Medication Question Classification

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 approaches, thanks to the huge GPU performance leapfrog and the public gorgeous labeled dataset. 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 some efficient technical responses to the NLP hard tasks.

**POSOS** French startup has submitted a NLP data challenge and it’s a good opportunity to verify humbly whether DL is an appropriate solution compared to traditional procedures, for a rookie person (like me) having very few experiences on NLP/DL area and very few CPU/GPU resources (DL has the reputation to be extremely greedy…).

***Data Challenge Description***

The data challenge is plainly described in the ENS school web site ([link](https://challengedata.ens.fr/en/challenge/33/predict_the_expected_answer.html)) and basically consists in classifying into **51** intents, drug related questions formulated in natural language (French to be precise). **POSOS** claimed to get good modeling result with 86% accuracy by utilizing DL: they don’t provide any details on the DL architecture nor any engineering clues except the possibility to extract some relevant information procured from **ANSM** (French drug administration).

The target values (question intent) are anonymized into indices from 0 to 50: hiding the intent semantic is probably aimed to avoid the use of intent-specific (and so biased) procedures. Training dataset contains only ~8000 questions: it’s really short to obtain a good learning outcome. Moreover, the text suffers from many types of “deformities” (misspelling, grammatical incorrectness, acronym usage, …) and employs specific medical vocabulary (drug name like “mirtazapine”, …): it makes the NLP challenge harder.

Here’s an example of malformed question text: “8 jrs avant la fin de ma plaquette d'evepar j'ai eu des saignement~~s~~ et des douleurs au bas ventre et au bas du dos dois je m'inquiéter»

***Project Objective***

As mentioned briefly in the Abstract section, the core objective is to compare the DL approach against traditional techniques to resolve the above tricky NLP challenge for a rookie data scientist from different perspectives:

* model accuracy
* operational aspect (tooling, hardware requirement, ...)
* modeling (how it’s difficult to setup the proper hyper-parameter / architecture / processing logic)

In fact, above aspects are somehow inter-related: for instance, model accuracy may be unexpectedly suboptimal due to operational considerations (memory shortage to complete the processing job, …).

The primary goal is not to get a fined-tuned/1st-ranked in the leaderboard predictive model with DL or other techniques, but an attempt to explore comparatively the end to end real life NLP problem for DL and co.

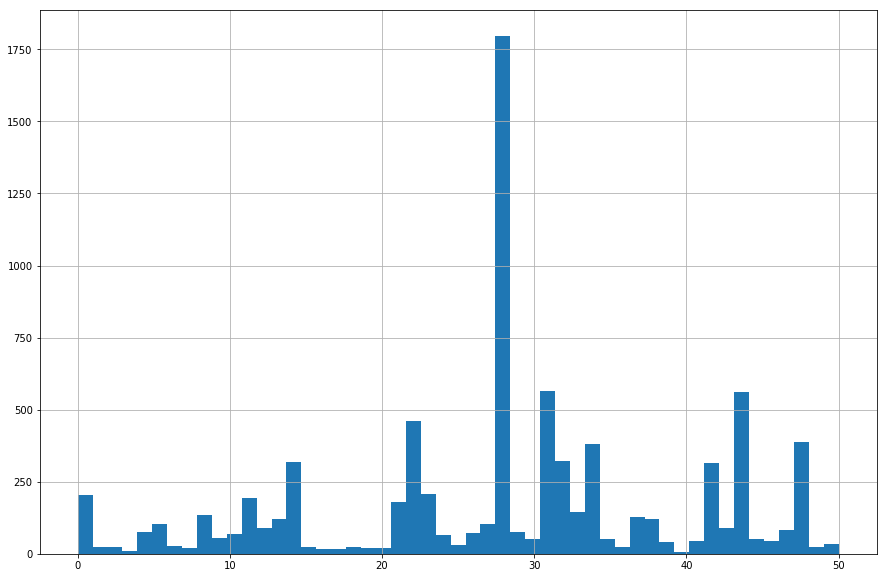
“Traditional techniques” track refers to any ML related algorithms which don’t rely on neural network principle (eg: Word2Vec is excluded): typically, XGBoost, SVM, logistic regression and PCA are possible candidate.

Similarly, DL based path should rely uniquely on neural network system but it can also benefit from “neutral” procedures (feature enrichment with external source, stemming, stopWords, …) for fairness sake.

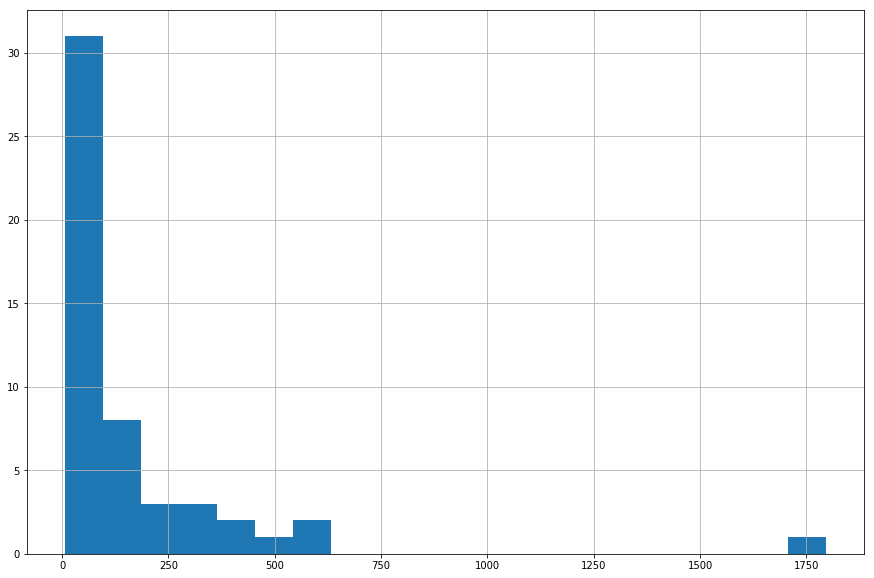
***Data Analysis***

Here’s the target distribution on training dataset revealing that it’s pretty imbalanced with a peak at intention=28.

Target distribution (number of classes = 51)



Most of labels are associated to relatively small number of samples: even half of classes have less than 100 rows.



Regardless of the ML algorithms and smart feature engineering, achieving a good enough classification rate is really tricky considering the small training size and imbalanced distribution. It’s sounds strange that the POSOS startup claims to succeed to obtain a 86% accuracy in such data shortage.

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

* **first mode**: intention=28

It looks like to be related to questions on drug side effects and somehow the contraindications



* **second mode**: intention=31

it concerns questions on symptom-drug adequacy/efficiency



* **multi-intents class**: intention=39

the commonality across text samples seems to the presence of multiple question mark tokens (counting it maybe a salient feature candidate)



The key problem of the classification task is to find a way to estimate the intent similarity between 2 questions written in a natural language.

Let’s focus on 2 questions related to the second mode category (symptom-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?”

The expression forms differ significantly: on one hand, a very concise expression putting the symptom entity and the drug entity in an adversarial fashion, on the other, the question is more detailed with one sentence to set up the context/fact and the second one to raise the concrete question of adequacy between the 2 entities mentioned previously.

This typically illustrates the stylish complexity and richness of the human language to convey an idea and more particularly an high level intent!

Moreover, notice that certain question are lexically and syntactically incorrect: words are misspelled especially when dealing with drug product names which are unfamiliar for most of non-professional persons.

Estimating the semantic similarity between words is more or less achieved successfully with word embedding approach like Word2Vec (shallow DL) and GLOVE (co-occurrence matrix factorization).

But how to handle sentence (ordered sequence of words) and paragraph (ordered sequence of sentences) level similarities?

Indeed, our case study involves paragraph. In fact, the consequent question is to know how to build classification model which is aware of ordered sequences instead of fixed bag of values (common feature vector). It depends on the track (DL vs non-DL): DL comes up with Recurrent Neural Network concept which is supposed to capture sequential nature of the training data.

BagOfWord (Co-occurrence) vs Context (example illustration)

Complex relations between words (hierarchical). Symbol meaning

Character/Word/Sentence

Multi-sentence representation ?

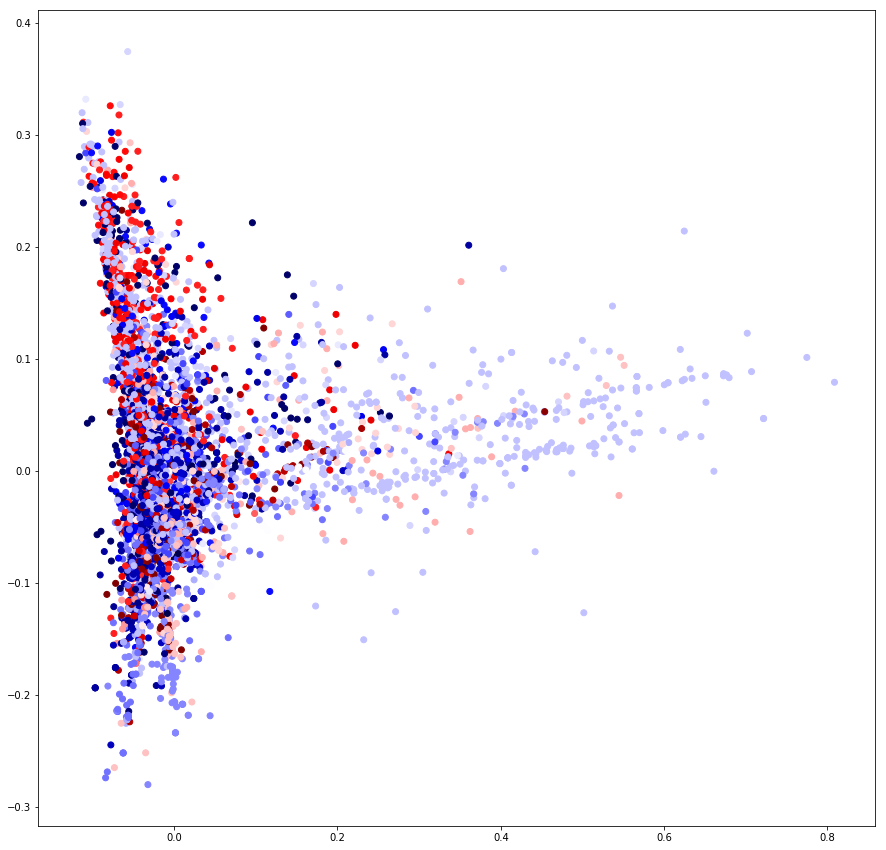
One sentence is elected (build a model per sentence of the document or last sentence as a question), ensemble per-sentence models or continuous/unique sentence (no punctuation)

Convolution => hierarchical representations of whole sentences

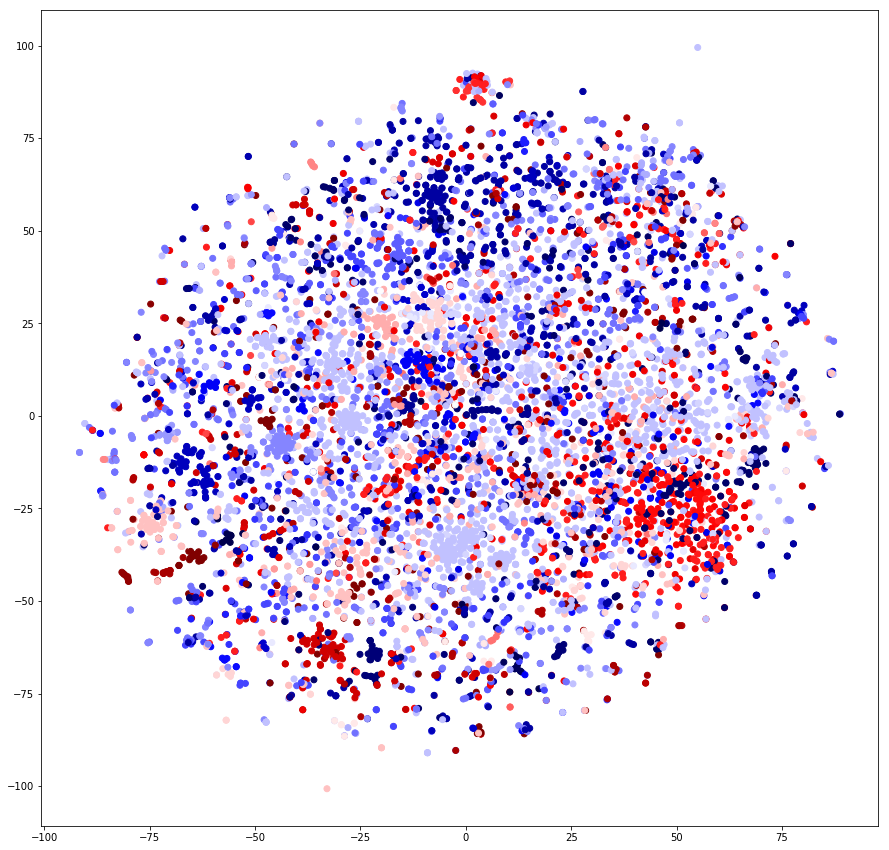
To have a fuzzy glimpse of how the classification would be difficult (separability in short), it’s worth to perform some visualizations of the feature space distribution.

The multi-sentence question text is basically converted into of bag of words which is then vectorized with TF-IDF transformation. Each question is then represented as a mere data point inside the global vocabulary space. To make such data points humanly observable, a dimension reduction of these features are required at the cost of some approximations: we will use both linear/fast PCA dimension reduction technique and non-linear/slow t-SNE. The data point color is determined by the associate target class.

**PCA-reduced feature space**



**t-SNE-reduced feature space**



Both shapes are very different but as expected, we observe on both visualizations that the class boundaries are extremely complex (no clear isolated clusters) and we can already assert that the data challenge raises the bar very high if the classification relies solely on word co-occurrence basis without sequential/context awareness.

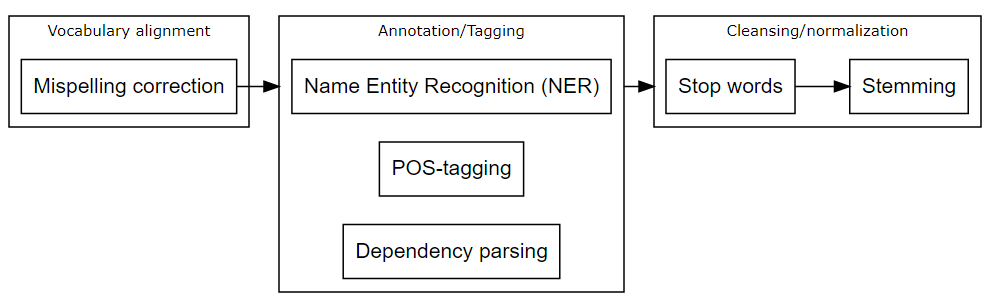
***General NLP Architecture***

Text classification is a common but non-trivial NLP topic following usually below principal steps:



* **Text pre-processing**

It’s all about operations on the raw text to make the text more reliable/workable to extract relevant information. It falls into 3 categories: vocabulary alignment, metadata tagging and finally cleansing/normalization.



* **Feature enrichment**

This step consists in adding a priori extra features which may explain the target much more than the raw features. These additional features can be derived from the raw text (eg: basic statistics like the count of question marks, the text length, …) or can be brought from external resources (eg: once a drug entity has been tagged/identified, associate to the sample the relevant characteristics of the concerned drug)**.**

* **Feature representation**

This is an instrumental stage where the “best” representation of the text should be determined so that the whole dataset will be encoded accordingly. Usually, the text input needs to be vectorized numerically because most of ML classifiers can only cope with numeric values as numerical algorithms and mathematics operate under the hood and they don’t care about symbol and semantic carried by the text content.

The “best” representation can be defined on specific criteria, typically by paying attention on the semantic similarity between words: Word2Vec can be utilized as an encoder for DL track and GLOVE for non-DL track (a.k.a traditional technique)).

* **Classification modeling**

Last but not the least, the final stage is aimed to build a classification model by learning on the encoded training dataset.

We explicitly distinguish here the 2 modeling tracks (DL vs non-DL).

This is the general NLP text classifier framework/guidance but for practical reasons, some steps will be skipped or greatly simplified to fit the project timeframe.

***ML Workbench Environment***

All experiments have been written in Python within the popular Jupyter environment: all notebooks are stored into a Github project whose details are provided in the annex section. This project makes use of diverse python packages to fulfill different requirements:

* data manipulation and visualization: pandas, numpy and matplotlib
* text processing (stemming, stopWords, …): nltk, standard regex and spellChecker (built from github)
* traditional ML algorithms (XGBoost, PCA, t-SNE): sklearn and XGBoost
* DL framework: Keras + Tensorflow

The rationale to select above packages from the ML python zoo will be given in the upcoming/specific DL and traditional sections.

In addition to the runtime packages, this project also takes advantage of public resources (nltk corpus, pre-trained FastText model, ….).

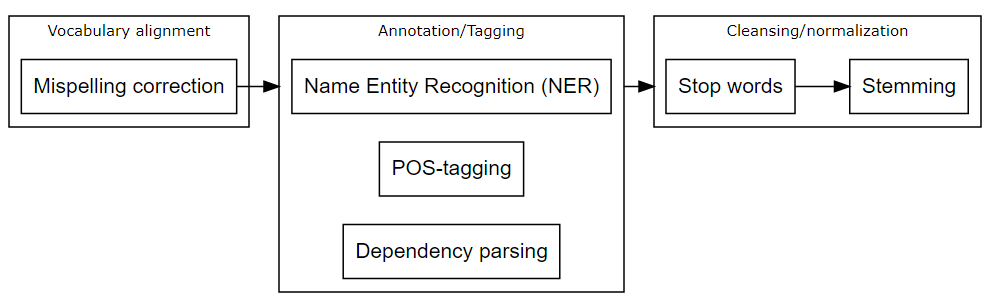
Most of the ML tasks had been executed with a 32-cores PC workstation (no NVIDIA GPU to take advantage of CUDA acceleration for both Tensorflow and XGBoost). For some data intensive jobs, an Amazon AWS GPU instance was allocated to unload slightly the PC workstation burden: for the anecdote, my first DL training running overnight has definitively damaged the MacBook pro laptop …

The Amazon EC2 infrastructure is flexible and convenient for temporary and well-defined training jobs but at a high cost (5$/hour): even if the GPU/CPU resource is not utilized, the data storage is charged permanently.

I finally bought a PC gamer machine with mid-range Nvidia GPU card to carry out the huge computing workload: the GPU acceleration was enable for XGBoost and DL learners.

***Text Preprocessing***

**Text Pre-Processing Elaboration**



The very first task is to detect and fix the misspelled terms on the fly. For that purpose, I used the package <https://github.com/barrust/pyspellchecker> which returns for each misspelled term the most probable word based on the word frequency distribution built from very large general purpose corpus.

As the question texts are medical domain oriented, I have created a custom vocabulary from the public **RCP** (Résumé des Caractéristiques du Produit) repository maintained by the ANSM administration. This ground truth corpus enables typically to extract specialized terms which are OOV (Out Of Vocabulary) related to the general purpose dictionary.

**NER** is a NLP subtask aimed to tag term from a text with predefined categories (eg: location, person, quantity, …). NER may be useful to identify terms related to specific medical entities (drug name, chemical composition, disease, …): indeed, the co-occurrence of certain medical entities and gold standard (time, quantity, …) in a question may indicate a particular intent.

**POS (Part Of Speech) tagging** reveals the grammar role of each word to possibly disambiguate its semantic. Such process can be useful to locate the interrogative proposal sentence and the word role (adverb, noun, verb, …).

**Dependency parsing** is a NLP technique to discover the grammar dependencies between words within a sentence. This is also a good annotation process to understand word interaction and connect the entities to each other: it’s probably helpful to get a better classification score.

Unfortunately, the most popular package supporting advanced POS tagging and dependency parsing features (namely Stanford NLP) has no official French corpus and only deal with English grammar.

For the **stopWords** and **stemming** final steps, I merely leveraged the nltk French corpus: this process simplifies gracefully the phrase structure but at the expense of its semantic and grammar correctness.

In complementary to the stemming process which encodes a word into its etymological root for normalization sake, one idea is to convert word into its most frequent synonym if existed: it likely reduces the distance between 2 semantically similar sentences. Again, unfortunately, the famous **wordnet** database available from nltk package doesn’t support officially French language.

As a reminder, all this pre-processing steps are part of the common trunk between the DL and non-DL predictive pipelines.

**Text Pre-Processing Implementation**

**TODO**

***Feature Enrichment***

**Feature Enrichment Elaboration**

The primary feature (pre-processed question text) with NLP annotation from the training dataset can be enriched with extra features:

* some handcrafted statistics on the text characteristics can be computed

eg: text length, number of question marks, number of drug occurrences (thanks to the custom NER tagging), number of quantity gold standard occurrences, times, …

* if the drug entity has been identified within the question text, it would be worth to incorporate as additional features, relevant characteristics of the concerned drug extracted from the ANSM database.



For instance, the drug product classes don’t provoke equally the same question natures/intents: the vaccine may raise more questions and concerns on contraindication (a class of intent?).

To combat the training size shortage impose by the data challenge, one option to enrich the former dataset is to complement it with other medication questions from public forum like the one hosted by doctissimo.fr. These extra observations are unlabeled (the 51 target classes are anonymized) and so cannot be used as regular dataset for the supervised training. Nevertheless, it can be consumed to complement gracefully the text corpus for all unsupervised learnings (eg: Word2Vec) to strengthen the modeling result. Such trick is also applicable for unlabeled test dataset of the challenge.

**Feature Enrichment Implementation**

**TODO**

***Traditional Technique***

**Feature Representation**

Once the text has been prepared, it’s time now to find a more suitable representation for machine operations. Indeed, most of ML algorithms operate upon structured data and cannot cope with unstructured data like text, audio or video: relational representation is the de facto standard where each data is organized by column (feature) holding a specific semantic (eg: gender, age, …)

The standard way to transform text into tabular representation is the **TF-IDF** (**T**erm **F**requency **I**nverted **D**ocument **F**requency) where each word of the vocabulary is represented as a column and the document (text) is represented as a row: the weight computation takes care of the term/word frequency from the considered 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 not sufficient. The usual solution to avoid this pitfall is the dimension reduction technique.

Stemming pre-processing already reduces the vocabulary size by projecting terms into root word space, but it doesn’t capture synonym relationship and in general, terms which are conceptually/semantically related like bird<->duck.

The **word embedding** addresses this matter indirectly: this technique estimates statistically from a large text corpus the co-occurrence between 2 words in a text context. This vectorization process makes that 2 words which are contextually similar (inter-changeable) are represented as 2 close numerical vectors in the target space.

**GloVe** (Global Vectors for Word Representation) implements the word embedding in a non-DL fashion (conversely to **Word2Vec**): it uses behind the scene matrix factorization method such as LSA (Latent Semantic Analysis).

Once again, probably due to the French language curse, GLOVE is only available for English language. I finally felt back to the **PCA** linear dimension reduction technique which basically keeps the top eigen vectors capturing the maximum of the data distribution variance: PCA is applied upon numerical space generated by the TF-IDF vectorization. This default option is not satisfactory at all because it’s not driven by any context/semantic considerations like GLOVE.

In short, each question (document) is encoded into a vector in a reduced numerical space.

TODO. How to model sequence?

**Classification Modeling Choice**

For the non-DL classifier selection, instead of putting into competition several candidates (SVM, random forest, logistic regression, …), I bet on the **XGBoost** library 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 classifier comes up with many hyper-parameters to tune. As usual, the parameter selection is driven by the cross-validated (k=5) error assessment.

To avoid endless grid search over all available parameters, a good practice is to focus on the most influent ones and fix them in a particular order:

* https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

The objective function is set to softprob for multi class classification task, the early stopping is set to 20 with the max number of estimators to 200 and other parameters are set to their respective default value.

**Implementation & Execution**

Build GPU xgboost

Misspelled > stopword > stemming

Detail the concrete work (processing time, hyper param search, early stopping, …)

CV error

Confusion Matrix

Intention 31

***Deep Learning Technique***

DL is considered as a universal estimator able of fulfilling any kind of modeling requirements from the feature representation layer to the final decision layer inside a single (and complex) neural network. The key benefit of this holistic capacity is that the loss function optimization to find out the modeling parameters (weights, …) operates consistently across all functional layers regardless of their respective purpose (embedding, decision making, …). In contrast, with traditional technique track, feature representation and classification are 2 concerns which are engineered/optimized separately.

**Feature Representation**

Doc or Sentence Embeddings Mean? (eg: Mean Word Embedding)

Word embedding is available in a DL flavor with **Word2Vec** (Google) and more recently with **FastText** (Facebook): both of them are issued from Mikolov’s research work.

My first thought was to build the embedding model from POSOS training set which contains domain specific terms (drug name like Xanax, active chemical ingredient, …), but the corpus is really too small and badly trustable (misspelled terms, familiar expression, …).

It’s preferable to take advantage of the embedding models learnt from very large and diverse text corpus, which are publicly available from the Web. I only found French trustable models for FastText which in addition offers the capacity to cope with **OOV** (Out Of Vocabulary) case: indeed the embedding model is trained at character level instead of word level.

I experimented both embedding model sources and the general purpose one provided definitively better classification performance (roughly 20% difference in term of classification rate).

The best of breed solution would be to extend the general embedding model by incorporating the pharmaceutical target corpus constituted by the training set and other reliable sources (ANSM, …). This is another and bigger story requiring huge amount of computational resources to rebuild this super model. Even the learning transfer in lieu of an embedding training from scratch is also complicated and out of reach in the context of this study project.

I finally used the French FastText model encoding words into a 300 dimensions space. Medical terms referred in the question texts are either drug product names, special symptoms or active chemical ingredients: such key words are probably absent from the FastText model’s corpus.

How to solve this predominant OOV (Out Of Vocabulary) matter?

One solution is to merely project such unknown words into its **hypernym** more common term with loss of details penalizing indirectly the final classification performance:

* drug product name (Xanax, Abboticine) is converted into constant term ‘médicament’
* active ingredient (eg: Acabavir) is converted into constant term ‘ingrédient’
* …

This projection into hypernym/entity space is possible only if the terms are correctly tagged in the text preprocessing phase (name entity recognition). It’s likely smarter than the common practice to encode unknown words into a random or fixed vector.

“These word embeddings are now the state-of-the-art in NLP. However, it is less clear how we should best represent a sequence of words. a whole sentence, which has complicated syntactic and semantic relations”

local and long-range dependencies

Variable sized vs fixed sized (padding)

**DL Architecture Elaboration**

A possible (but simple) architecture is merely composed of a **Doc2Vec** embedding layer plus a fully connected layer for the classification task.

In practice, **CNN** (Convolutional Neural Network) and **RNN** (Recurrent Neural Network) layouts appear to be predominantly chosen in ML competition for text classification: some practitioners/researchers even recommend combining them as they are complementary.

The hybrid solution mixing up CNN and RNN is not considered here for simplicity sake.

**Doc2Vec based Architecture**

This network has 3 main hidden layers:

* Doc2Vec vectorizes the whole multi-sentence question into a single numerical vector
* A series of fully-connected/dense layer plus dropout layer
  + the activation function would be the sigmoid as we are dealing with shallow network
  + the dropout operations ensure the learning regularization to avoid the overfitting pitfall
* last dense layer with softmax as activation function to produce normalized scores we can interpreted as classification probability

Identically to Word2Vec, it’s highly preferable that the embedding model is trained on a very large and representative corpus but I didn’t find any pre-trained public Doc2Vec models from the DL community. The consequence is that I built my own Doc2Vec model from the pre-processed training and test sets of the data challenge.

The primary downside of the Doc2Vec approach is that in contrast to Word2Vec, we lost the word sequence after the encoding, so that it’s impossible afterwards to model the document semantic based the context defined by the word sequence.

This proposition sounds very naïve but it can serve as a comparison baseline to illustrate the value proposition of more sophisticated CNN and RNN architectures which are capable to extract the context from the sequence.

**CNN Architecture**

In **C**omputer **V**ision (**CV**), convolution operation is well known to be efficient to extract the high level representation of an image by passing 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 capture the high level structure 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.

Some research studies demonstrate that convolution is powerful enough to cover embedding requirements making Word2Vec useless. Nevertheless, I preferred to keep going with an embedding layer relying on a proven algorithm (Word2Vec) associated with a convolution layer specialized to extract the high level representation of the text.

In short, here’s the neural network layout:

* word embedding layer as explained in the Feature Representation section
* a series of 1-dimensional convolutional and pooling layers
  + The best practice recommends building many convolutional layers with different filter sizes and/or strides
  + I used Relu (Rectified linear unit) activation function after the convolution operation and the popular maxpooling
* a series of dense layer plus dropout layer
* last dense layer with softmax as activation function

**RNN**

In a classical neural network, we assume that all inputs of a node are independent to each other but it’s not applicable in NLP because text is composed of an ordered sequence of words (inputs) which are inter-related.

Recurrent neural network architecture tries to leverage this sequential information with a particular 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: 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 bypass this long-tailed sequential dependency we typically observe in multi-sentence text analysis where the context may be specified upfront far 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 final architecture looks like below:

* word embedding layer as explained in the Feature Representation section
* bi-directional LSTM layer with dropout (for learning regularization)
* dense layer with softmax as activation function

**Implementation & Execution**

I coded all DL experiments with **Keras** as a frontline API on top of Tensorflow engine: Keras provides a very comprehensive and easy-to-use API hiding the Tensforflow verbosity.

Here’s an example of Keras code where the layers are defined in a very concise way thanks to helper object like Convolution2D, Activaction and so forth.

model = Sequential()

model.add(Convolution2D(32, kernel\_size=(3, 3),padding='same',input\_shape=(3 , 100, 100)))

model.add(Activation('relu'))

model.add(Convolution2D(64, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

In general, the model performance depends importantly on the selection of the hyper-parameter values: as the DL training is really slow even with the aid of GPU boosting, I was not able to test a lot of hyper-parameter selections and I proceeded with default or best guess values at the risk to be far from the optimum point.

**Common Setup**

GPU enable config

**Doc2Vec based Implementation**

**CNN Implementation**

**RNN Implementation**

CUDNNLTSM

**Experiment Results**

Nb params to optimize vs nb observation (overfitting) => show error estimation vs validation evolution

***DL vs non-DL Comparison***

**Implementation/Engineering**

**Model Accuracy**

Focus on dominant class error

***Potential Improvement Tracks***

**Feature Engineering**

Building a **custom** **embedding model** from text corpora which are specific to the data challenge domain is a real plus: indeed, the generic FastText model has been trained on clean corpora writing in an elevated literary style by professionals (Wikipedia, newspaper, …) whereas the POSOS text style is much more familiar as questions are mostly raised by uninitiated people.

As POSOS training size is not sufficient, it can be completed with other unlabeled drug question sources (eg: doctissimo.fr forum, …): just to remind that embedding is an unsupervised learning process requiring no exhaustive manual labeling.

Another improvement possibility is to refine the proposed **imputation** that replaces words unknown from the embedding model standpoint, by its named entity (hypernym). Typically, train our own NER (Named Entity Recognition) on French general/domain-specific corpora combination with nltk package: the trick here is to take advantage of the English NER model provided by Stanford NLP active community and the English-to-French word translation, in order to get for free the global standard label on French corpora.

ANSM source of information on drug product is poorly exploited: I only extracted dictionaries of drug product names and active ingredients to identify such entities into the text because it’s straight-forward (ANSM public dataset contains a drug product name and active ingredients columns). Other relevant entities (adverse effect, target body part, drug category, …) should be learnt from the ANSM corpus with manual entity labeling.

For the handcrafted features to characterize the text, thanks to a more complete entity identification (adverse effect, body part, quantity, …) , it would be possible to compute more count-based statistics on entity and even introduce selective entity distance to capture indirectly the word sequential distribution.

**Classification Modeling**

At the level of classification modeling layer, there’s room for improvement as well. Due to lack of time and computing resource shortage, no extensive hyper-parameters search has been performed mainly in the DL sub project: we know for sure that parameter optimization is crucial.

I artificially separated the DL and non-DL techniques in order to make an academic comparison but in practice, the data scientists build heterogeneous and complex predictive pipeline involving a variety of algorithms and technologies, by sticking with the ensemble learning principle. The DL architecture may be reinforced with handcrafted features utilized in the non-DL approach. In the same vein, CNN and RNN architectures are tested separately but they can be combined nicely into a single neural network.

***Conclusion***

In this present paper, I describe a walkthrough work attempting to solve a complex NLP text classification in a **DL (D**eep **L**earning) and non-DL fashion (a.k.a. traditional track). The goal of the study is to have a fair idea on the pro and cons between above 2 approaches from a practical standpoint on a real business use case.

The non-DL track enrolling notably the venerable XGBoost classifier produces poor modeling performance (X % accuracy in cross validation) mainly due to its inability to leverage natively the sequential context from the text structure: this approach is furthermore penalized by the absence of pre-trained word embedding model based on non-DL machinery (eg: GloVe matrix factorization) and not surprisingly, the handcrafted statistics on text don’t improve at all the final accuracy. Nevertheless, its implementation and execution are pretty straight/simple accommodating with mid-range CPU.

DL track with RNN/LSTM architecture outperforms the non-DL candidate in term of accuracy (X % on validation set), partly thanks to the available pre-trained FastText embedding model and its sequential structure awareness. The counterpart is that DL algorithm is hungry of GPU and badly runs on standard machine configuration: on top of that, defining the appropriate architecture (operation composition) and tuning the hyper parameters (activation function, dropout rate, …) demand some strong practical engineering experiences to achieve good enough result.

In addition to the suboptimal ML procedure I had built so far, the low accuracy on validation can also be explained by the learning materials: indeed, the training set is really too small (~8000) with regards to the 51 labels and most of the texts are written in an everyday language style with plenty of abbreviations and misspellings.

Even if the comparison is totally unfair/biased (as I had burned more brain cells, chipset and times on the DL sub project), Deep Learning turns out to be the most efficient ML technique in NLP task solving: it’s presently a very active research topic within the ML community as DL expressiveness/capacity are unbound similarly to the brain plasticity.