retournées à la ligne parce qu'on arrive "au bout de la zone". Ex : Aubergine 60,5% (aubergine, huile de tournesol), eau, oignon, huile de tournesol, jus de citron, concentré de tomate, huile d'olive vierge extra (2%), ail, sel, persil, basilic, poivre, thym, romarin. devient : Aubergine 60,5% (aubergine, huile de tournesol), eau, oignon, huile de tournesol, jus de citron, concentré de tomate, huile d'olive vierge extra (2%), ail, sel, persil, basilic, poivre, thym, romarin.
002	Si des tirets, des puces ou des listes numérotées sont présentes, on garde le marqueur de début de ligne tel quel. Ex : - 100% Semoule de BLE dur de qualité supérieure - Contient du gluten Si le numéro de lot contient la lettre N : peu contenir de l'œuf
003	La casse et les accents sont conservés tels quels.
004	Si le copier / coller depuis pdf ajoute des espaces ou autres artefacts, on les corrige pour avoir un texte qui correspond à ce qui est lu par un utilisateur Ex : A u b e r g i n e , 6 0 , 5 % (aubergine, huile de tournesol), ... devient: Aubergine 60,5% (aubergine, huile de tournesol),
005	La ponctuation est conservée telle quelle, en particulier on garde le point final s'il est présent.
008	On garde les fautes d'orthographe
010	On remplace l'accent aigu sans lettre ' par une apostrophe normale : '

	011	Remplacer l'e dans l'o (œ) par oe
	012	Pas de multiples retours à la ligne successifs (même s'ils apparaissent à la lecture)
014		<p>Si les ingrédients sont séparés par des retours chariot, on conserve cette mise en forme. Ex : eau sucre étrait de thé noir (1,4g/l) acidifiant (acide citrique) jus de citron à base de concentré (0,1%) arôme correcteur d'acidité (citrate trisodique) antioxydant (acide ascorbique)</p>
		PREFIXES A LA LISTE
036	006	<p>On ne garde pas les préfixes du type "Ingrédients :" ou "Composition :"</p> <p>Si le bloc d'ingrédient fait référence au règlement INCO, on ne conserve pas cette mention. Ex : Déclaration d'étiquetage Selon le règlement 1169/2011 UE : ananas, jus d'ananas, antioxydant : acide ascorbique, acidifiant : acide citrique</p> <p>On ne garde que : ananas, jus d'ananas, antioxydant : acide ascorbique, acidifiant : acide citrique</p>
		MENTION DES ALLERGENES
007	017b	<p>On garde les précisions relatives aux traces d'allergènes, aux risques de contamination croisées, ... Ex : Fabriqué dans une usine utilisant des FRUITS à COQUE, SOJA et ARACHIDES => On garde.</p> <p>Si une note de bas de page précise la présence d'un composant dans un ingrédient, on le mentionne. Ex : Eau, huile de colza, vinaigre d'alcool, MOUTARDE de Dijon (eau, graines de MOUTARDE, vinaigre d'alcool, sel, correcteur d'acidité : E330, conservateur : E224*), sel, sucre*, acidifiant : E330, épice, stabilisants : E412 et E415. *Contient : SULFITES. <=> On garde cette ligne</p>
035		<p>On ne garde pas de mention précisant l'absence d'allergènes : Ex : Filets de maquereaux scomber scombrus 60% (Pêchés en Atlantique Nord-Est), eau, concentré de tomates (30% de la sauce), huile de colza, vinaigre, sel, épaississant : gomme xanthane, arômes naturels. Allergènes majeurs (hors poisson) : absence. ==> on ne garde pas cette mention</p>
035b	035b	<p>On conserve les mentions précisant la présence d'un allergène si jamais la liste d'ingrédients ne permet pas de le déduire (sulfites uniquement ?) Ex : Huile de colza, à Aceto Balsamico di Modena IGP » (25%) (vinaigre de vin, moût de raisin cuit, colorant E-150d), huile d'olive vierge extra. Contient sulfites. <=> on garde cette ligne.</p>
		PRECISIONS SUR LA COMPOSITION - PRESENCE DE COMPOSANTS SPECIFIQUES HORS ALLERGENES
017		<p>On mentionne les composants et les recommandations qui peuvent être présentes. Ex : Maltodextrine, carraghénane E407a, amidon de blé, colorant : jus concentré de betterave E162, édulcorant : aspartame E951* (0,37%), arôme goût framboise (substances aromatisantes, substances aromatisantes naturelles). *Contient une source de phénylalanine, ne convient pas aux femmes enceintes. ==> On garde cette ligne</p>
		PRECISIONS SUR LA COMPOSITION - PROPORTIONS
009	009	<p>On garde les précisions relatives au détail de la composition (en particulier le taux de cacao) Ex : Taux de coumarine compris entre 1 et 3,5 % ==> on garde ----- sucre, pâte de cacao, beurre de cacao, émulsifiant : lécithine de tournesol (E322), arôme vanille cacao : 40%minimum ==> on garde ----- chocolat supérieur au lait 39% (sucre, lait en poudre, beurre de cacao, pâte de cacao, émulsifiants : lécithines [soja], vanilline), sucre, lait écrémé en poudre, huile de palme, beurre concentré, émulsifiants : lécithines [soja], vanilline. Sur le total : produits laitiers 33 % (lait écrémé en poudre, lait en poudre : 27,4 %, beurre concentré 5,6 %) - cacao 12,5 %. Le chocolat utilisé est un chocolat pur beurre de cacao. <=> On garde cette seconde partie ----- eau, sirop de glucose-fructose, sucre, épaississant : amidon modifié, graisse végétale non hydrogénée de noix de coco, jus concentré de citron 2%, gélifiant : pectines de fruits, arôme naturel de citron, émulsifiants E472C et E473, arôme, conservateur : sorbate de potassium, sel, colorant : E161b. Teneur en Lutéine : moins de 5 mg/kg <=> On garde cette partie</p>
009b		<p>Y compris lorsqu'accompagnée de mise en garde particulière : TENEUR ÉLEVÉE EN CAFÉINE, DÉCONSEILLÉ AUX ENFANTS ET AUX FEMMES ENCEINTES OU ALLAITANTES OU AUX PERSONNES SENSIBLES À LA CAFÉINE (21mg/100ml). CONSOMMER AVEC MODÉRATION. <=> On garde cette ligne</p>
		MENTIONS CONDITIONNELLES
034		<p>Si une mention précise une condition pour la présence d'un ingrédient, on conserve cette mention. Ex : Pommes 21%, poires 39%, eau, sucre, arôme, antioxydant : acide ascorbique, acidifiant : acide citrique (si le fruit n'en contient pas suffisamment). On garde la mention : (si le fruit n'en contient pas suffisamment)</p>
034b		<p>Si une mention précise une condition pour la présence d'une trace d'allergène, on conserve cette mention. Ex : - 100% Semoule de BLE dur de qualité supérieure - Contient du gluten Si le numéro de lot contient la lettre N : peu contenir de l'œuf ==> on conserve cette mention.</p>
034c		<p>Si une mention précise une condition pour la présence d'additif, on conserve cette mention. Ex : INGRÉDIENTS : Segments de pamplemousse, eau, sucre, acidifiant : acide citrique*, agent de fermeté : chlorure de calcium*. * En fonction des origines, des additifs peuvent être ajoutés. <=> On garde cette mention</p>
009c		<p>Si une note de bas de page précise la variabilité du taux dans la liste d'ingrédients, on conserve cette note. Ex : pulpe de pommes (50%)*, sucre, sirop de glucose, gélifiant : pectines, acidifiant : acide citrique, arômes, colorants : E100 - E163 - E160c - E141. * Pourcentage à la mise en œuvre pour chaque parfum. <=> On conserve cette ligne.</p>
		LISTES D'INGRÉDIENTS MULTIPLES POUR UNE MÊME FICHE TECHNIQUE

	Dans le cas d'assortiments, s'il y a plusieurs listes d'ingrédients successives, on les listes toutes en gardant les entêtes qui définissent le produit. Ex : Confiture de myrtilles et de cassis Ingrédients : fruits (myrtilles 41%, cassis 9%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits. Confiture de fraises et de groseilles Ingrédients : fruits (fraises 27 %, groseilles 23 %), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits. Confiture d'abricots et de pêches fruits (abricots 34%, pêches 16%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits. Malgré tous nos soins, cette confiture peut contenir des noyaux. Marmelade d'oranges douces et de mandarines Ingrédients : fruits (oranges douces 37%, mandarines 3%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits.
013b	S'il existe plusieurs manières différentes de présenter le produit, chacune avec sa liste d'ingrédients, on conserve les 2 de la même manière que lorsqu'il y a plusieurs composants. Ex: Vinaigre de vin rouge au jus d'échalote 7% d'acidité Vinaigre de vin rouge (sulfites), jus d'échalote (0,5%), arôme*, conservateur: sulfite acide de sodium Vinaigre de vin rouge aromatisé à l'échalote 7% d'acidité Vinaigre de vin rouge (sulfites), arômes échalote (1%), conservateur: sulfite acide de sodium
040	Si une liste d'ingrédient possède une variante en fonction du conditionnement, on conserve les informations relatives à ces précisions. Ex : Eau de source*/Eau**; jus de fruits à base de concentrés 12.4% (orange 11.4%, ananas 1%); sucre; acidifiants: acide citrique, acide malique; extraits d'orange; arômes; antioxydant: acide ascorbique; stabilisant: gomme arabique. *PET **CAN => On conserve les 2 dernières lignes qui définissent pour quel type de conditionnement la variante s'applique.
	ORIGINES Si une mention de l'origine de transformation est faite, on NE récupère PAS. Ex : Sucre, amidon de FROMENT, farine de FROMENT, amidon modifié, fibres solubles, poudre à lever : (E450, E500, E341), poudre de LAIT écrémé, émulsifiants : (E471, E472e), amidon, farine de FROMENT pré gélatinisée, sel, arôme naturel. Fabriqué en France. => on ne garde pas le "Fabriqué en France"
015	Si une mention de l'origine des ingrédients est faite, on la récupère. Ex : Pomme 77% (origine 100% France), abricot 20% (origine 100% France), sucre (origine 100% France), jus concentré de carotte, antioxydant: acide ascorbique. Certains ingrédients de ce produit de proviennent pas de France. => On garde cette ligne.
016	En particulier, pour les produits qui contiennent du poisson, si la zone de pêche est présente on la récupère. Ex : Sardines (Sardina pilchardus) 70 %, huile de tournesol, sel. Zone de pêche : Océan Atlantique Centre Est => On conserve cette mention
016b	Si la liste d'ingrédients précise l'origine végétale ou animale des ingrédients, on conserve cette information. Ex : 100% thé noir Origine végétale
038	DETAIL PAR PHASE Si une partie de la recette est détaillée, on conserve également la description de cette sous-partie. Ex : 018 Légumes (pommes de terre (14%), carottes (10%), champignons (9%), haricots verts, persil), sauce, thon* (22%). Composition de la sauce : Eau, huile de tournesol, vinaigre, moutarde de Dijon (eau, graines de moutarde, vinaigre, sel), sel, arômes, amidons transformés de maïs, épaississants : gomme guar et gomme xanthane. => On garde la composition de la sauce
	PRECISION SUR L'IDENTIFICATION D'UN COMPOSANT Si une note de bas de paragraphe précise le nom latin du poisson, on conserve cette note de bas de page. Ex : 019 Légumes (pommes de terre (14%), carottes (10%), champignons (9%), haricots verts, persil), sauce, thon* (22%). *Euthynnus (Katsuwonus) pelamis
019'	Si le nom latin d'un ingrédient hors poisson est présent dans la liste des ingrédients, on le conserve. Ex : semoule de blé dur blanche biologique (Triticum durum) <= On conserve Triticum durum
	LABELS DES INGREDIENTS Si une note de bas de paragraphe précise les ingrédients d'origine biologique, on conserve cette mention. Ex : 019b Pur thé vert* (100%). (*) Ingrediénts issus de l'Agriculture Biologique => on garde cette mention
019b	Si la note de bas de page mentionne également les références du certificat bio, on NE garde PAS ces références. Ex : 019b' Semoule de blé dur précurte* (gluten) (44,5%), semoule de blé dur complète précurte* (gluten) (29,5%), flocons de soja* (15%), flocons de céréales* (gluten) (blé* (3%), orge*, avoine*, seigle* (2,2%), riz* (0,7%). * Ingrediénts issus de l'Agriculture Biologique => On garde cette mention Produit issu de l'agriculture biologique certifié bio par Ecocert FR-BIO 01 – N/N identification ECOCERT : 44/20349 => On NE garde PAS cette mention
019c	Si une note de bas de paragraphe précise un label sur un des ingrédients, on conserve cette mention. Ex : infusion de thé noir 94% (eau, infusion intense de thé noir*), sucre, jus de pêche à base de concentré 1%, arômes naturels de thé, acidifiants: acide malique et acide citrique, arômes naturels, correcteur d'acidité: citrate de sodium, antioxydant: acide ascorbique. *Vérifié Rainforest Alliance => on garde cette mention
	ALLEGATIONS ET RESERVES On NE mentionne PAS les réserves relatives au traitement physique. Ex : On NE garde PAS ceci Malgré tous les soins apportés à nos produits, leur caractère naturel n'exclut pas la présence éventuelle de noyaux, de pépins, de restes de peaux ou de parties fibreuses.
021	On NE mentionne PAS l'absence d'OGM ou d'ingrédients ionisés. Ex : On NE garde PAS ceci Le produit ne contient ni OGM, ni ingrédients ionisés.
022	

024	On NE récupère PAS d'information de composition qui ne sont pas spécifiques à un ingrédient. Ex : Préparée avec 35g de fruits pour 100g de produit fini. ==> on ne récupère pas
025	On NE récupère PAS d'information en lien avec les données nutritionnelles. Ex : Teneur totale en sucres : 60g pour 100g. ==> on ne récupère pas.
030	On NE conserve PAS les informations relatives au processus de fabrication ou de conditionnement. Ex : Conditionné sous atmosphère protectrice.
031	On NE conserve PAS les consignes de conservation. Ex : A conserver à l'abri de la chaleur et de l'humidité.
033	On ne conserve pas les précisions d'utilisation positionnées dans la liste d'ingrédients. Ex : Ingrediénts : Ingrédients : Colorant : Caramel de sulfite caustique E150b (contient des sulfites), eau. Pour denrées alimentaires. On ne garde pas "Pour denrées alimentaires."

Annexe F

RÉSULTATS DU MODÈLE RETENU

TABLE 30 – Prédiction du meilleur modèle sur le set de test

Liste d'ingrédients prédicté	Liste d'ingrédients cible	Sim.
Ingrediénts : Café instantané, café torréfié moulu (3%). 36 mois à l'abri de la chaleur et de l'humidité	Café instantané, café torréfié moulu (3%).	75.00%
Poire 99,9%, antioxydant : acide ascorbique.	Lentilles blondes	14.89%
Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.	Poire 99,9%, antioxydant : acide ascorbique.	100.00%
Edulcorants sorbitol, isomalt, sirop de maltitol, aspartame, mannitol, sel d'aspartame-acesulfame, acesulfame-k, sucralose ; gomme base (contient de la lecithine de SOJA), arômes, épaississant gomme arabique, humectant glycerol, colorant E171, agent d'enrobage cire de carnauba, colorant E163, antioxygène BHA. Contient une source de PHENYLALANINE.	Gésier de dinde émincé 50%, graisse de canard 47%, sel, arômes naturels de poivre.	98.80%
Z16005 / 002 sucre, amidon modifié, LACTOSÉRUM en poudre, dextrose (contient GLUTEN), CRÈME en poudre, LACTOSE, LAIT écrémé en poudre, émulsifiants (E 472a, E 472b, E 339), sirop de glucose, correcteur d'acidité (E 263), arômes (contient LAIT), colorants (E160 b, E 101i), Traces éventuelles de FRUITS À COQUES, OEUF.	Edulcorants sorbitol, isomalt, sirop de maltitol, aspartame, mannitol, sel d'aspartame-acesulfame, acesulfame-k, sucralose ; gomme base (contient de la lecithine de SOJA), arômes, épaississant gomme arabique, humectant glycerol, colorant E171, agent d'enrobage cire de carnauba, colorant E163, antioxygène BHA. Contient une source de PHENYLALANINE.	99.14%
Gluten Crustacés Oeufs Poisson Soja Lait Fruits à coque - Arachides Céleri Moutarde Sésame Sulfites Lupin Mollusques	sucre, amidon modifié, LACTOSÉRUM en poudre, dextrose (contient GLUTEN), CRÈME en poudre, LACTOSE, LAIT écrémé en poudre, émulsifiants (E 472a, E 472b, E 339), sirop de glucose, correcteur d'acidité (E 263), arômes (contient LAIT), colorants (E160 b, E 101i), Traces éventuelles de FRUITS À COQUES, OEUF.	95.31%
paprika, poivre noir, coriandre, cumin, ail semoule, carvi noir cannelle, clou de girofle, muscade		14.66%
Maltodextrine, amidon de maïs, sel, farine de blé, colorant : caramel ordinaire ; arômes (blé, céleri), huile de palme, épaississant : gomme guar ; oignon, féculé	Maltodextrine, amidon de maïs, sel, farine de blé, colorant : caramel ordinaire ; arômes (blé, céleri), huile de palme, épaississant : gomme guar ; oignon, féculé de pomme de terre, extrait de levure, jus de cuisson de viande de bœuf (0,9%), acidifiant : acide citrique ; extrait de vin blanc, extraits d'ail, de thym et de poivre. Peut contenir : lait, oeuf.	98.33%
Boisson gazeuse aromatisée au jus de fruit à base de concentré S.Pellegrino Orange 33 cl (Aranciata) S.Pellegrino Citron 33 cl (Limonata)	<rien>	0.00%
Le plaisir des fruits à l'italienne		
LISTE D'INGREDIENTS Les ingrédients mis en œuvre pour fabriquer le produit sont les suivants (ordre d'importance pondérale décroissant) : Farine de BLÉ 20% - Huile de colza - OEUFS entiers 17,5% - Sucre - Graines de tournesol caramélisées (graines de tournesol 3,5%, sucre) - Stabilisant : glycérine - Farine de BLÉ complet 4,5% - Poudre à lever : diposphates, carbonates de sodium - Flocons d'AVOINE 1% - Poudre de LAIT écrémé - Protéines de LAIT - Emulsifiants : mono- et diglycérides d'acides gras (origine végétale) - Sel - Arômes naturels (contient alcool).	Farine de BLÉ 20% - Huile de colza - OEUFS entiers 17,5% - Sucre - Graines de tournesol caramélisées (graines de tournesol 3,5%, sucre) - Stabilisant : glycérine - Farine de BLÉ complet 4,5% - Poudre à lever : diposphates, carbonates de sodium - Flocons d'AVOINE 1% - Poudre de LAIT écrémé - Protéines de LAIT - Emulsifiants : mono- et diglycérides d'acides gras (origine végétale) - Sel - Arômes naturels (contient alcool).	74.78%
Persil feuilles	Persil	40.00%
mini poivrons jaunes, eau, sucre, sel, affermissant chlorure de calcium : E509, acidifiant : acide citrique, vinaigre, antioxydant : vitamine C E330	mini poivrons jaunes, eau, sucre, sel, affermissant chlorure de calcium : E509, acidifiant : acide citrique, vinaigre, antioxydant : vitamine C E330	98.67%
HARICOTS BLANCS BIO 5Kg	100% haricots blancs bio	4.17%
INGREDIENTS :	Poudre de lait écrémé, anti-agglomérant : E551 [nano]	52.05%
Poudre de lait écrémé, anti-agglomérant : E551 [nano]	Lait origine France	

Continued on next page

Liste d'ingrédients prédicté	Liste d'ingrédients cible	Sim.
Ingédients : sucre ; sirop de glucose ; amidon de maïs ; humectant : sirop de sorbitol ; farine de blé ; eau ; acidifiants : acide citrique, acide malique ; agent d'enrobage : cire de carnauba ; correcteurs d'acidité : citrate tricalcique, malate acide de sodium ; émulsifiant : mono- et diglycérides d'acides gras ; arôme ; concentrés de fruits et de plantes : cassis, baie de sureau, aronia, raisin ; extrait de baie de sureau ; colorants : curcumine, bleu patenté V.	HAPPY CHERRY sirop de glucose ; sucre ; gélatine ; dextrose ; acidifiant : acide citrique ; arôme ; concentrés de fruits et de plantes : carthame, pomme, spiruline ; colorant : anthocyanes ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba ; sirop de sucre inverti. WORMS sirop de glucose ; sucre ; gélatine ; acidifiant : acide citrique ; concentrés de fruits et de plantes : carthame, spiruline, pomme, kiwi, carotte, hibiscus, cassis, baie de sureau, aronia, raisin, orange, citron, mangue, fruit de la passion ; extrait de baie de sureau ; sirop de caramel ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba.	8.96%
OEUFS AU PLAT	OEUFS AU PLAT sucre ; sirop de glucose ; dextrose ; gélatine ; acidifiant : acide citrique ; arôme ; colorants : carotènes végétaux, anthocyanes ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba.	
L'OURS D'OR	L'OURS D'OR sirop de glucose ; sucre ; gélatine ; dextrose ; acidifiant : acide citrique ; jus de fruits concentrés : pomme, orange, citron, fraise, framboise, ananas ; arôme ; concentrés de fruits et de plantes : carthame, spiruline, pomme, kiwi, orange, baie de sureau, citron, mangue, fruit de la passion, cassis, aronia, raisin ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba.	
HAPPY COLA	HAPPY COLA sirop de glucose ; sucre ; gélatine ; dextrose ; acidifiant : acide citrique ; sirop de caramel ; arôme ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba.	
CROCO	CROCO sirop de glucose ; sucre ; dextrose ; gélatine ; acidifiant : acide citrique ; concentrés de fruits et de plantes : carthame, spiruline, pomme ; sirop de sucre inverti ; arôme ; colorants : carotènes végétaux, lutéine, anthocyanes ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba.	
RAINBOW PIK	RAINBOW PIK sucre ; sirop de glucose ; amidon de maïs ; humectant : sorbitols ; farine de blé ; dextrose ; amidon de blé ; graisse de palme ; acidifiants : acide citrique, acide malique ; correcteurs d'acidité : citrate tricalcique, malate acide de sodium, citrate trisodique ; agent d'enrobage : cire de carnauba ; gélatine ; émulsifiant : mono- et diglycérides d'acides gras ; arôme ; ...	
Farine de BLE, huile de colza non hydrogénée, OEUFS de poules élevées en plein air (21%), sucre, stabilisant : glycérol, sirop de glucose-fructose, émulsifiants : mono- et diglycérides d'acides gras, poudres à lever : diphosphates et carbonates de sodium (BLE), féculé, sel, arôme.	Farine de BLE, huile de colza non hydrogénée, OEUFS de poules élevées en plein air (21%), sucre, stabilisants : glycérol, sirop de glucose-fructose, émulsifiant : mono- et diglycérides d'acides gras, poudres à lever : diphosphates et carbonates de sodium (BLE), féculé, sel, arôme.	99.64%
MENTIONS D'ÉTIQUETAGE- LABELS MENTIONS INGREDIENTS : SIROP DE GLUCOSE / SUCRE / AROMES / ACIDIFIANT : ACIDE CITRIQUE E 330 / CONCENTRES (CASSIS, CAROTTE, POMME, CARTHAME, SPIRULINE, HIBISCUS)	SIROP DE GLUCOSE / SUCRE / AROMES / ACIDIFIANT : ACIDE CITRIQUE E 330 / CONCENTRES (CASSIS, CAROTTE, POMME, CARTHAME, SPIRULINE, HIBISCUS)	71.20%
Ingédients Olives vertes 98%denoyautées , huile tournesol, piments, sel ,poivre,(cayenne,noire) acides (citrique,LACTIQUE,ascorbutique), paprika rouge, poireau, oignon, persil, ail, tomate, bouillon (hydrolysat de protéines, SOYA et maïs, graisse de palme RPSO),	Olivs vertes 98%denoyautées , huile tournesol, piments, sel ,poivre,(cayenne,noire) acides (citrique,LACTIQUE,ascorbutique), paprika rouge, poireau, oignon, persil, ail, tomate, bouillon (hydrolysat de protéines, SOYA et maïs, graisse de palme RPSO), noix de muscade, Conservateur :E220.	81.12%
Pommes de terre 59.5 % - Céleris 40 % - Amidon de maïs - Sirop de glucose de maïs - Huile de colza - Emulsifiants : E322, E471 - Stabilisant : E450i - Curcuma - Conservateur : E223 - Antioxydant : E304 - Acidifiant : E330.	Pommes de terre 59.5 % - Céleris 40 % - Amidon de maïs - Sirop de glucose de maïs - Huile de colza - Emulsifiants : E322, E471 - Stabilisant : E450i - Curcuma - Conservateur : E223 - Antioxydant : E304 - Acidifiant : E330.	99.55%
Eau ; jus de fruit de la passion ; sucre ; épaisseur : pectine ; antioxydant : acide ascorbique	Eau ; jus de fruit de la passion ; sucre ; épaisseur : pectine ; antioxydant : acide ascorbique	100.00%
Eau, huile de tournesol, beurre 9.5 %, jaune d'oeuf 6 %, amidon modifié, sel, jus de citron concentré, amidon de maïs, épaisseurs (gomme guar, gomme xanthane), protéines de pois, sucre, arôme naturel, curcuma, extrait de paprika.	Eau, huile de tournesol, beurre 9.5 %, jaune d'oeuf 6 %, amidon modifié, sel, jus de citron concentré, amidon de maïs, épaisseurs (gomme guar, gomme xanthane), protéines de pois, sucre, arôme naturel, curcuma, extrait de paprika.	98.71%
Composition typique (Données inappropriées pour la demande des restitutions) pâte de cacao 50,5% ; sucre 47,5% ; beurre de cacao 1,5% ; émulsifiant : lécithine de soja <1% ; arôme naturel de vanille <1%	pâte de cacao 50,5% ; sucre 47,5% ; beurre de cacao 1,5% ; émulsifiant : lécithine de soja <1% ; arôme naturel de vanille <1%	60.49%
<rien>	<rien>	0.00%
Pommes de terre, eau, sel.	Pommes de terre, eau, sel.	100.00%
Farine de pois chiche	Farine de pois chiche	100.00%
Lentilles, eau, sel, arôme naturel (céleri)	Lentilles, eau, sel, arôme naturel (céleri)	97.73%
* Produits issus de l'Agriculture Biologique.	Sucre roux de canne* (64%), amidon de maïs*, poudre de LAIT écrémé*, poudre d'OEufs entiers*, gélifiants : carraghénanes, agar-agar* ; arôme naturel de vanille* et autres arômes naturels*, poudre de gousse de vanille*, curcuma*.	43.07%
o Ingrédient issu du commerce équitable. 65.1% des ingrédients d'origine agricole sont issus du commerce équitable (Sucre : Paraguay).	* Produits issus de l'Agriculture Biologique. o Ingrédient issu du commerce équitable. 65.1% des ingrédients d'origine agricole sont issus du commerce équitable (Sucre : Paraguay).	

Continued on next page

Liste d'ingrédients prédicté	Liste d'ingrédients cible	Sim.
Ingrediénts : Chocolat au lait (sucre, beurre de cacao, LAIT entier en poudre, pâte de cacao, lactose (LAIT), lactosérum en poudre (LAIT), émulsifiant : lécithine de SOJA), chocolat noir (pâte de cacao, sucre, beurre concentré (LAIT), beurre de cacao, émulsifiant : lécithine de SOJA), praliné 7% (sucre, AMANDES, NOISETTES), pâte d'amande 5% (sucre, AMANDES, NOISETTES), eau, sirop de sucre inverti, sirop de glucose, humectant : sorbitol, conservateur : sorbate de potassium, beurre concentré (LAIT), chocolat noir origine Madagascar 3% (pâte de cacao de Madagascar, sucre, beurre de cacao, émulsifiant : lécithine de SOJA), sucre, pâte de figues 1,5%, sirop de glucose, humectant : sorbitol, sirop de sucre inverti, beurre de cacao, LAIT concentré écrémé sucré, crème stérilisée (LAIT), LAIT entier en poudre, lactosérum en poudre (LAIT), graisses végétales (palme et palmiste), caramel aromatique (sucre, eau), meringues cacaotées 0,4% (sucre, cacao en poudre, blanc d'OEUF amidon de BLE, arôme), huile de tournesol, purée de PISTACHE, cacao maigre en poudre, émulsifiants : monostéarate de glycérol, lécithine de SOJA, fleur de sel de Guérande, sel, arômes naturels, arôme, enzyme : invertase, farine de BLE, malt d'ORGE. Contient lait, soja, amande, noisette, pistache, blé, orge et oeuf. Présence possible d'arachide, noix, noix de cajou et noix de pécan.	Chocolat au lait (sucre, beurre de cacao, LAIT entier en poudre, pâte de cacao, lactose (LAIT), lactosérum en poudre (LAIT), émulsifiant : lécithine de SOJA), chocolat noir (pâte de cacao, sucre, beurre concentré (LAIT), beurre de cacao, émulsifiant : lécithine de SOJA), praliné 7% (sucre, AMANDES, NOISETTES), pâte d'amande 5% (sucre, AMANDES, NOISETTES), eau, sirop de sucre inverti, sirop de glucose, humectant : sorbitol, conservateur : sorbate de potassium, beurre concentré (LAIT), chocolat noir origine Madagascar 3% (pâte de cacao de Madagascar, sucre, beurre de cacao, émulsifiant : lécithine de SOJA), sucre, pâte de figues 1,5%, sirop de glucose, humectant : sorbitol, sirop de sucre inverti, beurre de cacao, LAIT concentré écrémé sucré, crème stérilisée (LAIT), LAIT entier en poudre, lactosérum en poudre (LAIT), graisses végétales (palme et palmiste), caramel aromatique (sucre, eau), meringues cacaotées 0,4% (sucre, cacao en poudre, blanc d'OEUF amidon de BLE, arôme), huile de tournesol, purée de PISTACHE, cacao maigre en poudre, émulsifiants : monostéarate de glycérol, lécithine de SOJA, fleur de sel de Guérande, sel, arômes naturels, arôme, enzyme : invertase, farine de BLE, malt d'ORGE. Présence possible d'arachide, noix, noix de cajou et noix de pécan.	97.95%
Le chocolat utilisé est un chocolat pur beurre de cacao.	pâte à tartiner aux noisettes et au cacao 81,5 % (sucre, huile de palme, noisettes 13%, lait écrémé en poudre 8,7%, cacao maigre 7,4%, émulsifiants : lécithines [soja]; vanilline), farine de froment 16%, levure de bière, extrait de malt d'orge, sel, lait écrémé en poudre, émulsifiants : lécithines [soja]; protéines de froment, protéines de lait, eau. Le chocolat utilisé est un chocolat pur beurre de cacao.	13.63%
Sucre, farine de FROMENT, poudre d'OEUVFS entiers, amidon de FROMENT, poudre de LAIT fermenté, poudre à lever : (E500, E341, E450), amidon modifié, émulsifiants : (E472e, E472a, E472b), sel, poudre de LAIT écrémé, arôme. Fabriqué en France.	Sucre, farine de FROMENT, poudre d'OEUVFS entiers, amidon de FROMENT, poudre de LAIT fermenté, poudre à lever : (E500, E341, E450), amidon modifié, émulsifiants : (E472e, E472a, E472b), sel, poudre de LAIT écrémé, arôme.	90.87%
Amidon de maïs* - Lait écrémé* - Sel - Fécule de pomme de terre* - Tomate* - Oignon* - Arômes naturels - Poivres* 3 % (poivre vert*, poivre blanc*, poivre noir*) - Huile de tournesol* - Extrait de levure* - Sucre caramélisé* - Ail* - Maltodextrine de maïs*. * issus de l'agriculture biologique	Amidon de maïs* - Lait écrémé* - Sel - Fécule de pomme de terre* - Tomate* - Oignon* - Arômes naturels - Poivres* 3 % (poivre vert*, poivre blanc*, poivre noir*) - Huile de tournesol* - Extrait de levure* - Sucre caramélisé* - Ail* - Maltodextrine de maïs*. * issus de l'agriculture biologique	98.98%
< 10% tolerance + 3%	NOISETTES entières décortiquées.	12.50%
<rien>	Sucre; sirop de glucose; graisse de palme; humectant : sirop de sorbitol; gélatine; acidifiants : acide citrique; arôme.	0.00%
Gastronomie 70%A/30% R	100% Arabica	9.09%
- 100% Semoule de BLE dur de qualité supérieure - Contient du gluten	- 100% Semoule de BLE dur de qualité supérieure - Contient du gluten	93.43%
Si le numéro de lot contient la lettre N : peu contenir de l'oeuf	Si le numéro de lot contient la lettre N : peu contenir de l'oeuf	
Liste d'ingrédients : Lait écrémé 59.5%, lait en poudre reconstitué 20.5%, sucre, crème, semoule de blé, poudre de lait, extrait aromatique rhum orange (rum, infusion d'écorces d'oranges douces, alcool), épaisseur : alginate de sodium, rhum, correcteur d'acidité : citrate de sodium.	Lait écrémé 59.5%, lait en poudre reconstitué 20.5%, sucre, crème, semoule de blé, poudre de lait, extrait aromatique rhum orange (rum, infusion d'écorces d'oranges douces, alcool), épaisseur : alginate de sodium, rhum, correcteur d'acidité : citrate de sodium.	84.35%
Liste d'ingrédients : Lait écrémé 72%, caramel 10% crème, sucre, amidon transformé, beurre demi-sel 2% sel, épaisseur : gomme xanthane, correcteur d'acidité : citrate de sodium.	Lait écrémé 72%, caramel 10% crème, sucre, amidon transformé, beurre demi-sel 2% sel, épaisseur : gomme xanthane, correcteur d'acidité : citrate de sodium.	86.89%
Liste ingrédients : Pommes 89%, purée de fraises à base de concentré 6%, sirop de glucose-fructose, arôme naturel, jus concentré de carotte pourpre, antioxydant : acide ascorbique, acidifiant : acide citrique (si le fruit n'en contient pas suffisamment).	Pommes 89%, purée de fraises à base de concentré 6%, sirop de glucose-fructose, arôme naturel, jus concentré de carotte pourpre, antioxydant : acide ascorbique, acidifiant : acide citrique (si le fruit n'en contient pas suffisamment).	91.44%
- Soja fermenté naturellement (soja, sel, eau) - Extrait de Blé - Extrait de Légumes (Oignon, Radis, Ail, Varech, Cive, Chou, Carotte, Gingembre, Shiitake) - Extrait de Pollen Fermenté	- Soja fermenté naturellement (soja, sel, eau) - Extrait de Blé - Extrait de Légumes (Oignon, Radis, Ail, Varech, Cive, Chou, Carotte, Gingembre, Shiitake) - Extrait de Pollen Fermenté	97.35%
Farine de FROMENT, poudre de LACTOSERUM, sucre, poudre d'OEUF entier, poudres à lever : (E450, E500), matière grasse LAITIERE, sel. Peut contenir des traces de : soja, fruits à coques, lupin.	Farine de FROMENT, poudre de LACTOSERUM, sucre, poudre d'OEUF entier, poudres à lever : (E450, E500), matière grasse LAITIERE, sel. Peut contenir des traces de : soja, fruits à coques, lupin.	99.48%
1/2 1/4 1/8 1/16 1/32	Débris de truffes d'hiver, jus de truffes, sel	13.04%
Liste d'ingrédients : Maltodextrine, amidon modifié de pomme de terre, lentilles (10%), rous (10%) (farine de FROMENT, graisse de palme), LAIT fermenté traité thermiquement, sel iodé, sucre, tomate (4%), oignon (4%), LACTOSE, sel, amidon de pomme de terre, extrait de levure, paprika (2,2%), purée de tomates (2%), protéines de LAIT, ail, arômes, gingembre, farine de riz, cumin, jus d'oignon concentré, fructose, sauce soja (fève de SOJA, FROMENT), cannelle, jus de betteraves, huile de tournesol, jus de citron, persil tubéreux, dextrose. Peut contenir : œuf, céleri, moutarde.	Maltodextrine, amidon modifié de pomme de terre, lentilles (10%), rous (10%) (farine de FROMENT, graisse de palme), LAIT fermenté traité thermiquement, sel iodé, sucre, tomate (4%), oignon (4%), LACTOSE, sel, amidon de pomme de terre, extrait de levure, paprika (2,2%), purée de tomates (2%), protéines de LAIT, ail, arômes, gingembre, farine de riz, cumin, jus d'oignon concentré, fructose, sauce soja (fève de SOJA, FROMENT), cannelle, jus de betteraves, huile de tournesol, jus de citron, persil tubéreux, dextrose. Peut contenir : œuf, céleri, moutarde.	95.19%
Assaisonnement poudre de curry	Graine de coriandre (19.5%), ail déshydraté, curcuma (14.5%), cu-mi (13.5%), gingembre, graine de moutarde, piment, fenugrec, sel, macis, oignon déshydraté, fenouil, piment de la Jamaïque, girofle, laurier.	11.17%
- 100% Semoule de BLE dur de qualité supérieure - Contient du gluten	- 100% Semoule de BLE dur de qualité supérieure - Contient du gluten	94.16%
Si le numéro de lot contient la lettre N : peu contenir de l'oeuf	Si le numéro de lot contient la lettre N : peu contenir de l'oeuf	
Ingrediénts : cèpes 70% (Boletus edulis et respective famille), huile de tournesol, blanquette 5% (Tuber borchii Vitt.), oignon, beurre, sel, protéines du lait, farine de riz, amidon de maïs, dextrose, extrait de levure, épices, arôme truffée, arômes naturels, antioxydant : acide l-ascorbutique (E 300).	cèpes 70% (Boletus edulis et respective famille), huile de tournesol, blanquette 5% (Tuber borchii Vitt.), oignon, beurre, sel, protéines du lait, farine de riz, amidon de maïs, dextrose, extrait de levure, épices, arôme truffée, arômes naturels, antioxydant : acide l-ascorbutique (E 300).	85.16%
Haricots blancs issus de l'Agriculture Biologique	Haricots blancs issus de l'Agriculture Biologique	100.00%

Continued on next page

Liste d'ingrédients prédicté	Liste d'ingrédients cible	Sim.
Eau, maltodextrine, sel, arômes, sucre, arôme naturel de citronnelle, amidon modifié, ail en poudre, épices (combava, curcuma), extraits d'épices (gingembre, poivre), stabilisant (gomme xanthane).	Eau, maltodextrine, sel, arômes, sucre, arôme naturel de citronnelle, amidon modifié, ail en poudre, épices (combava, curcuma), extraits d'épices (gingembre, poivre), stabilisant (gomme xanthane).	99.49%
COD. PRODUIT : CPGLLAB 49 DESCRIPTION : Pate à tartiner au noisettes – Squeezed de 1 kg CODE ET DESCRIPTION CREME : pate à tartiner 13 % noisettes INGREDIENTS : Sucre, huiles végétales (Colza), noisettes (13%), poudre de cacao maigre (7%), lait écrémé en poudre , lactosérum en poudre, lactose. Emulsifiant lécithine de tournesol. Arôme : vanilline Allergènes : Le produit peut contenir traces éventuelles de soja arachide et autre fruits a coque.	Sucre, huiles végétales (Colza), noisettes (13%), poudre de cacao maigre (7%), lait écrémé en poudre , lactosérum en poudre, lactose. Emulsifiant lécithine de tournesol. Arôme : vanilline Allergènes : Le produit peut contenir traces éventuelles de soja arachide et autre fruits a coque.	62.66%
10 70 THON ALBACORE MORCEAUX NATUREL EN Croustilles de céréales enrobées de chocolat au lait et de pâte de spéculoos Ingrédients : chocolat au lait 68% (sucre, beurre de cacao, poudre de lait entier, pâte de cacao, émulsifiant : lécithines (soja), arôme naturel de vanille), pâte de spéculoos 16% [Spéculoos 60% (farine de blé, sucre, huiles et graisses végétales (palme, colza), sirop de candi, poudres à lever : E500(ii) - carbonate acide d'ammonium, sel, cannelle), huiles et graisses végétales (tournesol, colza, palme), sucre, émulsifiants : lécithines - E472c, correcteur d'acidité : E330], céréales croustillantes 12% (farine de blé, gluten de blé, sucre, farine de malt de blé, poudre à lever : E500(ii), sel, correcteur d'acidité : E170), beurre de cacao, sucre glace, cannelle moulue. Chocolat au lait : 33% de cacao min. Traces éventuelles de : œufs, fruits à coque.	<rien>	0.00%
Ingrédients : sucre ; sirop de glucose ; dextrose ; gélatine ; acidifiant : acide citrique ; arôme : sirop de caramel ; colorants : lutéine, charbon végétal ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba. Huile de colza, eau, vinaigre d'alcool, jaunes d'œufs, sucre, estragon (2,5%), échalote (2,4%), sel, cerfeuil, jus de citron à base de concentré, piment de Cayenne, arômes, amidon modifié, colorant : bêta-carotène, conservateur : sorbate de potassium. Ce produit peut contenir des traces de moutarde.	Thon Albacore (92%), eau (7%), sel (1%) Croustilles de céréales enrobées de chocolat au lait et de chocolat blanc arôme café chocolat blanc arôme café 48% (sucre, beurre de cacao, poudre de lait entier, arômes), chocolat au lait 38% (sucre, beurre de cacao, poudre de lait entier, pâte de cacao, émulsifiant : lécithines (soja), arôme naturel de vanille), céréales croustillantes 12% (farine de blé, gluten de blé, sucre, farine de malt de blé, poudre à lever : E500(ii), sel, correcteur d'acidité : E170), sucre glace. Chocolat au lait et blanc : cacao 33% min. Traces éventuelles de : œufs, fruits à coque. Croustilles de céréales enrobées de chocolat au lait et de pâte de spéculoos chocolat au lait 68% (sucre, beurre de cacao, poudre de lait entier, pâte de cacao, émulsifiant : lécithines (soja), arôme naturel de vanille), pâte de spéculoos 16% [Spéculoos 60% (farine de blé, sucre, huiles et graisses végétales (palme, colza), sirop de candi, poudres à lever : E500(ii) - carbonate acide d'ammonium, sel, cannelle), huiles et graisses végétales partiellement hydrogénées (tournesol, colza, palme), sucre, émulsifiants : lécithines - E472c, correcteur d'acidité : E330], céréales croustillantes 12% (farine de blé, gluten de blé, sucre, farine de malt de blé, poudre à lever : E500(ii), sel, correcteur d'acidité : E170), beurre de cacao, sucre glace, cannelle moulue. Chocolat au lait : 33% de cacao min. Traces éventuelles de : œufs, fruits à coque. Croustilles de céréales enrobées de praliné et de chocolat au lait avec praliné grain chocolat au lait 65% (sucre, beurre de cacao, poudre de lait entier, pâte de cacao, émulsifiant : lécithines (soja), arôme naturel de vanille), praliné 12% (noisettes 50%, sucre, émulsifiant : lécithines (soja), arôme naturel de vanille), céréales croustillantes 12% (farine de blé, gluten de blé, sucre, farine de malt de blé, poudre à lever : E500(ii), sel, correcteur d'acidité : E170), praliné grain 8% (sucre, amandes et noisettes torréfiées), beurre de cacao, agents d'enrobage : E414-E904. Chocolat au lait : cacao 33% min. Traces éventuelles de : œufs, autres fruits à coque.	15.38% 38.02%
Liste d'ingrédients : sel, exhausteurs de goût : glutamate, inosinate et guanylate de sodium, graisse de palme, amidon modifié de pomme de terre, viande de poule : 3,9%, graisse de poule : 3,5%, arômes, huile de tournesol, oignon, épices et aromates (curcuma, poivre, persil), caramel, maltodextrine, extrait de levure, antioxydant : extrait de romarin. Peut contenir : céleri.	sucre ; sirop de glucos ; dextrose ; gélatine ; acidifiant : acide citrique ; arôme ; sirop de caramel ; colorants : lutéine, charbon végétal ; agents d'enrobage : cire d'abeille blanche et jaune, cire de carnauba.	93.58%
OEUFS, farine de BLE, sucre, amidon de BLE, stabilisants : sorbitols- glycérol, cacao maigre en poudre (3,5%), émulsifiants : E472b - E477, poudres à lever : E450 - E500, sirop de glucose, LAIT écrémé en poudre, sel, conservateur : E202, épaississant : E410, arôme.	Huile de colza, eau, vinaigre d'alcool, jaunes d'œufs, sucre, estragon (2,5%), échalote (2,4%), sel, cerfeuil, jus de citron à base de concentré, piment de Cayenne, arômes, amidon modifié, colorant : bêta-carotène, conservateur : sorbate de potassium.Ce produit peut contenir des traces de moutarde.	98.02%
Liste des ingrédients : Purée de tomates mi réduite (64%), sucre, vinaigre, amidon modifié, sel, acidifiant : acide citrique	sel, exhausteurs de goût : glutamate, inosinate et guanylate de sodium, graisse de palme, amidon modifié de pomme de terre, viande de poule : 3,9%, graisse de poule : 3,5%, arômes, huile de tournesol, oignon, épices et aromates (curcuma, poivre, persil), caramel, maltodextrine, extrait de levure, antioxydant : extrait de romarin. Peut contenir : céleri.	90.08%
Suillus luteus (40%), Volvariella volvacea (25%), Pleurotus ostreatus (20%), lactarius deliciosus (10%), pholiota mutabilis (5%), eau, sel, acide citrique Chine, Vietnam, Chili, Europe de l'est	OEUFS, farine de BLE, sucre, amidon de BLE, stabilisants : sorbitols- glycérol, cacao maigre en poudre (3,5%), émulsifiants : E472b - E477, poudres à lever : E450 - E500, sirop de glucose, LAIT écrémé en poudre, sel, conservateur : E202, épaississant : E410, arôme.	99.62%
Liste des Ingrédients : Filets de maquereaux (61 %), eau, concentré de tomate (15%), huile de tournesol, vinaigre d'alcool, vinaigre de vin blanc, sel, vin blanc, épaississant : gomme xanthane, arômes.	Purée de tomates mi réduite (64%), sucre, vinaigre, amidon modifié, sel, acidifiant : acide citrique	79.37%
Salicornes de culture, eau, sel, acide citrique	Filets de maquereaux (61 %), eau, concentré de tomate (15%), huile de tournesol, vinaigre d'alcool, vinaigre de vin blanc, sel, vin blanc, épaississant : gomme xanthane, arômes.	78.57%
Ingrediénts : semoule de blé dur supérieure et de l'eau	Salicornes de culture, eau, sel, acide citrique	100.00%
Mélange de semoule de blé dur, lentilles, flocons de soja d'orge et d'avoine, carotte, petit pois, oignon et poireau déshydratés légèrement aromatisé	semoule de blé dur supérieure et de l'eau	75.93%
Boisson gazeuse aromatisée au jus de fruit à base de concentré S.Pellegrino Orange 33 cl (Aranciata) S.Pellegrino Citron 33 cl (Limonata) Le plaisir des fruits à l'italienne	Semoule de blé dur (71%), lentilles vertes précuites (6%), arômes naturels, flocon de soja (3%), carotte deshydratée (3%), petits pois deshydraté (2%), oignon rissolé deshydraté (1%) (oignon, huile de tournesol, antioxydant : extrait de romarin), huile de colza (1%), oignon deshydraté (1%), flocon d'orge (1%), flocon d'avoine (1%), poireau deshydraté.	25.21%
Boisson gazeuse aromatisée au jus de fruit à base de concentré S.Pellegrino Orange 33 cl (Aranciata) S.Pellegrino Citron 33 cl (Limonata) Le plaisir des fruits à l'italienne	<rien>	0.00%

Continued on next page

Liste d'ingrédients prédicté	Liste d'ingrédients cible	Sim.
Ingrediénts : Cerises noires 65%, sucre, gélifiant : pectine. Traces de beurre. Teneur totale en sucres : 50g pour 100g.	Cerises noires 65%, sucre, gélifiant : pectine. Traces de beurre.	53.72%
Eau, purée de tomate à base de concentré*, sucre, vinaigre, amidon modifié	Eau, purée de tomate à base de concentré*, sucre, vinaigre, amidon modifié de maïs, sel, conservateurs : E202-E211, arôme naturel	48.84%
INGREDIENTS : Pêches et poires en cubes (avec leur jus d'origine naturelle 6,25%*), segments d'ananas, raisins, eau, sirop de glucose-fructose, sucre, arôme naturel, acidifiant : acide citrique. *Sur une base de 100% de pêches et poires.	Pêches et poires en cubes (avec leur jus d'origine naturelle 6,25%*), segments d'ananas, raisins, eau, sirop de glucose-fructose, sucre, arôme naturel, acidifiant : acide citrique. *Sur une base de 100% de pêches et poires.	92.80%
Sirop de glucose, sucre, eau, stabilisants (E440i, E440ii, E415), acidifiants (E330, E450i), conservateur (E202). - 100% Semoule de blé dur de qualité supérieure - Contient du gluten - Fabriqué en France	Sirop de glucose, sucre, eau, stabilisants (E440i, E440ii, E415), acidifiants (E330, E450i), conservateur (E202). - 100% Semoule de blé dur de qualité supérieure	100.00% 48.94%
Farine de blé, beurre en poudre (7 %), sucre, émulsifiants, sel, dextrose, poudre de blancs d'oeufs, lactose, gluten de blé, poudre à lever : (E450, E500), arôme, antioxygène : E300, alpha amylase, hémicellulase.	Farine de blé, beurre en poudre (7 %), sucre, émulsifiants, sel, dextrose, poudre de blancs d'oeufs, lactose, gluten de blé, poudre à lever : (E450, E500), arôme, antioxygène : E300, alpha amylase, hémicellulase.	98.58%
LISTE DES INGREDIENTS SUCRE DE CANNE, EAU, JUS DE KIWI A BASE DE CONCENTRE (10%), ACIDIFIANT : ACIDE CITRIQUE, AROME, COLORANTS : E 102, E 133.	SUCRE DE CANNE, EAU, JUS DE KIWI A BASE DE CONCENTRE (10%), ACIDIFIANT : ACIDE CITRIQUE, AROME, COLORANTS : E 102, E 133.	83.45%
Flageolets verts. Jus : eau, sel, affermissant : chlorure de calcium (E509) <rien>	Flageolets verts. Jus : eau, sel, affermissant : chlorure de calcium (E509)	100.00%
Graine de courge, canneberge, graine de tournesol, graine de moutarde	Graine de courge, canneberge, graine de tournesol, graine de moutarde	100.00%
Liste d'ingrédients : vinaigre d'alcool, piment jalapeno, eau, sel, épaisseurs (E1422,E415), antioxydant (acide ascorbique).	vinaigre d'alcool, piment jalapeno, eau, sel, épaisseurs (E1422,E415), antioxydant (acide ascorbique).	81.25%
40% Tolérance +5% Riz long grain de qualité supérieure mg/kg	AMANDES décortiquées <rien>	5.00% 0.00%
Ingrédients : Sucre, sirop de glucose, farine de BLÉ, graisse de coco totalement hydrogénée, pâte de cacao, LAIT concentré sucré, sirop de sucre inverti, beurre de cacao, LAIT écrémé en poudre, poudre de lactosérum (de LAIT), matière grasse LAITIERE, huiles végétales (illipé,mangue,sal,karité et palme en proportions variables), huile de colza, sirop de sucre candi, cacao maigre en poudre, émulsifiant (lécithine de SOJA), lactose et protéines de LAIT, LAIT entier en poudre, sel, colorant (E150c), poudre à lever (carbonates acide d'ammonium, carbonate acide de sodium), arômes.PEUT CONTENIR OEUFS ET SESAME	100% Huile d'ARACHIDE raffinée Sucre, sirop de glucose, farine de BLÉ, graisse de coco totalement hydrogénée, pâte de cacao, LAIT concentré sucré, sirop de sucre inverti, beurre de cacao, LAIT écrémé en poudre, poudre de lactosérum (de LAIT), matière grasse LAITIERE, huiles végétales (illipé,mangue,sal,karité et palme en proportions variables), huile de colza, sirop de sucre candi, cacao maigre en poudre, émulsifiant (lécithine de SOJA), lactose et protéines de LAIT, LAIT entier en poudre, sel, colorant (E150c), poudre à lever (carbonates acide d'ammonium, carbonate acide de sodium), arômes.PEUT CONTENIR OEUFS ET SESAME	0.00% 0.00% 96.75%
Oeuf(Poudre de blanc d'oeuf) Sucre, amidon de maïs, arôme vanille à 32% minimum de cacao fortement dégraissé	<rien> Sucre, amidon de maïs, arôme vanille Sucre, cacao maigre en poudre (beurre de cacao : 11% minimum), arôme vanille. Cacao : 32% minimum	0.00% 100.00% 21.65%
Tofu* 76.6% (eau, soja* dépelliculé 20.7% , gélifiants : sulfate de Calcium, Nigari), Oignons* 7.5%, Concentré de tomate* 7.5%, Flocons d'avoine* complète, Sauce de soja* (eau, soja* entier, blé* entier, sel marin, alcool*, ferment), Basilic*, Amidon de blé*, Flocons de millet*, Protéine de soja* texturée, Ail*, Persil*, Sel de mer, Huile de tournesol*, Ciboulette*, Extrait de levure*, Poivre*, Thym*, Origan*	Tofu* 76.6% (eau, soja* dépelliculé 20.7%, gélifiants : sulfate de Calcium, Nigari), Oignons* 7.5%, Concentré de tomate* 7.5%, Flocons d'avoine* complète, Sauce de soja* (eau, soja* entier, blé* entier, sel marin, alcool*, ferment), Basilic*, Amidon de blé*, Flocons de millet*, Protéine de soja* texturée, Ail*, Persil*, Sel de mer, Huile de tournesol*, Ciboulette*, Extrait de levure*, Poivre*, Thym*, Origan*. * Ingrédients issus de l'agriculture biologique	88.45%
CocaCola Light mini 8 x 150 mlEAN5449000239808Marché(s) cible(s) zéfiécolorant : E150acidifiants : acide phosphorique, acide citriqueéducolorants : aspartame, acésulfameKarônes naturels dont caféinecontient une source de phénylalanineNutritionPour : 100mlPour : 150ml(%*)Energie:1kJ/1.5kJ/0.2kcal0.3kcal(0%)Matières grasses : 0g(0%)Glucides : 0g(0%)dont sucres : 0g(0%)Protéines : 0g(0%)Fibres : 0g(0%)Lipides : 0g(0%)Valeur nutritionnelle : calculéepour 100mlPour : 150mlÉnergie (kJ)11.5Énergie (kcal)20.3Matières grasses (g)0dont acides gras saturés (g)0Glucides (g)0dont sucres (g)0Protéines (g)0Sel (g)0Nutrition sur le devant de l'emballagePar portion150ml1.5kJ0.3kcal0%0g0%0g0%0g0%100ml : 1kJ / 0.2kcal.* Apports de référence pour un adulte type (8400kJ / 2000 kcal). Description du produitMarqueCoca-ColaCaractéristiquesGoulé légerSans calorieMarque normaliséeMarque CocaColaSousmarquesSousmarques LightNom de produit réglementéBoisson rafraîchissante aux extraits végétaux, avec éducolorants.CommercialisationNom de la sociétéCoca-Cola European Partners FranceAdresse de la société17 chemin de Bretagne CS 80050 92784 Issy les Moulineaux, CEDEX 9Informations diversesConditionné avec l'autorisation de The CocaCola Company.Description supplémentairePensez au tri ! Consigne pouvant varier localement > www.consignesdetri.frTrademark InformationCoca-Cola light est une marque déposée de The CocaCola Company. © 2018 The CocaCola CompanyAllégations NutritionnellesSans calorieHygiène & Mode de vieAdditifsÉducolorants artificiels ContenantStockage et UtilisationType De StockageTempérature ambiantePréparation et utilisationNe pas agiter. Se boit très fraisStockageConserver dans un endroit tempéré, sec et sans odeur.PackTaille du paquet8 canettes de 150mlDimension NumériqueDimension numérique 150 Infos RecyclageCarton RecyclablePays d'origine FranceType d'EmballageType BoîteOrigineTexte libre sur l'origine Produit en FranceRecyclage Aut...	Fraîcheur et gélification au ga-colorant : E150acidifiants : acide phosphorique, acide citriqueéducolorants : aspartame, acésulfameKarônes naturels dont caféinecontient une source de phénylalanineNutritionPour : 100mlPour : 150ml(%*)Energie:1kJ/1.5kJ/0.2kcal0.3kcal(0%)Matières grasses : 0g(0%)Glucides : 0g(0%)dont sucres : 0g(0%)Protéines : 0g(0%)Fibres : 0g(0%)Lipides : 0g(0%)Valeur nutritionnelle : calculéepour 100mlPour : 150mlÉnergie (kJ)11.5Énergie (kcal)20.3Matières grasses (g)0dont acides gras saturés (g)0Glucides (g)0dont sucres (g)0Protéines (g)0Sel (g)0Nutrition sur le devant de l'emballagePar portion150ml1.5kJ0.3kcal0%0g0%0g0%0g0%100ml : 1kJ / 0.2kcal.* Apports de référence pour un adulte type (8400kJ / 2000 kcal). Description du produitMarqueCoca-ColaCaractéristiquesGoulé légerSans calorieMarque normaliséeMarque CocaColaSousmarquesSousmarques LightNom de produit réglementéBoisson rafraîchissante aux extraits végétaux, avec éducolorants.CommercialisationNom de la sociétéCoca-Cola European Partners FranceAdresse de la société17 chemin de Bretagne CS 80050 92784 Issy les Moulineaux, CEDEX 9Informations diversesConditionné avec l'autorisation de The CocaCola Company.Description supplémentairePensez au tri ! Consigne pouvant varier localement > www.consignesdetri.frTrademark InformationCoca-Cola light est une marque déposée de The CocaCola Company. © 2018 The CocaCola CompanyAllégations NutritionnellesSans calorieHygiène & Mode de vieAdditifsÉducolorants artificiels ContenantStockage et UtilisationType De StockageTempérature ambiantePréparation et utilisationNe pas agiter. Se boit très fraisStockageConserver dans un endroit tempéré, sec et sans odeur.PackTaille du paquet8 canettes de 150mlDimension NumériqueDimension numérique 150 Infos RecyclageCarton RecyclablePays d'origine FranceType d'EmballageType BoîteOrigineTexte libre sur l'origine Produit en FranceRecyclage Aut...	1.07%

Continued on next page

Liste d'ingrédients prédicté	Liste d'ingrédients cible	Sim.
BISCUITS AU CHOCOLAT AU LAIT Ingrédients : chocolat au lait (38%)(sucre, beurre de cacao, pâte de cacao, lait écrémé en poudre, beurre pâtissier, émulsifiant : lécithine de soja, arôme) - farine de blé - sucre - graisse végétale (palme, colza) - sel - lactose et protéines de lait - lait entier en poudre - émulsifiant : lécithine de soja - poudres à lever : carbonates de sodium, carbonates d'ammonium, diphosphate disodique - noix de coco - arôme - œuf entier en poudre. Peut contenir des traces de sésame, de fruits à coque, d'avoine, d'orge et d'autres glucens.	chocolat au lait (38%)(sucre, beurre de cacao, pâte de cacao, lait écrémé en poudre, beurre pâtissier, émulsifiant : lécithine de soja, arôme) - farine de blé - sucre - graisse végétale (palme, colza) - sel - lactose et protéines de lait - lait entier en poudre - émulsifiant : lécithine de soja - poudres à lever : carbonates de sodium, carbonates d'ammonium, diphosphate disodique - noix de coco - arôme - œuf entier en poudre. Peut contenir des traces de sésame, de fruits à coque, d'avoine, d'orge et d'autres glucens.	90.55%
Filets de maquereaux scomber scombrus 60% (Pêchés en Atlantique Nord-Est), eau, concentré de tomates (30% de la sauce), huile de colza, vinaigre, sel, épaississant : gomme xanthane, arômes naturels. Allergènes majeurs (hors poisson) : absence. Absence d'OGM.	Filets de maquereaux scomber scombrus 60% (Pêchés en Atlantique Nord-Est), eau, concentré de tomates (30% de la sauce), huile de colza, vinaigre, sel, épaississant : gomme xanthane, arômes naturels.	68.99%
Carottes, eau, sucre, sel, vinaigre d'alcool, acidifiant : acide lactique. Présence forte de CELERI	Carottes, eau, sucre, sel, vinaigre d'alcool, acidifiant : acide lactique. Présence forte de CELERI	99.02%
Riz long étuvé de qualité supérieure issu de l'Agriculture Biologique Liste des Ingrédients : sucre*, LAIT en poudre*, beurre de cacao*, pâte de cacao*, émulsifiant : lécithine de tournesol (E322), extrait de vanille* * matière première issue de l'agriculture biologique	Riz long étuvé de qualité supérieure issu de l'Agriculture Biologique sucre*, LAIT en poudre*, beurre de cacao*, pâte de cacao*, émulsifiant : lécithine de tournesol (E322), extrait de vanille* * matière première issue de l'agriculture biologique cacao : 27% minimum	100.00% 75.98%
76 g dont sucre 0 g	<rien>	0.00%
Graisses	Tilleul (100%).	0.00%
Noix	<rien>	0.00%
Eau, haricots verts, sel.	Eau, haricots verts, sel.	100.00%
Légumes 43,2 % (pomme de terre, oignon, carotte, tomate, poireau) - Amidon modifié de pomme de terre - Extrait de levure - Sirop de glucose de maïs - Huile de colza - Sucre - Arôme naturel - Ail - Curcuma.	Légumes 43,2 % (pomme de terre, oignon, carotte, tomate, poireau) - Amidon modifié de pomme de terre - Extrait de levure - Sirop de glucose de maïs - Huile de colza - Sucre - Arôme naturel - Ail - Curcuma.	99.51%
Liste des Ingrédients : sucre, pâte de cacao, beurre de cacao, cacao maigre en poudre, émulsifiant : lécithine de tournesol (E322), arôme vanille	sucre, pâte de cacao, beurre de cacao, cacao maigre en poudre, émulsifiant : lécithine de tournesol (E322), arôme vanille cacao : 50% minimum	69.66%
100 g de sucre	Maltodextrine, édulcorant : sucralose (1,23%).	17.39%
Farine de BLÉ 63%, eau, sucre, huile de colza, sel, vinaigre, levure, farine de fèves, gluten de BLÉ, arôme (contient alcool), extract d'acérola. Peut contenir	Farine de BLÉ 63%, eau, sucre, huile de colza, sel, vinaigre, levure, farine de fèves, gluten de BLÉ, arôme (contient alcool), extract d'acérola. Peut contenir des traces d'OEufs, SOJA, LAIT, FRUITS À COQUE, GRAINES DE SÉSAME.	70.35%
Croquant sésame chocolat Barre croquante au sésame et au chocolat, source de fibres et riche en magnésium. Composition : Graines de sésame 37%, chocolat de couverture 25% (sucre, pâte de cacao, beurre de cacao, émulsifiant : lécithinesdesoja,E476,arôme),siropdeglucose,sucre.	Graines de sésame 37%, chocolat de couverture 25% (sucre, pâte de cacao, beurre de cacao, émulsifiant : lécithines de soja, E476, arôme), sirop de glucose, sucre. Fabriqué dans un atelier qui utilise de l'arachide, des fruits à coque et du lait.	27.11%
Liste ingrédients : Sirop de glucose-fructose, framboises 35%, sucre, gélifiant : pectines, acidifiant : acide citrique. Préparée avec 35g de fruits pour 100g de produit fini. Teneur totale en sucres : 60g pour 100g.	Sirop de glucose-fructose, framboises 35%, sucre, gélifiant : pectines, acidifiant : acide citrique.	45.87%
INGREDIENTS : Jus d'orange à base de concentré	Jus d'orange à base de concentré	65.96%
Jus d'orange à base de concentré		
Ingrediénts : Farine de BLÉ, sucre, huile de colza,, cacao maigre en poudre 6,5 %, sirop de glucose-fructose, amidon de BLÉ, poudre à lever (carbonate de potassium, carbonate d'ammonium, carbonate de sodium).huile de palme,sel, émulsifiants (lécithine de SOJA), arôme .PEUT CONTENIR LAIT.	Farine de BLÉ, sucre, huile de colza,, cacao maigre en poudre 6,5 %, sirop de glucose-fructose, amidon de BLÉ, poudre à lever (carbonate de potassium, carbonate d'ammonium, carbonate de sodium).huile de palme,sel, émulsifiants (lécithine de SOJA), arôme .PEUT CONTENIR LAIT.	94.16%
Piment rouge fort équeuté* (85%), cumin, ail moulu (3%), eau, arôme naturel d'ail, sel, sorbate de potassium. (* présence de sulfites)	Piment rouge fort équeuté* (85%), cumin, ail moulu (3%), eau, arôme naturel d'ail, sel, sorbate de potassium. (* présence de sulfites)	88.67%

Annexe G

LES NOTEBOOKS DE CE PROJET

,

ground_truth_constitution

Pierre MASSÉ

June 11, 2020

1 Constitution de l'échantillon de données étiquetées

L'objet de ce notebook est de produire un échantillon données du PIM, avec les fiches techniques associées. Elles seront ensuite associées manuellement à la liste d'ingrédients qu'elles contiennent.

1.1 Récupération des données

1.1.1 Préambule technique

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[2]: # imports and customization of display
import os
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline

from src.pimapi import Requester
from src.pimest import ContentGetter, PDFContentParser
```

```
[3]: # monkeypatch _repr_latex_ for better inclusion of dataframes output in report
def _repr_latex_(self, size='scriptsize',):
    return(f"\\"\\resizebox{{\\linewidth{{!}}}}{{{size}}}{{\\begin{{\\size}}}{{\\centering{{\\self.to_latex()}}}{{\\end{{\\size}}}}}}")
pd.DataFrame._repr_latex_ = _repr_latex_
```

1.1.2 Récupération des données, et de la présences de fiches techniques

Pour constituer l'échantillon, on va d'abord extraire quelques informations du PIM, et particulièrement le type de produit. On récupérera aussi le fait que les produits ont ou non une fiche technique fournisseur associée.

```
[4]: requester = Requester('prd')
# Let's fetch the full content of PIM system
requester.fetch_all_from_PIM()
requester.result
```

Done

```
[4]: [<Response [200]>,
<Response [200]>]
```

```
[5]: mapping = {'uid': 'uid',
             'designation': 'title',
             'state': 'state',
             'ingredients': 'properties.pprod:ingredientsList',
             'type': 'properties.pprod:typeOfProduct'}
df = requester.file_report_from_result(mapping=mapping, index='uid') # , record_path='entries')
df.sample(5)
```

uid	designation	state	ingredients	type	has_supplierdatasheet	has_supplierlabel
f2fc9f7a-ff2f-4502-a00e-7304fb7e6ee0	Mayonnaise allégée en eau 5,1 kg VALTONIA	product.validate	Eau, huile de colza 25 %, vinaigre, jaune d'OE...	grocery	False	False
3ec9c99-7ba2-449d-afc7-b22e465a192a	MARJULAINNE 1KG	product.waiting.sending.supplier	None	grocery	False	False
e8be45bd-070f-4086-9044-fa3cc65a40e4	TORK LINETTES IMPRÉGNÉES POUR NETTOYAGE DE SU...	product.validate	None	chemistry	True	True
a87747cb-e570-444e-8759-b74040451436	Baquette charcutière 150 cc en sachet de 500 ...	product.validate	None	hygiene	True	True
afee12cf-177e-4a69-9539-8cb689442503	DESTR D'EDDERS AIRATEXTILES 7500CX6 DEDDOR U2	product.waiting.supplier.validation	None	chemistry	False	False

1.2 Constitution de l'échantillon

On va constituer l'échantillon en appliquant les règles suivantes : - on construit un échantillon de 500 produits - on conserve les produits de type Epicerie et Boisson non alcoolisée - on conserve les produits qui possèdent une fiche technique fournisseur - on fait un échantillon stratifié par type de produit (Epicerie / Boisson)

```
[6]: filtered_df = df.loc[(df.type.isin(['grocery', 'nonAlcoholicDrink']))
                           & (df.has_supplierdatasheet)]
train, ground_truth_df = train_test_split(filtered_df,
                                           test_size=500,
                                           random_state=42,
                                           stratify=filtered_df.type)
ground_truth_df.sample(5)
```

uid	designation	state	ingredients	type	has_supplierdatasheet	has_supplierlabel
2a13382a-384d-4e50-8eef-27366988acef8	Penne Rigate sans-gluten en boîte 400 g BARILLA	product.waiting.supplier.validation	Farine de maïs blanc (60%), farine de maïs jau...	grocery	True	False
1806553-3115-49ed-b02c-71140407a789	Spécialité à la bûche en compotée 100 g CHA...	product.validate	Farine de pomme 95%, poudre de biscuit 4,2% (...	grocery	True	True
a1454f0-3115-49ed-b02c-71140407a789	Barquette soupe en boîte 10 g GANDUM	product.waiting.supplier.validation	Farine de riz (Oryza Sativa Linnae)	grocery	True	False
3a4e72b-b365-465b-9a51-48b78bde4cf2	GRANINE DU COURGE	product.validate	Graine de courge	grocery	True	True
8c147f0a-2388-4b1a-ac4a-51feb602278	Lasagnette à l'ancienne aux 7 sausages en sac 3 kg...	product.waiting.supplier.validation	Semoule de BLE dur de qualité supérieure, oeUF...	grocery	True	False

Remarque : malgré l'utilisation d'un random_state fixé, l'échantillon généré n'est pas toujours le même à chaque exécution. En effet, comme la liste de produits varie au fil du temps (nouveaux référencements, périmètre des filtres qui change), le résultat du train_test_split peut varier.

Il s'agit ici seulement d'illustrer la méthode utilisée.

1.3 Export des pièces jointes du PIM et constitution du fichier d'étiquettes

On exporte ensuite le contenu du PIM sur le disque, afin d'avoir les fiches techniques simplement à disposition.

Remarque : des lignes dans ce paragraphe pour ne pas télécharger à nouveau les pièces jointes ni écraser le résultat de l'étiquetage manuel.

```
[ ]: requester.fetch_list_from_PIM(ground_truth_df.index, batch_size=20)
# requester.dump_data_from_result(update_directory=False, root_path=os.path.join('.', 'ground_truth_to_del'))
# requester.dump_files_from_result(update_directory=False, root_path=os.path.join('.', 'ground_truth_to_del'))
```

On exporte également au format csv les uids des produits et les libellés associés (pour s'assurer qu'il n'y a pas eu de confusion lorsqu'on lit une fiche technique).

```
[ ]: # ground_truth_df['designation'].to_csv(os.path.join('.', 'ground_truth_to_del', 'uid.csv'),
#                                         header=True,
#                                         encoding='utf-8-sig')
```

On teste également la possibilité de recharger les données depuis le fichier csv, une fois qu'il a été renseigné à la main dans excel.

```
[ ]: # pd.read_csv(os.path.join('..', '..', 'ground_truth', 'manually_labelled_ground_truth.csv'),
#               sep=',',
#               encoding='latin-1',
#               index_col='uid')
```

1.4 Résultat de l'étiquetage manuel

Le résultat de l'étiquetage manuel est le suivant :

```
[7]: df_gt = pd.read_csv(os.path.join('..', '..', 'ground_truth', 'manually_labelled_ground_truth.csv'),
                      sep=',',
                      encoding='latin-1',
                      index_col='uid')

def to_latex_newline(text):
    return(text.replace('\n', ' '))
```

```

with pd.option_context("max_colwidth", 1000):
    pass
#     print(df_gt)
#     df_gt.to_latex(
#         Path('..') / 'tbls' / 'ground_truth.tex',
#         index=False,
#         index_names=False,
#         column_format='p{5cm}p{10cm}',
#         formatters=[to_latex_newline, to_latex_newline],
#         longtable=True,
#         na_rep='--',
#         escape=True,
#     )
#
df_gt.sample(5)

```

[7]:

	designation	ingredients
uid		
83dc5272-5e87-47b7-bd06-271bbac620a4	Flocon d'érable en sachet 170 g COULEUR QUEBEC	sirup d'érable pur à 100 %.
bbddc4ed-6d16-475c-acel-851c8b32d28b	DEMI POIRES WILLIAMS AU SIROP LÉGER	NaN
a3d51821-275c-4471-8df4-bifalefede25	Purée d'épinard sans sel ajouté en sachet 1 kg...	Pommes de terre 59,5 % - Epinards 40 % - Amido...
6dfae8fd-6111-4a57-862e-c20a39a195e0	Pain de mie sans croûte en tranches en paquet ...	Farine de BLÉ 63%, eau, sucre, huile de colza,...
1678fd52-dc4b-4818-81de-b9c1581dc272	Spécialité pomme-abricot en boîte 5/1 VALADE E...	Pommes 78%, purée d'abricots à base de concent...

1.5 Comparatif entre les données étiquetées et le contenu du PIM

On peut comparer le contenu des listes d'ingrédients du PIM et les données étiquetées.

[8]:

```

requester = Requester('prd')
requester.fetch_all_from_PIM()
requester.result

```

Done

[8]:

```

[<Response [200]>,
 <Response [200]>]

```

On récupère le contenu du PIM

[9]:

```

df = requester.result_to_dataframe()
pim_ds = df['properties.pprod:ingredientsList']
pim_ds.sample(5)

```

[9]:

uid	ingredients
e061a124-b312-4e5b-8311-d3f374ce0c01	80% Bolets jaunes Suillus Luteus, 20% Cèpes Bo...
f0a62940-5055-4433-83d0-8becdf6561e3	Sirop de glucose, sucre, eau, gélatine, acidif...
7b64e7a7-8c7c-45e3-80e9-48fdc6d44c64	None
115bf599-c6fa-40f7-ac08-622901756d0c	Haricots verts, eau, sel
1afa5387-e2e3-4fc2-b991-b896f7feacc9	None

Name: properties.pprod:ingredientsList, dtype: object

On charge le csv des données étiquetées :

[10]:

```

df_gt = pd.read_csv(os.path.join('..', '..', 'ground_truth', 'manually_labelled_ground_truth.csv'),
                    sep=';',
                    encoding='latin-1',
                    index_col='uid')
df_gt.sample(5)

```

[10]:

uid	designation	ingredients
d39f16bc-29f8-40b9-9d56-43295bdf5961	FLOWPACK PATAREV HIPPOPOTAMUS	PÂTE À MÂCHER ACIDE, AROMATISÉE : GOÛT FRAMBOI...
8097a8a8-86c0-4f9a-8c75-6d825a979e8c	Sucre cristal en sac 5 kg DADDY	Nan
4f83306f-66de-4545-9b12-779057b61ae	Nappage miroir neutre en eau 7 kg ANCEL	Sirop de glucose, sucre, eau, stabilisants (E4...
93e5d2af-10c5-4853-a437-b013673310cb	Riz long de Camargue IGP en sac 5 kg CANAVERE	Nan
5cb7f05a-3b2c-440e-af0d-01843fb38cbf	Assortiment de Malabar magic blue 3 parfums en...	Sucre, Gomme base, Sirop de glucose, Acidifiant...

Comme l'index de la series des données issues du PIM, et du dataframe de la ground truth est le même (l'uid du produit), on peut faire très simplement la jointure via la méthode join :

```
[11]: merged = (df_gt.join(pim_ds)
              .rename({'ingredients': 'Ingrédients de la ground truth',
                       'products.pprod:ingredientsList': 'Ingrédients du PIM'},
                     axis=1)
              )
merged.sample(5)
```

uid	designation	Ingrédients de la ground truth	Ingrédients du PIM
1de02b1c-f17e-4d46-b90f-3a6c37ecf6aa	AMANDES DECORTIQUEES GRILLEES SANS SEL	AMANDES décortiquées	AMANDES grillées
c2ef743e-f3f2-4e8a-aab0-1e6cbeb71666	Gâteau aux céréales et aux graines de tournesol...	Farine de BLÉ 20% - Huile de colza - OEUFS ent...	Huile de colza - Farine de BLÉ 19,5% - OEUFS ...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8	Macaroni en sachet 500 g PANZANI	- 100% Semoule de BLE dur de qualité supérieure	100% Semoule de BLE dur de qualité supérieure
5d24bc08-f7f6-4eb6-800a-55aa3a6dc76d	Boissons énergisante en canette 47,3 cl RED BULL	eau gazéifiée, saccharose, glucose, correcteur...	Eau gazéifiée, saccharose, glucose, acidifiant...
70500268-802d-4211-93ba-9edb16e0e7a3	COLIS KERMESSE 2019 A.P. * 22+8*	TAGADA\nsucré; sirop de glucose; gelatine; aci...	GADA: sucre; sirop de glucose; gélatine; acide...

On peut compter les égalités strictes entre les ingrédients du PIM et ceux de la ground truth :

```
[12]: merged['equals'] = (merged['Ingrédients du PIM'] == merged['Ingrédients de la ground truth'])
merged['equals'].value_counts()
```

```
[12]: False      452
      True       48
      Name: equals, dtype: int64
```

Seules 50 listes d'ingrédients sont strictement identiques. Si on compare les listes qui ne le sont pas, on obtient :

```
[13]: diff = merged.loc[~merged['equals'], ['Ingrédients du PIM', 'Ingrédients de la ground truth']]
for i in range(6):
    print('+'*60)
    print('Issu du PIM :')
    print(diff.iloc[i].loc['Ingrédients du PIM'])
    print('-----')
    print('Issu de la ground truth :')
    print(diff.iloc[i].loc['Ingrédients de la ground truth'])
    print('+'*60+\n')
```

```
+++++
Issu du PIM :
Farine de BLE T65, eau, levure, huile de colza, sel, vinaigre de cidre, assaisonnement poudre de curry,
agent de traitement de la farine : acide ascorbique, émulsifiant : E471
-----
```

```
Issu de la ground truth :
Farine de blé T65, eau, levure, vinaigre de cidre, huile de colza, assaisonnement poudre de curry, sel,
acide ascorbique, émulsifiant : E471
+++++
```

```
+++++
Issu du PIM :
100% Semoule de BLE dur de qualité supérieure
-----
```

```
Issu de la ground truth :
- 100% Semoule de BLE dur de qualité supérieure
- Contient du gluten
Si le numéro de lot contient la lettre N : peu contenir de l'oeuf
+++++
```

```
+++++
Issu du PIM :
Fève de tonka, taux de coumarine compris entre 1 et 3,5%
-----
```

```
Issu de la ground truth :
fève de tonka (graines ridées de 25 à 50mm de long)
Taux de coumarine compris entre 1 et 3,5 %
+++++
```

```
+++++
Issu du PIM :
Aubergine 60,5% (aubergine, huile de tournesol), eau, oignon, huile de tournesol, jus de citron, concentré
de tomate, huile d'olive vierge extra 2%, ail, sel, persil, basilic, poivre, thym, romarin.
```

Issu de la ground truth :
Aubergine 60,5% (aubergine, huile de tournesol), eau, oignon, huile de tournesol, jus de citron, concentré de tomate, huile d'olive vierge extra (2%), ail, sel, persil, basilic, poivre, thym, romarin.
++++++

++++++
Issu du PIM :
Ingrédients : Myrtille Cassis : Fruits (myrtilles 41%, cassis 9%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectines de fruits. Fraise Groseille : Fruits (fraises 27 %, groseilles 23 %), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectines de fruits. Abricot Pêche : Fruits (abricots 34%, pêches 16%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectines de fruits. Orange Douce Mandarine : Fruits (oranges douces 37%, mandarines 3%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectines de fruits.

Issu de la ground truth :
Confiture de myrtilles et de cassis
fruits (myrtilles 41%, cassis 9%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits.
Confiture de fraises et de groseilles
fruits (fraises 27 %, groseilles 23 %), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits.
Confiture d'abricots et de pêches
fruits (abricots 34%, pêches 16%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits.
Marmelade d'oranges douces et de mandarines
fruits (oranges douces 37%, mandarines 3%), sucre, sucre roux de canne, jus de citrons concentré, gélifiant : pectine de fruits.
++++++

++++++
Issu du PIM :
Pommes 95%, sirop de glucose-fructose, arôme, antioxydant : acide ascorbique

Issu de la ground truth :
Pommes 95%, sirop de glucose-fructose, arôme, antioxydant: acide ascorbique.
++++++

On peut sortir un tableau des données en écart, de manière basique :

```
[14]: with pd.option_context("max_colwidth", 100000):
    tex_str = (
        diff.sample(10, random_state=44)
        .to_latex(index=False,
                  index_names=False,
                  column_format='p{7cm}p{7cm}',
                  na_rep='--',
                  )
        .replace(r'\textbackslash n', '\\newline ')
    )
#     print(tex_str)

# with open(Path('..') / 'tbls' / 'ingredient_comparison.tex', 'w') as file:
#     file.write(tex_str)
diff.sample(10, random_state=44)
```

uid	Ingrédients du PIM	Ingrédients de la ground truth
bf633f9f-a89a-499b-afb8-1b874a477b08	Champignons, eau, sel , acidifiant : acide cit...	champignons, eau, sel, acidifiant : acide citr...
84d5c32f-92d0-49c9-9151-d1fd65238a2a	LAIT écrémé	Lait écrémé.
15d6958c-025e-43a6-9f3b-a0d923a61c3f	Pommes en tranches (43%), pêches en tranches (...	Pommes en tranches (35 à 56%), pêches en tranc...
eeca38ed-ff9b-467f-874d-298a350bd6c5	Pâtes alimentaires de semoule de BLÉ dur de qu...	SEMOUTRE DE BLE' DUR de qualité supérieure
6db330c3-26d0-4a46-93a5-74e704b107ff	Ecorce de citron (57%), sirop de glucose-fruct...	NaN
b70ed45-57ec-497d-bf18-af14fbbe955	BLE dur entier précuít	NaN
e67341d8-350f-46f4-9154-4dbb8035621	Sucré roux de canne*(64%), amidon de maïs*, p...	Sucré roux de canne* (64%), amidon de maïs*, ...
f9af1c71-59dd-4d11-8938-aa726ccffe6c	Eau, huile de tournesol, miel 10%, moutarde à ...	Eau, huile de tournesol, miel 10%, moutarde à ...
93fb1748-efa5-4679-b67e-51ff121c69e8	sucré de canne, eau, jus de mirabelle à base d...	SUCRE DE CANNE, EAU, JUS DE MIRABELLE A BASE D...
f9f2c425-07cd-43ef-aabe-ec777e89a6e7	Sucré 49,0%, NOISETTES 25,0%, AMANDES 25,0%, é...	sucré 49,0%; noisettes 25,0%; amandes 25,0%; é...

1.6 Analyse du contenu des pièces jointes téléchargées

On peut faire une estimation des pièces jointes dont les textes sont extractibles. On commence simplement par lister les pièces jointes relatives à la ground truth qui ont été téléchargées.

```
[15]: p = Path('..') / 'ground_truth_to_del'
```

```
[19]: files_df = pd.DataFrame(list(p.glob('**/*.pdf')), columns=['path'])
files_df['type'] = files_df['path'].apply(lambda x: x.name).apply(lambda x: x.split('.')[0])
files_df['uid'] = files_df['path'].apply(lambda x: x.parent.name)
files_df.set_index('uid', inplace=True)
files_df.sample(5)
```

[19]:

uid	path	type
ed969c94-33a2-4a82-bc84-0d4adc908f5c	ground_truth_to_del/ed969c94-33a2-4a82-bc84-0d...	FTF
244e14b8-8291-4315-8ca8-53fa85fcf23f6	ground_truth_to_del/244e14b8-8291-4315-8ca8-53...	FTF
f42e19ae-d433-410d-a28d-ca01127b0ded	ground_truth_to_del/f42e19ae-d433-410d-a28d-ca...	FTF
57877d62-ace0-44ad-81bf-ed63b7a37877	ground_truth_to_del/57877d62-ace0-44ad-81bf-ed...	FTF
194419d0-d9f2-4799-81ac-d9e3aa77fd27	ground_truth_to_del/194419d0-d9f2-4799-81ac-d9...	FTF

On utilise les transformateur du module pimest pour récupérer le contenu de ces fichiers dans le dataframe.

```
[ ]: transformer = make_pipeline(ContentGetter(), PDFContentParser())
files_df = transformer.fit_transform(files_df)
files_df
```

```
[ ]: files_df['empty'] = (files_df['text'].apply(lambda x: x.strip()) == '')
files_df.sample(5)
```

```
[ ]: (files_df.pivot_table(values='empty',
                           index='type',
                           aggfunc=['sum', 'count', 'mean'],
                           )
     .swaplevel(axis=1)
     .rename({'empty': 'Fichiers vides',
              'sum': 'Nombre de fichiers vides',
              'count': 'Nombre total de fichiers',
              'mean': 'Taux de vides',
              }, axis=1)
     .rename({'Etiquette': 'Etiquettes',
              'FTF': 'Fiches techniques',
              })
     .to_latex(
        #Path('../') / 'tbls' / 'empty_attached_files.tex',
        column_format='lccc',
        bold_rows=True,
        index_names=False,
        formatters=[lambda x: str(int(x)),
                   lambda x: str(int(x)),
                   lambda x: f'{int(x * 100):d}%',
                   ]
     )
)
pass
```

Text_analysis

Pierre MASSÉ

June 11, 2020

1 Préambule

1.1 Imports

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[201]: # imports and customization of display
import os
from functools import partial
from collections import defaultdict
import re
import numpy as np
from scipy.stats import linregress
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import clone
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.pipeline import Pipeline
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap
from sklearn.preprocessing import StandardScaler
from gensim.models import word2vec

from matplotlib import pyplot as plt
import matplotlib.patches as mpatch
import matplotlib.cm as cm
import matplotlib.colors as colors
from matplotlib.colors import Normalize
import matplotlib.ticker as mtick
import seaborn as sns

from src.pimest import ContentGetter
from src.pimest import PathGetter
from src.pimest import PDFContentParser
from src.pimest import BlockSplitter
from src.pimest import SimilaritySelector
from src.pimest import custom_accuracy
from src.pimest import text_sim_score
from src.pimest import text_similarity
from src.pimest import build_text_processor
```

1.2 Acquisition des données de la ground truth

```
[4]: ground_truth_df = pd.read_csv(Path('..') / '..' / 'ground_truth' / 'manually_labelled_ground_truth.csv',
                                    sep=';',
                                    encoding='latin-1',
                                    index_col='uid')
ground_truth_uids = list(ground_truth_df.index)

acqui_pipe = Pipeline([('PathGetter', PathGetter(ground_truth_uids=ground_truth_uids,
                                                train_set_path=Path('..') / '..' / 'ground_truth',
```

```

        ground_truth_path=Path('..') / '..' / 'ground_truth',
    )),
    ('ContentGetter', ContentGetter(missing_file='to_nan')),
    ('ContentParser', PDFContentParser(none_content='to_empty')),
],
verbose=True)

texts_df = acqui_pipe.fit_transform(ground_truth_df)
texts_df['ingredients'] = texts_df['ingredients'].fillna('')
texts_df

```

[Pipeline] ... (step 1 of 3) Processing PathGetter, total= 0.1s
[Pipeline] ... (step 2 of 3) Processing ContentGetter, total= 0.1s
Launching 8 processes.
[Pipeline] ... (step 3 of 3) Processing ContentParser, total= 39.1s

```
[4]: designation \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70 Concentré liquide Asian en bouteille 980 ml CHEF
d183e914-db2f-4e2f-863a-a3b2d054c0b8 Pain burger curry 80 g CREATIV BURGER
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 Macaroni en sachet 500 g PANZANI
...
e67341d8-350f-46f4-9154-4dbbb8035621 PRÉPARATION POUR CRÈME BRÛLÉE BIO 6L
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 Céréales instantanées en poudre saveur caramel...
0faad739-ea8c-4f03-b62e-51ee592a0546 FARINE DE BLÉ TYPE 45, 10KG

ingredients \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70 Eau, maltodextrine, sel, arômes, sucre, arôme ...
d183e914-db2f-4e2f-863a-a3b2d054c0b8 Farine de blé T65, eau, levure, vinaigre de ci...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 - 100% Semoule de BLE dur de qualité supérieure...
...
e67341d8-350f-46f4-9154-4dbbb8035621 Sucre roux de canne*° (64%), amidon de maïs*, ...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 Farine 87,1 % (Blé (GLUTEN), Blé hydrolysé (GL...
0faad739-ea8c-4f03-b62e-51ee592a0546 Farine de blé T45

path \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70 ../../ground_truth/a0492df6-9c76-4303-8813-65e...
d183e914-db2f-4e2f-863a-a3b2d054c0b8 ../../ground_truth/d183e914-db2f-4e2f-863a-a3b...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 ../../ground_truth/ab48a1ed-7a3d-4686-bb6d-ab4...

...
e67341d8-350f-46f4-9154-4dbbb8035621 ../../ground_truth/e67341d8-350f-46f4-9154-4db...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 ../../ground_truth/a8f6f672-20ac-4ff8-a8f2-3bc...
0faad739-ea8c-4f03-b62e-51ee592a0546 ../../ground_truth/0faad739-ea8c-4f03-b62e-51e...

content \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70 b'%PDF-1.5\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\n...
d183e914-db2f-4e2f-863a-a3b2d054c0b8 b'%PDF-1.5\r%\xe2\xe3\xcf\xd3\r\n4 0 obj\r<</L...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 b'%PDF-1.4%\xc7\xec\x8f\xa2\n5 0 obj\n<</Len...

...
e67341d8-350f-46f4-9154-4dbbb8035621 b'%PDF-1.7\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 b'%PDF-1.5\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\...
0faad739-ea8c-4f03-b62e-51ee592a0546 b'%PDF-1.5\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\...

text
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70 Concentré Liquide Asian CHEF® \n\nBouteille de...
d183e914-db2f-4e2f-863a-a3b2d054c0b8

ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 Direction Qualité \n\n \n\n \nPATES ALIMENTA...
...
e67341d8-350f-46f4-9154-4dbbb8035621 FICHE TECHNIQUE \n\nCREME BRÛLÉE 6L \n\nREF : ...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 81 rue de Sans Souci - CS13754 - 69576 Limones...
0faad739-ea8c-4f03-b62e-51ee592a0546 \n1050/10502066400 \n\n10502055300/1050202520...

[500 rows x 5 columns]
```

On fusionne les corpus

```
[5]: target_df = texts_df['ingredients'].rename('text').to_frame()
target_df.loc[:, 'source'] = 'target'
content_df = texts_df['text'].to_frame()
content_df['source'] = 'content'
corpus_df = pd.concat([target_df, content_df])
corpus_df = corpus_df.reset_index().set_index(['source', 'uid'])
corpus_df['text'].fillna('', inplace=True)
```

```
corpus_df
```

```
[5]: source    uid                                              text
      target   a0492df6-9c76-4303-8813-65ec5ccbfa70  Eau, maltodextrine, sel, arômes, sucre, arôme ...
      ...     d183e914-db2f-4e2f-863a-a3b2d054c0b8  Farine de blé T65, eau, levure, vinaigre de ci...
      ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 - 100% Semoule de BLE dur de qualité supérieur...
      ...
      content e67341d8-350f-46f4-9154-4dbbb8035621 FICHE TECHNIQUE \n\nCREME BRÛLÉE 6L \n\nREF : ...
      a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 81 rue de Sans Souci - CS13754 - 69576 Limones...
      0faad739-ea8c-4f03-b62e-51ee592a0546 \n1050/10502066400 \n\n10502055300/1050202520...
[1000 rows x 1 columns]
```

2 Analyses

2.1 Analyse des longueurs des textes

```
[6]: lengths = corpus_df['text'].apply(len)
```

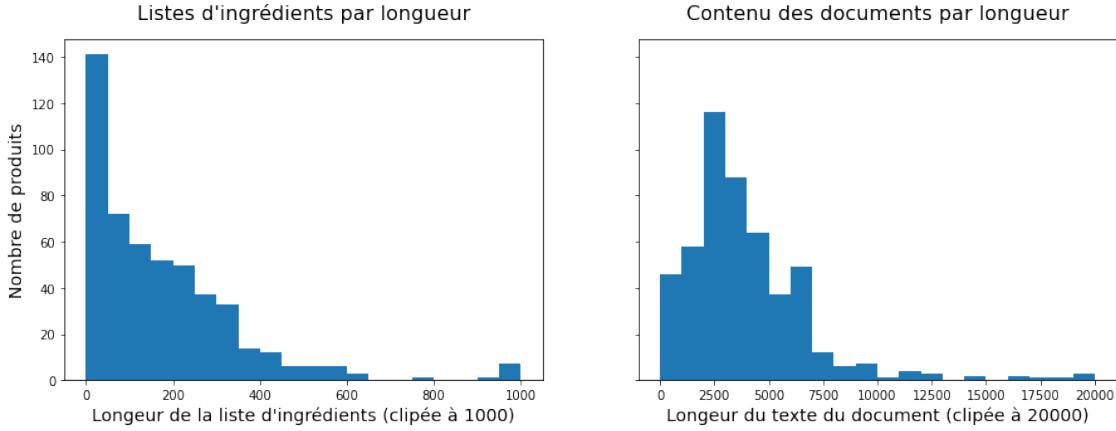
```
print()
lengths.reset_index()
    .groupby('source')
    .describe()
    .rename({'content': 'Texte des documents',
             'target': "Listes d'ingrédients"})
    .to_latex(#Path('..') / 'tbls' / 'text_lengths.tex',
              bold_rows=True,
              column_format='lccccccc',
              index_names=False,
            )
)
lengths.reset_index().groupby('source').describe()
```

```
\begin{tabular}{lccccccc}
\toprule
{} & \multicolumn{8}{l}{\text} \\
{} & count & mean & std & min & 25\% & 50\% & 75\% & max \\
\midrule
\textbf{Texte des documents} & 500.0 & 3937.630 & 3280.860732 & 0.0 & 2173.75 & 3247.5 & 5034.25 & 37322.0 \\
\textbf{Listes d'ingrédients} & 500.0 & 199.686 & 457.044723 & 0.0 & 43.00 & 122.0 & 250.75 & 7963.0 \\
\bottomrule
\end{tabular}
```

```
[6]:
```

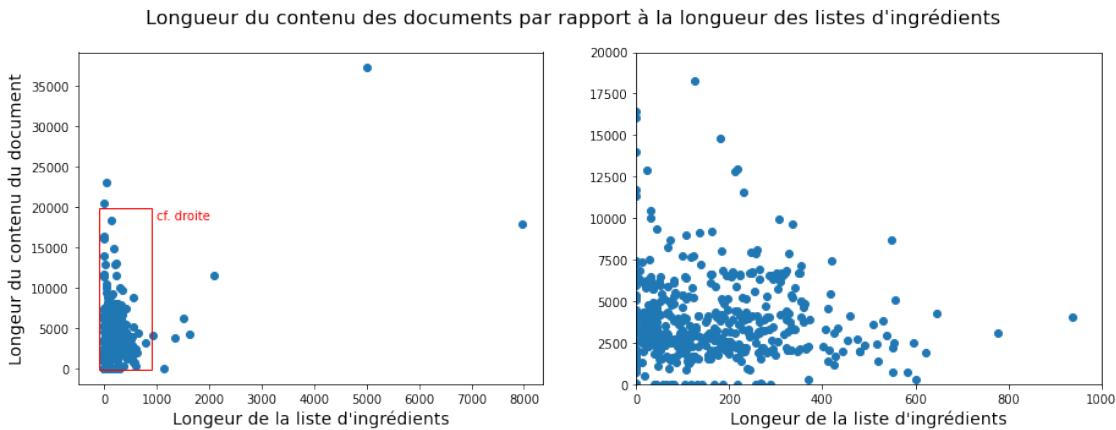
	text	count	mean	std	min	25%	50%	75%	max
source									
content	500.0	3937.630	3280.860732	0.0	2173.75	3247.5	5034.25	37322.0	
target	500.0	199.686	457.044723	0.0	43.00	122.0	250.75	7963.0	

```
[7]: fig, axs = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
axs[0].hist(lengths.loc['target'].clip(0, 1000), bins=20)
axs[0].set_title("Listes d'ingrédients par longueur", fontsize=16, pad=15)
axs[0].set_ylabel("Nombre de produits", fontsize=14)
axs[0].set_xlabel("Longeur de la liste d'ingrédients (clippée à 1000)", fontsize=14)
axs[1].hist(lengths.loc['content'].clip(0, 20000), bins=20)
axs[1].set_title("Contenu des documents par longueur", fontsize=16, pad=15)
axs[1].set_xlabel("Longeur du texte du document (clippée à 20000)", fontsize=14)
# fig.savefig(Path('..') / 'img' / 'text_lengths.png', bbox_inches='tight')
```



```
[8]: fig, axs = plt.subplots(ncols=2, figsize=(15, 5))

axs[0].scatter(x=lengths.loc['target'], y=lengths.loc['content'])
axs[0].set_xlabel("Longeur de la liste d'ingrédients", fontsize=14)
axs[0].set_ylabel("Longeur du contenu du document", fontsize=14)
axs[0].add_patch(mpatch.Rectangle((-100, -150), 1000, 20000, fill=False, color='red'))
axs[0].annotate('cf. droite', (1000, 18500), color='red')
axs[1].scatter(x=lengths.loc['target'], y=lengths.loc['content'])
axs[1].set_xlabel("Longeur de la liste d'ingrédients", fontsize=14)
axs[1].set_xlim(0, 1000)
axs[1].set_ylim(0, 20000)
fig.suptitle("Longueur du contenu des documents par rapport à la longueur des listes d'ingrédients", fontsize=16)
# fig.savefig(Path('..') / 'img' / 'text_lengths_2.png', bbox_inches='tight')
```



```
[9]: linregress(x=lengths.loc['target'], y=lengths.loc['content']).rvalue ** 2
```

```
[9]: 0.139326761283587
```

2.2 Analyse des mots

```
[76]: stop_words = {'de', 'et', 'la', 'en', 'du', 'les', 'des', 'le', 'dans', 'ce', 'un', 'pour',
               'par', 'sur', 'au', 'dont', 'of', 'and', 'pas', 'est', 'ou', 'que'}
print(stop_words)
```

```
{'par', 'pas', 'dans', 'des', 'les', 'pour', 'ce', 'est', 'de', 'of', 'un', 'and', 'la', 'sur', 'du', 'en',
 'au', 'ou', 'et', 'que', 'dont', 'le'}
```

```
[11]: tfidf_corpus = TfidfVectorizer(strip_accents='unicode',
                                    lowercase=True,
                                    stop_words=stop_words,
                                    ngram_range=(1, 1),
                                    max_df=1.0,
                                    #min_df=0.0, A TESTER !
                                    binary=False,
                                    norm=None, # set to l1 or l2
                                    use_idf=False,
                                    smooth_idf=False,
                                    sublinear_tf=False,
)
vectorized_corpus = tfidf_corpus.fit_transform(corpus_df['text'])
tokenizer = tfidf_corpus.build_analyzer()
```

Comptage des mots du vocabulaire du corpus :

```
[12]: inverse_corpus_voc = {val: key for key, val in tfidf_corpus.vocabulary_.items()}
word_counts = np.asarray(vectorized_corpus.sum(axis=0)).squeeze()
print(f'Corpus vocabulary size is ', len(tfidf_corpus.vocabulary_), '\n')
print('Most frequent words in vocabulary are:')
most_freq = dict()
for idx in word_counts.argsort()[:-1][:20]:
    most_freq[inverse_corpus_voc[idx].ljust(12)] = int(word_counts[idx])
    print(f'{inverse_corpus_voc[idx].ljust(12)}: {word_counts[idx]:5} occurrences')

str_cpt_corpus = f'{word.ljust(12)}: {most_freq[word]} occurrences \\\\' for word in most_freq.keys()]
```

Corpus vocabulary size is 15465

Most frequent words in vocabulary are:

produit	:	2198.0	occurrences
non	:	2164.0	occurrences
produits	:	1508.0	occurrences
10	:	1404.0	occurrences
kg	:	1290.0	occurrences
base	:	1119.0	occurrences
sel	:	1041.0	occurrences
poids	:	1021.0	occurrences
100	:	971.0	occurrences
palette	:	933.0	occurrences
ingredients	:	917.0	occurrences
date	:	904.0	occurrences
sucré	:	841.0	occurrences
12	:	803.0	occurrences
code	:	801.0	occurrences
lait	:	733.0	occurrences
absence	:	697.0	occurrences
fiche	:	688.0	occurrences
the	:	658.0	occurrences
france	:	657.0	occurrences

Constitution du vocabulaire des listes d'ingrédients

```
[13]: tfidf_ingred = clone(tfidf_corpus)
vectorized_ingred = tfidf_ingred.fit_transform(corpus_df.loc['target', 'text'])
```

```
[14]: inverse_ingred_voc = {val: key for key, val in tfidf_ingred.vocabulary_.items()}
word_counts = np.asarray(vectorized_ingred.sum(axis=0)).squeeze()
print(f'Ingredient vocabulary size is ', len(tfidf_ingred.vocabulary_), '\n')
print('Most frequent words in vocabulary are:')
most_freq = dict()
for idx in word_counts.argsort()[:-1][:20]:
    most_freq[inverse_ingred_voc[idx].ljust(12)] = int(word_counts[idx])
    print(f'{inverse_ingred_voc[idx].ljust(10)}: {word_counts[idx]:5} occurrences')

str_cpt_corpus_2 = f'{word.ljust(12)}: {most_freq[word]} occurrences & for word in most_freq.keys()
str_latex = '\n'.join([str2 + ' ' + str1 for str1, str2 in zip(str_cpt_corpus, str_cpt_corpus_2)])
#print(str_latex)
```

Ingredient vocabulary size is 1324

Most frequent words in vocabulary are:

sucré	:	363.0	occurrences
acide	:	255.0	occurrences
sel	:	240.0	occurrences
sirrop	:	180.0	occurrences
eau	:	178.0	occurrences

```

poudre      : 176.0 occurrences
arome       : 175.0 occurrences
lait        : 171.0 occurrences
ble         : 171.0 occurrences
huile       : 146.0 occurrences
critrique   : 132.0 occurrences
farine      : 128.0 occurrences
amidon      : 125.0 occurrences
glucose     : 124.0 occurrences
cacao       : 122.0 occurrences
extrait     : 117.0 occurrences
acidifiant  : 114.0 occurrences
aromes      : 98.0 occurrences
soja        : 96.0 occurrences
concentre   : 92.0 occurrences

```

On vérifie s'il existe des mots qui sont inclus dans les listes d'ingrédients mais pas dans les documents :

```
[15]: ingred_only = {word for word in tfidf_ingred.vocabulary_ if word not in tfidf_corpus.vocabulary_}
ingred_only
```

```
[15]: set()
```

Constitution d'une nouvelle feature qui sont les textes tokenisés via la méthode du CountVectorizer.

```
[16]: corpus_df['tokenized'] = corpus_df['text'].apply(tokenizer)
corpus_df
```

```

[16]:
source  uid
target  a0492df6-9c76-4303-8813-65ec5ccbf70 Eau, maltodextrine, sel, arômes, sucre, arôme ...
d183e914-db2f-4e2f-863a-a3b2d054c0b8 Farine de blé T65, eau, levure, vinaigre de ci...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 - 100% Semoule de BLE dur de qualité supérieur...
...
content e67341d8-350f-46f4-9154-4dbbb8035621 FICHE TECHNIQUE \n\nCREME BRûLÉE 6L \n\nREF : ...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 81 rue de Sans Souci - CS13754 - 69576 Limones...
0faad739-ea8c-4f03-b62e-51ee592a0546 \n1050/10502066400 \n\n10502055300/1050202520...
                                         text \
source  uid
target  a0492df6-9c76-4303-8813-65ec5ccbf70 [eau, maltodextrine, sel, aromes, sucre, arome...
d183e914-db2f-4e2f-863a-a3b2d054c0b8 [farine, ble, t65, eau, levure, vinaigre, cidr...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 [100, semoule, ble, dur, qualite, superieure, ...
...
content e67341d8-350f-46f4-9154-4dbbb8035621 [fiche, technique, creme, brulee, 6l, ref, nap...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 [81, rue, sans, souci, cs13754, 69576, limones...
0faad739-ea8c-4f03-b62e-51ee592a0546 [1050, 10502066400, 10502055300, 10502025200, ...
                                         tokenized
source  uid
target  a0492df6-9c76-4303-8813-65ec5ccbf70 [eau, maltodextrine, sel, aromes, sucre, arome...
d183e914-db2f-4e2f-863a-a3b2d054c0b8 [farine, ble, t65, eau, levure, vinaigre, cidr...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8 [100, semoule, ble, dur, qualite, superieure, ...
...
content e67341d8-350f-46f4-9154-4dbbb8035621 [fiche, technique, creme, brulee, 6l, ref, nap...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 [81, rue, sans, souci, cs13754, 69576, limones...
0faad739-ea8c-4f03-b62e-51ee592a0546 [1050, 10502066400, 10502055300, 10502025200, ...
                                         ...
[1000 rows x 2 columns]
```

3 Analyse de données

4 Représentation des mots

4.1 Document frequency

1- Déjà, on calcule une métrique qui permet de dire si un mot est plutôt un mot de type ingrédients ou un mot issu du corpus.

Représentation simple : on regarde simplement la document frequency de chacun des mots dans le corpus des ingrédients.

```
[17]: doc_freq_counter = clone(tfidf_ingred)
doc_freq_counter.set_params(binary=True).fit(corpus_df.loc['target', 'text'])
binary_ingred_counts = doc_freq_counter.transform(corpus_df.loc[:, 'text'])
```

```
[18]: doc_freq_ingred = binary_ingred_counts[:500]
doc_freq_ingred = np.array(doc_freq_ingred.sum(axis=0)).reshape(-1) / 500
print('shape of doc_freq_ingred :', doc_freq_ingred.shape)
doc_freq_ingred_translator = defaultdict(lambda: 0., {word: doc_freq_ingred[tfidf_ingred.vocabulary_[word]]
                                                       for word in tfidf_ingred.vocabulary_.keys()})

print(doc_freq_ingred_translator['sucré'])
print(doc_freq_ingred_translator['tabouillleeeeeeeiiiiiiii'])
print(doc_freq_ingred_translator['maltodextrine'])
```

```

shape of doc_freq_ingred : (1324,)
0.448
0.0
0.064

```

On constitue un array qui nous servira plus tard, avec le document frequency dans les ingrédients de tous les mots utilisés dans le contenu des documents.

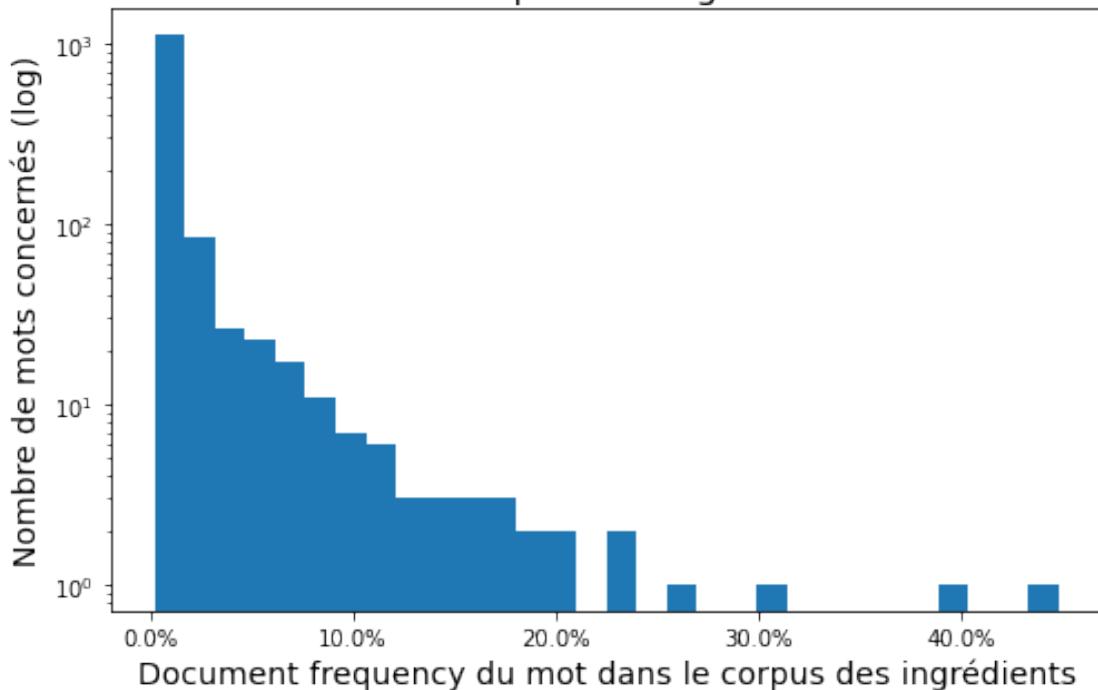
```
[19]: corpus_doc_freq_ingred = np.zeros(shape=(len(tfidf_corpus.vocabulary_)))
for i in range(len(tfidf_corpus.vocabulary_)):
    corpus_doc_freq_ingred[i] = doc_freq_ingred_translator[inverse_corpus_voc[i]]
corpus_doc_freq_ingred
```

```
[19]: array([0.004, 0.   , 0.   , ..., 0.   , 0.   , 0.   ])
```

On peut sortir les top scorers de la doc frequency :

```
[28]: fig, ax = plt.subplots(figsize=(8, 5))
ax.hist(doc_freq_ingred, bins=30, log=True)
ax.xaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
ax.set_xlabel("Document frequency du mot dans le corpus des ingrédients", fontsize=14)
ax.set_ylabel("Nombre de mots concernés (log)", fontsize=14)
fig.suptitle("Répartition des mots par documents frequency dans le corpus des ingrédients", fontsize=16)
# fig.savefig(Path('..') / 'img' / 'doc_freq_of_ingreds.png', bbox_inches='tight')
```

Répartition des mots par documents frequency
dans le corpus des ingrédients



```
[22]: for word_idx in doc_freq_ingred.argsort()[:-1][:-20]:
    print(inverse_ingred_voc[word_idx], '&', f'{doc_freq_ingred[word_idx]*100:.2f}%', '\\\\')
for word_idx in doc_freq_ingred.argsort()[:20]:
    print(inverse_ingred_voc[word_idx], '&', f'{doc_freq_ingred[word_idx]*100:.2f}%', '\\\\')
```

```

sucre & 44.80% \\
sel & 40.00% \\
eau & 30.20% \\
acide & 26.20% \\
arome & 23.00% \\
huile & 23.00% \\

```

```

ble & 20.60% \\
amidon & 20.00% \\
aromes & 19.00% \\
sirop & 18.40% \\
citrique & 18.00% \\
lait & 17.20% \\
acidifiant & 17.00% \\
farine & 16.40% \\
extrait & 16.20% \\
glucose & 16.00% \\
concentre & 14.80% \\
jus & 14.00% \\
poudre & 14.00% \\
colza & 13.20% \\
ferrique & 0.20% \\
d3 & 0.20% \\
datte & 0.20% \\
dattes & 0.20% \\
debris & 0.20% \\
decafeine & 0.20% \\
deciree & 0.20% \\
deconseille & 0.20% \\
nectar & 0.20% \\
necessaire & 0.20% \\
ne & 0.20% \\
delta & 0.20% \\
naturelles & 0.20% \\
denoyautes & 0.20% \\
dentier & 0.20% \\
depellicule & 0.20% \\
cysteine & 0.20% \\
negra & 0.20% \\
new & 0.20% \\
nicotinamide & 0.20% \\

```

```
[309]: # looking for first word in documents but not in ingredients
for word_idx in np.array(vectorized_corpus[500:]).sum(axis=0).reshape(15465).argsort()[:-1]:
    print(inverse_corpus_voc[word_idx])
    if inverse_corpus_voc[word_idx] not in tfidf_ingred.vocabulary_:
        print('trouvé !')
        break
```

```

produit
non
produits
10
kg
base
poids
trouvé !

```

```
[75]: ingred_scores.shape
```

```
[75]: (1324,)
```

```
[291]: ingred_scores = np.log2(doc_freq_ingred * 500 + 1)
sample = doc_freq_ingred.argsort()[:-1][[0, 10, 100, 250, 702]]
sample_scores = ingred_scores[sample]
labels = [inverse_ingred_voc[word_idx] + f'\n{ingred_scores[word_idx]:.2f}' for word_idx in sample] + ['poids\n0.00']
labels
```

```
[291]: ['sucré\n7.81',
'citrique\n6.51',
'issus\n4.39',
'si\n2.81',
'plantations\n1.58',
'poids\n0.00']
```

```
[292]: gradient = np.linspace(0, 8, 256)
gradient = np.vstack((gradient, gradient, gradient))

fig, ax = plt.subplots(figsize=(8, 3))

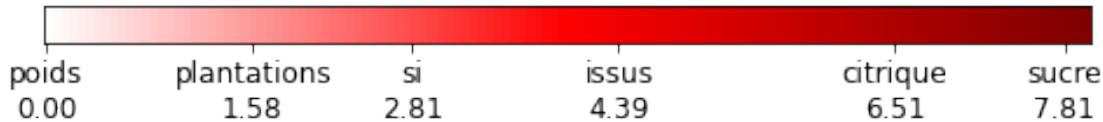
ax.imshow(gradient, aspect=3, cmap='seismic', vmin=-8, vmax=8)
ax.set_xticks(np.hstack((255 / 8 * sample_scores, np.array([0]))))
```

```

ax.set_xticklabels(labels, fontsize = 12)
#ax.set_xlim(0, 50)
ax.set_yticks([])
# fig.savefig(Path('..') / 'img' / 'scores_bar.png', bbox_inches='tight')

```

[292]: []



On calcule les document frequencies des mots issus des documents, et on compte sur les corpus.

```

[293]: doc_freq_counter_2 = clone(tfidf_corpus)
doc_freq_counter_2.set_params(binary=False).fit(corpus_df.loc[:, 'text'])
word_counts = np.array(doc_freq_counter_2.transform(corpus_df.loc[:, 'text']).todense())
delta_doc_counts = (word_counts[500:] - word_counts[:500])
binary_delta = np.where(delta_doc_counts > 0., np.ones(delta_doc_counts.shape), np.zeros(delta_doc_counts.shape))
docs_scores = np.log2(binary_delta.sum(axis=0) + 1)

```

[294]: docs_scores[tfidf_corpus.vocabulary_['produits']]

[294]: 8.005624549193879

```

[295]: sample = docs_scores.argsort()[:-1][[0, 90, 251, 550, 1200, 2200, 3803, 10005]]
sample_scores = docs_scores[sample]
labels = [inverse_corpus_voc[word_idx] + f'\n{docs_scores[word_idx]:.2f}' for word_idx in sample]
labels

```

```

[295]: ['produit\n8.72',
'version\n7.52',
'coque\n6.55',
'peuvent\n5.49',
'redigee\n4.32',
'update\n3.32',
'ok\n2.32',
'344kcal\n1.00']

```

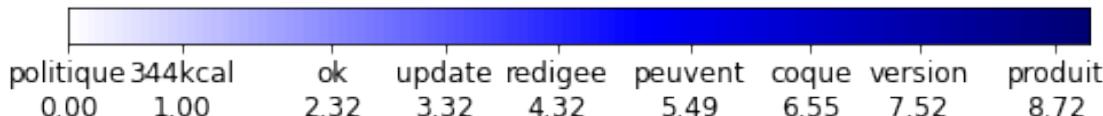
```

[298]: gradient = np.linspace(0, 8, 256)
gradient = np.vstack((gradient, gradient, gradient))

fig, ax = plt.subplots(figsize=(8, 3))

ax.imshow(gradient, aspect=3, cmap='seismic_r', vmin=-9, vmax=9)
ax.set_xticks(np.hstack((255 / 9 * sample_scores, np.array([0]))))
ax.set_xticklabels(labels + ["politique\n0.00"], fontsize = 12)
#ax.set_xlim(0, 50)
ax.set_yticks([])
# fig.savefig(Path('..') / 'img' / 'corpus_score_bar.png', bbox_inches='tight')

```



Et maintenant on calcule le score relatif de chacun des mots.

```

[299]: # new computation on ingredients, on corpus vocabulary indexing
binary_ingred = np.where(word_counts[:500] > 0., np.ones(word_counts[:500].shape), np.zeros(word_counts[:500].shape))
ingred_scores = np.log2(binary_ingred.sum(axis=0) + 1)
# check computation is coherent
print(ingred_scores[tfidf_corpus.vocabulary_['sucre']])

```

```
7.813781191217037
```

```
[300]: sample = np.hstack([relative_scores.argsort()[[0, 90, 251, 1200, 3000, 8007, 15464]], [14079]])
sample_scores = relative_scores[sample]
labels = [inverse_corpus_voc[word_idx] + f'\n{relative_scores[word_idx]:.2f}' for word_idx in sample]
labels
```

```
[300]: ['glucides\n-8.67',
 '800\n-7.04',
 'tout\n-5.93',
 'presence\n-4.09',
 'faraud\n-2.58',
 'proteges\n-1.00',
 'acidifiants\n3.91',
 'sucre\n1.51']
```

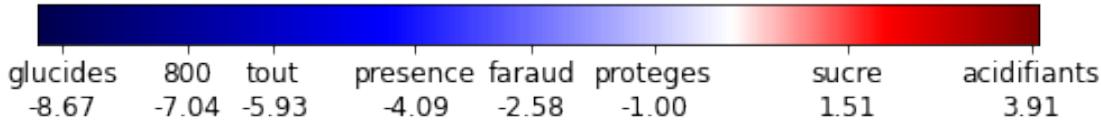
```
[301]: mmin, mmax, resolution = -9, 4, 1000
divnorm = colors.TwoSlopeNorm(vmin=mmin, vcenter=0, vmax=mmax)
gradient = np.linspace(mmin, mmax, resolution)
gradient = np.vstack([gradient, gradient])
# gradient = np.vstack(sorted(relative_scores), sorted(relative_scores))

def convert_to_width(num):
    return(int(resolution * (num - mmin) / (mmax - mmin)))

fig, ax = plt.subplots(figsize=(8, 3))

ax.imshow(gradient, aspect=20, cmap='seismic', norm=divnorm ) #, vmin=-9, vmax=9)
ax.set_xticks(list(map(convert_to_width, sample_scores)))
# ax.set_xticks(255 / 18 * sample_scores)
ax.set_xticklabels(labels, fontsize = 12)
#ax.set_xlim(0, 50)
ax.set_yticks([])
# fig.savefig(Path('..') / 'img' / 'relative_score_bar.png', bbox_inches='tight')
```

```
[301]: []
```



4.2 Word embeddings

4.2.1 Word2Vec

On calcule les embeddings avec Word2Vec

```
[302]: word2vec_model = word2vec.Word2Vec(corpus_df['tokenized'], min_count=1)
```

```
[303]: for i, word in enumerate(tfidf_corpus.vocabulary_.keys()):
    print('-----\n', word, ':', word2vec_model.wv[word][:5])
    if i > 4:
        break
```

```
-----
eau : [-1.0830716 -0.46610487 -0.43349764  0.55493397  1.1007366 ]
-----
maltodextrine : [ 0.01011625  0.15969634 -0.67031556  0.5559474   0.4602319 ]
-----
sel : [-0.8459699 -0.21097712 -0.6416249   0.84254736  0.9729415 ]
-----
aromes : [ 0.22419034  0.23973037 -1.4658073   1.1386586   1.0591131 ]
-----
sucre : [-0.74801284  0.1217097  -2.0248985   1.5143404   1.4470899 ]
-----
arome : [ 0.08999626  0.10789929 -2.2472398   1.5559452   1.5740927 ]
```

```
[265]: ndarray_words_embeddings = np.vstack([word2vec_model.wv[inverse_corpus_voc[i]]  
                                         for i in range(len(tfidf_corpus.vocabulary_))])  
ndarray_words_embeddings
```

```
[265]: array([[-1.9143574e+00, -1.0786384e+00, -1.6952591e+00, ...,  
           2.1197143e-04, 6.8014598e-01, -9.4786072e-01],  
           [-1.0596310e+00, -5.1570371e-02, -2.4971814e+00, ...,  
           -4.9154687e-01, 1.3782319e+00, -1.2871372e+00],  
           [-2.1143033e-01, -1.3124047e-01, -2.7956560e-01, ...,  
           -6.6144560e-03, 1.6432673e-02, -5.8027157e-03],  
           ...,  
           [-4.8991539e-02, -1.6540296e-02, -6.0264323e-02, ...,  
           3.1673540e-03, -9.9934675e-03, -1.2510464e-02],  
           [-8.2935727e-01, -6.4514387e-01, -8.7146807e-01, ...,  
           -1.7320616e-02, 3.3239821e-01, -3.0926281e-01],  
           [-2.5584301e-01, -1.5305966e-01, -5.1036954e-01, ...,  
           -1.5918270e-02, -3.0415934e-02, -2.4762219e-01]], dtype=float32)
```

On effectue une PCA :

```
[266]: scaled_words_embeddings = StandardScaler().fit_transform(ndarray_words_embeddings)  
scaled_words_embeddings
```

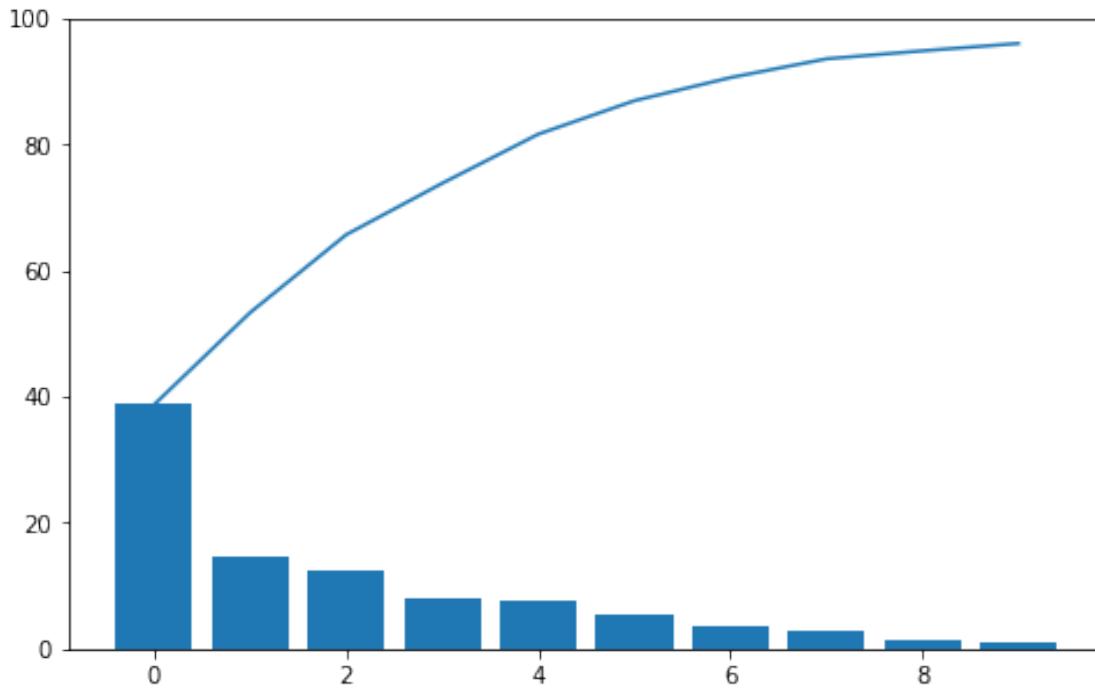
```
[266]: array([[-10.750973 , -7.621986 , -6.1712112 , ..., 0.1407848 ,  
           4.966931 , -4.985649 ],  
           [-5.818584 , -0.29638907, -9.341663 , ..., -5.478642 ,  
           9.752907 , -6.8654494 ],  
           [-0.92385375, -0.8646387 , -0.5741748 , ..., 0.06277785,  
           0.4166085 , 0.23393212],  
           ...,  
           [ 0.01353529, -0.04653592, 0.29284707, ..., 0.17455655,  
           0.23543474, 0.19676708],  
           [-4.489738 , -4.5300717 , -2.9142997 , ..., -0.05956358,  
           2.5828223 , -1.4474237 ],  
           [-1.180147 , -1.0202649 , -1.4866732 , ..., -0.04353869,  
           0.09542128, -1.1058966 ]], dtype=float32)
```

```
[267]: PCA_model = PCA(n_components=10)  
PCA_words_embeddings = PCA_model.fit_transform(scaled_words_embeddings)  
print(PCA_words_embeddings.shape)
```

(15465, 10)

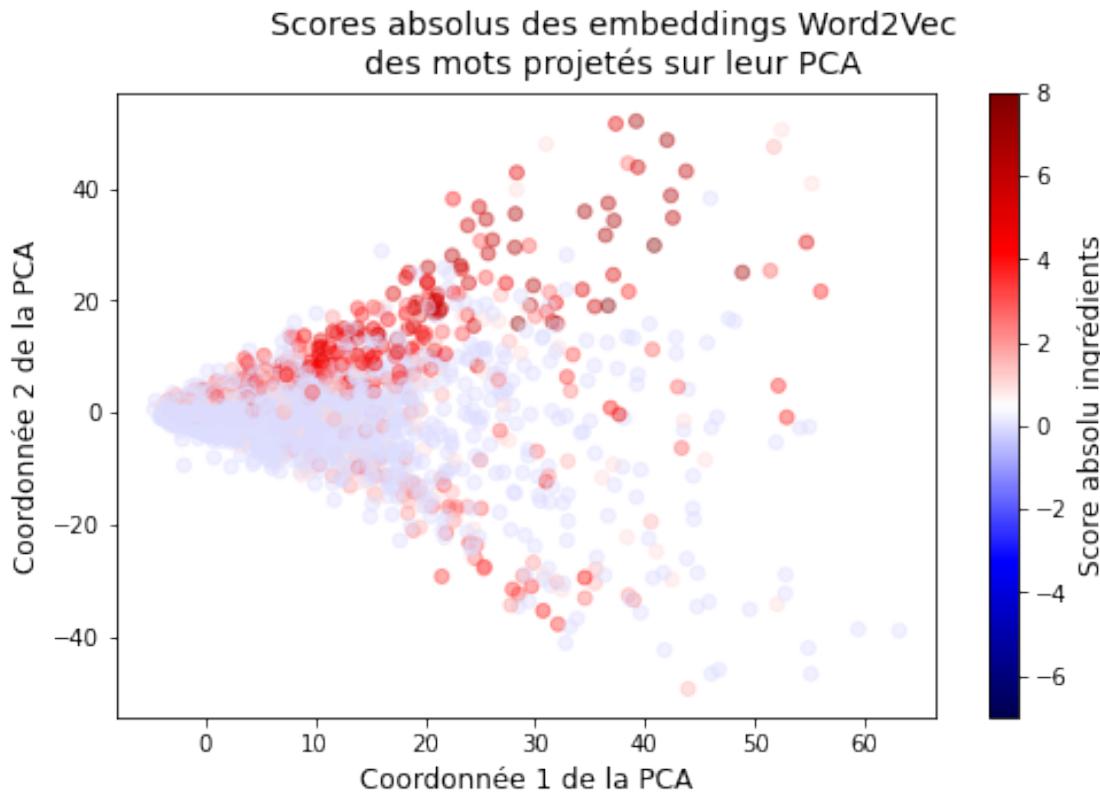
```
[268]: fig, ax = plt.subplots(figsize=(8, 5))  
ax.plot(PCA_model.explained_variance_ratio_.cumsum()*100)  
ax.bar(x=range(10), height=PCA_model.explained_variance_ratio_*100)  
ax.set_xlim(0, 100)
```

```
[268]: (0.0, 100.0)
```



On la dessine :

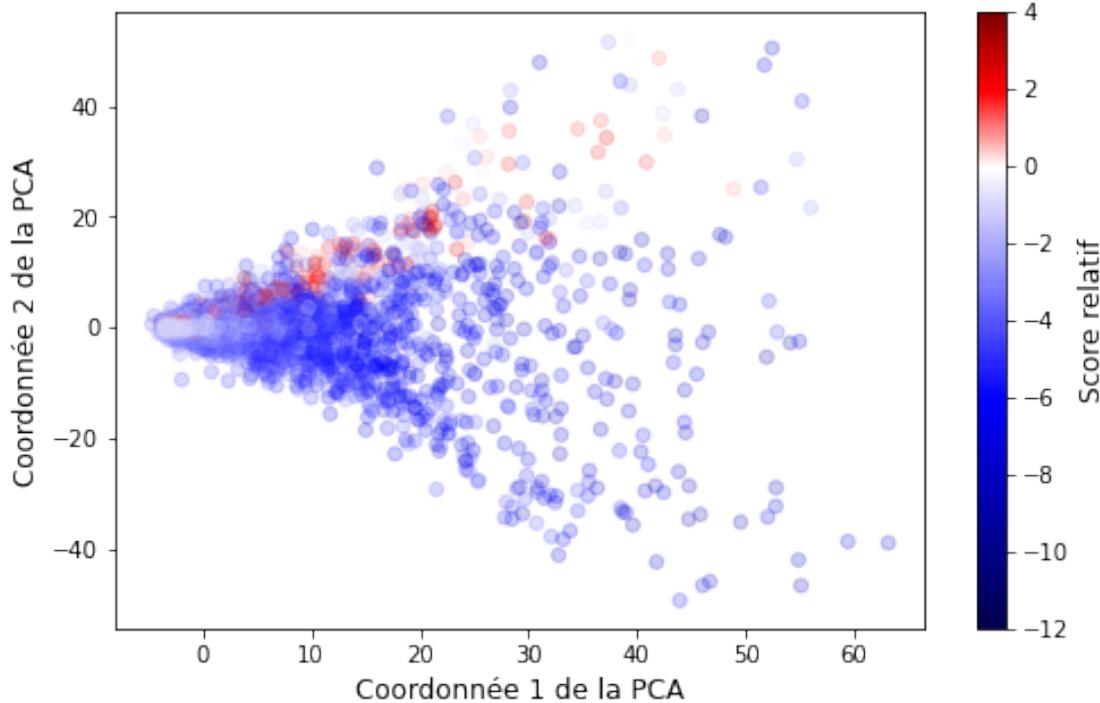
```
[304]: fig, ax = plt.subplots(figsize=(8, 5))
scat = ax.scatter(x=PCA_words_embeddings[:, 0],
                   y=PCA_words_embeddings[:, 1],
                   alpha=0.4,
                   cmap='seismic',
                   vmin=-7,
                   vmax=8,
                   c=np.log2(500 * corpus_doc_freq_ingred + 1),
                   )
ScalMappable = cm.ScalarMappable(norm=Normalize(vmin=-7, vmax=8, clip=True), cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=ax)
fig.axes[1].set_ylabel('Score absolu ingrédients', fontsize=12)
fig.axes[0].set_xlabel('Coordonnée 1 de la PCA', fontsize=12)
fig.axes[0].set_ylabel('Coordonnée 2 de la PCA', fontsize=12)
fig.suptitle('Scores absolus des embeddings Word2Vec\ndes mots projetés sur leur PCA', fontsize=14)
fig.savefig(Path('..') / 'img' / 'word2vec_PCA.png', bbox_inches='tight')
```



```
[305]: mmin, mmax, resolution = -12, 4, 1000
divnorm = colors.TwoSlopeNorm(vmin=mmin, vcenter=0, vmax=mmax)

fig, ax = plt.subplots(figsize=(8, 5))
scat = ax.scatter(x=PCA_words_embeddings[:, 0],
                   y=PCA_words_embeddings[:, 1],
                   alpha=0.2,
                   cmap='seismic',
                   norm=divnorm,
                   c=relative_scores,
                   )
ScalMappable = cm.ScalarMappable(norm=divnorm, cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=ax)
fig.axes[1].set_ylabel('Score relatif', fontsize=12)
fig.axes[0].set_xlabel('Coordonnée 1 de la PCA', fontsize=12)
fig.axes[0].set_ylabel('Coordonnée 2 de la PCA', fontsize=12)
fig.suptitle('Scores relatifs des Word2Vec des mots projetés sur leur PCA', fontsize=14)
fig.savefig(Path('..') / 'img' / 'word2vec_PCA_relative.png', bbox_inches='tight')
```

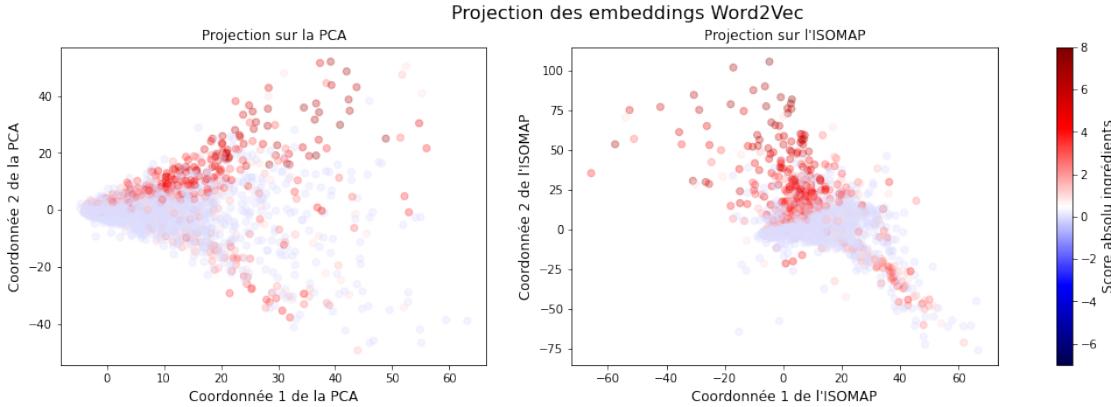
Scores relatifs des Word2Vec des mots projetés sur leur PCA



```
[275]: isomap_model = Isomap(n_neighbors=5, n_components=2, n_jobs=8)
isomap_words_embeddings = isomap_model.fit_transform(scaled_words_embeddings)
print(isomap_words_embeddings.shape)
```

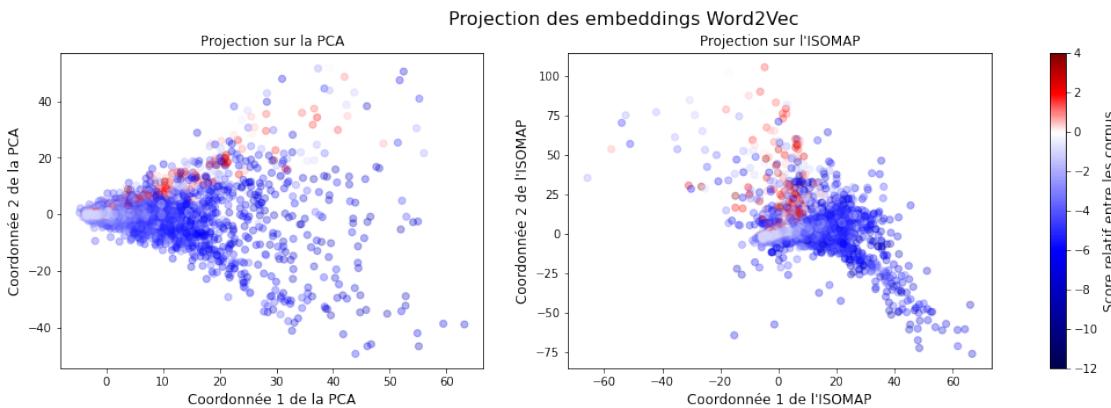
```
(15465, 2)
```

```
[309]: fig, axs = plt.subplots(ncols=2, figsize=(18, 5))
scat = axs[0].scatter(x=PCA_words_embeddings[:, 0],
                      y=PCA_words_embeddings[:, 1],
                      alpha=0.3,
                      cmap='seismic',
                      vmin=-7,
                      vmax=8,
                      c=np.log2(500 * corpus_doc_freq_ingred + 1),
                      )
scat = axs[1].scatter(x=isomap_words_embeddings[:, 0],
                      y=isomap_words_embeddings[:, 1],
                      alpha=0.3,
                      cmap='seismic',
                      vmin=-7,
                      vmax=8,
                      c=np.log2(500 * corpus_doc_freq_ingred + 1),
                      )
ScalMappable = cm.ScalarMappable(norm=Normalize(vmin=-7, vmax=8, clip=True), cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=axs)
fig.axes[2].set_ylabel("Score absolu ingrédients", fontsize=12)
fig.axes[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
fig.axes[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
fig.axes[0].set_title("Projection sur la PCA", fontsize=12)
fig.axes[1].set_xlabel("Coordonnée 1 de l'ISOMAP", fontsize=12)
fig.axes[1].set_ylabel("Coordonnée 2 de l'ISOMAP", fontsize=12)
fig.axes[1].set_title("Projection sur l'ISOMAP", fontsize=12)
fig.suptitle("Projection des embeddings Word2Vec", fontsize=16)
# fig.savefig(Path('..') / 'img' / 'word2vec_projection.png', bbox_inches='tight')
```



```
[310]: mmin, mmax, resolution = -12, 4, 1000
divnorm = colors.TwoSlopeNorm(vmin=mmin, vcenter=0, vmax=mmax)

fig, axes = plt.subplots(ncols=2, figsize=(18, 5))
scat = axes[0].scatter(x=PCA_words_embeddings[:, 0],
                       y=PCA_words_embeddings[:, 1],
                       alpha=0.3,
                       cmap='seismic',
                       norm=divnorm,
                       c=relative_scores,
)
scat = axes[1].scatter(x=isomap_words_embeddings[:, 0],
                       y=isomap_words_embeddings[:, 1],
                       alpha=0.3,
                       cmap='seismic',
                       norm=divnorm,
                       c=relative_scores,
)
ScalMappable = cm.ScalarMappable(norm=divnorm, cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=axes)
fig.axes[2].set_ylabel("Score relatif entre les corpus", fontsize=12)
fig.axes[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
fig.axes[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
fig.axes[0].set_title("Projection sur la PCA", fontsize=12)
fig.axes[1].set_xlabel("Coordonnée 1 de l'ISOMAP", fontsize=12)
fig.axes[1].set_ylabel("Coordonnée 2 de l'ISOMAP", fontsize=12)
fig.axes[1].set_title("Projection sur l'ISOMAP", fontsize=12)
fig.suptitle("Projection des embeddings Word2Vec", fontsize=16)
# fig.savefig(Path('..') / 'img' / 'word2vec_projection_relative.png', bbox_inches='tight')
```



4.3 Représentaions des textes

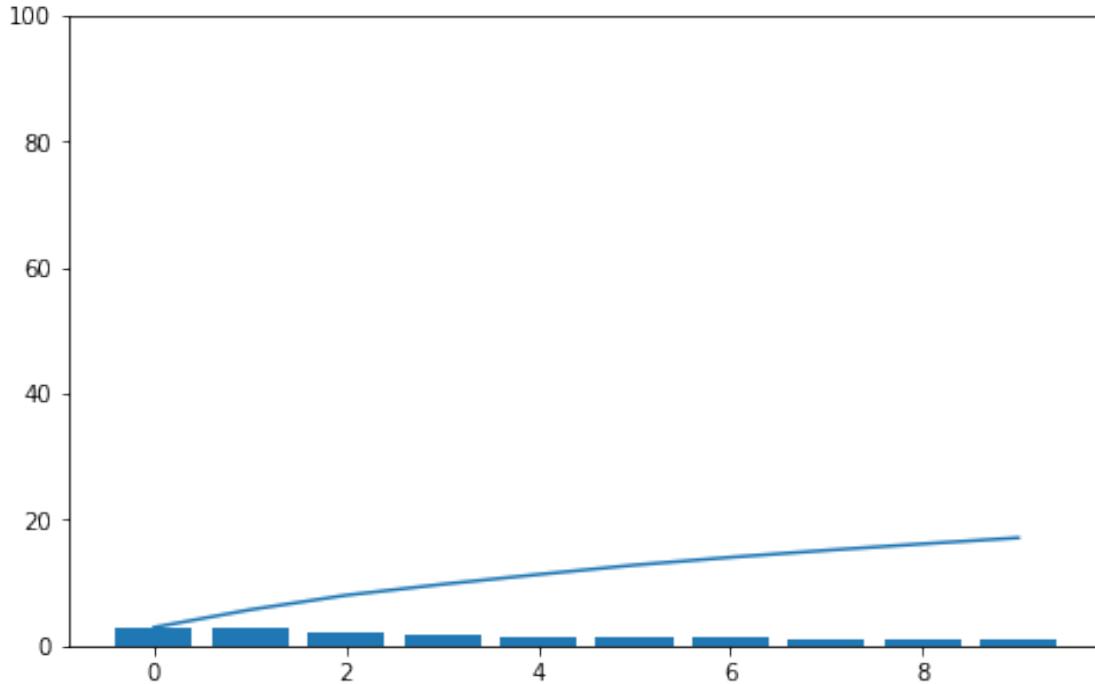
4.3.1 Bag of Words : comptes

Je fais une PCA sur les vecteurs de comptes. Mais d'abord, il faut standardiser.

```
[320]: vectorized_std = StandardScaler().fit_transform(np.array(vectorized_corpus.todense()))
PCA_model = PCA(n_components=10)
decomp = PCA_model.fit_transform(vectorized_std)
```

```
[321]: fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(PCA_model.explained_variance_ratio_.cumsum()*100)
ax.bar(x=range(10), height=PCA_model.explained_variance_ratio_*100)
ax.set_ylim(0, 100)
```

```
[321]: (0.0, 100.0)
```



```
[322]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []

ax[0].scatter(x=decomp[:500, 0],
               y=decomp[:500, 1],
               c=np.log2(np.array(vectorized_corpus[:500,:].sum(axis=1)).flatten() + 1),
               cmap='viridis',
               alpha=0.5)

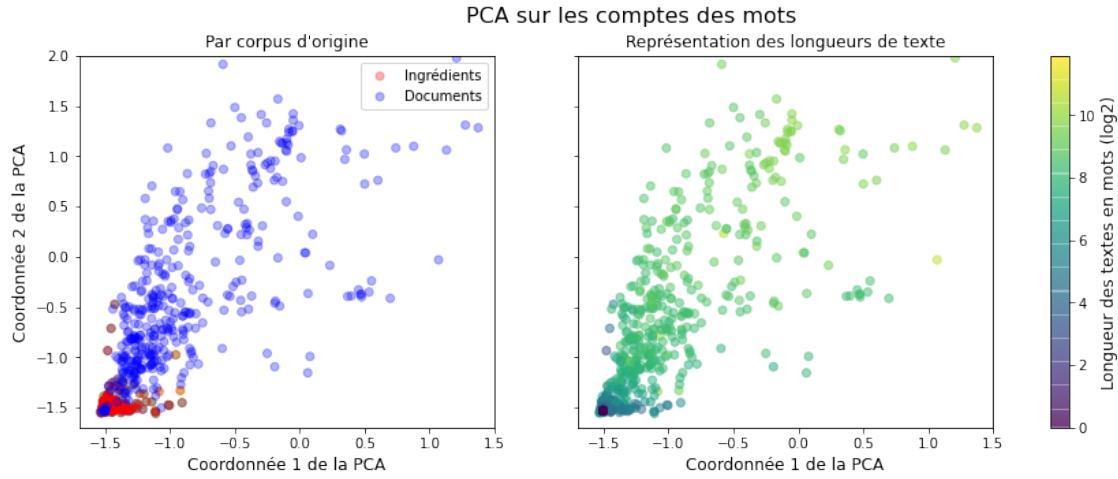
handles.append(ax[0].scatter(x=decomp[:500, 0],
                             y=decomp[:500, 1],
                             c=['red']*500,
                             alpha=0.3))
handles.append(ax[0].scatter(x=decomp[500:, 0],
                             y=decomp[500:, 1],
                             c=['blue']*500,
                             alpha=0.3))

handles.append(ax[1].scatter(x=decomp[:, 0],
                             y=decomp[:, 1],
                             c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                             cmap='viridis',
                             alpha=0.5))
```

```

ax[0].set_xlim(-1.7, 1.5)
ax[0].set_ylim(-1.7, 2)
ax[1].set_xlim(-1.7, 1.5)
ax[1].set_ylim(-1.7, 2)
ax[0].legend(handles[:-1], ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("PCA sur les comptes des mots", fontsize=16)
ax[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
# fig.savefig(Path('..') / 'img' / 'PCA_counts.png', bbox_inches='tight')

```



4.3.2 tSVD

Je commence par faire un tSVD sur le corpus, en différenciant les ingrédients du texte des fiches techniques. Dans ce premier cas, on a les comptes des mots dans la vectorisation.

[323]: `vectorized_corpus.todense()`

```

[323]: matrix([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ...,  0.,  0.,  0.],
   ...,
   [ 1.,  0.,  0., ...,  0.,  0.,  0.],
   [ 2.,  1.,  0., ...,  0.,  0.,  0.],
   [11.,  0.,  0., ...,  0.,  0.,  0.]])

```

[324]: `transformer = TruncatedSVD(n_components=100)`
`projected = transformer.fit_transform(vectorized_corpus)`

```

[325]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []
handles.append(ax[0].scatter(x=projected[:500, 0], y=projected[:500, 1], c=['red']*500, alpha=0.5))
handles.append(ax[0].scatter(x=projected[500:, 0], y=projected[500:, 1], c=['blue']*500, alpha=0.5))
handles.append(ax[1].scatter(x=projected[:, 0],
                            y=projected[:, 1],
                            c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                            cmap='viridis',
                            alpha=0.5))

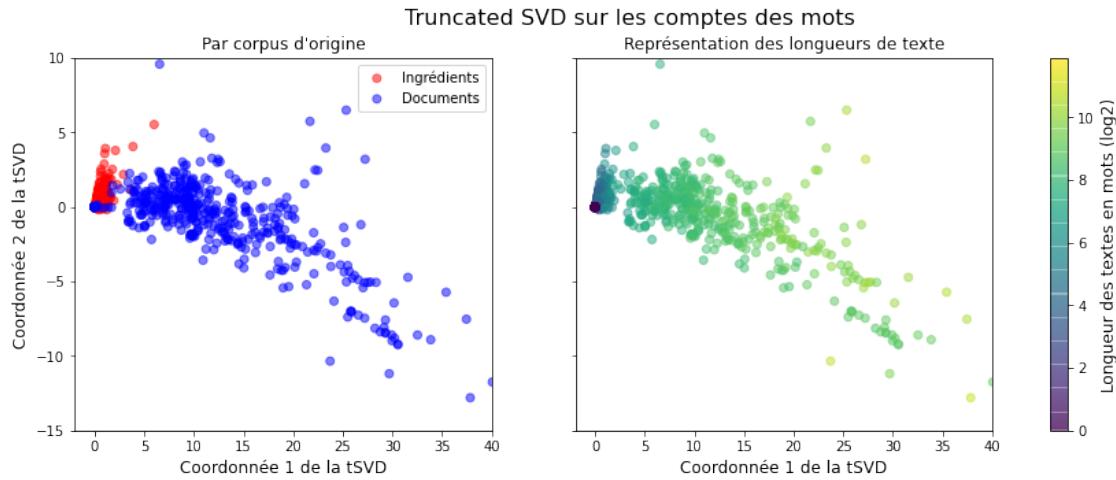
ax[0].set_xlim(-15, 10)
ax[0].set_ylim(-2, 40)
ax[1].set_xlim(-15, 10)
ax[1].set_ylim(-2, 40)
ax[0].legend(handles[:-1], ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("Truncated SVD sur les comptes des mots", fontsize=16)

```

```

ax[0].set_ylabel("Coordonnée 2 de la tSVD", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
# fig.savefig(Path('..') / 'img' / 'tSVD_counts.png', bbox_inches='tight')

```



4.3.3 PCA : fréquences

En utilisant cette fois la term frequency. PCA

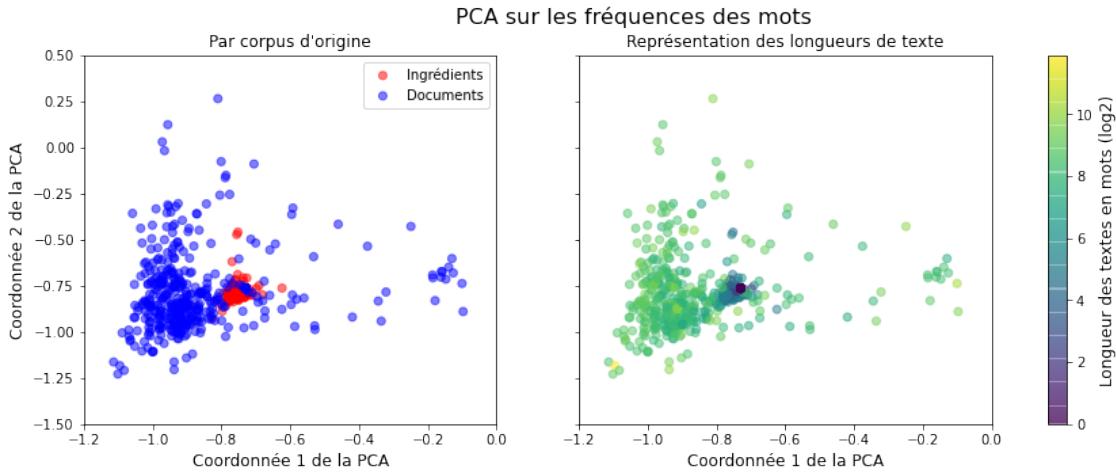
```
[326]: tf_corpus_model = clone(tfidf_corpus)
tf_corpus = tf_corpus_model.set_params(norm='l1').fit_transform(corpus_df.loc[:, 'text'])
```

```
[327]: vectorized_std = StandardScaler().fit_transform(np.array(tf_corpus.todense()))
decomp = PCA(n_components=10).fit_transform(vectorized_std)
```

```

[332]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []
handles.append(ax[0].scatter(x=decomp[:500, 0], y=decomp[:500, 1], c=['red']*500, alpha=0.5))
handles.append(ax[0].scatter(x=decomp[500:, 0], y=decomp[500:, 1], c=['blue']*500, alpha=0.5))
handles.append(ax[1].scatter(x=decomp[:, 0],
                             y=decomp[:, 1],
                             c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                             alpha=0.5,
                             cmap='viridis'))
ax[0].set_xlim(-1.2, 0)
ax[0].set_ylim(-1.5, 0.5)
ax[1].set_xlim(-1.2, 0)
ax[1].set_ylim(-1.5, 0.5)
ax[0].legend(handles[:-1], ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("PCA sur les fréquences des mots", fontsize=16)
ax[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
fig.savefig(Path('..') / 'img' / 'PCA_freq.png', bbox_inches='tight')

```

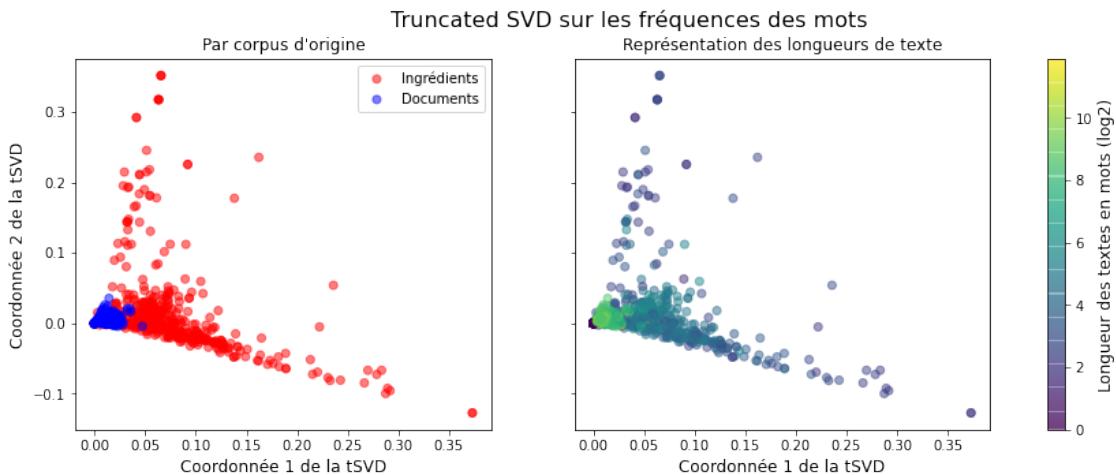


4.3.4 tSVD : fréquences

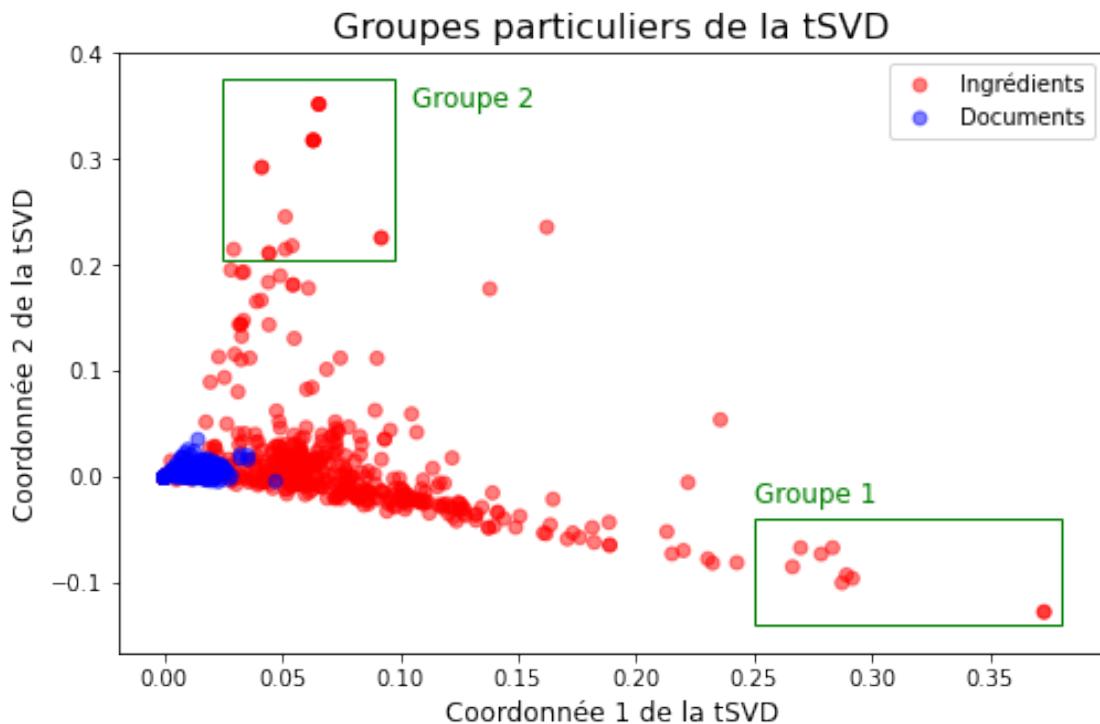
Voila.

```
[329]: transformer = TruncatedSVD(n_components=100)
projected = transformer.fit_transform(tf_corpus)
```

```
[330]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []
handles.append(ax[0].scatter(x=projected[:500, 0], y=projected[:500, 1], c=['red']*500, alpha=0.5))
handles.append(ax[0].scatter(x=projected[500:, 0], y=projected[500:, 1], c=['blue']*500, alpha=0.5))
handles.append(ax[1].scatter(x=projected[:, 0],
                            y=projected[:, 1],
                            c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                            alpha=0.5,
                            cmap='viridis'))
# ax.set_xlim(-15, 10)
# ax.set_ylim(-2, 40)
ax[0].legend(handles, ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("Truncated SVD sur les fréquences des mots", fontsize=16)
ax[0].set_xlabel("Coordonnée 2 de la tSVD", fontsize=12)
ax[0].set_ylabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
# fig.savefig(Path('..') / 'img' / 'tSVD_freq.png', bbox_inches='tight')
```



```
[433]: fig, ax = plt.subplots(figsize=(8, 5))
handles = []
handles.append(ax.scatter(x=projected[:, 0], y=projected[:, 1], c=['red']*500, alpha=0.5))
handles.append(ax.scatter(x=projected[:, 0], y=projected[:, 1], c=['blue']*500, alpha=0.5))
ax.add_patch(mpatch.Rectangle((0.25, -0.14), 0.13, 0.1, fill=False, color='green'))
ax.add_patch(mpatch.Rectangle((0.025, 0.205), 0.073, 0.17, fill=False, color='green'))
ax.annotate('Groupe 1', (0.25, -0.025), fontsize=12, color='green')
ax.annotate('Groupe 2', (0.105, 0.35), fontsize=12, color='green')
ax.legend(handles, ['Ingrédients', 'Documents'])
ax.set_ylabel("Coordonnée 2 de la tSVD", fontsize=12)
ax.set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax.set_title("Groupes particuliers de la tSVD", fontsize=16)
# fig.savefig(Path('..') / 'img' / 'tSVD_freq_groups.png', bbox_inches='tight')
```



```
[434]: sample = pd.concat([corpus_df.reset_index(), pd.DataFrame(projected[:, [0, 1]], columns=['x', 'y'])], axis=1)
```

```
[435]: idx1 = ((projected[:, 0] > 0.25) &
           (projected[:, 0] < (0.25+0.13)) &
           (projected[:, 1] < (-0.14+0.1)) &
           (projected[:, 1] > -0.14))
idx2 = ((projected[:, 0] > 0.025) &
        (projected[:, 0] < (0.025+0.073)) &
        (projected[:, 1] < (0.205+0.17)) &
        (projected[:, 1] > 0.205))

sample.loc[idx1, 'group'] = 'Groupe 1'
sample.loc[idx2, 'group'] = 'Groupe 2'
with pd.option_context("max_colwidth", 100000):
    tex_str = (sample.loc[idx1 | idx2]
                .set_index('group')
                .sort_index()
                .loc[:, ['text', 'x', 'y']]
                .rename({'text': 'Texte'}, axis=1)
                .to_latex(multirow=True,
                          column_format='llcc',
                          index_names=False,
                          ))
```

```

.replace(r'\textbackslash n', r' \newline '))
print(tex_str)
# with open(Path('..') / 'tbls' / 'tSVD_sample.tex', mode='w') as file:
#     file.write(tex_str)

\begin{tabular}{llcc}
\toprule
{} & & Texte & \\
\midrule
Groupe 1 & Thon, eau, sel & 0.373079 & -0.127052 \\
Groupe 1 & Pois chiches, eau, sel. & 0.266423 & -0.084501 \\
Groupe 1 & Haricots beurre, eau, sel & 0.283409 & -0.066561 \\
Groupe 1 & Câpres, eau, vinaigre, sel & 0.291819 & -0.095479 \\
Groupe 1 & Thon albacore, eau, sel & 0.287423 & -0.099627 \\
Groupe 1 & Eau, haricots verts, sel. & 0.278588 & -0.072474 \\
Groupe 1 & Pommes de terre, eau, sel. & 0.289291 & -0.092162 \\
Groupe 1 & Carottes rondelles, eau, sel, sucre & 0.269849 & -0.066600 \\
Groupe 1 & Thon, eau, sel & 0.373079 & -0.127052 \\
Groupe 2 & - 100\% Sémoule de blé dur de qualité supérieure & 0.065551 & 0.351800 \\
Groupe 2 & 100\% haricots blancs & 0.051373 & 0.214921 \\
Groupe 2 & 100\% Sémoule de blé dur de qualité supérieure & 0.065551 & 0.351800 \\
Groupe 2 & Sémoule de blé dur* \newline *issu de l'agriculture biologique & 0.054211 \\
0.218192 \\
Groupe 2 & Tilleul (100\%). & 0.041323 & 0.292199 \\
Groupe 2 & 100\% semoule de blé dur de qualité supérieure & 0.065551 & 0.351800 \\
Groupe 2 & Farine de blé T45 & 0.092002 & 0.225689 \\
Groupe 2 & - Sémoule de blé dur de qualité supérieure \newline - 30\% oeufs frais & 0.044446 \\
0.211188 \\
Groupe 2 & - 100\% Sémoule de blé dur de qualité courante \newline - Contient du gluten & 0.051337 \\
0.245649 \\
Groupe 2 & SEMOULE DE BLE' DUR de qualité supérieure & 0.063201 & 0.317653 \\
Groupe 2 & 100\% Arabica & 0.041324 & 0.292222 \\
Groupe 2 & agaragar 100\% & 0.029466 & 0.214845 \\
Groupe 2 & SEMOULE DE BLE' DUR de qualité supérieure & 0.063201 & 0.317653 \\
Groupe 2 & SEMOULE DE BLE' DUR de qualité supérieure & 0.063201 & 0.317653 \\
Groupe 2 & - Sémoule de blé dur de qualité supérieure \newline - 30\% oeufs frais & 0.044446 \\
0.211188 \\
Groupe 2 & Farine de blé T45 & 0.092002 & 0.225689 \\
\bottomrule
\end{tabular}

```

[]:

open_model

Pierre MASSÉ

June 11, 2020

1 Modèle “ouvert”

L'objet de ce notebook est de démontrer la faisabilité de prédire les listes d'ingrédients depuis des fiches techniques

1.1 Préambule technique

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[2]: # imports and customization of display
# import os
# from functools import partial
import numpy as np
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
# from sklearn.feature_extraction.text import CountVectorizer
# from sklearn.model_selection import train_test_split
# from sklearn.model_selection import cross_val_score, cross_validate
# from sklearn.pipeline import Pipeline
# from matplotlib import pyplot as plt

from src.pimapi import Requester
from src.pimest import PIMIngredientExtractor
# from src.pimest import ContentGetter
# from src.pimest import PathGetter
# from src.pimest import PDFContentParser
# from src.pimest import BlockSplitter
# from src.pimest import SimilaritySelector
# from src.pimest import custom_accuracy
```

```
[3]: # monkeypatch _repr_latex_ for better inclusion of dataframes output in report
def _repr_latex_(self, size='scriptsize',):
    return(f"\\"\\resizebox{{\\linewidth}}{{!}}{{\\begin{{\\size}}}{\\centering{{\\self.to_latex()}}}{\\end{{\\size}}}}\"")
pd.DataFrame._repr_latex_ = _repr_latex_
```

1.2 Extraction des données

On extrait les données depuis le PIM :

```
[4]: requester = Requester('prd')
requester.fetch_all_from_PIM()
requester.result
```

Done

```
[4]: [<Response [200]>,
<Response [200]>,
```

```
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
```

```
[5]: df = requester.result_to_dataframe(record_path='entries', index='uid')
```

1.3 Constitution du périmètre

On conserve les produits qui : - sont de type Epicerie ou Boisson non alcoolisée - portent une liste d'ingrédients - sont en qualité : - soit ont terminé le processus de migration, soit ont été créés après la reprise initiale - et ont le statut "Validé"

```
[6]: # filter by product type
type_mask = df['properties.pprodtop:typeOfProduct'].isin(['grocery', 'nonAlcoholicDrink'])

# keep only those who have ingredients
ingredient_mask = pd.notna(df['properties.pprodc:ingredientsList'])

# filter out those who have not finished migration
df['begin_mig'] = df['facets'].apply(lambda x: 'beginningMigration' in x)
df['end_mig'] = df['facets'].apply(lambda x: 'endMigration' in x)
migration_mask = df.loc[:, 'end_mig'] | ~df.loc[:, 'begin_mig']

# filter out those who are not validated
status_mask = (df.loc[:, 'state'] == 'product.validate')

scope_mask = type_mask & ingredient_mask & migration_mask & status_mask

scope_df = df.loc[scope_mask]
print(f'After filters, there are {len(scope_df)} records in the dataset.')
out_of_scope_df = df.loc[~df.index.isin(scope_df.index)]
print(f'and {len(out_of_scope_df)} records left out.')
```

After filters, there are 3407 records in the dataset,
and 9893 records left out.

1.4 Entraînement : constitution du vocabulaire

On entraîne le modèle sur les listes d'ingrédients du périmètre. Cela revient à fitter le CountVectorizer sous-jacent.

```
[7]: model = PIMIngredientExtractor('prd')
model.fit(scope_df['properties.pprodc:ingredientsList'])
```

```
[7]: <src.pimest.PIMIngredientExtractor at 0x7ff230e1fb20>
```

On peut imprimer une partie du vocabulaire qui a été construit :

```
[8]: print(f'Vocabulary consists in {len(model._count_vect.vocabulary_)} words.\n')
print('Some words examples are :')

for i, word in enumerate(model._count_vect.vocabulary_.keys()):
    print(f'- {word}')
    if i > 6:
        break
```

Vocabulary consists in 2514 words.

Some words examples are :
- morilles
- kombu
- déshydraté
- 100
- eau
- graines
- de
- moutarde

On peut également afficher les mots les plus fréquents dans le corpus de listes d'ingrédients d'entraînement. On constitue d'abord la matrice des textes transformés :

```
[9]: vectorized = model._count_vect.transform(scope_df['properties.pprodc:ingredientsList'])
vectorized.shape
```

[9]: (3407, 2514)

On a bien 3412 documents projetés sur 2509 mots. Si on extrait les plus fréquents, on obtient :

```
[10]: inverse_voc = {val: key for key, val in model._count_vect.vocabulary_.items()}
word_counts = np.asarray(vectorized.sum(axis=0)).squeeze()
print('Most frequent words in vocabulary are:')
for idx in word_counts.argsort()[:-10:-1]:
    print(f'{inverse_voc[idx].ljust(7)}: {word_counts[idx]:5} occurrences')
```

```
Most frequent words in vocabulary are:
de      : 11544 occurrences
sucre   : 2069 occurrences
sel     : 1647 occurrences
eau     : 1266 occurrences
acide   : 1245 occurrences
huile   : 1241 occurrences
lait    : 1228 occurrences
poudre  : 1099 occurrences
en      : 972 occurrences
arôme   : 940 occurrences
```

1.5 Prédiction

Le wrapper `PIMIngredientExtractor` permet de simplement récupérer les informations du PIM et les pièces jointes associées, et de faire tourner le modèle pour extraire le bloc le plus similaire aux listes d'ingrédients.

```
[20]: exec_count = 5
uids = list(out_of_scope_df.sample(exec_count, ).index)
i = 0

for uid in uids:
    try:
        model.compare_uid_data(uid)
        print('\n=====')
        '\n=====\\n')
        i += 1
    except:
        pass
    if i > exec_count:
        break
```

```
Fetching data from PIM for uid 78dd3d32-5c23-495e-a4f9-71f6b3c692d7...
Done
```

```
-----
```

```
Ingredient list from PIM is :
```

```
Sens gourmet Sarl - 15/17 rue du travy ZI sénia 715 94657 Thiais T 01 49 79 98 29 F 01 48 85 36 32
info@sensgourmet.com Code 58050000 (500 gr) Code EAN 8414933570004 RSIPAC N° 31-04482/CAT-31.01506B
DESCRIPTION : Ingrédient d'origine naturel. Agent gélifiant pour des gelées complètement transparentes à
base d'eau. Origine : Europe APPLICATIONS : La gélatine est un ingrédient d'origine naturelle qui a une
grande capacité d'absorption des molécules d'eau. Il s'agit d'une gélatine très élastique avec une bonne
absorption de l'eau. Une température de gélification : 60°C. Dosage : 50 g/kg COMPOSITION : Maltodextrine,
agent épaississant: carrageenan (E407), dextrose, chlorure de potassium (E508), acidifiant: trisodium
citrate (E331ii), agent épaississant: gomme de caroube (E410), saccharose.
```

```
-----
```

```
Supplier technical datasheet from PIM for uid 78dd3d32-5c23-495e-a4f9-71f6b3c692d7 is:
https://produits.groupe-pomona.fr/nuxeo/nxfile/default/78dd3d32-5c23-495e-a4f9-71f6b3c692d7/pprodad:technica
1Sheet/FT-163464_Gelatine%20vegetale%20pot%20500Gx6_SENS%20GOURMET.pdf?changeToken=25-0
```

```
-----
```

```
Downloading content of technical datasheet file...
```

```
Done!
```

```
-----
```

```
Parsing content of technical datasheet file...
```

```
Done!
```

```
-----
```

```
Ingredient list extracted from technical datasheet:
```

```
Code
Code EAN
RSIPAC N°
```

```
DESCRIPTION :
```

```
Ingrédient d'origine naturel. Agent gélifiant pour des gelées complètement transparentes à base d'eau.
Origine : Europe
```

APPLICATIONS :

La gélatine est un ingrédient d'origine naturelle qui a une grande capacité d'absorption des molécules d'eau.
Il s'agit d'une gélatine très élastique avec une bonne absorption de l'eau. Une température de gélification :
60°C. Dosage : 50 g/kg

COMPOSITION :

Maltodextrine, agent épaississant: carrageenan (E407), dextrose, chlorure de potassium (E508), acidifiant: trisodium citrate (E331iii), agent épaississant: gomme de caroube (E410), saccharose.

PARAMETRES ORGANOLEPTIQUES

Aspect : poudre fine, couleur blanche
Goût : neutre, doux
Odeur : neutre

PROPRIETES PHYSIQUES ET CHIMIQUES

Humidité : maximum
ASH

VALEURS NUTRITIONNELLES

Energie
Graisses
Saturées
Glucides
Sucres
Protéines
Sel
Fibres

PROPRIETES MICROBIOLOGIQUES :

microorganismes aérobies mésophiles (cfu/g) Max 5000/g
E.coli en 0.1g
Salmonelle spp (cfu/25gr)
Moisisures (cfu/g)
Levures (cfu/g)
Total coliforms

Sens gourmet Sarl - 15/17 rue du travy - ZI sénia 715 - 94657 Thiais
T 01 49 79 98 29 - F 01 48 85 36 32 - info@sensgourmet.com

=====

=====
=====

Fetching data from PIM for uid 49f82f07-94af-4437-b433-351aaa6837d8...
Done

Ingredient list from PIM is :

None

Supplier technical datasheet from PIM for uid 49f82f07-94af-4437-b433-351aaa6837d8 is:
https://produits.groupe-pomona.fr/nuxeo/nxfile/default/49f82f07-94af-4437-b433-351aaa6837d8/pprodad:technicaleSheet/FT-182074_Bisc%20avoine%20myrt%20Bio%20bte%2045G%20TBegin_Mononaturel.docx.pdf?changeToken=21-0

Downloading content of technical datasheet file...
Done!

Parsing content of technical datasheet file...
Done!

Ingredient list extracted from technical datasheet:

45 g
50 g
50 X 40 X H 85 mm
4751018890904

24 cookies
1008 g
1200 g
170X160X H 190 mm

800 X 1000 X H 1400 mm
4704 units
196 boxes
28 boxes
215 kg
235 kg

=====
=====
Fetching data from PIM for uid e6fdec57-8df2-477a-942a-7d47bba41fd4...
Done

Ingredient list from PIM is :

None

Supplier technical datasheet from PIM for uid e6fdec57-8df2-477a-942a-7d47bba41fd4 is:
https://produits.groupe-pomona.fr/nuxeo/nxfile/default/e6fdec57-8df2-477a-942a-7d47bba41fd4/pprodad:technica1Sheet/FT-178340_Sabots%20securite%20T36%20lic_Mutexil.pdf?changeToken=25-0

Downloading content of technical datasheet file...
Done!

Parsing content of technical datasheet file...
Fetching data from PIM for uid c771677a-b017-43ff-a5de-0d8be410a830...
Done

Ingredient list from PIM is :

Huile de colza, huile d'olive (24 %), vinaigre de vin affiné (conservateur : E222), eau, sel, épice.

Supplier technical datasheet from PIM for uid c771677a-b017-43ff-a5de-0d8be410a830 is:
https://produits.groupe-pomona.fr/nuxeo/nxfile/default/c771677a-b017-43ff-a5de-0d8be410a830/pprodad:technica1Sheet/FT-67277_Vinaigrette%20huile%20olive%20col%2024G_Gyma.pdf?changeToken=34-0

Downloading content of technical datasheet file...
Done!

Parsing content of technical datasheet file...
Done!

Ingredient list extracted from technical datasheet:

Huile de colza, huile d'olive (24 %), vinaigre de vin affiné (conservateur : E222*), eau, sel, épice.
*Contient :
SULFITES.
Rapeseed oil, olive oil (24%), refined wine vinegar (preservative: E222*), water, salt, spice. *Contains:
SULFITES.

=====
=====
Fetching data from PIM for uid cbbd03b7-9cf4-47b3-bae9-28a9af1263ba...
Done

Ingredient list from PIM is :

Potiron 49%, graisse de palme, amidon de pomme de terre, pomme de terre 7,7%, sel, arômes (dont BLÉ, ORGE), huile de maïs, sucre, LACTOSE, oignon grillé 1,9%, extrait de levure, maltodextrine, protéines de LAIT, épices (poivre, noix de muscade).

Supplier technical datasheet from PIM for uid cbbd03b7-9cf4-47b3-bae9-28a9af1263ba is:
https://produits.groupe-pomona.fr/nuxeo/nxfile/default/cbbd03b7-9cf4-47b3-bae9-28a9af1263ba/pprodad:technica1Sheet/FT-86667_Creme%20potiron%20bte%201,155K_Knorr.pdf?changeToken=26-0

Downloading content of technical datasheet file...
Done!

Parsing content of technical datasheet file...
Done!

Ingredient list extracted from technical datasheet:

Liste d'ingrédients: Potiron 49%, graisse de palme, amidon de pomme de terre, pomme de terre 7,7%, sel, arômes (dont BLÉ, ORGE), huile de maïs, sucre, LACTOSE, oignon grillé 1,9%, extrait de levure, maltodextrine, protéines de LAIT, épices (poivre, noix de muscade). Peut contenir : œuf, céleri et moutarde.

=====

[]:

gt_based_model

Pierre MASSÉ

June 11, 2020

1 Modèle basé sur les données manuellement étiquetées

L'objet de ce notebook est de mettre en place le modèle basé sur les données manuellement étiquetées.

1.1 Récupération des données

1.1.1 Préambule technique

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[2]: # imports and customization of display
import os
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width = 108
pd.options.display.latex.repr = True
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

from src.pimapi import Requester
from src.pimest import ContentGetter
from src.pimest import PathGetter
from src.pimest import PDFContentParser
from src.pimest import BlockSplitter
from src.pimest import SimilaritySelector
```

```
[3]: # monkeypatch _repr_latex_ for better inclusion of dataframes output in report
def _repr_latex_(self, size='scriptsize',):
    return(f"\\"\\resizebox{{\\linewidth}}{{!}}{{\\begin{{\\size}}}{\\centering{{\\self.to_latex()}}}{\\end{{\\size}}}}\"")
pd.DataFrame._repr_latex_ = _repr_latex_
```

1.1.2 Chargement du fichier des données manuellement étiquetées

On commence par charger le fichier csv contenant les données manuellement étiquetées.

```
[4]: ground_truth_df = pd.read_csv(Path('..') / '..' / 'ground_truth' / 'manually_labelled_ground_truth.csv',
                                 sep=';',
                                 encoding='latin-1',
                                 index_col='uid')
ground_truth_df.head()
```

uid	designation	ingredients
a0492df6-9c76-4303-8813-65ec5cbfa70	Concentré liquide Asian en bouteille 980 ml CHEF	Eau, maltodextrine, sel, arômes, sucre, arôme ...
d183e914-db2f-4e2f-863a-a3b2d054c0b8	Pain burger curry 80 g CREATIV BURGER	Farine de blé T65, eau, levure, vinaigre de ci...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8	Macaroni en sachet 500 g PANZANI	- 100% Sémoule de BLE dur de qualité supérieur...
528d4be3-425c-4f8b-8a87-12f1bc645ddd	Fève de Tonka en sachet 100 g COMPTOIR COLONIAL	fève de tonka (graines ridées de 25 à 50mm de ...
51b38427-b2ea-4c56-93e8-4242361ef31b	Caviar d'aubergine en pot 500 g PUGET RESTAURA...	Aubergine 60,5% (aubergine, huile de tournesol...)

```
[5]: ground_truth_uids = list(ground_truth_df.index)
ground_truth_uids[:5]
```

```
[5]: ['a0492df6-9c76-4303-8813-65ec5ccbfa70',  
     'd183e914-db2f-4e2f-863a-a3b2d054c0b8',  
     'ab48a1ed-7a3d-4686-bb6d-ab4f367cadab',  
     '528db4e3-425c-4f8b-8a87-1f21bc645dd',  
     '51b38427-b2ea-4c56-93e8-4242361ef31b']
```

1.1.3 Pipeline d'acquisition du contenu des données

On commence par construire un premier pipeline d'acquisition des données. Il fonctionne en 3 étapes : - détermination du chemin vers lequel aller chercher les fiches techniques - récupération du contenu binaire du fichier - conversion de ce contenu binaire en texte

```
[6]: acqui_pipe = Pipeline([('PathGetter', PathGetter(ground_truth_uids=ground_truth_uids,
                                                    train_set_path=Path('..') / '..' / 'ground_truth',
                                                    ground_truth_path=Path('..') / '..' / 'ground_truth',
                                                    )),
                           ('ContentGetter', ContentGetter(missing_file='to_nan')),
                           ('ContentParser', PDFContentParser(None_content='to_empty'))),
                          ],
                         verbose=True)
```

```
[7]: texts_df = acqui_pipe.fit_transform(ground_truth_df)
texts_df.sample(5)
```

```
[Pipeline] ... (step 1 of 3) Processing PathGetter, total= 0.1s
[Pipeline] ... (step 2 of 3) Processing ContentGetter, total= 0.6s
Launching 8 processes.
[Pipeline] ... (step 3 of 3) Processing ContentParser, total= 37.2s
```

[7]:	designation	ingredients	path	content	text
uid : e4d79418-1910-4810-8974-9002c2ec39e					
e4d79418-3501-4614-9154-4dbbb035021	CREPINE de cèpe à la croute en boîte 4/4 DEMETRA	cèpe 70% (Boletus edulis et respectives familles), huile végétale, sel, levure de boulangerie, huile de TBE TYPE 45, 250g	./.../ground_truth/0/e4d79418-1910-4810-8974-9002c2ec39e	b7F9F-1.1#at 0 obj\act\verb\Name{[4 0 8 0 8]}\n	FICHE TECHNIQUE DU PRODUIT \verb\Name{[4 0 8 0 8]}\n
cl3deeb4-9571-4533-8826-11061068203	PIZZA aux tomates et au poulet au COMPLET	farine de blé T45, huile d'olive, huile de colza, huile de tournesol, huile de BEURRE, sel, levure de boulangerie, huile de TBE TYPE 45, 250g	./.../ground_truth/0/cl3deeb4-9571-4533-8826-11061068203	b7F9F-1.1#at 0 obj\act\verb\Name{[4 0 8 0 8]}\n	FICHE TECHNIQUE \verb\Name{[4 0 8 0 8]}\n
b7f47621a-f0de-4389-8074-e1769864ca	Haricots verts braisés en poche 1,7 kg SOMBU	Haricots verts braisés en poche 1,7 kg SOMBU, sel, huile de TBE TYPE 45, 250g	./.../ground_truth/b7f47621a-f0de-4389-8074-e1769864ca	b7F9F-1.1#at 0 obj\act\verb\Name{[4 0 8 0 8]}\n	FICHE TECHNIQUE \verb\Name{[4 0 8 0 8]}\n

On peut afficher quelques textes récupérés par le pipeline :

```
[8]: with pd.option_context("max_colwidth", 1000):
    pass
#     print(texts_df.sample(3, random_state=42)['text'])
#     (texts_df.sample(3, random_state=42)['text']
#      .to_latex(Path('..') / 'tb1s' / 'processed_FT.tex',
#                 index=False,
#                 index_names=False,
#                 column_format='p{\linewidth}',
#                 na_rep='-', 
#                 escape=True,
#                 ))
# 
```

1.2 Découpage en blocs

On découpe les longs textes en blocs. Chaque texte devient une liste de strings plus court.

```
[9]: def splitter(text):
        return(text.split('\n\n'))
```

```
[10]: split_transfo = BlockSplitter(splitter_func=splitter)
       splitted_df = split_transfo.fit_transform(texts_df)
       splitted_df.sample(5)
```

Launching 8 processes.

On peut afficher un exemple de texte découpé en blocs :

```
[11]: sep = '\n-----\n'
sample = splitted_df.sample(1, random_state=39)[['blocks']].iloc[0]
print(sep.join(sample))

tex_str = (
pd.DataFrame(sample, columns=['BloCs'])
.to_latex(column_format='p{10cm}', 
          index=False,
```

```

        index_names=False,
        escape=True,
    )
.replace(r'\textbackslash n', '\\newline ')
)

#with open(Path('..') / 'tbls' / 'block_example.tex', mode='w') as file:
#    file.write(sep.join(sample).replace('\n', r' \newline '))

```

30/12/19

Date d'impression :

Remarque :

Les informations contenues dans cette fiche technique sont données de bonne foi, en l'état actuel de nos connaissances, et selon les indications communiquées par le producteur ou le fournisseur. Il appartient au client de vérifier la conformité de la marchandise par rapport à l'usage qu'il en fait.

Création :

12/06/12

12 rue René Cassin
37390 NOTRE DAME

Tél :
02 47 85 55 00
Fax :02 47 41 33 32

FICHE TECHNIQUE

Mélange du trappeur, 70 g
Trapper blend, 70g

Code article KEREX
Nom latin (si disponible)
/ EAN Code
Code barre

/ KEREX Code

/ (Latin name)

TEEPTRAPPEUR
X
3760063322262

Poids net
Poids brut
Origine

/ net weight
/ gross weight
/ Origin

0,07 Kilogramme
0,125 Kilogramme
CANADA

/ General information

Informations générales
DLUO conseillée / "Best before date" recommended
Nomenclature douanière / Customs code
Conditions idéales de stockage
/ Conditions of storage
Ingrédients :

Conserver dans un endroit frais et sec
Store in a cool dry place

5 ans / 5 years
0910999900

Sucre, poivre noir, coriandre, légumes déshydratés (ail, oignon, poivron rouge), sel de mer, sucre d'éryable, arôme d'éryable naturel, huile végétale (canola)
Sugar, black pepper, coriander, dehydrated vegetables (garlic, onion, red bell pepper), sea salt, maple sugar, natural maple aroma,

vegetable oil (canola)

/ Ingredients

Contaminants / Contaminating
Ionisation / Irradiation

OGM / GMO

Pesticides/ Pesticides

Métaux Lourds

/ Heavy Metals

Allergènes et leurs dérivés (si présents)
/ Allergens (if existing)

Conformité à la directive 1999/2/CE (22/02/99)
Produit non ionisé et ne contenant pas d'ingrédients ionisés.

Not irradiated
accordingly with the Reg 1999/2/CE (22/02/99).

Free from GMO
Ne contient pas d'OGM, est non soumis à l'étiquetage sur les OGM
Conforme à la directive 396/2005 /CE.
In accordance with Reg 396/2005 /CE.
Conforme au règlement 1881/2006 /CE
In accordance with Reg 1881/2006 /CE..

Gluten

Crustacés

Oeufs

Poisson

Soja

Lait

Fruits à coque - Arachides

Céleri

Moutarde

Sésame

Sulfites

Lupin

Mollusques

/ Gluten

/ Crustaceans

/ Eggs

/ Fish

/ Soy

/ Milk

/ Peanuts and Treenuts

/ Celery and celeriac

/ Mustarde

/ Sésame

/ Sulphites

/ Lupin

/ Shellfish

Absence

Caractères microbiologiques

/ Microbiological characteristics

Microorganismes aérobies 30 °C

Escherichia coli

Salmonelles

Levures

Moississures

Aflatoxine Total

```

Aflatoxine B1
-----
/ Total plat count (APC)
E. Coli
/
/ Salmonella
/ Yeasts
/ Moulds
/ Total aflatoxin
B1 aflatoxin
/
-----
NF V05-051 < 6 000 000 / g
NF V08-053 < 10 / g
NF V08-052 Absence dans 25g
NF V08-059 < 10 000 / g
NF V08-059 < 10 000 / g
Kit Enzymatique < 10 ppb
Kit Enzymatique < 5 ppb
-----

```

1.3 Train / Test split

On procède au découpage en un jeu d'entraînement et un jeu de test en gardant 400 produits pour l'entraînement et 100 produits pour le test :

```
[12]: train, test = train_test_split(splitted_df, train_size=400, random_state=42)
```

1.4 Entraînement sur le jeu d'entraînement

On entraîne un modèle SimilaritySelector, sur le set d'entraînement :

```
[13]: model = SimilaritySelector()
```

```
[14]: model.fit(train['blocks'], train['ingredients'])
```

```
[14]: <src.pimest.SimilaritySelector at 0x7f048240de20>
```

```
[15]: len(model.count_vect.vocabulary_)
```

```
[15]: 1204
```

```
[16]: predicted = pd.Series(model.predict(test['blocks']),
                           index=test.index,
                           name='predicted')
predicted = pd.concat([test['ingredients'], predicted], axis=1)
predicted.sample(5)
```

uid	ingredients	predicted
eadb972c-6623-472d-a11d-489a7faf6f11	- Soja fermenté naturellement (soja, sel, eau)...	Céréales contenant du gluten \net des produits...
6267bf8-2529-4bc6-ba4b-26760f0522b3	eau gazeifiée\ncolorant : E150d\nacidifiants ...	CocaCola Light mini 8 x 150 mlEAN544900023980...
04235024-80f3-46c2-bad0-aae0dfab024	Persil	Céréales contenant du gluten (à savoir blé, se...
e51b7fd6-d878-47f8-a36b-f10f8d4087bd	Débris de truffes d'hiver, jus de truffes, sel	\nOrigine Truffes et Sel : ...
6566e18d-3bdd-43d8-ab0c-de51894621f9	Pommes 89%, purée de fraises à base de concent...	Liste ingrédients : Pommes 89%, purée de frais...

```
[17]: predicted['pred_len'] = predicted['predicted'].apply(len)
sub_sample = predicted.loc[predicted['pred_len'] <= 500, ['ingredients', 'predicted']]
sub_sample.head(5)
```

uid	ingredients	predicted
2892dd68-e3a6-474c-b543-3ebfd3490658	Café instantané, café torrefié moulu (3%).	- NESTLÉ a un système de management de la qual...
3634fb1e-ee79-41d1-8aaa-084c1fae5bd5	Poire 99,9%, antioxydant: acide ascorbique.	Ce produit est une purée de fruits obtenue à p...
345591f4-d887-4ddc-bb40-21337fa9269d	Gésier de dinde émincé 50%, graisse de canard ...	Gésier de dinde émincé 50%, graisse de canard...
13980d31-9002-457d-8d49-b451f08f473c	Educorants sorbitol, isomalt, sirop de maltit...	Educorants sorbitol, isomalt, sirop de maltit...
74297717-3fa8-4aed-95cc-e8737cia6157	sucré, amidon modifié, LACTOSERUM en poudre, d...	Z16005 / 002\nsucré, amidon modifié, LACTOSERU...

On constitue une table pour intégration dans le rapport :

```
[18]: with pd.option_context("max_colwidth", 100000):
    tex_str = (
        sub_sample.sample(20, random_state=41)
        .replace(r'^\s*$', np.nan, regex=True)
        .to_latex(index=False,
                   index_names=False,
                   column_format='p{7cm}p{7cm}',
                   na_rep='<rien>',
                   longtable=True,
                   header=["Liste d'ingrédients cible", "Liste d'ingrédients prédicté"],
                   label='tbl:GT_prediction_sample',
                   caption="Extrait des résultats de la prédition",
                   )
        .replace(r'\textbackslash n', r' \newline ')
        .replace(r'\\', r'\\ \hline')
    )

# with open(Path('..') / 'tbls' / 'GT_prediction_sample.tex', 'w') as file:
#     file.write(tex_str)
```

Performance_measurement

Pierre MASSÉ

June 11, 2020

1 Mesure de la performance du modèle

L'objet de ce notebook est d'illustrer la méthodologie de mesure de la performance du modèle.

1.1 Préambule technique

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[2]: # imports and customization of display
import os
from functools import partial
import numpy as np
from scipy.stats import linregress
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.pipeline import Pipeline
from matplotlib import pyplot as plt
import matplotlib.ticker as mtick
import seaborn as sns

from src.pimest import ContentGetter
from src.pimest import PathGetter
from src.pimest import PDFContentParser
from src.pimest import BlockSplitter
from src.pimest import SimilaritySelector
from src.pimest import custom_accuracy
from src.pimest import text_sim_score
from src.pimest import text_similarity
from src.pimest import build_text_processor
```

```
[4]: # monkeypatch _repr_latex_ for better inclusion of dataframes output in report
def _repr_latex_(self, size='scriptsize',):
    return(f"\\"\\resizebox{{\\linewidth{{!}}}}{{{size}}}{{\\begin{{{size}}}}\\centering{{{self.to_latex()}}}\\end{{{size}}}}")
pd.DataFrame._repr_latex_ = _repr_latex_
```

1.2 Acquisition des données

On récupère les données manuellement étiquetées et on les intègre dans un dataframe

```
[5]: ground_truth_df = pd.read_csv(Path('..') / '..' / 'ground_truth' / 'manually_labelled_ground_truth.csv',
                                    sep=';',
                                    encoding='latin-1',
                                    index_col='uid')
ground_truth_uids = list(ground_truth_df.index)

acqui_pipe = Pipeline([
    ('PathGetter', PathGetter(ground_truth_uids=ground_truth_uids,
                             train_set_path=Path('..') / '..' / 'ground_truth',
                             ground_truth_path=Path('..') / '..' / 'ground_truth',
                             )),
    ('ContentGetter', ContentGetter(missing_file='to_nan')),
    ('ContentParser', PDFContentParser(none_content='to_empty')),
```

```
[5]:
    ],
    verbose=True)

texts_df = acqui_pipe.fit_transform(ground_truth_df)
texts_df['ingredients'] = texts_df['ingredients'].fillna('')
texts_df.sample(4)
```

[Pipeline] ... (step 1 of 3) Processing PathGetter, total= 0.1s
[Pipeline] ... (step 2 of 3) Processing ContentGetter, total= 0.6s
Launching 8 processes.
[Pipeline] ... (step 3 of 3) Processing ContentParser, total= 37.0s

uid	designation	ingredients	path	content	text
baf7852-d4cd-4860-9ab3-19c38a5dfc4b	Café moulu Sélection 100% arabica Bio en paquet...	Café*, *Ingrédient issu de l'agriculture biolo...	..././ground_truth/d5aaef70-6b1e-4535-9d44-dbe...	b'10DF-1.5v7\ue2\lxa3\xcf\xd3\y\al13 0 obj\r</...	www.destination-bio.com - contact@destination...
ca4f64ed-47ac-4071-a16b-8fb479a3b0d	Ravioilli pur bœuf en boîte 1/2 MAIN COUR...	Ravioilli pur bœuf (semoule de blé dur, eau, g...	..././ground_truth/baf17852-d4cd-4860-9ab3-19c...	b'10DF-1.7v\ue2\lxd9\ya3\xcb4\xcf\xd3\y\al13 0 obj\r</...	Service Qualité CCQ/Univision Plats Cuisinés...
07772edf-f0d8-4b01-9517-59a3b0a5cb	Confiture extra de fraise en pot verre 30 g RS...	Fraises*, sucre, eau, eau de vannerie, jus de c...	..././ground_truth/c441449c-ac9-4071-a16b-8fb...	b'10DF-1.4v2\ue2\lxa3\xcf\xd3\y\al13 0 obj\r</...	route de Givry/la Boîte à Miel 39 2...
	Tarte sablée aux fruits en forme de cœur...	Tarte sablée aux fruits en forme de cœur, fruits de la...	..././ground_truth/07772edf-f0d8-4b01-9517-59a...	b'10DF-1.\ue2\lxa3\xcf\xd3\y\al13 0 obj\r</...	12 HEART TART SHELL WITH BUTTERFLY>Addition: 10...

On splitte les textes en blocs de manière basique.

```
[6]: def splitter(text):
    return(text.split('\n\n'))

split_transfo = BlockSplitter(splitter_func=splitter)
splitted_df = split_transfo.fit_transform(texts_df)
splitted_df.sample(4)
```

Launching 8 processes.

uid	designation	ingredients	path	content	text	blocks
45a4ddde-9e07-42ab-b77f-700540c41c1	Mix long Indice, épices en sac 10 kg EPISIENNE	..././ground_truth/07772edf-f0d8-4b01-9517-59a...	b'10DF-1.4.5\ue2\lxa3\xcf\xd3\y\al13 0 obj\r</...	Infusion Baobab Sharp Mix...		
42070101-07b8-472b-912b-11543b2a2027	CONFITURE DE FRAMBOISE BIO FRUITIERE PROACT...	Bonbons	..././ground_truth/07772edf-f0d8-4b01-9517-59a...	Infusion Baobab 'la'20 pyra...	[Infusion Bonbons , 20 pyram...]	
22703016-f12a-4233-924d-1ad1c1cf7928	CHIPS PRALINE AMANDE FOURRÉ 1.25KG	Chocolat au lait (sucre, beurre de cacao, LIMT...	..././ground_truth/07772edf-f0d8-4b01-9517-59a...	Bébés Chocoletier 'la'la180 rue Clément...	[Bébés Chocoletier , 180 rue Clément AD...]	
1970a040-57ae-4270-9f18-af1473bb0955	Elle dur patate en sac 5 kg VIVIEN PAULLE	..././ground_truth/07772edf-f0d8-4b01-9517-59a...	b'10DF-1.\ue2\lxa3\xcf\xd3\y\al13 0 obj\r</...	ELLE DUR PATATE Additif saveur PG2007-6.0.7uler...	[Elle Dur PATATE, Référence PG2007-6.0.7uler...]	

1.3 Train/Test split, entraînement et transformation

On effectue classiquement les étapes de train/test split, on entraîne le modèle sur le set d'entraînement et on le lance sur le set de test.

```
[7]: train, test = train_test_split(splitted_df, train_size=400, random_state=42)
model = SimilaritySelector(similarity='projection')
model.fit(train['blocks'], train['ingredients'])
predicted = pd.Series(model.predict(test['blocks']),
                       index=test.index,
                       name='predicted')
predicted = pd.concat([test['ingredients'], predicted], axis=1)
predicted.sample(4)
```

uid	ingredients	predicted
de22b1fa-039d-479e-b36e-608946b75bb8	NOISETTES entières décortiquées.	Les noisettes décortiquées entières sont les f...
6bdb201d-5879-423b-96b8-3ae88b857818	vinaigre d'alcool, piment jalapeno, eau, sel, ...	Liste d'ingrédients : vinaigre d'alcool, pimen...
7f622727-e4ad-45cc-9af4-4509acf91154	Eau, huile de tournesol, beurre 9,5 %, jaune d...	Une sauce onctueuse et savoureuse avec un équi...
194419d0-d9f2-4799-81ac-d9e3aa77fd27	Pommes de terre, eau, sel.	Anhydride sulfureux et sulfites en \nconcentra...

1.4 Mesure de la performance : Précision

1.4.1 Approche naïve

Dans cette première version, on calculera une précision brute, où seuls les strings parfaitement identiques sont considérés comme ok.

```
[8]: predicted['result'] = (predicted['ingredients'].fillna('') == predicted['predicted'].fillna(''))
predicted['result'].value_counts()
```

```
[8]: False    99
True      1
Name: result, dtype: int64
```

On a une précision très faible, 1%. L'unique liste d'ingrédients du set de test correctement prédite est la suivante :

```
[9]: print(predicted[predicted['result']].iloc[0, 0])
```

Sirup de glucose, sucre, eau, stabilisants (E440i, E440ii, E415), acidifiants (E330, E450i), conservateur (E202).

1.4.2 Cross-validation de l'approche naïve

Pour avoir une vision plus précise de la performance du modèle, on peut effectuer une cross-validation sur le set d'entraînement.

On commence par définir une fonction de scoring, qui pourra être appelée par la fonction standard de cross-validation de scikit-learn. Comme précédemment, il s'agit d'une fonction d'accuracy basique :

```
[10]: def accuracy_scorer(estim, X, y):
        y_pred = estim.predict(X)
        return((y_pred == y).mean())
```

On retrouve évidemment le même score que précédemment lorsqu'on utilise cette fonction sur le set de test :

```
[11]: accuracy_scorer(model, test.reset_index()['blocks'], test.reset_index()['ingredients'])
```

```
[11]: 0.01
```

Si on lance la cross-validation avec les paramètres par défaut ($cv=5$), on obtient le résultat suivant :

```
[12]: X = splitted_df.reset_index()['blocks'].copy()
y = splitted_df.reset_index()['ingredients'].copy()

cross_val = cross_validate(model,
                           X=X,
                           y=y,
                           scoring=accuracy_scorer,
                           )
print(f'Strict accuracy yields a result of {np.mean(cross_val["test_score"]):.2%} +/-{np.std(cross_val["test_score"]):.2%}')
print(cross_val['test_score'])
```

```
Strict accuracy yields a result of 2.00% +/-0.63%
[0.02 0.02 0.02 0.01 0.03]
```

On voit que sur chacun des 5 folds (validation sur 400 produits), l'accuracy varie entre 1 et 3%.

Si on trace l'accuracy et la standard deviation pour plusieurs valeurs de cv , on obtient les résultats suivants :

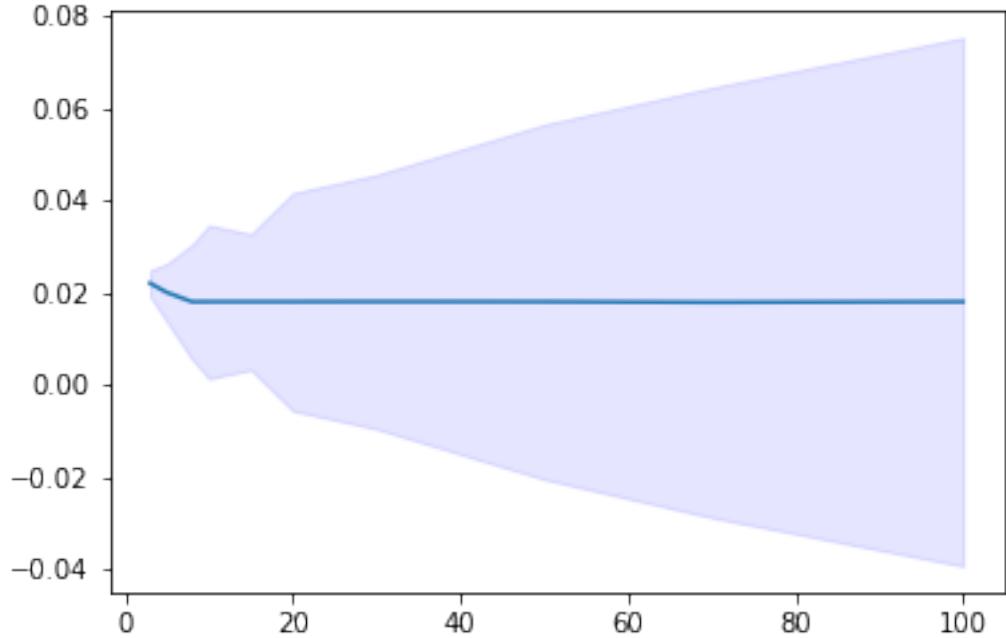
```
[13]: x = [3, 5, 8, 10, 15, 20, 30, 50, 70, 100]
mean = np.array([])
std = np.array([])
for n_cv in x:
    cross_val = cross_validate(model,
                               X=splitted_df['blocks'],
                               y=splitted_df['ingredients'],
                               scoring=accuracy_scorer,
                               cv=n_cv,
                               )
    mean = np.append(mean, [np.mean(cross_val['test_score'])], axis=0)
    std = np.append(std, [np.std(cross_val['test_score'])], axis=0)

print('mean:', mean, '\nstandard dev:', std)
```

```
mean: [0.02200418 0.02      0.01798515 0.018      0.01800357 0.018
0.01801471 0.018      0.01785714 0.018      ]
standard dev: [0.0028574  0.00632456  0.01245353  0.01661325  0.01470345  0.02357965
0.02753429 0.03841875 0.04656573 0.05723635]
```

```
[14]: fig, ax = plt.subplots()
ax.plot(x, mean)
ax.fill_between(x, (mean - std), (mean + std), color='b', alpha=.1)
```

```
[14]: <matplotlib.collections.PolyCollection at 0x7fa4ac4a66d0>
```



Il apparaît que l'accuracy se situe aux alentours de 2%, avec un écart type important si on le compare à cette accuracy.

```
[15]: cross_val = cross_validate(model,
                               X=splitted_df['blocks'],
                               y=splitted_df['ingredients'],
                               scoring=accuracy_scorer,
                               cv=10,
                               )
print(f'Strict accuracy yields a result of {np.mean(cross_val["test_score"]):.2%} +/-{np.std(cross_val["test_score"]):.2%}')
print(cross_val['test_score'])
```

Strict accuracy yields a result of 1.80% +/-1.66%
[0.04 0. 0. 0.04 0.02 0. 0.02 0. 0.04]

1.4.3 Ajout d'une étape de text-postprocessing

On utilise la fonction `custom_accuracy` définie dans le module `pimest` pour calculer l'accuracy avec du text processing. Elle prend en paramètre les mêmes attributs que le `CountVectorizer` de scikit-learn, en plus d'un attribut ```tokenize`'' qui va tokenizer le résultat (pour prise en compte des whitespace et de la ponctuation).

```
[16]: custom_accuracy(model,
                      test['blocks'].fillna(''),
                      test['ingredients'].fillna(''),
                      tokenize=True,
                      strip_accents='unicode',
                      lowercase=True,
                      )
```

[16]: 0.14

L'accuracy est maintenant estimée à 14% (vs. 1%) sur le set de test, après entraînement sur le set d'entraînement.

On peut manuellement inspecter les blocks identiques, en reproduisant le comportement de la fonction `d'accuracy` :

```
[17]: def text_processor(text, **kwargs):
    unused_model = CountVectorizer(**kwargs)
    prepro = unused_model.build_preprocessor()
    token = unused_model.build_tokenizer()
    return(' '.join(token(prepro(text))))
```

```
partial_processor = partial(text_processor, strip_accents='unicode', lowercase=True)
```

```
[18]: prediction = model.predict(test['blocks'].fillna('')).rename('predicted')
processed_prediction = prediction.apply(partial_processor)
processed_prediction.sample(3)
```

```
[18]: uid  
5a7f235d-eba4-43b2-ab52-2c2b93f68a67  
b7d7621a-fcdd-4487-9b38-e07fae698c4a  
df1caa23-9714-4659-803b-33501d64eed  
Name: predicted, dtype: object  
cette fiche technique pas de valeur contractue...  
egoutter ne pas rincer faire sauter minutes av...  
liste des ingredients sucre pate de cacao beur...
```

```
[19]: processed_ground_truth = test['ingredients'].fillna('').apply(partial_processor)
processed_ground_truth.sample(3)
```

```
[19]: uid  
484ac00a-a670-46a9-a9c4-5114174d9e3b  
63968dc3-6e7c-4056-bd53-820c6cc925be  
8dec0469-c9f5-4139-be25-efa258959444  
Name: ingredients, dtype: object  
pommes de terre 59 celeris 40 amidon de mais s...  
carottes eau sucre sel vinaigre alcool acidifi...  
sucre sirop de glucose graisse de palme humect...
```

```
[20]: corrects = test.join(prediction).loc[processed_prediction == processed_ground_truth , ['ingredients', 'predicted']]  
corrects
```

[20]:	ingredients	predicted
uid		
345591f4-d887-4ddc-bb40-21337fa9269d	Gésier de dinde émincé 50%, graisse de canard ...	Gésier de dinde émincé 50%, graisse de canard...
13980d31-9002-457d-8d49-b451f08f473c	Edulcorants sorbitol, isomalt, sirop de maltit... mini poivrons jaunes, eau, sucre, sel, afferm... mini poivrons jaunes, eau, sucre, sel, afferm...	Edulcorants sorbitol, isomalt, sirop de maltit... mini poivrons jaunes, eau, sucre, sel, afferm...
c3b64df-e586-4f10-8e58-15fbf0816acb	Farine de BLE, huile de colza non hydrogénée, ... Pommes de terre 59,5 % - Céleris 40 % - Amidon...	Farine de BLE, huile de colza non hydrogénée, ... Pommes de terre 59,5 % - Céleris 40 % - Amidon...
04819d1b-9653-42e7-b25-9dc9b87b0f62	Pommes de terre 59,5 % - Céleris 40 % - Amidon...	Pommes de terre 59,5 % - Céleris 40 % - Amidon...
484ac00a-a670-46a9-a9c4-511474d9e3b3		
49b11281-34ea-44b0-a11c-4ae21d4c58e3		
d59d96cb-0230-4090-8220-78ce8496fd91	Amidon de maïs* - Lait écrémé* - Sel - Fécule ...	Amidon de maïs* - Lait écrémé* - Sel - Fécule ...
b8cbe6f9-71d4-4e51-a169-1c163d49a561	Farine de FROMENT, poudre de LACTOSERUM, sucre...	Farine de FROMENT, poudre de LACTOSERUM, sucre...
a0492df6-9c76-4303-8813-65ec5ccbfa70	Eau, maltodextrine, sel, arômes, sucre, arôme ...	Eau, maltodextrine, sel, arômes, sucre, arôme ...
09e45b38-4da1-4eb5-888a-3eb437a2291	OEUFS, farine de BLE, sucre, amidon de BLE, st...	OEUFS, farine de BLE, sucre, amidon de BLE, st...
4f83306f-66de-4545-9b12-7790b57b61ae	Sirop de glucose, sucre, eau, stabilisants (E4...	Sirop de glucose, sucre, eau, stabilisants (E4...
5ceee698-e6ff-4b91-c232-1d8fb1ff85a7	Flageolets verts. Jus : eau, sel, affermissant...	Flageolets verts. Jus : eau, sel, affermissant...
63986dc3-6e7c-4056-bd53-820c6cc925be	Carottes, eau, sucre, sel, vinaigre d'alcool, ...	Carottes, eau, sucre, sel, vinaigre d'alcool, ...
dc536305-82fd-4afe-a472-5056ca0e21ea	Légumes 43,2 % (pomme de terre, oignon, carott...	Légumes 43,2 % (pomme de terre, oignon, carott...

```
[21]: with pd.option_context("max_colwidth", 100000):
    tex_str = (
        corrects.replace(r'^\s*$', np.nan, regex=True)
            .to_latex(index=False,
                       index_names=False,
                       column_format='p{7cm}p{7cm}',
                       na_rep='<rien>',
                       longtable=False,
                       header=[["Liste d'ingrédients cible", "Liste d'ingrédients prédicté"],
                               # label='tbl:GT_postprocessed_corrects',
                               # caption="Prédictions identifiées comme correctes après postprocessing",
                               ],
                       )
            .replace(r'\textbackslash n', r' \newline ')
            .replace(r'\\', r'\\ \\hline')
    )
    # with open(Path('..') / 'tbls' / 'GT_postprocessed_corrects.tex', 'w') as file:
    #     file.write(tex_str)
```

1.4.4 Cross-validation de l'approche avec text processing

On fait tourner une cross-validation sur l'ensemble du dataset. On définit d'abord la fonction qui va permettre de calculer le score avec l'ensemble des fonctionnalités de text processing : - retrait des accents - remplacement des whitespaces par des espaces simples - retrait de la ponctuation - mise en minuscule

```
[22]: processed_accuracy = partial(custom_accuracy,
                                 tokenize=True,
                                 strip_accents='unicode',
                                 lowercase=True,
                                 )
cross_val = cross_validate(model,
                           X=splitted_df['blocks'].fillna(''),
                           y=splitted_df['ingredients'].fillna(''),
                           scoring=processed_accuracy,
                           )
print(f'Processed accuracy yields a result of {np.mean(cross_val["test_score"]):.2%} +/-{np.std(cross_val["test_score"]):.2%}')
```

```
print(cross_val['test_score'])
```

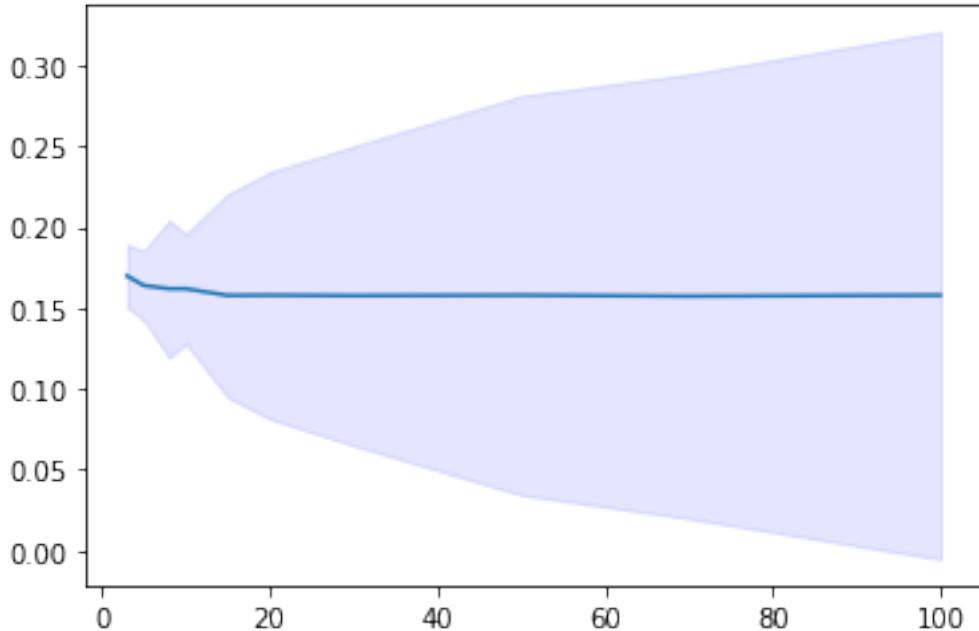
Processed accuracy yields a result of 16.40% +/-2.15%
[0.19 0.15 0.17 0.13 0.18]

```
[23]: x = [3, 5, 8, 10, 15, 20, 30, 50, 70, 100]  
mean = np.array([])  
std = np.array([])  
for n_cv in x:  
    cross_val = cross_validate(model,  
                                X=splitted_df['blocks'].fillna(''),  
                                y=splitted_df['ingredients'].fillna(''),  
                                scoring=processed_accuracy,  
                                cv=n_cv,  
                                )  
    mean = np.append(mean, [np.mean(cross_val['test_score'])], axis=0)  
    std = np.append(std, [np.std(cross_val['test_score'])], axis=0)  
  
print('mean:', mean, '\nstandard dev:', std)
```

mean: [0.16994926 0.164 0.16199437 0.162 0.15787285 0.158
0.15772059 0.158 0.15739796 0.158]
standard dev: [0.0193811 0.02154066 0.04252373 0.034 0.06292686 0.07639372
0.09256313 0.12343419 0.13740531 0.16320539]

```
[24]: fig, ax = plt.subplots()  
ax.plot(x, mean)  
ax.fill_between(x, (mean - std), (mean + std), color='b', alpha=.1)
```

```
[24]: <matplotlib.collections.PolyCollection at 0x7fa4ac6de9d0>
```



```
[25]: cross_val = cross_validate(model,  
                                X=splitted_df['blocks'].fillna(''),  
                                y=splitted_df['ingredients'].fillna(''),  
                                scoring=processed_accuracy,  
                                cv=10,  
                                )  
print(f'Processed accuracy yields a result of {np.mean(cross_val["test_score"]):.2%} +/-{np.std(cross_val["test_score"]):.  
~2%}')  
print(cross_val['test_score'])
```

```
Processed accuracy yields a result of 16.20% +/-3.40%
[0.2 0.18 0.18 0.12 0.12 0.22 0.14 0.12 0.16 0.18]
```

1.5 Mesure de la performance : Similarité

1.5.1 Mesures

On peut également mesurer la similarité plutôt qu'uniquement l'accuracy. Cela permet de valoriser les textes qui ``ressemblent'' aux listes d'ingrédients cibles plutôt que les compter comme des erreurs.

```
[26]: similarity_kinds = ['levenshtein',
                      'damerau-levenshtein',
                      'jaro',
                      'jaro-winkler',
                     ]

sim_dict = dict()

for similarity in similarity_kinds:
    sim = text_sim_score(model,
                          test['blocks'].fillna(''),
                          test['ingredients'].fillna(''),
                          similarity=similarity,
                         )
    sim_dict[similarity] = f'{sim:.2%}'
print(f'Similarity with {similarity} similarity is {sim:.2%}')


Similarity with levenshtein similarity is 47.88%
Similarity with damerau-levenshtein similarity is 47.88%
Similarity with jaro similarity is 63.30%
Similarity with jaro-winkler similarity is 65.28%
```

Les similarités de Levenshtein et Damerau-Levenshtein donnent des résultats identiques, à presque 50% de similarité moyenne. Celles basées sur Jaro tournent aux alentours de 65%, comme on s'y attendait dans la mesure où elle est très ``indulgente'' sur les textes longs.

Si on effectue des cross validations sur chacune de ces distances sur le dataset complet, on obtient :

```
[27]: similarities = {similarity: partial(text_sim_score, similarity=similarity) for similarity in similarity_kinds}

cross_vals = dict()

for similarity in similarity_kinds:
    cross_vals[similarity] = cross_validate(model,
                                              splitted_df['blocks'].fillna(''),
                                              splitted_df['ingredients'].fillna(''),
                                              scoring=similarities[similarity],
                                              cv=10,
                                             )

for similarity in similarity_kinds:
    print(f'Model evaluated with {similarity} similarity a result of '
          f'{np.mean(cross_vals[similarity]["test_score"]):.2%} '
          f'+/-{np.std(cross_vals[similarity]["test_score"]):.2%}')


Model evaluated with levenshtein similarity a result of 48.96% +/-3.95%
Model evaluated with damerau-levenshtein similarity a result of 48.97% +/-3.94%
Model evaluated with jaro similarity a result of 62.34% +/-3.39%
Model evaluated with jaro-winkler similarity a result of 63.81% +/-3.39%
```

On transforme en tableau latex pour insertion dans le rapport.

```
[28]: result_strings = dict()

for similarity in similarity_kinds:
    result_strings[similarity] = {'train/test set': sim_dict[similarity],
                                  'cross validation': f'{np.mean(cross_vals[similarity]["test_score"]):.2%} '
                                                     f'+/-{np.std(cross_vals[similarity]["test_score"]):.2%}'}

result_df = pd.DataFrame(result_strings).T
print(result_strings)
labs = {'levenshtein': 'Levenshtein',
        'damerau-levenshtein': 'Damerau-Levenshtein',
        'jaro': 'Jaro',
        'jaro-winkler': 'Jaro-Winkler',
       }
# (result_df.rename(labs)
#      .to_latex(Path('..') / 'tbls' / 'similarities_result.tex',
#      column_format='lcc',
#      bold_rows=True,
```


)

```
{'levenshtein': {'train/test set': '47.88%', 'cross validation': '48.96% +/-3.95%'}, 'damerau-levenshtein': {'train/test set': '47.88%', 'cross validation': '48.97% +/-3.94%'}, 'jaro': {'train/test set': '63.30%', 'cross validation': '62.34% +/-3.39%'}, 'jaro-winkler': {'train/test set': '65.28%', 'cross validation': '63.81% +/-3.39%'}}
```

1.5.2 Illustration

On illustre les différents niveaux de similarité sur le set de test après entraînement sur le set d'entraînement.

```
[29]: # building the dataframe
y_pred = model.predict(test['blocks']).rename('predicted')
comp_df = pd.concat([test['ingredients'].fillna(''), y_pred], axis=1)
processed_df = comp_df.applymap(build_text_processor())

# computing similarities and ranks
sim_funcs = {sim: partial(text_similarity, similarity=sim) for sim in similarity_kinds}
for sim in similarity_kinds:
    processed_df[sim] = processed_df.apply(lambda x: sim_funcs[sim](x['ingredients'], x['predicted']), axis=1)
    processed_df[sim + '_rank'] = processed_df[sim].rank(axis=0, method='first', ascending=False)
```

```
[31]: (processed_df.join(comp_df, lsuffix='_')
      .sort_values(['levenshtein'], ascending=False)
      .loc[(processed_df['ingredients'] != '') & (processed_df['predicted'] != '')]
) .sample(4)
```

```
[32]: (
    processed_df.sort_values('levenshtein_rank')
        .loc[(processed_df['ingredients'] != '') &
              (processed_df['predicted'] != '') &
              (processed_df['predicted'].apply(len) <=300)]
        .join(comp_df, lsuffix='_')
        .iloc[np.r_[0:3, 47:50, -3:0]]
)
```

```
[33]: # outputting to latex
with pd.option_context("max_colwidth", 100000):
    tex_str = (processed_df.sort_values('levenshtein_rank')
                .loc[(processed_df['ingredients'] != '') &
                      (processed_df['predicted'] != '') &
                      (processed_df['predicted'].apply(len) <=300)]
                .join(comp_df, lsuffix='_')
                .iloc[np.r_[0:3, 47:50, -3:0]]
                .to_latex(columns=['ingredients', 'predicted', 'levenshtein', # 'levenshtein_rank',
                                   'damerau-levenshtein', 'jaro',
                                   'jaro-winkler'],
                           index=False,
                           index_names=False,
                           column_format='p[5cm]p[5cm]cccc',
                           formatters={'levenshtein': lambda x: f'{x:.2%}',
                                       'levenshtein_rank': lambda x: f'{x:.1Of}' ,
                                       'damerau-levenshtein': lambda x: f'{x:.2%}' ,
                                       #'damerau-levenshtein_rank': lambda x: f'{x:.1Of}' ,
                                       'jaro': lambda x: f'{x:.2%}' ,
                                       #'jaro_rank': lambda x: f'{x:.1Of}' ,
                                       'jaro-winkler': lambda x: f'{x:.2%}' ,
                                       #'jaro-winkler_rank': lambda x: f'{x:.1Of}' ,
                                       },
                           header=["Listes d'ingrédients cibles", "Listes d'ingrédients prédictes",
                                   'Lev', 'Dam-Lev', 'Jaro', 'Jaro-Win'],
                           na_rep = '<rien>',
                           )
                .replace(r'\textbackslash n', r' \newline ').replace(r'\\', r'\\ \\hline')

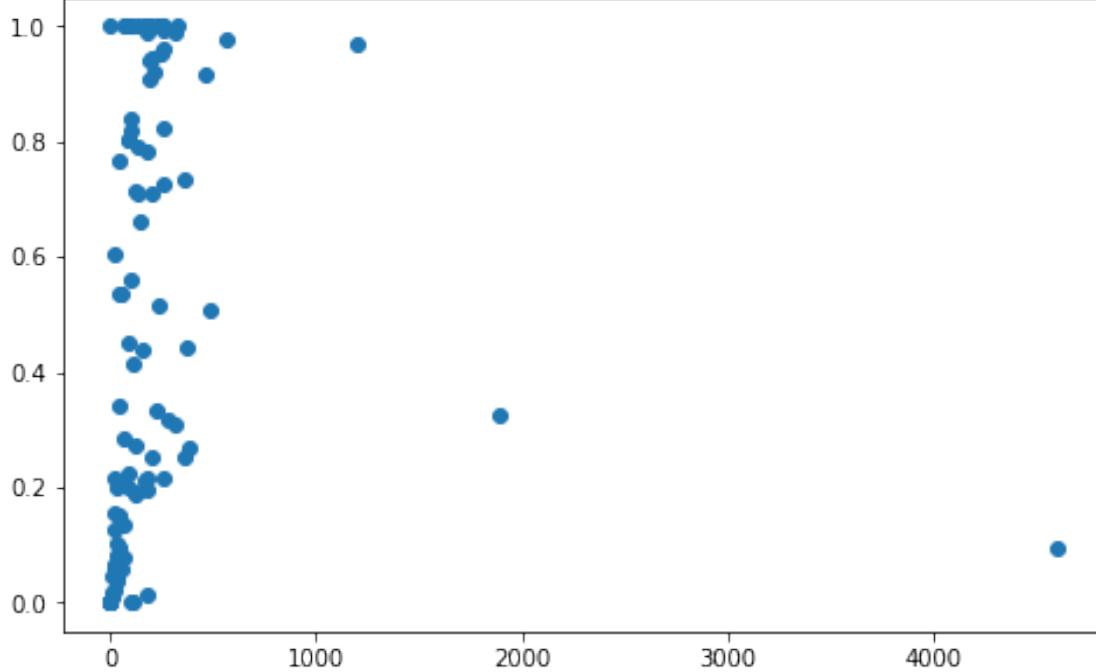
# with open(Path('..') / 'tbls' / 'similarity_illustration.tex', 'w') as file:
```

1.6 Similarité vs. longueur des listes d'ingrédients

On compare le score de similarité obtenu par rapport à la longueur des listes d'ingrédients cibles.

```
[34]: fig, ax = plt.subplots(figsize=(8, 5))
ax.scatter(y=processed_df['levenshtein'], x=processed_df['ingredients'].apply(len))

[34]: <matplotlib.collections.PathCollection at 0x7fa4963cfac0>
```



On a des outliers qui vont avoir trop d'importance sur les résultats. On va les filtrer.

```
[35]: filtered = processed_df.loc[processed_df['ingredients'].apply(len) <= 350]

[36]: print('r2 : ', linregress(x=filtered['ingredients'].apply(len), y=filtered['levenshtein']).rvalue ** 2)
linregress(x=filtered['ingredients'].apply(len), y=filtered['levenshtein'])

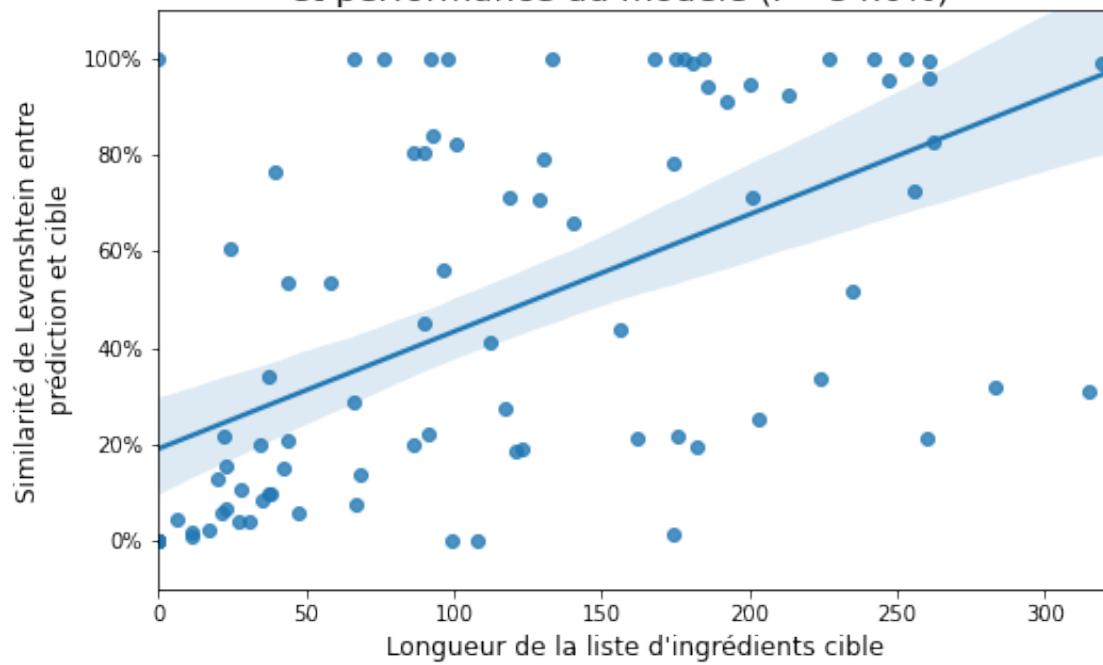
r2 :  0.3249291855359084
```

```
[36]: LinregressResult(slope=0.002425601437356884, intercept=0.19097219058918768, rvalue=0.5700256007723762,
pvalue=4.512591966582692e-09, stderr=0.0003726991570207304)
```

```
[37]: fig, ax = plt.subplots(figsize=(8, 5))
sns.regplot(x=filtered['ingredients'].apply(len), y=filtered['levenshtein'], ax=ax)
ax.set_xlabel("Longueur de la liste d'ingrédients cible", fontsize=12)
ax.set_ylabel("Similarité de Levenshtein entre\nprédiction et cible", fontsize=12)
fig.suptitle("Relation entre longueur de la liste d'ingrédients\net performance du modèle ( $r^2=34.0\%$ )", fontsize=16)
ax.yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
ax.set_xlim(-0.1, 1.1)
# fig.savefig(Path('..') / 'img' / 'perf_vs_length.png', bbox_inches='tight')

[37]: (-0.1, 1.1)
```

Relation entre longueur de la liste d'ingrédients
et performance du modèle ($r^2=34.0\%$)



model_tuning

Pierre MASSÉ

June 11, 2020

1 Tuning du modèle

L'objet de ce notebook est d'illustrer les différentes étapes de tuning du modèle.

1.1 Préambule

1.1.1 Imports

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[2]: # imports and customization of display
# import os
import re
from functools import partial
from itertools import product
import numpy as np
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
# from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
# from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from matplotlib import pyplot as plt
import matplotlib.patches as mpatch
import matplotlib.ticker as mtick
import seaborn as sns

from src.pimest import ContentGetter
from src.pimest import PathGetter
from src.pimest import PDFContentParser
from src.pimest import BlockSplitter
from src.pimest import SimilaritySelector
from src.pimest import custom_accuracy
from src.pimest import text_sim_score
from src.pimest import text_similarity
# from src.pimest import build_text_processor
```

```
[3]: # monkeypatch _repr_latex_ for better inclusion of dataframes output in report
def _repr_latex_(self, size='scriptsize'):
    return(f"\\"\\resizebox{{\\linewidth}}{{!}}{{\\begin{{\\size}}}{\\centering{{\\self.to_latex()}}}{\\end{{\\size}}}}\"")
pd.DataFrame._repr_latex_ = _repr_latex_
```

1.1.2 Acquisition des données

On récupère les données manuellement étiquetées et on les intègre dans un dataframe

```
[5]: ground_truth_df = pd.read_csv(Path('..') / '..' / 'ground_truth' / 'manually_labelled_ground_truth.csv',
                                    sep=';',
                                    encoding='latin-1',
                                    index_col='uid')
ground_truth_uids = list(ground_truth_df.index)
```

```

acqui_pipe = Pipeline([
    ('PathGetter', PathGetter(ground_truth_uids=ground_truth_uids,
                                train_set_path=Path('..') / '..' / 'ground_truth',
                                ground_truth_path=Path('..') / '..' / 'ground_truth',
                                )),
    ('ContentGetter', ContentGetter(missing_file='to_nan')),
    ('ContentParser', PDFContentParser(None_content='to_empty')),
], verbose=True)

texts_df = acqui_pipe.fit_transform(ground_truth_df)
texts_df['ingredients'] = texts_df['ingredients'].fillna('')
texts_df.sample(3)

```

```
[Pipeline] ... (step 1 of 3) Processing PathGetter, total= 0.1s
[Pipeline] ... (step 2 of 3) Processing ContentGetter, total= 0.1s
Launching 8 processes.
[Pipeline] ... (step 3 of 3) Processing ContentParser, total= 35.9s
```

1.1.3 Train / Test split

On va appliquer une grid search pour déterminer les meilleurs paramètres de notre modèle. Pour ne pas surestimer la performance du modèle, il est nécessaire de bien séparer le jeu de test du jeu d'entraînement, y compris pour la grid search !

```
[6]: train, test = train_test_split(texts_df, test_size=100, random_state=42)
```

Dans toute la suite, on utilisera le jeu d'entraînement pour effectuer le tuning des hyperparamètres

1.2 Ajustement de la fonction de découpage des textes

L'objectif de cette partie est d'optimiser la fonction de découpage des textes en blocs. On va tester quelques fonctions candidates, via une GridSearch.

1.2.1 Définition des fonctions candidates

On définit les fonctions de split :

```
[7]: # definitions of splitter funcs
splitter_funcs = []
def split_func1(text):
    return(text.split('\n\n'))
splitter_funcs.append(split_func1)
def split_func2(text):
    return(text.split('\n'))
splitter_funcs.append(split_func2)
def split_func3(text):
    regex = r'\s*\n\s*\n\s*'
    return(re.split(regex, text))
splitter_funcs.append(split_func3)
```

1.2.2 Mise en place du pipeline

On construit ensuite un pipeline de traitement du texte. Le `SimilaritySelector` prenant en entrée une `pandas.Series`, on définit entre le `BlockSplitter` (dont la méthode `transform()` retourne un `pandas.DataFrame`) et le `SimilaritySelector` une fonction utilitaire qui sélectionne la colonne 'blocks'.

```
[8]: def select_col(df, col_name='blocks'):
        return(df[col_name].fillna(''))
col_selector = FunctionTransformer(select_col)
```

```
[9]: process_pipe = Pipeline([('Splitter', BlockSplitter()),  
                           ('ColumnSelector', col_selector),  
                           ('SimilaritySelector', SimilaritySelector())  
                         ],  
                          verbose=False)
```

On peut tester le fonctionnement de ce Pipeline. Attention, les résultats ne sont pas représentatifs, on entraîne et on prédit sur le même jeu de données !

```
[10]: process_pipe.fit(train, train['ingredients'])
process_pipe.predict(train).sample(4)
```

```

Launching 8 processes.
Launching 8 processes.

[10]: uid
5cb7f05a-3b2c-440e-af0d-01843fb38cbf      INGRÉDIENTS : Sucre, Gomme base, Sirop de gluc...
7e5ff57f-5a13-46bf-bb1c-b59b40727515      MORCEAUX DE THON AU NATUREL
8266604c-1ea2-47ca-a4fa-649b4147e733

7528ded6-bec2-418a-b0be-5d06387b2f88     Arômes et couleurs plus ou moins prononcés sui...
dtype: object

```

1.2.3 Helper fonction

On doit faire varier dans la grid search des paramètres qui sont packés sous forme de dictionnaires avant d'être passés au SimilaritySelector. On construit une fonction qui permet de construire le produit cartésien qui va bien pour ces paramètres.

```

[11]: def prod_params(dict_to_prod):
    """
    In : dict of dicts.
    First level key : parameter name
    Second level key : name of scenario with this parameter value
    Values : parameter value

    Returns a tuple:
        - list of labels to name scenario
        - list of dictionaries to pass to count_vect_kwarg
    """
    label_lists = [list(dict_.keys()) for dict_ in dict_to_prod.values()]
    labels = list(map(lambda x: ', '.join(x), list(product(*label_lists))))
    values_iter = list(product(*[list(dict_.values()) for dict_ in dict_to_prod.values()]))
    parms_names = list(dict_to_prod.keys())
    dict_out = [{key: val for (key, val) in zip(parms_names, values_)} for values_ in values_iter]
    return(labels, dict_out)

```

```
[12]: prod_params({'stop_words': {'no_stopwords removal': None, 'with_stopwords removal' : {'de', 'le'}},
                 'ngram_ranges': {'no_ngram': (1, 1), 'bigrams': (1, 2)}})
```

```
[12]: ([['no stopwords removal, no_ngram',
       'no stopwords removal, bigrams',
       'with stopwords removal, no_ngram',
       'with stopwords removal, bigrams'],
      [{'stop_words': None, 'ngram_ranges': (1, 1)},
       {'stop_words': None, 'ngram_ranges': (1, 2)},
       {'stop_words': {'de', 'le'}, 'ngram_ranges': (1, 1)},
       {'stop_words': {'de', 'le'}, 'ngram_ranges': (1, 2)}]])
```

1.2.4 Stockage des résultats dans un dataframe

Au fil des lancements des grid search, on stockera les données dans un dataframe afin de pouvoir les analyser plus simplement après coup.

```
[14]: result_df = pd.DataFrame()
```

1.2.5 Application de la GridSearch : tuning du text preprocessing (run 1)

On applique ensuite une grid search en faisant varier les fonctions de text preprocessing : - fonction de split du texte des documents en blocs - retrait ou non de stopwords - prise en compte de ngrams - juste pour une première comparaison, choix du candidat par projection 11/12 ou par similarité cosinus

On scorera via la similarité de Levenshtein.

```

[15]: lev_scorer = partial(text_sim_score, similarity='levenshtein')

[16]: stop_words = {'pas', 'le', 'en', 'pour', 'ou', 'ce', 'de', 'dans', 'du', 'and', 'un', 'sur', 'et',
                  'of', 'est', 'par', 'la', 'les', 'dont', 'au', 'des', 'que'}

[17]: ngram_ranges = {'no_ngram': (1, 1), 'bigrams': (1, 2), 'trigrams': (1, 3)}

[18]: kwargs_to_prod = prod_params({'stop_words': {'no_stopwords removal': None, 'with_stopwords removal' : stop_words},
                                   'ngram_range': ngram_ranges,
                                   'strip_accents': {'keep accents': None, 'remove accents': 'unicode'}
                                 })
```

```
[19]: param_grid = [{'Splitter__splitter_func': splitter_funcs,
                  ''SimilaritySelector__similarity': ['projection', 'cosine'],
                  ''SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
                }]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                    ).fit(train, train['ingredients'])
```

Fitting 8 folds for each of 72 candidates, totalling 576 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

[Parallel(n_jobs=-1)]: Done 34 tasks | elapsed: 13.4s
 [Parallel(n_jobs=-1)]: Done 184 tasks | elapsed: 1.2min
 [Parallel(n_jobs=-1)]: Done 434 tasks | elapsed: 2.9min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 576 out of 576 | elapsed: 3.9min finished

```
[20]: labels = list(product(kwargs_to_prod[0], ['Projection 12/11', 'Cosinus'], ['Split 1', 'Split 2', 'Split 3']))
labels = list(map(lambda x: ', '.join(x), labels))
```

```
[21]: for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

no stopwords removal, no_ngram, keep accents, Projection 12/11, Split 1 50.15% +/- 5.61%
 no stopwords removal, no_ngram, keep accents, Projection 12/11, Split 2 38.90% +/- 4.52%
 no stopwords removal, no_ngram, keep accents, Projection 12/11, Split 3 52.65% +/- 6.09%
 no stopwords removal, no_ngram, keep accents, Cosinus, Split 1 40.30% +/- 4.91%
 no stopwords removal, no_ngram, keep accents, Cosinus, Split 2 25.87% +/- 2.25%
 no stopwords removal, no_ngram, keep accents, Cosinus, Split 3 41.98% +/- 5.27%
 no stopwords removal, no_ngram, remove accents, Projection 12/11, Split 1 49.47% +/- 5.77%
 no stopwords removal, no_ngram, remove accents, Projection 12/11, Split 2 38.56% +/- 4.19%
 no stopwords removal, no_ngram, remove accents, Projection 12/11, Split 3 52.02% +/- 6.05%
 no stopwords removal, no_ngram, remove accents, Cosinus, Split 1 40.93% +/- 5.02%
 no stopwords removal, no_ngram, remove accents, Cosinus, Split 2 26.06% +/- 2.00%
 no stopwords removal, no_ngram, remove accents, Cosinus, Split 3 42.68% +/- 5.42%
 no stopwords removal, bigrams, keep accents, Projection 12/11, Split 1 55.47% +/- 5.22%
 no stopwords removal, bigrams, keep accents, Projection 12/11, Split 2 43.12% +/- 2.32%
 no stopwords removal, bigrams, keep accents, Projection 12/11, Split 3 58.38% +/- 5.06%
 no stopwords removal, bigrams, keep accents, Cosinus, Split 1 41.53% +/- 5.73%
 no stopwords removal, bigrams, keep accents, Cosinus, Split 2 26.94% +/- 2.14%
 no stopwords removal, bigrams, keep accents, Cosinus, Split 3 43.45% +/- 6.40%
 no stopwords removal, bigrams, remove accents, Projection 12/11, Split 1 55.40% +/- 5.11%
 no stopwords removal, bigrams, remove accents, Projection 12/11, Split 2 43.27% +/- 2.78%
 no stopwords removal, bigrams, remove accents, Projection 12/11, Split 3 58.31% +/- 5.23%
 no stopwords removal, bigrams, remove accents, Cosinus, Split 1 41.49% +/- 5.88%
 no stopwords removal, bigrams, remove accents, Cosinus, Split 2 27.27% +/- 1.95%
 no stopwords removal, bigrams, remove accents, Cosinus, Split 3 43.70% +/- 6.48%
 no stopwords removal, trigrams, keep accents, Projection 12/11, Split 1 56.41% +/- 5.05%
 no stopwords removal, trigrams, keep accents, Projection 12/11, Split 2 43.75% +/- 3.06%
 no stopwords removal, trigrams, keep accents, Projection 12/11, Split 3 59.56% +/- 5.12%
 no stopwords removal, trigrams, keep accents, Cosinus, Split 1 41.54% +/- 5.25%
 no stopwords removal, trigrams, keep accents, Cosinus, Split 2 26.95% +/- 2.41%
 no stopwords removal, trigrams, keep accents, Cosinus, Split 3 43.54% +/- 6.16%
 no stopwords removal, trigrams, remove accents, Projection 12/11, Split 1 56.43% +/- 5.07%
 no stopwords removal, trigrams, remove accents, Projection 12/11, Split 2 43.83% +/- 3.11%
 no stopwords removal, trigrams, remove accents, Projection 12/11, Split 3 59.58% +/- 5.11%
 no stopwords removal, trigrams, remove accents, Cosinus, Split 1 42.08% +/- 5.14%
 no stopwords removal, trigrams, remove accents, Cosinus, Split 2 27.28% +/- 2.39%
 no stopwords removal, trigrams, remove accents, Cosinus, Split 3 44.18% +/- 6.17%
 with stopwords removal, no_ngram, keep accents, Projection 12/11, Split 1 54.94% +/- 5.62%
 with stopwords removal, no_ngram, keep accents, Projection 12/11, Split 2 42.12% +/- 4.32%
 with stopwords removal, no_ngram, keep accents, Projection 12/11, Split 3 58.06% +/- 6.52%
 with stopwords removal, no_ngram, keep accents, Cosinus, Split 1 51.06% +/- 6.89%
 with stopwords removal, no_ngram, keep accents, Cosinus, Split 2 29.73% +/- 4.76%
 with stopwords removal, no_ngram, keep accents, Cosinus, Split 3 53.35% +/- 7.12%
 with stopwords removal, no_ngram, remove accents, Projection 12/11, Split 1 54.89% +/- 5.81%
 with stopwords removal, no_ngram, remove accents, Projection 12/11, Split 2 42.22% +/- 4.32%
 with stopwords removal, no_ngram, remove accents, Projection 12/11, Split 3 57.99% +/- 6.66%
 with stopwords removal, no_ngram, remove accents, Cosinus, Split 1 51.40% +/- 6.72%

```

with stopwords removal, no_ngram, remove accents, Cosinus, Split 2 30.44% +/- 4.69%
with stopwords removal, no_ngram, remove accents, Cosinus, Split 3 53.69% +/- 7.13%
with stopwords removal, bigrams, keep accents, Projection 12/11, Split 1 57.74% +/- 5.74%
with stopwords removal, bigrams, keep accents, Projection 12/11, Split 2 44.54% +/- 3.57%
with stopwords removal, bigrams, keep accents, Projection 12/11, Split 3 61.08% +/- 5.91%
with stopwords removal, bigrams, keep accents, Cosinus, Split 1 52.31% +/- 7.10%
with stopwords removal, bigrams, keep accents, Cosinus, Split 2 26.86% +/- 4.47%
with stopwords removal, bigrams, keep accents, Cosinus, Split 3 54.85% +/- 7.51%
with stopwords removal, bigrams, remove accents, Projection 12/11, Split 1 57.99% +/- 5.69%
with stopwords removal, bigrams, remove accents, Projection 12/11, Split 2 44.43% +/- 3.30%
with stopwords removal, bigrams, remove accents, Projection 12/11, Split 3 60.86% +/- 5.65%
with stopwords removal, bigrams, remove accents, Cosinus, Split 1 51.79% +/- 6.83%
with stopwords removal, bigrams, remove accents, Cosinus, Split 2 26.90% +/- 4.71%
with stopwords removal, bigrams, remove accents, Cosinus, Split 3 54.43% +/- 7.18%
with stopwords removal, trigrams, keep accents, Projection 12/11, Split 1 58.76% +/- 5.47%
with stopwords removal, trigrams, keep accents, Projection 12/11, Split 2 45.36% +/- 3.58%
with stopwords removal, trigrams, keep accents, Projection 12/11, Split 3 61.94% +/- 5.59%
with stopwords removal, trigrams, keep accents, Cosinus, Split 1 51.65% +/- 6.90%
with stopwords removal, trigrams, keep accents, Cosinus, Split 2 25.74% +/- 4.50%
with stopwords removal, trigrams, keep accents, Cosinus, Split 3 54.28% +/- 7.24%
with stopwords removal, trigrams, remove accents, Projection 12/11, Split 1 58.82% +/- 5.48%
with stopwords removal, trigrams, remove accents, Projection 12/11, Split 2 45.54% +/- 3.61%
with stopwords removal, trigrams, remove accents, Projection 12/11, Split 3 62.01% +/- 5.61%
with stopwords removal, trigrams, remove accents, Cosinus, Split 1 51.54% +/- 6.74%
with stopwords removal, trigrams, remove accents, Cosinus, Split 2 26.04% +/- 4.42%
with stopwords removal, trigrams, remove accents, Cosinus, Split 3 54.26% +/- 7.21%

```

On tire de ce premier test : - que le modèle est bien plus performant avec le retrait des stopwords - que le split le plus efficace est la fonction qui applique la regex (deux retours chariots parmi des whitespaces) - split 3 - que la prise en compte de bigrammes améliore, avec les trigrammes en plus on ne gagne rien - que la similarité cosinus semble sensiblement moins performante que le choix par projection (12/11)

Remarque : la standard dev est quand même assez élevée (de l'ordre de 5-6%). Les scénarios avec peu d'écart entre leurs moyennes (2-3%) ne sont pas départageables via cette grid search.

On sauvegarde les résultat dans le dataframe qu'on analysera à la fin

```
[22]: try:
    result_df = result_df.loc[result_df['run'] != 1].copy()
except:
    pass
result_df = pd.DataFrame(search.cv_results_)
result_df['run'] = 1
```

1.2.6 Application de la Grid Search : tuning du calcul de similarité (run 2)

On va maintenant déterminer, sur la base des paramètres déjà retenus, le mode de calcul de similarité le plus performant. Seul le calcul par projection est paramétrique (norme dans l'espace de départ vs. norme sur l'espace projeté), on fera uniquement varier ces paramètres (en plus de la comparaison avec la similarité cosinus).

On comparera également la performance du modèle selon qu'on vectorise les textes via les comptes de mots, ou bien seulement via un identifiant binaire (présence ou absence du mot).

```
[23]: process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                               })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words,
                                              'ngram_range': {'bigrams': (1, 2)},
                                              'binary': {'counts': False, 'binary flag': True},
                                              'strip_accents': {'remove accents': 'unicode'}
                                             })
param_grid = [{"SimilaritySelector__source_norm": ['l1'],
               "SimilaritySelector__projected_norm": ['l1'],
               "SimilaritySelector__similarity": ['projection'],
               "SimilaritySelector__count_vect_kwparams": kwargs_to_prod[1],
             },
             {"SimilaritySelector__source_norm": ['l2'],
               "SimilaritySelector__projected_norm": ['l2'],
               "SimilaritySelector__similarity": ['projection'],
               "SimilaritySelector__count_vect_kwparams": kwargs_to_prod[1],
             },
             {"SimilaritySelector__source_norm": ['l2'],
               "SimilaritySelector__projected_norm": ['l1'],
               "SimilaritySelector__similarity": ['projection'],
               "SimilaritySelector__count_vect_kwparams": kwargs_to_prod[1],
             }]
```



```

        'SimilaritySelector__projected_norm': ['l15'],
        'SimilaritySelector__similarity': ['projection'],
        'SimilaritySelector__count_vect_kw_args': kw_args_to_prod[1],
    },
    {
        'SimilaritySelector__similarity': ['cosine'],
        'SimilaritySelector__count_vect_kw_args': kw_args_to_prod[1],
    }
]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
).fit(train, train['ingredients'])

```

Fitting 8 folds for each of 36 candidates, totalling 288 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:  14.9s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.2min
```

Launching 8 processes.

```
[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed:  1.8min finished
```

```
[24]: labels = ['l1, l1',
              'l2, l2',
              'l2, l1',
              'l3, l2',
              'l4, l3',
              'l5, l4',
              'l6, l5',
              'l3, l1',
              'l4, l2',
              'l5, l3',
              'l6, l4',
              'l7, l5',
              'l4, l1',
              'l5, l2',
              'l6, l3',
              'l7, l4',
              'l8, l5',
              'cosine',
            ]
labels = list(product(labels, kw_args_to_prod[0]))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"[{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.cv_results_['std_test_similarity'][i]:.2%}]"
    print(labels[i], str_result)
```

```
l1, l1, with stopwords removal, bigrams, counts, remove accents 17.87% +/- 2.52%
l1, l1, with stopwords removal, bigrams, binary flag, remove accents 17.90% +/- 2.55%
l2, l2, with stopwords removal, bigrams, counts, remove accents 17.87% +/- 2.52%
l2, l2, with stopwords removal, bigrams, binary flag, remove accents 17.90% +/- 2.55%
l2, l1, with stopwords removal, bigrams, counts, remove accents 60.86% +/- 5.65%
l2, l1, with stopwords removal, bigrams, binary flag, remove accents 61.00% +/- 5.61%
l3, l2, with stopwords removal, bigrams, counts, remove accents 61.55% +/- 5.25%
l3, l2, with stopwords removal, bigrams, binary flag, remove accents 62.05% +/- 4.73%
l4, l3, with stopwords removal, bigrams, counts, remove accents 59.61% +/- 4.00%
l4, l3, with stopwords removal, bigrams, binary flag, remove accents 62.61% +/- 4.29%
l5, l4, with stopwords removal, bigrams, counts, remove accents 56.23% +/- 3.40%
l5, l4, with stopwords removal, bigrams, binary flag, remove accents 61.82% +/- 3.67%
l6, l5, with stopwords removal, bigrams, counts, remove accents 51.58% +/- 3.78%
l6, l5, with stopwords removal, bigrams, binary flag, remove accents 60.91% +/- 3.62%
l3, l1, with stopwords removal, bigrams, counts, remove accents 59.33% +/- 5.34%
l3, l1, with stopwords removal, bigrams, binary flag, remove accents 59.06% +/- 5.44%
l4, l2, with stopwords removal, bigrams, counts, remove accents 61.11% +/- 5.30%
l4, l2, with stopwords removal, bigrams, binary flag, remove accents 61.00% +/- 5.61%
l5, l3, with stopwords removal, bigrams, counts, remove accents 59.28% +/- 3.60%
l5, l3, with stopwords removal, bigrams, binary flag, remove accents 61.70% +/- 5.21%
l6, l4, with stopwords removal, bigrams, counts, remove accents 55.73% +/- 4.26%
l6, l4, with stopwords removal, bigrams, binary flag, remove accents 62.05% +/- 4.73%
l7, l5, with stopwords removal, bigrams, counts, remove accents 50.18% +/- 2.42%
l7, l5, with stopwords removal, bigrams, binary flag, remove accents 61.96% +/- 4.23%
```

```

14, 11, with stopwords removal, bigrams, counts, remove accents 58.42% +/- 5.53%
14, 11, with stopwords removal, bigrams, binary flag, remove accents 57.44% +/- 5.05%
15, 12, with stopwords removal, bigrams, counts, remove accents 60.29% +/- 4.59%
15, 12, with stopwords removal, bigrams, binary flag, remove accents 59.67% +/- 5.18%
16, 13, with stopwords removal, bigrams, counts, remove accents 58.12% +/- 3.67%
16, 13, with stopwords removal, bigrams, binary flag, remove accents 61.00% +/- 5.61%
17, 14, with stopwords removal, bigrams, counts, remove accents 53.31% +/- 2.96%
17, 14, with stopwords removal, bigrams, binary flag, remove accents 61.50% +/- 5.35%
18, 15, with stopwords removal, bigrams, counts, remove accents 47.87% +/- 2.99%
18, 15, with stopwords removal, bigrams, binary flag, remove accents 61.80% +/- 5.20%
cosine, with stopwords removal, bigrams, counts, remove accents 54.43% +/- 7.18%
cosine, with stopwords removal, bigrams, binary flag, remove accents 53.36% +/- 7.86%

```

On tire de ce second test les conclusions suivantes : - comme lors du premier test, l'identification du meilleur candidat par similarité cosinus est moins performante que par projection - plusieurs configurations de paramètres permettent d'obtenir des performances similaires via la projection : - 12/11 - 12/11b - 13/12 - 13/12b - 13/11b - 14/12 - 14/12b

```
[25]: result_df = result_df.loc[result_df['run'] != 2].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(2)
len(result_df)
```

[25]: 108

1.2.7 Application de la Grid Search : impact des mots non vus en entraînement (run 3)

On va également voir si l'utilisation d'un vectorizer de type HashingVectorizer, qui permet de prendre en compte des mots non vus lors de l'entraînement a un impact sur la performance (ou son écart type, qui est très élevé...).

```
[26]: process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                               })
kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words},
                             'ngram_range': {'bigrams': (1, 2)},
                             'binary': {'counts': False, 'binary flag': True},
                             })
param_grid = [ {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
                'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__source_norm': ['14'],
                'SimilaritySelector__projected_norm': ['12'],
               },
               {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
                'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__source_norm': ['13'],
                'SimilaritySelector__projected_norm': ['12'],
               },
               {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
                'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__source_norm': ['12'],
                'SimilaritySelector__projected_norm': ['11'],
               },
               {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
                'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
                'SimilaritySelector__similarity': ['cosine'],
               },
             ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                      ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 16 candidates, totalling 128 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:   13.7s
Launching 8 processes.
[Parallel(n_jobs=-1)]: Done 128 out of 128 | elapsed:   52.8s finished
```

```
[27]: labels = [
    '14/12',
    '13/12',
    '12/11',
    'cosine'
]

labels = list(product(labels, kwargs_to_prod[0], ['TfidfVectorizer', 'HashingVectorizer']))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"[{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.cv_results_['std_test_similarity'][i]:.2%}]"
    print(labels[i], str_result)
```

14/12, with stopwords removal, bigrams, counts, TfidfVectorizer 61.11% +/- 5.30%
 14/12, with stopwords removal, bigrams, counts, HashingVectorizer 52.46% +/- 3.06%
 14/12, with stopwords removal, bigrams, binary flag, TfidfVectorizer 61.00% +/- 5.61%
 14/12, with stopwords removal, bigrams, binary flag, HashingVectorizer 54.13% +/- 4.12%
 13/12, with stopwords removal, bigrams, counts, TfidfVectorizer 61.55% +/- 5.25%
 13/12, with stopwords removal, bigrams, counts, HashingVectorizer 43.37% +/- 4.64%
 13/12, with stopwords removal, bigrams, binary flag, TfidfVectorizer 62.05% +/- 4.73%
 13/12, with stopwords removal, bigrams, binary flag, HashingVectorizer 47.26% +/- 5.17%
 12/11, with stopwords removal, bigrams, counts, TfidfVectorizer 60.86% +/- 5.65%
 12/11, with stopwords removal, bigrams, counts, HashingVectorizer 57.89% +/- 5.60%
 12/11, with stopwords removal, bigrams, binary flag, TfidfVectorizer 61.00% +/- 5.61%
 12/11, with stopwords removal, bigrams, binary flag, HashingVectorizer 58.26% +/- 5.01%
 cosine, with stopwords removal, bigrams, counts, TfidfVectorizer 54.43% +/- 7.18%
 cosine, with stopwords removal, bigrams, counts, HashingVectorizer 53.01% +/- 6.20%
 cosine, with stopwords removal, bigrams, binary flag, TfidfVectorizer 53.36% +/- 7.86%
 cosine, with stopwords removal, bigrams, binary flag, HashingVectorizer 50.56% +/- 7.20%

L'utilisation d'un HashingVectorizer à la place d'un TfidfVectorizer, pour prendre en compte les mots non vus lors de l'entraînement, n'a pas d'impact positif sur la performance du modèle. Au contraire, elle semble globalement diminuer de quelques points.

```
[28]: result_df = result_df.loc[result_df['run'] != 3].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(3)
len(result_df)
```

[28]: 124

1.2.8 Application d'une grid search : pondération des mots (run 4)

On va en plus appliquer une pondération absolue et relative des mots, dans la recherche de similarité par cosinus.

Les différentes possibilités pour le vecteur cible sont : - moyenne des vecteurs de textes des listes d'ingrédients, avec uniquement un flag binaire (présence / absence du mot) : la cible est la document frequency moyenne des mots des listes d'ingrédients - moyenne des vecteurs de textes des listes d'ingrédients, avec en prenant en compte les comptes des mots dans chacun des textes : la cible est la term frequency moyenne des mots au sein des listes d'ingrédients - moyenne des scores ``absolus'' de chacun des mots au sein des listes d'ingrédients. Il s'agit d'une ``smooth document frequency'' (elle croît logarithmiquement) - moyenne des scores ``relatifs'' de chacun des mots entre liste d'ingrédients et contenu des fiches techniques. Ici on compare la doc frequency entre les deux corpus, pour donner plus de poids aux mots qui sont plus présents dans les listes d'ingrédients que dans le reste du corps du texte.

On comparera à la projection 14/12b, qui porte jusque là les meilleurs résultats.

```
[29]: process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                             'SimilaritySelector__count_vect_type': 'TfidfVectorizer',
                             })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal': stop_words},
                             'ngram_range': {'no_ngrams': (1, 1), 'bigrams': (1, 2)},
                             'binary': {'counts': False, 'binary flag': True},
                             'use_idf': {'without idf': False, 'with idf': True},
                             'strip_accents': {'remove accents': 'unicode'},
                             })

param_grid = [{}
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__scoring': ['default', 'absolute_score', 'relative_score'],
               'SimilaritySelector__similarity': ['cosine'],
               },
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['14'],
               'SimilaritySelector__projected_norm': ['12'],
               }]
```

```

        },
    ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= {'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
).fit(train, train['ingredients'])

```

```

Fitting 8 folds for each of 32 candidates, totalling 256 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   14.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   1.4min
Launching 8 processes.
[Parallel(n_jobs=-1)]: Done 256 out of 256 | elapsed:  1.8min finished

```

```

[30]: labels = [
    'cosine',
]

labels = list(product(labels, kwargs_to_prod[0], ['default', 'absolute score', 'relative score']))
labels.extend(list(product(['projection 14/12'], kwargs_to_prod[0])))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"[{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.cv_results_['std_test_similarity'][i]:.2%}]"
    print(labels[i], str_result)

```

```

cosine, with stopwords removal, no_ngrams, counts, without idf, remove accents, default 53.69% +/- 7.13%
cosine, with stopwords removal, no_ngrams, counts, without idf, remove accents, absolute score 54.47% +/- 6.87%
cosine, with stopwords removal, no_ngrams, counts, without idf, remove accents, relative score 29.42% +/- 5.32%
cosine, with stopwords removal, no_ngrams, counts, with idf, remove accents, default 55.47% +/- 6.70%
cosine, with stopwords removal, no_ngrams, counts, with idf, remove accents, absolute score 52.47% +/- 6.91%
cosine, with stopwords removal, no_ngrams, counts, with idf, remove accents, relative score 32.73% +/- 5.21%
cosine, with stopwords removal, no_ngrams, binary flag, without idf, remove accents, default 53.29% +/- 7.86%
cosine, with stopwords removal, no_ngrams, binary flag, without idf, remove accents, absolute score 54.15% +/- 8.17%
cosine, with stopwords removal, no_ngrams, binary flag, without idf, remove accents, relative score 28.02% +/- 5.00%
cosine, with stopwords removal, no_ngrams, binary flag, with idf, remove accents, default 54.91% +/- 7.31%
cosine, with stopwords removal, no_ngrams, binary flag, with idf, remove accents, absolute score 51.80% +/- 8.13%
cosine, with stopwords removal, no_ngrams, binary flag, with idf, remove accents, relative score 31.39% +/- 5.59%
cosine, with stopwords removal, bigrams, counts, without idf, remove accents, default 54.43% +/- 7.18%
cosine, with stopwords removal, bigrams, counts, without idf, remove accents, absolute score 55.48% +/- 7.11%
cosine, with stopwords removal, bigrams, counts, without idf, remove accents, relative score 33.39% +/- 6.11%
cosine, with stopwords removal, bigrams, counts, with idf, remove accents, default 55.86% +/- 6.78%
cosine, with stopwords removal, bigrams, counts, with idf, remove accents, absolute score 52.00% +/- 6.91%
cosine, with stopwords removal, bigrams, counts, with idf, remove accents, relative score 39.00% +/- 5.38%
cosine, with stopwords removal, bigrams, binary flag, without idf, remove accents, default 53.36% +/- 7.86%
cosine, with stopwords removal, bigrams, binary flag, without idf, remove accents, absolute score 55.36% +/- 7.36%
cosine, with stopwords removal, bigrams, binary flag, without idf, remove accents, relative score 32.74% +/- 5.87%
cosine, with stopwords removal, bigrams, binary flag, with idf, remove accents, default 54.73% +/- 7.39%
cosine, with stopwords removal, bigrams, binary flag, with idf, remove accents, absolute score 51.12% +/- 6.71%
cosine, with stopwords removal, bigrams, binary flag, with idf, remove accents, relative score 39.45% +/- 5.47%
projection 14/12, with stopwords removal, no_ngrams, counts, without idf, remove accents 57.08% +/- 5.38%
projection 14/12, with stopwords removal, no_ngrams, counts, with idf, remove accents 17.38% +/- 1.45%
projection 14/12, with stopwords removal, no_ngrams, binary flag, without idf, remove accents 57.26% +/- 5.89%
projection 14/12, with stopwords removal, no_ngrams, binary flag, with idf, remove accents 22.50% +/- 4.14%
projection 14/12, with stopwords removal, bigrams, counts, without idf, remove accents 61.11% +/- 5.30%
projection 14/12, with stopwords removal, bigrams, counts, with idf, remove accents 32.00% +/- 3.51%

```

```
projection l4/l2, with stopwords removal, bigrams, binary flag, without idf, remove accents 61.00% +/- 5.61%
projection l4/l2, with stopwords removal, bigrams, binary flag, with idf, remove accents 38.00% +/- 5.85%
```

```
[31]: result_df = result_df.loc[result_df['run'] != 4].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(4)
len(result_df)
```

```
[31]: 156
```

On en déduit : - que la similarité par projection reste le mode de détermination du candidat le plus efficace - que dans ce mode, l'utilisation de l'idf dégrade la performance - néanmoins, dans le cadre de la similarité cosinus, l'utilisation de l'idf a un impact positif pour la fonction de scoring par défaut, ou relative.

1.2.9 Application de la grid search : embeddings des mots (run 5)

On mesure l'impact sur la performance de l'utilisation d'embeddings de mots.

```
[32]: process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                             'SimilaritySelector__count_vect_type': 'TfidfVectorizer',
                             })

kargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words,
                                             'ngram_range': {'no_ngram': (1, 1)},
                                             'binary': {'counts': False, 'binary flag': True},
                                             'use_idf': {'without_idf': False, 'with_idf': True},
                                             'strip_accents': {'remove accents': 'unicode'}},
                             })

param_grid = [{}
              'SimilaritySelector__count_vect_kwargs': kargs_to_prod[1],
              'SimilaritySelector__scoring': ['default', 'absolute_score', 'relative_score'],
              'SimilaritySelector__similarity': ['cosine'],
              'SimilaritySelector__embedding_method': [None, 'Word2Vec', 'tSVD'],
              },
              ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                      error_score='raise',
                      ).fit(train, train['ingredients'])
```

Fitting 8 folds for each of 36 candidates, totalling 288 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:   27.6s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  2.3min
```

Launching 8 processes.

```
[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed:  3.7min finished
```

```
[33]: labels = ['default', 'absolute_score', 'relative_score']

labels = list(product(kargs_to_prod[0],
                      [None, 'Word2Vec', 'tSVD'],
                      ['default', 'absolute score', 'relative score'],
                      ))
# labels.extend(list(product(['projection l4/l2'], kargs_to_prod[0])))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"[{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.cv_results_['std_test_similarity'][i]:.2%}]"
    print(labels[i], str_result)
```

```
with stopwords removal, no_ngram, counts, without_idf, remove accents, No embed, default 53.69% +/- 7.13%
with stopwords removal, no_ngram, counts, without_idf, remove accents, No embed, absolute score 54.47% +/- 6.87%
with stopwords removal, no_ngram, counts, without_idf, remove accents, No embed, relative score 29.42% +/- 5.32%
with stopwords removal, no_ngram, counts, without_idf, remove accents, Word2Vec, default 53.70% +/- 6.19%
with stopwords removal, no_ngram, counts, without_idf, remove accents, Word2Vec, absolute score 53.17% +/-
```

```

5.87%
with stopwords removal, no_ngram, counts, without_idf, remove accents, Word2Vec, relative score 10.19% +/- 2.68%
with stopwords removal, no_ngram, counts, without_idf, remove accents, tSVD, default 51.60% +/- 7.44%
with stopwords removal, no_ngram, counts, without_idf, remove accents, tSVD, absolute score 49.99% +/- 7.59%
with stopwords removal, no_ngram, counts, without_idf, remove accents, tSVD, relative score 8.05% +/- 1.59%
with stopwords removal, no_ngram, counts, with_idf, remove accents, No embed, default 55.47% +/- 6.70%
with stopwords removal, no_ngram, counts, with_idf, remove accents, No embed, absolute score 52.47% +/- 6.91%
with stopwords removal, no_ngram, counts, with_idf, remove accents, No embed, relative score 32.73% +/- 5.21%
with stopwords removal, no_ngram, counts, with_idf, remove accents, Word2Vec, default 53.70% +/- 6.06%
with stopwords removal, no_ngram, counts, with_idf, remove accents, Word2Vec, absolute score 53.58% +/- 6.26%
with stopwords removal, no_ngram, counts, with_idf, remove accents, Word2Vec, relative score 10.16% +/- 2.75%
with stopwords removal, no_ngram, counts, with_idf, remove accents, tSVD, default 49.47% +/- 6.39%
with stopwords removal, no_ngram, counts, with_idf, remove accents, tSVD, absolute score 46.09% +/- 6.51%
with stopwords removal, no_ngram, counts, with_idf, remove accents, tSVD, relative score 8.96% +/- 1.99%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, No embed, default 53.29% +/- 7.86%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, No embed, absolute score 54.15% +/- 8.17%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, No embed, relative score 28.02% +/- 5.00%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, Word2Vec, default 53.16% +/- 6.10%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, Word2Vec, absolute score 52.54% +/- 6.31%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, Word2Vec, relative score 10.18% +/- 2.67%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, tSVD, default 53.84% +/- 8.26%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, tSVD, absolute score 54.32% +/- 8.38%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, tSVD, relative score 9.44% +/- 1.98%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, No embed, default 54.91% +/- 7.31%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, No embed, absolute score 51.80% +/- 8.13%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, No embed, relative score 31.39% +/- 5.59%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, Word2Vec, default 53.39% +/- 6.64%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, Word2Vec, absolute score 53.62% +/- 6.58%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, Word2Vec, relative score 10.20% +/- 2.68%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, tSVD, default 50.53% +/- 7.64%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, tSVD, absolute score 49.87% +/- 8.01%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, tSVD, relative score 9.13% +/- 2.54%

```

```
[34]: result_df = result_df.loc[result_df['run'] != 5].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(5)
len(result_df)
```

[34]: 192

1.2.10 Random search : validation finale de l'ensemble des critères (run 6)

On applique enfin une random search, afin de voir si les conclusions qui avaient été tirée lors des explorations systématiques de certains domaines sont viables.

```
[35]: kwargs_to_prod = prod_params({'stop_words': {'with_stopwords_removal': stop_words, 'keep_stopwords': None},
                                    'use_idf': {'with_idf': True, 'no_idf': False},
                                    'binary': {'counts': False, 'binary_flag': True},
                                    'ngram_range': {'no_ngram': (1, 1), 'bigrams': (1, 2), 'trigrams': (1, 3)},
                                    'strip_accents': {'remove_accents': 'unicode', 'keep_accents': None},
                                    })
param_grid = [
    {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
     'SimilaritySelector__scoring': ['default', 'absolute_score', 'relative_score'],
     'SimilaritySelector__similarity': ['cosine'],
     'SimilaritySelector__embedding_method': [None, 'Word2Vec', 'tSVD'],
     },
    {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
     }
```

```

        'SimilaritySelector__scoring': ['default'],
        'SimilaritySelector__similarity': ['projection'],
        'SimilaritySelector__source_norm': ['l2', 'l3', 'l4', 'l5'],
        'SimilaritySelector__projected_norm': ['l1', 'l2', 'l3', 'l4'],
    },
],
len(kwargs_to_prod[1])

search = RandomizedSearchCV(process_pipe,
                            param_grid,
                            n_iter=50,
                            cv=8,
                            scoring=({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                            refit='similarity',
                            n_jobs=-1,
                            verbose=1,
                            ).fit(train, train['ingredients'])

```

Fitting 8 folds for each of 50 candidates, totalling 400 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 184 tasks | elapsed: 2.3min
Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed: 6.1min finished

[36]: search.best_params_

[36]: {'SimilaritySelector__source_norm': 'l4',
 'SimilaritySelector__similarity': 'projection',
 'SimilaritySelector__scoring': 'default',
 'SimilaritySelector__projected_norm': 'l3',
 'SimilaritySelector__count_vect_kwarg': {'stop_words': None,
 'use_idf': False,
 'binary': True,
 'ngram_range': (1, 3),
 'strip_accents': None}}

[37]: search.best_score_

[37]: 0.622970201742938

1.2.11 Tuning des méthodes de vectorisation (run 7)

```

[38]: process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
})

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal': stop_words},
                             'use_idf': {'with idf': True, 'no idf': False},
                             'binary': {'counts': False, 'binary flag': True},
                             'ngram_range': {'no_ngram': (1, 1),
                                             'bigrams': (1, 2),
                                             'trigrams': (1, 3),
                                             'quadgrams': (1, 4),
                                             'quintgrams': (1, 5)},
                             'strip_accents': {'remove accents': 'unicode'},
})

```

```

param_grid = [{"SimilaritySelector__count_vect_kwarg": kwargs_to_prod[1],
               "SimilaritySelector__similarity": ['cosine'],
}, {
    "SimilaritySelector__count_vect_kwarg": kwargs_to_prod[1],
    "SimilaritySelector__similarity": ['projection'],
    "SimilaritySelector__source_norm": ['l4'],
    "SimilaritySelector__projected_norm": ['l3'],
}]

```

```

search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring=({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,

```

```

        verbose=1,
        error_score='raise',
    ).fit(train, train['ingredients'])

Fitting 8 folds for each of 40 candidates, totalling 320 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Launching 8 processes.
[Parallel(n_jobs=-1)]: Done 320 out of 320 | elapsed: 2.5min finished

[39]: labels = list(product(['cosine', 'Proj 14/13'],
                           kargs_to_prod[0],
                           ))
# labels.extend(list(product(['projection l4/l2'], kargs_to_prod[0])))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"[{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.cv_results_['std_test_similarity'][i]:.2%}]"
    print(labels[i], str_result)

cosine, with stopwords removal, with idf, counts, no_ngram, remove accents 55.47% +/- 6.70%
cosine, with stopwords removal, with idf, counts, bigrams, remove accents 55.86% +/- 6.78%
cosine, with stopwords removal, with idf, counts, trigrams, remove accents 55.01% +/- 6.50%
cosine, with stopwords removal, with idf, counts, quadgrams, remove accents 55.44% +/- 6.40%
cosine, with stopwords removal, with idf, counts, quintgrams, remove accents 55.62% +/- 6.47%
cosine, with stopwords removal, with idf, binary flag, no_ngram, remove accents 54.91% +/- 7.31%
cosine, with stopwords removal, with idf, binary flag, bigrams, remove accents 54.73% +/- 7.39%
cosine, with stopwords removal, with idf, binary flag, trigrams, remove accents 54.68% +/- 7.98%
cosine, with stopwords removal, with idf, binary flag, quadgrams, remove accents 54.23% +/- 7.59%
cosine, with stopwords removal, with idf, binary flag, quintgrams, remove accents 54.10% +/- 8.03%
cosine, with stopwords removal, no idf, counts, no_ngram, remove accents 53.69% +/- 7.13%
cosine, with stopwords removal, no idf, counts, bigrams, remove accents 54.43% +/- 7.18%
cosine, with stopwords removal, no idf, counts, trigrams, remove accents 54.26% +/- 7.21%
cosine, with stopwords removal, no idf, counts, quadgrams, remove accents 54.12% +/- 7.28%
cosine, with stopwords removal, no idf, counts, quintgrams, remove accents 54.29% +/- 7.21%
cosine, with stopwords removal, no idf, binary flag, no_ngram, remove accents 53.29% +/- 7.86%
cosine, with stopwords removal, no idf, binary flag, bigrams, remove accents 53.36% +/- 7.86%
cosine, with stopwords removal, no idf, binary flag, trigrams, remove accents 52.39% +/- 7.54%
cosine, with stopwords removal, no idf, binary flag, quadgrams, remove accents 51.73% +/- 7.89%
cosine, with stopwords removal, no idf, binary flag, quintgrams, remove accents 51.04% +/- 8.52%
Proj 14/13, with stopwords removal, with idf, counts, no_ngram, remove accents 10.23% +/- 1.89%
Proj 14/13, with stopwords removal, with idf, counts, bigrams, remove accents 9.05% +/- 2.18%
Proj 14/13, with stopwords removal, with idf, counts, trigrams, remove accents 8.95% +/- 2.33%
Proj 14/13, with stopwords removal, with idf, counts, quadgrams, remove accents 8.97% +/- 2.30%
Proj 14/13, with stopwords removal, with idf, counts, quintgrams, remove accents 8.97% +/- 2.30%
Proj 14/13, with stopwords removal, with idf, binary flag, no_ngram, remove accents 9.50% +/- 1.73%
Proj 14/13, with stopwords removal, with idf, binary flag, bigrams, remove accents 9.28% +/- 2.34%
Proj 14/13, with stopwords removal, with idf, binary flag, trigrams, remove accents 9.49% +/- 2.21%
Proj 14/13, with stopwords removal, with idf, binary flag, quadgrams, remove accents 9.65% +/- 2.34%
Proj 14/13, with stopwords removal, with idf, binary flag, quintgrams, remove accents 9.65% +/- 2.34%
Proj 14/13, with stopwords removal, no idf, counts, no_ngram, remove accents 56.36% +/- 4.14%
Proj 14/13, with stopwords removal, no idf, counts, bigrams, remove accents 59.61% +/- 4.00%
Proj 14/13, with stopwords removal, no idf, counts, trigrams, remove accents 60.39% +/- 3.35%
Proj 14/13, with stopwords removal, no idf, counts, quadgrams, remove accents 60.72% +/- 3.33%
Proj 14/13, with stopwords removal, no idf, counts, quintgrams, remove accents 60.87% +/- 3.19%
Proj 14/13, with stopwords removal, no idf, binary flag, no_ngram, remove accents 59.16% +/- 6.49%
Proj 14/13, with stopwords removal, no idf, binary flag, bigrams, remove accents 62.61% +/- 4.29%
Proj 14/13, with stopwords removal, no idf, binary flag, trigrams, remove accents 63.31% +/- 3.83%
Proj 14/13, with stopwords removal, no idf, binary flag, quadgrams, remove accents 63.21% +/- 3.84%
Proj 14/13, with stopwords removal, no idf, binary flag, quintgrams, remove accents 63.21% +/- 3.83%

[40]: result_df = result_df.loc[result_df['run'] != 7].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(7)
len(result_df)

```

[40]: 232

1.2.12 Dépouillement des résultats

Préparation du dataframe On va maintenant interpréter le contenu du dataframe portant les résultats. On commence par renommer les colonnes qui ont de longs noms.

```
[41]: col_rename = {
    'param_SimilaritySelector__similarity': 'similarity',
    'param_Splitter__splitter_func': 'split_func',
    'param_SimilaritySelector__projected_norm': 'projected_norm',
    'param_SimilaritySelector__source_norm': 'source_norm',
    'param_SimilaritySelector__count_vect_type': 'count_vect_type',
    'param_SimilaritySelector__scoring': 'scoring',
    'param_SimilaritySelector__embedding_method': 'embedding_method',
}
result_df.rename(col_rename, axis=1, inplace=True)
```

On récupère maintenant le contenu des dictionnaires en tant que colonnes.

```
[42]: dict_cols = {'param_SimilaritySelector__count_vect_kwargs'}
to_concat = list()
for dict_col in dict_cols:
    to_concat.append(result_df[dict_col].apply(pd.Series))
    result_df.drop(dict_col, axis=1, inplace=True)
result_df = pd.concat([result_df, *to_concat], axis=1)
```

```
[43]: result_df.sample(3)
```

```
[43]:
```



On effectue ensuite quelques prétraitements pour améliorer la lisibilité. On renomme les fonctions avec leur nom, et on applique une valeur par défaut.

```
[44]: def rename_funcs(func):
    try:
        return(func.__name__)
    except:
        return('split_func3')
result_df['split_func'] = result_df.loc[:, 'split_func'].apply(rename_funcs)
```

On met 2 critères pour le retrait des stopwords : retrait d'une liste, ou non retrait.

```
[45]: result_df['stop_words'].fillna('no_stopword_removal', inplace=True)
result_df.loc[result_df['stop_words'] != 'no_stopword_removal', 'stop_words'] = 'stop_word_list'
```

On renomme les ngram_ranges

```
[46]: ngram_dict = {(1, 1): 'no_ngram',
                  (1, 2): 'bigrams',
                  (1, 3): 'trigrams',
                  (1, 4): 'quadgrams',
                  (1, 5): 'quintgrams',
                  }
result_df['ngram_range'] = result_df['ngram_range'].map(ngram_dict, na_action='ignore')
```

On renomme le retrait des accents.

```
[47]: accent_dict = {None: 'no_accent_removal',
                  'unicode': 'accent_removal',
                  }
result_df['strip_accents'] = result_df['strip_accents'].map(accent_dict, na_action='ignore')
```

```
[48]: result_df.drop('params', axis=1, inplace=True)
```

On renomme les embeddings

```
[49]: embed_dict = {None: 'no_embedding',
                  'Word2Vec': 'Word2Vec',
                  'tSVD': 'tSVD',
                  }
result_df['embedding_method'] = result_df['embedding_method'].map(embed_dict, na_action='ignore')
```

On applique ensuite les valeurs par défaut pour les colonnes sur lesquelles on n'a pas fait varier les critères.

```
[50]: # by default, accents are stripped
result_df['strip_accents'] = result_df['strip_accents'].fillna('accent_removal')
# scoring and embedding method are not applicable if projection
result_df.loc[result_df['similarity'] == 'projection', ['scoring', 'embedding_method']] = 'not_applicable'
# add default value to scoring for remainder
result_df.loc[pd.isna(result_df['scoring']), 'scoring'] = 'default'
```

```

# default value for embedding for remainder
result_df['embedding_method'] = result_df['embedding_method'].fillna('no_embedding')
# projected and source norm are not applicable if similarity is cosine
result_df.loc[result_df['similarity'] == 'cosine', ['source_norm', 'projected_norm']] = 'not_applicable'
# default value for norms for remainder
result_df['source_norm'] = result_df['source_norm'].fillna('l2')
result_df['projected_norm'] = result_df['projected_norm'].fillna('l1')

```

```

[51]: fillna_dict = {
    'similarity': 'projection',
    'split_func': 'split_func3',
    'count_vect_type': 'TfidfVectorizer',
    'use_idf': False,
    'binary': False,
}
for key, val in fillna_dict.items():
    result_df[key] = result_df[key].fillna(val)

```

```

[52]: parms = ['split_func',
            'stop_words',
            'strip_accents',
            'binary',
            'use_idf',
            'ngram_range',
            'similarity',
            'projected_norm',
            'source_norm',
            'count_vect_type',
            'scoring',
            'embedding_method',
        ]
result_df.reset_index().set_index(parms).drop('index', axis=1).sort_index().sample(5)

```

[52]:

```
[53]: result_df.to_csv(Path('.') / 'model_tuning_results.csv')
```

```
[54]: result_df = pd.read_csv(Path('.') / 'model_tuning_results.csv')
```

Sélection des fonctions de preprocessing

```

[55]: fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[result_df['run'] == 1],
         'x': 'stop_words',
         'hue': 'strip_accents',
     }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
                          color='blue',
                          ax=axs[i],
                          )
    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )
    axs[i].set_xlim(0,1)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Gestion des stopwords', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Pas de retrait de stopwords', 'Retrait des stopwords'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][2:],
```

```

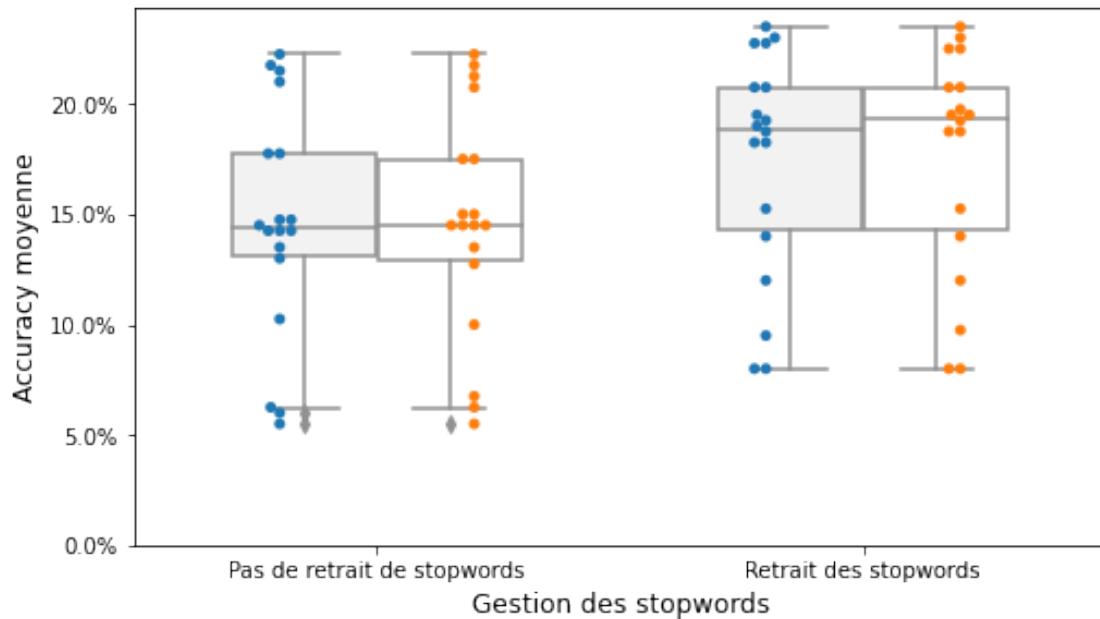
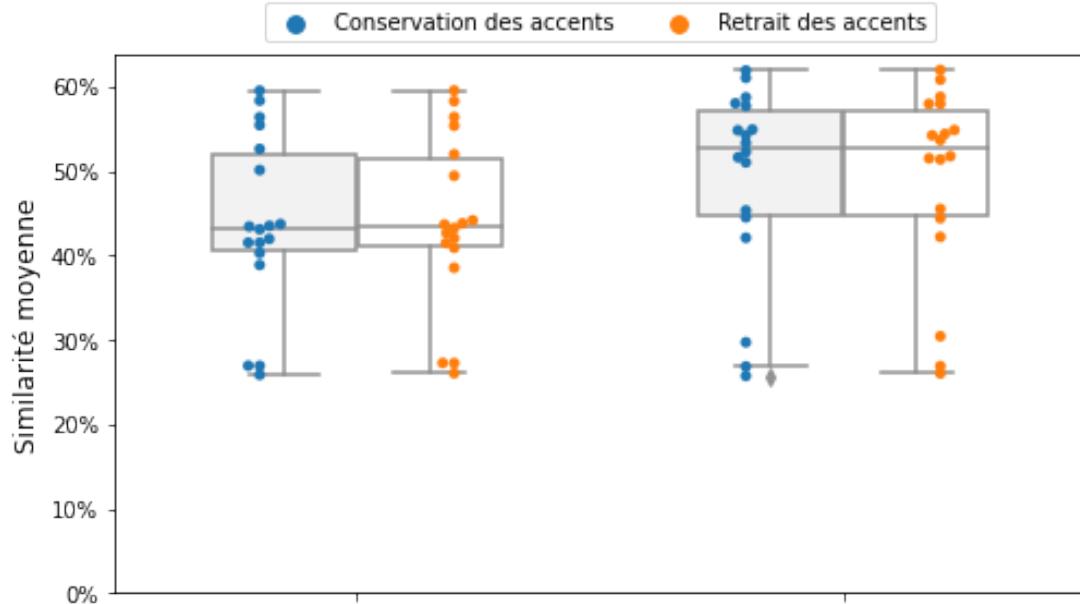
    labels=['Conservation des accents', 'Retrait des accents'],
    loc='center',
    ncol=2,
    bbox_to_anchor=(0, 1, 1, 0.12),
    bbox_transform= axs[0].transAxes,
)

fig.suptitle('Comparaison des fonctions de preprocessing', fontsize=16, y=.95)
# fig.savefig(Path('..') / 'img' / 'tuning_prepro.png', bbox_inches='tight')

```

[55]: Text(0.5, 0.95, 'Comparaison des fonctions de preprocessing')

Comparaison des fonctions de preprocessing



Sélection de la fonction de split

```
[56]: fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[result_df['run'] == 1],
         'x': 'split_func',
         'hue': None,
         }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
                          #
                          color='blue',
                          ax=axs[i],
                          )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

    axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    #     axs[i].get_legend().remove()
    axs[i].set_xlabel('')

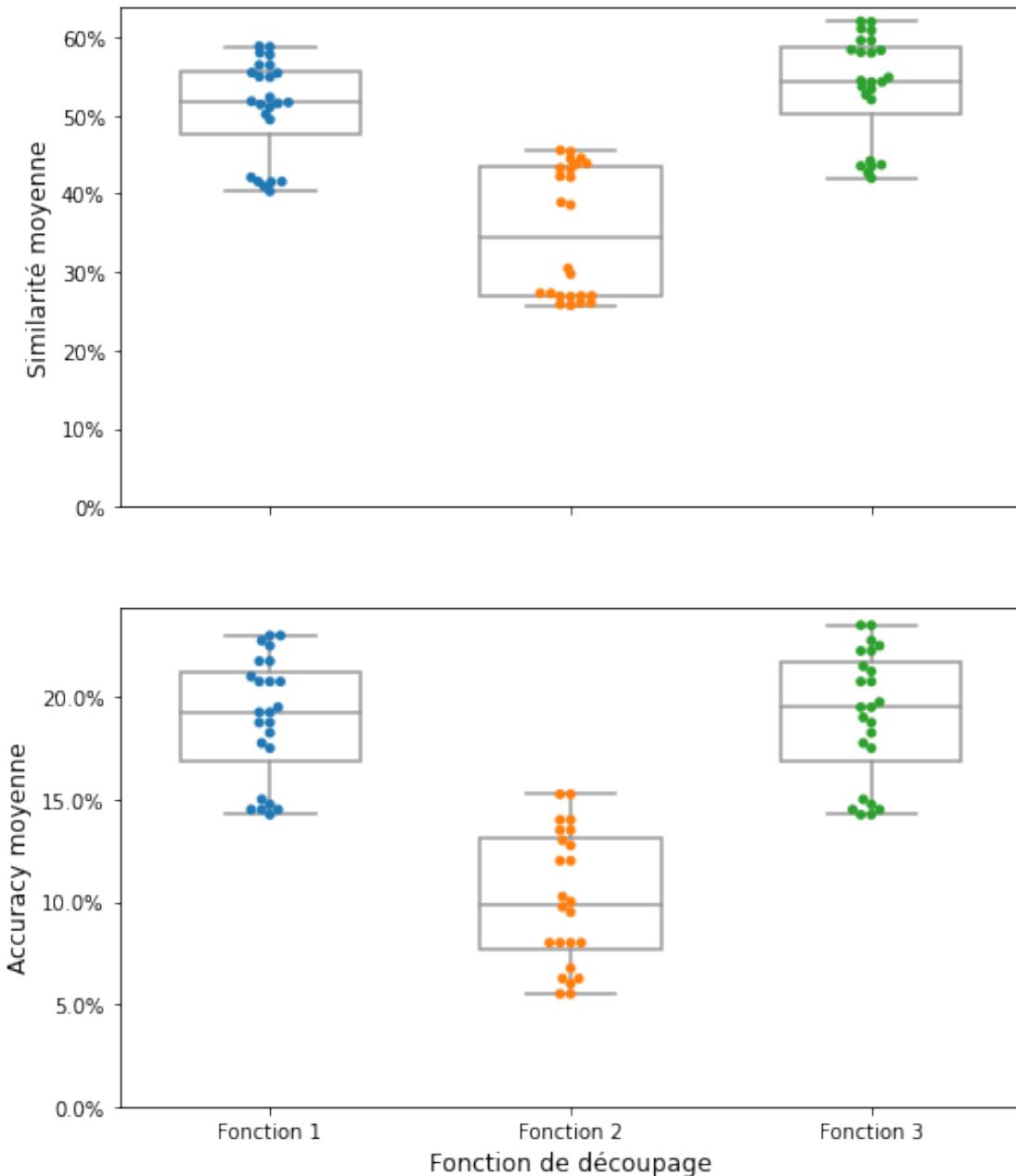
axs[1].set_xlabel('Fonction de découpage', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Fonction 1', 'Fonction 2', 'Fonction 3'])

# fig.legend(handles=axs[0].get_legend_handles_labels()[0][2:],
#             labels=['Conservation des accents', 'Retrait des accents'],
#             loc='center',
#             ncol=2,
#             bbox_to_anchor=(0, 1, 1, 0.12),
#             bbox_transform=axs[0].transAxes,
#             )

fig.suptitle('Comparaison des fonctions de découpage', fontsize=16, y=.92)
# fig.savefig(Path('..') / 'img' / 'tuning_split.png', bbox_inches='tight')
```

```
[56]: Text(0.5, 0.92, 'Comparaison des fonctions de découpage')
```

Comparaison des fonctions de découpage



Comparatif des similarités

```
[57]: result_df['simil_kind'] = result_df['similarity'] + result_df['source_norm'] + result_df['projected_norm']
```

```
[59]: fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 2) &
                               (result_df['strip_accents'] == 'accent_removal') &
                               (result_df['stop_words'] == 'stop_word_list') &
                               (result_df['scoring'].isin({'default', 'not_applicable'})) &
                               (result_df['embedding_method'].isin({'no_embedding', 'not_applicable'}))],
```

```

'x': 'simil_kind',
'hue': None,
}

patch_list = [
    mpatch.Rectangle((-1, 0), 2.5, 1, color='green', alpha=.2, edgecolor=None),
    mpatch.Rectangle((1.5, 0), 5, 1, color='blue', alpha=.2, edgecolor=None),
    mpatch.Rectangle((6.5, 0), 5, 1, color='orange', alpha=.2, edgecolor=None),
    mpatch.Rectangle((11.5, 0), 5, 1, color='red', alpha=.2, edgecolor=None),
    mpatch.Rectangle((16.5, 0), 2, 1, color='purple', alpha=.2, edgecolor=None),
    mpatch.Rectangle((-1, 0), 2.5, 1, color='green', alpha=.2, edgecolor=None),
    mpatch.Rectangle((1.5, 0), 5, 1, color='blue', alpha=.2, edgecolor=None),
    mpatch.Rectangle((6.5, 0), 5, 1, color='orange', alpha=.2, edgecolor=None),
    mpatch.Rectangle((11.5, 0), 5, 1, color='red', alpha=.2, edgecolor=None),
    mpatch.Rectangle((16.5, 0), 2, 1, color='purple', alpha=.2, edgecolor=None),
]

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
                          #
                          color='blue',
                          ax=axs[i],
                          )

    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].set_xlabel('')
    for j in range(len(patch_list) // 2):
        axs[i].add_patch(patch_list[i * len(patch_list) // 2 + j])

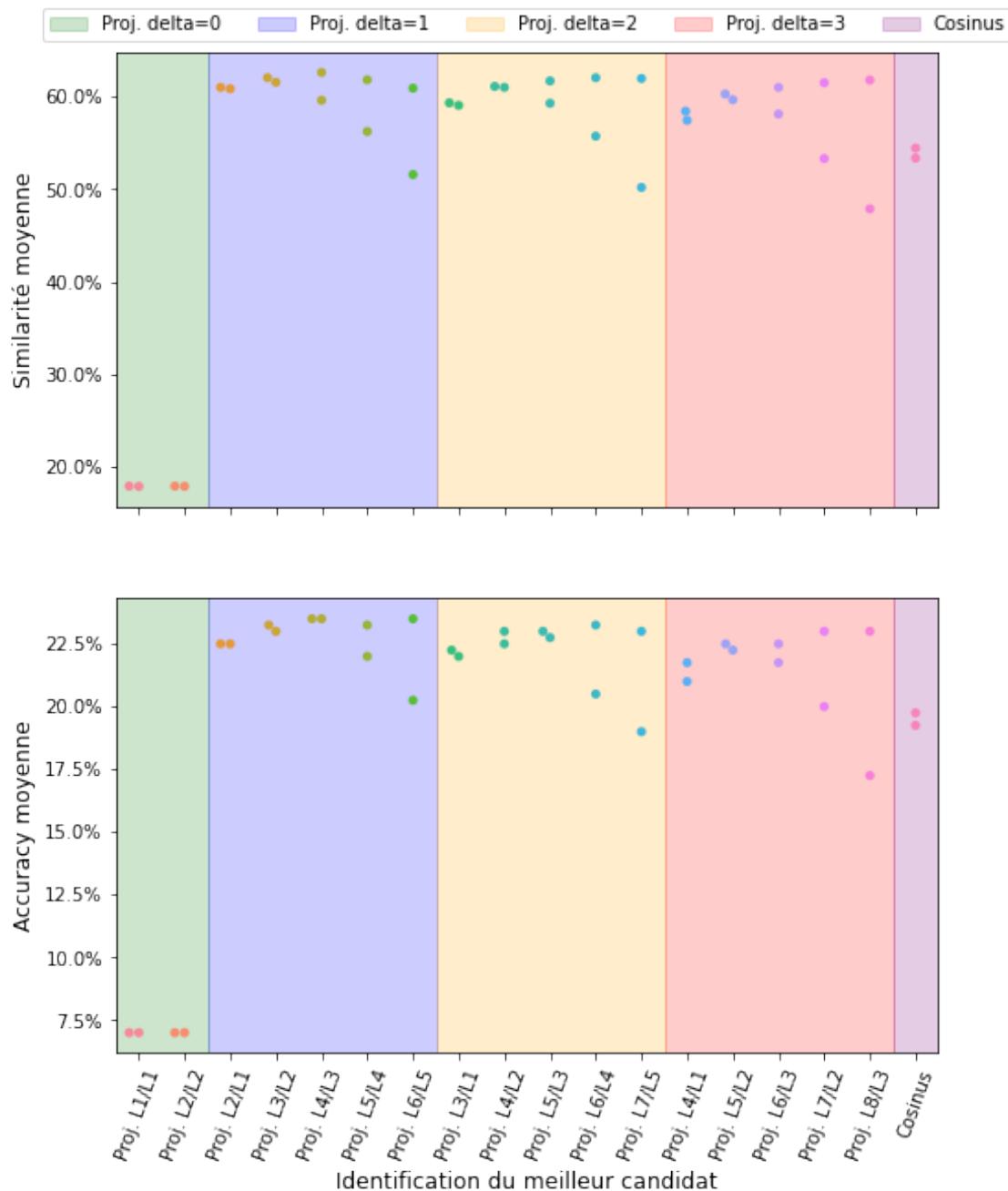
axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
labels = ['Proj. L1/L1',
          'Proj. L2/L2',
          'Proj. L2/L1',
          'Proj. L3/L2',
          'Proj. L4/L3',
          'Proj. L5/L4',
          'Proj. L6/L5',
          'Proj. L3/L1',
          'Proj. L4/L2',
          'Proj. L5/L3',
          'Proj. L6/L4',
          'Proj. L7/L5',
          'Proj. L4/L1',
          'Proj. L5/L2',
          'Proj. L6/L3',
          'Proj. L7/L2',
          'Proj. L8/L3',
          'Cosinus']
plt.setp(axs[1].xaxis.get_majorticklabels(), rotation=70)
axs[1].set_xticklabels(labels)
fig.legend(handles=patch_list[len(patch_list) // 2:],
           labels=['Proj. delta=0', 'Proj. delta=1', 'Proj. delta=2', 'Proj. delta=3', 'Cosinus'],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
           )

fig.suptitle("Comparaison des méthodes d'identification du meilleur candidat", fontsize=16, y=.945)
# fig.savefig(Path('..') / 'img' / 'tuning_similarite.png', bbox_inches='tight')

```

[59]: Text(0.5, 0.945, "Comparaison des méthodes d'identification du meilleur candidat")

Comparaison des méthodes d'identification du meilleur candidat



```
[60]: fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[result_df['run'] == 7],
         'x': 'similarity',
         'hue': 'use_idf',
         }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
```

```

        dodge=True,
        color='blue',
        ax=axs[i],
    )

sns.boxplot(**parms,
            y=feature,
            ax=axs[i],
            color='white',
            width=.6,
            )

axs[i].set_ylim(0,)
axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
axs[i].get_legend().remove()
axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Cosinus', 'Projection L4/L3'])

fig.legend(handles= axs[0].get_legend_handles_labels()[0][2:],
           labels=[ "Sans inverse document frequency", "Avec calcul de l'inverse document frequency"],
           loc='center',
           ncol=2,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
           )

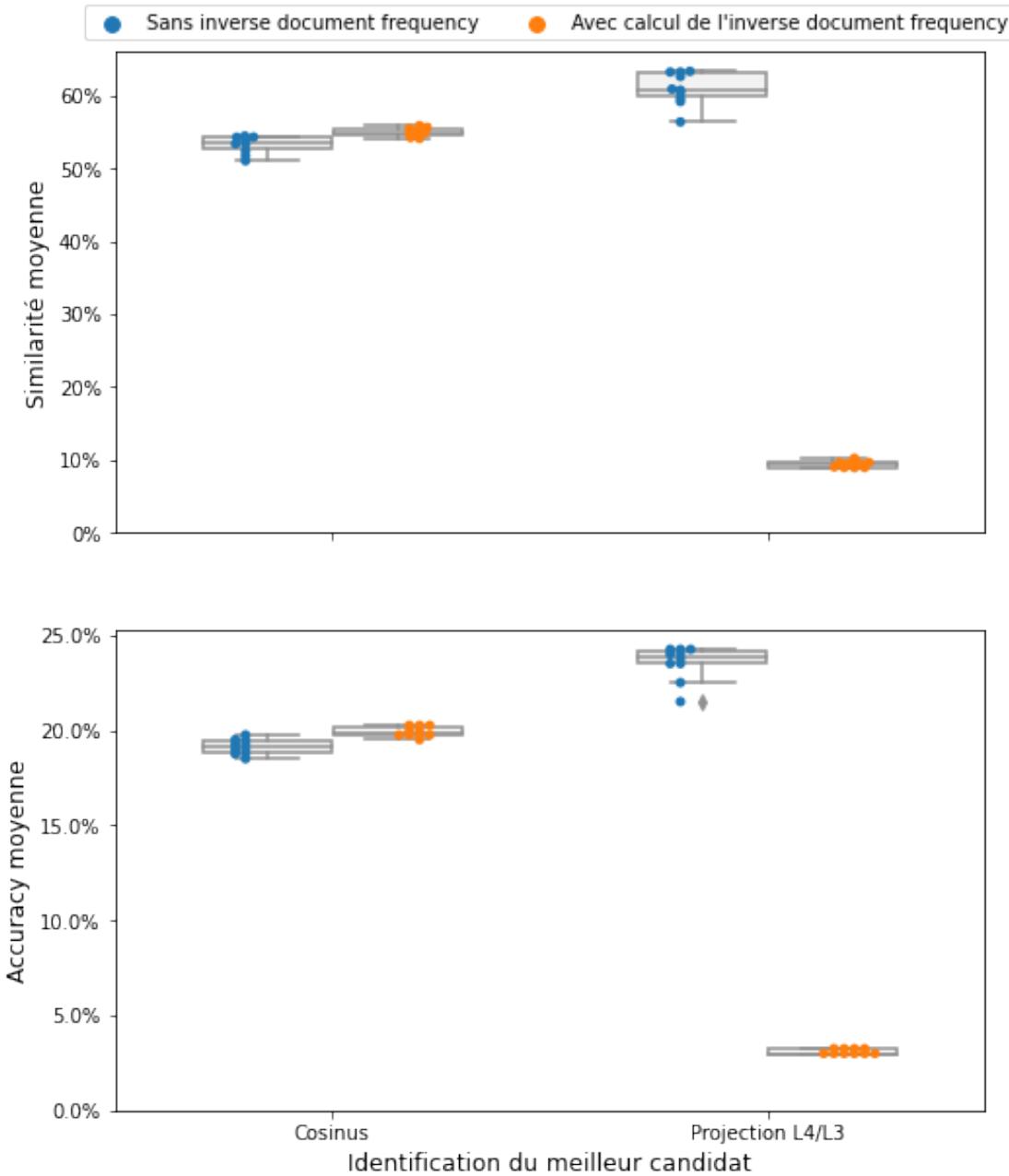
fig.suptitle('Comparaison des modes de vectorisation : idf', fontsize=16, y=.95)

# fig.savefig(Path('..') / 'img' / 'tuning_idf.png', bbox_inches='tight')

```

[60]: Text(0.5, 0.95, 'Comparaison des modes de vectorisation : idf')

Comparaison des modes de vectorisation : idf



```
[61]: fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 7) &
                               ((result_df['similarity'] == 'cosine') | # & result_df['use_idf'] /
                                (result_df['similarity'] == 'projection') & ~result_df['use_idf'])],
          'x': 'similarity',
          'hue': 'ngram_range',
         }

for i, feature in enumerate(feats):
```

```

swarm = sns.swarmplot(**parms,
                      y=feature,
                      dodge=True,
#                      color='blue',
                      ax=axs[i],
                     )

#      sns.boxplot(**parms,
#                  y=feature,
#                  ax=axs[i],
#                  color='white',
#                  width=.6,
#                 )

#      axs[i].set_ylim(0)
axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
axs[i].get_legend().remove()
axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Cosinus', 'Projection L4/L3'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][:],
           labels=['Monogrammes', 'Bigrammes', 'Trigrammes', '4-grammes', '5-grammes'],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
          )

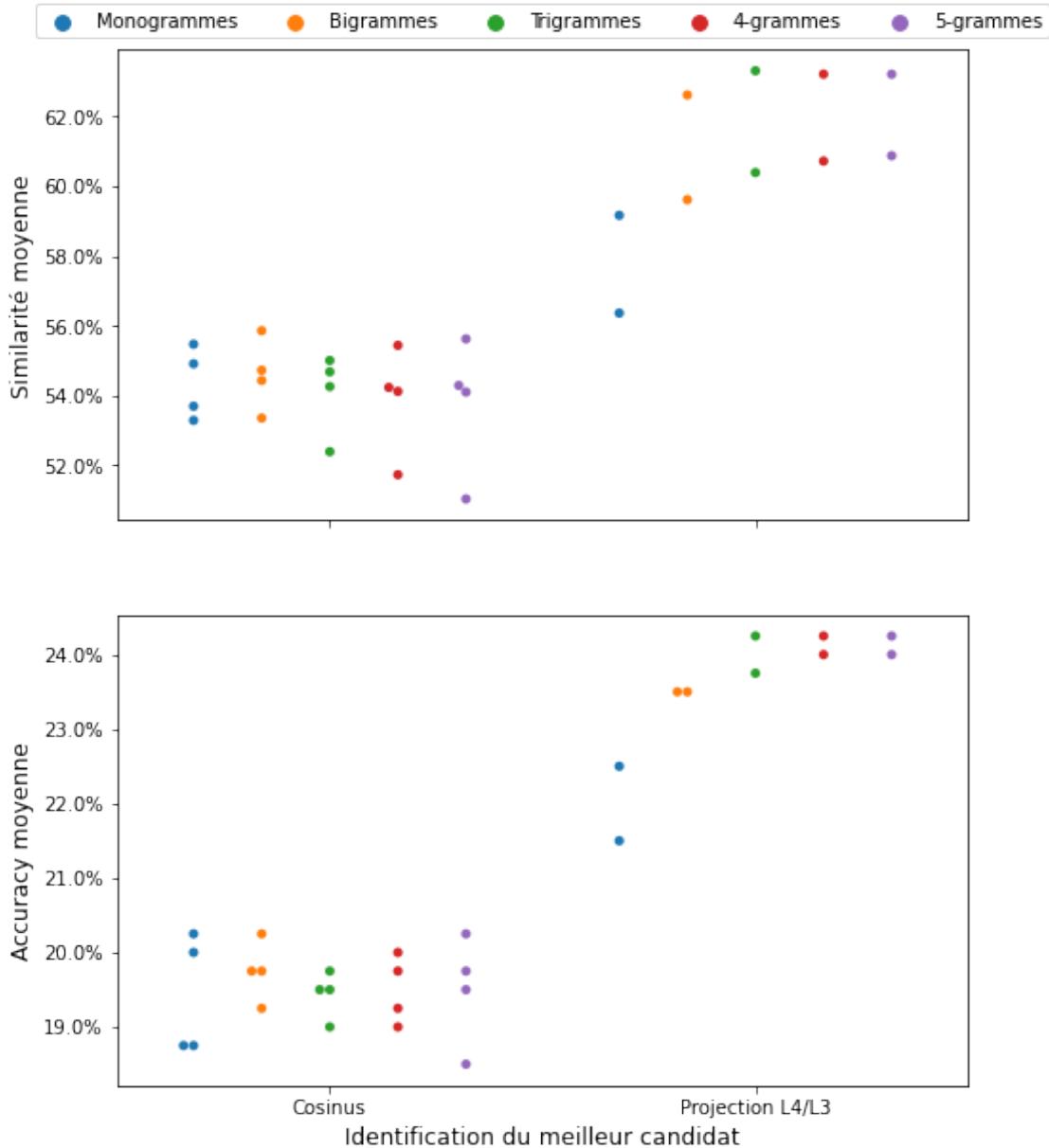
fig.suptitle('Comparaison des modes de vectorisation : n-grams', fontsize=16, y=.95)

# fig.savefig(Path('..') / 'img' / 'tuning_ngrams.png', bbox_inches='tight')

```

[61]: Text(0.5, 0.95, 'Comparaison des modes de vectorisation : n-grams')

Comparaison des modes de vectorisation : n-grams



```
[62]: # fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 5)
                               ],
          'x': 'scoring',
          'hue': 'use_idf',
          }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
                          color='blue',
                          ax=axs[i],
```

```

        )

sns.boxplot(**parms,
            y=feature,
            ax=axs[i],
            color='white',
            width=.6,
            )

#     axs[i].set_ylim(0)
axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
axs[i].get_legend().remove()
axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Sans score', 'Score absolu', 'Score relatif'])

fig.legend(handles= axs[0].get_legend_handles_labels()[0][2:],
           labels=["Sans idf", "Avec calcul de l'idf"],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
           )

fig.suptitle('Comparaison des modes de vectorisation : Scores spécifiques', fontsize=16, y=.95)

# fig.savefig(Path('..') / 'img' / 'tuning_score.png', bbox_inches='tight')

```

[62]: Text(0.5, 0.95, 'Comparaison des modes de vectorisation : Scores spécifiques')

```

[63]: fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 5)
                               ],
          'x': 'embedding_method',
          'hue': 'scoring',
          }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
                          color='blue',
                          ax=axs[i],
                          )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

#     axs[i].set_ylim(0)
axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
axs[i].get_legend().remove()
axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Sans embedding', 'Word2Vec', 'tSVD'])

fig.legend(handles= axs[0].get_legend_handles_labels()[0][3:],
           labels=["Sans scoring", "Score absolu", "Score relatif"],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
           )

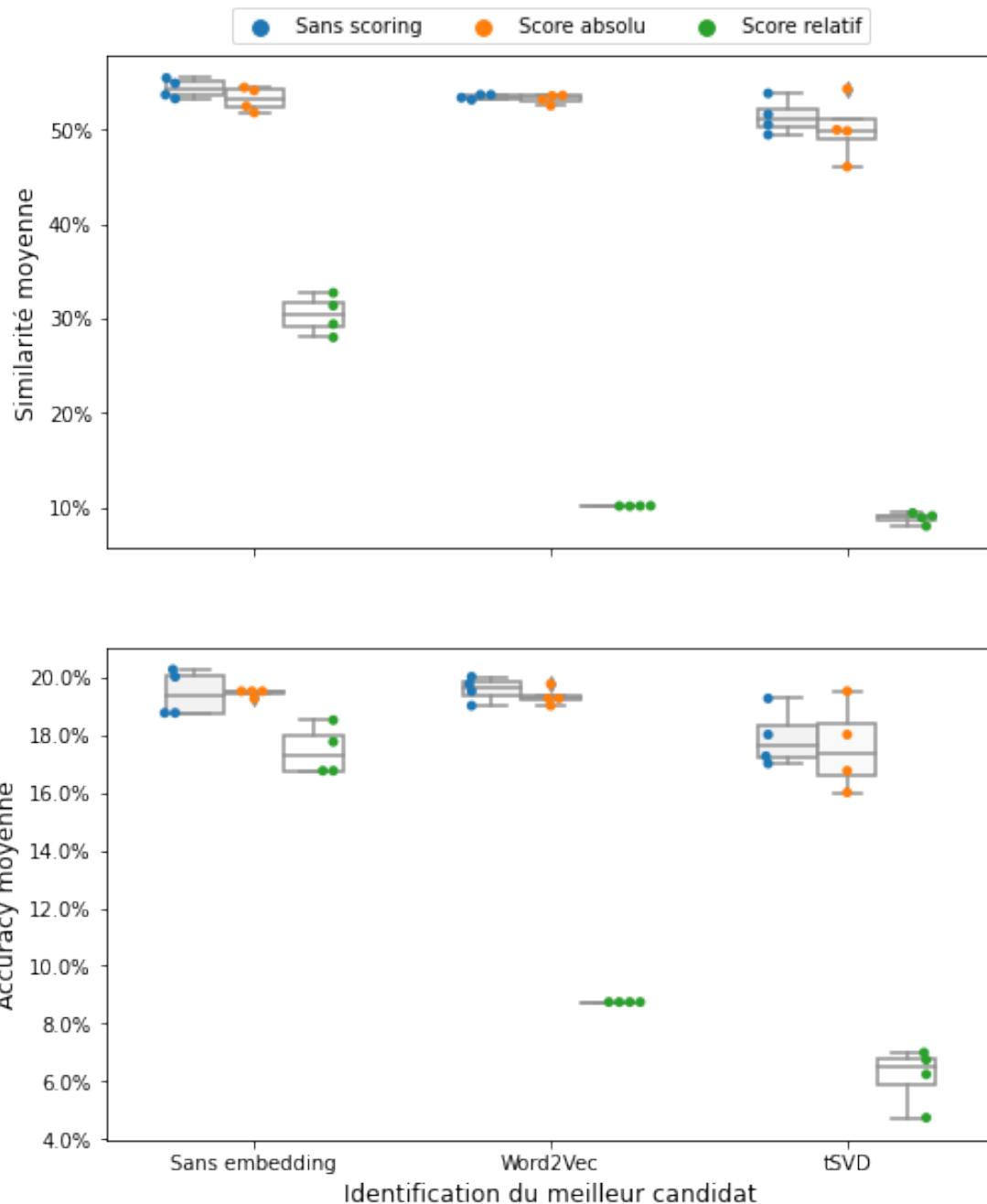
fig.suptitle('Comparaison des modes de vectorisation : embeddings', fontsize=16, y=.95)

# fig.savefig(Path('..') / 'img' / 'tuning_embedding.png', bbox_inches='tight')

```

```
[63]: Text(0.5, 0.95, 'Comparaison des modes de vectorisation : embeddings')
```

Comparaison des modes de vectorisation : embeddings



1.3 Evaluation finale

On évalue la performance du modèle avec les meilleurs paramètres sur le set de test, après entraînement sur le set d'entraînement.

```
[64]: parm_dict = {'Splitter__splitter_func': splitter_funcs[2],  
                 'SimilaritySelector__count_vect_type': 'TfidfVectorizer',  
                 'SimilaritySelector__similarity': 'projection',
```

```

        'SimilaritySelector__source_norm': 'l14',
        'SimilaritySelector__projected_norm': 'l13',
        'SimilaritySelector__count_vect_kwargs': {'ngram_range': (1, 3),
                                                'stop_words': stop_words,
                                                'strip_accents': 'unicode',
                                                'binary': True,
                                                'use_idf': False,
                                                }
    }

process_pipe.set_params(**parm_dict)
process_pipe.fit(train, train['ingredients'])
print(f"Levenshtein similarity at final evaluation: {lev_scorer(process_pipe, test, test['ingredients']):.2%}")
print(f"Accuracy at final evaluation: {custom_accuracy(process_pipe, test, test['ingredients']):.2%}")

```

Launching 8 processes.
 Launching 8 processes.
 Levenshtein similarity at final evaluation: 67.18%
 Launching 8 processes.
 Accuracy at final evaluation: 27.00%

```
[65]: predicted = process_pipe.predict(test)
lev_sim = partial(text_similarity, similarity='levenshtein')
```

Launching 8 processes.

```
[66]: comparison = (predicted.rename('Predicted')
                   .to_frame()
                   .join(test['ingredients'])
                   .rename({'ingredients': 'Target'}, axis=1))
comparison['Similarity'] = comparison.apply(lambda x: f"{lev_sim(x['Predicted'], x['Target']):.2%}", axis=1)
comparison.sample(5)
```

uid	Predicted	Target	Similarity
e51b7fd6-d878-47f8-a36b-f10f8d4087bd	1/2 1/4 1/8 1/16 1/32	Débris de truffes d'hiver, jus de truffes, sel	13.04%
f45db604-11ad-4756-aeab-3a5a1a34f914	Ingrédients: sucre; sirop de glucose; dextrose...	sucre; sirop de glucose; dextrose; gélatine; a...	93.58%
2ca5dc9e-8058-499a-affe-3ec9c06d5b57	Gastronomie \n70%/A/30% R	100% Arabica	9.09%
2286f782-9d2e-410f-84c4-4ab88003a002	INGREDIENTS: Pêches et poires en cubes (avec l...	Pêches et poires en cubes (avec leur jus d'ori...	92.80%
21233a00-bc20-40fc-acb9-ee2e2321cac2	Boisson gazeuse aromatisée au jus de fruit à b...		0.00%

```
[68]: with pd.option_context("max_colwidth", 2100):
    tex_str = (
        comparison.replace(r'^\s*$', np.nan, regex=True)
        .to_latex(index=False,
                  index_names=False,
                  column_format='p{7cm}p{7cm}c',
                  na_rep='<nien>',
                  longtable=True,
                  header=["Liste d'ingrédients prédicté", "Liste d'ingrédients cible", "Sim."],
                  label='tbl:final_prediction',
                  caption="Prédictions du meilleur modèle sur le set de test",
                  )
        .replace(r'\textbackslash n', r' \newline ')
        .replace(r'\\', r'\\ \\hline')
    )
# with open(Path('..') / 'tbls' / 'final_prediction.tex', 'w') as file:
#     file.write(tex_str)
```

Analyse quantitative

Pierre MASSÉ

June 11, 2020

1 Analyse quantitative multibranche

```
[1]: # data analysis
import pandas as pd
pd.options.display.width=108
import numpy as np

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib_venn import venn3_unweighted, venn3
import matplotlib as mpl
# mpl.rcParams['text.usetex'] = True
# plt.rcParams['text.latex.preamble'] = [r'\usepackage{lmodern}']

# utils
from pathlib import Path
```

Définition des couleurs :

```
[2]: c_pomona = tuple(val / 255 for val in [0, 92, 132])
c_terreazur = tuple(val / 255 for val in [0, 152, 170])
c_episaveurs = tuple(val / 255 for val in [255, 69, 0])
c_passionfroid = tuple(val / 255 for val in [109, 32, 124])
c_deliceetcreation = tuple(val / 255 for val in [97, 45, 28])
c_saveursdantoine = tuple(val / 255 for val in [156, 34, 63])
```

On charge les données d'un fichier exporté du système de gestion des branches RHD (SAP).

```
[3]: path = Path('..') / 'data' / 'export2020.csv'

types = {
    'material': 'object',
    'branch': 'int',
    'plant': 'object',
    'type': 'object',
    'designation': 'object',
    'del_mand': 'bool',
    'del_plant': 'bool',
    'march_group': 'object',
    'storage_cond': 'object',
    'hier': 'object',
}
df = pd.read_csv(path,
                  sep=';',
                  encoding='latin-1',
                  engine='python',
                  header=0,
                  skipfooter=1, # footer line with totals in export
                  dtype=types,
                  true_values=['X'], # for del_mand and del_plant
                  false_values=['', np.nan], # for del_mand and del_plant
)
df = df[types.keys()] #filter and reorder columns
```

Parmi les colonnes conservées, on a : - le code article (material)

- le code de branche de création (branch).
 - 1: PassionFroid
 - 2: EpiSaveurs
 - 3: TerreAzur
- le code d'activation sur une branche (plant).
 - 1PPF: PassionFroid

- 2PES: EpiSaveurs
- 3PTA: TerreAzur
- le type d'article (type). Seuls ZNEG et ZPRE représententent des articles de marchandises.
 - ZNEG: Négoce
 - ZPRE: Prestation
 - ZENG: Article d'engagement (fictif pour facturation)
 - ZEMB: Article d'emballage (ex: palette)
 - ZSER: Article de service
- le libellé de l'article (designation)
- si l'article est marqué pour suppression pour toutes les branches (del_mand)
- si l'article est marqué pour suppression sur la branche mentionnée dans la colonne plant (del_plant).
- le groupe de marchandises (march_group) :
 - ZSURGE: Surgelés
 - ZFRAIS: Frais (PassionFroid)
 - ZEPI: Epicerie
 - ZBOI: Boissons
 - ZHYG: Hygiène et chimie
 - ZFLF: Fruits et légumes (TerreAzur)
 - ZPMF: Produits de la mer (TerreAzur)
 - ZFP: Fleurs et plantes
 - ZELAB: Produits élaborés (TerreAzur)
- la condition de stockage (storage_cond) :
 - FR: Frais (PassionFroid)
 - SU: Surgelé,
 - EP: Epicerie,
 - AL: Alcool
 - HY: Hygiène et chimie
 - FL: Fruits et légumes (TerreAzur)
 - FP: Fleurs et plantes
 - MA: Marée
 - SA: Saurisserie (produits élaborés de la mer)
 - SE: Articles de Service
 - PL: Articles de publicité
- la hiérarchie produit (hier). Un plan de classement sur 6 niveaux, représentés par 2 caractères numériques chacun.

On crée une nouvelle feature qui correspond au niveau 1 de la hiérarchie produit.

```
[4]: # Creation of first level of product hierarchy
df.loc[:, 'hier1'] = df.hier.str[:2]
```

On définit un dictionnaire permettant de rappeler les libellés long des divers codes présents dans le dataset.

```
[5]: # Label names
lab = {'type': 'Type de produit',
       'march_group': 'Groupe de marchandises',
       'storage_cond': 'Condition de stockage',
       'hier1': 'Niveau 1 hiérarchie produit',
       '1PPF': 'PassionFroid',
       '2PES': 'EpiSaveurs',
       '3PTA': 'TerreAzur',
       'ZNEG': 'Article de négoce',
       'ZPRE': 'Article de prestation',
       'ZSURGE': 'Surgelés',
       'ZFRAIS': 'Frais',
       'ZEPI': 'Epicerie',
       'ZBOI': 'Boissons',
       'ZHYG': 'Hygiène',
       'ZFLF': 'Fruits et Légumes',
       'ZPMF': 'Produits de la mer',
       'ZELAB': 'Produits élaborés',
       'ZFP': 'Fleurs et plantes',
       'ZAUTRE': 'Autres',
       'SU': 'Surgelés',
       'FR': 'Frais',
       'EP': 'Epicerie',
       'AL': 'Alcool',
       'HY': 'Hygiène',
       'FL': 'Fruits et légumes',
       'MA': 'Marée',
       'FP': 'Fleurs et plantes',
       'SA': 'Saurisserie',
       'PL': 'Publicié',
       '10': 'Beurre, oeufs, fromage',
       '20': 'Elaborés',
       '30': 'Garnitures et fruits',
       '40': 'Produits carnés',
       '50': 'Produits de la mer',
       '60': 'Consommables',
       '70': 'Emballage',
```

```
'80': 'Publicité sur le lieu de vente',
'83': 'Epicerie',
'85': 'Liquides',
'87': 'Hygiène et entretien',
'90': 'Services',
'92': 'Fruits',
'94': 'Légumes',
'96': 'Produits de la mer Frais',
'98': 'Fleurs - plantes',
}
```

[6]: df.loc[[5000, 90000, 100000, 130000, 110000], :]

```
material branch plant type designation del_mand del_plant \
5000    15712      2 2PES PSVNX CERN BRISURE S/AZ SAC 1KGX12 CERNO   True    True
90000   153086     3 3PTA ZNEG MANGUE KENT 351/550G PAD 12F DELIC BR°  False   False
100000  165387     1 1PPF ZNEG SALADE PLT 1KGX12 HAMAL  False   False
130000  203582     1 1PPF ZPRE EFFILOCHE BOEUF BARBACOA (2KGX6)/12KG CS  False   False
110000  177238     2 2PES ZNEG COMP POIRE ALL BIO BTE 5/1X3 STM   False   False

march_group storage_cond hier hier1
5000      ZEPI        EP 832020500505  83
90000     ZFLF        FL 920518010405  92
100000    ZFRAIS       FR 202520150505  20
130000    ZSURGE       SU 401015051505  40
110000    ZEPI        EP 832005451505  83
```

On va définir deux masques, permettant de filtrer : - les articles actifs (i.e. non supprimé niveau mandant ni branche) - les articles actifs de marchandises (i.e. qui ne sont pas des articles "spéciaux")

[7]: active_mask = ~df.del_mand & ~df.del_plant
active_march_mask = active_mask & df.type.isin(['ZNEG', 'ZPRE'])

On peut calculer la volumétrie d'articles et la représenter comme un histogramme. Les données de Délice et Création et Saveurs d'Antoine sont issue d'estimations fournies par le métier.

```
counts = df.groupby('plant')['material'].count().rename('Total')
filtered_counts = df[active_mask].groupby('plant')['material'].count().rename('Actifs')
filtered_counts2 = df[active_march_mask].groupby('plant')['material'].count().rename('Marchandises')

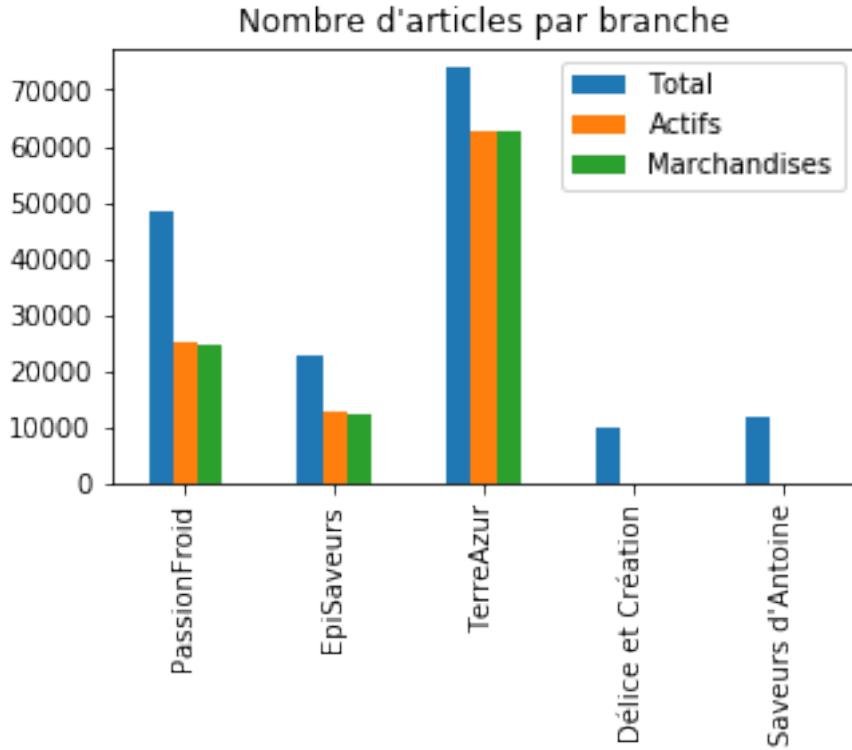
report = pd.concat([counts, filtered_counts, filtered_counts2], axis=1)
report.loc['Délice et Création', :] = [10000, np.nan, np.nan]
report.loc['Saveurs d\'Antoine', :] = [12000, np.nan, np.nan]
report.rename({'1PPF': 'PassionFroid',
               '2PES': 'EpiSaveurs',
               '3PTA': 'TerreAzur'}, inplace=True)
report.index.rename('Branche', inplace=True)
report = report.astype('Int64')
report.to_latex(Path('..') / 'tbls' / 'Articles par branche.tex',
                bold_rows=True,
                column_format='lccc',
                na_rep='-'
               )
report
```

[8]:

Branche	Total	Actifs	Marchandises
PassionFroid	48478	24898	24554
EpiSaveurs	22498	12798	12241
TerreAzur	73804	62789	62710
Délice et Création	10000	NaN	NaN
Saveurs d'Antoine	12000	NaN	NaN

[9]:

```
fig, ax = plt.subplots(figsize=(5,3))
report.plot(kind='bar', ax=ax)
ax.set_title('Nombre d\'articles par branche')
ax.set_xlabel('')
fig.savefig(Path('..') / 'img' / 'Articles par branche.png', bbox_inches='tight')
```



On peut également construire le diagramme de Venn des articles pour les branches RHD :

```
[10]: # Filtering the dataset with active materials, and active merchandize materials
branch_sets = [set(df.loc[df.plant == branch_, 'material']) for branch_ in ['1PPF', '2PES', '3PTA']]
filtered_df = df.loc[active_mask]
filtered_sets = [set(filtered_df.loc[filtered_df.plant == branch_, 'material']) for branch_ in ['1PPF', '2PES', '3PTA']]
filtered_march_df = df.loc[active_march_mask]
filtered_march_sets = [set(filtered_march_df.loc[filtered_march_df.plant == branch_, 'material'])
for branch_ in ['1PPF', '2PES', '3PTA']]
```



```
[11]: # This function is used to add label on Venn diagrams axes without showing spines
# (matplotlib-venn disables totally axis's, and spines need to get erased after
# axis's reactivation)
def labelize(ax, label, where='bottom', **kwargs):
    ax.set_axis_on()
    for spine in ['top', 'bottom', 'left', 'right']:
        ax.spines[spine].set_visible(False)
    if where == 'bottom':
        ax.set_xlabel(label, **kwargs)
    elif where == 'left':
        ax.set_ylabel(label, **kwargs)
    else:
        raise ValueError(f'Unexpected "where" argument: {where}')
```



```
[12]: # Construction of the diagrams
scope = ['Total', 'Actifs', 'Marchandises']
types = ['Non pondéré', 'Pondéré']
nrows, ncols = len(types), len(scope)

fig, axs = plt.subplots(nrows, ncols, sharex='col', sharey='row', figsize=(18, 8))

for col, source_df in enumerate([branch_sets, filtered_sets, filtered_march_sets]):
    for row, venn_kind in enumerate([venn3_unweighted, venn3]):
        venn_kind(source_df, set_labels=['PassionFroid', 'EpiSaveurs', 'TerreAzur'], ax=axs[row, col])
        if col == 0:
            labelize(axs[row, col], types[row], where='left', fontsize=18, labelpad=10)
```

```

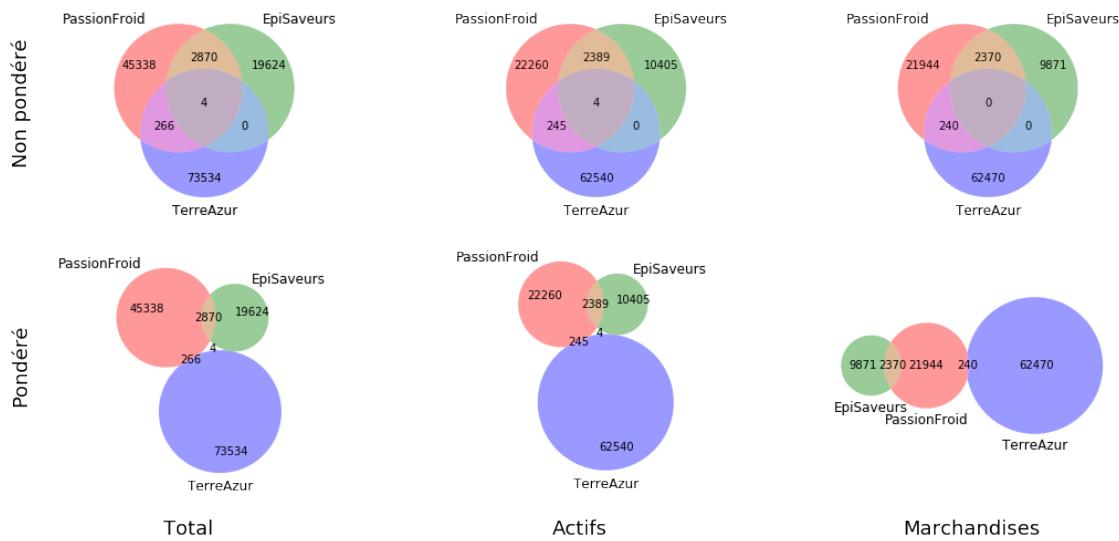
if row == 1:
    labelize(axes[row, col], scope[col], where='bottom', fontsize=18, labelpad=40)

# Adjusting the min and max of axes lims, as they are not the same by default
xmin = min([axes[row][col].get_xlim()[0] for row in range(nrows) for col in range(ncols)])
xmax = max([axes[row][col].get_xlim()[1] for row in range(nrows) for col in range(ncols)])
ymin = min([axes[row][col].get_ylim()[0] for row in range(nrows) for col in range(ncols)])
ymax = max([axes[row][col].get_ylim()[1] for row in range(nrows) for col in range(ncols)]) + 0.1

for row in range(nrows):
    for col in range(ncols):
        axes[row, col].set_xlim(xmin, xmax)
        axes[row, col].set_ylim(ymin, ymax)

# Saving the file to disk so that it is included in the report
fig.savefig(Path('..') / 'img' / 'Diagrammes de Venn articles.png', bbox_inches='tight')

```



On peut constater que les articles utilisés par les 3 branches RHD sont des articles "spéciaux".

```
[13]: df[df.material.isin(df.material.value_counts()[df.material.value_counts() >= 3].index)]
```

```

[13]:      material branch plant type designation del_mand \
144564    DECOMPTE     1  2PES ZSER ARTICLE DE DECOMPTE CONDITIONS ARRIERES False
144565    DECOMPTE     1  3PTA ZSER ARTICLE DE DECOMPTE CONDITIONS ARRIERES False
144566    DECOMPTE     1  1PPF ZSER ARTICLE DE DECOMPTE CONDITIONS ARRIERES False
144612    FC41849      1  1PPF ZSER RÉGUL SURFACTURATION DÉCONDITIONNEMENT False
144613    FC41849      1  2PES ZSER RÉGUL SURFACTURATION DÉCONDITIONNEMENT False
144614    FC41849      1  3PTA ZSER RÉGUL SURFACTURATION DÉCONDITIONNEMENT False
144642    LOT_ENGT     1  1PPF ZENG          LOT ENGAGEMENT False
144643    LOT_ENGT     1  3PTA ZENG          LOT ENGAGEMENT False
144644    LOT_ENGT     1  2PES ZENG          LOT ENGAGEMENT False
144719  S_PALETTE_PERDUE  3  3PTA ZEMB PALETTE 80X120 PERDUE False
144720  S_PALETTE_PERDUE  3  2PES ZEMB PALETTE 80X120 PERDUE False
144721  S_PALETTE_PERDUE  3  1PPF ZEMB PALETTE 80X120 PERDUE False

      del_plant march_group storage_cond      hier hier1
144564    False     ZAUTRE       NaN 900505050505   90
144565    False     ZAUTRE       NaN 900505050505   90
144566    False     ZAUTRE       NaN 900505050505   90
144612    False     ZAUTRE       NaN 900505050505   90
144613    False     ZAUTRE       NaN 900505050505   90
144614    False     ZAUTRE       NaN 900505050505   90
144642    False       NaN       NaN       NaN       NaN
144643    False       NaN       NaN       NaN       NaN
144644    False       NaN       NaN       NaN       NaN
144719    False     ZAUTRE       NaN 700510050505   70
144720    False     ZAUTRE       NaN 700510050505   70
144721    False     ZAUTRE       NaN 700510050505   70

```

On peut ensuite essayer de représenter les comptages d'articles sur les diverses variables catégorielles.

```
[14]: # Definition of feature and order to show
features = {'type': None,
            'march_group': ['ZFRAIS', 'ZSURGE', 'ZEPI', 'ZHYG', 'ZBOI', 'ZFLF', 'ZPMF', 'ZELAB', 'ZFP'],
            'storage_cond': ['FR', 'SU', 'EP', 'HY', 'FL', 'MA', 'SA', 'FP', 'PL'],
            'hier1': None,
        }

# Definition of color palette
palette = {'1PPF': c_passionfroid,
            '2PES': c_episaveurs,
            '3PTA': c_terreazur,
        }

[15]: fig, axs = plt.subplots(nrows=len(features), ncols=2, figsize=(13, 15))
# for each feature, draw counts without and with hue
for idx, (feature, order) in enumerate(features.items()):
    # drawing without hue
    sns.countplot(data=df.loc[active_march_mask],
                  x=feature,
                  order=order,
                  ax=axs[idx][0],
                  color=c_pomona)
    # remove y label, and set x label to full length text
    axs[idx][0].set_ylabel('')
    axs[idx][0].set_xlabel(lab[feature], fontsize=16)
    # drawing with hue
    sns.countplot(data=df.loc[active_march_mask],
                  x=feature,
                  hue='plant',
                  order=order,
                  palette=palette,
                  ax=axs[idx][1],
                  )
    # remove y label, and set x label to full length text
    axs[idx][1].set_ylabel('')
    axs[idx][1].set_xlabel(lab[feature], fontsize=16)
    # hide legend for each axis
    axs[idx][1].legend().set_visible(False)

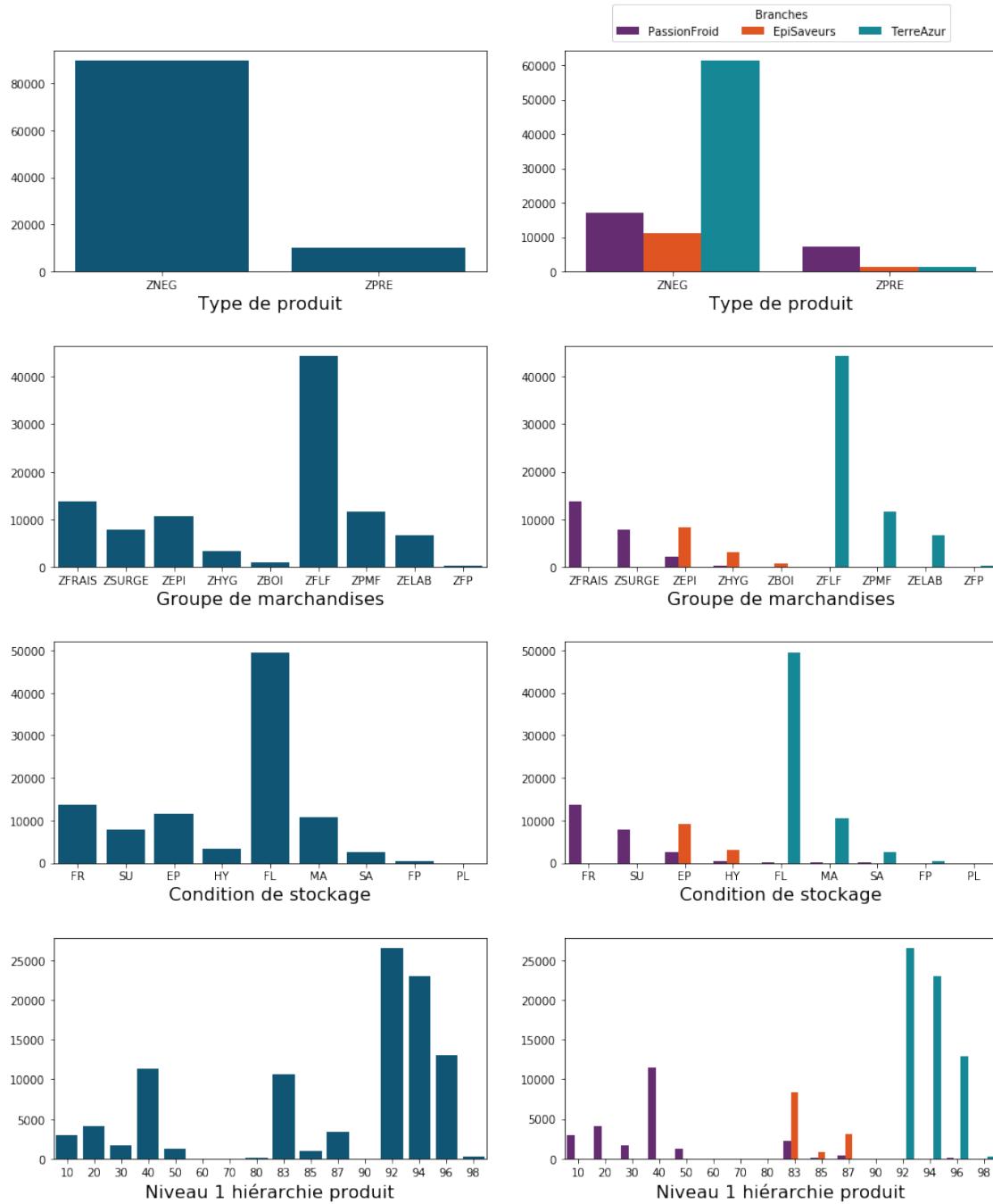
# redraw legend for the whole figure, above, centered and
# expanded
handles, labels = axs[3][1].get_legend_handles_labels()
fig.legend(handles,
            [lab[label] for label in labels],
            ncol=len(handles),
            title='Branches',
            loc='center',
            bbox_to_anchor=(0, 1, 1, 0.25),
            bbox_transform=axs[0][1].transAxes,
            # mode='expand',
        )

# adding a title
fig.suptitle('Répartition des articles selon les features catégorielles',
             fontsize=24,
             y=1.025,
             va='bottom',
        )

# adding padding between plots
fig.tight_layout(pad=3.0)

# saving to disk
fig.savefig(Path('..') / 'img' / 'Repartition articles categories.png', bbox_inches='tight')
```

Répartition des articles selon les features catégorielles



```
[24]: def long_lab(label):
    if label in lab:
        return(label + ' - ' + lab[label])
    else:
        return(label)

for feature in features.keys():
    # Construct the pivot table for the feature
    piv = pd.pivot_table(df.loc[active_march_mask],
                         columns='plant',
```

```

        index=feature,
        values='material',
        aggfunc='count',
        fill_value=0,
    )
# Add a 'Total' column
piv['Total'] = piv['1PPF'] + piv['2PES'] + piv['3PTA']

# Changing 0s to '-'
piv = piv.replace(0, '-')

# Reorder indices so that they follow the order defined in
# lab dictionary
if np.all(piv.index.isin(lab.keys())): # check to avoid filtering piv!
    piv = piv.reindex([key for key in lab.keys() if key in piv.index])

# Rename indices, columns and axes for pretty printing
piv = (piv.rename(long_lab, axis=0)
       .rename(lab, axis=1)
       .rename_axis(lab[feature])
       .rename_axis('Branche', axis=1))

print(piv)
print('-----')
# Save to LaTeX format to be included in report
piv.to_latex(Path('..') / 'tbls' / ('Repartition par ' + feature + '.tex'),
             bold_rows=True,
             column_format='lcccc',
             na_rep='-', )
)

```

Branche	PassionFroid	EpiSaveurs	TerreAzur	Total
Type de produit				
ZNEG - Article de négoce	17166	11048	61273	89487
ZPRE - Article de prestation	7388	1193	1437	10018
<hr/>				
Branche	PassionFroid	EpiSaveurs	TerreAzur	Total
Groupe de marchandises				
ZSURGE - Surgelés	7756	-	-	7756
ZFRAIS - Frais	13785	6	4	13795
ZEPI - Epicerie	2298	8305	-	10603
ZBOI - Boissons	126	826	-	952
ZHYG - Hygiène	350	3078	-	3428
ZFLF - Fruits et Légumes	4	-	44133	44137
ZPMF - Produits de la mer	142	-	11594	11736
ZELAB - Produits élaborés	91	-	6644	6735
ZFP - Fleurs et plantes	-	-	297	297
ZAUTRE - Autres	2	26	38	66
<hr/>				
Branche	PassionFroid	EpiSaveurs	TerreAzur	Total
Condition de stockage				
SU - Surgelés	7758	-	-	7758
FR - Frais	13781	6	3	13790
EP - Epicerie	2430	9155	-	11585
HY - Hygiène	344	3080	-	3424
FL - Fruits et légumes	78	-	49508	49586
MA - Marée	126	-	10501	10627
FP - Fleurs et plantes	-	-	286	286
SA - Saurisserie	34	-	2408	2442
PL - Publicié	2	-	1	3
<hr/>				
Branche	PassionFroid	EpiSaveurs	TerreAzur	Total
Niveau 1 hiérarchie produit				
10 - Beurre, oeufs, fromage	3010	6	1	3017
20 - Elaborés	4150	2	6	4158
30 - Garnitures et fruits	1701	-	-	1701
40 - Produits carnés	11413	-	-	11413
50 - Produits de la mer	1214	-	2	1216
60 - Consommables	1	-	-	1
70 - Emballage	-	1	-	1
80 - Publicité sur le lieu de vente	34	25	37	96
83 - Epicerie	2306	8296	-	10602
85 - Liquides	135	836	-	971
87 - Hygiène et entretien	348	3075	-	3423
90 - Services	10	-	-	10
92 - Fruits	35	-	26543	26578
94 - Légumes	37	-	22929	22966
96 - Produits de la mer Frais	160	-	12891	13051
98 - Fleurs - plantes	-	-	301	301

Analyse données du PIM

Pierre MASSÉ

June 11, 2020

1 Analyse des données du PIM

1.1 Extraction des données

1.1.1 Préambule technique

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

```
[2]: # imports and customization of display
import io
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.patches as mpatch

from src.pimapi import Requester
```

```
[3]: # monkeypatch _repr_latex_ for better inclusion of dataframes output in report
def _repr_latex_(self, size='scriptsize'):
    return(f"\\"\\resizable{{\\linewidth{{!}}}{\\begin{{{size}}}}\\centering{{\\self.to_latex()}}\\end{{{size}}}}\"")
pd.DataFrame._repr_latex_ = _repr_latex_
```

1.1.2 Récupération des données

Le requêtage des données dans le PIM s'appuie sur la classe `Requester` du module `pimapi`.

```
[5]: requester = Requester('prd')
# Let's fetch the full content of PIM system
requester.fetch_all_from_PIM()
requester.result
```

Done

```
[5]: [<Response [200]>,
<Response [200]>]
```

A ce stade, les données sont chargées en mémoire sous la forme de fichiers JSON. La conversion des données récupérées par l'API se fait via la méthode `result_to_dataframe` de la classe `Requester`.

```
[6]: df = requester.result_to_dataframe()
```

1.2 Définitions pour les mises en formes

1.2.1 Descriptifs longs

On définit un dictionnaire permettant de “traduire” les codes de champs en libellés long.

```
[7]: lab = {
    'code': 'Code produit',
    'supplier': 'Code fournisseur',
    'type': 'Type de produit',
    'GTIN': 'GTIN',
    'base_unit': 'Unité de base',
    'net_weight': 'Poids net',
    'gross_weight': 'Poids brut',
    'dry_weight': 'Poids net égoutté',
    'volume': 'Volume',
    'total_life': 'Durée de vie totale',
    'remaining_life': 'Durée minimale restante',
    'type_cons': 'Type de conservation',
    'before_open': 'Conservation avant ouv.',
    'after_open': 'Conserveration après ouv.',
    'cons_temp': 'Température',
    'temp_des': 'Libellé temporaire',
    'supplier_des': 'Désignation produit fournisseur',
    'supplier_code': 'Code interne fournisseur',
    'brand': 'Marque commerciale',
    'regulatory_des': 'Dénomination réglementaire',
    'properties.pprodi:supplierDesignation': 'Désignation produit fournisseur',
    'properties.pprod:ingredientsList': "Liste d'ingrédients",
}
```

1.2.2 Champs intéressants

On liste également les champs intéressants pour un affichage plus court du dataframe.

```
[12]: def_fields = {'properties.vig:code': 'code',
                 'properties.psec:supplierCode': 'supplier',
                 'properties.pprodt:typeOfProduct': 'type',
                 'properties.pprodi:gtin': 'gtin',
                 'properties.pprodi:supplierDesignation': 'designation'}
df.rename(def_fields, axis=1).loc[:, list(def_fields.values())].sample(4)
```

```
[12]:
```

uid	code	supplier	type	type	gtin	designation
dbe6bb33-9067-4033-a6e7-b23799b052cc	PIMP-0000011355	PIMF-0000000283	pomProduct	hygiene	3342690158379	Kit part dessert rond fond noir et couvercle c...
d702e415-9b92-4e00-9052-14e0c10dcda97	PIMP-0000007911	PIMF-0000000357	pomProduct	grocery	5000189327290	Quality street en boite 480 g QUALITY STREET
3677fd51-26d2-4e3e-899b-8ce3ddc7dfac	PIMP-0000005604	PIMF-0000000255	pomProduct	nonAlcoholicDrink	3180290022177	Sirop de miel en bouteille verre 1 L GIFFARD
a0b41419-ddd6-47d6-8cf3-77501a7c4ccd	PIMP-0000001185	PIMF-0000000283	pomProduct	hygiene	3342690088829	#Cuillere plate nr sac (50U)x24 AF

1.3 Description des attributs des produit

1.3.1 Volumétrie des attributs

On constate que chaque produit porte un très grand nombre d'attributs :

```
[13]: print('Count of columns in df:', len(df.columns))
print('\nInfo of df:')
df.info()
```

Count of columns in df: 487

```
Info of df:
<class 'pandas.core.frame.DataFrame'>
Index: 13300 entries, afee12c7-177e-4a68-9539-8cbb68442503 to 6dfce29e-fd4c-4670-9f9c-5c02a5b4d52a
Columns: 487 entries, entity-type to properties.notif:notifications
dtypes: bool(12), float64(60), int64(2), object(413)
memory usage: 48.5+ MB
```

De plus, de par la nature hiérarchique du format JSON, certains attributs dits ``multivalués'' sont parfois stockés sous forme de liste dans le dataframe ``à plat''. Par exemple, on peut voir que le pays de transformation, ou les facettes, peuvent être multivalués.

```
[14]: df.loc[['609af223-2f14-4f83-a553-cef276f2eca7',
            'c94013e4-0dca-441a-85c1-0b29ecb54d0a',
            '82d1af25-2bdd-4315-9670-67784b70dfa7'],
```

```
[14]: ['properties.pprodg:transfoCountries',
      'facets']]
```

[14]:

	properties.pprodg:transfoCountries	facets
uid		
609af223-2f14-4f83-a553-cef276f2eca7	[PL, FR, ES]	[Versionable, Folderish, Commentable, beginnin...
c90413e4-0dca-441a-85c1-0b29ecb54d0a	[DE, NO, BE, RU, CH, BG, LT, GR, FR, UA, HU, E...	[endMigration, Versionable, Folderish, Comment...
82d1af25-2bdd-4315-9670-67784b70dfa7	[FR]	[endMigration, Versionable, Folderish, Comment...

De plus, certains attributs sont dits ``complexes'', car chacune des valeurs de la liste est elle-même un dictionnaire d'attribut. La combinaison des deux, des attributs ``complexes multivalués'' existe également. On a alors une liste de dictionnaires. On peut comme ceci imbriquer des niveaux jusqu'à n'importe quelle profondeur.

C'est par exemple le cas des labels qui sont multivalués (un produit peut porter plusieurs labels), qui sont des complexes portant : - le type de label (bio, Label Rouge, ...) - la date de fin de validité du label (si applicable) - le fichier de certification du label (si applicable), qui est lui-même un complexe...

```
[15]: multilabel_ds = df.loc[df['properties.pprod1:labels'].apply(len) > 1, 'properties.pprod1:labels']
for uid, label_list in multilabel_ds.head(3).iteritems():
    print('product uid:', uid)
    for cpt, label in enumerate(label_list):
        print('\n\tlabel', cpt + 1, ':')
        for key, val in label.items():
            print('\t\t', key, ':', val)
    print('-----')
```

product uid: 41e2b3ca-2c56-4404-8381-352943c9bcef

```
label 1 :
    labelCertificateEndDate : None
    typeOfLabel : 20
    labelCertificateFile : {'name': 'San Marzano Statement_12.03.20.pdf', 'mime-type':
'application/pdf', 'encoding': None, 'digestAlgorithm': 'MD5', 'digest': '847073d2dd4af83bb1cd63739d6f75f',
'length': '175829', 'data': 'https://produits.groupe-pomona.fr/nuxeo/nxfile/default/41e2b3ca-2c56-4404-8381-
352943c9bcef/pprod1:labels/0/labelCertificateFile/San%20Marzano%20Statement_12.03.20.pdf?changeToken=130-0'}

label 2 :
    labelCertificateEndDate : None
    typeOfLabel : 20
    labelCertificateFile : {'name': 'Agro Qualita - New Document 22-Mag-2020 18-57-21(1).pdf',
'mime-type': 'application/pdf', 'encoding': None, 'digestAlgorithm': 'MD5', 'digest':
'4ec126e8b458e66fd5614f253aa4fb91', 'length': '1632788', 'data': 'https://produits.groupe-pomona.fr/nuxeo/nx
file/default/41e2b3ca-2c56-4404-8381-352943c9bcef/pprod1:labels/1/labelCertificateFile/Agro%20Qualita%20-%20
New%20Document%2022-Mag-2020%2018-57-21(1).pdf?changeToken=130-0'}

label 3 :
    labelCertificateEndDate : None
    typeOfLabel : 20
    labelCertificateFile : {'name':
'disciplinare_Pomodoro_San_marzano_dell_Agro_Sarnese_Nocerino_26.8.2019.pdf', 'mime-type':
'application/pdf', 'encoding': None, 'digestAlgorithm': 'MD5', 'digest': '42bc7b3fd33dec7361d57f4a3926e5fe',
'length': '228489', 'data': 'https://produits.groupe-pomona.fr/nuxeo/nxfile/default/41e2b3ca-2c56-4404-8381-
352943c9bcef/pprod1:labels/2/labelCertificateFile/disciplinare_Pomodoro_San_marzano_dell_Agro_Sarnese_Noceri
no_26.8.2019.pdf?changeToken=130-0'}
```

product uid: 362e6230-ba3a-4396-8a47-728b0a1d56db

```
label 1 :
    labelCertificateEndDate : 2024-12-30T23:00:00.000Z
    typeOfLabel : 80
    labelCertificateFile : {'name': 'KCC Coleshill Tissue Paper Ecolabel Renewal Certificate
Mar 2020.pdf', 'mime-type': 'application/pdf', 'encoding': None, 'digestAlgorithm': 'MD5', 'digest':
'6615e3027ff2e014fdc3fa37e67851bb', 'length': '425662', 'data': 'https://produits.groupe-pomona.fr/nuxeo/nx
file/default/362e6230-ba3a-4396-8a47-728b0a1d56db/pprod1:labels/0/labelCertificateFile/KCC%20Coleshill%20Tiss
ue%20Paper%20Ecolabel%20Renewal%20Certificate%20Mar%202020.pdf?changeToken=45-0'}
```

```
label 2 :
    labelCertificateEndDate : 2022-09-16T22:00:00.000Z
    typeOfLabel : NA
    labelCertificateFile : {'name': 'FCC_DoC_Coleshill Mill_PW_blue_Ref13351_Eng V01.pdf',
'mime-type': 'application/pdf', 'encoding': None, 'digestAlgorithm': 'MD5', 'digest':
```

```
'78cfcc67b8bf0f693e060088f7d97c48', 'length': '84473', 'data': 'https://produits.groupe-pomona.fr/nuxeo/nxfile/default/362e6230-ba3a-4396-8a47-728b0a1d56db/pprodl:labels/1/labelCertificateFile/FCC_DoC_Coleshill%20Mil1_PW_blue_Ref13351_Eng%20V01.pdf?changeToken=45-0'}
-----
product uid: 3c2a8d1a-634d-40bb-9852-81eb8a340114

    label 1 :
        labelCertificateEndDate : None
        typeOfLabel : 30
        labelCertificateFile : None

    label 2 :
        labelCertificateEndDate : None
        typeOfLabel : 40
        labelCertificateFile : None
-----
```

1.3.2 Description des principaux attributs

On commence par déclarer des utilitaires permettant de mettre en forme les représentations.

```
[16]: # Defining main data to explore
mappings = {
    'identification': {
        'properties.vig:code': 'code',
        'properties.psec:supplierCode': 'supplier',
        'properties.pprodtop:typeOfProduct': 'type',
        'properties.pprodःgtin': 'GTIN',
    },
    'dimensions': {
        'properties.pprodtop:baseUnit': 'base_unit',
        'properties.pprodःnetWeight': 'net_weight',
        'properties.pprodःgrossWeight': 'gross_weight',
        'properties.pprodःdryWeight': 'dry_weight',
        'properties.pprodःvolume': 'volume',
    },
    'conservation': {
        'properties.pprodःtotalLife': 'total_life',
        'properties.pprodःguaranteedLife': 'remaining_life',
        'properties.pprodःtypeOfConservation': 'type_cons',
        'properties.pprodःconservationBeforeOpening': 'before_open',
        'properties.pprodःconservationAfterOpening': 'after_open',
        'properties.pprodःconservationTemperature': 'cons_temp',
    },
    'designation': {
        'properties.pprodःtemporaryUnitLabel': 'temp_des',
        'properties.pprodःsupplierDesignation': 'supplier_des',
        'properties.pprodःinternalSupplierProductCode': 'supplier_code',
        'properties.pprodःsupplierCommercialBrand': 'brand',
        'properties.pprodःregulatoryName': 'regulatory_des',
    }
}

# Helper function to transform pandas `to_latex` method output to a tabularx env instead.
def to_tabularx(stringio):
    text = stringio.getvalue()
    text = text.replace(r'\begin{tabular}', r'\begin{tabularx}{\linewidth}')
    text = text.replace(r'\end{tabular}', r'\end{tabularx}')
    return(text)

# Function that saves dataframe as Latex tabularx files as input
def save_to_disk(df, path, lab=lab, tex_label=None):
    text = io.StringIO()
    c_format = 'l' + 'X' * len(df.columns)
    (df.rename(lab, axis=1)
     .to_latex(text,
               bold_rows=True,
               column_format=c_format,
               na_rep='-' ,
               label=tex_label,
               ))
    with open(path, mode='w') as file:
        file.write(to_tabularx(text))
```

1.3.3 Analyses spécifiques : Statuts

On commence par s'intéresser aux différents statuts des produits. On commence récupérer certaines valeurs de facettes intéressantes : - ``beginningMigration``: elle signifie que le produit a été repris du système historique (le GIP) - ``endMigration``: elle signifie que le produit qui a été créé par reprise fait l'objet d'un processus complet de récupération

```
des données et de contrôle par Pomona
```

```
[17]: df['begin_mig'] = df['facets'].apply(lambda x: 'beginningMigration' in x)
df['end_mig'] = df['facets'].apply(lambda x: 'endMigration' in x)

(df.loc[:, ['begin_mig', 'end_mig']]
 .reset_index()
 .pivot_table(values='uid',
              index=['begin_mig'],
              aggfunc='count',
              columns='end_mig',
              fill_value=0,
              )
 .rename({True: 'Créé au démarrage', False: 'Créé après le démarrage'}, axis=0)
 .rename({True: 'Facette fin de migration : Oui', False: 'Facette fin de migration : Non'}, axis=1)
).to_latex(
#     Path('..') / 'tbls' / 'migration_status.tex',
    index_names=False,
    column_format='lcc',
    bold_rows=True,
)

print(
(df.loc[:, ['begin_mig', 'end_mig']]
 .reset_index()
 .pivot_table(values='uid',
              index=['begin_mig'],
              aggfunc='count',
              columns='end_mig',
              fill_value=0,
              )
 .rename({True: 'Créé au démarrage', False: 'Créé après le démarrage'}, axis=0)
 .rename({True: 'Facette fin de migration : Oui', False: 'Facette fin de migration : Non'}, axis=1)
)
```

	Facette fin de migration : Non	Facette fin de migration : Oui
begin_mig		
Créé après le démarrage	1745	0
Créé au démarrage	7177	4378

On peut également voir les statuts courants des produits dans le PIM.

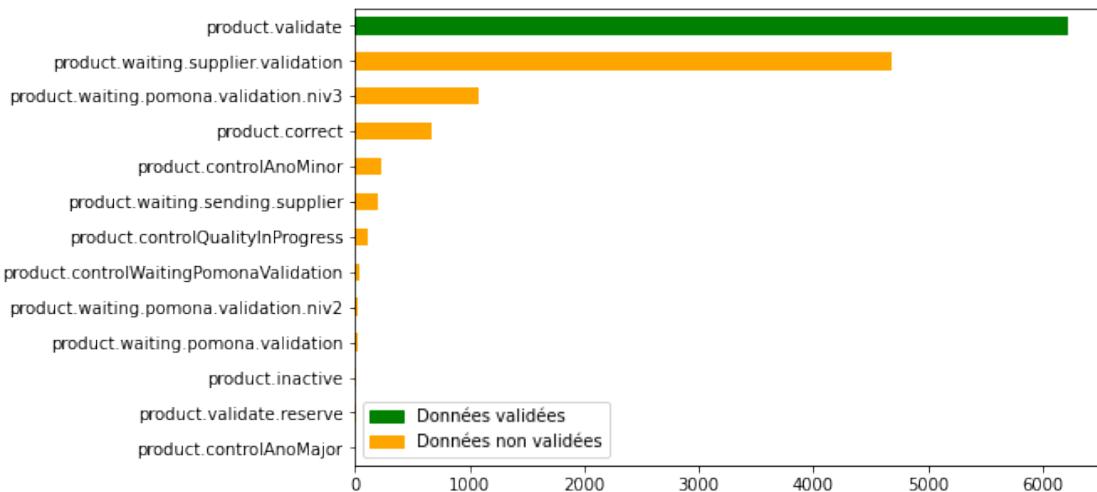
```
[18]: df['state'].value_counts().to_frame().to_latex(
#     Path('..') / 'tbls' / 'products_status.tex',
    column_format = 'lc',
    bold_rows=True,
)
print(df['state'].value_counts())
```

product.validate	6230
product.waiting.supplier.validation	4683
product.waiting.pomona.validation.niv3	1080
product.correct	662
product.controlAnoMinor	237
product.waiting.sending.supplier	201
product.controlQualityInProgress	108
product.controlWaitingPomonaValidation	35
product.waiting.pomona.validation.niv2	28
product.waiting.pomona.validation	20
product.inactive	12
product.validate.reserve	3
product.controlAnoMajor	1
Name: state, dtype: int64	

```
[19]: fig, ax = plt.subplots(figsize=(8,5))
color = ['orange'] * len(df['state'].value_counts())
color[-1] = 'g'
incorrect_data = mpatch.Patch(color='orange', label='Données non validées')
correct_data = mpatch.Patch(color='g', label='Données validées')
df['state'].value_counts().sort_values().plot.bart(ax=ax, color=color)
fig.suptitle('Répartition des produits en fonction de leurs statuts',
             fontsize=16)
ax.legend(handles=[correct_data, incorrect_data])
# fig.savefig(Path('..') / 'img' / 'products_status.png', bbox_inches='tight')
```

```
[19]: <matplotlib.legend.Legend at 0x7f2320e9c460>
```

Répartition des produits en fonction de leurs statuts



On calcule un statut spécifique à la validité des données en combinant le statut de migration et le statut du produit.

```
[20]: migration_mask = df.loc[:, 'end_mig'] | ~df.loc[:, 'begin_mig']
status_mask = (df.loc[:, 'state'] == 'product.validate')
df['data_ok'] = migration_mask & status_mask
```

```
[21]: cur_df = (df.loc[:, 'data_ok']
             .value_counts()
             .rename({False: 'Hors qualité', True: 'En qualité'})
             .rename('Répartition produit par qualité')
             .to_frame()
            )

cur_df.to_latex(
#           Path('..') / 'tbls' / 'products_quality.tex',
#           bold_rows=True,
#           column_format='lc',
#           na_rep='-', 
#           index=True,
#           index_names=True,
#           )
```

cur_df

[21]:

Répartition produit par qualité

Hors qualité	8718
En qualité	4582

```
[22]: (df.groupby('data_ok')
      [['properties.pprodq:conservationBeforeOpening',
        'properties.pprodg:guaranteedLife']]
      .describe(include='all')
      .stack())
```

[22]:

		properties.pprodq:conservationBeforeOpening	properties.pprodg:guaranteedLife
data_ok			
False	count	6361	6060.000000
	unique	6	NaN
	top	ambientTemperature	NaN
	freq	5698	NaN
	mean	NaN	346.446700
	std	NaN	372.476931
	min	NaN	0.000000
	25%	NaN	160.000000
	50%	NaN	270.000000
	75%	NaN	480.000000
	max	NaN	9999.000000
True	count	3660	3462.000000
	unique	7	NaN
	top	ambientTemperature	NaN
	freq	2553	NaN
	mean	NaN	360.798671
	std	NaN	392.404169
	min	NaN	0.000000
	25%	NaN	180.000000
	50%	NaN	360.000000
	75%	NaN	480.000000
	max	NaN	9999.000000

1.3.4 Exports des exemples et des descriptions des données structurées

On boucle sur les différents mappings, et on les sauvegarde dans des tableaux latex pour intégration au rapport.

```
[23]: df_dict = dict()

for map_type, mapping in mappings.items():
    cur_df = df.loc[:, list(mapping.keys()) + ['data_ok']].rename(mapping, axis=1).fillna(np.nan)
    df_dict[map_type] = cur_df.copy()
    desc = cur_df.groupby('data_ok').describe(include='all').stack()
    samp = cur_df.sample(n=5, random_state=42)
    print(map_type)
    print(samp.rename(lab, axis=1))
    print('-----')
    print(desc.rename(lab, axis=1)
          .round(3))
    )
    print('-----')

# Writing dataframes to .tex files
text = io.StringIO()
c_format = 'l' + 'X' * len(cur_df.columns)
(samp.rename(lab, axis=1)
 .to_latex(text,
            bold_rows=True,
            column_format=c_format,
            na_rep='-' ))
#
#     with open(Path('..') / 'tbls' / ('Exemple ' + map_type + '.tex'), mode='w') as file:
#         file.write(to_tabularx(text))

text = io.StringIO()
(desc.rename(lab, axis=1)
 .rename({False: 'Hors qualité', True: 'En qualité'})
 .round(3)
 .to_latex(text,
            bold_rows=True,
            column_format=c_format,
            na_rep='-' ,
            #index_names=False,
            multirow=True,
            )))
#
#     with open(Path('..') / 'tbls' / ('Desc ' + map_type + '.tex'), mode='w') as file:
#         file.write(to_tabularx(text))
```

identification

Code produit Code fournisseur Type de produit

GTIN \

uid

0891b705-0b7d-43ea-bab3-9df5875b8a96	PIMP-0000008707	PIMF-0000000018	grocery	3123070402608
731bf4e6-2380-4ce4-a859-0d084f3006cd	PIMP-0000001308	PIMF-0000000357	grocery	7613031886999
d92860c1-c252-4aee-82fe-a471de719af6	PIMP-0000003646	PIMF-0000000287	grocery	3083681003338
ebbd6b-ec4e-46b1-acbb-79ba0b68c1c5	PIMP-0000011491	PIMF-0000000448	hygiene	3573972384302
5717f881-afad-4f96-9c47-a4b597077e5c	PIMP-0000012513	PIMF-0000000565	hygiene	3661679121181

data_ok

uid	0891b705-0b7d-43ea-bab3-9df5875b8a96	False
731bf4e6-2380-4ce4-a859-0d084f3006cd	False	
d92860c1-c252-4aee-82fe-a471de719af6	False	
ebbd6b-ec4e-46b1-acbb-79ba0b68c1c5	True	
5717f881-afad-4f96-9c47-a4b597077e5c	True	

	Code produit	Code fournisseur	Type de produit	GTIN
data_ok				
False	count	8718	8718	8718 7641
	unique	8718	553	5 7423
	top	PIMP-0000009044	PIMF-0000000250	grocery
	freq	1	311	5742 68
True	count	4582	4582	4582 4450
	unique	4582	340	5 4133
	top	PIMP-0000008918	PIMF-0000000179	grocery
	freq	1	304	3051 281

	Unité de base	Poids net	Poids brut	Poids net égoutté	Volume	\
dimensions						
uid	0891b705-0b7d-43ea-bab3-9df5875b8a96	SAC	1.000	1.250	NaN	NaN
731bf4e6-2380-4ce4-a859-0d084f3006cd	BTE	0.900	1.000	NaN	NaN	
d92860c1-c252-4aee-82fe-a471de719af6	PCH	2.250	2.425	NaN	NaN	
ebbd6b-ec4e-46b1-acbb-79ba0b68c1c5	SAC	0.838	0.897	NaN	NaN	
5717f881-afad-4f96-9c47-a4b597077e5c	COL	5.712	5.847	NaN	NaN	

	data_ok					
uid	0891b705-0b7d-43ea-bab3-9df5875b8a96	False				
731bf4e6-2380-4ce4-a859-0d084f3006cd	False					
d92860c1-c252-4aee-82fe-a471de719af6	False					
ebbd6b-ec4e-46b1-acbb-79ba0b68c1c5	True					
5717f881-afad-4f96-9c47-a4b597077e5c	True					

	Unité de base	Poids net	Poids brut	Poids net égoutté	Volume
dimensions					
data_ok					
False	count	8718	8342.000	8342.000	717.000 877.000
	unique	29	NaN	NaN	NaN NaN
	top	BTE	NaN	NaN	NaN NaN
	freq	2238	NaN	NaN	NaN NaN
	mean	NaN	3.803	3.377	1.581 5.808
	std	NaN	75.226	53.165	1.265 47.049
	min	NaN	0.000	0.000	0.000 0.000
	25%	NaN	0.500	0.548	0.500 0.500
	50%	NaN	1.000	1.142	1.560 0.850
	75%	NaN	3.000	3.330	2.400 3.000
	max	NaN	4900.000	4730.500	20.000 1000.000
True	count	4582	4582.000	4582.000	312.000 1067.000
	unique	26	NaN	NaN	NaN NaN
	top	SAC	NaN	NaN	NaN NaN
	freq	1078	NaN	NaN	NaN NaN
	mean	NaN	2.187	2.406	1.288 9.794
	std	NaN	3.131	3.327	1.265 98.146
	min	NaN	0.000	0.000	0.000 0.000
	25%	NaN	0.442	0.514	0.360 0.500
	50%	NaN	1.000	1.075	0.600 1.000
	75%	NaN	3.000	3.151	2.300 2.000
	max	NaN	33.474	40.589	10.000 3100.000

	conservation					
	Durée de vie totale	Durée minimale restante	Type de conservation	\		
uid	0891b705-0b7d-43ea-bab3-9df5875b8a96	540.0	360.0	AM		
731bf4e6-2380-4ce4-a859-0d084f3006cd	548.0	120.0	AM			
d92860c1-c252-4aee-82fe-a471de719af6	720.0	120.0	AM			
ebbd6b-ec4e-46b1-acbb-79ba0b68c1c5	NaN	NaN	AM			
5717f881-afad-4f96-9c47-a4b597077e5c	NaN	NaN	AM			

	Conservation avant ouv.	Convervation après ouv.	Température	data_ok
uid	0891b705-0b7d-43ea-bab3-9df5875b8a96	ambientTemperature	coolAndDryPlace	NaN False
731bf4e6-2380-4ce4-a859-0d084f3006cd	coolAndDryPlace	coolAndDryPlace	NaN False	

d92860c1-c252-4aee-82fe-a471de719af6	ambientTemperature	coldFor48Hours	NaN	False	
ebbdbd6b-ec4e-46b1-acbb-79ba0b68c1c5	NaN	NaN	NaN	True	
5717f881-afad-4f96-9c47-a4b597077e5c	NaN	NaN	NaN	True	
<hr/>					
	Durée de vie totale	Durée minimale restante	Type de conservation	Conservation avant ouv. \	
data_ok					
False	count	5585.000	6060.000	8320	6361
	unique	NaN	NaN	2	6
	top	NaN	NaN	AM	ambientTemperature
	freq	NaN	NaN	8297	5698
	mean	651.076	346.447	NaN	NaN
	std	490.475	372.477	NaN	NaN
	min	0.000	0.000	NaN	NaN
	25%	360.000	160.000	NaN	NaN
	50%	540.000	270.000	NaN	NaN
	75%	900.000	480.000	NaN	NaN
	max	9999.000	9999.000	NaN	NaN
True	count	3394.000	3462.000	4582	3660
	unique	NaN	NaN	2	7
	top	NaN	NaN	AM	ambientTemperature
	freq	NaN	NaN	4569	2553
	mean	656.737	360.799	NaN	NaN
	std	482.590	392.404	NaN	NaN
	min	0.000	0.000	NaN	NaN
	25%	360.000	180.000	NaN	NaN
	50%	540.000	360.000	NaN	NaN
	75%	730.000	480.000	NaN	NaN
	max	9999.000	9999.000	NaN	NaN
<hr/>					
	Conserveration après ouv. Température				
data_ok					
False	count	6339	10		
	unique	18	10		
	top	coolAndDryPlace	ambiant		
	freq	2773	1		
	mean	NaN	NaN		
	std	NaN	NaN		
	min	NaN	NaN		
	25%	NaN	NaN		
	50%	NaN	NaN		
	75%	NaN	NaN		
	max	NaN	NaN		
True	count	3653	13		
	unique	18	5		
	top	coolAndDryPlace	<10°C		
	freq	1753	5		
	mean	NaN	NaN		
	std	NaN	NaN		
	min	NaN	NaN		
	25%	NaN	NaN		
	50%	NaN	NaN		
	75%	NaN	NaN		
	max	NaN	NaN		
<hr/>					
designation		Libellé temporaire \			
uid					
0891b705-0b7d-43ea-bab3-9df5875b8a96	Morille spéciale en sachet 1 kg ROGER D				
731bf4e6-2380-4ce4-a859-0d084f3006cd	Fonds brun lié demi-glace en boîte 900				
d92860c1-c252-4aee-82fe-a471de719af6	Courgettes cuisinées à la provençale en				
ebbdbd6b-ec4e-46b1-acbb-79ba0b68c1c5	Barquette 2 compartiments bagasse 700 m				
5717f881-afad-4f96-9c47-a4b597077e5c	PAP TOIL MAX JUMB 350M BLC 2P PREDEC X6				
<hr/>					
	Désignation produit fournisseur \				
uid					
0891b705-0b7d-43ea-bab3-9df5875b8a96	Morille spéciale en sachet 1 kg ROGER DUTRY				
731bf4e6-2380-4ce4-a859-0d084f3006cd	Fonds Brun Lié Demi-Glace CHEF Boite de 900 g ..				
d92860c1-c252-4aee-82fe-a471de719af6	Courgettes cuisinées à la provençale en poche ..				
ebbdbd6b-ec4e-46b1-acbb-79ba0b68c1c5	Barquette 2 compartiments bagasse 700 ml en sa..				
5717f881-afad-4f96-9c47-a4b597077e5c	ECO 350				
<hr/>					
	Code interne fournisseur Marque commerciale \				
uid					
0891b705-0b7d-43ea-bab3-9df5875b8a96	702021.004	ROGER DUTRY			
731bf4e6-2380-4ce4-a859-0d084f3006cd	12066191	CHEF			
d92860c1-c252-4aee-82fe-a471de719af6	60792	BONDUELLE			
ebbdbd6b-ec4e-46b1-acbb-79ba0b68c1c5	VF38430	SOLIA			
5717f881-afad-4f96-9c47-a4b597077e5c	812118X	LUCART			
<hr/>					
uid	Dénomination réglementaire data_ok				

0891b705-0b7d-43ea-bab3-9df5875b8a96	Morille spéciale séchées 1er choix	False
731bf4e6-2380-4ce4-a859-0d084f3006cd	Fonds brun lié demi-glace déshydraté.	False
d92860c1-c252-4aee-82fe-a471de719af6	Mélange de légumes cuisinés	False
ebbd6b-ec4e-46b1-acbb-79ba0b68c1c5	Barquette pulpe de canne laminée 2 cpts 179x12...	True
5717f881-afad-4f96-9c47-a4b597077e5c	Papier toilette en jumbo ECO 350	True
<hr/>		
Libellé temporaire \		
data_ok		
False	count	8718
	unique	8072
	top	Eau minérale naturelle gazeuse en boute
	freq	23
True	count	4582
	unique	4404
	top	Barquette gastronomique thermoscellable e
	freq	13
<hr/>		
Désignation produit fournisseur Code interne fournisseur \		
data_ok		
False	count	8348
	unique	8110
	top	Cacao Barry Chocolate Pistoles
	freq	8
True	count	4582
	unique	4539
	top	Barquette gastronomique thermoscellable en sachet...
	freq	3
<hr/>		
Marque commerciale Dénomination réglementaire		
data_ok		
False	count	8332
	unique	1198
	top	CGMP
	freq	217
True	count	4540
	unique	954
	top	NEFF MADA Potage instantané déshydraté
	freq	295
<hr/>		

1.3.5 Analyses spécifiques : GTIN

On peut mettre en évidence les produits qui portent les mêmes GTIN en double. En y jetant un œil rapide, quelques explications peuvent être trouvées : - il peut s'agir d'un changement de code fournisseur (les 2 premières lignes ne portent pas le même code fournisseur) - il peut s'agir d'un changement de recette côté industriel, qui a décidé de conserver le même GTIN (second couple) - il peut s'agir d'une erreur, et de produits en doublon dans le système (troisième couple) - ...

```
[24]: GTIN_mask = (df['properties.pprod:gtin'] != '') & ~pd.isna(df['properties.pprod:gtin'])
dups_mask = df.loc[GTIN_mask, 'properties.pprod:gtin'].duplicated(keep=False)
examples = (df.loc[GTIN_mask & dups_mask, def_fields.keys()]
    .rename(def_fields, axis=1)
    .sort_values('gtin')
    .head(8))
# save_to_disk(
#     examples,
#     Path('..') / 'tbls' / 'Duplicated_GTIN.tex',
#     lab={},
#     tex_label='tab:dup_gtin',
# )
print(examples)
```

code	supplier	type	gtin	\
uid				
4de8ce87-8df5-440c-959d-3d77d59bb4f3	PIMP-0000013159	PIMF-0000000182	grocery	0020176760607
048712e3-f145-4f40-b8ad-7c0b912983bd	PIMP-0000009515	PIMF-0000000420	grocery	0020176760607
7e455046-def3-4526-a28b-bc5c0e6e64fc	PIMP-0000011456	PIMF-0000000290	grocery	03344540125906
a92c6ac5-d5be-4f92-98b3-9f6c588f7613	PIMP-0000013198	PIMF-0000000290	grocery	03344540125906
27e20042-dc53-46b4-874c-f970db554aec	PIMP-0000001494	PIMF-0000000250	grocery	3011360083845
66590f04-5eae-4829-b0da-c899a18dd9cb	PIMP-0000010839	PIMF-0000000250	grocery	3011360083845
02803e27-487a-43e3-9324-9ad1660b63b2	PIMP-0000002338	PIMF-0000000348	grocery	3038353024906
52d3f309-e402-4931-974c-b6b6fa721aff	PIMP-0000002337	PIMF-0000000348	grocery	3038353024906
<hr/>				
uid			designation	
4de8ce87-8df5-440c-959d-3d77d59bb4f3			QUICHE FEUILLETÉE	
048712e3-f145-4f40-b8ad-7c0b912983bd			42 QUICHE FEUILL SG 11CM	
7e455046-def3-4526-a28b-bc5c0e6e64fc			622028 SAUCE FUEGO SQUEEZE DE 580 G "O'TACOS"	
a92c6ac5-d5be-4f92-98b3-9f6c588f7613			622029 SAUCE FUEGO (NR) SQUEEZE DE 580 G "O'TA...	
27e20042-dc53-46b4-874c-f970db554aec			Jus de poulet en boîte 750 g KNORR	
66590f04-5eae-4829-b0da-c899a18dd9cb			Jus de poulet en boîte 750g KNORR	

02803e27-487a-43e3-9324-9ad1660b63b2

Torti aux œufs en sac 5 kg PANZANI
52d3f309-e402-4931-974c-b6b6fa721aff Tagliatelle aux œufs en colis 5 kg PANZANI

Si l'on produit la répartition du nombre de produit portant un GTIN donné dans le système, on obtient :

```
[25]: df2 = (df.pivot_table(values='properties.vig:code',
                           index='properties.pprod:gtin',
                           aggfunc='count')
            .rename({'properties.vig:code': 'code_count'}, axis=1)
            )

df2 = (df2.reset_index()
       .loc[df2.index != '']
       .pivot_table(index='code_count',
                    aggfunc='count',
                    values='code_count')
       .rename({'properties.pprod:gtin': 'Nb de GTIN'},
              axis=1)
       )

df2.index.rename('Pdt portant le GTIN', inplace=True)

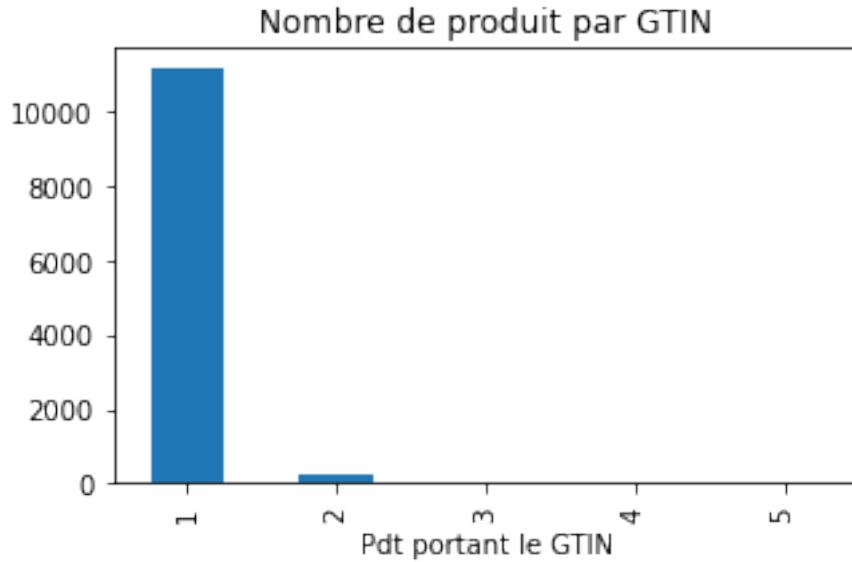
print(df2)

# save_to_disk(df2,
#               Path('..') / 'tbls' / 'gtin_counts.tex')

fig, ax = plt.subplots(figsize=(5,3))
df2.plot(kind='bar', legend=None, title='Nombre de produit par GTIN', ax=ax)
# fig.savefig(Path('..') / 'img' / 'repartition_gtin.png', bbox_inches='tight')
```

Pdt portant le GTIN	Nb de GTIN
1	11134
2	230
3	21
4	20
5	1

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f22d21100a>



1.3.6 Analyse spécifique : distribution par fournisseur

On peut représenter la distribution produit, par fournisseur.

```
[27]: # construction the counts
counts = (df.loc[:, list(def_fields.keys())]
```

```

    .rename(def_fields, axis=1)
    .pivot_table(values='code',
                 index='supplier',
                 aggfunc='count')
    .reset_index()
    .pivot_table(values=['supplier', 'code'],
                 index='code',
                 aggfunc={'supplier': 'count',
                           'code': 'sum'})
)
# aligning index to have it continuous
new_idx = pd.RangeIndex(start=1, stop=max(counts.index) + 1)

counts = (counts.reindex(new_idx)
          .fillna(0)
)

counts = pd.concat([counts,
                    counts.cumsum().rename({'code': 'cum_code', 'supplier': 'cum_supplier'},
                                           axis=1),
                    ],
                   axis=1)

for feature in ['supplier', 'code']:
    counts['cump_' + feature] = 100 * counts['cum_' + feature] / counts.loc[:, 'cum_' + feature].iloc[-1]

counts.head(5)

```

[27]:

	code	supplier	cum_code	cum_supplier	cump_supplier	cump_code
1	80.0	80.0	80.0	80.0	13.114754	0.601504
2	138.0	69.0	218.0	149.0	24.426230	1.639098
3	180.0	60.0	398.0	209.0	34.262295	2.992481
4	188.0	47.0	586.0	256.0	41.967213	4.406015
5	180.0	36.0	766.0	292.0	47.868852	5.759398

[28]:

```

fig, axs = plt.subplots(nrows=2,
                       ncols=2,
                       figsize=(14, 6),
                       gridspec_kw={'width_ratios': [2, 1]})

axs2 = [[ax.twinx() for ax in axrow] for axrow in axs]

for i, feature in enumerate(['supplier', 'code']):
    axs[i][0].bar(data=counts.loc[:, feature].reset_index(), x='index', height=feature)
    axs2[i][0].plot('index', 'cump_' + feature, data=counts.loc[:, 'cump_' + feature].reset_index(),
                    color='red', linestyle='--')
    axs2[i][0].grid(True, axis='y', color='red', alpha=0.5, linestyle='--')

    axs[i][1].bar(data=counts.loc[:, feature].reset_index(), x='index', height=feature)
    axs2[i][1].plot('index', 'cump_' + feature, data=counts.loc[:, 'cump_' + feature].reset_index(),
                    color='red', linestyle='--')
    axs2[i][1].grid(True, axis='y', color='red', alpha=0.5, linestyle='--')

    axs[i][1].set_xlim(0, 30)

for i in range(len(axs)):
    for j in range(len(axs[i])):
        axs2[i][j].set_ylim(0, 100)
        # remove all bottom ticks except for bottom line
        # set_yticks does not work as it removes the grid
        if i < len(axs) - 1:
            axs[i][j].set_xticklabels([])
            for tic in axs[i][j].xaxis.get_major_ticks():
                tic.tick1line.set_visible(False)
                tic.tick2line.set_visible(False)
        # remove all right ticks except for right column
        # set_yticks does not work as it removes the grid
        if j < len(axs[i]) - 1:
            axs2[i][j].set_yticklabels([])
            for tic in axs2[i][j].yaxis.get_major_ticks():
                tic.tick1line.set_visible(False)
                tic.tick2line.set_visible(False)
        # remove all left ticks except for first column

```

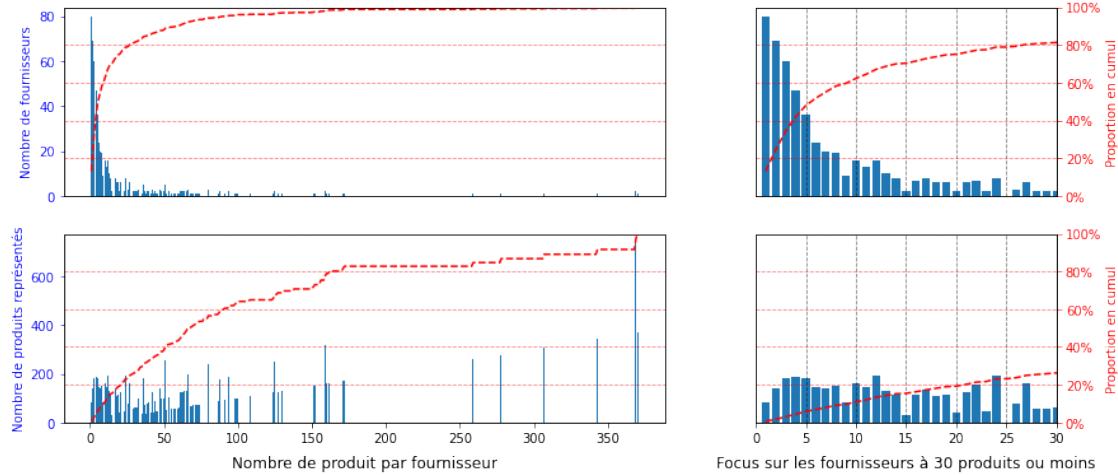
```

if j > 0:
    axes[i][j].set_yticks([])
if j == len(axes[i]) - 1:
    axes2[i][j].tick_params(axis='y', colors='red')
    axes2[i][j].yaxis.set_major_formatter(ticker.PercentFormatter())
    axes2[i][j].set_ylabel('Proportion en cumul', color='red')
    axes[i][j].grid(True, axis='x', color='k', alpha=0.5, linestyle='--')
if j == 0:
    axes[i][j].tick_params(axis='y', colors='blue')
    if i == 0:
        axes[i][j].set_ylabel('Nombre de fournisseurs', color='blue')
    if i == 1:
        axes[i][j].set_ylabel('Nombre de produits représentés', color='blue')
axes[1][0].set_xlabel('Nombre de produit par fournisseur',
                      fontsize=12,
                      labelpad=8,
                      )
axes[1][1].set_xlabel('Focus sur les fournisseurs à 30 produits ou moins',
                      fontsize=12,
                      labelpad=8,
                      )
fig.suptitle('Distribution des fournisseurs fonction du nombre de produit par fournisseur',
             fontsize=16,
             )
# fig.savefig(Path('..') / 'img' / 'distribution_fournisseurs_par_prd_count.png', bbox_inches='tight')

```

[28]: Text(0.5, 0.98, 'Distribution des fournisseurs fonction du nombre de produit par fournisseur')

Distribution des fournisseurs fonction du nombre de produit par fournisseur



On peut également représenter le nombre de produits ``récupérés'' si on prend les fournisseurs par nombre de produits décroissant.

```

[30]: # construction the counts
counts = (df.loc[:, list(def_fields.keys())]
          .rename(def_fields, axis=1)
          .pivot_table(values='code',
                      index='supplier',
                      aggfunc='count')
          .sort_values('code', ascending=False)
          )
counts['code_cumsum'] = counts['code'].cumsum()
counts.head(5)

```

[30]:

	<code>code</code>	<code>code_cumsum</code>
supplier		

PIMF-0000000179	370	370
PIMF-0000000250	369	739
PIMF-0000000283	369	1108
PIMF-0000000074	343	1451
PIMF-0000000392	307	1758

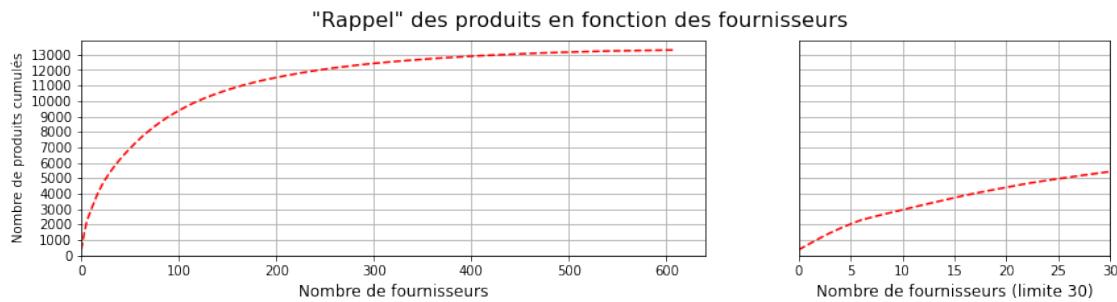
```
[31]: fig, axs = plt.subplots(nrows=1,
                           ncols=2,
                           figsize=(14, 3),
                           gridspec_kw= {'width_ratios': [2, 1]})

for j in range(len(axs)):
    axs[j].plot('index',
                'code_cumsum',
                data=counts.reset_index().reset_index(),
                color='red',
                linestyle='--',
                )
    axs[j].set_xlabel('Nombre de fournisseurs', fontsize=12)
    axs[j].set_xlim(0)
    axs[j].set_xlim(0)
    axs[j].grid(True)
    axs[j].yaxis.set_ticks(np.arange(0, 14000, 1000))

axs[0].set_ylabel('Nombre de produits cumulés')
axs[1].set_xlim(0, 30)
axs[1].set_xlabel('Nombre de fournisseurs (limite 30)', fontsize=12)
axs[1].set_yticklabels([])
for tic in axs[1].yaxis.get_major_ticks():
    tic.tick1line.set_visible(False)
    tic.tick2line.set_visible(False)

fig.suptitle('"Rappel" des produits en fonction des fournisseurs', fontsize=16)
# fig.savefig(Path('..') / 'img' / 'rappel_produit_par_fournisseur.png', bbox_inches='tight')
```

[31]: `Text(0.5, 0.98, '"Rappel" des produits en fonction des fournisseurs')`



1.3.7 Export de quelques désignations et listes d'ingrédients

On exporte quelques désignations pour illustrer.

```
[33]: df_des = df_dict['designation']
export_df = (df_des.loc[df_des['data_ok'],
                       ['temp_des',
                        'supplier_des',
                        'supplier_code',
                        'brand',
                        'regulatory_des']]
             .rename(lab, axis=1)
             .sample(5, random_state=40))
)
export_df

# Do not overwrite current saved file!
# It has been corrected manually (midrule was not well placed)
#export_df.to_latex(Path('..') / 'tbls' / 'designation_example.tex',
#                   index=False,
#                   column_format='p{4cm}p{4cm}p{2cm}p{2cm}p{3cm}',
#                   index_names=False,
#                   )
```

uid	Libellé temporaire	Désignation produit fournisseur	Code interne fournisseur	Marque commerciale	Dénomination réglementaire
d76f0535-2a1c-40a4-abcc-9eb132cc855c	Riz long étuvé pour risotto et paëlla e	Riz long étuvé pour risotto et paëlla en sac 2...	3039820204104	VIVIEN PAILLE	Riz long étuvé de qualité supérieure
adbcf49f-fcfc-47dd-8339-b6d793524d6f	Compote de pomme-fraise allégée en berl	COMPOTE ALLÉGÉE EN SUCRES POMME FRAISE BERLING...	42033	SAINT MAMET	Compote de pomme fraise allégée en sucre *\\n...
843066ef-9fb8-491d-8c96-2d27846f1335	Boîte avec couvercle 450 cc en sachet d	BOITE COUVERCLE CHALEUR 450 CC	RC450	ALPHA FORM	BOITE COUVERCLE CHALEUR 450 CC
84bc6bf8-4ca5-4d5c-94e5-d4e9243cd0f	Café soulu 50% arabica, 50 % robusta en	FILTRE DOSE MIKO REST CD 12X465G - 50X4-50VA	509223	MIKO	CAFE MOULU FILTRE DOSE MIKO R. COLLECTIVE
Sae42ed5-454f-4301-9dea-394e64734cf9	Bouillon de légume hyposodé en boite 80	Bouillon de légume hyposodé en boite 800 g NEF...	7089	NEFF MADA	Bouillon déshydraté

```
[36]: ingred_mask = df['data_ok'] & ~pd.isna(df['properties.pprodः:ingredientsList'])

sample_ingred = (df.loc[ingred_mask, ['properties.pprodः:supplierDesignation',
                                       'properties.pprodः:ingredientsList']]
                  .rename(lab, axis=1)
                  .sample(10, random_state=42))

with pd.option_context("max_colwidth", 1000):
    pass
    #    print(sample_ingred)
    #    sample_ingred.to_latex(Path('..') / 'tbls' / 'ingredient_example.tex',
    #                           index=False,
    #                           index_names=False,
    #                           column_format='p{5cm}p{10cm}',
    #                           )
sample_ingred
```

uid	Désignation produit fournisseur	Liste d'ingrédients
0635f409-5543-4260-8d67-b55cf4679d30	Pâte d'arachide en boite 4/4 DAKATINE	97,5% ARACHIDES grillées, émulsifiant d'origine...
9e4e4b6e-4bf9-4950-8427-23ee82527269	Madeleine pur beurre	BEURRE 26%, OEUFS frais 25 %, farine de BLE te...
c40d1993-69a2-4500-86ee-a346ac6a921c	Coca-cola en bouteille verre 25 cl COCA-COLA	Eau gazeifiée, sucre, colorant : E150d, acidif...
e564f0be-fa60-a00d-b663-8be406f0d62	Vinaigre balsamique de Modène en bouteille ver...	Vinaigre de vin, moût de raisin cuit, colorant...
69cab499-8541-424e-9fe4-a2389c5f18a3	Carottes râpées en boite 5/1 EPISAVEURS	Carottes, eau, sel, acidifiants (acide acétiqu...
a477326e-38a0-4643-973f-1bacdefa0a70	CÔTES DU RHÔNE ROUGE AOC EN BOUTEILLE 25 CL LE...	VIN contient des SULFITES
1f9e6e3c-fdb3-492d-a142-61c247887a78	STICKS STEVIA 0 CALORIE DADDY 60 GR	Agent de charge: érythritol, édulcorant: glyco...
5810e22e-01bb-414d-b3dd-db1a29edcbcd	JUS DE PAMPLEMOUSSE EN BRUIQUE 1L BRICI	Jus de pamplemousse à base de jus de pamplemou...
fdec3861-1607-49a5-a396-d74d7fd51273	Jus de pomme en briquette 20 cl PLEIN FRUIT	Jus de pomme à base de concentré
c3e4164b-d589-4d0e-a657-773bd22882f4	riz rond en sac 5 kg TAUREAU AILE	100% Riz rond de qualité supérieure

1.4 Analyse de la complétude des pièces jointes

On jette un oeil au niveau de renseignement des pièces jointes dans le PIM.

```
[38]: file_df = requester.file_report_from_result(mapping={'uid': 'uid'}, index='uid', record_path='entries')
file_df.sample(5)
```

uid	has_supplierdatasheet	has_supplierlabel
Oec61fa6-238f-4383-b175-7686edaf9bbb	True	True
1748bfc2-e1b1-4093-9390-6418f76047f5	True	True
c8f655ce-3efa-4e4c-840a-b213b38cccc6e	True	True
4f3c1903-f883-4f24-a9b9-894408dcfb2d	True	False
cacdd8af-8814-47ef-830d-f463eb8fd04f	True	True

```
[39]: file_df = file_df.join(df)

[41]: total_df = file_df.groupby('properties.pprodtop:typeOfProduct').size().to_frame(name='total')
new_idx = pd.MultiIndex.from_tuples([('total', 'total')])
total_df.columns = new_idx
print(total_df)

      total
      total
properties.pprodtop:typeOfProduct
alcoholicDrink          607
chemistry                467
grocery                 8793
hygiene                  2526
nonAlcoholicDrink        907

[42]: report_df = (
    file_df.pivot_table(values=['has_supplierdatasheet', 'has_supplierlabel'],
                         columns=['data_ok'],
                         index=['properties.pprodtop:typeOfProduct'],
                         aggfunc=['sum', 'count'],
                         margins=True,
                         )
)
report_df2 = report_df.swaplevel(axis=1, i=0, j=2).sort_index(axis=1).stack([0, 1])
report_df2['percent'] = report_df2['sum'] / report_df2['count']
report_df3 = report_df2.unstack(level=[-2, -1]).swaplevel(axis=1, i=0, j=1).sort_index(axis=1)
report_df3.columns.rename(['data_ok', 'attached', 'func'], inplace=True)
report_df3

[42]:
```

properties.pprodtop:typeOfProduct	False			True			All		
	has_supplierdatasheet	count	percent	has_supplierlabel	count	percent	has_supplierdatasheet	count	percent
All	8718	0.845033	7367.0	8718	0.323239	2818.0	4582	1.0	4582.0
alcoholicDrink	353	0.835694	295.0	353	0.206799	73.0	254	1.0	254.0
chemistry	329	0.936170	308.0	329	0.389058	128.0	138	1.0	138.0
grocery	5742	0.832637	4781.0	5742	0.344131	1976.0	3051	1.0	3051.0
hygiene	1743	0.872633	1521.0	1743	0.253586	442.0	783	1.0	783.0
nonAlcoholicDrink	551	0.838475	462.0	551	0.361162	199.0	356	1.0	356.0


```
[43]: type_index = pd.Index(['grocery', 'nonAlcoholicDrink', 'alcoholicDrink', 'hygiene', 'chemistry', 'All'], name='type')
idxs = [[True, False, 'All'], ['has_supplierdatasheet', 'has_supplierlabel'], ['sum', 'count', 'percent']]
col_index = pd.MultiIndex.from_product(idxs, names=['data_ok', 'attached', 'func'])
report_df4 = report_df3.reindex(col_index, axis=1).reindex(type_index)
report_df4
```



```
[43]:
```

type	True			False			All		
	has_supplierdatasheet	sum	count	has_supplierlabel	sum	count	has_supplierdatasheet	sum	count
grocery	3051.0	3051	1.0	3040.0	3051	0.996395	4781.0	5742	0.822637
nonAlcoholicDrink	356.0	356	1.0	353.0	356	0.991573	462.0	551	0.838475
alcoholicDrink	254.0	254	1.0	254.0	254	1.000000	295.0	329	0.835694
hygiene	783.0	783	1.0	702.0	783	0.896552	1521.0	1743	0.872633
chemistry	138.0	138	1.0	138.0	138	1.000000	308.0	329	0.936170
All	4582.0	4582	1.0	4487.0	4582	0.979267	7367.0	8718	0.323239


```
[44]: ### WARNING !!! This tex export requires some manual adjustments to have
### multiindex headers centered !!!

(
report_df4.rename({'has_supplierdatasheet': 'Fiche technique',
                   'has_supplierlabel': 'Etiquette',
                   'True': 'En qualité',
                   'False': 'Hors qualité',
                   'All': 'Total',
                   'sum': 'cpt',
                   'count': 'sur',
                   'percent': '%',
                   }, axis=1)
.rename({'grocery': 'Epicerie',
         'nonAlcoholicDrink': 'Boissons',
         'alcoholicDrink': 'Alcools',
         'hygiene': 'Hygiène',
         'chemistry': 'Chimie',
         'All': 'Total'})
.to_latex(
    #Path('..') / 'tbls' / 'attached_files_counts.tex',
    column_format='|l|ccc|ccc|ccc|ccc|ccc|',
    bold_rows=True,
    index_names=False,
    formatters=[lambda x: str(int(x)),
```

```
[44]: '\\begin{tabular}{|l|ccc|ccc|ccc|ccc|ccc|ccc|}\n\\toprule\n{} & \\multicolumn{6}{l}{En qualité} &\n\\multicolumn{6}{l}{Hors qualité} & \\multicolumn{6}{l}{Total} \\\\n{} & \\multicolumn{3}{l}{Fiche technique} &\n\\multicolumn{3}{l}{Etiquette} & \\multicolumn{3}{l}{Fiche technique} & \\multicolumn{3}{l}{Etiquette}\n\\\\n{} & cpt & sur & \\% & cpt & sur & \\% & cpt & sur & \\% & cpt & sur & \\%\n\\\\n\\midrule\n\\textbf{Epicerie} & 3051 & 3051 & 100\\% & 3040 & 3051 & 99\\% &\n4781 & 5742 & 83\\% & 1976 & 5742 & 34\\% & 7832 & 8793 & 89\\% & 5016 & 8793 & 57\\%\n\\\\n\\textbf{Boissons} & 356 & 356 & 100\\% & 353 & 356 & 99\\% & 462 &\n551 & 83\\% & 199 & 551 & 36\\% & 818 & 907 & 90\\% & 552 & 907 & 60\\%\n\\\\n\\textbf{Alcools} & 254 & 254 & 100\\% & 254 & 254 & 100\\% & 295 &\n353 & 83\\% & 73 & 353 & 20\\% & 549 & 607 & 90\\% & 327 & 607 & 53\\%\n\\\\n\\textbf{Hygiène} & 783 & 783 & 100\\% & 702 & 783 & 89\\% & 1521 &\n1743 & 87\\% & 442 & 1743 & 25\\% & 2304 & 2526 & 91\\% & 1144 & 2526 & 45\\%\n\\\\n\\textbf{Chimie} & 138 & 138 & 100\\% & 138 & 138 & 100\\% & 308 &\n329 & 93\\% & 128 & 329 & 38\\% & 446 & 467 & 95\\% & 266 & 467 & 56\\%\n\\\\n\\textbf{Total} & 4582 & 4582 & 100\\% & 4487 & 4582 & 97\\% & 7367 &\n8718 & 84\\% & 2818 & 8718 & 32\\% & 11949 & 13300 & 89\\% & 7305 & 13300 & 54\\%\n\\\\n\\bottomrule\n\\end{tabular}'
```

1.5 Analyse de la complétude des listes d'ingrédients

On peut aussi faire l'analyse de la complétude des listes d'ingrédients.

```
[50]: df['has_ingredients'] = ~df['properties.pprodc:ingredientsList'].isna()
```

```
[63]: report_df = (
    df.pivot_table(values=['has_ingredients'],
                  columns=['data_ok', ],
                  index=['properties.pprodtop:typeOfProduct'],
                  aggfunc=['sum', 'count'],
                  margins=True,
                  )
)
report_df2 = report_df.swaplevel(axis=1, i=0, j=2).sort_index(axis=1).stack([0, 1])
report_df2['percent'] = report_df2['sum'] / report_df2['count']
report_df2
report_df3 = report_df2.unstack(level=[-2, -1]).swaplevel(axis=1, i=0, j=1).swaplevel(axis=1, i=2, j=1).sort_index(axis=1)
report_df3.columns.rename(['data_ok', 'attached', 'func'], inplace=True)
report_df3.columns = report_df3.columns.droplevel(level=1)
type_index = pd.Index(['grocery', 'nonAlcoholicDrink', 'alcoholicDrink', 'hygiene', 'chemistry', 'All'], name='type')
idxs = [[True, False, 'All'], ['sum', 'count', 'percent']]
col_index = pd.MultiIndex.from_product(idxs, names=['data_ok', 'func'])
report_df4 = report_df3.reindex(col_index, axis=1).reindex(type_index)
report_df4
```

[63] :

data_ok	True			False			All		
	sum	count	percent	sum	count	percent	sum	count	percent
grocery	3051.0	3051	1.000000	5317.0	5742	0.925984	8368.0	8793	0.951666
nonAlcoholicDrink	356.0	356	1.000000	504.0	551	0.914701	860.0	907	0.948181
alcoholicDrink	254.0	254	1.000000	259.0	353	0.733711	513.0	607	0.845140
hygiene	0.0	783	0.000000	0.0	1743	0.000000	0.0	2526	0.000000
chemistry	0.0	138	0.000000	3.0	329	0.009119	3.0	467	0.006424
All	3661.0	4582	0.798996	6083.0	8718	0.697752	9744.0	13300	0.732632

[64]: *### WARNING !!! This tex export requires some manual adjustments to have
multiindex headers centered !!!*

```
(report_df4.rename({'has_supplierdatasheet': 'Fiche technique',
                   'has_supplierlabel': 'Etiquette',
                   'True': 'En qualité',
                   'False': 'Hors qualité',
                   'All': 'Total',
                   'sum': 'cpt',
                   'count': 'sur',
                   'percent': '%',
                   }, axis=1)
.rename({'grocery': 'Epicerie',
        'nonAlcoholicDrink': 'Boissons',
        'alcoholicDrink': 'Alcools',
        'hygiene': 'Hygiène',
        'chemistry': 'Chimie',
        'All': 'Total'})
.to_latex(
    #Path('..') / 'tbls' / 'ingredients_counts.tex',
    column_format='|l|ccc|ccc|ccc|',
    bold_rows=True,
    index_names=False,
    formatters=[lambda x: str(int(x)),
               lambda x: str(int(x)),
               lambda x: f'{int(x * 100):d}%',
               lambda x: str(int(x)),
               lambda x: str(int(x)),
               lambda x: f'{int(x * 100):d}%',
               lambda x: str(int(x)),
               lambda x: str(int(x)),
               lambda x: f'{int(x * 100):d}%',
               ]
)
)

(
report_df3.rename({'has_supplierdatasheet': 'Fiche technique',
                   'has_supplierlabel': 'Etiquette',
                   'True': 'En qualité',
                   'False': 'Hors qualité',
                   'All': 'Total',
                   'sum': 'cpt',
                   'count': 'sur',
                   'percent': '%',
                   }, axis=1)
.rename({'grocery': 'Epicerie',
        'nonAlcoholicDrink': 'Boissons',
        'alcoholicDrink': 'Alcools',
        'hygiene': 'Hygiène',
        'chemistry': 'Chimie',
        'All': 'Total'})
)
```

[64]:

data_ok	Hors qualité			En qualité			Total		
	sur	%	cpt	sur	%	cpt	sur	%	cpt
Total	8718	0.697752	6083.0	4582	0.798996	3661.0	13300	0.732632	9744.0
Alcools	353	0.733711	259.0	254	1.000000	254.0	607	0.845140	513.0
Chimie	329	0.009119	3.0	138	0.000000	0.0	467	0.006424	3.0
Epicerie	5742	0.925984	5317.0	3051	1.000000	3051.0	8793	0.951666	8368.0
Hygiène	1743	0.000000	0.0	783	0.000000	0.0	2526	0.000000	0.0
Boissons	551	0.914701	504.0	356	1.000000	356.0	907	0.948181	860.0

Annexe H

LE CODE DES DIFFÉRENTS MODULES

H.1 Gestion du fichier de configuration - Module conf

Ce petit module a pour but de permettre de gérer les paramètres du programme dans un fichier de configuration (afin de simplifier la maintenance). Il est utilisé dans l'ensemble des autres modules de ce projet.

```
"""Configuration Module
This module provides a class that enables to fetch configuration stored in the
cfg folder and make it available to other python modules.
"""

import yaml
import os

class Config:
    def __init__(self, env):
        if env not in {'dev', 'int', 'rec', 'qat', 'prd'}:
            raise ValueError(f'Specified env : {env} not expected')
        self.env = env
        self.path = os.path.join(os.path.dirname(__file__), 'cfg', 'config_local.yaml')
        stream = open(self.path, 'r')
        data = yaml.safe_load(stream)
        config_keys = data['cross-env'].keys()
        for k in config_keys:
            setattr(self, k, data['cross-env'][k])
        config_keys = data[env].keys()
        for k in config_keys:
            setattr(self, k, data[env][k])
```

Un exemple de fichier de configuration (dont certains champs ont été anonymisés pour des raisons de confidentialité) est présenté ci-dessous.

```
cross-env:
    suffixid: nuxeo/api/v1/id/
    uiddirectory: directory.csv
    rootuid: 00000000-0000-0000-0000-000000000000
    maxpage: -1
    pagesize: 1000
    filedefs:
        supplierdatasheet:
            nuxeopath:
                - properties
                - pprodad:technicalSheet
        supplierlabel:
            nuxeopath:
                - properties
                - pprodad:label
    apikeys:
        azure: <anonymise>
```

```

google: <anonymise>

dev:
  baseurl: http://devpim/
  password: <anonymise>
  user: <anonymise>

int:
  baseurl: http://intpim/
  password: <anonymise>
  user: <anonymise>

rec:
  baseurl: https://rec-produits.groupe-pomona.fr/
  proxies:
    http: http://redhat:<anonymise>@pomproxy:8080
    https: http://redhat:<anonymise>@pomproxy:8080
  password: <anonymise>
  user: consult

qat:
  baseurl: None
  password: None
  user: None

prd:
  baseurl: https://produits.groupe-pomona.fr/
  proxies:
    http: http://redhat:<anonymise>@pomproxy:8080
    https: http://redhat:<anonymise>@pomproxy:8080
  password: <anonymise>
  user: consult

```

H.2 Extraction des données du PIM - Module pimapi

```

"""PIM API Module
This module aims to enable to fetch data from PIM system, into local folders.

import requests
import os
import json
import warnings
import pandas as pd
import numpy as np
import copy
import threading
from requests.exceptions import ConnectionError
from . import conf

warnings.simplefilter('always', UserWarning)

class Requester(object):
    """Requester class to retrieve information from PIM
    """
    def __init__(self, env, proxies='default', auth=None):
        self.cfg = conf.Config(env)
        self.session = requests.Session()
        if not auth:
            self.session.auth = (self.cfg.user, self.cfg.password)
        else:
            self.session.auth = auth
        if proxies == 'default':
            try:

```

```

        target_proxies = self.cfg.proxies
    except (AttributeError):
        print(f'No proxy conf found for env : \'{self.cfg.env}\'.')
        f'No proxy will be used.')
    target_proxies = None
else:
    target_proxies = proxies
self.session.proxies = target_proxies
if proxies == 'default':
    # We try the connection with the default proxy conf
    # If it fails, we retry with no proxy configuration
    # (e.g. in the case of working from outside the network)
    try:
        self.check_connection()
    except (ConnectionError):
        self.session.proxies = None
        self.check_connection()
    else:
        # If a proxy configuration has been passed as an argument, we
        # only validate the connexion with that configuration.
        self.check_connection()
self.rlock = threading.RLock()
self.result = []
try:
    self._load_directory()
except FileNotFoundError:
    warnings.warn('No directory found for current env : '
                  f' \'{self.cfg.env}\'. A new directory should be '
                  'set.')
def check_connection(self):
    """Checks wether the connection with the environments works

    The methods tries to get content of PIM system homepage. It returns
    True if the request is handled properly, or raises an exception.
    Note: this method does NOT check whether credentials are correct.
    """
    resp = self.session.get(self.cfg.baseurl)
    if resp.status_code != 200:
        raise ConnectionError('Connection could not be validated during '
                              'check_connection method call for '
                              f'environment : \'{self.cfg.env}\'')

    return(True)

def check_credentials(self):
    """Checks wether the credentials provided allow to connect to PIM

    The methods tries to get content of PIM root document. It returns
    True if the request is handled properly, or raises an exception.
    Note: if the connection is invalid for another reason (e.g. incorrect
    proxy configuration), an exception will be raised. the
    'check_connection' method should be used prior to checking the
    credentials.
    """
    resp = self.session.get(self.cfg.baseurl +
                           self.cfg.suffixid +
                           self.cfg.rootuid)
    if resp.status_code != 200:
        raise ConnectionError('Connection could not be validated during '
                              'check_credentials method call for '
                              f'environment : \'{self.cfg.env}\'')

    return(True)

def fetch_all_from_PIM(self, nx_properties='*', max_page=None,
                      page_size=None):
    """Fetches all product data from PIM into result

    This method fetches all product data from PIM. It implements
    multithreading to speed up retrieval.
    """
    query = (f"SELECT * "
             "FROM Document "
             "WHERE ecm:primaryType='pomProduct' "
             "AND ecm:isVersion='0'")
    headers = {'Content-Type': 'application/json',
               'X-NXproperties': nx_properties}
    params = {'query': query}
    url = (self.cfg.baseurl +
           self.cfg.suffixid +
           self.cfg.rootuid + '/'+
           '@search')
    result_count = self.query_size(headers, params, url)
    self.result = []
    max_page = max_page if max_page else self.cfg.maxpage
    page_size = page_size if page_size else self.cfg.pagesize
    params['pageSize'] = page_size
    thread_count = result_count // page_size + 1
    if max_page != -1 and thread_count > max_page:
        thread_count = max_page
        warnings.warn(f'\nMax size reached ! \n'
                      f'Only {max_page * page_size} results will be '
                      f'fetched out of {result_count} results\n')
    threads = []
    for page_index in range(thread_count):
        t = threading.Thread(target=self.get_page_from_query,
                             args=(url, headers, params, page_index))
        t.start()
        threads.append(t)
    for t in threads:
        t.join()
    print('Done')

def get_page_from_query(self, url, headers, params, currentPageIndex=0):
    """Fetches data for a single page of results

    This methods fetches data for a prepared query for a single page. It
    is used to implement multithreading for 'fetch_all_from_PIM'
    """
    try:
        local_params = copy.deepcopy(params)
        local_params['currentPageIndex'] = currentPageIndex
        resp = self.session.get(url,
                               headers=headers,
                               params=local_params)
        with self.rlock:
            self.result.append(resp)
    except Exception as e:
        print(f'An error occurred in this thread! {e}')
        print(e)

def fetch_list_from_PIM(self, iter_uid, batch_size=50, nx_properties='*'):
    """Fetches data from an uid iterable, from PIM

    This method fetches the data from PIM based on a iterable of uids.
    It creates threads to fetch the complete list, and bases on the
    Nuzeo @search API with a WHERE clause. It may therefore be less
    fast than a full fetch.
    Due to http limitations, it requires that no more than 50 uid be
    retrieved at a time in a single thread. Failing to enforce will result
    in incomplete responses an missed results.
    """
    if batch_size > 50:
        raise ValueError(f'batch_size needs to be < 50. '
                         f'Call with:(batch_size)')
    query = (f"SELECT * "
             "FROM Document "
             "WHERE ecm:primaryType='pomProduct' "
             "AND ecm:isVersion='0'")
    headers = {'Content-Type': 'application/json',
               'X-NXproperties': nx_properties}
    url = (self.cfg.baseurl +
           self.cfg.suffixid +
           self.cfg.rootuid + '/'+
           '@search')
    uid_lists = [iter_uid[i:i+batch_size]
                 for i in range(0, len(iter_uid), batch_size)]
    thread_list = []
    self.result = []
    for uid_list in uid_lists:
        uid_string = ', '.join(['"' + str(uid) + '"' for uid in uid_list])
        local_query = query + f" AND ecm:uid in ({uid_string})"
        params = {'query': local_query}
        t = threading.Thread(target=self.get_page_from_query,
                             args=(url, headers, params))
        t.start()
        thread_list.append(t)
    for thread in thread_list:
        thread.join()
    print('Done')

def query_size(self, headers, params, url):
    """Runs a single result query to get the number of results

    This methods takes a query to be sent to Nuzeo in order to count the
    number of results.
    It then execute the query with page_size=1 and max_page=1 to fetch a
    single result, and extracts the total number of results from the
    response.
    """
    loc_headers = copy.deepcopy(headers)
    loc_headers['X-NXproperties'] = ''
    loc_params = copy.deepcopy(params)
    loc_params['pageSize'] = 1
    loc_params['currentPageIndex'] = 0
    resp = self.session.get(url,
                           headers=loc_headers,
                           params=loc_params)
    return(resp.json()['resultsCount'])

def _root_path(self):
    """Returns the root path for the dumps
    """
    return(os.path.join(os.path.dirname(__file__),
                       '..',
                       'dumps',
                       self.cfg.env))

def dump_data_from_result(self, filename='data.json',
                          update_directory=True, root_path=None):
    """Dumps data from result attribute as JSON files

    This method dumps data from result as JSON files.
    Note that result MUST be an iterable of responses (be it from PIM or
    from disk), each response should have an 'entries' list of documents.
    """
    if update_directory and not hasattr(self, '_directory'):
        self._load_directory()
    now = pd.Timestamp.now(tz='UTC')
    threads = []
    for single_result in self.result:
        t = threading.Thread(target=self.dump_data_from_single_result,
                             args=(single_result, filename, now),
                             kwargs={'update_directory': update_directory,
                                     'root_path': root_path})
        t.start()
        threads.append(t)
    for t in threads:
        t.join()
    print('Done')

def dump_data_from_single_result(self, single_result, filename, now,
                                 update_directory=True, root_path=None):
    """Dumps data from a single result

    This method dumps data from a single result passed as an argument.
    It is used for multithreading the result list.
    """

```

```

"""
try:
    doc_list = single_result.json()['entries']
    s_list = []
    if root_path is None:
        root_path = self._root_path()
    for document in doc_list:
        path = os.path.join(root_path, document['uid'])
        if not os.path.exists(path):
            os.makedirs(path)
        full_path = os.path.join(path, filename)
        with open(full_path, 'w+') as outfile:
            json.dump(document, outfile)
        s_list.append(pd.Series(now,
                               index=[document['uid']],
                               name='lastFetchedData'))
    df = pd.concat(s_list, axis=0)
    if update_directory:
        with self.rlock:
            self._directory.update(df)
            self._save_directory()
except Exception as e:
    print('An error occurred in this thread!')
    print(e)

def dump_files_from_result(self, update_directory=True, root_path=None):
    """Dumps attached files from result items on disk

This method dumps files from PIM on disk.
It also updates the local directory with current datetime.
Note that result MUST be an iterable of responses (be it from PIM or
from disk), each response should have an 'entries' list of documents,
and each document mention the files attached to it.
Attached files definition MUST be set in the config.yaml file.
"""
if update_directory and not hasattr(self, '_directory'):
    self._load_directory()
now = pd.Timestamp.now(tz='UTC')
threads = []
print(f'Launching {len(self.result)} threads.')
for single_result in self.result:
    t = threading.Thread(target=self.dump_files_from_single_result,
                          args=(single_result, now),
                          kwargs={'update_directory': update_directory,
                                  'root_path': root_path})
    t.start()
    threads.append(t)
for t in threads:
    t.join()
print('Done')

def dump_files_from_single_result(self, single_result, now,
                                  update_directory=True, root_path=None):
    """Dumps attached files from items on disk - for a single result

This method dumps files from a single result into the disk. It is used
for multithreading in 'dump_files_from_result' method.
"""
try:
    doc_list = single_result.json()['entries']
    s_list = []
    if root_path is None:
        root_path = self._root_path()
    for document in doc_list:
        path = os.path.join(root_path, document['uid'])
        if not os.path.exists(path):
            os.makedirs(path)
        self.dump_attached_files(document, path)
        s_list.append(pd.Series(now,
                               index=[document['uid']],
                               name='lastFetchedFiles'))
    df = pd.concat(s_list, axis=0)
    if update_directory:
        with self.rlock:
            self._directory.update(df)
            self._save_directory()
    print('Thread complete!')
except Exception as e:
    print('An error occurred in this thread!')
    print(e)

def _dump_file(self, file_url, path, filename='file'):
    """Dumps a file on disk from its url"""
    resp = self.session.get(file_url,
                           auth=(self.cfg.user, self.cfg.password),
                           stream=True)
    full_path = os.path.join(path, filename)
    with open(full_path, 'wb') as outfile:
        outfile.write(resp.content)

def dump_attached_files(self, document, path):
    """Dumps attached files from a nuxeo document

This function fetches attached files from a nuxeo document (stored as
a JSON).
Files definitions are stored in the configuration file.
"""
for filekind, filedef in self.cfg.filedefs.items():
    try:
        pointer = document
        for node in filedef['nuxepath']:
            pointer = pointer[node]
        pointer = pointer['name']
        nxfilename = pointer['name']
        url = (self.cfg.baseurl +
               self.cfg.suffixfile +
               self.cfg.nxrepo +
               document['uid'] + '/' +
               filedef['nuxepath'][-1])
    except Requester.compute_extension(nxfilename):
        filename = filedef['dumpfilename'] + '.' + ext
        self._dump_file(url, path, filename)
    except TypeError:
        pass

def get_directory(self, **kwargs):
    """Get the uid directory for the environment as a pandas DataFrame

This function fetches the uid directory data for the current
environment as a pandas DataFrame.
To fetch all the results, set max_page attribute to -1.
This function returns data as is from PIM, and requires it to be
formatted as a directory later on with '_init_as_directory' or
'_format_as_directory' methods. (else, columns are not consistent with
directory definition)
"""
    self.fetch_all_from_PIM(nx_properties='', **kwargs)
    df_list = []
    for single_result in self.result:
        df_list.append(pd.DataFrame(single_result.json()['entries'],
                                    columns=['lastModified']))
    df = pd.concat(df_list)
    df['lastModified'] = pd.to_datetime(df.loc[:, 'lastModified'])
    return(df)

@staticmethod
def _directory_headers():
    """Returns the directory headers
    """
    headers = ['type',
               'title',
               'lastModified',
               'lastRefreshed',
               'lastFetchedData',
               'lastFetchedFiles']
    return(headers)

def _load_directory(self, filename=None):
    """Loads directory from disk into the tool attribute
    """
    directory_filename = filename if filename else self.cfg.udidirectory
    full_path = os.path.join(self._root_path(), directory_filename)
    self._directory = (pd.read_csv(full_path,
                                   encoding='utf-8-sig',
                                   parse_dates=['lastModified',
                                                'lastRefreshed',
                                                'lastFetchedData',
                                                'lastFetchedFiles'])
                      .set_index('uid'))
    self._directory = Requester._format_as_directory(self._directory)

@staticmethod
def _init_as_directory(df):
    """Sets initial values for dataframes to be used as directories
    """
    df['lastRefreshed'] = pd.Timestamp.now(tz='UTC')
    df['lastFetchedData'] = np.nan
    df['lastFetchedFiles'] = np.nan
    return(Requester._format_as_directory(df))

@staticmethod
def _format_as_directory(df):
    """Formats a dataframe as a directory from dtypes point of view
    """
    This method formats a dataframe columns as to be consistent with
    directory definition. It DOES NOT set initial values (see
    '_init_as_directory').
    """
    types = {'lastFetchedData': 'datetime64[ns, UTC]',
             'lastFetchedFiles': 'datetime64[ns, UTC]'}
    df = df.astype(types)
    df = df.loc[:, Requester._directory_headers()]
    return(df)

def reset_directory(self, page_size=None, max_page=None, filename=None):
    """COMPLETELY RESETS the directory for the environment

Warning: this function will completely reset the directory for the
current environment. It should only be used when first setting the
directory or if it requires being remade from scratch.
Warning: this function should never be used with max_page parameter
different from -1 as it could result in incomplete data to erase
current directory. Only relevant usage of max_page != -1 is for
debugging purpose.
"""
    self._directory = Requester._init_as_directory(
        self.get_directory(max_page=max_page, page_size=page_size))
    self._save_directory()

def refresh_directory(self, max_page=None, filename=None, page_size=None):
    """Refreshes the directory for the environment

Warning: this function should never be used with max_page parameter
different from -1 except for debugging purpose. Yet, doing so does
not corrupt or lose data whatsoever.
"""
    new_dir = self.get_directory(max_page=max_page, page_size=page_size)
    new_dir = Requester._init_as_directory(new_dir)
    self._load_directory(filename=filename)
    cur_dir = Requester._format_as_directory(self._directory)
    cur_dir.update(new_dir)
    cur_dir = pd.concat([cur_dir,
                        new_dir[new_dir.index.isin(cur_dir.index)]])
    self._directory = cur_dir
    self._save_directory()

def _save_directory(self, filename=None):

```

```

    """Saves current directory to disk
This methods saves the current directory to disk.
"""
directory_filename = filename if filename else self.cfg.uiddirectory
full_path = os.path.join(self._root_path(), directory_filename)
self._directory.to_csv(full_path, encoding='utf-8-sig')

def modified_items(self, what='any', max_results=None):
    """Returns modified items according to directory

This methods compares the date of last modification stored in the
directory to the date of last fetching of data or attached files.
If what = 'any', all outdated items will be returned
If what = 'data', it will return only items with outdated data
If what = 'files', it will return only items with outdated files
max_results enables to fetch only a limited count of results.
This returns an uid list
"""
if not hasattr(self, '_directory'):
    self._load_directory()
df = self._directory
if what == 'any':
    mask = (df.lastFetchedFiles.isna() |
            (df.lastModified > df.lastFetchedFiles) |
            df.lastFetchedData.isna() |
            (df.lastModified > df.lastFetchedData))
elif what == 'data':
    mask = (df.lastFetchedData.isna() |
            (df.lastModified > df.lastFetchedData))
elif what == 'files':
    mask = (df.lastFetchedFiles.isna() |
            (df.lastModified > df.lastFetchedFiles))
else:
    raise ValueError(f"Unexpected 'what' argument: {what}")
uid_list = mask.index[mask].tolist()
if max_results:
    uid_list = uid_list[:max_results]
return(uid_list)

def modification_report(self):
    """Prints some elements about current state

This methods bases on current state of directory.
"""
if not hasattr(self, '_directory'):
    self._load_directory()
print(f'Number of items: {len(self._directory)}')
print(f'Number of items with outdated data: '
      f'{len(self.modified_items(what="data"))}')
print(f'Number of items with outdated files: '
      f'{len(self.modified_items(what="files"))}')

@staticmethod
def compute_extension(filename):
    """Computes the extension from a filename

Returns the extension. If the filename has no "." (dot) in it, returns
an empty string. If the computed extension has strictly more than 4
characters, returns the empty string."""
splitted = filename.split('.')
if len(splitted) == 1:
    return('')
elif len(splitted[-1]) > 4:
    return('')
else:
    return(filename.split('.')[1].lower())

def result_to_dataframe(self,
                       record_path='entries',
                       meta=None,
                       mapping=None,
                       index='uid'):
    """Formats result content as a dataframe with defined format

record_path and meta are pandas json_normalize method arguments
mapping is a key : adress mapping that maps return dataframe keys to
data adress in the JSON. Reminder: json_normalize default separator
is '.'.
So, for example if you want to have a record path down to the
ingredients table (that is, down to
record > properties > pprod:mainIngredients), with the uid product as
index, you should use the following parameters:
    record_path=['entries', 'properties', 'pprod:mainIngredients'],
    meta=[['entries', 'uid']]"""

```



```

    index='entries.uid'
    index is the field identifier(s) for the field(s) to be used as index
"""
result_json = [result.json() for result in self.result]
if mapping:
    with CleanJSONDataFrame(result_json,
                            record_path=record_path,
                            meta=meta) as df:
        for key, path in mapping.items():
            try:
                df[df[key]] = df[df.prefix + path]
            except KeyError:
                df[df[key]] = np.nan
df = df[df]
else:
    try:
        df = pd.json_normalize(result_json,
                               record_path=record_path,
                               meta=meta)
    except AttributeError:
        df = pd.io.json.json_normalize(result_json,
                                       record_path=record_path,
                                       meta=meta)
if index:
    df.set_index(index, inplace=True)
return(df)

def file_report_from_result(self,
                           mapping,
                           index='uid',
                           record_path='entries'):
    """Returns a dataframe about the files contained in the result

This methods analyzes the content of the result, and generates a
dataframe which enables the analysis of the files on the products
Files to report on are based on the content of the config.yaml
configuration file.
"""
if not record_path:
    record_path = 'entries'
for filekind, filedef in self.cfg.filedefs.items():
    mapping[filekind] = '.'.join(filedef['nuxepath']) + '.name'
df = self.result_to_dataframe(record_path=record_path,
                             mapping=mapping,
                             meta=None,
                             index=index)
for filekind in self.cfg.filedefs:
    df[f'has_{filekind}'] = df[filekind].notna()
    del(df[filekind])
return(df)

class CleanJSONDataFrame(object):
    """Context manager class for cleanly importing data from JSON

This class enables to create a context manager that enables to read JSON
data into a dataframe, by specifying the data to keep (this feature is not
provided yet by the pandas json_normalize). It does so by following these
steps:
- read data from a JSON into a new dataframe
- enable one to duplicate loaded data into new fields
- delete the loaded data when exiting, thus keeping only duplicated data
Complicated prefix is set to avoid duplicates in column names.
"""
def __init__(self, data, record_path=None, meta=None,
            prefix='_prev_duplic1'):
    try:
        self.df = pd.json_normalize(data, record_path=record_path,
                                    meta=meta, record_prefix=prefix,
                                    meta_prefix=prefix)
    except AttributeError:
        self.df = pd.io.json.json_normalize(data,
                                             record_path=record_path,
                                             meta=meta,
                                             record_prefix=prefix,
                                             meta_prefix=prefix,
                                             )
    self.prefix = prefix
    self.columns = list(self.df.columns.values)

def __enter__(self):
    return(self)

def __exit__(self, exc_type, exc_val, exc_tb):
    self.df.drop([c for c in self.columns], axis=1, inplace=True)

```

H.3 Conversion des pièces jointes en textes - Module pimpdf

```

    """PIM PDF Module

This module aims to parse the content of PDF files into text.
"""

import pandas as pd
from multiprocessing import cpu_count
from pathos.multiprocessing import ProcessPool as Pool
from io import StringIO, BytesIO
from functools import partial

from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfparser import PDFParser

class PDFDecoder(object):

```

```

"""Tool that provides basic pdf decoding functionnalities
"""

@staticmethod
def content_to_text(content,
                    none_content='raise'):
    """Decodes the binary passed as argument

    content arg must be a BytesIO object.
    none_content arg must be raise (if it is not expected to have an empty
    input, the default) or to_empty (which will cause to return an empty
    string)
    """
    if none_content not in {'raise', 'to_empty'}:
        raise ValueError(f'Unexpected value for none_content parameter. '
                         f'Got {none_content} but only \'raise\' or '
                         f'\'to_empty\' are expected.')
    if not content.read():
        if none_content == 'raise':
            raise RuntimeError('PDFMiner got an empty bytesIO object to '
                               'parse')
        if none_content == 'to_empty':
            return('')
    output_string = StringIO()
    parser = PDFParser(content)
    doc = PDFDocument(parser)
    rsrcmgr = PDFResourceManager()
    device = TextConverter(rsrcmgr, output_string,
                           laparams=LAParams())
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    for page in PDFPage.create_pages(doc):
        interpreter.process_page(page)
    return(output_string.getvalue())

@staticmethod
def path_to_text(path, missing_file='raise'):
    """Decodes file at local path in the form of a long string
    """

    if missing_file not in {'raise', 'ignore'}:
        raise ValueError(f'Unexpected value for missing_file parameter. '
                         f'Got {missing_file} but only \'raise\' or '
                         f'\'ignore\' are expected.')
    try:
        with open(path, mode='rb') as content:
            return(PDFDecoder.content_to_text(content))
    except FileNotFoundError:
        if missing_file == 'raise':
            raise
        if missing_file == 'ignore':
            print(f'File not found at path: {path}')
            return('')
    except Exception as e:
        print(e)
        print(f'An error happened at path: {path}')
    return('')

@staticmethod
def text_to_blocks(text, split_func=lambda x: x.split('\n\n')):
    """Splits text passed as an argument with splitter function

    split_func should be defined as to return a list like object.
    """
    return(split_func(text))

@staticmethod
def text_to_blocks_series(text, index=None,
                         split_func=lambda x: x.split('\n\n'),
                         return_type='along_index'):
    """Splits text passed as an argument (using splitter func) to a Series

    The return_type can be:
    - 'along_index': the return is a pandas Series of length the number
                      of blocks (with the index provided)
    - 'as_list' : the return is a pandas Series of length 1, with
                  the provided scalar as index, and the value of the
                  Series being the blocks as a list of strings
    If return_type is 'along_index' (the default), the index arg is passed
    as provided to pd.Series constructor, or if it is a scalar it is
    broadcasted as a constant on all values.
    """
    blocks_list = PDFDecoder.text_to_blocks(text,
                                             split_func=split_func,
                                             )
    if return_type not in {'along_index', 'as_list'}:
        raise ValueError(f'Unexpected value for return_type parameter. '
                         f'Got {return_type} but only \'along_index\' or '
                         f'\'as_list\' are expected.')
    if return_type == 'as_list':
        return(pd.Series([blocks_list], index=[index]))
    elif return_type == 'along_index':
        try:
            return(pd.Series(blocks_list, index=index))
        except TypeError:
            index = [index] * len(blocks_list)
            return(pd.Series(blocks_list, index=index))

@staticmethod
def path_to_blocks(path, split_func=lambda x: x.split('\n\n'),
                   missing_file='raise'):
    """Decodes file at local path in the form of a list of blocks

    Blocks are part of the original string separated by at least 2
    carriage returns (i.e. with at least a single blank line between them)
    """
    text = PDFDecoder.path_to_text(path, missing_file=missing_file)
    return(PDFDecoder.text_to_blocks(text, split_func=split_func))

    @staticmethod
    def path_to_blocks_series(path,
                             index=None,
                             split_func=lambda x: x.split('\n\n'),
                             missing_file='raise'):
        """Decodes file at local path in the form a pd Series of blocks

        text = PDFDecoder.path_to_text(path, missing_file=missing_file)
        return(PDFDecoder.text_to_blocks_series(text,
                                              split_func=split_func,
                                              index=index))

    @staticmethod
    def paths_to_blocks(path_series, split_func=lambda x: x.split('\n\n'),
                        missing_file='raise'):
        """Decodes files for each path in path list as a blocks Dataseries

        Blocks are part of the original string after the splitting function
        has been applied.
        The input must be a pandas dataseries of paths.
        The output is another pd Series, with the same indexes as the initial
        series (broadcasted to match the block count for each path)
        """
        ds_list = []
        for uid, path in path_series.items():
            ds = (PDFDecoder
                  .path_to_blocks_series(path,
                                         split_func=split_func,
                                         index=uid,
                                         missing_file=missing_file))
            ds_list.append(ds)
        return(pd.concat(ds_list, axis=0))

    @staticmethod
    def threaded_paths_to_blocks(path_series, processes=None,
                                 split_func=lambda x: x.split('\n\n'),
                                 missing_file='raise',
                                 ):
        """Threaded version of paths_to_blocks method

        It takes as input a series which index is the uid of the products,
        and the values are the path to the document.
        processes argument is the number of processes to launch. If omitted,
        it defaults to the number of cpu cores on the machine.
        """
        processor = partial(PDFDecoder.path_to_blocks_series,
                            split_func=split_func, missing_file=missing_file)
        processes = processes if processes else cpu_count()
        print(f'Launching {processes} processes.')
        # Pool with context manager do not seem to work due to issue 38501 of
        # standard python library. It hangs when running tests through pytest
        # see: https://bugs.python.org/issue38501
        # Below content should be tested again whenever this issue is closed
        #
        # with Pool(nodes=processes) as pool:
        #     ds_list = pool.map(processor,
        #                        path_series, path_series.index)
        #     # End of block
        #
        # This temporary solution should be removed when tests mentioned above
        # are successful.
        # This just closes each pool after execution or exception.
        try:
            pool = Pool(nodes=processes)
            pool.restart(force=True)
            ds_list = pool.map(processor,
                               path_series,
                               path_series.index)
        except Exception:
            pool.close()
            raise
        pool.close()
        # End of block
        return(pd.concat(ds_list, axis=0))

    @staticmethod
    def threaded_contents_to_text(content_series,
                                  processes=None,
                                  none_content='raise',
                                  ):
        """Threaded version of content_to_text method

        It takes as input a series which index is the uid of the products,
        and the values are the content (in the form of bytes) of the
        documents.
        processes argument is the number of processes to launch. If omitted,
        it defaults to the number of cpu cores on the machine.
        none_content arg can be 'raise' (default) or to_empty
        """
        processor = partial(PDFDecoder.content_to_text,
                            none_content=none_content,
                            )
        processes = processes if processes else cpu_count()
        print(f'Launching {processes} processes.')
        in_ds = content_series.apply(BytesIO)
        #
        # with Pool(nodes=processes) as pool:
        #     tuples = (list(in_ds.index),
        #               pool.map(processor, in_ds))

```

```

# End of block

# This temporary solution should be removed when tests mentioned above
# are successful.
# This just closes each pool after execution or exception.
try:
    pool = Pool(nodes=processes)
    pool.restart(force=True)
    tuples = (list(in_ds.index), pool.map(processer, in_ds))
except Exception:
    pool.close()
    raise
pool.close()
# End of block

ds = pd.Series(tuples[1], index=tuples[0])
return(ds)

@staticmethod
def threaded_texts_to_blocks(text_series, processes=None,
                             split_func=lambda x: x.split('\n\n'),
                             return_type='along_index'):
    """Threaded version of text_to_blocks_series method

It takes as input a series which index is the uid of the products,
and the values are the content (in the form of bytes) of the
documents..
processes argument is the number of processes to launch. If omitted,
it defaults to the number of cpu cores on the machine.
As for text_to_blocks_series function, return_type can be 'along_axis'
or 'list_like'.
    """

```

```

"""
processer = partial(PDFDecoder.text_to_blocks_series,
                    split_func=split_func,
                    return_type=return_type)
processes = processes if processes else cpu_count()
print(f'Launching {processes} processes.')

# Pool with context manager do not seem to work due to issue 38501 of
# standard python library. It hangs when running tests through pytest
# see: https://bugs.python.org/issue38501
# Below content should be tested again whenever this issue is closed
#
# with Pool(nodes=processes) as pool:
#     ds_list = pool.map(processer, text_series, text_series.index)
#
# End of block

# This temporary solution should be removed when tests mentioned above
# are successful.
# This just closes each pool after execution or exception.
try:
    pool = Pool(nodes=processes)
    pool.restart(force=True)
    ds_list = pool.map(processer, text_series, text_series.index)
except Exception:
    pool.close()
    raise
pool.close()
# End of block

ds = pd.concat(ds_list, axis=0)
return(ds)

```

H.4 Transformateurs et estimateurs spécifiques - Module pimest

Ce module définit divers estimateurs et transformateurs utilisés dans les modèles. Les fonctions de scoring, servant à la mesure de la performance, en font également partie. Les classes ‘IngredientExtractor’ et ‘PIMIngredientExtractor’ ont été construite au début des travaux sur le sujet, et sont donc plus « brouillonnées » que le reste du code.

```

"""PIM Estimator module

This modules enables to create an estimator to identify which block is the
ingredient list from an iterable of text blocks.
"""

import os
from io import BytesIO
import numpy as np
from scipy.sparse.linalg import norm as sparse_norm
from scipy.sparse import csr_matrix
import pandas as pd
from pathlib import Path
from functools import partial

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.utils.validation import check_is_fitted
from sklearn.preprocessing import normalize
from sklearn.base import clone
from gensim.models import Word2Vec
from jellyfish import damerau_levenshtein_distance
from Levenshtein import distance as levenshtein_distance
from Levenshtein import jaro as jaro_similarity
from Levenshtein import jaro_winkler as jaro_winkler_similarity

from .pimapi import Requester
from .pimpdf import PDFDecoder
from .conf import Config

class CustomTransformer(object):
    """Abstract class for custom transformers
    """
    def __init__(self, source_col, target_col, target_exists):
        self.source_col = source_col
        self.target_col = target_col
        self.target_exists = target_exists

    def raise_if_not_a_df(self, X):
        if not isinstance(X, pd.DataFrame):
            raise TypeError(f'This transformer expects a pandas Dataframe '
                           f'object. Got an object of type \'{type(X)}\' '
                           f'instead.')

    def check_target_exists(self):
        if self.target_exists not in {'raise', 'ignore', 'overwrite'}:
            raise ValueError(f'target_exists parameter should be set to '
                           f'\'raise\' or \'ignore\' or \'to_nan\'. Got '
                           f'\'{self.target_exists}\' instead.')

```

```

def raise_if_target(self, X):
    if self.target_col in X.columns and self.target_exists == 'raise':
        raise RuntimeError(f'Column \'{self.target_col}\' already exists '
                           f'in input DataFrame.')

```

```

def raise_if_no_source(self, X):
    if self.source_col and self.source_col not in X.columns:
        raise KeyError(f'Input DataFrame has no \'{self.source_col}\' '
                      f'column.')

```

```

def fit(self, X, y=None):
    self.check_target_exists()
    self.raise_if_not_a_df(X)
    self.raise_if_target(X)
    self.raise_if_no_source(X)

```

```

def transform(self, X):
    check_is_fitted(self)
    self.check_target_exists()
    self.raise_if_not_a_df(X)
    self.raise_if_target(X)
    self.raise_if_no_source(X)
    if self.target_col in X.columns and self.target_exists == 'ignore':
        return(X)

```

```

def fit_transform(self, X, y=None):
    return(self.fit(X).transform(X))

```

```

def get_params(self, deep=True):
    params = dict()
    params['source_col'] = self.source_col
    params['target_col'] = self.target_col
    params['target_exists'] = self.target_exists
    return(params)

```

```

def set_params(self, **parameters):
    for parameter, value in parameters.items():
        setattr(self, parameter, value)
    return(self)

```

```

class IngredientExtractor(object):
    """Estimator that identifies the most 'ingredient like' block from a list
    """
    def __init__(self):
        """Constructor method of ingredient extractor"""
        pass

    def fit(self, X, y=None):
        """Fitter method of ingredient extractor

```

```

X is an iterable of ingredient lists in the form of strings
y is just here for compatibility in sklearn pipeline usage
"""
self._count_vect = CountVectorizer()
self.vectorized_texts_ = self._count_vect.fit_transform(X)
self.vocabulary_ = self._count_vect.vocabulary_
self.mean_corpus_ = self.vectorized_texts_.mean(axis=0)
return(self)

def predict(self, X):
    """Predictor method of ingredient extractor

X is a list of text blocks.
This methods returns the index of the text block that is most likely
to hold the ingredient list"""
XAgainstIngrd_voc = self._count_vect.transform(X)
X_norms = sparse_norm(CountVectorizer().fit_transform(X), axis=1)
X_dot_ingred = np.array(XAgainstIngrd_voc.sum(axis=1)).squeeze()
pseudo_cosine_sim = np.divide(X_dot_ingred,
                               X_norms,
                               out=np.zeros(X_norms.shape),
                               where=X_norms != 0)
self.similarity_ = pseudo_cosine_sim
return(np.argmax(pseudo_cosine_sim))

def show_emphasize(self, X):
    """Method that prints strings with words from vocabulary emphasized
    """
    for text in self.emphasize_texts(X):
        print(text)

def emphasize_texts(self, X):
    """Method that returns strings with words from vocabulary emphasized

This method shows how some candidates texts are projected on the
vocabulary that has been provided or gotten from fitting.
It is useful to see how different blocks compare.
X argument is an iterable of block candidates.
"""
check_is_fitted(self)
preprocessor = self._count_vect.build_preprocessor()
tokenizer = self._count_vect.build_tokenizer()
vocabulary = self._count_vect.vocabulary_
emphasized_texts = []
for block in X:
    text = self.emphasize_words(block,
                                 preprocessor=preprocessor,
                                 tokenizer=tokenizer,
                                 vocabulary=vocabulary,
                                 )
    emphasized_texts.append(text)
return(emphasized_texts)

def emphasize_words(self,
                    text,
                    preprocessor=None,
                    tokenizer=None,
                    vocabulary=None,
                    ansi_color='\033[92m', # green by default
                    ):
    """Method that returns a string with words emhasized

This methods takes a string and returns a similar string with the words
emphasized (with color markers)
"""
check_is_fitted(self)
ansi_end_block = '\033[0m'
if not preprocessor:
    preprocessor = self._count_vect.build_preprocessor()
if not tokenizer:
    tokenizer = self._count_vect.build_tokenizer()
if not vocabulary:
    vocabulary = self._count_vect.vocabulary_
preprocessed_text = preprocessor(text)
tokenized_text = tokenizer(preprocessed_text)
idx = 0
emphasized_text = ''
for token in tokenized_text:
    if token in vocabulary:
        while preprocessed_text[idx: idx + len(token)] != token:
            emphasized_text += text[idx]
            idx += 1
        emphasized_text += (ansi_color + text[idx: idx + len(token)] +
                            ansi_end_block)
        idx += len(token)
    emphasized_text += text[idx:]
return(emphasized_text)

def score(self, X, y):
    """Scorer method of ingredient extractor estimator

X is an iterable of ingredient lists in the form of string
y is the target as the index of the correct block.
"""
pass

class PIMIngredientExtractor(IngredientExtractor):
    """Wrapped estimator that directly extracts the ingredient list from uid
    """
    def __init__(self, env='prd', **kwargs):
        self.requester = Requester(env, **kwargs)
        super().__init__()

    def compare_uid_data(self, uid):
        check_is_fitted(self)
        print(f'Fetching data from PIM for uid {uid}...')

        self.requester.fetch_list_from_PIM([uid])
        try:
            ingredient_list = (self.requester.result[0]
                               .json()['entries'][0]['properties']
                               ['pprod:ingredientsList'])
        except IndexError:
            raise IndexError(f'Fetching data with uid {uid} ')
        print('-----')
        print(f'Ingredient list from PIM is :\n{n}(ingredient_list)')
        print('-----')
        print(f'Supplier technical datasheet from PIM for uid {uid} is :')
        nuxeo_path = (self.requester.cfg.filedefs['supplierdatasheet']
                      ['nuxepath'])
        pointer = self.requester.result[0].json()['entries'][0]
        for node in nuxeo_path:
            pointer = pointer[node]
        file_url = pointer['data']
        print(file_url)
        print('-----')
        print(f'Downloading content of technical datasheet file...')
        self.resp = self.requester.session.get(file_url,
                                                stream=True)
        resp = self.resp
        print('Done!')
        print('-----')
        print(f'Parsing content of technical datasheet file...')
        blocks = (PDFDecoder.content_to_text(BytesIO(resp.content))
                  .split('\n\n'))
        idx = self.predict(blocks)
        print('Done!')
        print('-----')
        print(f'Ingredient list extracted from technical datasheet:\n{n}')
        print(blocks[idx])
        print('-----')

    def print_blocks(self):
        blocks = (PDFDecoder.content_to_text(BytesIO(self.resp.content))
                  .split('\n\n'))
        for i, block in enumerate(blocks):
            print(i, '|', block, '\n')

    class PathGetter(CustomTransformer):
        """Class that gets path for documents on disk

This class aims to compute the path to documents, in order to
fetch documents from the correct folder (depending on whether
they are from train set or from ground truth)
All these can be set at initialization, if such is not the case
then their values is gotten from the configuration file.
"""
        def __init__(self,
                     env='prd',
                     ground_truth_uids=None,
                     train_set_path=None,
                     ground_truth_path=None,
                     path_factory=lambda x: x,
                     filename_factory=lambda x: 'FTF.pdf',
                     source_col=None,
                     target_col='path',
                     target_exists='raise'):
            self.env = env
            self.ground_truth_uids = ground_truth_uids
            self.train_set_path = train_set_path
            self.ground_truth_path = ground_truth_path
            self.path_factory = path_factory
            self.filename_factory = filename_factory
            super().__init__(source_col=source_col, target_col=target_col,
                            target_exists=target_exists)

        def fit(self, X, y=None):
            """No fit is required for this class.
            """
            super().fit(X)
            self.cfg = Config(self.env)
            if not self.train_set_path:
                self.train_set_path = os.path.join(*self.cfg.trainsetpath)
            if not self.ground_truth_path:
                self.ground_truth_path = os.path.join(*self.cfg.groundtruthpath)
            self.fitted_ = True
            return(self)

        def transform(self, X):
            """Returns the paths for the uids"""
            super().transform(X)
            df = X.copy()
            df[self.target_col] = None
            for uid in X.index:
                if uid in self.ground_truth_uids:
                    path = os.path.join(self.ground_truth_path,
                                        self.path_factory(uid),
                                        self.filename_factory(uid),
                                        )
                else:
                    path = os.path.join(self.train_set_path,
                                        self.path_factory(uid),
                                        self.filename_factory(uid),
                                        )
            df.loc[uid, self.target_col] = path
            return(df)

        def get_params(self, deep=True):
            parms = super().get_params()
            parms['env'] = self.env
            parms['ground_truth_uids'] = self.ground_truth_uids
            parms['train_set_path'] = self.train_set_path
            parms['ground_truth_path'] = self.ground_truth_path

```

```

parms['path_factory'] = self.path_factory
parms['filename_factory'] = self.filename_factory
return(parms)

class ContentGetter(CustomTransformer):
    """Class that fetches the content of documents on disk

This class fetches the data from documents on disk as bytes.
It requires a dataframe with a path column"""
def __init__(self,
             missing_file='raise',
             target_exists='raise',
             source_col='path',
             target_col='content',
             ):
    self.missing_file = missing_file
super().__init__(source_col=source_col,
                 target_col=target_col,
                 target_exists=target_exists,
                 )

def fit(self, X, y=None):
    super().fit(X)
    if self.missing_file not in {'raise', 'ignore', 'to_nan'}:
        raise ValueError(f'missing_file parameter should be set to '
                         f'\'raise\' or \'ignore\' or \'to_nan\'. Got '
                         f'\'{self.missing_file}\'' instead.')
    self._raise_if_no_file(X)
    self.fitted_ = True
    return(self)

def _raise_if_no_file(self, X):
    if self.missing_file == 'raise':
        mask = pd.DataFrame(index=X.index)
        mask['file_exists'] = X['path'].apply(ContentGetter.file_exists)
        if not mask['file_exists'].all():
            example_uid = mask.loc[~mask['file_exists']].index[0]
            example_path = X.loc[example_uid, 'path']
            raise RuntimeError(f'No file found for uid \'{example_uid}\'
                               at path \'{example_path}\'')

def transform(self, X):
    super().transform(X)
    self._raise_if_no_file(X)
    X = X.copy()
    mask = pd.DataFrame(index=X.index)
    mask['file_exists'] = X['path'].apply(ContentGetter.file_exists)
    mask['target'] = X['path'].apply(ContentGetter.read_to_bytes)
    if self.missing_file == 'to_nan':
        idx_to_update = mask.index
    else:
        idx_to_update = mask['file_exists']
    X.loc[idx_to_update, 'content'] = mask.loc[idx_to_update, 'target']
    return(X)

@staticmethod
def read_to_bytes(path):
    try:
        return(Path(path).read_bytes())
    except FileNotFoundError:
        return(None)

@staticmethod
def file_exists(path):
    path = Path(path)
    return(path.is_file())

def get_params(self, deep=True):
    parms = super().get_params()
    parms['missing_file'] = self.missing_file
    return(parms)

class PDFContentParser(CustomTransformer):
    """Class that parses pdf content to text

This class converts a file content (in the form of bytes) into text, using
pypdf functionalities (based on pdfminer.six)
"""

def __init__(self,
             target_exists='raise',
             source_col='content',
             target_col='text',
             none_content='raise',
             ):
    self.none_content = none_content
super().__init__(source_col=source_col,
                 target_col=target_col,
                 target_exists=target_exists,
                 )

def fit(self, X, y=None):
    super().fit(X)
    self.fitted_ = True
    return(self)

def transform(self, X):
    super().transform(X)
    X = X.copy()
    tran = (PDFDecoder
            .threaded_contents_to_text(X[self.source_col],
                                       none_content=self.none_content))
    X[self.target_col] = tran
    return(X)

def get_params(self, deep=True):
    parms = super().get_params()

parms['none_content'] = self.none_content
return(parms)

class BlockSplitter(CustomTransformer):
    """Class that splits texts into blocks

This class converts a text string into blocks (a list of string), using
the splitter function provided
"""
def __init__(self,
             target_exists='raise',
             source_col='text',
             target_col='blocks',
             splitter_func=(lambda x: x.split('\n\n')):
             ):
    self.splitter_func = splitter_func
super().__init__(target_exists=target_exists,
                 source_col=source_col,
                 target_col=target_col,
                 )

def fit(self, X, y=None):
    super().fit(X)
    self._check_splitter_callable()
    self.fitted_ = True
    return(self)

def _check_splitter_callable(self):
    self.splitter_func('')

def transform(self, X):
    super().transform(X)
    X = X.copy()
    blocks = (PDFDecoder
              .threaded_texts_to_blocks(X[self.source_col],
                                         split_func=self.splitter_func,
                                         return_type='as_list',
                                         ))
    X[self.target_col] = blocks
    return(X)

def get_params(self, deep=True):
    parms = super().get_params()
    parms['splitter_func'] = self.splitter_func
    return(parms)

class SimilaritySelector():
    """Class that select the most similar block from a block list

This class provides functionalities to fit an estimator on a topic
specific vocabulary, and to retrieve the best candidate amongst these
blocks.
"""
def __init__(self,
             count_vect_type='TfidfVectorizer',
             count_vect_kwargs=None,
             similarity='projection',
             source_norm='l2',
             projected_norm='l1',
             scoring='default',
             embedding_method=None,
             embedding_parms=None,
             ):
    self.count_vect_type = count_vect_type
    self.count_vect_kwargs = count_vect_kwargs
    self.similarity = similarity
    self.source_norm = source_norm
    self.projected_norm = projected_norm
    self.scoring = scoring
    self.embedding_method = embedding_method
    self.embedding_parms = embedding_parms

def get_params(self, deep=True):
    parms = dict()
    parms['count_vect_type'] = self.count_vect_type
    parms['count_vect_kwargs'] = self.count_vect_kwargs
    parms['similarity'] = self.similarity
    parms['source_norm'] = self.source_norm
    parms['projected_norm'] = self.projected_norm
    parms['scoring'] = self.scoring
    parms['embedding_method'] = self.embedding_method
    parms['embedding_parms'] = self.embedding_parms
    return(parms)

def set_params(self, **parameters):
    for parameter, value in parameters.items():
        setattr(self, parameter, value)
    return(self)

def fit(self, X, y):
    self._validate_similarity()
    self._validate_vectorizer_type()
    self._validate_norms()
    self._validate_embedding_method()
    # if norm not specified in count_vect_kwargs, set to None
    # no normalization except if specifically asked for.
    if not self.count_vect_kwargs:
        self.count_vect_kwargs = dict()
    if not self.embedding_parms:
        self.embedding_parms = dict()
    if 'norm' not in self.count_vect_kwargs:
        self.count_vect_kwargs['norm'] = None
    if (self.count_vect_type == 'TfidfVectorizer'
        and 'use_idf' not in self.count_vect_kwargs):
        self.count_vect_kwargs['use_idf'] = False
    if self.count_vect_type == 'HashingVectorizer':

```

```

        self.count_vect_kwargs['alternate_sign'] = False
    if 'strip_accents' not in self.count_vect_kwargs:
        self.count_vect_kwargs['strip_accents'] = 'unicode'
    try:
        count_vect = self._vectorizer_class(**self.count_vect_kwargs)
    except (TypeError):
        raise ValueError('Unexpected argument at init in '
                         "'count_vect_kwargs.'")
    raise
    self.source_count_vect = count_vect
    docs = [text for block_list in X for text in block_list]
    try:
        self.source_count_vect.fit(docs)
    except Exception:
        print('Exception raised at fitting source vectorizer. See full '
              'stack for details.')
    raise

    if self.embedding_method:
        self.compute_embeddings(X, y)

    if self.similarity == 'projection':
        # do NOT modify instance attribute
        kwargs = self.count_vect_kwargs.copy()
        # remove alternate_sign if present, as it is only allowed for
        # HashingVectorizer and count_vect MUST be a TfidfVectorizer
        if 'alternate_sign' in kwargs:
            del(kwargs['alternate_sign'])
        # default use_idf to False. Why ?
        # kwargs['use_idf']=False
        if 'use_idf' in kwargs:
            raise ValueError('Unexpected argument at init in '
                             "'count_vect_kwargs.'")
        raise
    try:
        self.source_count_vect.fit(y.fillna(''))
    except (ValueError):
        raise ValueError('Unexpected argument at fit in '
                         "'count_vect_kwargs.'")
    raise

    if self.similarity == 'cosine':
        if self.scoring == 'default':
            # compute target vector in docs corpus space
            # if binary : document frequency of words in ingred corpus
            # if not : term counts in ingred corpus
            target_vector = self.source_count_vect.transform(y)
            target_vector = np.asarray(target_vector.mean(axis=0))
        elif self.scoring == 'absolute_score':
            # smooth doc frequency (log2(1+df))
            target_vector = np.asarray(self.compute_score(X, y))
        elif self.scoring == 'relative_score':
            # smooth relative doc frequency
            target_vector = np.asarray(self.compute_score(X, y,
                                              kind='relative'))
        # if embeddings have been computed, target gets embedded to
        # if hasattr(self, 'embeddings'):
        target_vector = np.dot(target_vector, self.embeddings)
        # normalize target vector to compute cosine sim via dot product
        normalize(target_vector, norm='l2', axis=1, copy=False)
        self.target_vector = target_vector.ravel()
        self.fitted_ = True
    return(self)

def _validate_similarity(self):
    if self.similarity not in {'projection', 'cosine'}:
        raise ValueError(f"similarity parameter should be set to "
                         f"'projection' or 'cosine'. Got "
                         f"'{self.similarity}' instead.")

def _validate_vectorizer_type(self):
    vect_types = {'TfidfVectorizer': TfidfVectorizer,
                  'HashingVectorizer': HashingVectorizer}
    if self.count_vect_type not in vect_types.keys():
        raise ValueError(f'count_vect_type parameter should be set to '
                         f"'TfidfVectorizer' or 'HashingVectorizer'. "
                         f"Got '{self.count_vect_type}' instead.")
    self._vectorizer_class = vect_types[self.count_vect_type]

def _validate_norms(self):
    test_mat = csr_matrix([[0, 1], [2, 3]])
    for norm in ('source_norm', 'projected_norm'):
        norm_val = getattr(self, norm)
        try:
            if norm_val[0] == '1':
                l_order = int(norm_val[1:])
                norm_func = partial(sparse_norm, axis=1, ord=l_order)
                setattr(self, norm, norm_func)
            except Exception:
                pass
            norm_val = getattr(self, norm)
        try:
            norm_val(test_mat)
        except (TypeError, ValueError) as e:
            print(f'Incorrect {norm} provided, see full stack for '
                  f'details')
            raise ValueError(e)

def _validate_embedding_method(self):
    if self.embedding_method not in {'Word2Vec', 'tSVD', None}:
        raise ValueError(f'embedding_method parameter should be set to '
                         f"'Word2Vec' or 'tSVD'. Got "
                         f"'{self.embedding_method}' instead.'")

def predict(self, X):
    """
    Function to predict best candidate
    X : a pandas Series of block lists, or a list of block lists.
    """
    check_is_fitted(self)
    self.computed_sims_ = []
    predicted_texts = []
    for block_list in X:
        if self.similarity == 'projection':
            texts = self.source_count_vect.transform(block_list)
            # project texts on corpus space
            projected_texts = self.count_vect.transform(block_list)
            # Compute norm of source texts
            texts_norms = self.source_norm(texts)
            # Compute norm of projected texts
            projected_norms = self.projected_norm(projected_texts)
            sim = np.divide(projected_norms,
                           texts_norms,
                           texts_norms,
                           out=np.zeros(texts_norms.shape),
                           where=texts_norms != 0)
        if self.similarity == 'cosine':
            candidates = self.source_count_vect.transform(block_list)
            # if embeddings are set, transform the candidates with these
            if hasattr(self, 'embeddings'):
                candidates = np.dot(candidates.toarray(), self.embeddings)
            # normalize candidates
            normalize(candidates, norm='l2', axis=1, copy=False)
            # convert to array if still a csr_matrix (case no embedding)
            try:
                candidates = candidates.toarray()
            except AttributeError:
                pass
            # compute cosine sim via dot product (normalized vectors)
            sim = np.dot(candidates, self.target_vector)
            self.computed_sims_.append(sim)
        predicted_texts.append(block_list[np.argmax(sim)])
    if isinstance(X, pd.Series):
        return(pd.Series(predicted_texts, index=X.index))
    else: # for example, X is a list
        return(pd.Series(predicted_texts))

def fit_predict(self, X, y):
    self.fit(X, y)
    return(self.predict(X))

def compute_score(self, X, y, kind='absolute', diff=True):
    """
    Method that computes the scores of words based on 2 corpora
    It takes into input X (list of block list or list of string) and y
    (same formats), and returns a vector with the score of each word in
    this estimator vocabulary.
    The vectorizer must have been fitted before.
    This method computes :
    - absolute scores for words in y : log2(1 + df) in y
    - relative scores between X and y (the higher, the more y-ish the word)
    Its value is log2(1 + df_y) - log2(1 + df_X)
    If diff is set to True, words counts from y will be deducted from X
    before computing document frequencies (e.g. when y text is supposed to
    be included in X).
    """
    if kind not in {'absolute', 'relative'}:
        raise ValueError(f"Unexpected value for 'kind' argument. "
                         f"'absolute' or 'relative' expected, got {kind} "
                         f"instead.")
    X = self.list_flatten(X)
    y = self.list_flatten(y)

    # set up word_counter with binary = True
    word_counter = clone(self.source_count_vect)
    word_counter.set_params(binary=True,
                           norm=None,
                           )
    try:
        if word_counter is a TfidfVectorizer, then use vocabulary from it
        # and no idf
        voc = self.source_count_vect.vocabulary_
        word_counter.set_params(use_idf=False,
                               vocabulary=voc)
    except Exception:
        # do nothing if it is a HashingVectorizer
        pass
    word_counter.fit(self.list_flatten(X))

    if kind == 'absolute':
        doc_freq = word_counter.transform(y).sum(axis=0) / len(y)
        return(np.log2(doc_freq + 1))

    if kind == 'relative':
        if diff:
            X = self.compute_diff(X, y)
        else:
            X = word_counter.transform(X)
        doc_freq = word_counter.transform(y).sum(axis=0) / len(y)
        doc_freq2 = X.sum(axis=0) / X.shape[0]
        return(np.log2(doc_freq + 1) - np.log2(doc_freq2 + 1))

def compute_diff(self, text, text_to_subtract, binary=True):
    """
    Method that subtracts the word of second texts to the first ones
    This method takes non vectorized texts, as it will work with term
    counts in every case (and vectorization can be made binary with this
    estimator).
    It returns a csr_matrix with resulting term counts.
    text : an iterable of strings or block lists (list of strings)

```

```

text_to_subtract : an iterable of strings or block lists (list of
strings)
"""
# this method requires the estimator has already been fitted
if not self._count_vect_type == 'HashingVectorizer':
    check_is_fitted(self._source_count_vect)
word_counter = clone(self._source_count_vect)
try:
    voc = self._source_count_vect.vocabulary_
    word_counter.set_params(binary=False,
                           vocabulary=voc)
except AttributeError:
    word_counter.set_params(binary=False)

# step 1: flatten inputs
text = self._list_flatten(text)
text_to_subtract = self._list_flatten(text_to_subtract)

# step 2: produce csr_matrix counts for each text list
text_ = word_counter.fit_transform(text)
text_to_subtract_ = word_counter.transform(text_to_subtract)

# step 3: compute difference
if binary:
    return(text_ > text_to_subtract_)
else:
    raise NotImplementedError('Non binary difference not yet '
                             'implemented.')

def list_flatten(self, block_list_iterable):
    """
    This methods takes an iterable of block list, and convert already
    tokenized blocks back to string
    """
    texts = [' '.join(block_list) if not isinstance(block_list, str)
            else block_list
            for block_list in block_list_iterable
            ]
    return(texts)

def compute_embeddings(self, X, y):
    """
    This method computes embeddings from corpus
    It instantiates the self.embedding attribute with an array having
    as many rows as there are words in the vocabulary.
    These embeddings are then used during prediction.
    Embeddings are computed on X full texts (i.e. complete docs)
    """
    X = X.copy(deep=True)
    if self._count_vect_type == 'HashingVectorizer':
        raise NotImplementedError('Cannot compute embeddings with '
                               'HashingVectorizer')
    if self.embedding_method == 'Word2Vec':
        # step 1: construct a tokenized corpus
        X = self._sentencize_corpus(X)
        # step 2: train a Word2Vec instance
        if 'min_count' not in self.embedding_params.keys():
            self.embedding_params['min_count'] = 1
        if 'size' not in self.embedding_params.keys():
            self.embedding_params['size'] = 100
        model = Word2Vec(X, **self.embedding_params)
        # step 3: construct the embeddings nparray
        feat_count = model.wv[list(model.wv.vocab.keys())[0]].shape[0]
        words_count = len(self._source_count_vect.vocabulary_)
        embedding_ = np.zeros((words_count, feat_count))
        for word in model.wv.vocab.keys():
            idx = self._source_count_vect.vocabulary_[word]
            embeddings[idx] = model.wv[word]
    if self.embedding_method == 'tSVD':
        # step 1: reconstruct a full corpus
        X = X.apply(lambda x: '\n\n'.join(x))
        # step 2: vectorize this new text
        X = self._source_count_vect.transform(X)
        # step 3: compute embeddings
        if 'n_components' not in self.embedding_params.keys():
            self.embedding_params['n_components'] = 100
        model = TruncatedSVD(**self.embedding_params).fit(X)
        embeddings = model.components_.T
        self.embeddings = embeddings

    def _tokenize(self):
        prepro = self._source_count_vect.build_preprocessor()
        token = self._source_count_vect.build_tokenizer()
        stop_words = self._source_count_vect.stop_words

        def tokenize(text):
            words = token(prepro(text))
            if stop_words:
                return([word for word in words if word not in stop_words])
            else:
                return(words)

        return(tokenize)

    def sentencize_corpus(self, X):
        """
        This method takes a corpus splitted in blocks back to a continuous
        doc and then tokenizes it.
        Each doc becomes a sentence.
        ex : [
            ['salade ninja', 'très bon'] # doc 1: 2 blocs
            ['fourmi joséphine', 'pingouin pédaïo', 'sel'] # doc 2: 3 blocs
        ]
        devient :
        [
            ['salade', 'ninja', 'tres', 'bon']
            ['fourmi', 'josephine', 'pingouin', 'pedalo', 'sel']
        ]
        Il s'agit du format attendu par Word2Vec de gensim.
        """
        tokenizer = self._tokenizer()
        return(X.apply(lambda x: tokenizer('\n\n'.join(x)))))

class DummyEstimator(object):
    """
    Dummy estimator that predicts y as exactly X
    This estimator has been developped for testing purposes, as pytest
    does not support yet class fixtures.
    """
    def fit(self, X, y=None):
        self.fitted_ = True

    def predict(self, X):
        return(X.copy())

    def build_text_processor(tokenize=True,
                           lowercase=True,
                           strip_accents='unicode',
                           **kwargs,
                           ):
        """
        Generates a text preprocessor from sklearn CountVectorizer tools
        It is based on sklearn CountVectorizer functionalities.
        tokenize means that the input string will be tokenized as words before
        being glued back with single spaces. Its purpose is to handle
        whitespaces (newlines, tabs, multiple spaces, ...) and punctuation.
        kwargs are directly passed to CountVectorizer constructor, and will
        serve to process the texts. Most useful args are 'stripAccent' and
        'lowercase'.
        """
        preprocessor_countvect = CountVectorizer(lowercase=lowercase,
                                                strip_accents=strip_accents,
                                                **kwargs,
                                                )
        preprocessor = preprocessor_countvect.build_preprocessor()
        tokenizer = preprocessor_countvect.build_tokenizer()
        if tokenize:
            def transformer(x):
                return(' '.join(tokenizer(preprocessor(x))))
        else:
            transformer = preprocessor
        return(transformer)

    def custom_accuracy(estimator, X, y, **kwargs):
        """
        Scorer that computes accuracy of estimator, for strings
        This function enables to score an estimator that returns long texts,
        with some text processing.
        It computes an accuracy after text processing.
        See build_text_processor for information on arguments.
        """
        transformer = build_text_processor(**kwargs)
        y_pred = pd.Series(estimator.predict(X).apply(transformer))
        return((y_pred == y).apply(transformer)).mean()

    def text_similarity(a, b, *, similarity, jw_prefix_weight=0.1):
        """
        Function that computes similarity of texts with selected method
        Similarity can be : 'levenshtein', 'damerau-levenshtein', 'jaro',
        'jaro-winkler'.
        In the case of jaro-winkler, a prefix weight can be passed
        """
        if not similarity:
            raise ValueError('similarity is a mandatory')
        if similarity == 'levenshtein':
            dist = levenshtein_distance(a, b)
        elif similarity == 'damerau-levenshtein':
            dist = damerau_levenshtein_distance(a, b)
        elif similarity == 'jaro':
            return(jaro_similarity(a, b))
        elif similarity == 'jaro-winkler':
            return(jaro_winkler_similarity(a, b, jw_prefix_weight))
        else:
            raise NotImplementedError(f'similarity set to {similarity}. This '
                                    f'method has not been implemented yet')

        try:
            return(1 - (dist / max(len(a), len(b))))
        except ZeroDivisionError:
            return(1.)

    def text_sim_score(self, estimator,
                      X,
                      y,
                      *,
                      similarity,
                      tokenize=True,
                      jw_prefix_weight=0.1,
                      **kwargs,
                      ):
        """
        Scorer that computes mean similarity for an estimator
        This function enables to score an estimator that returns long texts,
        with some text processing, by computing a mean similarity between
        predicted and target texts.
        It computes this similarity after text processing.
        See build_text_processor for information on arguments.
        """
        transformer = build_text_processor(tokenize, **kwargs)
        y_pred = pd.Series(estimator.predict(X))
        y_trans, y_pred_trans = y.apply(transformer), y_pred.apply(transformer)
        df = pd.concat([y_trans, y_pred_trans], axis=1)
        similarity = partial(text_similarity,

```

```
similarity=similarity,  
jw_prefix_weight=jw_prefix_weight,  
)  
  
similarities = df.apply(lambda x: similarity(x.iloc[0], x.iloc[1]), axis=1)  
return(similarities.mean())
```