Formation DS-IML de marc.lefevre@noos.fr

Rapport PROJET-5

Crash course NLP – Tagger des questions Stack Overflow

Mission :

L'objet de cet exposé est d'emmener le lecteur à la découverte du NLP, pour Natural Language Processing, ou tout du moins de certains de ses grands concepts, en suivant du début à la fin l'accomplissement d'un projet.

Ce projet consistait à mettre au point un modèle grâce auquel nous pourrions « tagger » automatiquement des questions que nous souhaiterions hypothétiquement poser sur le célèbre site d'entraide informatique s**tackoverflow.com**.

De l'acquisition des données à l'obtention d'un modèle prêt à être déployé, le lecteur pourra suivre le cheminement qui a été le nôtre et avoir, à travers ce cas concret, un aperçu de certaines des problématiques et pratiques rencontrées en NLP.

Ce compte-rendu est complémentaire des trois notebooks du projet qui contiennent de manière exhaustive tous les résultats et tout le « code » du projet.

Sommaire :

1. Introduction

1.1. Le NLP

L'acronyme NLP renvoie donc au Natural Language Processing, qui désigne la branche du machine learning traitant du langage (en l'occurrence ici il s'agira de texte en anglais). Il est notamment au coeur de nombreuses applications mettant en œuvre des « chatbots » textuels ou vocaux qu'on utilise aujourd'hui au quotidien. En ce sens, tout le monde a déjà rencontré le NLP[1]…

1.2. Les différentes phases d'un projet NLP

De manière schématique, on peut découper un projet de NLP en cinq phases principales :

- **Collecte et première mise en forme des données**. Il s'agit de la phase de construction du corpus de données.

- **Traitements des données sous leur forme de texte**. Une fois le corpus constitué, par choix ou par nécessité, différents traitements des textes sont effectués.

- **Vectorisation et traitement des données sous leur forme numérique**. Etant donné que les outils de machine learning ne travaillent qu'avec des données numériques, une 3e phase commençant par une « numérisation », ou plutôt par une « vectorisation » des données textes, est nécessaire. Vectorisation après laquelle viennent souvent d'autres traitements allant dans un sens de simplification des données.

- **Modélisation.** Une fois les données sous une forme adéquate, on procède à l'entrainement et à la création des modèles de classification recherchés.

- **Déploiement**. Pour terminer, vient la phase de déploiement du modèle afin qu'il soit accessible et utilisé comme il se doit.

1.3. Terrain et but du jeu

Avec un peu de hauteur, nous pouvons résumer la nature de notre travail de la façon suivante.

L'essentiel de l'action, c'est-à-dire l'essentiel du travail de recherche et d'amélioration afin d'obtenir de bons résultats, se passe dans les phases 2, 3 et 4.

Il s'agit de tester quelques-unes des options de traitement possibles en phases 2 et 3 afin d'en déduire une séquence de traitements adéquate, séquence permettant à un algorithme « bien choisi » de nous procurer les meilleurs résultats en phase 4.

2. Obtention et préparation des données

2.1. Stack-Exchange

La récupération des données s'est faite depuis l'API publique Stack Exchange qui permet l'accès à la base de données (et donc au contenu) du site Stack Overflow.

Au moyen de requêtes SQL adéquates, prenant en compte la pertinence et la qualité des questions à travers le critère de leur nombre de vues (seuil de 40 000 dégressif), et prenant aussi en compte leur récence (rien d'antérieur à 2015), nous nous sommes constitué un corpus de 25 555 questions (titre, corps de la question et tags correspondants)[2].

2.2. Scrapping

Le corps des questions ayant été obtenu sous forme de texte au format HTML (parsemé de balises de mise en page HTML…) il a été nécessaire de recourir à des outils de web scrapping afin de le transformer en texte « propre ». Nous avons utilisé dans ce but le module Beautiful Soup qui nous a aussi permis ce faisant de nous séparer des « gros blocs » de code présents dans les questions, entre les balises de pré-formatage <pre>, que nous ne jugions pas souhaitable de garder.

Ce texte « nettoyé » était ensuite réuni aux titres des questions et le tout passé en caractères minuscules[3].

2.3. Création des labels.

Les étiquettes dont nous avions besoin pour les parties supervisées de notre projet ont été construites à partir des très nombreux tags liés à chaque question.

Nous les avons d'abord nettoyés des chevrons les entourant dans les données originales puis simplifiés au moyen d'expressions régulières. Ainsi par exemple, des tags « python2 » ou « python3 » devenaient simplement « python ».

Ensuite, après avoir géré des cas particuliers des doublons (« react » et « react.js ») ou de termes non pertinents (« studio » lié originellement soit à « androïd » soit à « visual »), nous avons extrait les 24 tags les plus courants. Comme au moins un de ceux-ci était présent dans 80% des tags des questions, nous avons décidé qu'avec le renfort d'un nouveau tag « misc » qui concernerait

les 20% de questions jusqu'alors « sans tag », ils constitueraient le nouveau jeu d'étiquettes « cibles » que nous utiliserions.

Ce nouvel étiquetage a été effectué avec notamment l'outil « phrase matcher » de la bibliothèque SpaCy[4].

2.4. Echantillonnages

A ce stade, et pour terminer la phase 1, nous avons effectué les habituels échantillonnages « train / test », séparant les données d'entrainement et d'évaluation des futurs modèles, et « features / labels » séparant les nos questions des tags à prédire.

2.5. Traitements des textes

Voici maintenant les différents traitements des données, sous forme de texte, que nous avons été susceptibles d'appliquer en phase 2 selon les cas, par choix ou par nécessité.

Ils ont pour but, comme souvent en Machine Learning, de simplifier les données. Lorsque l'on traite du texte, cela consiste en partie à nettoyer les données d'éléments conventionnels ou syntaxiques non directement porteurs de sens.

Cela a été fait au moyen de fonctions python ad-hoc utilisant les modules Re, SpaCy et NLTK. Elles nous permettaient :

- Le filtrage des chiffres, de la ponctuation et des caractères spéciaux (à part exceptions…) au moyen de l'expression régulière suivante **[^a-z#+.\s]** (toujours).
- Le filtrage des « stopwords » (pronoms, préposition, articles… toujours effectué…)
- Le filtrage des formes verbales (optionnel…), dans les cas où nous ne jugions pas pertinent de les garder.
- De normaliser le texte, c'est-à-dire d'en diminuer le nombre de mots par **lemmatisation** (remplacement d'un mot par sa forme canonique) ou par **stemming** (« désuffixation »…).
- La mise en forme (optionnelle) « bag of words » des données. Il s'agissait de transformer du texte en une liste de mots alors appelés « tokens ». Cette mise en forme dépendait de si certains outils fonctionnent avec du texte ou des tokens en entrée.

3. Approche LDA, non supervisées et mixtes

3.1. LDA pour Latent Dirichlet Allocation

3.1.1. Intuition de la LDA

La **LDA**, qui signifie **Latent Dirichlet Allocation**, est un outil très courant en **NLP**. Il s'agit d'un procédé génératif probabiliste non supervisé qui sert généralement à l'extraction de thème.

On peut l'envisager comme une « machine » servant à traiter des documents en suivant deux grands principes.

Le premier est qu'un document est une distribution de probabilités de thèmes, le second est qu'un thème est une distribution de probabilités de mots, ces deux types de distributions provenant de distributions de Dirichlet, objets mathématiques interprétables comme des « densités de probabilités ».

A partir de ces principes et de distributions qu'on lui indique, notre « machine » LDA est capable de générer des documents aléatoires. Mais dans l'autre sens, et c'est ce qui nous intéresse, elle est aussi capable de procéder au « reverse engineering » de documents qu'on lui soumet pour en déduire les distributions de thèmes avec leurs distributions de mots associées.

Si on est familier d'un algorithme de segmentation comme K-Means, on peut en une certaine mesure considérer la LDA comme un « K-Means sémantique ». On lui soumet un corpus de documents sensés aborder K thèmes différents, et l'algorithme est en mesure d'assigner de manière probabiliste des thèmes principaux à chaque document[5].

3.1.2. Deux façons d'utiliser la LDA

Comme chacune de nos questions pouvait se voir assigner par LDA un thème principal (celui qui avait la plus forte probabilité) et que chaque thème avait un « mot le plus représentatif » (celui qui avait la plus forte probabilité pour ce thème) une approche totalement non supervisée était tentante. Celle-ci consistait à tagger chaque question avec le mot le plus représentatif de son thème principal.

Seconde possibilité, puisque transformant chaque question en une distribution de probabilité de K thèmes (en l'occurrence 25), la LDA pouvait par ailleurs être utilisée en tant que simple mais efficace outil de réduction dimensionnelle après l'utilisation duquel procéder à des modélisations supervisées.

Nous avons tenté les deux.

3.2. Approche non supervisée

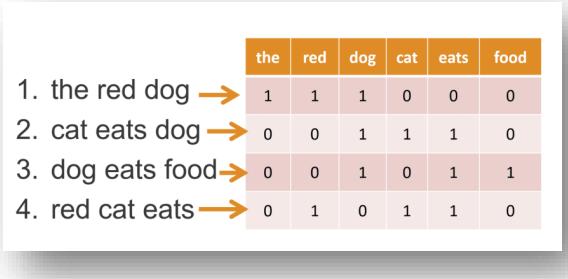
3.2.1. Vectorisation TF / TF-IDF

Avant d'utiliser la LDA, les données textes ont subi les filtrages habituels obligatoires (caractères, stopwords) ainsi que celui des formes verbales car celles-ci ne sont pas pertinentes dans un cadre d'extraction de thème.

La LDA étant un outil de phase 3, avant de la mettre en action il nous a aussi fallu numériser, ou vectoriser, les données texte. Nous avons utilisé dans ce but la technique de vectorisation TF pour « term frequency ».

Elle consiste à modéliser un corpus de textes en une matrice dont :

- chaque colonne correspondrait à un mot présent dans ce corpus
- chaque rang correspondrait à une subdivision de ce corpus (phrase d'un texte, ou dans notre cas une question de notre corpus de questions…)
- et dont chaque élément serait le nombre de fois qu'apparait tel mot dans telle subdivision.



	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Signalons une variante également très courante de cette technique, la **vectorisation TF-IDF**. Avec celle-ci, chaque terme de la matrice est divisé par le nombre total de fois qu'un mot apparait dans le corpus entier, ce qui permet de donner plus de poids à des mots rares ou spécifiques à certains éléments du corpus.

	the	red	dog	cat	eats	food
1. the red dog →	1	1/2	1/3	0	0	0
2. cat eats dog →	0	0	1/3	1/2	1/3	0
3. dog eats food →	0	0	1/3	0	1/3	1
4. red cat eats →	0	1/2	0	1/2	1/3	0

C’est parce ces techniques, chaque élément est représenté par des coordonnées dans un repère commun, qu’on parle de « vectorisation » plutôt que de « numérisation ».

Comme avec elles chaque mot d’un corpus de textes équivaut virtuellement à une dimension supplémentaire des données vectorisées, il ressort que les vectorisations TF et TF-IDF ont pour premier défaut de souvent produire des données de dimensions extrêmement élevées.

3.2.2. Approche non supervisée inopérante

Il était tentant de penser qu’une application d’un algorithme non supervisé pouvait peut-être à elle seule remplir notre mission. Néanmoins cette idée s’est révélée immédiatement inopérante du simple fait que les mots les plus représentatifs des thèmes extraits par LDA ne correspondaient pas à nos étiquettes cibles (6 tags communs obtenus seulement…).

Tags cible

```
['android', 'angular', 'asp.net', 'c#', 'css', 'docker', 'flutter', 'google', 'html', 'ios', 'java', 'ja
vascript', 'jquery', 'laravel', 'misc', 'node.js', 'pandas', 'php', 'python', 'react', 'spring', 'sql',
'swift', 'typescript', 'visual']
```

Tags déterminés par LDA

```
['key', 'android', 'java', 'server', 'table', 'angular', 'button', 'react', 'error', 'list', 'version',
'string', 'python', 'div', 'google', 'image', 'variable', 'user', 'object', 'page', 'git', 'test', 'wind
ows', 'file', 'function']
```

Sans même aborder d’autres problèmes qui seraient apparus par la suite (comme l’assignation de plusieurs tags à des questions le nécessitant…) cette approche était caduque.

L’inadéquation, assez prévisible, entre les tags obtenus et les tags voulus, montre tout simplement que l’approche mécanique de la LDA n’a pu modéliser une liste de mots résultant non pas de principes mathématiques mais d’interactions humaines. Et c’est précisément dans ce cas de figure que l’apprentissage par l’exemple supervisé prend tout son sens.

3.3. Approches LDA mixtes

3.3.1. Cadre général des modélisations supervisées du projet

Avant d'aborder les premières classifications supervisées faites à partir de la réduction en topic score à 25 dimensions obtenue par LDA, nous devons préciser leurs modalités, ainsi que celles des autres classifications du projet.

- ***Cadre de classification multi-label***

  Etant donné que les questions de notre corpus étaient susceptibles d'avoir plusieurs tags, nous étions dans un cadre de classification dite « multi-label » qui, contrairement aux classifications binaires ou multiclasse plus habituelles, nécessite une démarche particulière.

  Nous avons utilisé les outils du module **scikit.ml**, et parmi les différentes options possibles, nous avons choisie l'approche **binary relevance**. Celle-ci est pertinente lorsqu'on considère que les différentes étiquettes sont indépendantes les unes des autres, et revient à considérer qu'une classification multi-label à N labels équivaut à une série de N classifications binaires. Ce dernier point a d'ailleurs eu d'importantes répercussions sur notre projet[6].

- ***Métrique du projet***

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

$$\text{F1-Score} = 2\,\frac{precision * recall}{(precision + recall)}$$

  Tout le long du projet, nous avons mesuré les performances de nos modèles avec le **F1-Score weighted**.

  En tant que moyenne harmonique de la précision et du rappel d'un classificateur, le F1-Score ne tient pas compte des vrais négatifs et est donc la métrique idéale si on considère que la performance à mesurer est la capacité d'un modèle à ne prédire uniquement que les vrais positifs. Il était donc particulièrement adapté à notre cas, où lors de chaque classification le nombre de vrais positifs à trouver sera toujours très inférieur aux vrais négatifs.

  Dans un cadre de binary-relevance en multilabel, le F1-Score Weighted est la moyenne balancée (par la fréquence des classes à prédire) des « sous-F1-Score » de chaque « sous-classification »[7].

◆ ***Mauvaises manières…***

En raison des conditions de notre projet, à cause desquelles chaque classification multi-label revenait à effectuer en fait 25 classifications binaires, nous avons dérogé à certaines bonnes pratiques habituelles du machine learning.

Nous n'avons ainsi pas pratiqué de validations croisées (ce qui aurait encore multiplié les temps de modélisation).

Nous n'avons pas non plus effectué d'hyper-paramétrages par recherches randomisées ou par grille. A la place, nous nous sommes contentés d'hyper-paramétrages « light » des modèles par optimisation bayésienne, et ce grâce aux outils du module bayes-opt. En une quinzaine de modélisations (au lieu de beaucoup plus…) ce procédé nous a permis d'obtenir, en plus du plaisir d'insérer un peu d'inférence bayésienne dans le projet, des gains de performance sensibles[8].

3.3.2. Première modélisation, baseline et choix de l'algorithme.

Pipeline : filtrages habituels + verbes – vectorisation TF – LDA

Cette première modélisation a été l'occasion d'établir une baseline et de procéder au choix de l'algorithme que nous utiliserions pour le reste du projet.

La baseline était le résultat d'une prédiction naïve et constante du tag « misc » qui était le plus fréquent. Elle était de :

➔ **F1w baseline : 0.046**

Quant à notre choix d'algorithme, après avoir également testé la régression logistique et random forest, il s'est porté sur la « star » XGBoost qui nous a immédiatement procuré les meilleurs résultats pour un besoin de ressource jugé raisonnable.

Après optimisation bayésienne le score obtenu sur cette première modélisation a été de :

➔ **F1w lda : 0.36** **Tps d'entrainement : 36s.**

3.3.3. Seconde modélisation à partir de textes normalisés.

Une seconde modélisation a été effectué à partir d'une autre LDA effectuée à partir de données texte normalisées. Le pipeline était alors le suivant :

Pipeline : filtrages habituels + verbes – lemmatisation + stemming – vectorisation TF – LDA – XGBoost

Lors de différents essais, nous avons remarqué que l'étape de normalisation ne semblait que peu modifier nos données texte. Probablement à cause de leur caractère très spécialisé et d'un vocabulaire souvent hors des dictionnaires utilisés par certains outils de normalisation (sans oublier le filtrage des formes verbales). Du coup, nous avons été assez surpris d'observer que même si l'étape de normalisation avait sûrement moins apporté dans notre cas, elle nous avait quand même permis une amélioration visible du score :

➔ **F1w lda norm : 0.38 Tps : 26s.**

4. Les bienfaits du word embedding

4.1. Bilan d'étape du projet

Pour accomplir notre projet, la première piste à laquelle nous avons pensé a été la LDA en tant qu'outil d'extraction de thème. Sur un malentendu cela pouvait peut-être marcher mais cela n'a pas été le cas.

Ensuite nous avons utilisé la LDA en tant que modèle capable de modéliser nos questions vectorisées dans un espace réduit à 25 dimensions, ce qui nous a permis d'obtenir les résultats ci-dessus.

Mais quelles options s'offraient alors à nous pour aller plus loin ?

L'idée immédiate était de tenter des modélisations à partir des données texte vectorisées TF, sauf que les dimensions de celles-ci (plusieurs milliers) annonçaient des opérations beaucoup trop complexes et coûteuses.

En fait nous avions besoin d'un nouveau moyen, intervenant en phase 3 au regard de notre découpage du projet, nous permettant de vectoriser, si possible d'une façon pertinente, une grande quantité de texte dans un espace de dimension « acceptable ». Et justement, une telle technique existe, c'est le word embedding !

4.2. Qu'est-ce que le word embedding ?

Apparue en 2013, le word embedding a révolutionné le NLP. Il s'agit d'une technique non supervisée de vectorisation de texte basée sur l'idée générale qu'un mot dépend de son contexte, c'est-à-dire de ses voisins dans un texte.

Ce procédé est intéressant à au moins deux titres. Il permet de modéliser le sens des mots, ou du moins de rendre compte des proximités sémantiques de

différents mots. Il permet aussi de modéliser d'immense corpus, voire des langues entières, dans des espaces relativement réduits de l'ordre de quelques centaines de dimensions. Ces deux apports, une vectorisation plus « riche », ainsi que la possibilité de dompter le « fléau de la dimensionnalité », ont permis de grands progrès en NLP.

On peut pratiquer le word embedding en créant ses propres modèles ou en utilisant des modèles déjà pré-entrainés. Cette technologie, et certaines de ses déclinaisons, est disponible dans des bibliothèques python libres d'accès.

Dans le cadre de ce projet, nous avons utilisé le word embedding Spacy, le word2vec de Gensim, ainsi que la technologie FastText, pour transformer nos données et voir laquelle de ces options nous permettait d'effectuer ensuite la meilleure modélisation.

Dernier point, une transformation par word embedding consiste à remplacer un mot par son vecteur[9]. Une « feature », c'est-à-dire un texte, une phrase, ou dans notre cas une question de Stack Overflow, est la moyenne des vecteurs des mots la composant[10].

4.3. Réduisons toujours plus avec la PCA !

Même si en lui-même le word embedding semble régler ou contenir les problèmes de dimension, en machine learning la lutte contre la complexité, à des fins d'économie de ressources, ne s'arrête jamais.

C'est pourquoi, après les vectorisations par word embedding, nous avons systématiquement tenté de réduire encore plus nos données par PCA.

La PCA (pour Principal Composant Analysis) revient à exprimer les données dans un nouveau repère dont les nouveaux axes sont les axes de variances maximales des données. Cette opération, qui s'apparente en un sens à un jeu d'écriture, permet souvent de réduire la dimension de données, ou d'opérer des compromis, en permettant de sacrifier une petite part de la « richesse » des données contre des gains de performance lors des modélisations[11].

4.4. Mise en pratique

4.4.1. Word embedding SpaCy

Nous avons commencé par le modèle pré-entrainé SpaCy de langue anglaise « large ». Il s'agit d'un modèle à 300 dimensions riche de 1.5 millions de mots dont l'utilisation est facilitée par le fait qu'il accepte qu'on lui soumette les mots hors de son vocabulaire.

Le pipeline était le suivant, filtrage standard, vectorisation avec SpaCy, standardisation (nécessaire avant PCA), réduction à 150 dimensions par PCA (au prix de 10% de la richesse des données…) puis XGBoost avec optimisation bayésienne.

Pipeline : filtrages hab. – SpaCy – STD – PCA n = 150 – XGBoost opti.

➔ **F1w spacy : 0.54** **Tps : 264s.**

Le score avait fait un véritable bond par rapport au précédent obtenu. Néanmoins, si SpaCy « acceptait » les mots hors de son vocabulaire, il n'en tenait cependant pas compte. Et au final, ce score avait donc été obtenu en laissant de côté 74% des mots uniques présents dans nos données…

4.4.2. Word2vec Gensim

Comme précédemment, avec les outils du module Gensim nous avons d'abord cherché à utiliser un modèle pré-entrainé supposé robuste. Il s'agissait en l'occurrence d'une modélisation encore à 300 dimensions, riche de 3 millions de mots, et entrainée sur toute l'encyclopédie en ligne Wikipedia par des équipes Google.

Malheureusement, ce modèle n'acceptant aucun terme hors de son vocabulaire, nous avons dû filtrer plus des trois-quarts des mots uniques des données. A la suite de quoi les résultats de modélisation ont été mauvais.

Du coup nous avons entrainé notre propre modèle word2vec ad-hoc, que nous avons choisi à 100 dimensions. Avec le pipeline suivant, nous avons obtenu :

Pipeline : filtrages habituels – custom W2V – STD – PCA n = 90 – XGBoost opti.

➔ **F1w cust w2v : 0.59** **Tps : 194s.**

4.4.3. Fast Text

Enfin pour terminer, nous avons testé la technologie Fast Text également disponible dans le module Gensim. Mise au point par des équipes Facebook, il s'agit d'une déclinaison du word embedding initial qui ne se base pas

directement sur les mots, mais sur leur morphologie, en s'intéressant aux sous-parties qui les composent.

Comme cette technologie est réputée efficace pour modéliser les corpus restreints, nous avons directement créé notre propre modèle avec le pipeline suivant :

Pipeline : filtrages habituels – custom fastText – STD – PCA n = 75 – XGBoost opti.

➔ **F1w cust fasttext : 0.61 Tps : 184s.**

Il s'agissait de notre meilleur résultat qui serait, pensions-nous, notre résultat final.

4.4.4. Remarques

En comparant les résultats obtenus avec le modèle pré-entrainé SpaCy (un modèle donc robuste) et les suivants obtenus avec nos propres modèles Word2vec et Fast Text (plus sommaires mais entrainés sur toutes les données), on peut remarquer dans notre cas d'une part qu'ils sont assez proches, mais aussi d'autre part que la prise en compte des données l'avait emporté sur la qualité des modèles.

Cela faisait dès lors ressortir l'importance de pouvoir entrainer ses propres modèles si les données à disposition sont éloignées de celles avec lesquelles les modèles pré-entrainés avaient été bâtis.

En machine learning, l'idée générale que plus on dispose de données, meilleurs seront les résultats tient de l'évidence. Mais c'était doublement vrai dans notre cas, où en plus de permettre une meilleure modélisation finale, plus de données auraient aussi favorisé de meilleures modélisations word embedding, ce qui aurait également contribué à une amélioration des performances en bout de chaîne.

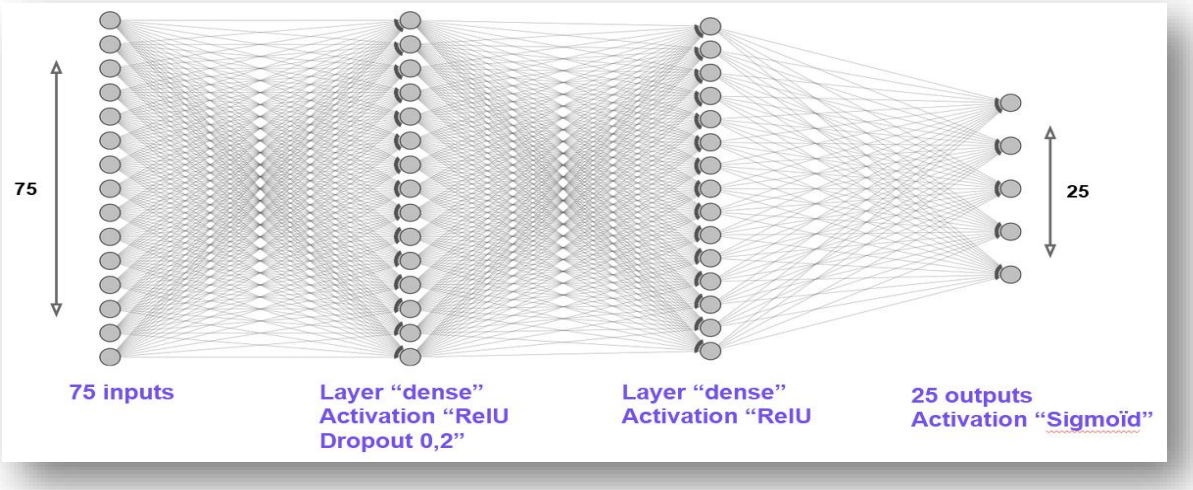
5. Conclusion : la piste du deep learning

A ce stade, après que le lecteur ait probablement déjà rencontré beaucoup de nouvelles choses à digérer, nous avions prévu de conclure. Mais plutôt que de revenir sur le chemin parcouru, nous souhaitons au contraire terminer en poussant les choses encore plus loin avec le fameux **deep learning**.

En effet, même si cela transpire plus dans les notebooks, où figurent les blocs de code et les temps de modélisation, que dans ce compte rendu, la nature multi-label

des modélisations effectuées a significativement compliqué le projet. Or, il se trouve qu'en respectant des conditions simples, le deep learning gère justement de façon native le multi-label.

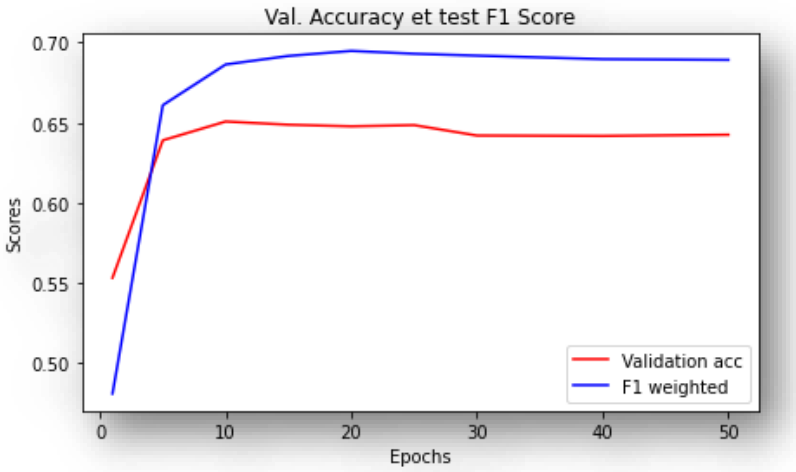
Sans rentrer dans les détails, juste pour stimuler la curiosité du lecteur, nous allons donc conclure sur une dernière modélisation utilisant un réseau de neurones, que voici :



Prévu pour fonctionner à partir du « pipeline » de notre dernière modélisation (Fast Text et PCA avec n=75…), ce réseau avait pour caractéristiques :

- 75 entrées
- 2 couches denses à 75 neurones avec activation RelU, la première avec en plus un dropout de 0,2.
- 25 sorties avec activation sigmoïde (conditions pour gérer naturellement notre multi-label à 25 classes).

Entrainé sur 20 « epochs », ce qui revenait à un temps d'entrainement d'environ 7 secondes, ce réseau neuronal très simple a donné les résultats suivants :



Pipeline : filtrages habituels – custom fastText – STD – PCA n = 75 – NN

➔ **F1w nn : 0.69 Tps: 7s.**

Le gain de performance et l'économie de ressources ont été spectaculaires.

Soulignons que ce résultat a été obtenu au moyen d'un **neural network** totalement basique. Il existe des types de réseaux plus avancés, capables de prendre en compte des séquences de mots, qui sont encore bien plus performants en NLP (Réseaux récurrents, LSTM, technologie BERT…).

Alors que nous arrivons à la fin de ce compte-rendu, qui avait pour but de familiariser le lecteur avec certaines bases du NLP, nous espérons que ce petit twist de fin l'incitera à creuser le sujet plus loin !

ANNEXES :

Annexe 1 : Tester le modèle en ligne

Les choses évoquées dans ce rapport ne sont pas que virtuelles !

Grâce au module **flask** et au service **eu.pythonanywhere.com**, nous avons créé une API permettant de tester pour un temps limité notre modèle final à l'adresse suivante :

markofr.eu.pythonanywhere.com

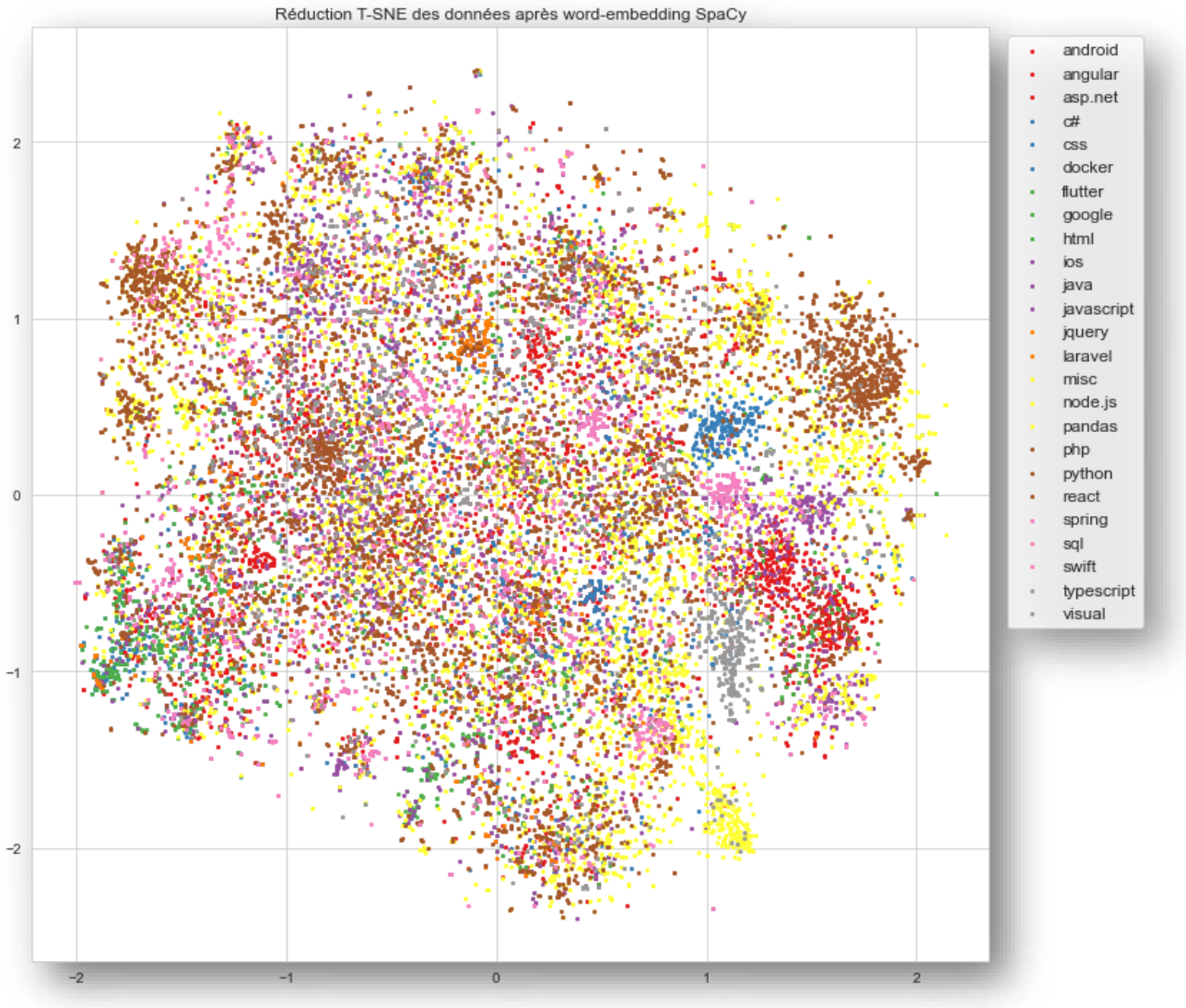
Annexe 2 : Représentations T-SNE de données

T-SNE est un algorithme de réduction dimensionnelle de données qui permet des représentations particulières rendant compte des **proximités** entre éléments.

Nous avons produit quelques figures de réductions T-SNE des données d'entrainements, après vectorisation par word embedding, afin de vérifier si leurs représentations traduisaient en images d'éventuelles proximités sémantiques.

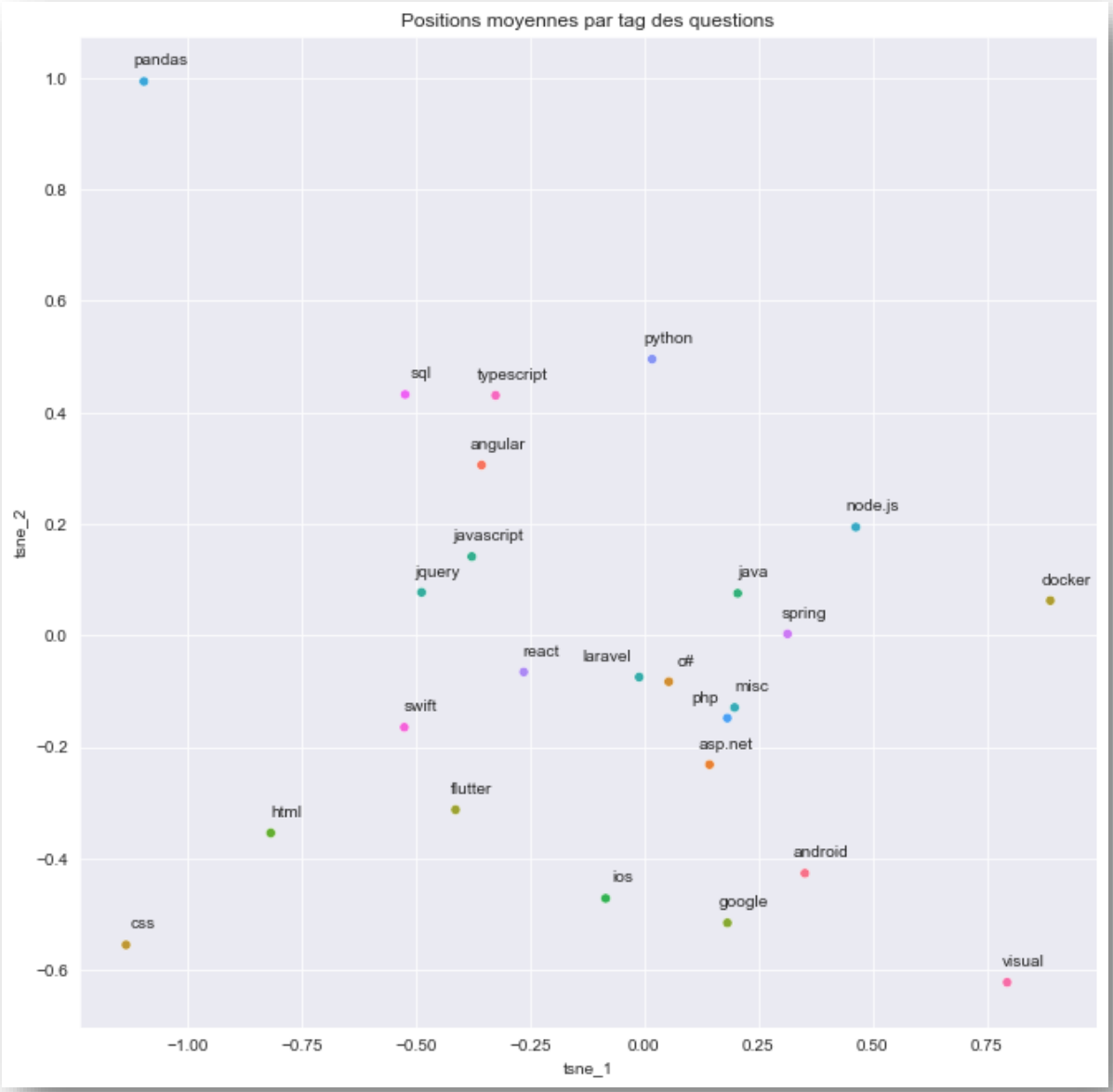
(illustration en page suivante…)

- **Représentation globale**



Même si le résultat reste assez brouillon, d'autant qu'il s'agit ici de représentation de données vectorisées par le modèle pré-entrainé SpaCy (qui ne tenait pas compte d'une grande partie des données ne figurant pas dans son « vocabulaire »), on observe en effet divers regroupements de couleurs illustrant la proximité de questions partageant des tags identiques.

- **Représentation par tag**



Si nous représentons maintenant les coordonnées moyennes des questions regroupées par tag, nous pouvons encore observer des proximités intéressantes, par exemple « google », « android » et « IOS » qui traitent d'applications mobiles, etc.

[1] Réf. : https://algorithmia.com/blog/introduction-natural-language-processing-nlp

[2] Réf. : https://stackexchange.com/

[3] Réf. : https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[4] Réf. : https://spacy.io/usage/spacy-101

[5] Réf. : https://www.mygreatlearning.com/blog/understanding-latent-dirichlet-allocation/

[6] Réf. : http://scikit.ml/

[7] Réf. : https://datascience.stackexchange.com/questions/40900/whats-the-difference-between-sklearn-f1-score-micro-and-weighted-for-a-mult

[8] Réf. : https://github.com/fmfn/BayesianOptimization

[9] Réf. : https://datascientest.com/nlp-word-embedding-word2vec

[10] Réf. : https://radimrehurek.com/gensim/auto_examples/tutorials/run_fasttext.html

[11] Réf. : https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60