Creation Of A Pipeline For Medical Sentences Classification

Ludovic Guyader[1], Paul Oriat[1]

[1]URF Sciences, Université Paris-Saclay

Abstract

In this project, we will try to create a pipeline for the classification of medical sentences along five categories : BACKGROUND, OBJECTIVE, METHODS, CONCLUSION and RESULTS. To do so, we will use a dataset of 200k texts composed of around 11 sentences each. We will also use a sub-dataset of 20k texts to experiment and test with reduced calculation time. We will start by engineering some features to provide more input to our classifiers, then we will use word embedding to prepare the sentences for the classifiers. Finally, we will test and fit different classifiers to better our predictions performances.

Part I

Pre-processing

Baseline model

We have opted for a word embeddings model based on the Word2Vec algorithm. This algorithm allows for representing the vectorial position of words from a large corpus of text. Thanks to the Word2Vec() function in the Gensim library, we can customize several parameters to achieve the best performance for our model.

Training data

The training data for the baseline model consists of the 'sentences'. We have decided to test 3 options for preprocessing the textual data.

Option 1: Preservation of raw sentences

No preprocessing is required in this option.
Example :
To investigate the efficacity of 6 weeks of daily low-dose oral prednisolone in improving pain , mobility , and systemic low-grade inflammation in the short term and whether the effect would be sustained at 12 weeks in older adults with moderate to severe knee osteoarthritis (OA) .

Option 2: Tokenization of sentences

In this text preprocessing approach, we focused on segmenting each sentence of the corpus into meaningful lexical units, tokenization. By using functions from the SciSpacy library, we are able to ensure consistency with the medical domain.
Example:
['investigate', 'efficacy', '6', 'weeks', 'daily', 'low-dose', 'oral', 'prednisolone', 'improving', 'pain', 'mobility', 'systemic', 'low-grade', 'inflammation', 'short', 'term', 'effect', 'sustained', '12', 'weeks', 'older', 'adults', 'moderate', 'severe', 'knee', 'osteoarthritis', '(', 'OA', ')', '.']

Option 3: Lemmatization and preservation of only entities

This approach to text preprocessing focuses on two essential aspects:

- Lemmatization, which normalizes words by reducing them to their base form
- Exclusive preservation of Named Entity Recognition (NER) in each sentence

For the same reasons as tokenization, we leveraged the advanced features of SciSpacy.

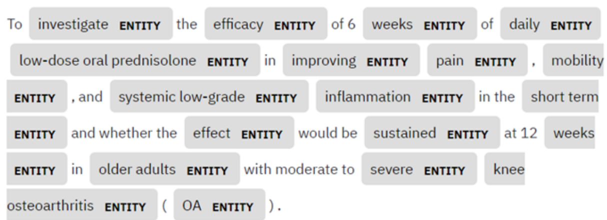

Figure 1: *Example NER.*

Example : *['investigate', 'efficacy', 'week', 'daily', 'low-dose oral prednisolone', 'improve', 'pain', 'mobility', 'systemic low-grade', 'inflammation', 'short term', 'effect', 'sustained', 'week', 'old adult', 'severe', 'knee osteoarthritis', 'oa']*

Option	Mean accuracy
1	0.487
2	0.335
3	0.418

Table 1: *Test with RandomForest classifier*

Results

We clearly see that Model 1 outperforms the others. Our hypothesis for this surprising result is that Models 2 and 3 lose information during their preprocessing steps, such as:

- Verb tenses
- Punctuation and operators

Hyperparameters

After selecting the training data, we focused on choosing the hyperparameters.

Hyperparamètres	**Objectif**
sg	The algorithm type used for training: • 0 for CBOW • 1 for Skip-gram
vector_size	The dimension of the word vectors generated by the model
min_count	The minimum number of occurrences required for a word to be included in the model's vocabulary
epochs	Number of iterations in the corpus
window	Maximum distance between the current and predicted word within a sentence

Table 2: *Description of Hyperparameters*

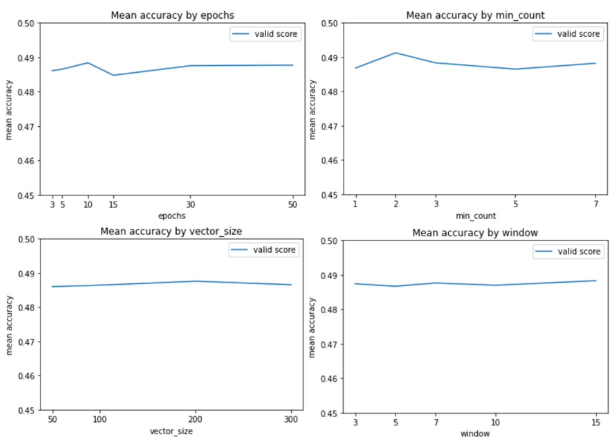

Figure 2: *Mean accuracy as a function of different hyperparameters*

The performance gap among the different choices of these hyperparameters is relatively small. Thus, we can prioritize the less computationally intensive options.

Hyperparamètres	**Final choices**
sg	0 (because that was the project's instruction)
vector_size	200 (best rate)
min_count	2 (best rate)
epochs	10 (best rate))
window	3 (best rate/speed)

Table 3: *Hyperparameters choice*

Model with biomedical word embeddings

We found 2 pre-trained Word2Vec word embedding trained on medical data. The first one was 200 dimensional and was trained on the 200k dataset, so the one we are using for this project (`https://github.com/ncbi-nlp/BioSentVec`, *BioWordVec vector 13GB (200dim, trained on PubMed+MIMIC-III, word2vec bin format)*); And the second one was only 100 dimensions and trained on data from PubMed, but not directly on the 200k dataset (`https://huggingface.co/garyw/clinical-embeddings-100d-w2v-cr`). We decided to go for the 100 dimensions word embedding cause it was taking way less space (300Mo vs 13Go) and made our future tests faster.

Feature engineering

Label encoding

Firstly, label encoding is essential for the machine learning model to function properly. Here's how we encoded our labels using the LabelEncoder() function from SKlearn.

Before encoding	**After encoding**
'BACKGROUND'	0
'CONCLUSIONS'	1
'METHODS'	2
'OBJECTIVE'	3
'RESULTS'	4

Table 4: *Encoding Before and After*

Encoding the feature 'sentence'

For encoding the feature 'sentence', we took the average of the coordinates of each word using our Word2Vec model.
Dimension of the dataframe : (2211861, n) (where n is the value of the hyperparameter 'vector_size' (200 for the baseline model)).

Principal Composant analysis (PCA)

Part b adds a feature with n dimensions, which, as a reminder, corresponds to the average of the vector locations of each word in the sentence. To prevent our classifier from being too computationally intensive, we preferred to perform PCA.
Dimension of the dataframe : (2211861, 10)

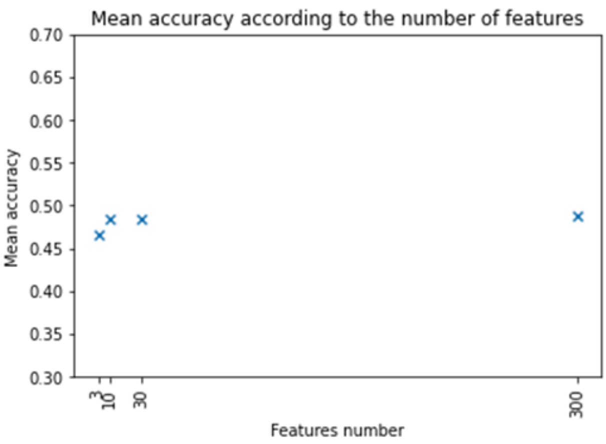

Figure 3: *Mean accuracy as a function of dimensions.*

Adding features

To improve the performance of our models, we explored various possibilities after analyzing the composition of our data.

Good possibilities :
Here are the possibilities that enable our classifier to identify highlighted classes or remove classes sidelined by the feature:

- Word count in a sentence (1st column)
- Proportion of numbers in a sentence (2nd column)
- Quantity of specific words: usage of personal pronouns, tense markers, punctuation, operators, etc. (others columns)

	Word Count	Number Proportion	http	NCT	.gov	%	(	ed	;	=	vs	result	objective
BACKGROUND	21	0,013	0,014	0,062	0,062	0,015	0,308	0,604	0,022	0,002	0,003	0,028	0,018
CONCLUSIONS	22	0,011	0,003	0,022	0,019	0,021	0,064	0,653	0,026	0,002	0,002	0,084	0,004
METHODS	25	0,059	0,000	0,008	0,006	0,041	0,357	0,871	0,048	0,060	0,005	0,007	0,006
OBJECTIVE	26	0,007	0,000	0,003	0,002	0,021	0,460	0,622	0,015	0,003	0,014	0,024	0,035
RESULTS	30	0,086	0,000	0,000	0,000	0,297	0,562	0,682	0,146	0,240	0,081	0,026	0,002

= Highlights classes   = Removes classes

Figure 4: *Representation of the new features by label*

Bad possibilities :
Here are the possibilities we discarded because we believe the classifier won't effectively highlight or remove classes:

- Proportion of common nouns, proper nouns, or adjectives
- Document number for a label (risk of overfitting)
- Label before and after each sentence -> Result: very effective but cheating

	Proper noun proportion	Common noun proportion	Adjective proportion
BACKGROUND	0,094	0,060	0,060
CONCLUSIONS	0,052	0,075	0,075
METHODS	0,073	0,073	0,073
OBJECTIVE	0,053	0,064	0,064
RESULTS	0,079	0,064	0,064

Figure 5: *Representation of the features discarded by label*

Long processing and no class differentiation.

	Mean accuracy	Mean f1_score
classic (just sentence feature)	0,483	0,34
num_doc	0,417	0,33
before and after	0,847	0,82

Figure 6: *Analysis of Scores (on Model 1)*

We then add the good features to the 10 features corresponding to 'sentence'.
Dimension of the dataframe : (2211861, 28)

Results

With the addition of these 18 features, although our model overfits, it remains more performant.

	Train accuracy	Valid accuracy
Without features	0.531	0.483
With features	0.788	0.586

Table 5: *Train and Valid accuracy*

Conclusion

We observe that the baseline model performs poorly. To address this, we turned to a substitute model. Instead of using the Word2Vec algorithm, we employed the CountVectorizer() function from sklearn. After preprocessing the sentences in this function, we followed the same process as with Word2Vec():

- PCA, but this time with the hyperparameter n_components=30
- Then adding our 18 features

We can observe that the model generalizes better using the classic CBOW.

	Word2vec()		CountVectorizer()	
	without features	with features	without features	with features
Mean acc.	0.483	0.586	0.679	0.720
Mean F1	0.34	0.50	0.59	0.65

Table 6: *Comparative table of the 2 baseline models (with RandomForest classifier)*

Part II

Classifiers

All the models presented below were trained and fit on the 200k dataset.

SKlearn models

We decided to start with SKLearn models as they are the only ones we are familiar with. We used both Random Forest et SVC classifiers for our multi-class classification since they are reputed for these tasks. We used both word2vec models and the features we created.

Random Forest

We use the data passed in the homemade and the pre-trained Word2Vec as input for the model, as well as the other hand-crafted features. The data used is the 20k texts dataset to lower the calculation time. We make the assuption that the difference between the models will be conservated between the 20k and 200k dataset. Even though the dataset is reduced, we quickly realised the importance of computation time. After the first attempt to fit the RandomForestClassifier, we had to check how to accelerate the process. We first thought the calculation could be done with our GPU, but SKlearn does not support that, but supports multi-threading CPU computation. That is done by setting the hyperparameter *n_jobs = -1*, helping us taking full advantage of our CPU performances. The calculations were made on a 6-cores AMD CPU, thus lowering the computation time considerably. We started by projecting the differences between the different models (homemade W2V & internet W2V) with and without PCA.

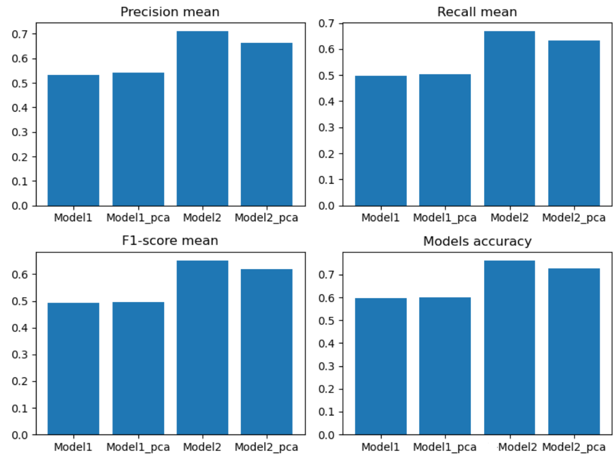

Figure 7: *RandomForest models and PCA comparaison*

We see as it was expected that the model 2 performs better than model 1 by roughly 30%, and that in both models, the pca only slightly reduces the performances. In addition to that, we measured the time elapsed for each of the model 2 measures : Model2 took *1158s* whereas Model2_pca only took *291s* so almost a factor 4.
So taking in account these last two elements, we will allow ourselves to use PCA in future analysis to lower computation time if needed.

Optimization We used a RandomizedSearchCV to optimize our randomforest on the following parameters : *n_estimators*, *min_samples_split*, *min_samples_leaf* and *bootsrap*. We didn't touch *max_depth* because we wanted to let the *min_samples_split* act freely.
Best parameters : *n_estimators=500*, *max_features=5*, *min_samples_split=2*, *min_samples_leaf=1*, *bootstrap=False*.
The results weren't necessarely much better than without optimization, our parameters grid may not be well fit for our problem, but unfortunately we didn't try another grid. We decided to take these best parameters and evaluate them when we change the *max_depth*, we got those results :

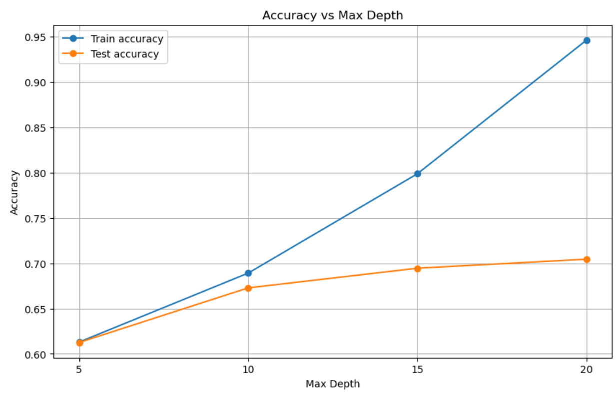

Figure 8: *Accuracy as a function of max_depth on optimized RF*

We can see that there is a lot of overfitting when the *max_depth* goes up, but the test accuracy still goes up. This is a phenomenon that we do not understand as we were always told that overfitting is bad for the model. In conclusion, we think that taking a *max_depth* of approximatively 14 is a good compromise between not overfitting and having a good test accuracy. Here are the metrics for the final Random-Forest :

Class	Precision	Recall	F1-score	Support
0	0.67	0.24	0.35	2663
1	0.51	0.71	0.59	4426
2	0.75	0.83	0.79	9751
3	0.53	0.29	0.37	2377
4	0.78	0.79	0.78	10276

Table 7: *Classification Report for the best fit RF*

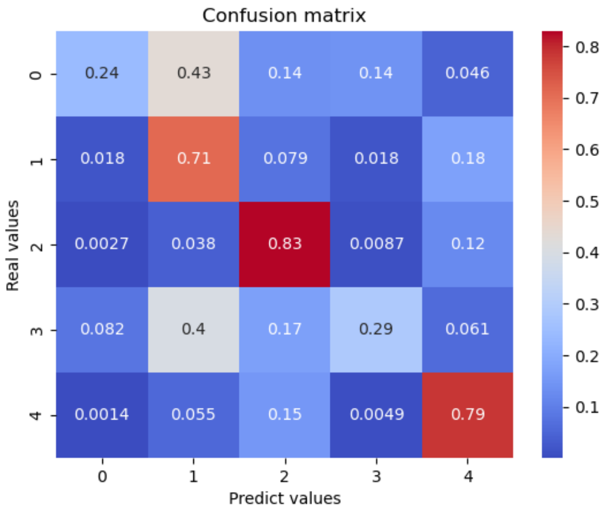

Figure 9: *Confusion matrix for the optimized RF*

Additionnaly, we tried with *class_weight = 'balanced'* :

Class	Precision	Recall	F1-score	Support
0	0.54	0.36	0.44	2663
1	0.50	0.72	0.59	4426
2	0.79	0.76	0.78	9751
3	0.40	0.51	0.45	2377
4	0.83	0.72	0.77	10276

Table 8: *Confusion matrix for the optimized RF, class_weight balanced*

In conclusion, we think the Random Forest is a decent model, it gives good performances. Adjusting the *class_weight* is interesting, depending on our will to trade precision for recall on the under-represented classes.

SVC

We tried to train a SVC classifier on our 20k data with the same protocol than for the Random Forest (projecting the differences between the different models (homemade W2V & internet W2V) with and without PCA). To ensure we would be optimizing our ressources, we decided to use a BaggingClassifier, that permits multi-threading on SVC (by splitting the dataset into subset and making the calculations on each subset with a core). The execution took 5 full hours, and the results were very well below our expectations, so we decided not to further explore nor optimize this classifier.

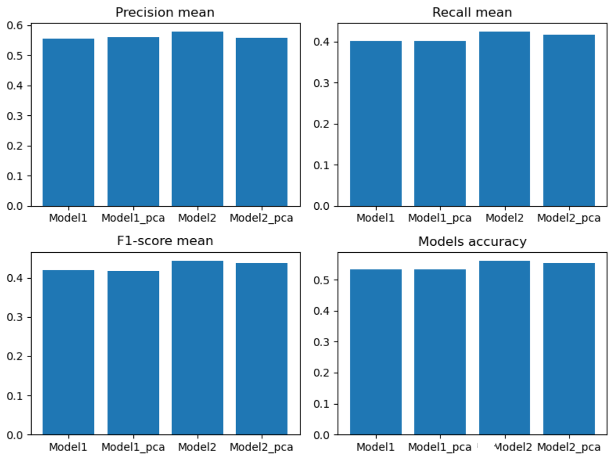

Figure 10: *SVC models and PCA comparaison*

DeepLearning models

We explored two models, that we deemed accessible to us knowing we have no background in DeepL models. Those models are the Recurrent Neural Network (RNN) and the Convolutionnal Neural Network (CNN). To use these models, we used chatGPT to understand some parameters and the code, online figures to try to understand the architectures of the layers, and some papers[1] to understand which combinations of hyperparameters were commonly used in biology text classification.
We trained the models on an AMD GPU config, so we couldn't use Cuda (only available for NVidia GPUs) and optimize calculation time on TensorFlow, so TensorFlow worked with the CPU. There seems to be a way arund it, but it didn't seem so trustworthy and easy to apply.
In this part, we didn't use our additionnal features as we were told Neural Network only took the text (word2vec'd) as an input.

[1] `https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1044-0`

Recurrent Neural Network (RNN)

We used an architecture with a Dense input layer with 64 filters, a unique inside layer with the ReLU activation function and 32 filters and finally an output layer returning 5 dimensions as we have 5 different classes and a softmax activation function. All those have a 0.5 dropout inbetween. We compiled the model with the sparse_categorical_crossentropy loss function and the Adam optimizer (advised by chatGPT and said papers) then we fitted the model on 10 epochs. We tested the RNN on both our models *(RNN1 on model 1 and RNN2 on model2)* :

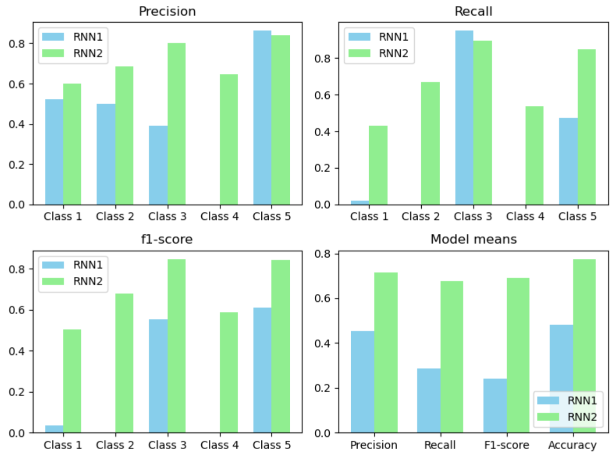

Figure 11: *Metrics comparaison of 2 models of the RNN*

There is clearly a problem between our homemade word2vec and the RNN. The lack of knowledge we have conecerning the RNN unallows us to further explain the poor performances of RNN1. That being said, we can observe that CNN2 is very interesting and proposes very good results. These results are similar to that of the RandomForest with the exception that we obtain a way better recall on under-represented classes than we do with the Random Forest : on class 1 we see a 79% increase in recall, and a 90% increase on class 4.
We conclude that this model is better than the RandomForest and provides better support for unbalanced dataset.

Convolutional Neural Network (CNN)

We are using the same activation functions, optimizer and loss function as the RNN. Here are the results on both models :

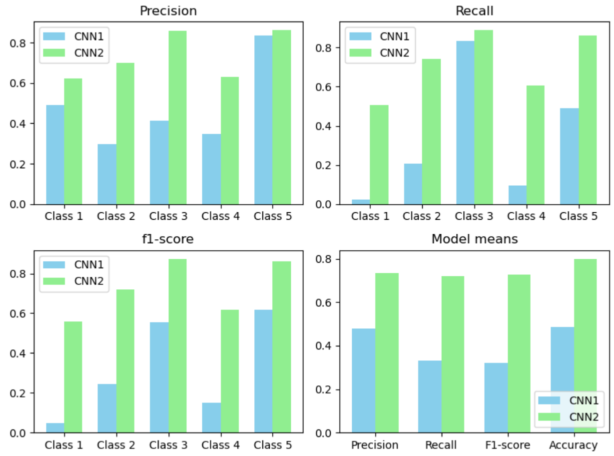

Figure 12: *Metrics comparaison of 2 models of the CNN*

As we can see, there is the same problem as there was with the RNN, model1 runs completely out of line. CNN1 makes very few predictions on class 1, 2 and 4 and a very big amount of predictions on class 3 (thus artificially augmenting the recall score on this class). We still do not understand where that problem comes from, the class unbalance can be a reason, but probably isn't the whole reason why. CNN2 performs pretty well and has decent scores across the classes, and very good scores on the most represented classes (3 and 5).

Final results

We compared our 2 best Deep Learning models, RNN and CNN on model 2.

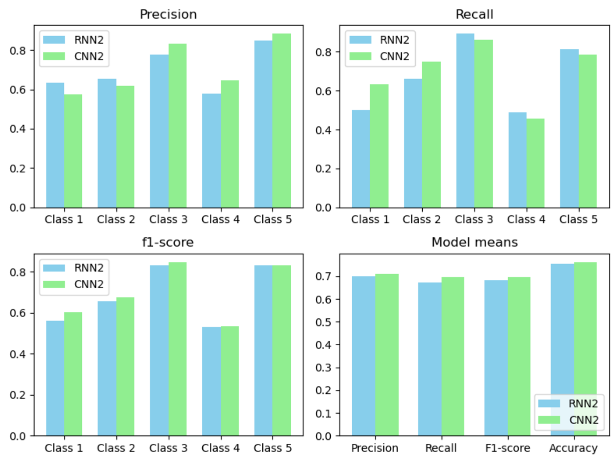

Figure 13: *Metrics comparaison CNN and RNN on model 2*

We can see CNN generally performs better than RNN, but the main difference is that CNN has better recall performances on under-represented classes.
The final model we would be presenting is the CNN2 (Convolutional Neural Network on the second model of word embedding), here are its performances :
We believe our model could be more effective if we

Class	Precision	Recall	F1-score	Support
0	0.62	0.51	0.56	2663
1	0.70	0.74	0.72	4426
2	0.86	0.89	0.87	9751
3	0.63	0.61	0.62	2377
4	0.86	0.86	0.86	10276

Table 9: *Classification Report for CNN2*

were to adapt the amount of layers, filters,... the data goes through. But it would require a better understanding of the model we are using, which we do not have right now.

Part III

Conclusion

Our study aimed to develop a pipeline for the classification of medical sentences into five categories. Using a dataset of 200k texts, we explored different preprocessing methods, feature engineering techniques, and classification models. We found that the RandomForest model, optimized with specific hyperparameters, gave really good results. Additionally, we compared the performance of two Deep Learning models, RNN and CNN, and found that CNN2, outperformed others in terms of recall, especially for underrepresented classes. In conclusion, our solution brings a relatively effective solution for automatic classification of medical sentences while being simple enough and not too ressource-consuming.