Comparative Study To Solve Text Categorization

CES Data Scientist 2017-2018 (Telecom ParisTech)

Jacques Doan-Huu

Abstract

In the last decade, **D**eep **L**earning (**DL**) has demonstrated outstanding performance in the field of computer vision beating indisputably traditional methods, thanks to the GPU performance leapfrog and the huge amount of labeled datasets. A bit more recently, DL also went into the **N**atural **L**anguage **P**rocessing (**NLP**) field battle to solve common NLP problems like text classification and translation, with very promising perspectives and results: in particular, word embedding and **R**ecurrent/**C**onvolutional **N**eural **N**etwork (**RNN/CNN**) architectures provide efficient technical responses to NLP challenge.

POSOS French startup has submitted a data challenge for which I took the opportunity to verify humbly whether DL is a suitable solution compared to traditional methods, for a beginner like me having very few experiences on NLP/DL area and low-end hardware system (DL has the bad reputation to be numerically intensive…).

Contents

Statement of the Problem

The data challenge is plainly described at ENS school web site (link) and it consists in categorizing into **51** intents, drug related questions written in natural language (French to be precise). **POSOS** claimed to get good performance with 86% accuracy by choosing DL: they don't supply any details on the DL architecture nor any engineering clues except the recommendation to extract some key information procured by the French drug administration (**ANSM**).

The target categories (question intent) have been intentionally anonymized into indices from 0 to 50: hiding their respective semantic is probably aimed to avoid the usage of topic-specific (and so biased) procedures. Training dataset contains only ~8000 questions: it's pretty short to produce a good learning outcome. Besides, the text suffers from many anomalies (misspelling, grammatical incorrectness, familiar acronym, …) and employs specific medical vocabulary (drug name like “mirtazapine”, …): it hardens the challenge level of difficulty.

Project Motivation

The purpose of this study is to compare fairly the strength and weakness between DL and non-DL approaches from different perspectives:

- model accuracy
- model interpretability
- practicality
  - tooling
  - hardware constraints
  - training time
- sustainability

The idea is not to achieve a good performance at any price, but an attempt to explore comparatively the end to end methodology to tackle a text categorization problem throughout 2 distinct technologies.

“Traditional techniques” refer to any ML algorithms which don’t rely on neural network theory (eg: Word2Vec is excluded): to quote some of them, Hidden Markov Model, XGBoost, SVM, logistic regression and PCA are eligible.

Conversely, DL option should rely uniquely on neural network but it can as well benefit from “neutral” text preprocessing (feature enrichment with external source, stemming, stopWords, …) for fairness sake.

ML Workbench Environment

All experiments have been written in Python in the popular Jupyter environment: the notebooks are available publicly as a github project whose details are provided in the annex section. I used many python packages to satisfy various requirements:

data manipulation and visualization: pandas, numpy and matplotlib

text processing (stemming, stopWords, …): NLTK, standard regex and spellChecker (built from github)

ML algorithms (XGBoost, PCA, t-SNE): sklearn and XGBoost

DL framework: Keras + Tensorflow

Most of packages have been installed as is, except for XGBoost I recompiled locally from its github source code to get the GPU accelerated version which is not shipped officially.

Besides above runtime packages, this project also takes advantage of public resource or pretrained models (NLTK corpus, FastText word embedding model, ….).

ML jobs had been initially executed with an old MacBook Pro whose chipset was damaged by the heating due to the overnight DL train, then with a many CPU core/low-end GPU PC workstation.

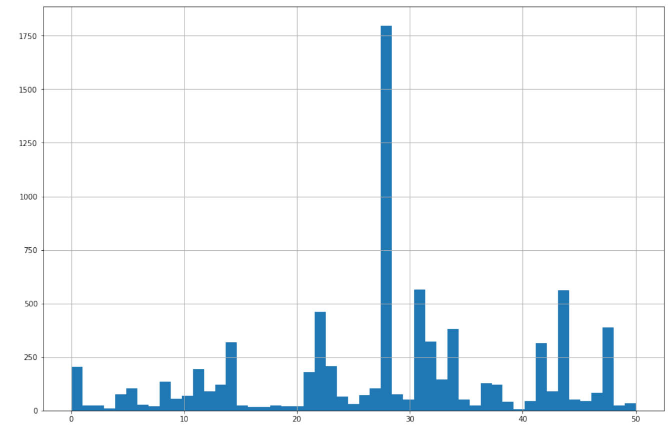
I finally rent a GPU boosted Amazon instance: even if the GPU/CPU resource is not utilized, the data storage is charged permanently making the overall cost very high (5$/h) during 2 full days.

At the very end, the best money saving option was to buy a PC gamer machine with mid-range Nvidia GPU card to carry out the computing workload: I remarked an important speedup at training time.

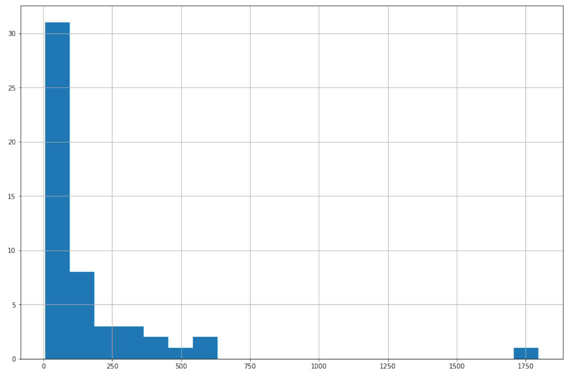
Data Exploration

Data Distribution

The target distribution of the 51 labels is imbalance with a peak at intention=28.



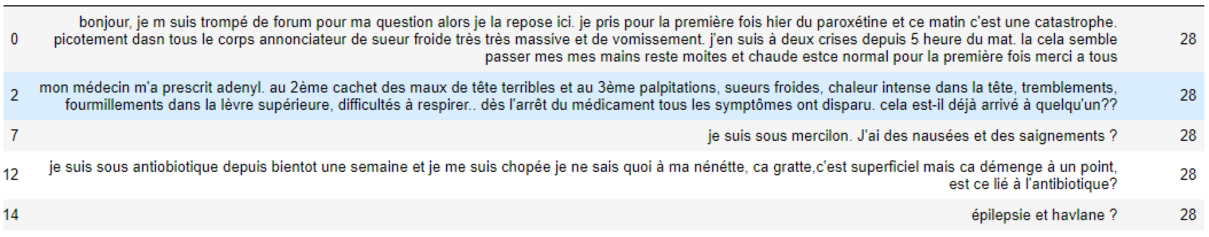
Most of labels are associated to pretty small number of samples: half of classes have less than 100 rows, it sounds that the classifier would underperform on such labels with low input features.



Let's try to guess intuitively the hidden semantic of the most frequent labels:

first mode: intention=28

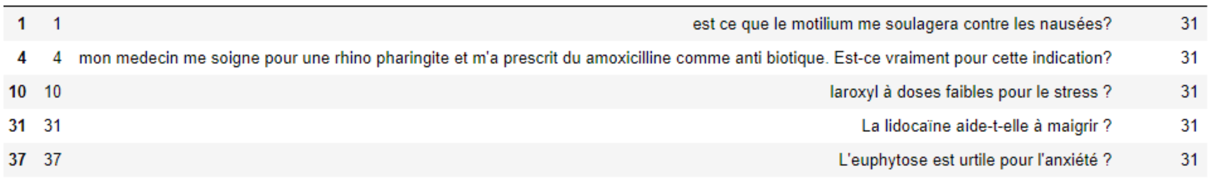
It's likely relate to drug adverse effects (contraindications)



0	bonjour, je m suis trompé de forum pour ma question alors je la repose ici. je pris pour la première fois hier du paroxétine et ce matin c'est une catastrophe. picotement dasn tous le corps annonciateur de sueur froide très très massive et de vomissement. j'en suis à deux crises depuis 5 heure du mat. la cela semble passer mes mes mains reste moites et chaude estce normal pour la première fois merci a tous	28
2	mon médecin m'a prescrit adenyl. au 2ème cachet des maux de tête terribles et au 3ème palpitations, sueurs froides, chaleur intense dans la tête, tremblements, fourmillements dans la lèvre supérieure, difficultés à respirer.. dès l'arrêt du médicament tous les symptômes ont disparu. cela est-il déjà arrivé à quelqu'un??	28
7	je suis sous mercilon. J'ai des nausées et des saignements ?	28
12	je suis sous antiobiotique depuis bientot une semaine et je me suis chopée je ne sais quoi à ma nénétte, ca gratte,c'est superficiel mais ca démenge à un point, est ce lié à l'antibiotique?	28
14	épilepsie et havlane ?	28

second mode: intention=31

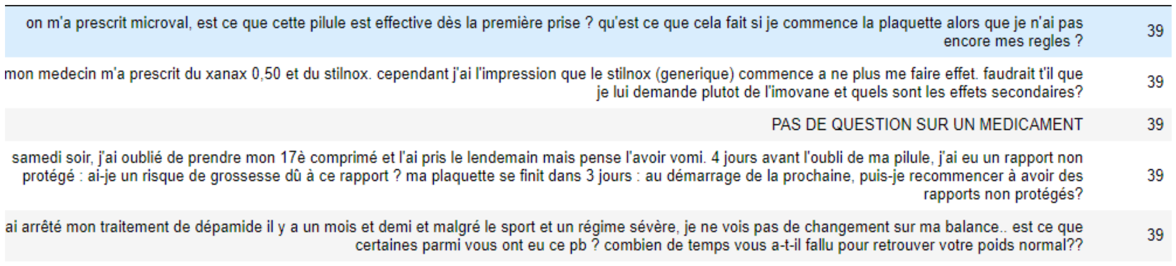
it's about drug>disease indication/efficiency



1	1	est ce que le motilium me soulagera contre les nausées?	31
4	4	mon medecin me soigne pour une rhino pharingite et m'a prescrit du amoxicilline comme anti biotique. Est-ce vraiment pour cette indication?	31
10	10	laroxyl à doses faibles pour le stress ?	31
31	31	La lidocaïne aide-t-elle à maigrir ?	31
37	37	L'euphytose est urtile pour l'anxiété ?	31

multi-topic class: intention=39

the commonality across text samples seems to the presence of multiple question mark tokens (counting it may be a good option to predict mult-topic label)

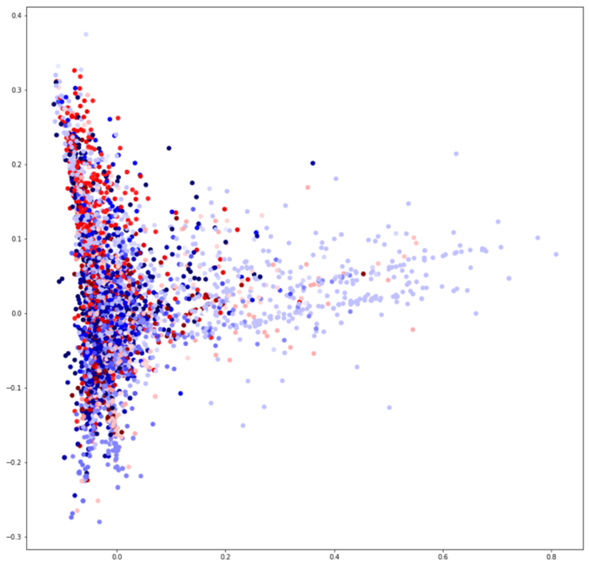


on m'a prescrit microval, est ce que cette pilule est effective dès la première prise ? qu'est ce que cela fait si je commence la plaquette alors que je n'ai pas encore mes regles ?	39
mon medecin m'a prescrit du xanax 0,50 et du stilnox. cependant j'ai l'impression que le stilnox (generique) commence a ne plus me faire effet. faudrait t'il que je lui demande plutot de l'imovane et quels sont les effets secondaires?	39
PAS DE QUESTION SUR UN MEDICAMENT	39
samedi soir, j'ai oublié de prendre mon 17è comprimé et l'ai pris le lendemain mais pense l'avoir vomi. 4 jours avant l'oubli de ma pilule, j'ai eu un rapport non protégé : ai-je un risque de grossesse dû à ce rapport ? ma plaquette se finit dans 3 jours : au démarrage de la prochaine, puis-je recommencer à avoir des rapports non protégés?	39
ai arrêté mon traitement de dépamide il y a un mois et demi et malgré le sport et un régime sévère, je ne vois pas de changement sur ma balance.. est ce que certaines parmi vous ont eu ce pb ? combien de temps vous a-t-il fallu pour retrouver votre poids normal??	39

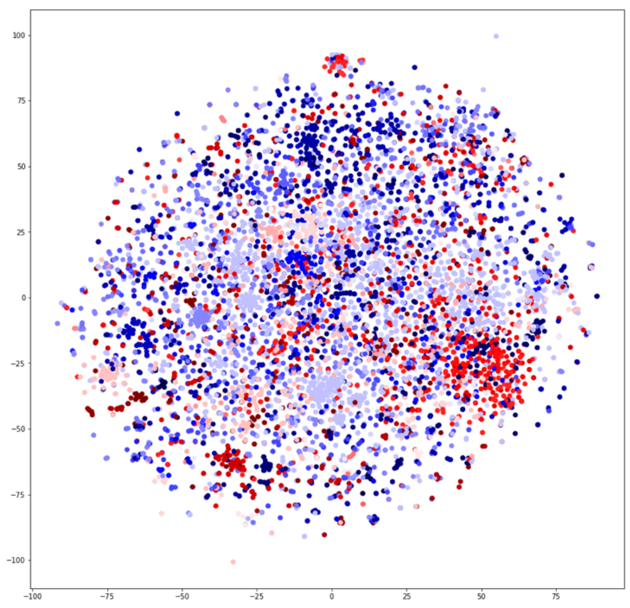
To have an idea on the classification task difficulty, it's a common practice to visualize the feature space distribution.

The document is basically transformed into of **BOW** (Bag Of Words) which is then vectorized with TF-IDF encoder: each document is consequently represented as a data point within the global vocabulary space. To make such data points human readable, a dimension reduction of these features is necessary at the cost of some approximations: I used both linear/fast PCA dimension reduction and non-linear/slow t-SNE techniques. The data point color determines the associated target label.

PCA-reduced feature space



t-SNE-reduced feature space



Both 2D distribution shapes are very dissimilar but they consistently tend to indicate that feature space cannot be partitioned per label just by considering the occurrence of words. To get a chance to achieve a better classification performance, it's obvious that the raw text has to be encoded more smartly into a suitable space where the semantic proximity between documents is prevailing.

Text Anatomy Analysis

Let's look at 2 samples having the same label (disease-drug adequacy):

"épilepsie et havlane?"

"mon medecin me soigne pour une rhino pharingite et m'a prescrit du amoxicilline comme anti biotique. Est-ce vraiment pour cette indication?"

Even if they share the same question topic, the writing styles are completely opposite: on one hand, a very concise expression putting the disease entity and the drug entity in an adversarial fashion, on the other hand, the second sample is more descriptive and spread over 2 sentences.

This example is a manifest of the stylish complexity of the human language to convey an idea and a topic!

The second sample has 2 sentences: the first one installs the question context and the current situation ("mon médecin me soigne…" whereas the second one raises the effective question ("Est-ce que….").

In multi-sentence documents, I observed generally this sequential structure: first the context setup, followed by the concrete question.

3 entities appear to be salient here:

- the drug product name ("amoxicilline", "havlane")
- the disease ("épilepsie", "rhino-pharingite")
- the link entity between above ("et", "indication")

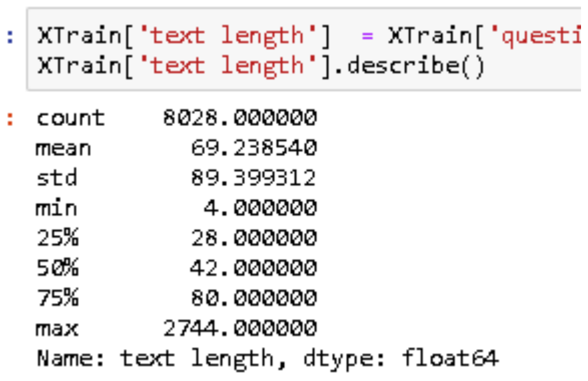
Identifying the first 2 entities is doable just based on lexical semantic domain: basically, build an exhaustive list of symbols related to drug product or disease. It's related to **NER** (Named Entity Recognition) task.

The last entity connecting the 2 other entities is much more difficult to locate: the entity semantic is contextual and depends on the presence of other entities and its relative position within the text grammatical structure. It means that the learning procedure should be a **sequence modeling**.

Other tokens (mon, médecin, me, soigne, …) seem to be superfluous to extract the question intent: text processing to remove irrelevant words is highly recommended (**stopwords**, custom regular expression, ..)

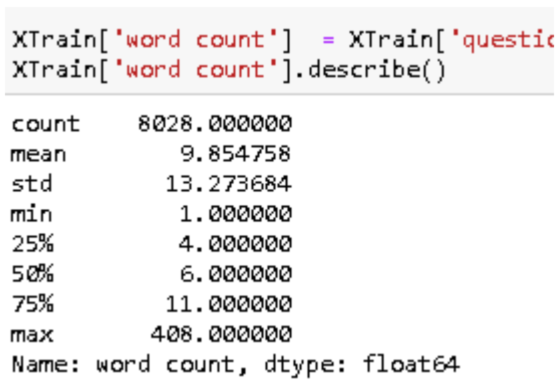
Last but not the least, some documents are lexically and syntactically incorrect: words are misspelled especially when dealing with drug product names which are unfamiliar for most of non-professional persons. A **misspelling correction** is required in the text processing phase.

Following statistics on text shows that on average, the document is concise (10 words, 69 characters)



```
XTrain['text length']  = XTrain['questi
XTrain['text length'].describe()
```

```
count     8028.000000
mean        69.238540
std         89.399312
min          4.000000
25%         28.000000
50%         42.000000
75%         80.000000
max       2744.000000
Name: text length, dtype: float64
```

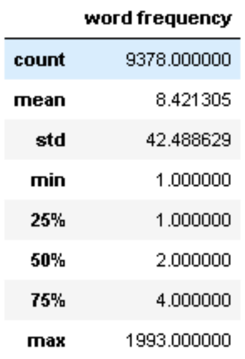


```
XTrain['word count']  = XTrain['questic
XTrain['word count'].describe()
```

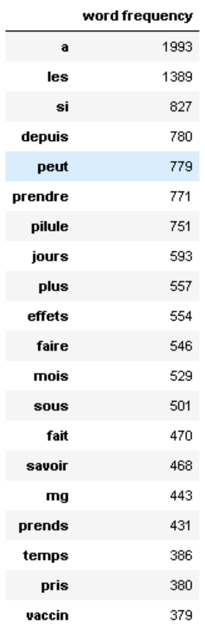
```
count    8028.000000
mean        9.854758
std        13.273684
min         1.000000
25%         4.000000
50%         6.000000
75%        11.000000
max       408.000000
Name: word count, dtype: float64
```

The vocabulary is significantly big (9378 words) and words are on average infrequent (8 occurrences). When inspecting the most frequent words, common used (but low informative) terms rank first unsurprisingly and only 2 medication related terms come out (pillule and vaccin).

Another interesting point is the presence of 3 morphological variants of the root “prendre” in the top 20: a **stemming/lemmatization** or **word embedding** are welcome to collapse such semantically equivalent variants into a single representative.



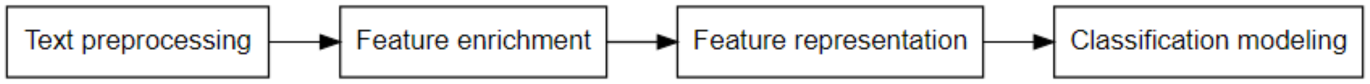
	word frequency
count	9378.000000
mean	8.421305
std	42.488629
min	1.000000
25%	1.000000
50%	2.000000
75%	4.000000
max	1993.000000



	word frequency
a	1993
les	1389
si	827
depuis	780
peut	779
prendre	771
pilule	751
jours	593
plus	557
effets	554
faire	546
mois	529
sous	501
fait	470
savoir	468
mg	443
prends	431
temps	386
pris	380
vaccin	379

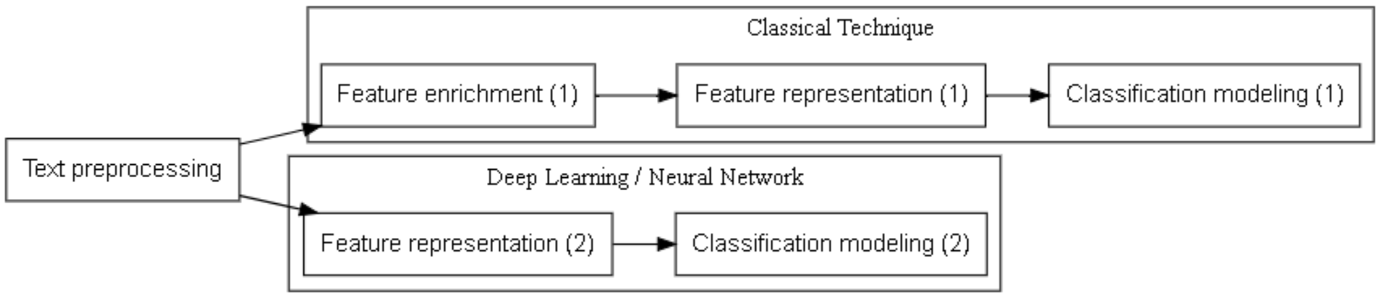
General NLP Architecture

Text classification is a common but non-trivial NLP topic going through the following main steps:



This is the general NLP text classifier framework/guidance but for practical reasons, some processing steps are skipped or significantly simplified to fit the project timeframe but also because of the lack of French language support.

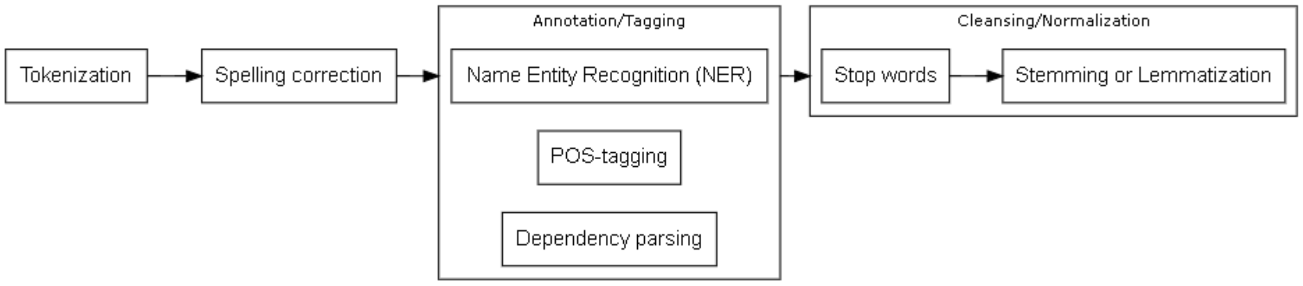
In fact, here's the concrete pipeline I built per technical scenario:


Classical Technique
Feature enrichment (1)
Feature representation (1)
Classification modeling (1)
Text preprocessing
Deep Learning / Neural Network
Feature representation (2)
Classification modeling (2)

Each processing unit will be described more precisely in the next sections.

The overall modeling procedure should capture the sequential nature of the text to exploit efficiently the contextual information: typically, the feature representation should preserve the word/symbol order and the classification process should be based on **sequence modeling**.

Text Preprocessing *(common trunk)*



It's all about operations on the raw text to make it more reliable/workable in order to extract relevant characteristics. It falls into 4 categories:

- tokenization breaking down the sentence into a sequence of atomic words
- spelling correction on misspelled words
- lexical and grammar tagging which basically decorates the text tokens with metadata
- text cleansing and normalization simplifying the sentence composition

Tokenization

This operation is a commonplace: I simply used the python string split() function. I tokenized the whole document by ignoring the punctuation like ".",":",";", "!" and "?".

Spelling correction

I assumed as misspelled all words which don't belong to any trustworthy vocabularies, also known as **OOV** (Out Of Vocabulary) word.

I retained 3 reference vocabularies in the following priority order:

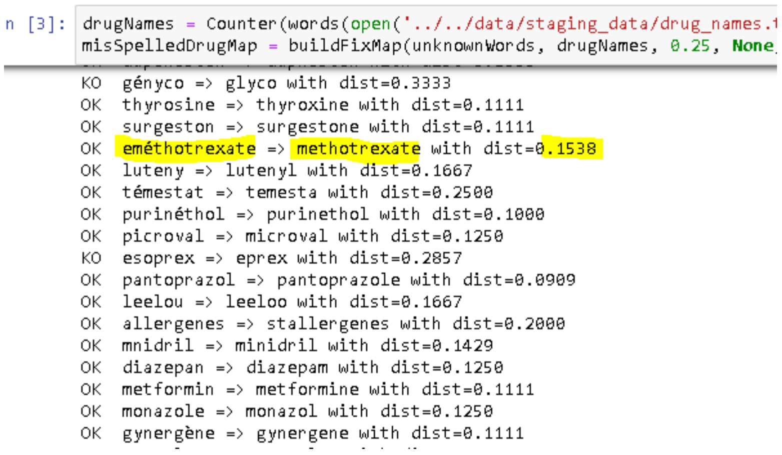
- vocabulary from the word embedding model used downstream in the processing pipeline
    - indeed, it's very important to avoid random vectorization on OOV words
- Custom vocabulary to capture the specific drug domain, typically on drug product and active ingredient entities where misspelling is frequent. It has been built from the public RCP (Résumé des Caractéristiques du Produit) repository supplied by ANSM.
- Predefined general purpose vocabulary from the github python project pysspellchecker https://github.com/barrust/pyspellchecker

The curative algorithm finds from a set of vocabularies, the closest word candidate from **Levenhstein** distance standpoint: this distance measures the minimum number of character operations (change, remove, add) required betwwen 2 words.

I defined an arbitrary threshold to accept the closest word as a fix on the misspelled word: the ratio between the number of atomic operations and the total number of characters should be under 25%.

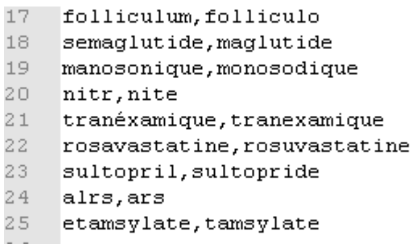
I applied this algorithm with the last 2 vocabularies: the first vocabulary layer only filters out the recognized words, the unfixed words at the second layer are then passed to the third layer.

Here's the python output showing the fix on more than 400 drug product names with a reasonable error rate (~ 15%):



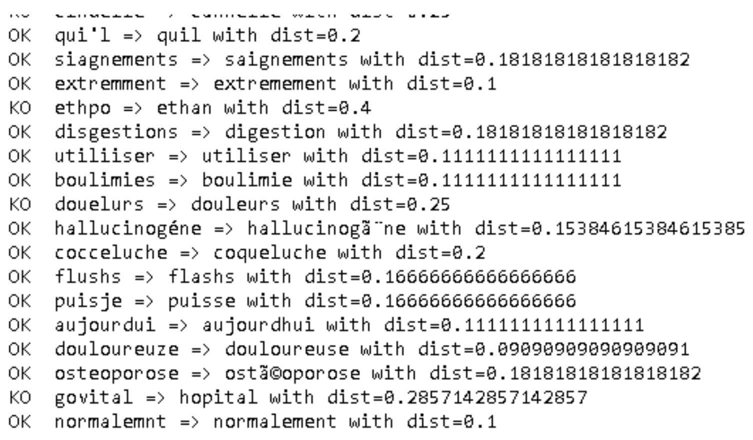
```
n [3]:  drugNames = Counter(words(open('../../data/staging_data/drug_names.t
        misSpelledDrugMap = buildFixMap(unknownWords, drugNames, 0.25, None

        KO  gényco => glyco with dist=0.3333
        OK  thyrosine => thyroxine with dist=0.1111
        OK  surgeston => surgestone with dist=0.1111
        OK  eméthotrexate => methotrexate with dist=0.1538
        OK  luteny => lutenyl with dist=0.1667
        OK  témestat => temesta with dist=0.2500
        OK  purinéthol => purinethol with dist=0.1000
        OK  picroval => microval with dist=0.1250
        KO  esoprex => eprex with dist=0.2857
        OK  pantoprazol => pantoprazole with dist=0.0909
        OK  leelou => leeloo with dist=0.1667
        OK  allergenes => stallergenes with dist=0.2000
        OK  mnidril => minidril with dist=0.1429
        OK  diazepan => diazepam with dist=0.1250
        OK  metformin => metformine with dist=0.1111
        OK  monazole => monazol with dist=0.1250
        OK  gynergène => gynergene with dist=0.1111
```

For the active ingredient, only 25 fixes have been detected.



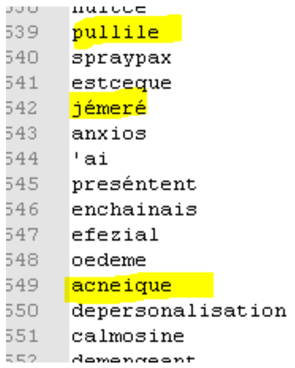
```
folliculum,folliculo
semaglutide,maglutide
manosonique,monosodique
nitr,nite
tranéxamique,tranexamique
rosavastatine,rosuvastatine
sultopril,sultopride
alrs,ars
etamsylate,tamsylate
```

The general vocabulary fixes up 430 words with relative high error rate (~25%): as accent encoding is badly handled by pyspellchecker module, I fixed it manually afterwards.



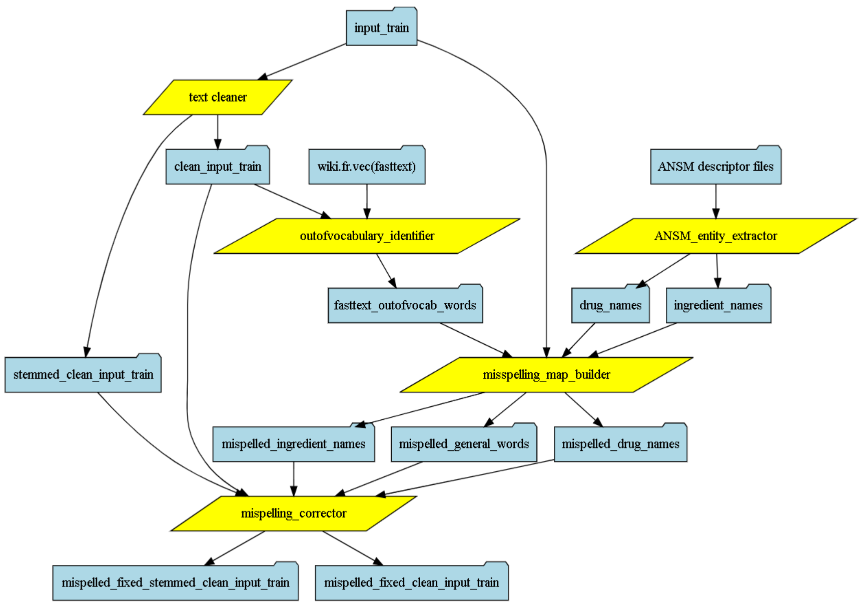
```
OK  qui'l => quil with dist=0.2
OK  siagnements => saignements with dist=0.18181818181818182
OK  extremment => extremement with dist=0.1
KO  ethpo => ethan with dist=0.4
OK  disgestions => digestion with dist=0.18181818181818182
OK  utiliiser => utiliser with dist=0.1111111111111111
OK  boulimies => boulimie with dist=0.1111111111111111
KO  douelurs => douleurs with dist=0.25
OK  hallucinogéne => hallucinogã¨ne with dist=0.15384615384615385
OK  cocceluche => coqueluche with dist=0.2
OK  flushs => flashs with dist=0.16666666666666666
OK  puisje => puisse with dist=0.16666666666666666
OK  aujourdui => aujourdhui with dist=0.1111111111111111
OK  doulouereze => douloureuse with dist=0.09090909090909091
OK  osteoporose => ostã©oporose with dist=0.18181818181818182
KO  govital => hopital with dist=0.2857142857142857
OK  normalemnt => normalement with dist=0.1
```

At the very end, it remains 583 unfixed words over an intial2108 unknown words (25%): that corresponds to hard cases where the word is unexpectedly a concatenation of multiple word or transcribed phonetically.



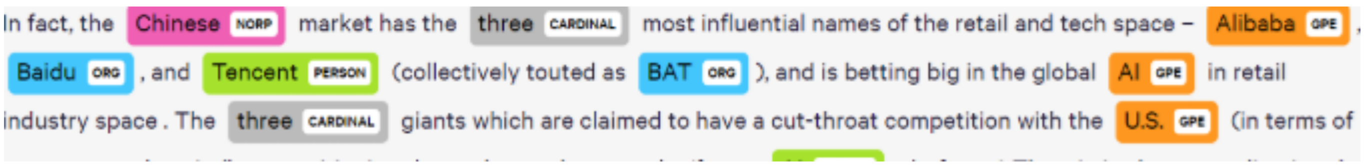
```
pullile
spraypax
estceque
jémeré
anxios
'ai
preséntent
enchainais
efezial
oedeme
acneique
depersonalisation
calmosine
```

Below diagram shows the different python notebooks (parallelogram in yellow) necessary to fix word misspelling: the blue folder represents the file consumed or produced by the python processing unit.



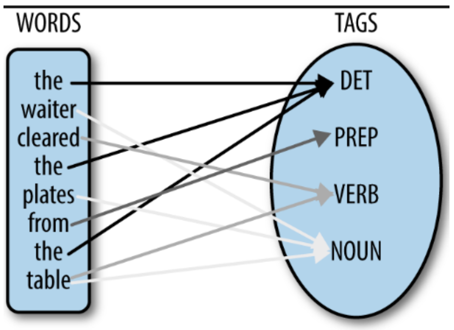
Lexical and Grammar Tagging

NER is a NLP process to tag text token with predefined categories (location, person, quantity, …), permitting to count such entities as explanatory feature. Typically, distinct drug product counting may be a discriminating feature to predict the “drug interaction” label.

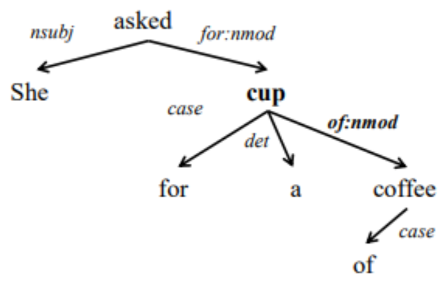


In fact, the Chinese NORP market has the three CARDINAL most influential names of the retail and tech space – Alibaba GPE , Baidu ORG , and Tencent PERSON (collectively touted as BAT ORG), and is betting big in the global AI GPE in retail industry space . The three CARDINAL giants which are claimed to have a cut-throat competition with the U.S. GPE (in terms of

POS (Part Of Speech) tagging is a process to markup text tokens with lexical categories (noun, adjective, verb, …), enabling to compute tag frequency distribution as explanatory feature.



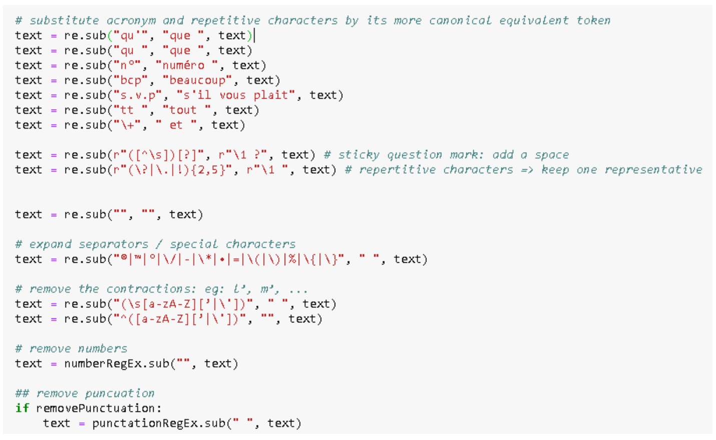
Dependency parsing is a more sophisticated process than POS tagging to discover the grammatical dependencies between words within a sentence. It produces an annotated dependency tree revealing the nature of the interaction between the words.



I finally didn’t employ none of these advanced tagging methods because French language is not well supported by most of NLP packages (Spacy, Standorf NLP or NLTK). The only basic NER I put in place is to locate the drug name or active ingredient entities based on the list of words extracted from the RCP repository: such entities are central and their identification among the sentence will be used later on to create additional features.

Text Cleansing and Normalization

I implemented some ad-hoc cleansing/normalization rules based on regular expression to tackle special characters, repetitive number, punctuation characters or usual acronyms.



```
# substitute acronym and repetitive characters by its more canonical equivalent token
text = re.sub("qu'", "que ", text)
text = re.sub("qu ", "que ", text)
text = re.sub("n°", "numéro ", text)
text = re.sub("bcp", "beaucoup", text)
text = re.sub("s.v.p", "s'il vous plait", text)
text = re.sub("tt ", "tout ", text)
text = re.sub("\+", " et ", text)

text = re.sub(r"([^\s])[?]", r"\1 ?", text) # sticky question mark: add a space
text = re.sub(r"(\?|\.|!){2,5}", r"\1 ", text) # repertitive characters => keep one representative


text = re.sub("", "", text)

# expand separators / special characters
text = re.sub("®|™|°|\/|-|\*|•|=|\(|\)|%|\{|\}", " ", text)

# remove the contractions: eg: l’, m’, ...
text = re.sub("(\s[a-zA-Z][’|\'])", " ", text)
text = re.sub("^([a-zA-Z][’|\'])", "", text)

# remove numbers
text = numberRegEx.sub("", text)

## remove puncuation
if removePunctuation:
    text = punctationRegEx.sub(" ", text)
```

In a second time, for regular words, stopWords eliminates semantically irrelevant and frequent tokens whereas stemming/lemmatization reduces the morphological variants into their etymological root.

For this job, I utilized the NLTK package: this process simplifies gracefully the phrase structure but at the expense of its lexical and grammar correctness.

The stemming/lemmatization preprocessing is counterproductive to word embedding model learnt from corpora which haven't been stemmed or lemmatized upfront: they are so mutually incompatible and for DL scenario, I made use of word embedding excluding de facto this root normalization.

Classical Technique

Abstract

HMM (Hidden Markov Model) is a probabilistic and transitional graph modeling which is appropriate to model sequence of words. For example, it's typically capable to learn on text corpus and predict POS tags but some research studies indicated that HMM is also applicable to text categorization with good performance: unfortunately, robust HMM python implementation is missing.

The arguable fallback is to switch to non-parametric statistical inference method like SVM, decision true and so on, with the crucial loss of sequence awareness. To compensate slightly such discarding, the feature extraction/enrichment should include some handmade tricks trying to grasp some contextual information from the word sequence.

Feature Enrichment

This step adds a-priori extra features which may discriminate the label much more than the original features: they can be calculated from the text or can originate from external sources**.**

I incorporated above basic statistics giving insights on the text structure and composition:

- count of sentences
- count of words
- distinct count of drug name entities
- distinct count of active ingredient entities
- count of question marks (typically to identify specifically multi-intent label)
- individual count of interrogative pronoun entities (one column per pronoun: quand, qui, quoi, ou, comment, pourquoi, combien, quel(s|le,..)
- distinct count of time entities (eg: jours, après midi, soir, année, 12h, mardi, samedi, temps....)
- distinct count of quantity entities (eg: 5mg, 10ml, ...)
- count of association entities (eg: et, avec, ou, ...)
- distance between interrogative pronoun and drug name entities
- distance between active ingredient and drug name entities
- distance between quantity and drug name entities
- distance between time and drug name entities
- distance between question marks and drug name entities

They are either **count-based or distance-based statistics**: distance variant is intended to catch the word context by measuring the relative distance between key entities. This computation needs to put in place the domain-based (list of distinct values) or custom regular expression NER (Named Entity Recognition) so that it’s possible to locate the key entities in consideration.

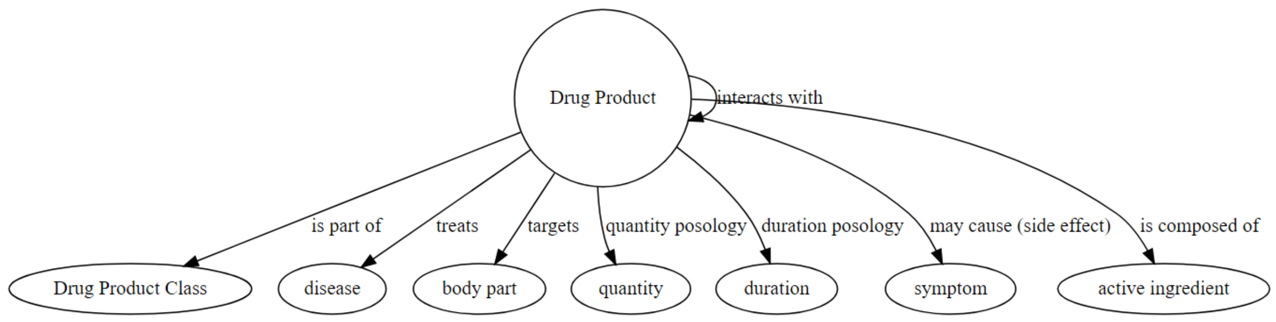
An extra calculated column is added to the train data frame per statistics as shown below:

```
XTrain['drugCount'] = XTrain['question'].map(lambda text: getEntityCount(text, drugNames, True))
XTrain['ingredientCount'] = XTrain['question'].map(lambda text: getEntityCount(text, ingredientNames, True))
```

```
XTrain['timeCount'] = XTrain['question'].map(lambda text : len(re.findall(timeRegEx, text)))
XTrain['quantitiesCount'] = XTrain['question'].map(lambda text : len(re.findall(quantityRegEx, text)))
XTrain['questionMarkCount'] = XTrain['question'].map(lambda text : len(re.findall(questionMarkRegEx, text)))
XTrain['sentenceCount'] = XTrain['question'].map(lambda text : 1 + len(re.findall(sentenceSeparatorRegEx, text)))
XTrain['wordCount'] = XTrain['question'].map(lambda text : getWordCount(text))
```

If the text sample has well identified drug product entities, it’s valuable to extend the primary feature vector with relevant information related to these drug products.

I represent herein a specialized **knowledge sub graph** centered on the drug product entity with some interesting relationships to other entities (quantity, human body part , …).



Indeed, such related entities characterize well the drug product and they can improve the detection of the commonality between texts sharing same label: for instance, a drug product class (eg: antidepressant family) may raise particular questions.

Unfortunately, this knowledge graph model is not available publicly and should be built by our own: the ANSM provides online the full description of the drug usage indication in HTML format. Such resource can feed a learning system to extract above salient related entities.

I didn’t implement this information extraction from ANSM source because it’s a colossal workload which is incompatible with the project scope.

Feature Representation

The document (composed of sentences) should be converted into numerical vector because most of ML classifiers can only cope with numeric values and they don’t care about symbol and semantic conveyed by the word.

First basic solution is the **BOW** (Bag Of Words) representation where each word of the vocabulary is defined in column and the text in row: the cell value stores the word frequency.

I didn’t consider **n-gram** document representation because as specified earlier, the classical technique scenario doesn’t employ sequence modeling like HMM which is able to treat n-gram structure.

The problem of the **BOW** representation is that rare term which in general discriminates well the document are under estimated in regards with commonly used but irrelevant terms (eg: generic verb, …).

TF-IDF (**T**erm **F**requency **I**nverted **D**ocument **F**requency) overcomes this pitfall by overweighting terms which are identified as rare for a given corpus.

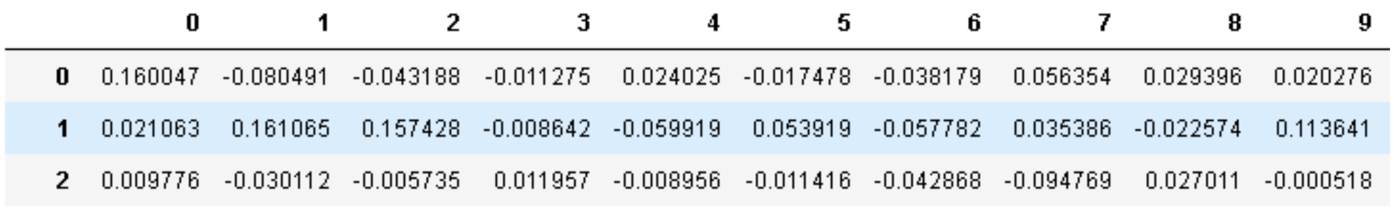
The shortcoming is that such vectorization generates a very high dimensional space depending on the vocabulary size. We fall into the well-known **curse of dimensionality** where data distribution is extremely sparse making classification task inefficient when training size is too short.

The space dimension should be reduced consequently:

stop words and stemming processes already reduce upfront the vocabulary size

I applied the **PCA** (Principal Component Analysis) linear dimension reduction which keeps the top eigen vectors capturing the maximum of the data distribution variance: PCA is a process which is totally semantic unaware in contrary to word embedding I will tackle later on

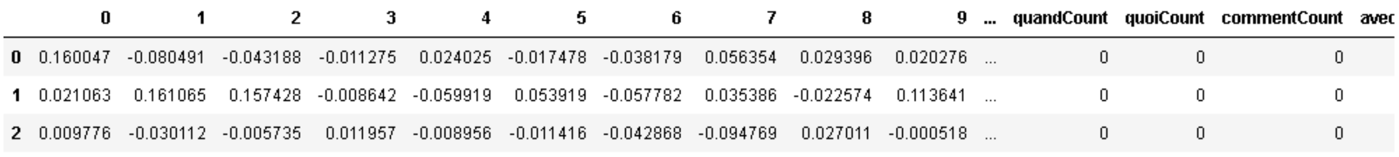
TF-IDF application and PCA reduction produce a low dimensional numerical vector per document as below:



	0	1	2	3	4	5	6	7	8	9
0	0.160047	-0.080491	-0.043188	-0.011275	0.024025	-0.017478	-0.038179	0.056354	0.029396	0.020276
1	0.021063	0.161065	0.157428	-0.008642	-0.059919	0.053919	-0.057782	0.035386	-0.022574	0.113641
2	0.009776	-0.030112	-0.005735	0.011957	-0.008956	-0.011416	-0.042868	-0.094769	0.027011	-0.000518

Classification Modeling

The classifier takes as input a feature space combining the reduced BOW representation and the handcrafted statistics:



	0	1	2	3	4	5	6	7	8	9	...	quandCount	quoiCount	commentCount	avec
0	0.160047	-0.080491	-0.043188	-0.011275	0.024025	-0.017478	-0.038179	0.056354	0.029396	0.020276	...	0	0	0	
1	0.021063	0.161065	0.157428	-0.008642	-0.059919	0.053919	-0.057782	0.035386	-0.022574	0.113641	...	0	0	0	
2	0.009776	-0.030112	-0.005735	0.011957	-0.008956	-0.011416	-0.042868	-0.094769	0.027011	-0.000518	...	0	0	0	

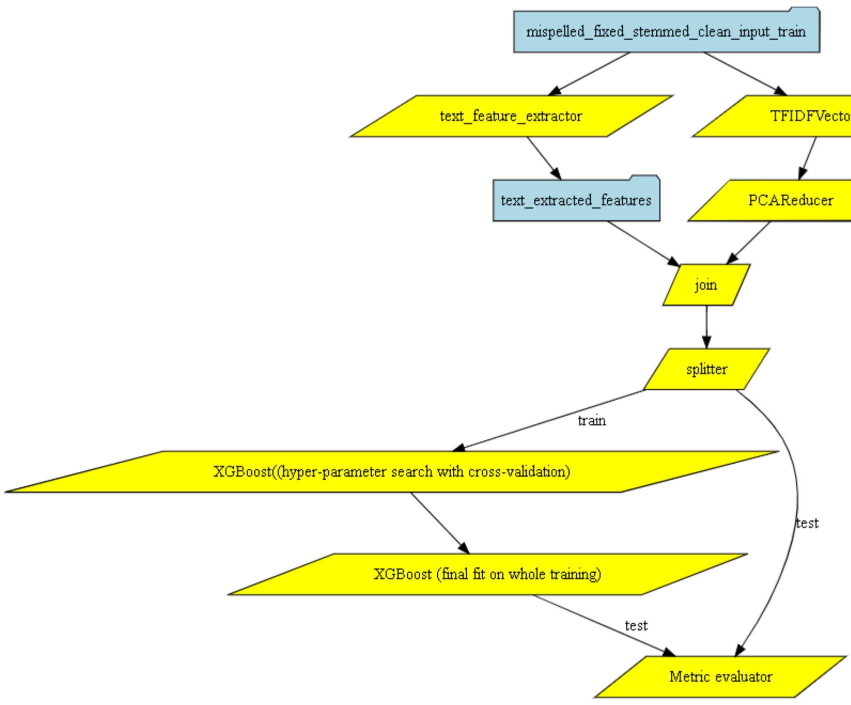
I bet on the **XGBoost** classifier delivering excellent accuracy in a reasonable time (it's multi-thread friendly): XGBoost is based on boosting ensemble technique combining sequentially weak classifiers (in general decision tree) where at each iteration, the weighting on incorrected classified observations is increased to enforce the next classifier to focus its attention on feature sub space with high error.

XGBoost comes up with many hyper-parameters to tune: an inappropriate selection usually leads to suboptimal model.

I followed the standard methodology and best practices:

- find out the optimal hyper-parameters by testing different selective combinations. I retained the one delivering the best accuracy on unseen dataset (validation) with **cross validation** enable as training is very small
- fit the final model with the above fixed hyper-parameters on the whole training and assess the generalization error on test

Here's the learning pipeline for the classical technique track:


mispelled_fixed_stemmed_clean_input_train
text_feature_extractor
TFIDFVectorizer
text_extracted_features
PCAReducer
join
splitter
train
XGBoost((hyper-parameter search with cross-validation)
test
XGBoost (final fit on whole training)
test
Metric evaluator

I focused my attention on the following parameters which are the most instrumental to the final accuracy:

max_depth

this parameter drives the decision tree complexity to partition the feature space

a low value usually prevents from overfitting and favor the weak learner synergy

I tested empirically 3 values: 4 , 6, 8

min_child_weight

under the threshold, the learner stops splitting and generates a leaf node

it controls as well the tree complexity and consequently the overfitting

I tested empirically 3 values: 2, 5, 10

n_estimators

this parameter sets the maximum number of stacked trees

I fixed it empirically to 100

learning_rate (eta)

it controls an important parameter of the gradient descent optimizer

I tested empirically 2 values: 0.05, 0.1

cross validation fold

cross validation ensures a more reliable generalization error indicator which is not biased by a particular split (test set). It's valuable typically in imbalanced label or small dataset situation (it's the data challenge case)

I fixed it empirically to 4

Other parameters settings rely on the XGBoost defaulting to avoid excessive processing time caused by the grid search combinatory explosion: by crossing max_depth, min_child_weight, learner rate and cross validation fold, it represents 72 (3x3x4x2) learning units to reveal the optimal parameter values.

For the final model fit, I set up the early stopping parameter to 10 to avoid useless extra tree stacking.

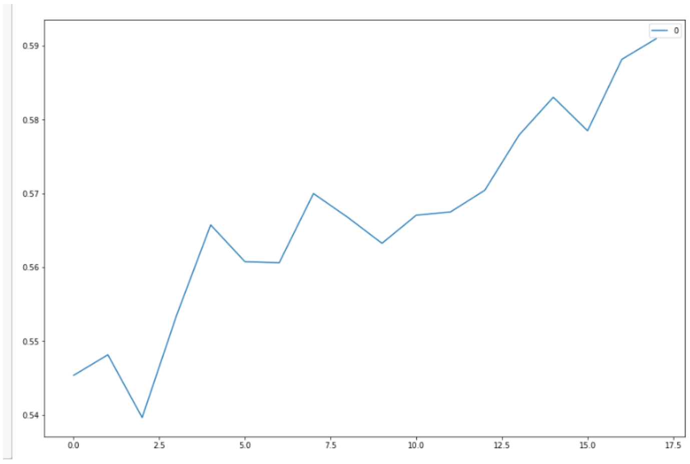
Result Analysis

The accuracy on test is low with a 59% F1-score

	precision	recall	f1-score	support
avg / total	0.63	0.62	0.59	1205

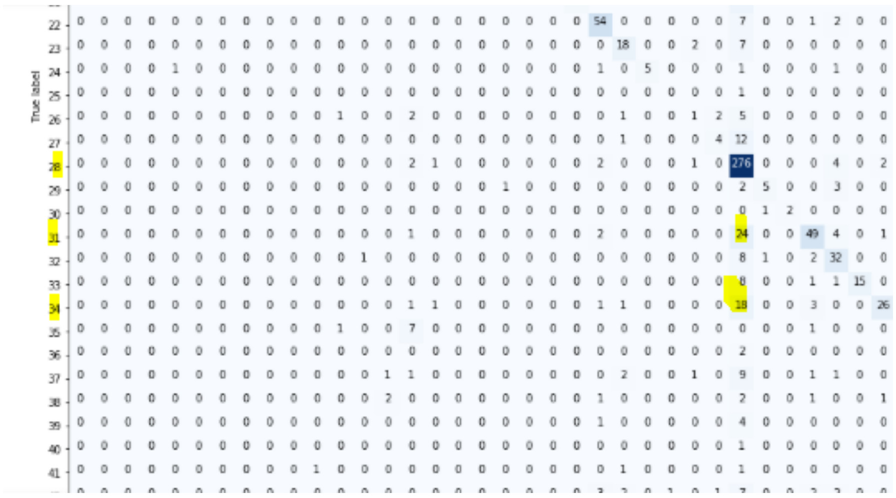
This figure displays the score for each hyperparameter selection and confirms that tuning is a key element of accuracy (score ranges from 0.54 to 0.59): the best score is obtained wih learning_rate=0.1, max_depth=8 and min_child_weight=10.

The grid search with CV (fitting 4 folds for each of 18 candidates, totalling 72 fits) took 2 hours to find the optimal parameters.



When analyzing the 51x51 confusion matrix, the highest confusion occurs on intent 31 (drug>disease **indication**) which is incorrectly predicted to intent 28 (drug>disease **contraindication**).

The high confusion error between 31 and 28 confirms the weakness of BOW based approach I used here: 2 entities occur simultaneously (drug product and disease) but the context linking them is not understood (indication vs contraindication).



22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54	0	0	0	0	0	7	0	0	1	2	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	2	0	7	0	0	0	0	0	0
24	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	5	0	0	0	1	0	0	0	1	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	1	0	0	1	2	5	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	4	12	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	2	0	0	0	1	0	276	0	0	0	4	0	2
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	5	0	0	3	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	0	0	0	0	0	24	0	0	49	4	0	1
32	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	1	0	2	32	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	1	1	15	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	18	0	0	3	0	0	26
35	0	0	0	0	0	0	0	0	0	0	0	1	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	2	0	0	1	0	9	0	0	1	1	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	1	0	0	1
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	4	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0

Deep Learning Technique

Abstract

DL is commonly recognized as an universal estimator capable of fulfilling any sort of learning requirements from feature representation to the predictive modeling within a single neural network. The key strength of this all-in-one learning is that the loss optimization to find out the best modeling parameters (weights, …) operates consistently across all functional layers regardless of their respective purpose (embedding, decision making, …). In contrast, with traditional method, feature representation and classification are 2 sub tasks which are engineered/optimized separately.

I specifically looked at its sequence modeling capacity carried by 2 architecture types:

- **RNN** (Recurrent Neural Network with **LSTM** (Long Short Term Memory) unit
- **CNN** (Convolutional Neural Network)

The hybrid option mixing up CNN and RNN is not considered here for simplicity sake even if some practitioners recommends this winning combination to get cutting edge performance.

Furthermore, DL also provides a very good support of word embedding which can be combined nicely with above architectures as upstream layer.

Feature Enrichment

I intentionally excluded extra features to verify how a DL sequence modeling can give some good results without manual contributions (statistics on text, …).

Feature Representation

Sequential representation

As the predictive modeling layer is sequence aware, the text representation should be **n-gram** where n is the number of words to keep: if the number of words is insufficient, it's necessary to apply a padding to get at the end a fixed sequence length for all documents.

As observed in the "Data Exploration" section, lengthy document usually starts with the description of the question context and ends up with concrete question (eg: "Je suis suivi par un médecin … Qu'est ce que c'est recommandé?"). It would make sense as the document can be truncated due to the fixed sequence length constraint to keep the n-th last words and not the n-th first words to not lose the question part.

In short, each document is shaped as a fixed_sequence_length x vocabulary_size matrix: again, vocabulary size can be huge leading to inappropriate high dimensional feature space and a dimension reduction is mandatory.

Word embedding

Text corpus

Instead of applying a generic PCA, a better alternative is the popular **word embedding**: it's an unsupervised method which learns from a very large text corpora to optimize a lower dimensional vector representation where words sharing similar context (within a sentence) are close to each other. The wonder of this dimension reduction is that vector proximity is governed by semantic similarity.

Word embedding is implemented in a DL flavor (Word2Vec or FastText) and in a non-DL way too (GloVe project) with nearly similar performance.

The question now is to determine the **text corpus** used to build this embedding model:

consume directly model tediously pre-trained by the GAFA companies

such model is based on very large general purpose vocabulary but probably miss domain specific vocabulary (our study case in fact)

build a custom embedding from the POSOS corpus

it overcomes the domain specific vocabulary lack (drug product name, …) but it's not complete and robust enough considering the small training dataset with many misspelling/incorrectness in the text.

Perform a model transfer from GAFA base with specific vocabulary coming from POSOS corpus

In practice, this ideal solution is undoable because it's required corporate level hardware to rebuild a merge embedding model combining general and specific vocabulary

I finally experimented the custom and general embedding models (transferred model is out of scope). For the general embedding option, I opted for the 300-dimensional **FastText** model which is gracefully available in French language.

In conclusion, the training dataset is represented as a n x k x v numerical matrix where:

- n is the number of observation (document)
- k embedded space dimension
- v fixed sequence length

I decided empirically to set fixed sequence length to the word count means observed in POSOS train dataset.

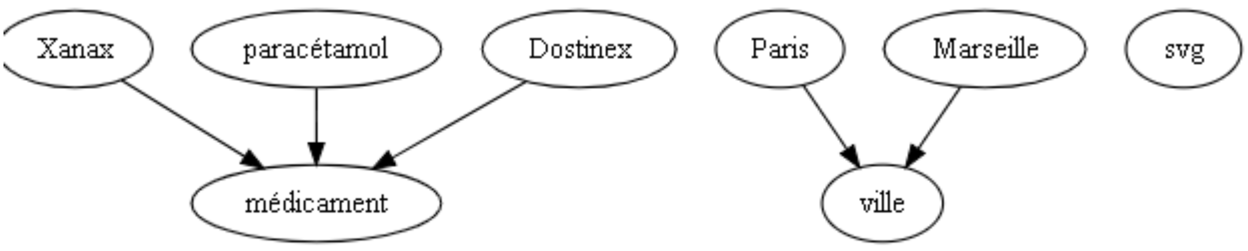
OOV handling stratregy

Embedding layer can only deal with word which exists in its vocabulary: if the learnt corpus and the corpus to vectorize are dissimilar, OOV is potentially frequent. The common practice is to encode unknown words into a random embedded vector with the risk to generate noisy feature representation.

A more elegant alternative is to merely project such unknown words into its **hypernym** entity (having a type-of relationship with the concerned word) guaranteeing a semantic proximity in the embedded space for entities of the same class/hypernym:

- all drug product entities (eg: Xanax, Abboticine) is replaced by 'médicament'
- all active ingredients (eg: Acabavir) is replaced by 'médicament'

To not completely lose the subtle distinction between entities sharing the same class, I added a very small stochastic variation vector based on the entity name so that all 'Xanax' entities have exactly the same vector and are also close to 'Paracétemol' entities.



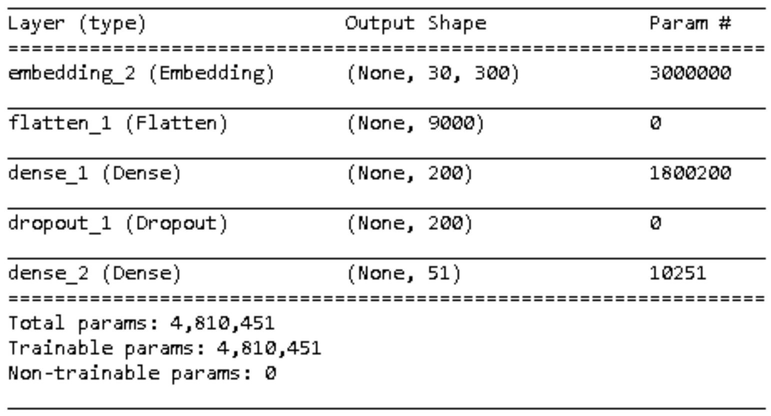
The custom extension of FastText model is managed by the fasttext_embedding_extension_builder.ipynb script.

DNN Architecture

I setup a test with DNN (Dense Neural Network) which is not a sequence modeler, as a comparison baseline for the more sophisticated architectures like RNN and CNN. It would give a gut feeling on the performance gain with sequence awareness in the modeling procedure.

Moreover, the embedding layer is built from the POSOS corpus to define again a comparison baseline to measure the gain (or loss) when opting for general purpose corpus.

The concrete architecture is described by the summary output generated by Keras:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 30, 300)           3000000
_________________________________________________________________
flatten_1 (Flatten)          (None, 9000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               1800200
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_2 (Dense)              (None, 51)                10251
=================================================================
Total params: 4,810,451
Trainable params: 4,810,451
Non-trainable params: 0
_________________________________________________________________
```

There are 2 dense layers with different activation functions: relu at the first layer and softmax at the decision layer. The dropout layer is placed between to introduce some random perturbation to combat overfitting. I set the embedding dimension to 300 in accordance with the pretrained FastText model.

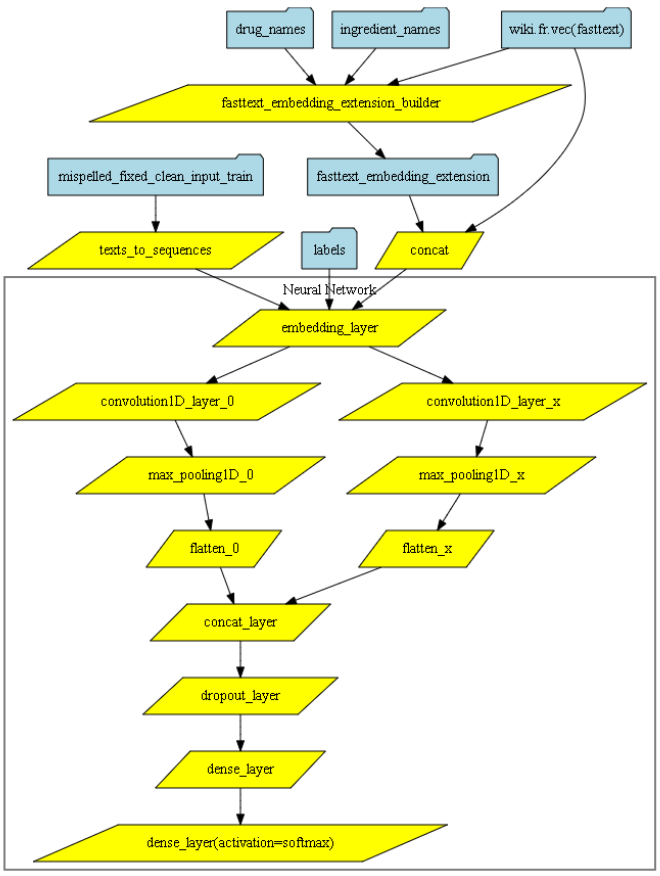
CNN Architecture

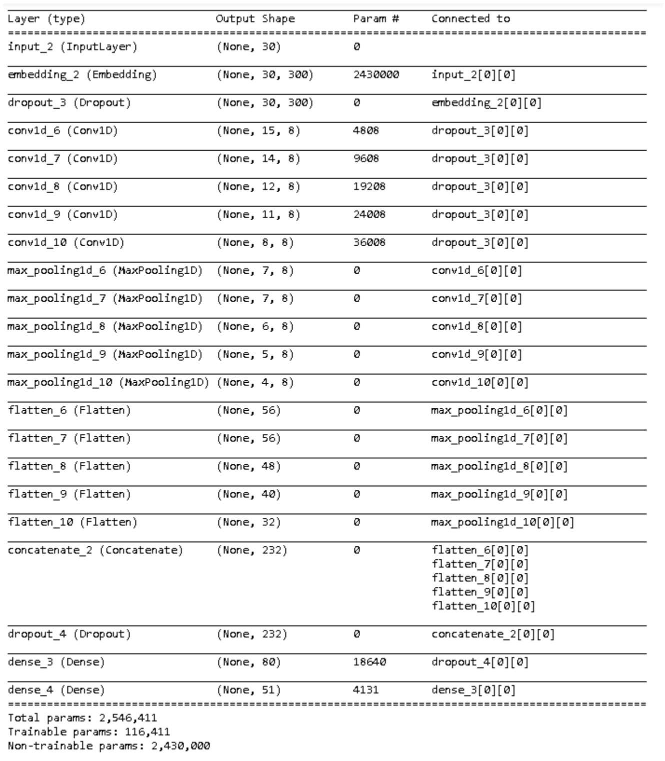
In **C**omputer **V**ision (**CV**), convolution operation is well known to be remarkable in extracting the high level representation of an image by applying a sliding window filter and computing consequently an average value for each filter position as output. These convoluted values are then activated with usual non-linear function and down-sampled thanks to the pooling layer. This pixel-wise processing is inspired by how the visual cortex analyzes the signal sent by the eye receptors.

Surprisingly, such biological inspiration also works well to catch the structural sense of word sequence in NLP. The convolution operates in a 1-dimensional array (word sequence) instead of 2D (pixel matrix) in CV. The sequential filter enforces the neural network to focus its attention on local context which establishes connection between words.

Even if some research studies demonstrate that convolution is expressive enough to cover embedding contribution, I setup my CNN architecture with embedding layer upfront. As usual, some dropout layers are intermittently inserted intot the neural network.

The best practice recommends building many convolutional layers with different filter sizes and/or strides whose outputs are then concatenated to each other and this is the resulting predictive pipeline:

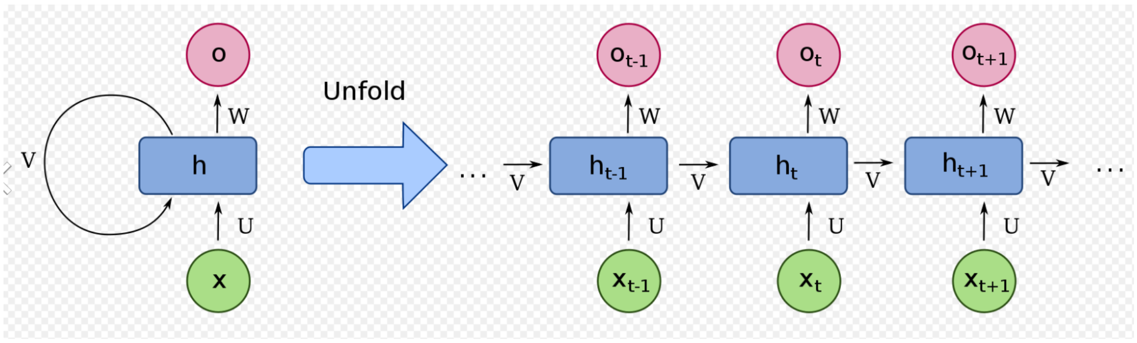



```
_______________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
===============================================================================================
input_2 (InputLayer)            (None, 30)           0
_______________________________________________________________________________________________
embedding_2 (Embedding)         (None, 30, 300)      2430000     input_2[0][0]
_______________________________________________________________________________________________
dropout_3 (Dropout)             (None, 30, 300)      0           embedding_2[0][0]
_______________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, 15, 8)        4808        dropout_3[0][0]
_______________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, 14, 8)        9608        dropout_3[0][0]
_______________________________________________________________________________________________
conv1d_8 (Conv1D)               (None, 12, 8)        19208       dropout_3[0][0]
_______________________________________________________________________________________________
conv1d_9 (Conv1D)               (None, 11, 8)        24008       dropout_3[0][0]
_______________________________________________________________________________________________
conv1d_10 (Conv1D)              (None, 8, 8)         36008       dropout_3[0][0]
_______________________________________________________________________________________________
max_pooling1d_6 (MaxPooling1D)  (None, 7, 8)         0           conv1d_6[0][0]
_______________________________________________________________________________________________
max_pooling1d_7 (MaxPooling1D)  (None, 7, 8)         0           conv1d_7[0][0]
_______________________________________________________________________________________________
max_pooling1d_8 (MaxPooling1D)  (None, 6, 8)         0           conv1d_8[0][0]
_______________________________________________________________________________________________
max_pooling1d_9 (MaxPooling1D)  (None, 5, 8)         0           conv1d_9[0][0]
_______________________________________________________________________________________________
max_pooling1d_10 (MaxPooling1D) (None, 4, 8)         0           conv1d_10[0][0]
_______________________________________________________________________________________________
flatten_6 (Flatten)             (None, 56)           0           max_pooling1d_6[0][0]
_______________________________________________________________________________________________
flatten_7 (Flatten)             (None, 56)           0           max_pooling1d_7[0][0]
_______________________________________________________________________________________________
flatten_8 (Flatten)             (None, 48)           0           max_pooling1d_8[0][0]
_______________________________________________________________________________________________
flatten_9 (Flatten)             (None, 40)           0           max_pooling1d_9[0][0]
_______________________________________________________________________________________________
flatten_10 (Flatten)            (None, 32)           0           max_pooling1d_10[0][0]
_______________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 232)          0           flatten_6[0][0]
                                                                 flatten_7[0][0]
                                                                 flatten_8[0][0]
                                                                 flatten_9[0][0]
                                                                 flatten_10[0][0]
_______________________________________________________________________________________________
dropout_4 (Dropout)             (None, 232)          0           concatenate_2[0][0]
_______________________________________________________________________________________________
dense_3 (Dense)                 (None, 80)           18640       dropout_4[0][0]
_______________________________________________________________________________________________
dense_4 (Dense)                 (None, 51)           4131        dense_3[0][0]
===============================================================================================
Total params: 2,546,411
Trainable params: 116,411
Non-trainable params: 2,430,000
```

RNN Architecture

Recurrent Neural Network architecture tries to leverage this sequential information from the sentence with a special layout where each layer at the i-th position is fed with the i-th element of the input sequence and the output of the direct preceding layer (i-1 th position): each layer captures somehow the hidden state (memory) of the preceding sub sequence of inputs (words).

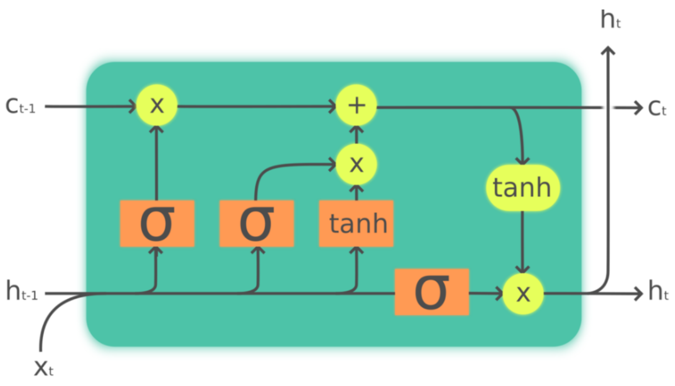
RNN can be **bi-directional** (instead of forward only) where the i-th layer also depends on the computational output of the direct successor (i+1 th position).



(source: Wikipedia)

In practice, the simple layer computational unit ("h" blue box in above diagram) exhibits inability to catch long term dependencies to distant input x(t-n) where n is significant high, due to the vanishing gradient problem.

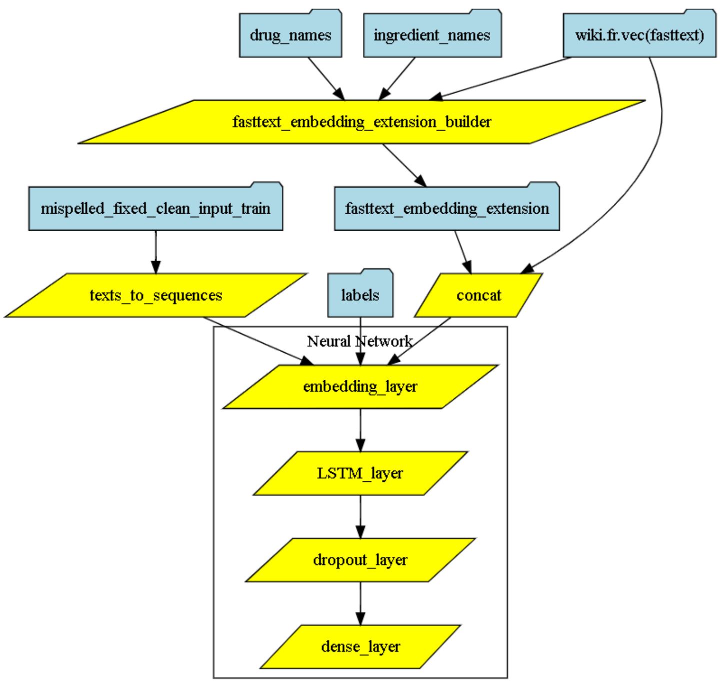
The **LSTM** (**L**ong **S**hort **T**erm **M**emory) cell unit has been invented to treat this long-tailed sequential dependency we can find in multi-sentence text analysis where the context may be specified upfront far away from the concerned word.

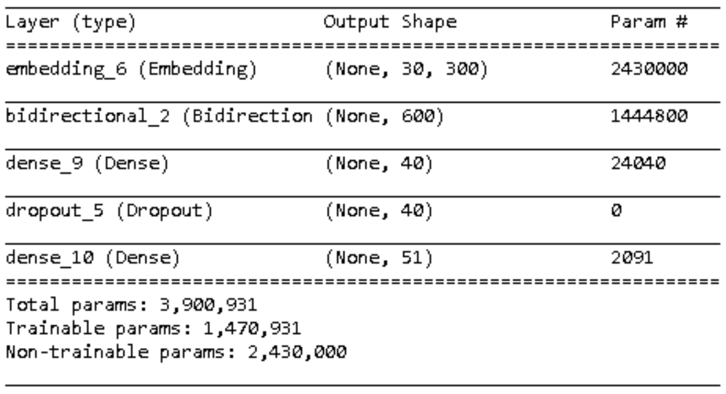


(source: Wikipedia)

This processing unit adds a secondary flow (upper stream) to update gradually the memory (cell state denoted as C(t)) with contributions controlled by several input gates (shown at the bottom).

The DL architecture with the word embedding extension is represented below:

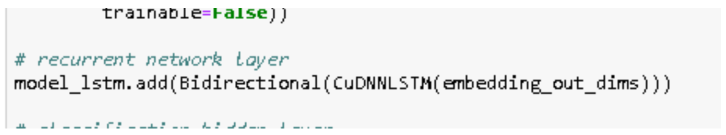




```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 30, 300)           2430000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 600)               1444800
_________________________________________________________________
dense_9 (Dense)              (None, 40)                24040
_________________________________________________________________
dropout_5 (Dropout)          (None, 40)                0
_________________________________________________________________
dense_10 (Dense)             (None, 51)                2091
=================================================================
Total params: 3,900,931
Trainable params: 1,470,931
Non-trainable params: 2,430,000
_________________________________________________________________
```

CuDNNLSTM is a very convenient Tensorflow/Keras class automating the construction of the recursive processing unit pattern: furthermore, it's based on the CuDNN Nvidia library boosting learning time by a factor of 10.

The bidirectional mode has been enabled just by wrapping the CuDNNLTSM construct with Bidirectional.

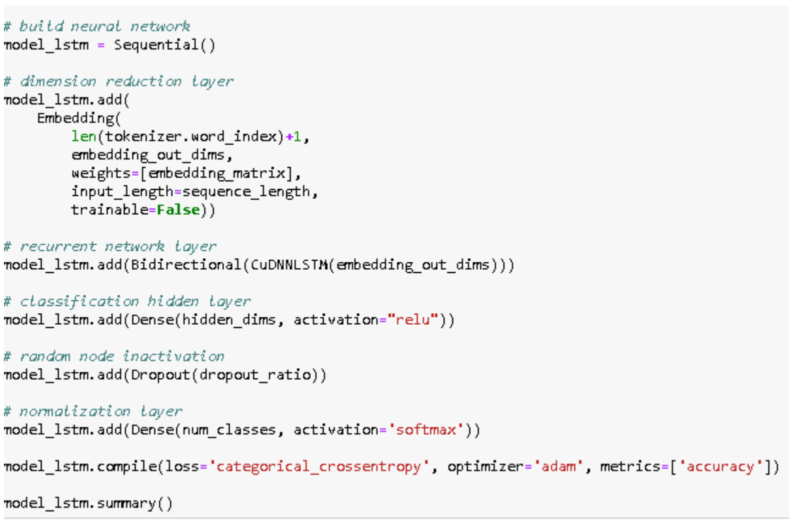


```
        trainable=False))

# recurrent network layer
model_lstm.add(Bidirectional(CuDNNLSTM(embedding_out_dims)))
```

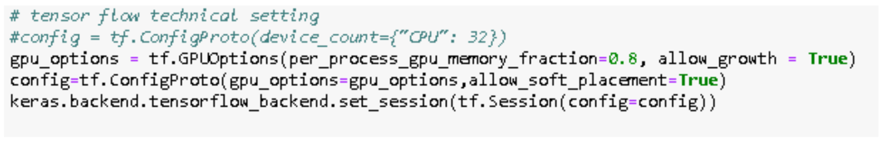
DL Implementation and Execution

I have implemented all DL networks with the high level **Keras** model which runs on top of **Tensorflo**. Keras provides a very friendly API that simplifies dramatically the neural network construction by hiding all the technical boiler plates (TensorFlow session handling, many default parameters are set, …).

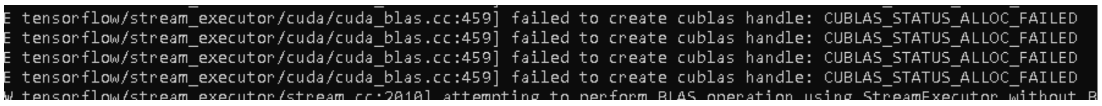
Here's an example of Keras code where the layers are built in a very concise manner thanks to wrapper objects like Convolution2D, Activaction and so forth.


```
# build neural network
model_lstm = Sequential()

# dimension reduction layer
model_lstm.add(
    Embedding(
        len(tokenizer.word_index)+1,
        embedding_out_dims,
        weights=[embedding_matrix],
        input_length=sequence_length,
        trainable=False))

# recurrent network layer
model_lstm.add(Bidirectional(CuDNNLSTM(embedding_out_dims)))

# classification hidden layer
model_lstm.add(Dense(hidden_dims, activation="relu"))

# random node inactivation
model_lstm.add(Dropout(dropout_ratio))

# normalization layer
model_lstm.add(Dense(num_classes, activation='softmax'))

model_lstm.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model_lstm.summary()
```

Nevertheless, it's still possible to access to the specific APIs of the underlying concrete framework to typically configure the execution parameters as shown below.

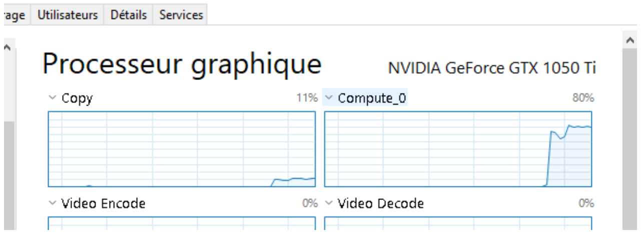

```
# tensor flow technical setting
#config = tf.ConfigProto(device_count={"CPU": 32})
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.8, allow_growth = True)
config=tf.ConfigProto(gpu_options=gpu_options,allow_soft_placement=True)
keras.backend.tensorflow_backend.set_session(tf.Session(config=config))
```

With CUDA/GPU activation, Tensorflow runs really faster than using the regular CPU (even with a powerful 32-core machine) but it asks for a particular attention on the DL configuration: when inappropriately parameterized, it causes in worst case crashes (allow_growth parameter should be defined) or memory allocated error.


```
E tensorflow/stream_executor/cuda/cuda_blas.cc:459] failed to create cublas handle: CUBLAS_STATUS_ALLOC_FAILED
E tensorflow/stream_executor/cuda/cuda_blas.cc:459] failed to create cublas handle: CUBLAS_STATUS_ALLOC_FAILED
E tensorflow/stream_executor/cuda/cuda_blas.cc:459] failed to create cublas handle: CUBLAS_STATUS_ALLOC_FAILED
E tensorflow/stream_executor/cuda/cuda_blas.cc:459] failed to create cublas handle: CUBLAS_STATUS_ALLOC_FAILED
W tensorflow/stream_executor/stream.cc:2010] attempting to perform BLAS operation using StreamExecutor without B
```

GPU memory consumption is sensitive to the training size and the batch size parameter: higher value means higher needed GPU memory but batch size selection impacts the optimizer behavior and consequently the resulting model.

My GTX1050Ti graphic card with 4Gb memory is a bit short to handle mid-complex DL training: setting the memory threshold to 80% implies more data movement (Copy operation) between the motherboard and GPU memories.



Result Analysis

Here's the classification score per architecture:

	Precision	Recall	F1-Score
DNN / Custom embedding	0.60	0.59	0.58
CNN / Fasttext embedding	0.52	0.53	0.51
LSTM / Fasttext embedding	0.66	0.65	0.64

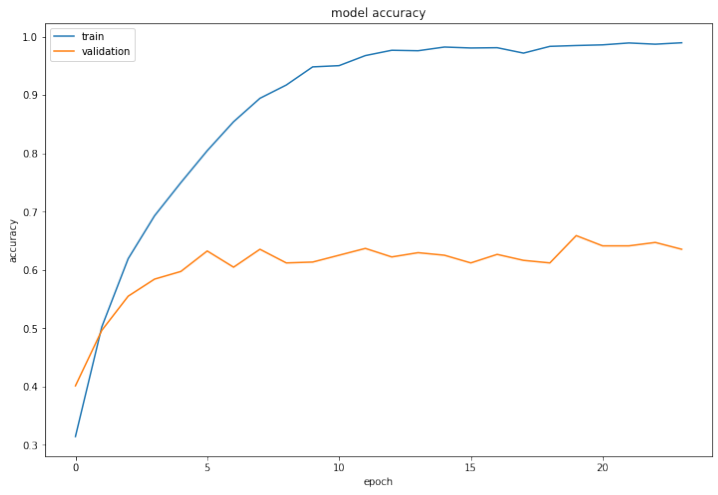
Such results have been obtained with very few different hyperparameters I tested manually (in opposition to the systemic parameter grid search for XGBoost estimator).

LSTM outperforms slightly CNN and DNN architectures: CNN exhibits unexpectedly very disappointing score, some deeper investigations are needed to identify the root cause.

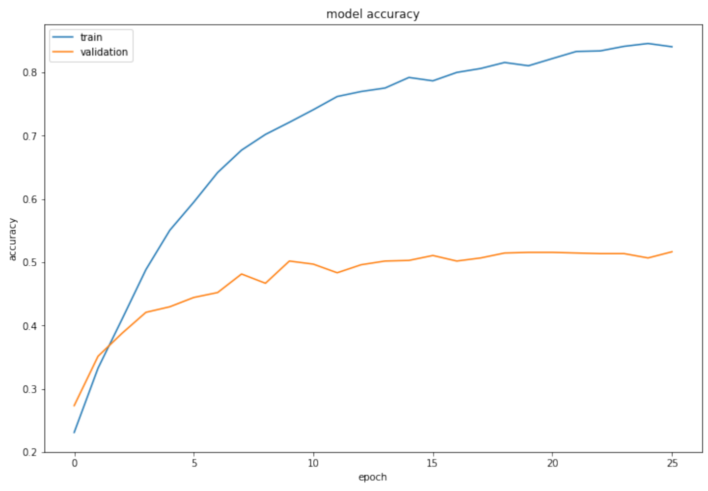
LSTM learning curve indicates that the model is overfitting on train beyond 12 epochs but there’s no observed accuracy improvement on validation set (the accuracy chart is stationarized around 63% beyond 6 epochs).

This means that this LSTM (in general a neural network) is capable to fit perfectly any data samples but its learning is badly applicable to unseen dataset (validation): this is typically an overfitting risk.

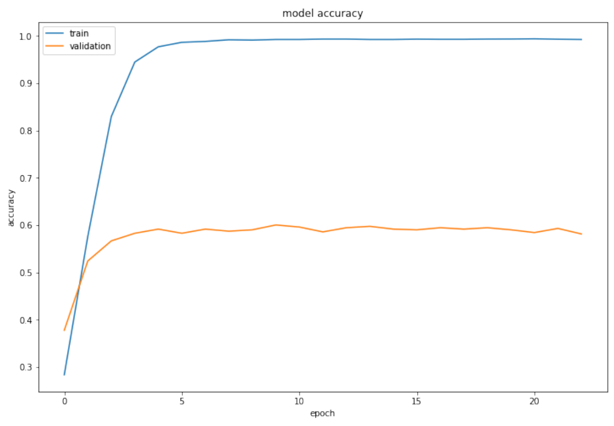
CNN and LSTM architectures need more iterations than the simple DNN to reach good fit on train.



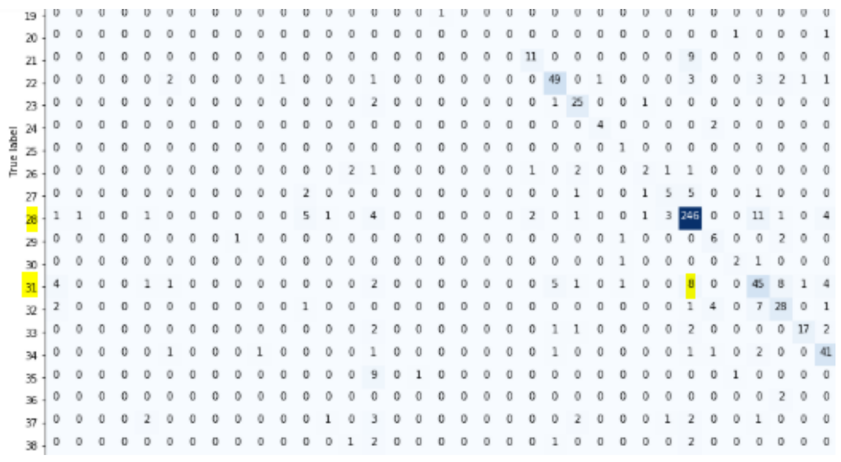
CNN learning curve



DNN learning curve

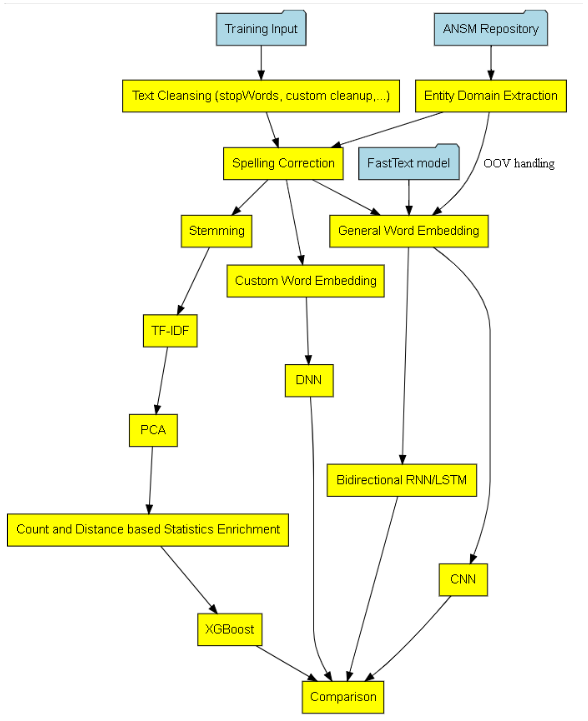


Comparatively to XGBoost, LTSM model does a better job to distinguish drug>disease indication and contraindication topics: the sequence/context awareness seems to pay off.



Comparative Study

I represent below the overall comparative experiment where 4 modeling candidates are evaluated holistically (accuracy, interpretability, ease of use, …).



Model Accuracy

LTSM architecture undoubtedly delivers the best performance over classic technique, but the hyper parameter tuning is tricky and requires practical background to find out empirically the "good" configuration.

The scores on test I measured should be considered with caution: indeed, I hold out 15% of the training dataset (8000), representing only 1200 rows for 51 labels. It would be more fair to do a cross validation error measurement on test: I suspected that the models tend to overfit with significant standard deviation on the fold errors.

Model Interpretability

DL has the good reputation to provide excellent prediction accuracy when the architecture engineering is well conducted, but the theoretical/mathematical foundation is not rock solid yet (some mathematicians are working on).

Additionally, model interpretability (feature importance, individual contribution at prediction time) is not supported natively with DL, except using exogeneous explainer like LIME (Local Interpretable Model-Agnostic Explanations) which can provide some model interpretations agnostically.

XGBoost is basically the opposite: better native interpretability support (approximation of individual contributions, feature importance, …), better math foundation but it's lacking on complex modeling type (sequence modeling for instance).

This is the common trade-off between model accuracy and model interpretability.

I would say that classical method wins the match here.

Sustainability

XGBoost/scikit-learn and Keras/Tensorflow combos are both very active open source projects: contributions to Keras/Tensforflow mainly come from Google organization whereas XGBoost/scikit-learn project is the fruit of academic field.

Nevertheless, DL technology is much more popular to tackle unstructured data (video, image, text, …) and benefits a larger support of the ML researchers in the area of NLP.

DL method wins 1 point on this sustainability aspect.

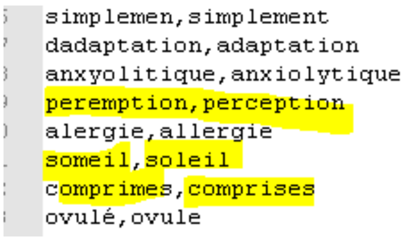
Improvement Tracks

There are for sure a lot of improvement rooms on the learning procedure I have elaborated so far. Here are some possibilities to enhance the model accuracy, I feel like to implement if I had more times and means (GPU farm, ..).

Spelling Correction

Spelling correction relies on the Levenshtein distance and the fix suggestion is not influenced by more appropriate criteria:

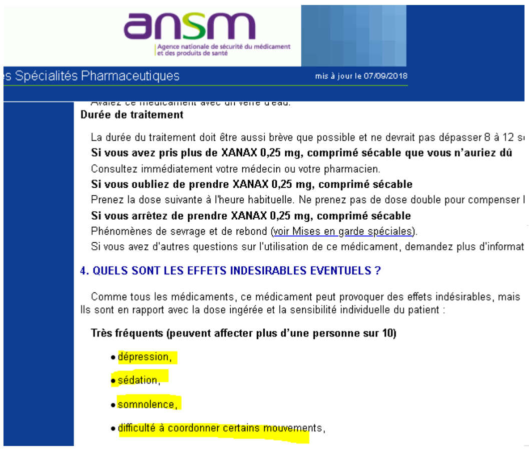
- in case of equality, favor probable words where the difference is located on the accent (eg: reveiller vs réveiller)
- take into account of the word frequency observed in the specific corpus (drug/medical) (eg: for misspelled someil, sommeil is more frequent than soleil in the drug question corpus)
- favor phonetically close words (eg: méson should be fixed into maison and not téton)



```
simplemen,simplement
dadaptation,adaptation
anxyolitique,anxiolytique
peremption,perception
alergie,allergie
someil,soleil
comprimes,comprises
ovulé,ovule
```

Named Entity Recognition

I have underexploited the ANSM repository providing in particular medication guide per drug. Information extraction can be performed quite easily by leverage the structure of the HTML page: for instance there's a dedicated/fixed section on adverse effect ("Quels sont les effets indésirables ..”?) with a formatted list of undesired consequences.



ansm
Agence nationale de sécurité du médicament et des produits de santé
s Spécialités Pharmaceutiques
mis à jour le 07/09/2018

Durée de traitement

La durée du traitement doit être aussi brève que possible et ne devrait pas dépasser 8 à 12 s

Si vous avez pris plus de XANAX 0,25 mg, comprimé sécable que vous n'auriez dû

Consultez immédiatement votre médecin ou votre pharmacien.

Si vous oubliez de prendre XANAX 0,25 mg, comprimé sécable

Prenez la dose suivante à l'heure habituelle. Ne prenez pas de dose double pour compenser l

Si vous arrêtez de prendre XANAX 0,25 mg, comprimé sécable

Phénomènes de sevrage et de rebond (voir Mises en garde spéciales).

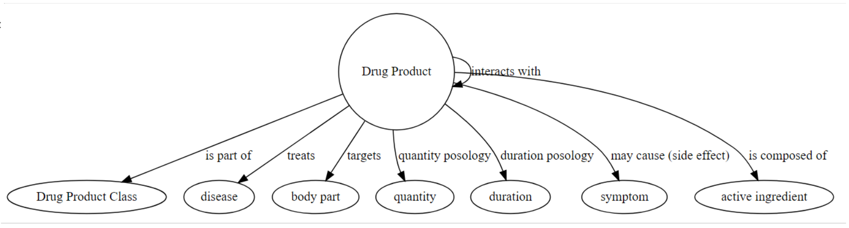
Si vous avez d'autres questions sur l'utilisation de ce médicament, demandez plus d'informat

4. QUELS SONT LES EFFETS INDESIRABLES EVENTUELS ?

Comme tous les médicaments, ce médicament peut provoquer des effets indésirables, mais Ils sont en rapport avec la dose ingérée et la sensibilité individuelle du patient :

Très fréquents (peuvent affecter plus d'une personne sur 10)

- dépression,
- sédation,
- somnolence,
- difficulté à coordonner certains mouvements,



With a more complete knowledge graph on drug, it would be possible to build a wider NER system extending the basic one (drug product and active ingredient entities) with drug product class, adverse effect, disease [contra]indication, …

Another trick to get a more satisfactory French NER system is to use the English NER combined with an English to French translator.

Count/Distance based Statistics

With above extended NER system, it's worth to compute extra statistics on the new entities. For instance, count on adverse effect entities present in the sentence may be informative to explain the target.

Word Embedding

Custom embedding has been built on a too small corpus (training) and the result is badly robust. It would be valuable to extend this corpus with:

- test corpus (input_test.csv) even if there's no label (embedding is unsupervised)
- external source (eg: doctissimo.fr web site hosts discussion forum on drurg)

Better solution is to merge above corpora with the ones used by FastText model and build our own embedding model: such learning involves a tremendous amount of resource (GPU, memory) and processing time.

OOV Handling

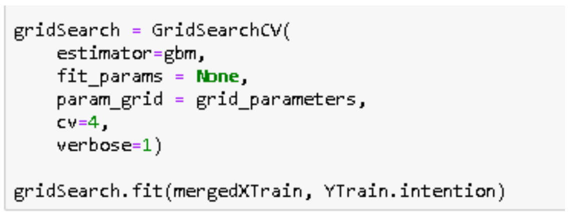
I proposed a "better-than-random" handling in case of out of vocabulary: project the drug product entities into its class/hypernym ("médicament") with a very small stochastic perturbation.

To reduce the taxonomical loss, a possible enhancement is to leverage the drug name class provided the above extended NER system: the drug product entities would be converted into their respective drug class ("antidépresseur", …)

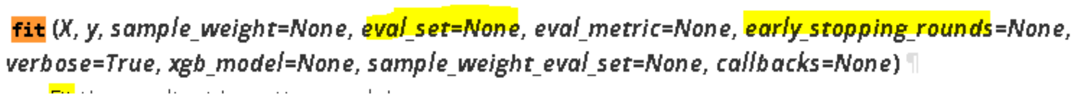
Early Stopping With Grid Search CV

Scikit-learn GridSearchCV cannot leverage the early stopping of XGBoost on the fold (validation set) GridSearch wrapper defines internally: indeed, XGBoost.fit() asks for an explicit DMatrix for eval_set which is used by the early stopping mechanism.

This inability to use the early stopping on the fold during the grid search means that in a second time, once the best hyper-parameters are found, we need to run a XGBoost.fit() with early stopping on validation set in order to determine the best early stopping value.



```
gridSearch = GridSearchCV(
    estimator=gbm,
    fit_params = None,
    param_grid = grid_parameters,
    cv=4,
    verbose=1)

gridSearch.fit(mergedXTrain, YTrain.intention)
```



```
fit (X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None,
verbose=True, xgb_model=None, sample_weight_eval_set=None, callbacks=None) ¶
```

Neural Network Tuning

For the classical method, scikit-learn framework offers convenient wrapper to tune the hyper-parameters with cross-validation. On the DL side, Keras doesn't support natively cross validation nor kind of grid search wrapper to ease the execution and scoring of different hyperparameters combination.

Having said that, it's still feasible to write ad-hoc python in order to simulate the equivalent of grid search with cross validation: the practical issue is that DL learning unit is slower than XGBoost.

Other Modeling Candidates

I purposely imposed that each candidate relies on an unique modeling principle to make the comparison more academic and distinctive. The best practice in general is to combine all the techniques in the hope to provoke a synergy effect where each estimator strength would overtake on average.

For example, associating RNN/LSTM and CNN in the same neural network would be a good candidate to be tested. For classical technique, building a general purpose French embedding model with matrix factorization (similarly to GloVe) and combining it with XGBoost are likely worthwhile.

HMM (Hidden Markov Model) track was also discarded unfairly whereas it's a sequence modeling.

Conclusion

In this present paper, I describe a walkthrough return of experience on text categorization problem in a **DL (D**eep **L**earning) and non-DL fashion (a.k.a. classical method).

First of all, the POSOS problem is very challenging because the learning materials are not sufficient (8000 questions) for a multiclass classification task and the writing style is really familiar with many misspellings. In addition, the high cardinality of target makes the problem harder: indeed, the good performance in text categorization exhibited in different research papers or blogs deals with binary classification (sentiment analysis) or reasonable cardinality in multiclass (less than 10 labels).

The classical method enrolling the famous XGBoost classifier delivers poor modeling performance (59% F1-score on test) for various specific reasons:

- no available word embedding model in French and a mere generic PCA as dimension reduction
- too limited count/distance bases statistics due to the lack of named entities (only drug and active ingredient)
    - this feature extraction is a fallback supposed to compensate the XGBoost inability to model sequence

DL experiment plays with several architectures (DNN, CNN, LSTM): unsurprisingly, as a native sequence modeling, RNN/LTSM overtakes slightly other variants with a 64% F1-score on test. I intuitively expected a bigger gap with classical method: it's probably due to the lack of hyper-parameter tuning.

On the other hand, traditional method with the combined sklearn/XGBoost offers a better interpretability (feature importance, individual contributions) and methodology support (cross validation, grid search).

Even if the comparison is totally unfair/biased, Deep Learning turns out to be the winning ML technique in NLP task solving: it's presently a very active research topic within the Data Science community as DL expressiveness/capacity are unbound similarly to the brain plasticity.

This short study with non-conclusive performance result, has at least the educational benefit to make me practice on a large spectrum of domains that a data scientist should master:

- data analysis to experimental result debriefing
- unsupervised (word embedding) vs supervised (classifier)
- training implementation and execution (GPU activation, python programming, AWS computing infrastructure, …)
- Deep learning architecture variety (RNN, CNN, DNN, …)
- general best practice on classification task (cross validation, early stopping, hyper-parameter search, …).

Annex

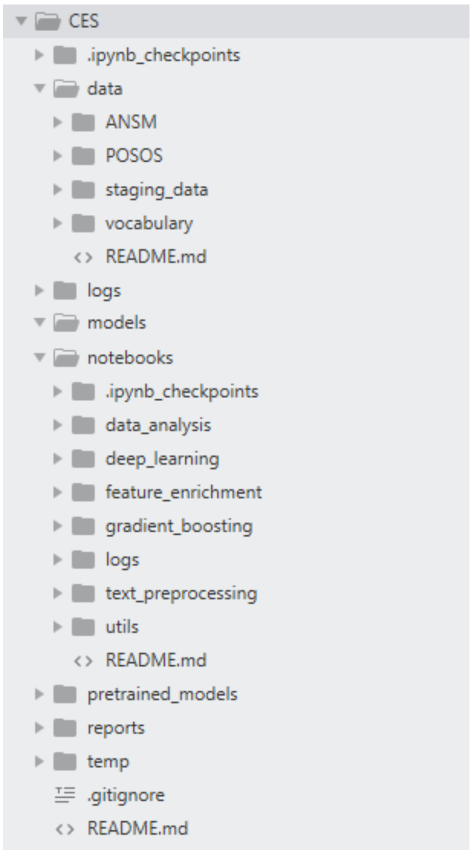
Github project

The project is available from the public github repository at the following URL:

https://github.com/jhuu32/CES

It contains Jupyter notebooks allowing to reproduce all learning experiments mentioned in this report: the only missing artifact is the FastText embedding model which is too large to be pushed into github (2Gb), but the README.md gives the necessary information to download it.

Here's the project source tree



```
CES
  .ipynb_checkpoints
  data
    ANSM
    POSOS
    staging_data
    vocabulary
    README.md
  logs
  models
  notebooks
    .ipynb_checkpoints
    data_analysis
    deep_learning
    feature_enrichment
    gradient_boosting
    logs
    text_preprocessing
    utils
    README.md
  pretrained_models
  reports
  temp
  .gitignore
  README.md
```

References

POSO challenge: https://challengedata.ens.fr/fr/challenge/33/predisez_la_reponse_attendue.html

ANSM repository: http://agence-prd.ansm.sante.fr/php/ecodex/index.php

Fasttext model: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

Spell checker: https://github.com/barrust/pyspellchecker/blob/master/docs/source/quickstart.rst

Scikit-learn: http://scikit-learn.org/stable/

XGBoost: https://xgboost.readthedocs.io/en/latest/

Keras: https://keras.io/

Tensorflow: https://www.tensorflow.org/

NLTK: https://www.nltk.org/

CNN: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

LTSM: https://www.kaggle.com/kredy10/simple-lstm-for-text-classification