Personalized Medicine: Redefining Cancer Treatment

Predict the effect of Genetic Variants to enable Personalized Medicine

Kaggle competition

Master 2 bio-informatics, teaching unit « Learning, Artifcial intelligence and Optimization »

October 29th, 2021

Authors : Alix de Thoisy,
Ali Hamraoui,
Abdelmajid Omarjee

Keywords : Predictive medecine, natural language processing, deep learning, neural networks

Link to the Python Notebook : https://github.com/abdelom/kaggle/blob/main/NLP.ipynb

Introduction

This project is a natural language processing competition hosted on the Kaggle platform and entitled “Personalized Medicine: Redefining Cancer Treatment". It aims to automatically classify genetic mutations according to whether they are “driver” (highly contribute to the development of a cancer) or “passenger” (do not contribute), in a 9-classes scale. The analysis is based on clinical evidence using thousands of mutations annotated manually by world-class researchers and oncologists.

Data

Data presentation

The dataset provided for the model development consists of a comma separated file containing the mutation ID, the gene where it is located, the variation and the class (1, passenger, to 9, driver), and text file with the clinical evidence used for the classification (Figure 1).

	id	gene	variation	class	text
0	0	FAM58A	Truncating Mutations	1	cyclindependent kinase cdks regulate variety f...
1	1	CBL	W802*	2	abstract background nonsmall cell lung cancer ...
2	2	CBL	Q249E	2	abstract background nonsmall cell lung cancer ...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutation monomeric casitas blineage ...

Figure 1: First five lines of the dataset

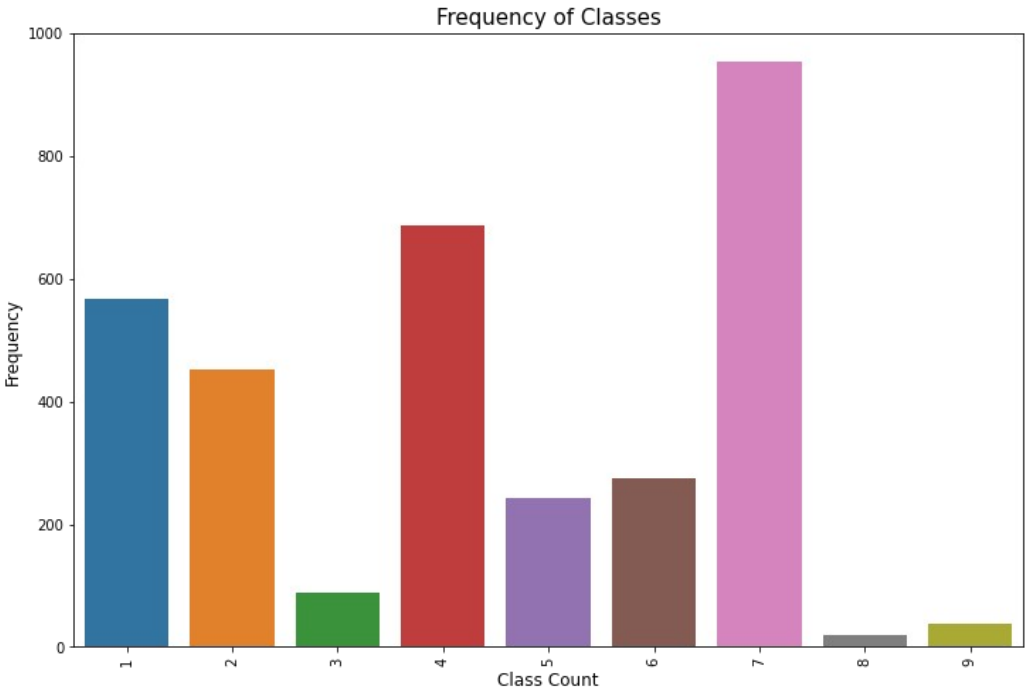

Figure 2: Distribution of the 9 classes in the entire data set.

The mutations are not equally distributed within the 9 classes (Figure 2). Indeed, class 8 contains 29% of the total mutations (n=953) and 8 only 0.6% (n=19). This issue will be addressed by several methods later on.

Early data processing

Tokenization, text cleaning and lemmatization

The first step of processing the data consisted of word tokenization. To this purpose, we used the *Natural Language Toolkit (nltk)* library to set all the words to vectors. All the characters were set to lowercase and the empty tokens deleted.
We then removed numerical values using regular expressions. *ntlk* methods allowed us to remove punctuation and stop words, as well as stemming (only keeping the radical of each word) and lemmatizing (saving words in a tree-like architecture)

Besides the common biological terms, some words seem to occur more often in certain classes, such as “p”, likely standing for “p53” or similar, and “BRCA”, both well known by oncologists (Figure 3).

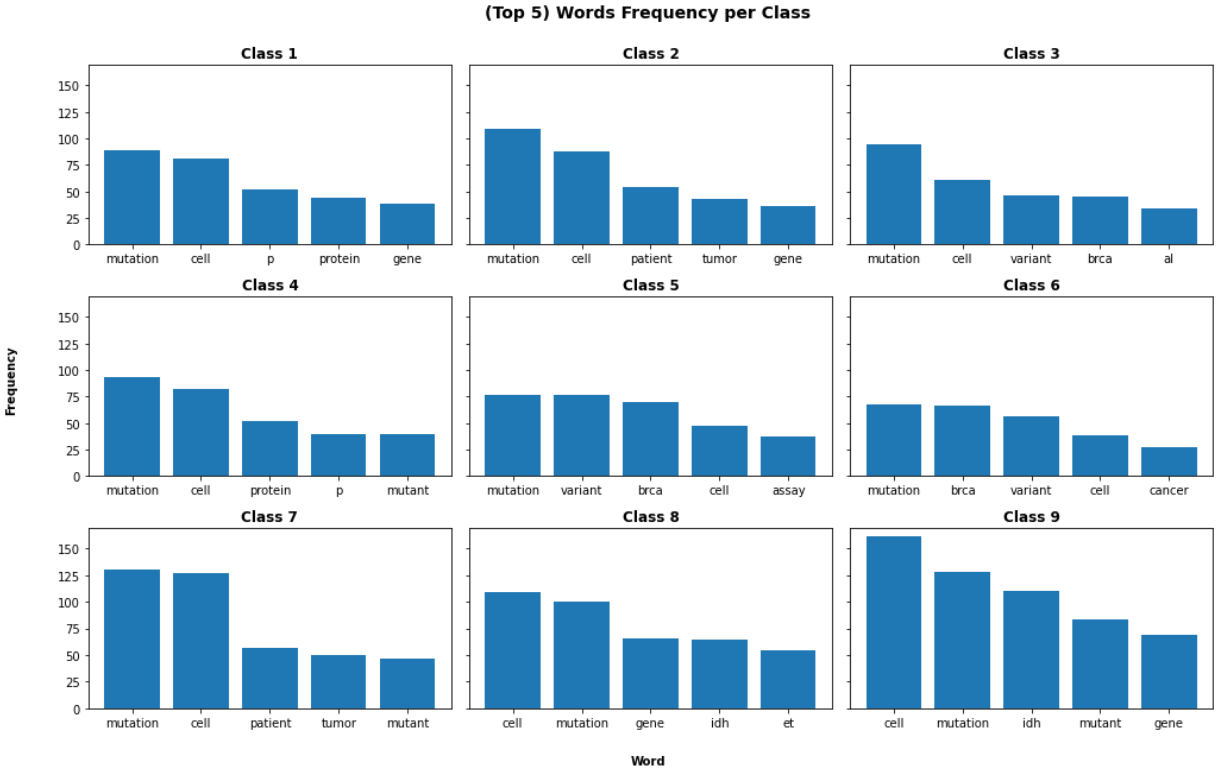

Figure 3 : Most represented tokens per class.

The same process has been conducted using the *Sci-Spacy* library, which is trained on medical vocabulary from PubMed articles and has the benefit of turning acronyms to full words. However, it has the major drawback of being very long to compute and the first runs using it returned similar results; it was therefore not used further.

TF-IDF and dimension reduction

In order to reflect the importance of words in each document, we performed a term frequency-inverse document frequency (TF-IDF) transformation (*TfidfVectorizer* from the *Scikit-learn Feature extraction* class), which returns a single numerical value matrix. The value of a word is its frequency in the particular document divided by the natural logarithm of its frequency in all documents. The 6000 highest values were kept.

This method may miss some information in comparison to word-embedding approaches but facilitates further analysis. Word-embedding techniques have been explored but its first results were less promising and are known to be less efficient on unblanced classes; they were therefore left aside.

The total dimension of the TF-IDF outpt is rather large (number of different words times the number of texts), leading us to investigate means to reduce it. A principal component analysis (PCA) with the *Scikit-learn decomposition library* helped us to visualize the contribution of the words for classification.

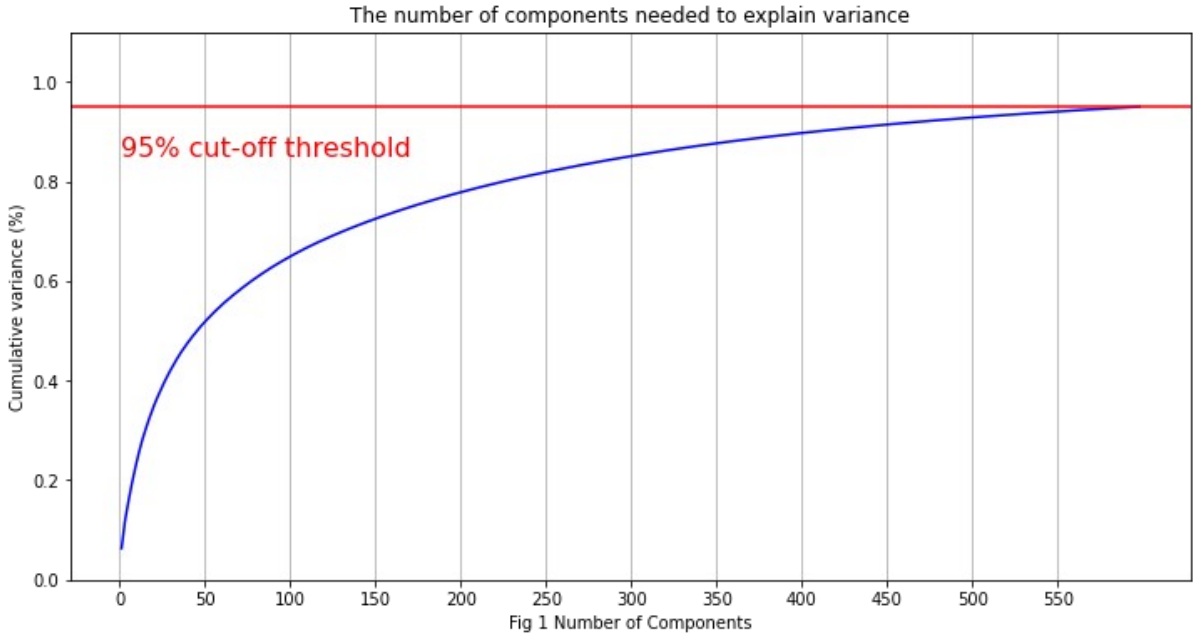

Figure 4 : Number of components neeed to explain 95% of the variance, according to the Principal Component Analysis

Figure 4 shows that less than 600 components are required to explain 95% of the total variance. The PCA method returned the value of 598, which we used to trim the data.
By reducing the number of parameters, we diminished the data noise and allowed the models to run faster.

T-SNE

T-distributed Stochastic Neighbor Embedding (T-SNE) converts similarities between texts to joint probabilities and tries to minimize divergence between the joint probabilities of the low-dimensional and the high-dimensional embedding. The plot shows some clear clusters of the 9 predefined classes, arguing that the TF-IDF transformation managed to grap well the variability in the texts (Figure 5).

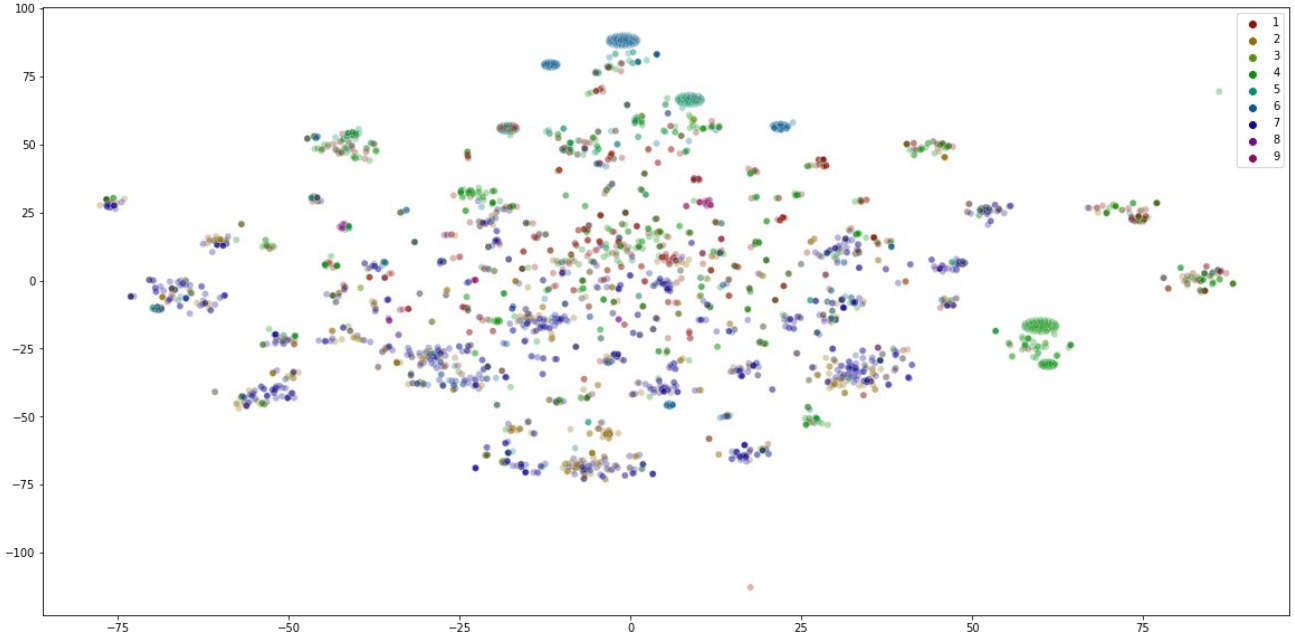

Figure 5 : T-SNE plot, visualisation of TF-IDF matrix in 2 Dimension embedded space.

Word-embedding

Word-embedding is the other widely-used technique to process data from natural language texts. Unlike TF-IDF, it creates a very large vector for each word that holds information about the “meaning” of the word, using pre-trained models.

First models and results

We went through an exploratory analysis of different models in order to have a rapid overview of the results and select the one to tune.

LSTM

The first model we tried was a Long Short-Term Memory (LSTM) algorithm. The architecture is classical and commnly found in the literature (Figure 6). It takes as input the word-embedding vector.

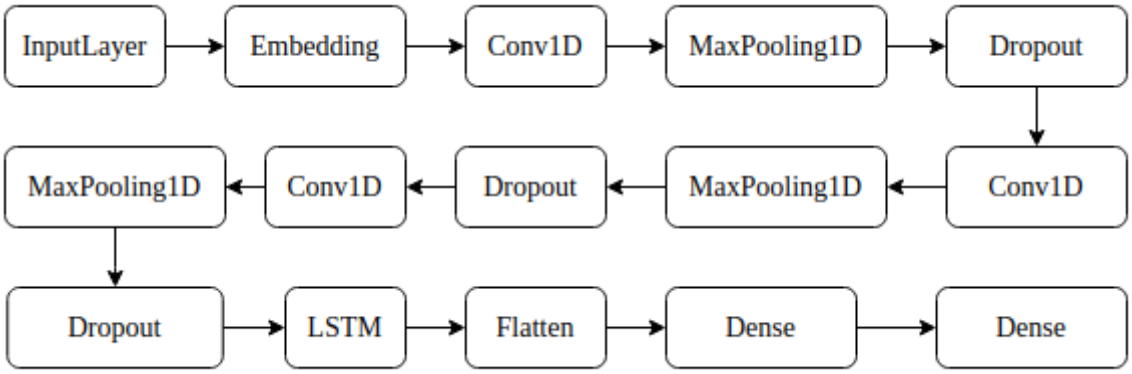

Figure 6 : Long Short-Term Memory architecture

CNN

A convolutional neural network (CNN) with a common architecture has been trained and tested on the TF-IDF transformation (Figure 7).
Adding a maxpooling layer did not seem much needed as TF-IDF does not grap “information” about the meaning of a word and so we are not looking for local “patterns” in the data. An average pooling could have made sense but has not been used since it was not required to diminish the runing time.
The first value of spatial droptout has been chosen randomly and the number of layers sized by the number of components.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 598, 1)]          0
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 598, 1)            0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 594, 128)          768
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 590, 128)          82048
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 586, 128)          82048
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 582, 128)          82048
_________________________________________________________________
conv1d_9 (Conv1D)            (None, 578, 128)          82048
_________________________________________________________________
flatten_1 (Flatten)          (None, 73984)             0
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 665865
=================================================================
Total params: 994,825
Trainable params: 994,825
Non-trainable params: 0
```

Figure 7 : Architecture of the Convolutional Neural Network

ResNet

The ResNet has an architecture similar to the one of the CNN (Figure 8). It usually performs better by addressing the vanishing gradient problem.

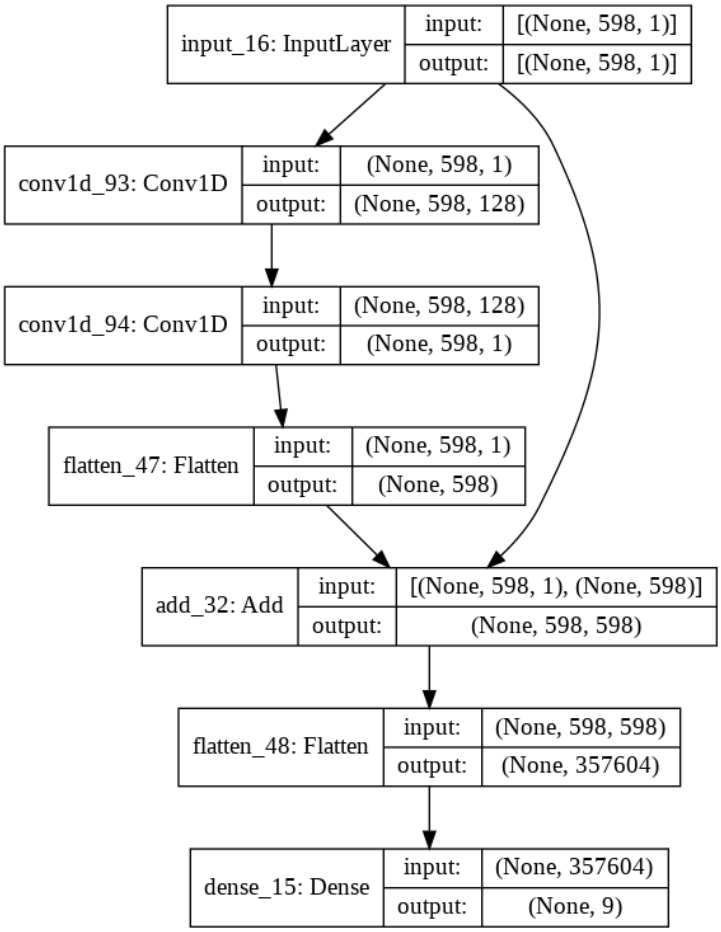

Figure 8 : Architecture of the ResNet.

Comparison

	Precision	Recall	Accuray training	Accuracy test
WE + CNN + LSTM	-	-	55	54
TF-IDF + CNN	66	53	64	60.05
TF-IDF + ResNet	53	55	79	59.7

Figure 9 : Comparison of different algorithms with commun parameters and architectures (values in %)

	Precision	Recall	Accuracy training	Accuracy test	Cross validation
SMOTE + TF-IDF + ResNet	-	-	55	54	-
SMOTE + TF-IDF + CNN	66	53	64	60.05	-
SMOTE + TF-IDF + CNN + hyperparameters tuning	53	55	79	59.7	-

Figure 12 : Comparison of the different models (values in %)

Approaches starting with a TF-IDF transformation outperform the classical word-embedding + CNN + LSTM combination (Figure 9), leading us to select the former. We decided to focus on one type of algorithm and process a long fine-tuning rather than trying out many techniques with default or commun parameters.

Oversampling

To tackle the unbalance of the classes, we ran the Synthetic Minority Oversampling Technique (SMOTE*)* technique on the training set. This data augmentation method synthesizes examples from minority classes that are close in the feature space. It does so by selecting, for a random instance of a minority class, k neighbors that belong to the same class, and by adding instances between them.

The sampling strategy has been set to “not majority” so all the classes contain the same number of values as in the second biggest class (Figure 10).

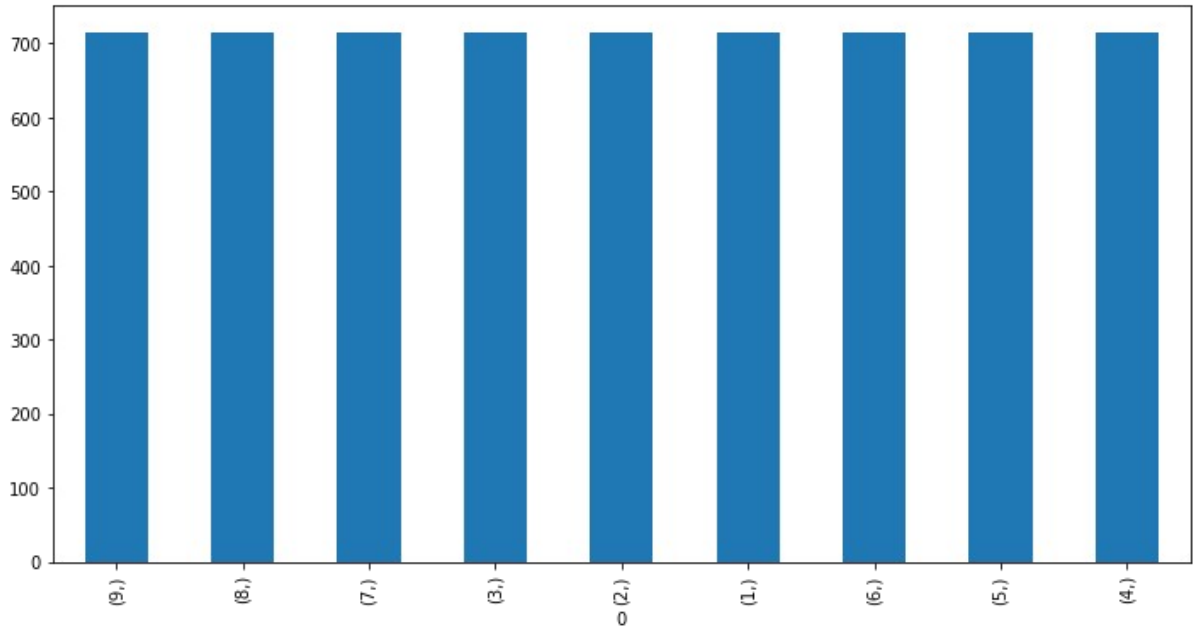

Figure 10: Perfectly bablanced distribution of the 9 classes in the SMOTE-transformed training set.

Hyperparameters tuning

Grid search

Two hyperparameters of convolutional neural networks have been tuned using a grid search approach: the rate of drop-out and the batch. The first one ranged from 0.0 to 0.3 by a rate of 0.05, the second the batch size from 100 to 200 with steps of 20. The accuracy scores were compared using the Scikit-learn *Model selection GridSearchCV* function and a 2-fold cross-validation.

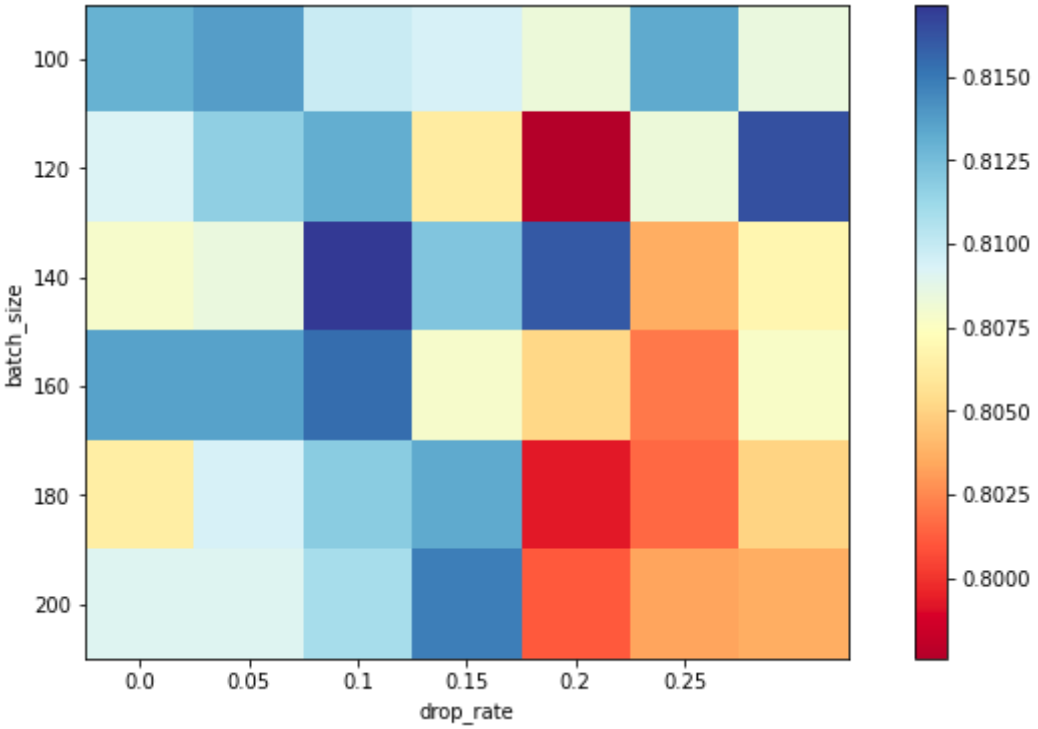

Figure 11 : Accuracies of CNN model according to the drop-out and batch size using 2-fold cross validation.

The accuracy is maximal with a batch size of 120 and a drop-out rate of 0.3. The exact tuning of these two hyperparameters seem however to have a low impact on the accuracy.

Bayesian optimization

We then performed an optimization of multiple hyperparameters at once with the *BayesianOptimization* Scikit tool.
The space of the hyperparameters has been defined as follows:

- number of CNN blocks : 2 to 4
- number of filters : 1x16 to 4x16
- kernel size : 2 to 5
- batch size : 1x32 to 5x32
- drop-out rate : 0.05 to 0.5
- validation split : 0.05 to 0.5

It came out that the hyperparameters that maximize the performance are 4 layers, 4x16 filters , kernel size of 5, batch size of 1x32, a drop-out rate of 0.1 and validation split of 0.1.

Final results

An overall comparison of the results puts forward the combinaison of SMOTE, TF-IDF and a convolutional neural network for perfoming best, with an accuracy of 81% on unseen data. (Figure 12)

	Precision	Recall	Accuracy training	Accuracy test	Cross validation
SMOTE + TF-IDF + ResNet	67	81	86.1	73	71
SMOTE + TF-IDF + CNN	76	85	84.9	81	81.67
SMOTE + TF-IDF + CNN + hyperparameters tuning	78	83	83.8	80.8	79

Figure 12 : Comparison of the different models (values in %)

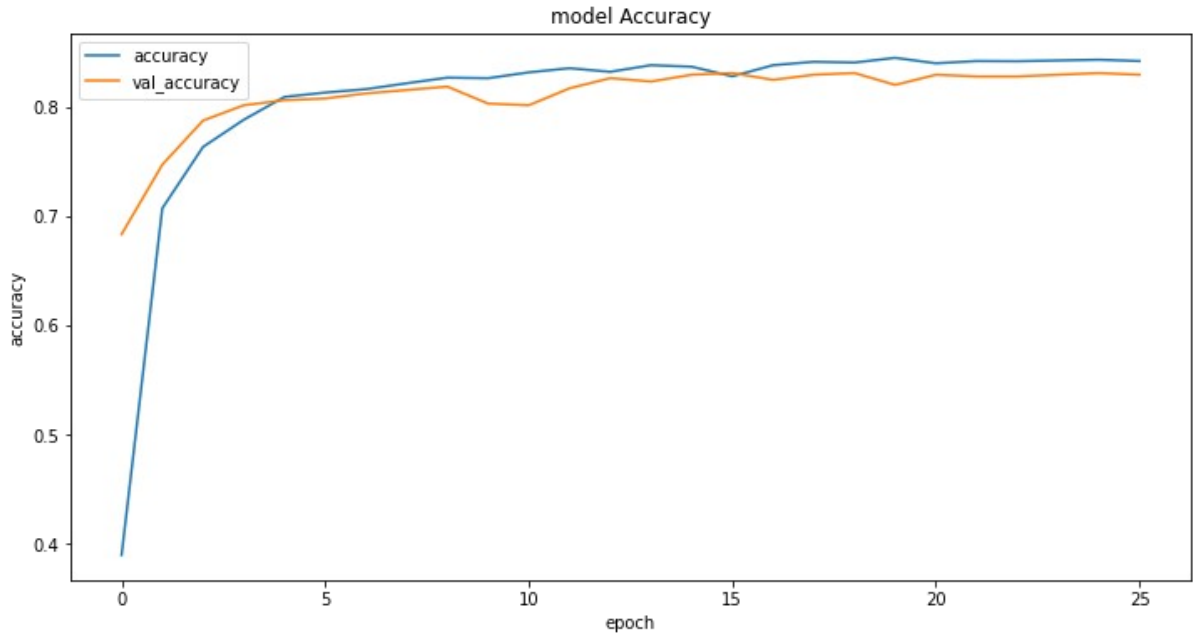

Figure 13: Learning curves of the tuned CNN after SMOTE and TF-IDF transformations

The learning curves on figure 13 shows that the model is well stabilized and does not tend to overfit.

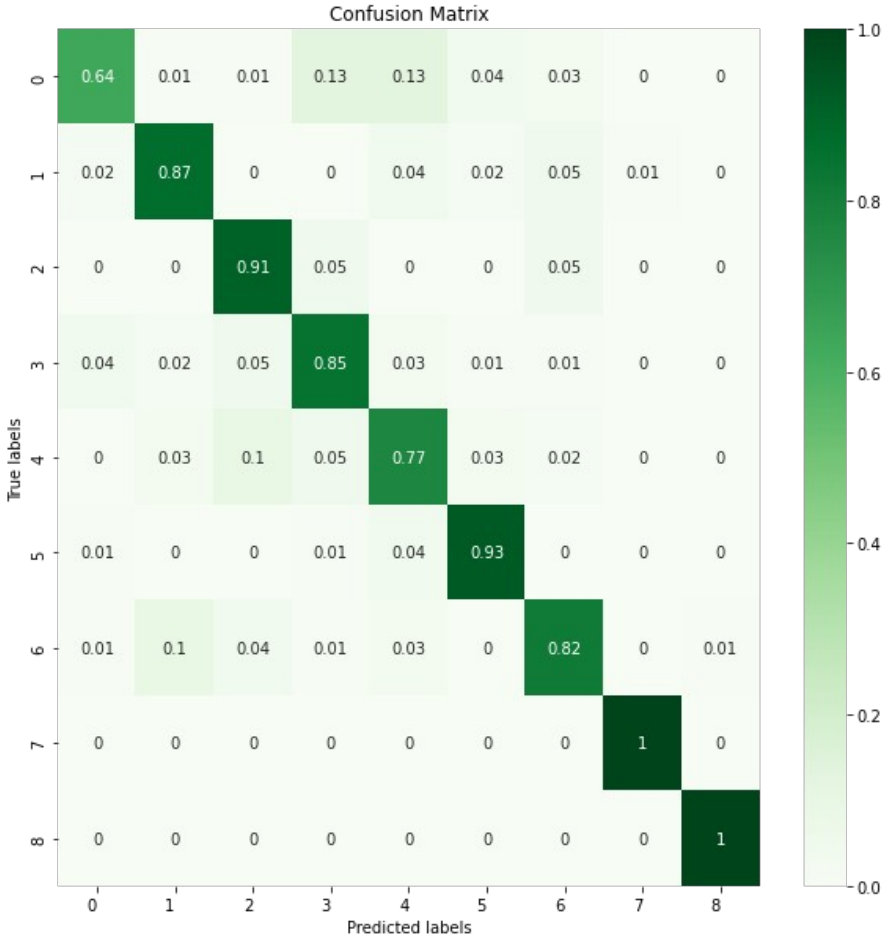

Figure 14: Sensibility confusion matrix of the SMOTE+TF-IDF+CNN on 5-fold CV.

	precision	recall	f1-score	support
0	0.88	0.64	0.74	142
1	0.77	0.87	0.81	113
2	0.44	0.91	0.60	22
3	0.85	0.85	0.85	172
4	0.57	0.77	0.65	61
5	0.84	0.93	0.88	69
6	0.94	0.82	0.87	238
7	0.83	1.00	0.91	5
8	0.82	1.00	0.90	9
micro avg	0.81	0.81	0.81	831
macro avg	0.77	0.86	0.80	831
weighted avg	0.84	0.81	0.82	831
samples avg	0.81	0.81	0.81	831

Figure 15: Perfomance scores of the SMOTE+TF-IDF+CNN on 5-fold CV, per class.

The combination of SMOTE / TF-IDF transformation and a convolutional neural network run in a 5-fold cross validation (CV) leads to very good sensitivy (>0.6 everywhere, often > 0.8) for every class (Figure 14). The F1-scores are in the same range (Figure 15), allowing us to think that this model is quite efficient at predicting whether a mutation is driver or passenger.

To gain an understanding of the missclassifications, we compare the length of texts. Mutations that were wrongly predicted come from significantly shorter texts (Figure 16). Should we take into account the fact that some texts in the dataset have themselves been generated by machine-learning algorithms, the accuracy may reach some overhead.

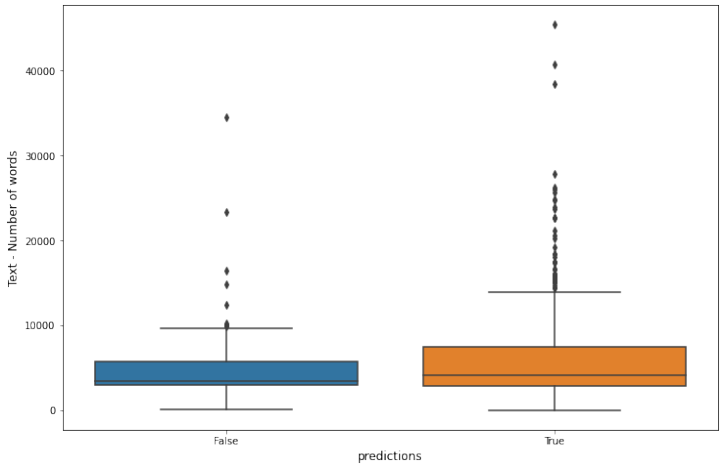

Figure 16: Length of texts about mutations wrongly and correctly predicted.

Discussion

In this project, we proposed a deep-learning approach to predict the contribution of mutations to the development of cancer using natural language processing.

At each step of the development, exploratory analysis led us to make decisions by selecting one type of model over others, and to focus our efforts on its tuning. That is, it is not guaranteed different highly-tuned models would not perfom as well or better. The good results tend nevertheless to consolidate our choices.

Further work on this topic would problably start by considering sentence-embedding approaches, which are known to do well for unbalanced classification.