NLP/Representation learning

Paul-Louis Delacour, Mathieu Chevalley, Tristan Meynier, Daniel Garellick

April 9, 2020

1 Task 1: Predict readmission

The dataset is unbalanced with $\sim 60\%$ of readmitted and $\sim 40\%$ of non-readmitted data. It contains various missing values and undesirable characters. NLP and feature extraction are necessary before running any ML model.

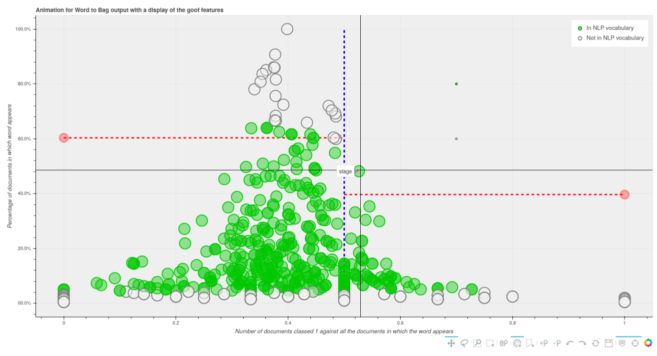
1.1 Prepocessing

1.1.1 NLP for textual data

We used NLP on `diag_1_desc`, `diag_2_desc` and `diag_3_desc`. One first option was to use NLP on the three corpus separately and then combine the score vectors. Alternatively one could initially combine the three corpus together and then run the algorithm. We implemented both methods and the second one performed better, particularly with Bag of Words using n-grams. This is probably because n-grams is able to extract relations between diagnostics for a fixed patient which is not possible when analysing `diag_1_desc`, `diag_2_desc` and `diag_3_desc` separately. Hence in what follows we only consider the three corpus combined.

Our first NLP trial aimed to be simple: we used a simple vocabulary and a simple implementation of Bag of Words. We build the simple vocabulary by organizing, stemming, filtering stop-words, punctuation and non alphabetical characters and finally filtering too short or too frequent words. Also for Bag of Words the score of a word of the vocabulary is just 1 if the word appears in the document and 0 otherwise.
This first implementation performed modestly on the ML models with F1 and ROC scores around $\sim 60\%$ (see table of results). One bottleneck could be the dimension and the sparsity of the vector scores which makes it difficult for the ML models to extract the adequate features.

With this is mind, we tried to extract a better vocabulary by taking the most relevant words. The goal being to reduce the dimension of the final score matrices. Concretely, we developed a visualization tool displaying all the words as points in a 2D dimensional space as shown below. Points colored green correspond to words in the vocabulary.



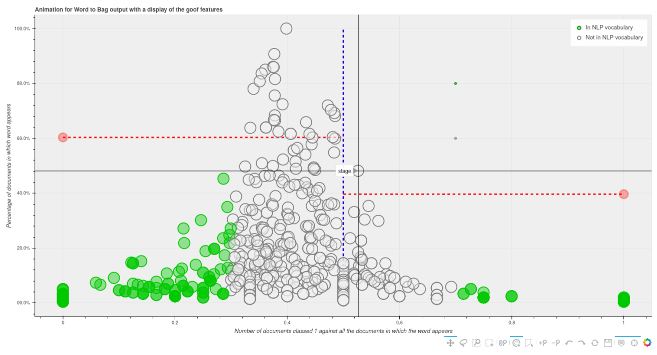


Figure 1: Visualization used to go from a simple vocabulary (up) to a better vocabulary (down)

Explanation of Fig 1: The horizontal axis shows the proportion of readmitted documents a word appears in against the number of all the documents it occurs in. This is a number between 0 and 1, where 1 corresponds to a word only in readmitted documents. The vertical axis is the frequency of appearance of a word in any document. A perfect word is in all the non-readmitted documents only (left red point at $(0, 60\%)$) or in all readmitted documents only (right red point at $(1, 40\%)$).

We started from the original simple vocabulary (upper plot) and removed non-meaningful words appearing approximately half of the times in both types of documents (center of the plot). The goal being to favor words close to one of the red points. In both plots we see the cut-off in low and high frequencies of appearance as well as the unbalancedness of the data sets. Also, placing the cursor over a point generates a display of the corresponding word. This is useful to manually visualize the relevant words and compare them in both the test and train data sets.

With the new vocabulary we were able to reduce the dimensionality of the score matrix considerably. However the TSNE of the output scores (see subsection 1.1.2) before and after feature extraction shows that better words alone is not sufficient to make the data easily separable. To improve further the predictions we needed a more sophisticated Bag of Words. It should be able to not only score the importance of a word alone but also to consider its close neighborhood (i.e relations between words in the document). Toward this, we implemented BOW with TFIDF scores and n-grams. With this we achieved better scores on most of the models (see subsection 1.3).

In what follows we summarize the results in two parts: Simple NLP = {simple vocabulary with simple BOW} and Complex NLP = {complex vocabulary with complex BOW}.

1.1.2 Object feature extraction

The remaining features are of two types : **integer** (int64) and **object** value. Since the integers are already encoded as numbers, we didn't change them. For the object values, we convert them into categorical values by building a mapping on object to the corresponding integer. For the `NaN` values, we decided to map it to another category since there is no point in using an already existing one and the corresponding value could be of a new type.
Combining both NLP and this feature extraction we obtain a new feature vector for each sample. Using the Simple NLP we ends up with features of size 481, and using the Complex NLP the features are of size 442.

1.2 Plotting the extracted features

We used TSNE algorithm to reduce the number of dimensions from 442 dimensions (481 for simple NLP) to 2. The 2D plots of the datasets are shown in Figure 2 . From this figure, we see that the 2 classes are not really separated which indicates, that separating them is a hard task.

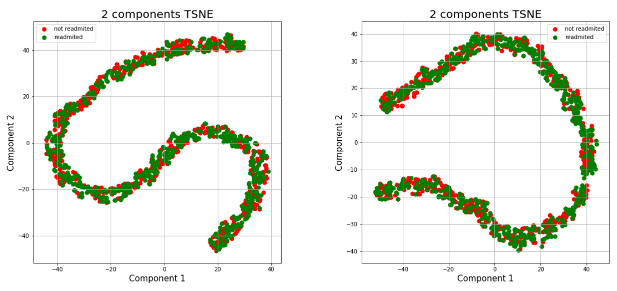


Figure 2: Visualization of samples after feature extraction using *simple_NLP* (left), and *complex_NLP* (right)

1.3 Models

Method	F1 score	ROC
SVM		
-Simple	59.15 %	56.82 %
-Complex	**60.12 %**	**59.63 %**
Neural Net		
-Simple	42.27%	**57.58%**
-Complex	**43.71%**	57.42%
Adaboost		
-Simple	43.89%	59.68%
-Complex	**44.92%**	**60.22%**
Xgboost		
-Simple	65.62%	63.04%
-Complex	**65.73%**	**63.19%**
LightGBM		
-Simple	63.01%	70.24%
-Complex	**64.17%**	**70.53%**
Random Forest		
-Simple	63.39%	69.76%
-Complex	**64.15%**	**70.60%**

Table 1: Comparison of the results on all methods for simple and complex NLP

The models achieve modest results in overall. Some of the poorest models like SVM struggle in dealing with the sparsity and high dimensionality of the vector scores as well as the unbalancedness of the data sets. As a result, it was common that a model predicts every documents as non-readmitted to maximize correct predictions.
To avoid such situation we tested various models. In all cases we managed to obtain a reasonable confusion matrix where the model still correctly predicts readmitted documents. This was even more frequent when using a more sophisticated vocabulary and BOW (see bold results in table above). This shows our ability to (still) learn and correctly predict some documents when using a relevant vocabulary as well as n-grams to extract more complex relations between words.

The better scores for tree-based model versus SVM can be explained by their capacity to better deal with data of various scales (categorical and TFIDF scores). The final confusion matrix for Random Forest (our best model) is: $\begin{bmatrix} 1041 & 166 \\ 502 & 291 \end{bmatrix}$

Cross-validation was used to choose the correct parameters for each model.

1.4 Conclusion and reproducibilty

From the prediction results and visualization, it is a hard task to identify words or even relations of words helping to separate samples of the 2 classes. For example on the visualization animation, words allowing us to make distinction in the training set are often not the same as in the testing set.

From our perspective, we think that we could reach better results with more medical expertise. Quickly distinguishing names of diseases, treatments or drugs could have help us to manually (through the visualization animation) recognize trends and relations between diagnostics. An other idea for better prediction could be to use GAN to produce more readmitted documents and so to limit the unbalancedness of the data set.

You can find the code related to our results in the folder `task1`. It contains a jupyter notebook `main.ipynb` with all models and plots.

2 WIP Task 2: What has been published about medical care?

2.1 Done Work

In this task, we are aiming at extracting information on medical care for diseases similar to CoVID-19. To achieve that, we use NLP method, more precisely text embedding. We use the Spacy package and ScispaCy pretrained word vectors to vectorize the abstracts. ScispaCy is a modified version of Spacy tailored for biomedical texts. With those vectors, we compute the similarity between a query and all the abstracts. Our query is made of key-words extracted from Kaggle task headline questions. We then take the 1000 most similar abstracts.

We use this subsets of relevant papers to answer the subtasks of the Kaggle task. To do so, we apply the same embedding that we used for the abstracts, but this times on all the sentences of the actual content of the selected articles. We use this embedding to query the ten most similar sentences to a query sentence. These query sentences simply are the subquestions of the Kaggle task. We then merge all the results in a summary json file.

The results are of varying quality. For the abstract similarity, most abstracts seems to answer the query. However, to improve the results, we have to add a lot of key-words in the query, otherwise the results are not necessarily related to Coronavirus-like diseases. The problem is the same with the sentence query: it happens that the results are not related to the subject of interest. Also, the results are often very close to the query sentences (i.e are lot of words a the same) which does not offer additional information. This could be circumvented by using larger parts of the texts to be vectorized, for example full paragraphs.

2.2 Future Work

Currently we have implemented a pre-processing pipeline. We begin by making everything lower text and splitting the text into senteces and then into tokens. Punctuation and stopwords are removed to make the information contained in the text more valuable. It is worth noting that some words contain punctuation such as dots "." for abbreviations or "-". The first is less of an issue as we are able to resolve abbreviations (to varying accuracy). We may also remove the "-" from stopwords' list if it impacts the results. We further implement lemmatisation and stemming which will be turned on and off to asses their impact. This has currently been done using NLTK library but may be switched to SciPy for consistency.

As another approach we have used the UMLS linker provided in ScispaCy package to extract the CUI and TUI of each medical "entity". The CUI is a unique concept ID and the TUI is the semantic type for example "Clinical Drug" or "Disease or Syndrome". From this we aim to create a "Bag of CUI" with more discriminatory power to achieve better results. It has proven cumbersome to extract the aforementioned identifiers not least because the UMLS linker is an alpha feature. Another obstacle is that Covid-19 and related words have not yet been included in the vocabulary provided by ScispaCy which is the one linked to UMLS. This may imply that the method will have to be used once the articles have been pre-filtered in terms of relevancy to Covid-19.