Machine Learning Engineer Nanodegree - Udacity Capstone Project

Predictor of Eligibility in Clinical Trials on Cancer
Exploration of NNs and Deep Learning
Toward an Evidence-Based Clinical Decision Support System

Aurelia Bustos, MD, BsC

I. Definition

Project Overview

Clinical trials (CTs) provide the evidence needed to determine the safety and effectiveness of new treatment methods in medicine. CTs are the basis for clinical practice guidelines[1] and lead clinician in their treatment decisions on the daily practice. However, eligibility criteria used in CTs in oncology are too restrictive[2] -typically patients are excluded based on comorbidity, past or concomitant treatments and older ages- and therefore these highly selected patients do not mimic clinical practice. Consequently, the results obtained in CTs can not be extrapolated to patients if their clinical profiles were excluded from the clinical trial protocols. Finding out if a patient case -given his clinical characteristics, type of cancer and intended treatment- is represented or not in the available corpus of CTs requires the manual review of numerous eligibility criteria, which is impracticable for clinicians on a daily basis, and therefore the process would greatly benefit from an evidence-based clinical decision support system (CDSS)

On this project, I constructed a dataset using the clinical trial protocols published in the largest public registry available[3] and used it to train and validate a model able to predict if short free-text statements (describing clinical information like medical history, concomitant medication, type and features of tumor, cancer therapy etc.) were considered as eligible or not eligible criterion in clinical trials. This model is intended to inform clinicians if the results obtained in the CTs - and therefore if the recommendation in the standard guidelines - can be confidently applied to a particular patient.

The ultimate goal of this project was to serve as a proof of concept that the latest NLP deep learning and ML techniques can be successfully applied to extract the medical knowledge available on clinical trial protocols, opening the avenue to more involved and complex projects.

Problem Statement

The problem consists in predicting if short free-text statements (describing clinical information like medical history, concomitant medication, type and features of tumor, cancer therapy etc.) are considered as *Eligible* or *Not eligible* criteria in clinical trials. Eligibility criteria are expressed in natural language and medical concepts are expressed in many different ways. Many studies have focused on the problem of formalizing eligibility criteria and obtaining a computational model that could be used for clinical trial matching and other semantic reasoning tasks. There are several languages which could be applied for expressing eligibility criteria e.g. Arden syntax, Gello, ERGO and others. Weng et al.present a rich overview of existing options[4]. SemanticCT[5] allows the formalization of eligibility criteria using Prolog rules. Milian et[6] al applied ontologies and regular expressions to express eligibility criteria as semantic queries. Still the problem of structuring eligibility criteria in clinical trials to obtain a generalizable computational model that can represent them remains unsolved.
On this project, I explore the use of neural networks (NNs) and deep learning techniques (DL) to achieve semantic interpretation directly that, in contrast to traditional NLP approaches, omit the constraints and limitations of previous steps like tokenization, stemming, syntactic analysis, named entity recognition (NER), tagging of concepts to ontologies, rules definition or manual selection of features to construct the model. Concretely, the problem with discrete representations or words, like taxonomies and ontologies, is that they miss nuances, new words (e.g. it is impossible for them to keep up to date with the new drugs in cancer research), are subjective, requires human labor to create and adapt and it is hard to compute accurate word similarity[7].
The proposed solution will first preprocess the text to construct a training and validation labeled set where the labels are *Eligible* or *Not Eligible*, then after extracting bigrams and word-embeddings it will explore using them in two different models using state of the

---

[1] https://www.nccn.org/professionals/physician_gls/f_guidelines.asp; http://www.esmo.org/Guidelines
[2] J. Clin. Oncol., 14 (4) (1996), pp. 1364–1370
[3] https://clinicaltrials.gov/
[4] 10.1016/j.jbi.2009.12.004
[5] http://wasp.cs.vu.nl/sct/
[6] http://dx.doi.org/10.1016/j.jbi.2015.05.005
[7] Deep Learning for Natural Language Processing (Richard Socher, Salesforce)

art neural network text-classifiers. Finally after validating and comparing the final two text classifier models, the best model will be further tested against an independent testing set.

Metrics

To measure the performance of the model, the F1-score, precision and recall, confusion matrix and the Cohen's Kappa coefficient of agreement will be reported.

The Cohen's Kappa is a statistic which measures inter-rater agreement for qualitative (categorical) items. It is generally thought to be a more robust measure than simple percent agreement calculation, since κ takes into account the possibility of the agreement occurring by chance.[8] The Kappa is calculated as:

$$\kappa = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e},$$

where Po is the relative observed agreement among raters, and Pe is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly saying each category.

Precision (also called positive predictive value) is the fraction of retrieved instances that are relevant, while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved. Precision and are calculated as:

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

F1-score is the harmonic mean of precision and sensitivity. F1-score is calculated as:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

II. Analysis

Data Exploration

A total of 6,186,572 labeled clinical statements were extracted from 49,201 interventional CT protocols on cancer published in clinicaltrials.gov (to download them use the query below [9], last downloaded on Feb 2017). A random subsample (10^6 samples) of the resulting dataset after the data preprocessing (see steps below) is available in Kagle[10]

Each CT downloaded is a XML file that follows an structure of fields defined by an XML schema.[11] The relevant data for this project are derived from the intervention, condition and eligibility fields written in unstructured free-text language. The information in the eligibility criteria - both exclusion and inclusion criteria - are sets of phrases and/or sentences displayed in free format like paragraphs, bulleted lists, enumeration lists etc.. None of those fields use common standards and none of them enforce the use of standardized terms from medical dictionaries and ontologies and the language suffered from both polysemy and synonymy .

To exploit the original data, eligibility criteria together with study condition and intervention were merged and transformed in lists of short labeled clinical statements as shown in Fig. 1

---

[8] https://en.wikipedia.org/wiki/Cohen's_kappa

[9] https://clinicaltrials.gov/ct2/results?term=neoplasm&type=Intr&show_down=Y

[10] https://www.kaggle.com/auriml/eligibilityforcancerclinicaltrials

[11] https://clinicaltrials.gov/ct2/html/images/info/public.xsd

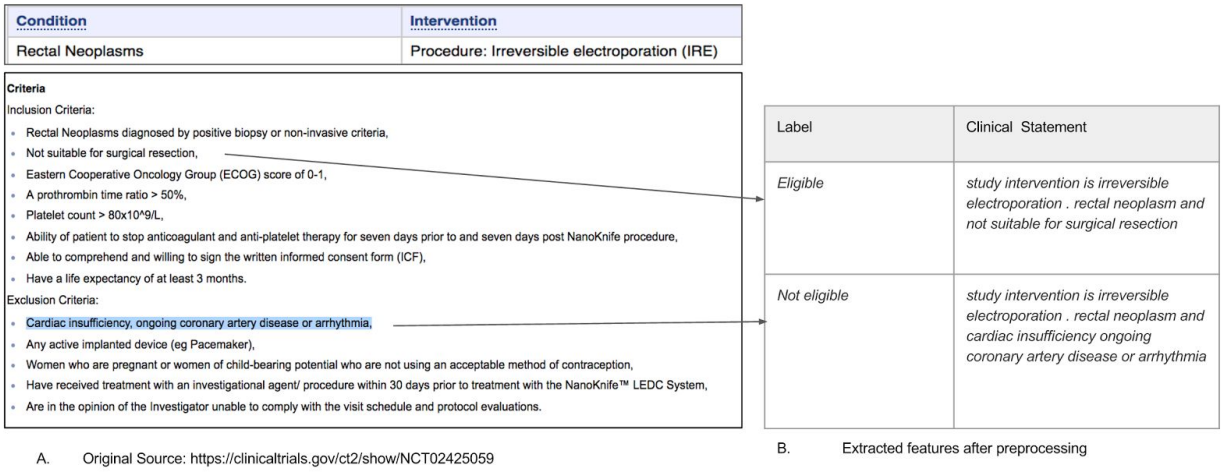


Fig. 1: Extraction of labeled short clinical statements. With two examples obtained from its original source after preprocessing

After the preprocessing and cleaning phase the available set had more than 6.1 M of labeled short clinical statements (see Fig. 2).

```
Loading text dataset
          eligible                                        eligibility
count      6186572                                         6186572
unique           2                                         5483936
top      __label__0  study interventions are Cyclophosphamide . lym...
freq       4150835                                             450
```

Fig. 2: Dataset. The column “Eligible” contains the label binary variable with possible values: __label__0 for *Eligible* and __label__1 for *Not eligible*. The column “Eligibility” is the clinical statement or eligibility criterion variable that contains the short text description.

The label variable was imbalanced: there were up to 4.1 M (61%) entries labeled as *“Eligible”*. Imbalanced data is a typical problem in labeled datasets that will be properly addressed on this project.

The “eligibility” variable containing the text for each criterion, as expected in NLP, had a highly sparse distribution and only 450 entries were repeated.

Exploratory Visualization

A relevant task is to analyze and visualize in an histogram (see Fig. 3) the distribution of number of words in the variable eligibility which is expected to contain short text statements:

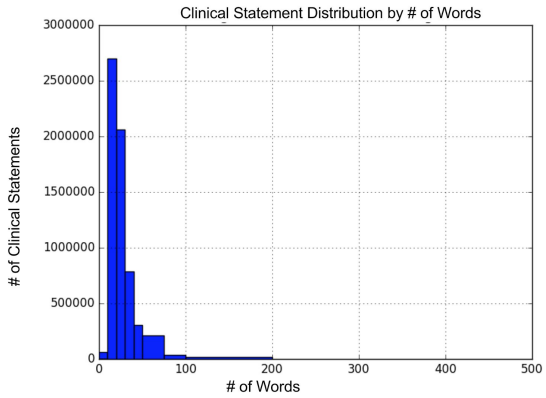


Fig. 3: Histogram of clinical statements by number of words: # of words= 148.038.397, # of statements=6,186,572, minmax # of words=(6, 439), mean=23.9 words/statement, variance=171.3, skewness=3.13, kurtosis=21.05

The majority of statements contain less than 50 words (mean 23.9). The skewness of the distribution, with a long and thin right tail for statements longer than 75 words, manifests that the text splitting in the preprocessing is not separating properly a small subset of clinical statements into short sentences and could be further improved (beyond the scope of this project).

Algorithms and Techniques

Once the data was preprocessed and bigrams have been substituted in the input text (see section Data Preprocessing) I attempted sentence classification exploring two different approaches based in NNs using FastText and a deep learning model using CNNs:

CNN text Classifier using TensorFlow and prebuilts word-embeddings

The input to NLP tasks are sentences represented as matrix. Each row of the matrix corresponds to one token, typically a word. Each row is a vector that represents a word and these vectors can be word-embeddings or one-hot vectors. On this project I used word-embeddings because of their advantages over one-hot vectors:

- One-hot vectors are high-dimensional and sparse. Using them as a feature in a classifier, causes the feature vector to grow with the vocabulary size (n = 49222 different words). They do not have the ability to generalize and a classifier model using them as features will only be able to consider words they have seen during training. In addition they do not capture semantic or syntactic relationships: vectors of semantically similar words has cosine similarity 0

- Word-embeddings are more computationally efficient, are low-dimensional and dense. More importantly, they have the ability to generalize due to semantically similar words having similar vectors. They are learned representation of the semantic meaning into the geometric space. Similar words will be clustered in a specific region of this multidimensional space. As a result, the cosine distance between any two word vectors capture part of the semantic relationship between those words.
  For example, by adding and subtracting word-embeding vectors we can obtain analogies like this:
  king - man + woman = queen

Inputs to neural networks over text are word-embeddings. Learning and re-using word-embeddings is a form of transfer learning. Pre-generating word-embeddings, instead of training them on the classifier model (see next section), allows reusability (with potential transfer-learning when trained with a larger dataset that the one used to train the classifier model) and facilitates evaluation. To evaluate them before using them as input layer in the classifier model I analyzed the word-clusters and also words were explored and visualized in the vector space.
Word embeddings models learn geometrical encodings (vectors) of words from their co-occurrence information (how frequently they appear together in large text corpora). Word2vec[12] is a "predictive" model, whereas GloVe[13] is a "count-based" model. Word2Vec takes 'raw' text as input and learns a word by predicting its surrounding context (in the case of the continuous BoW model) or predict a word given its surrounding context (in the case of the skip-gram model) using gradient descent with randomly initialized vectors. On this project I used the Word2Vec skip-gram model.

Pre-trained word-embeddings were used as the input layer to a 1D - Convolutional NN model with a final dense output layer.

CNN text Classifier using TensorFlow and word-embeddings initialization

On the next experiment I trained the word-embeddings within the classifying task to learn them from scratch. As the training data was large enough and also the vocabulary coverage was appropriate for the domain of cancer research, it could be expected that the model would benefit from training the embeddings on the task, at the cost of reusability.

FastText Classifier

Fast Text classifier[14] reportedly obtains an accuracy almost as good as the state of the art in deep learning, but it is several orders of magnitude faster. It starts off with an efficient embedding layer which maps the vocab indices into embedding_dims dimensions. It then adds a GlobalAveragePooling1D layer, which average the embeddings of all words in the document. Finally it projects it onto a single unit output layer, and squash it with a sigmoid. [15]

Benchmark for sentence classification

[12] https://code.google.com/p/word2vec/
[13] http://nlp.stanford.edu/projects/glove/
[14] arXiv:1607.01759v3
[15] https://github.com/fchollet/keras/blob/master/examples/imdb_fasttext.py

Depending on the dataset used different accuracy results have been published. For instance the reported accuracy for classification of the hackernews posts in 20 different categories using a similar method was 95%[16], while for “Movie reviews” the reported performance was 81.6%. [17]

In the medical domain, a model with high performance is necessary to be potentially useful in a CDSS. Based on previously published computer aids systems[18], I defined for accuracy a minimum benchmark of 90% and for Cohen’s Kappa a minimum of 0.61–0.80 (substantial agreement) or 0.81–1 (almost perfect agreement).[19]

III. Methodology

Data Preprocessing

Preprocessing before word-embeddings:

The task consisted in transforming all eligibility criteria to sequences of plain words (and bigrams) separated by a whitespace augmented with information on study intervention and cancer type. It was done by:

1) Splitting text in statements: I implemented a sentence splitter that took into account different kinds of bullets and lists and also did not mistakenly split as sentences abbreviations like mutations or any medical notations which included “.” or “:” or “-”
2) Removing punctuation, white-space characters, all non alphanumeric symbols, separators, single-character words from the extracted text. All words were lower-cased.
3) Transforming numbers, arithmetic signs (+/-) and comparators (>, <, =, ...) to text.
4) Detecting bigrams and replacing them in the text. Bigrams are commonly found phrases or multiword expressions and are very frequent in medicine. Phrases are collocations (frequently co-occurring tokens). Bigrams can represents idiomatic phrases that are not compositions of the individual words, therefore feeding them as a single entities to the word-embedding instead of each of its word separately, allows learning those phrase representations. Some examples of bigrams on these dataset are:
    - sunitinib malate
    - glioblastoma multiforme
    - immuno histochemistry
    - von willebrand
    - dihydropyrimidine dehydrogenase
    - li fraumeni
    - gamma knife
    - vinca alkaloid
    - myasthenia gravis

Preprocessing for classification model:

The goal was to obtain labeled short clinical statement from eligibility criteria, study conditions and interventions. It included all the above described steps plus the following ones:

1) Labeling each statement with the label “Eligible” (Inclusion criteria) or “Not eligible” (Exclusion criteria) based on:
    a) Their position relative to the phrases “inclusion criteria” or “exclusion criteria” which usually preceded the respective lists. If those phrases were not found, then the statement were labeled “Eligible”.
    b) Negation identification and transformation: negated inclusion criteria starting with “no” were transformed to a positive statement and labeled “Not eligible”. All other possible ways of negative statements were expected to be handled intrinsically by the NN classifier models.
2) Augmenting data by adding the cancer types and interventions under study to each criterion using statements like: “patients diagnosed with [cancer type]”. For CTs that studied multiple cancer types or interventions I replicated each criterion by each intervention and condition.

Note: I also attempted to classify treatments in order to use them as labels instead of “Eligible” or “Not eligible” (see Improvements section). The source code already include those methods but need additional improvements in the preprocessing that is left for a future project. The preprocessing code is in the preprocessor.py script

---

[16] “HackerNews” https://blog.keras.io/category/tutorials.html

[17] "Movie reviews with one sentence per review" (Pang and Lee, 2005)

[18] https://en.wikipedia.org/wiki/Clinical_decision_support_system

[19] doi:10.2307/2529310

Implementation and Refinements

Bigrams

For phrase (collocation) detection I used gensim API[20]. The threshold parameter defines which phrases will be detected based on their score. The score formula[21] that it applies is:

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}.$$

For this dataset, after several tests, the best identified threshold was set to 500 and the discounting coefficient δ was based on a min count of 20. The discounting factor prevents too many phrases consisting of very infrequent words.

After running it, it collected 792091 word types from a corpus of 37153036 words (unigram + bigrams) and 2245967 sentences. A total of 875 different bigrams were retrieved from the corpus.

Word-embeddings

I explored two different APIs (FastText[22] and Gensim[23][24]) to generate Word2Vec based on the skip-gram and cbow. models The main differentiating characteristic of Fast Text embeddings, which apply a char n-gram approach, is that it takes into account the internal structure or words while learning word representations[25] - this is especially useful for morphologically rich languages. FastText models with char n-grams do significantly better on syntactic tasks vs semantic tasks, because of the syntactic questions being related to morphology of the words.

1) FastText word2vec model:

The hyperparameters used to generate 100 dimensional embedding with fastText were:

-lr 0.025 -dim 100 -ws 5 -epoch 1 -minCount 5 -neg 5 -loss ns -bucket 2000000 -minn 3 -maxn 6 -thread 4 -t 1e-4 -lrUpdateRate 100

It took 6.7 min to train the model (read 16 M words, number of words 21703, loss: 1.799). Using those resulting word vectors I generated word clusters fitting a k-Means model using sklearn. The model parameters were all set to default except for the number of clusters, that was calculated by applying a reduction factor of 0.1 over the total number of words to read (max 10.000). See implementation in clusterWordVectors.py. The resulting clusters are available in github[26]. Sampling at random 20 clusters a total of 16 were relevant judged on whether their words were syntactically or semantically related. Some examples of them are:

['mri', 'scan', 'imaging', 'radiographic', 'magnetic', 'resonance', 'scans', 'abdomen', 'radiological', 'mr', 'radiologic', 'image', 'technique', 'images', 'perfusion', 'sectional', 'weighted', 'spectroscopy', 'mris', 'dce', 'imaged', 'lp', 'neuroimaging', 'volumetric', 'mrs', 'multiparametric', 'mrsi', 'imagery']

['pelvis', 'skull', 'bones', 'skeleton', 'femur', 'ribs', 'sacrum', 'sternum', 'sacral', 'lfour', 'rib', 'humerus']

['pulmonary', 'respiratory', 'obstructive', 'asthma', 'copd', 'restrictive', 'emphysema', 'bronchiectasis', 'bronchodilator', 'bronchitis', 'bronchospasm', 'pneumothorax', 'ssc', 'bronchopulmonary', 'cor', 'expired', 'onel', 'congestion', 'airflow']

['abuse', 'alcohol', 'substance', 'dependence', 'alcoholism', 'addiction', 'dependency', 'illicit', 'recreational', 'user', 'illegal', 'misuse', 'abusers']

[20] https://radimrehurek.com/gensim/models/phrases.html
[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.
[22] https://pypi.python.org/pypi/fasttext
[23] https://radimrehurek.com/gensim/models/word2vec.html
[24] https://github.com/RaRe-Technologies/gensim/blob/develop/gensim/models/word2vec.py
[25] P. Bojanowski*, E. Grave*, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information
[26] https://github.com/auriml/capstone/blob/master/wordEmbeddings/fastText_cluster_words.txt

Note that medical abbreviations (like for example 'lfour' - L4 or 'mri' - Magnetic Resonance Imaging or 'copd' - Chronic Obstructive Pulmonary Disease') are also correctly clustered.

2) Gensim word2vect model:

Gensin word2vec model was trained with 3 workers on 22489 vocabulary and 100 features, using skip-gram and also cbow models, negative sampling (negative = 5), sample=0.001 and window=5. Model training spent 35 minutes.

Data was loaded directly from disk using an iterable of `sentences` and collected 49222 word types from a corpus of 39399003 words (unicode strings) and 2245967 sentences. Setting min_count (min word frequency) to 5 retained 22489 unique words (45% of original 49222, drops 26733), leaved 39352720 word corpus (99% of original 39399003, drops 46283). It also removed the 56 most-common words (using sample = 0.001) leading to a downsampling that leaved estimated 28755091 word corpus (73.1% of prior 39352720). The estimated required memory for 22489 words and 100 dimensions was 29235700 bytes

Once trained, the model was able to accurately resolve analogy problems like "Tamoxifen is used to treat breast cancer as X is used to treat prostate cancer?" using the built-in analogy functions of gensim :

```
model.wv.most_similar_cosmul(positive=['prostate', 'tamoxifen'], negative=['breast'])
```

[('enzalutamide', 0.9977753162384033), ('antiandrogens', 0.9721243977546692), ('abiraterone', 0.9515050649642944), ('finasteride', 0.9493220448493958), ('zoladex', 0.9456212520599365), ('adt', 0.9331877827644348), ('dutasteride', 0.9272363781929016), ('acetate', 0.9228686094284058), ('flutamide', 0.9159576296806335), ('leuprolide', 0.9102472066879272)]

Indeed, those are very precise results because all those drugs belongs to the hormone-therapy family of drugs used specifically to treat prostatic cancer, which are the equivalents of tamoxifen (hormone-therapy) for breast cancer. In other words, the model learned the abstract concept "hormone-therapy" as a family of drugs and was able to apply it distinctively depending on the tumor type.

TextClassifier

As previously described the dataset was imbalanced with only 39% of classes labeled "*Not eligible"*. To correct it, as the dataset was large enough, I used random balanced undersampling [27] resulting in a dataset reduced to 4,071,474 labeled samples. The implementing code is available in github in the script util.py.

- FastText

The implementation code is available at fasttext_text_classifier.py script. It uses a Python interface for Facebook fastText[28].

The script's method run_classifier accepts as argument the size to train and validate the model. To build the learning curves, it was run with 4 values for SET_SIZE (1000, 10000, 100000, 1000000) which returned samples with sizes limited by SET_SIZE or limited (if set to None) by balanced undersampling yielding a 4,07 M sample set. The method "generate_small_set" ensured that each set was balanced by class labels and randomly sampled from the full set.

For each sample set the following steps were done:

1. Using the sklearn StratifiedShuffleSplit cross-validator, the sample set was splitted 5 times into a training and a validation sets with proportion 0.2. Then on each split, the classifier model - using fasttext supervised method- was trained and validated using 100 epochs and 4 threads.
2. The scoring function used by the k-fold cross-validation was the average (with its standard deviation) of the F1-scores obtained in the loop.

To choose the parameters for the classifier model, the F1 score was compared between successive experiments: the best model was obtained with learning rate 0.1 and number of dimensions 100. Adding or not pre trained word-embedding or bigrams (as we can appreciate on similar validation curves on Fig. 5) did not impacted on results.

[27] http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[28] https://pypi.python.org/pypi/fasttext/0.8.3

- Tensorflow:

The Tensor Flow implementation used Keras library[29] . Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It is very intuitive and has been developed with a focus on enabling fast experimentation. The model implementation (available in fasttext_text_classifer.py) on this project, is inspired in the text classifier model using Keras on the 20 Newsgroup database[30] and after the necessary adaptations has the following steps:

1. Convert all sentences in the dataset into sequences of word indices. A "word index" is just an integer ID for the word. I will only consider the top 20,000 most commonly occurring words in the dataset, and truncate the sequences to a maximum length of 1000 words.
2. Shuffle, stratify and split sequences of word indices into a training and a validation set (0.2)
3. Prepare an "embedding matrix" which will contain at index *i* the embedding vector for the word of index *i* in the word index. Load this embedding matrix into a Keras embedding layer, set to be frozen (its weights, the embedding vectors, will not be updated during training).
4. Build on top of it a 1D convolutional neural network, ending in a softmax output over 2 categories.
5. Shuffle data with random seed during training before each epoch (n of epochs = 10)

The algorithms were first successfully run with a reduced test sample of 1000 sentences which yielded an accuracy on the validation data of 0.72. However, trying the entire dataset, while building the matrix that contained the sentence vectors to be used as inputs to the CNN, caused the process to run out of memory. This problem was solved by transforming those methods to yield generators instead of large matrices of sentence vectors.

At this point, the model training for the entire dataset on a MacBook Pro using CPU (note: I have installed CUDA on the mac, but Tensor Flow did not use GPU) was extremely slow, taking almost 4 hours for each epoch. For this reason, I installed tensorflow-gpu in an ubuntu machine using graphic card Nvidia GTX 1080.

The first attempt using tensorflow-gpu on the whole dataset, the process was excessively slow and the first epoch (the only one that I explored before terminating the process) yielded a very low accuracy (57%) both on the training and validation set. This poor performance was attributed to the generator returning sequence by sequence, so that the gradients were computed and updated for each sequence. Indeed gpu usage was only 27%, so the next improvement consisted in implementing batches. After implementing batches of 128 sentences to be returned by the generator, CUDA increased utilisation to 92% usage, gaining speed (see fig 4) and increasing also the predictive performance from the first epoch.

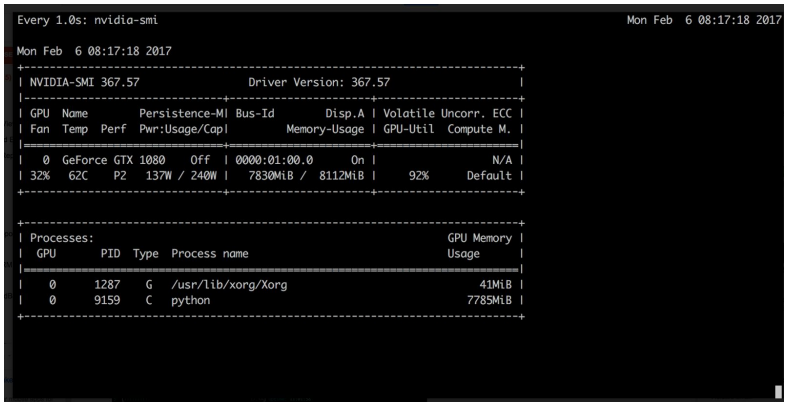


Fig. 4: Tensor Flow GPU usage during the CNN model training using batches of 128 sentences

IV. Results

Model Evaluation and Validation

Fast Text Classifier Model Results

After reading the reference literature and doing several modifications on the hyper-parameters space the predictive performance (using F1-score) of the classifier was optimized for size of word vectors = 100 (n. of dimensions) and 100 epochs with all other values set to default params. Results were similar with the use or not of pre-trained word-embeddings (I explored both gensim and

[29] https://keras.io/

[30] https://github.com/fchollet/keras/tree/master/examples

fasttext pregenerated word-embeddings) . Only the number of dimensions and epochs had a large impact on the predictive or computational performance of the model.

The params used on the final model were:

label prefix ['__label__']
learning rate [0.1]
change the rate of updates for the learning rate [100]
size of word vectors [100]
size of the context window [5]
number of epochs [100]
minimal number of word occurrences [1]
number of negatives sampled [5]
max length of word ngram [1]
loss function [softmax]
min length of char ngram [0]
max length of char ngram [0]
number of threads [4]
sampling threshold [0.0001]
use of a pretrained word vectors for supervised learning [Yes]

I also tested the predictive performance of this final model using or not pregenerated bigrams integrated in the examples and final results were not significantly impacted (see Fig. 5).

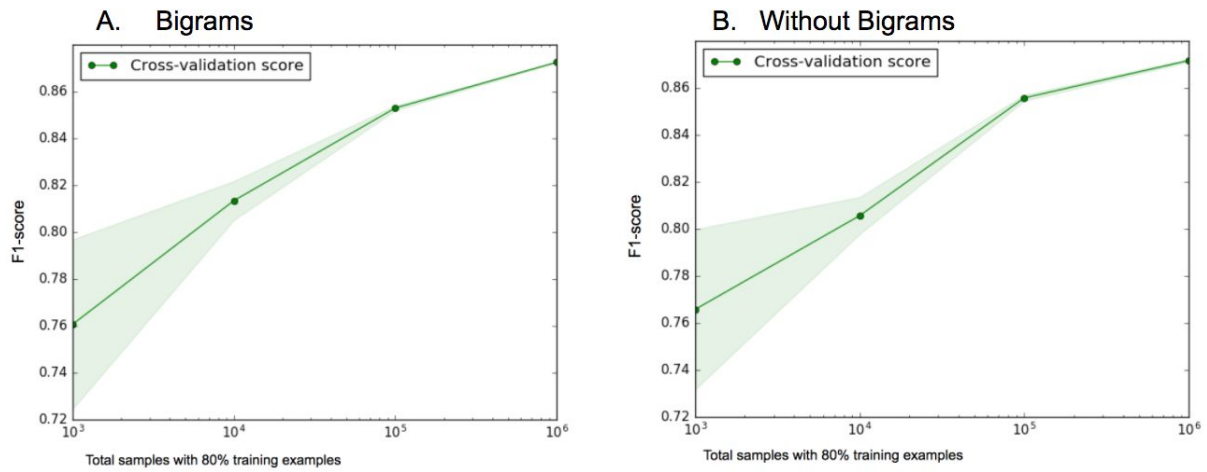


Fig 5. Learning curves with k-folds = 5 on FastText classifier model on text with or without bigrams

Using 10^6 samples (800000 training examples), the F1-score achieved was 0.87 on a validation set of 200.000 samples (see Table 1 and 2) . The Cohen's Kappa coefficient of agreement between the predicted and the true labels on the validation set was 0.74452 (regarded as substantial agreement).

Read 21M words
Number of words: 843368
Number of labels: 2
Progress: 100.0% words/sec/thread: 3144827 lr: 0.000000 loss: 0.079081 eta: 0h0m 14m
end
P@1: 0.87226
R@1: 0.87226

Table 1: Confusion Matrix on Validation Set (total size 10^6)

	True Negatives	**True Positives**
Predicted Positives	11522	85974

Predicted Negatives	88478	14026

Table 2: Summary Results on Validation Set (total size 10^6)

Class Label	precision	recall	f1-score	support
Eligible	0.88	0.86	0.87	100000
Not eligible	0.86	0.88	0.87	100000
avg / total	**0.87**	**0.87**	**0.87**	**200000**

The learning curve (see Fig. 6) shows training F1-score of the model for varying numbers of training samples from 800 to 800.000 . The model seems to underfit the data and although it increased performance with increasingly larger datasets it reached a top in a low-medium score. Adding more training data the F1-score increased from 0.72 to 0.87 but the curve converged with the score obtained in the training sample (0.89) to a maximum of 0.88 using the entire data set 4 x 10^6 samples, (see results detailed in Tables 1 and 2) which indicates that the estimator suffered more from a bias error than a variance error. The fact that the validation score converges with the training score and that the estimator does not benefit much from more training data denotes a bias error. The phenomena of not being able to increase performance with additional data has been reported to be overcomed with the use of deep learning models applied to complex problems[31] in contrast to a fast but thin architecture as FastText (and will be probed later). On the contrary, the model did not suffered from a variance error and therefore it can be concluded that it generalizes well to unseen data as we have obtained similar scores when manipulating input data: cross-validation was used to assess how the results of the model generalized to unseen data sets and obtained robust averaged validation results with a decreasing standard deviation over the k folds.

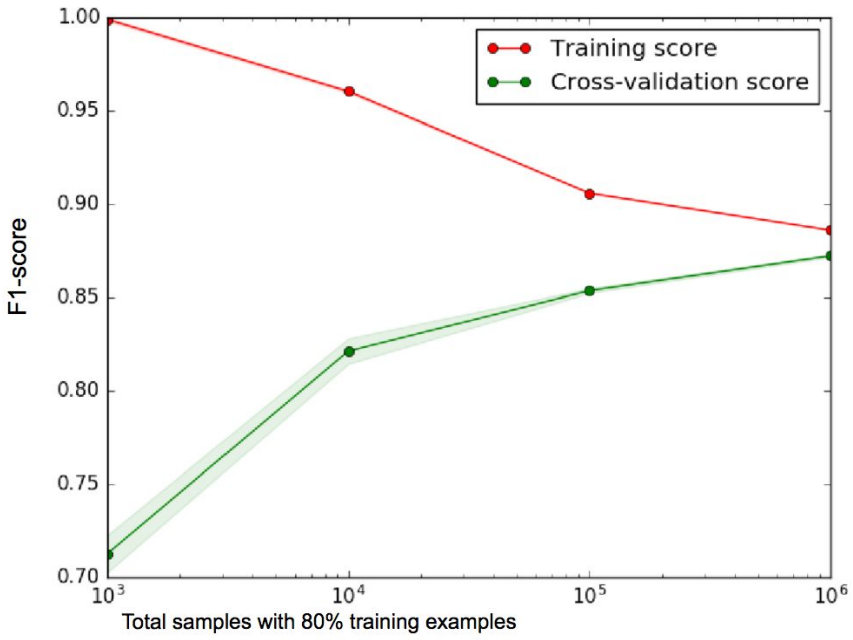


Fig 6. Learning curve with k-folds = 5 on FastText classifier model

Using 4.1 x 10^6 samples (3.257.179 training examples), bigrams and pre trained word embeddings, the F1-score achieved was 0.88 on a validation set of 814.295 samples (see Table 1 and 2) . The Cohen's Kappa coefficient of agreement between the predicted and the true labels on the validation set was 0.757908381109 (regarded as substantial agreement).

Read 88M words
Number of words: 3314921
Number of labels: 2

[31] https://www.slideshare.net/ExtractConf by Andrew Ng

Progress: 100.0% words/sec/thread: 2687871 lr: 0.000000 loss: 0.283239

Table 1: Confusion Matrix on Validation set (total size 4.1 x 10^6 samples)

	True Negatives	True Positives
Predicted Positives	44915	353496
Predicted Negatives	362232	53652

Table 2: Summary Results of Validation set

Class Label	precision	recall	f1-score	support
Eligible	0.89	0.87	0.88	407148
Not eligible	0.87	0.89	0.88	407147
avg / total	**0.88**	**0.88**	**0.88**	**814295**

Cohen's Kappa coefficient of agreement = 0.757908381109

CNN Classifier Model Results:

After doing several modifications on the hyper-parameters space the predictive performance (measured with the F1-score) of the classifier was optimized for size of word vectors = 100 (number of dimensions or embedding dimensions), epochs = 10 and batch size = 128 with all other values set to the params shown below. Results were similar using or not using pre-trained word-embeddings (I explored both gensim and fasttext pregenerated word-embeddings) . The number of epochs and batch size had a large impact on both the predictive and speed performance of the model. The long training needed (more than 4 hours) even with the use of GPU limited the exploration or greater number of epochs, but as shown in Fig.8, the last epochs did not significantly contributed to reduce the loss value, so that we can conclude that more than 10 epochs would not benefit the model. Regarding the batch size, that controls the frequency with which the gradients are computed, I explored sizes of 1, 10, 64, 128, and 512 and, as expected, the higher the more time efficient. Noisiness of the gradient estimate were reduced in mini-batch sizes with higher values. This can be explained because updating by one single sample is noisy if the sample is not a good representation of the whole data. We should consider a mini-batch with a size that is representative of the whole dataset. For values higher than 128, the predictive performance deteriorated in earlier epochs during training, so I choose 128. In fact, it has been reported[32] that the loss function landscape of deep neural networks is such that large-batch methods are almost invariably attracted to regions with sharp minima and that, unlike small batch methods, are unable to escape basins of these minimizers. So, when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize.

The params used on the final model were:

max number of words [20000]
max sequence length [1000]
size of word vectors [100]
loss function [categorical_crossentropy]
optimizer [rmsprop] using learning rate [0.001], rho[0.9], epsilon[1e-08] and decay[0.0]
number of epochs [10]
batch size [128]

The CNN learning curve (Fig. 7), similar as for FastText model, showed that it was capable to generalise and showed robustness where changes of the training data using cross-validation did not greatly affected the predictive results. Nonetheless it also showed a bias error, but on this case it achieved higher scores for both the training and the validation sets, converging to a max limit of 0.91, beyond of with adding more data seems to not benefit much more the model. One additional difference with FastText learning curve is that the CNN model needs, in comparison with FastText, much more data to learn. This is reflected by the fact that the model was underfitting and not properly learning for sample size 10^3 with a training score of only 0.72 while for FastText and sample size 10^3 the model was overfitting with a training score of 0.999. When adding more data to the CNN model the curves show a logarithmical

[32] https://arxiv.org/abs/1609.04836

increase in the predictive performance with a separation of the training and cross-validation curves for samples sizes 10^3 and 10^5 and finally converging on samples sizes 10^6 of magnitude order to a maximal score of 0.91. Beyond of this point adding more data does not benefit more.

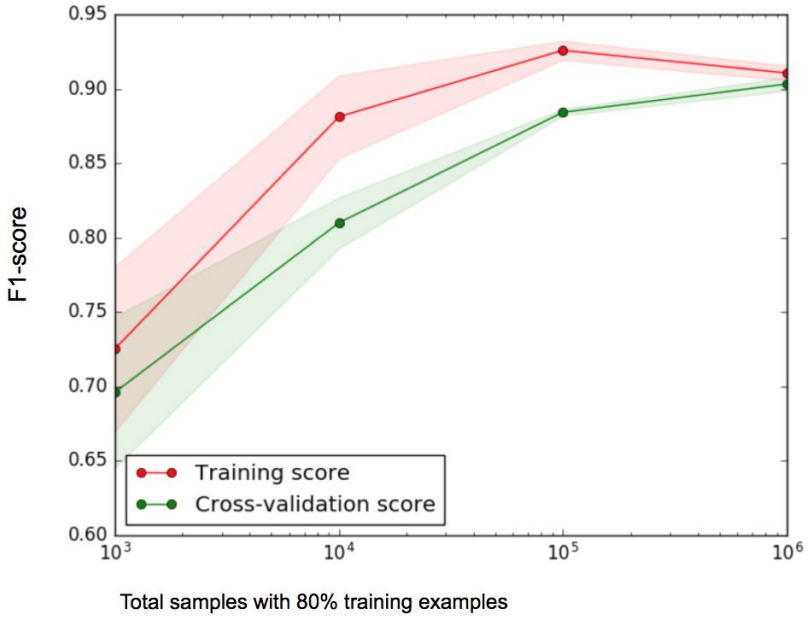


Fig 7. Learning curve with k-folds = 5 on CNN classifier model

Finally the model on the whole dataset using 3.257.179 training examples, bigrams and pre trained word embeddings, yielded an accuracy of 0.91 on the validation set comprised of 814.295 samples:

```
Generator length: 814295
3257216/3257179 [==============================] - 665s - loss: 0.2738 - acc: 0.8889 - val_loss: 0.2330 - val_acc: 0.9084
Epoch 2/10
3257216/3257179 [==============================] - 662s - loss: 0.2451 - acc: 0.9045 - val_loss: 0.2451 - val_acc: 0.9015
Epoch 3/10
3257216/3257179 [==============================] - 662s - loss: 0.2402 - acc: 0.9089 - val_loss: 0.2345 - val_acc: 0.9082
Epoch 4/10
3257216/3257179 [==============================] - 661s - loss: 0.2401 - acc: 0.9113 - val_loss: 0.2235 - val_acc: 0.9146
Epoch 5/10
3257216/3257179 [==============================] - 660s - loss: 0.2425 - acc: 0.9123 - val_loss: 0.2386 - val_acc: 0.9152
Epoch 6/10
3257216/3257179 [==============================] - 662s - loss: 0.2489 - acc: 0.9132 - val_loss: 0.2390 - val_acc: 0.9127
Epoch 7/10
3257216/3257179 [==============================] - 662s - loss: 0.2522 - acc: 0.9142 - val_loss: 0.3000 - val_acc: 0.9108
Epoch 8/10
3257216/3257179 [==============================] - 662s - loss: 0.2583 - acc: 0.9139 - val_loss: 0.2292 - val_acc: 0.9173
Epoch 9/10
3257216/3257179 [==============================] - 662s - loss: 0.2649 - acc: 0.9141 - val_loss: 0.2403 - val_acc: 0.9171
Epoch 10/10
3257216/3257179 [==============================] - 663s - loss: 0.2639 - acc: 0.9140 - val_loss: 0.2497 - val_acc: 0.9151
Bye
```

Fig. 8: Output during training of the CNN model on the whole dataset

The Cohen's Kappa coefficient of agreement between the predicted and the true labels on the validation set was 0.832219 (regarded as almost perfect agreement). This result implies that the model can be trusted.

Lastly, to assess its potential as a clinical decision support system, it would be relevant to assess its performance on a clinical practice simulation. With this purpose the two final models were further tested with unseen inputs consisting in a small set of short clinical statements that would be used in routine clinical practice and yielded very promising results (Cohen's Kappa coefficient of agreement was 0.92). Though the test size (13) is insufficient to prove it, it favours the hypothesis that such a model may be able to generalise to a different source of data, i.e. routine clinical practice notes - beyond clinical trial protocol eligibility criteria text which is the source used to build and validate this model.

As an example, below are some examples of statements correctly classified and that would require expertise oncology knowledge to judge them as cases being studied or not in available clinical trials (Yes, No):

'lapatinib to treat breast cancer with brain metastasis', → 'Yes'
'pertuzumab to treat breast cancer with brain metastasis', → 'No'
'CAR to treat lymphoma', → 'Yes'
'TCR to treat breast cancer' → 'No'

Justification

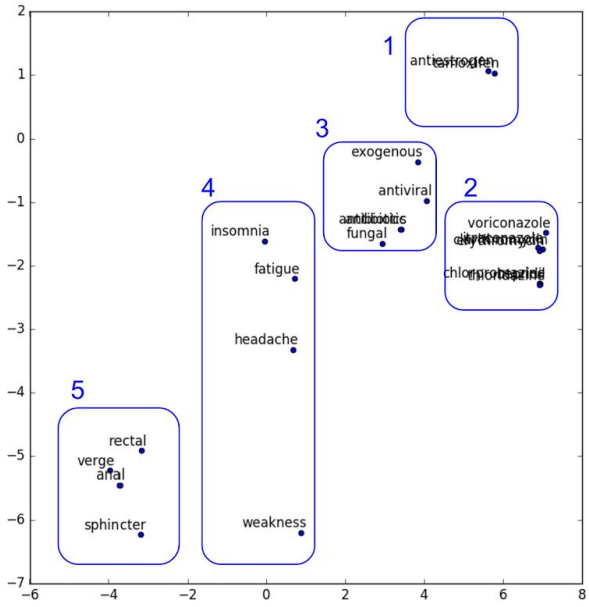
Both the results and the solution are significant enough to have solved the problem of predicting if short clinical statement extracted from eligibility criteria were considered eligible or not in the available corpus of clinical trials on cancer. The results achieved with the CNN classifier text model fit the expectations with a F1-score of 0.91 and an almost perfect agreement measured 0.83 with the Cohen's Kappa.

V. Conclusion

Free-Form Visualization

One interesting part of this project are word embeddings: On this project adding pre-trained word-embeddings to the classifiers did not altered the classification results, however the embeddings by themselves were interesting enough to be displayed and discussed using word space visualizations:

1) **T-SNE representation of a subset of words:** Using the vectorial word representation obtained with FastText, where each word is located in a 100 dimensional space, we will visualize a subset of the words in a reduced space. For that we will use t-Distributed Stochastic Neighbor Embedding (t-SNE) from van der Mateen and Hinton (2008)[33]. t-SNE is a dimensionality reduction method that is particularly well suited for the visualization of high-dimensional datasets. The idea is to compute the probability distribution of pairs of high dimensional objects in such a way that similar objects have high probability of being clustered together and dissimilar objects have low probability of being clustered together. Afterwards, the algorithm projects these probabilities in the low dimensional space and optimizes the distance with respect to the object's location in that space. Therefore, at the end of the optimization, similar objects are close in the low dimensional space. We define the words that we are going to analyze out of the complete corpus (as this is not possible to visualize all 26.893 words), and get the word vectors of these words. The Fig 9 shows two aspects: On the one hand, the words are grouped by semantic similarities and on the other hand the groups seems to follow a spatial distribution in different regions through a diagonal direction from intrinsic/internal to extrinsic/external concepts with respect to the human body (body organs [G5] → body symptoms [G4] → infections, cancer (not shown) [G3] → treatments [G1,G2])



Group 1: Hormone therapy ('tamoxifen', 'antiestrogen')

Group 2: Antibiotics & Antimycotics ('itraconazole', 'clarithromycin', 'erythromycin', 'voriconazole')

Group 3: Anti-infective concepts ('antiviral', 'antibiotics', 'antimycotics','fungal', 'exogenous')

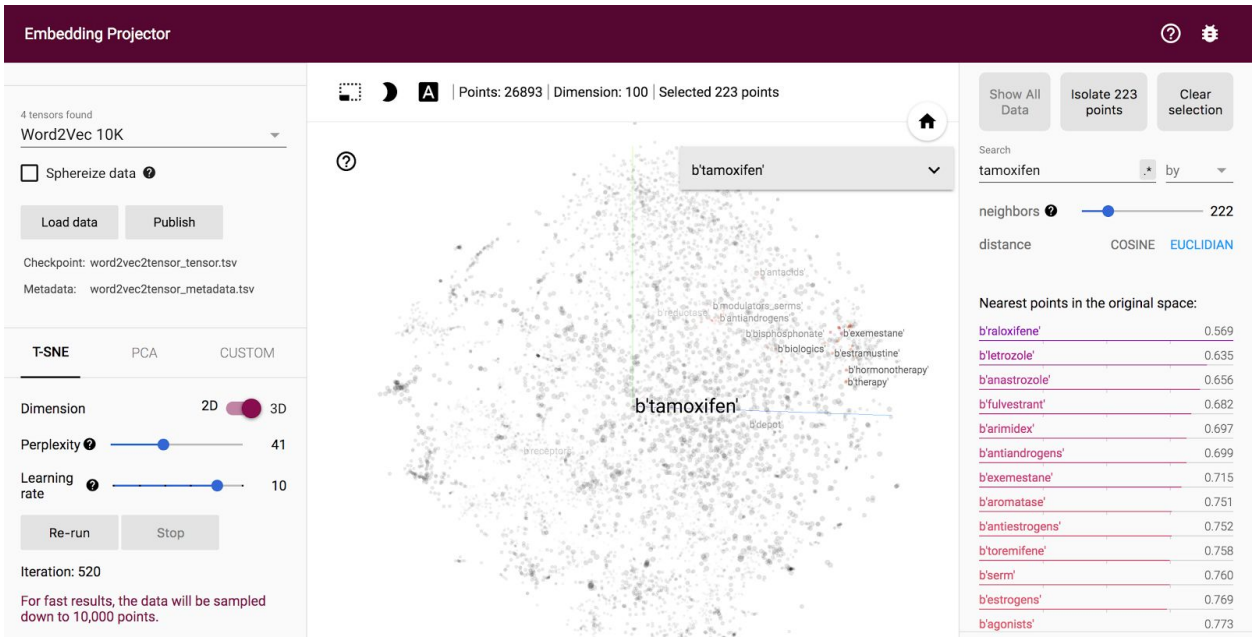
Group 4: Clinical symptoms ('fatigue', 'insomnia', 'headache', 'weakness')

Group 5: Anatomic locations ('rectal', 'anal', 'sphincter', 'verge')

[33] http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

Fig 9: Words representation in a reduced space using T-SNE.

2) **Interactive visualization of the whole set of words**: TensorBoard[34] has a built-in visualizer, called the Embedding Projector, for interactive visualization and analysis of high-dimensional data like embeddings. The Word2Vec embeddings obtained with Gensim were formated to Tensorflow 2D tensor and metadata formats for Embedding Visualization. The first screen capture from the Embedding Projector shows that the model succesfully extracted hormonal therapies for breast cancer ('raloxifen', 'anastrozole', 'letrozole', 'fulvestrant',..) as the nearest points to "Tamoxifen" in the original space. Similarly, in second screen capture, the 87 nearest points to ultrasound were all related to explorations, mainly in imaging.



Reflection

This project trained, validated and compared two text classifier models - FastText vs Deep Learning using a 1-D Convolutional Neural Network with pre-trained word-embeddings - on the corpus of cancer clinical trial protocols published in clinicaltrial.gov. The models classifies short free-text sentences (describing clinical information like medical history, concomitant medication, type and features of tumor, cancer therapy etc.) as eligible or not eligible criterion to volunteer in clinical trials.

[34] https://www.tensorflow.org/versions/master/how_tos/embedding_viz/

Both models were evaluated both using cross-validation with k-folds = 5 on incremental sample sizes (1K, 10K, 100K, 1M samples) and on the largest available balanced set (using undersampling) with 4.01 M labeled samples from a total of 6 M labeled samples. Overall the models showed robustness and capacity to generalize.

The highest predictive performance was achieved with the 1D-CNN using a balanced sample of the whole dataset. The results fit the expectations with a F1-score of 0.91 and an almost perfect agreement measured 0.83 with the Cohen's Kappa.

Finally this CNN model is also preliminarily tested in an independent source of clinical data opening the avenue to potentially use it -taking into account pending improvements- in a clinical support system for oncologists using their clinical notes.

Along the way, word-embedding models achieved high quality clusters as well as demonstrated the capability of semantic reasoning, being able to identify the equivalent treatments for a type of tumor by analogy with the drugs used to treat others tumors. These interesting reasoning qualities would merit to be studied in a separate project using this dataset.

Challenging aspects of the project, that requires cross-knowledge between medical and IT domain, were 1) preprocessing and dataset building to obtain an exploitable and classified corpus large enough to explore NNs on the medical domain and 2) the design of the problem to be solved, so that it could be answered with available data.

The knowledge I acquired during the ML nanodegree helped me mainly on the model validation and evaluation as well as in the understanding of PCA and T-SNE techniques to visualize word-embeddings, but as it did not covered NLP, nor NNs and Deep Learning techniques I dedicated most of the time to learn them and to choose and implement those technologies.

Improvements

The fact that the CNN model outperformed FastText may be explained by the higher depth of the CNN model. Consequently next improvement for this problem would be to explore if using more complex and deeper learning techniques like Recurrent Neural Networks (RNN) and Long Short Term Memory Network (LSTM) to build the model will benefit more. In fact those techniques are thought to fit more naturally NLP problems compared with CNN models, as the last was initially designed to solve image problems.[35]

But, before moving to more involved models, more effort should be directed to experiment with variations in the CNN architecture itself: testing different layer compositions, altering the dimensionality of the output (number of kernels) on each layer, altering the spatial extension of each filter, modifying the max pooling after each layer.

The present project serves as a proof of concept that clinical trial protocols on cancer, which are freely available, can be meaningful exploited applying latest NLP AI techniques, opening the potential to explore more ambitious goals applying additional efforts required to build the appropriate dataset:

- The most immediate improvement is to include in the model the effectiveness of CTs interventions so that we can not only predict if a patient case has been studied or not, but also if the proposed treatment is expected to be effective or not based on CTs. The problem would still be a classification problem where the labels would be 1) effective and studied, 2) potentially effective but not studied, 3) not effective and studied, 4) potentially not effective and not studied. The main effort on this case is in the preprocessing to include the effectiveness on the label for each criteria. As only a subset of CTs (n= 5754, 11%) have the results reported on clinicaltrials.gov, it means that for this goal, it would be necessary to augment data from other sources like pubmed.
- The next experiment would be to try predict which cancer treatments could be given to a patient based on CTs evidence. On this case, the sentences in the input would include the effectiveness and the labels for each criteria would be the treatment itself. The main task on this case lay as well in the preprocessing phase to label treatments and use the appropriate category for each. I attempted this on the project using both MetaMap[36] and LexEVS[37] APIs to tag treatments. For this purpose I explored using a webservice I implemented that retrieves the hierarchical classification (i.e family of drugs) of recognised terms using the latest version of the cancer ontology NCI Thesaurus (see webservice here[38]). After testing this approach, only 72% of interventions were correctly recognised using both MetaMap and LexEVS (failing those that included long sentences and those with the newest treatments not yet included on dictionaries), and I concluded that more preprocessing was needed in order to increase precision and recall and effectively build the dataset needed.

[35] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

[36] https://metamap.nlm.nih.gov/

[37] https://wiki.nci.nih.gov/display/LexEVS/LexEVS

[38] http://nlp.medbravo.org/c.groovy?concept=Brentuximab%20vedotin