INF 582 : Kaggle final report

Hatim Ezbakhe, Daniel Tordjman, Etienne Pochet
Team name : A_TEAM

Features Engineering

- Articles graph: We created a graph using the networkx library, where each node represents a paper, and an edge exists between two nodes if one paper cites the other.
- Title completion: We decided to complete short titles with, at most, 10 of their most important terms from a tfidf perspective. We ordered the abstract's words according to their tfidf weight and filtered them: we kept only those with a weight superior to 0.8 * max_abstract_tfidf_weight, with a limit of 10 words.

In the preprocessing part we build a dataset with the following features:

- number of overlapping words in the tittles (public)
- difference between years of publication (public)
- boolean indicating if the target article was published before the source article
- number of common authors
- number of citations between the authors
- cosine distance between td-idf vectors of the abstract
- number of citations of the target article
- cosine distance between fasttext vectors of the abstract
- boolean indicating if the articles belong to the same cluster
- number of source neighbors in our networkx graph (= number of article that mentions the source or that are mentioned by the source)
- number of target neighbors in our networkx graph (= number of article that mentions the target or that are mentioned by the target)
- number of common neighbors between the source and target

Distances between the abstracts

We taught about two vector representations for the abstract:

- the tf-idf vector
- fastText vector

Tf-idf vectors first allowed to improve the score. We tried two distance functions: the cosine and Euclidean distance, but the cosine distance provided better results. Then we thought about vectors standing for the semantic relatedness between articles. The FastText library appeared quickly as the perfect solution. FastText vectors are computed with a representation learning process. A neural network is trained on the corpus to predict the context in which we find a word. An abstract layer is used as a word representation. Once again the cosine distance between fastText vectors brought information, and the score increased thanks to this feature.

Using FastText vetors for clustering

In the corpus we may find clusters with articles close from each other on the sematic point of view. So, we use the fastText vectors and proceed with a k-mean algorithm on the corpus. In order to determine into how many cluster we would divide our dataset, we plotted the silhouette score evolution according to the number of clusters. We decided to divide it into 12 clusters (see results). Then we computed a Boolean feature indicating if two articles belong to the same cluster or not. We had some difficulties to evaluate the results of the clustering, we

had some doubts regarding the efficiency of the feature. But it appeared that this Boolean feature was correlated with the "category" and improved the score.

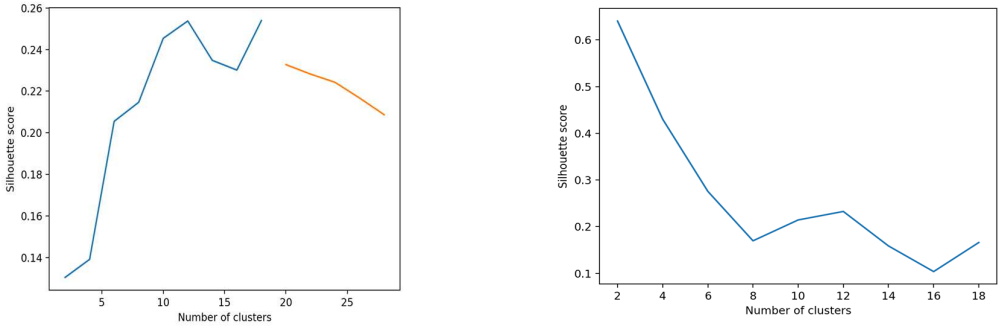


Figure: Evolution of the silhouette score according to the number of clusters, performing our kmean algorithm on TFIDF vectors(left), and on fasttext vectors (right)

Number of citations of the target article

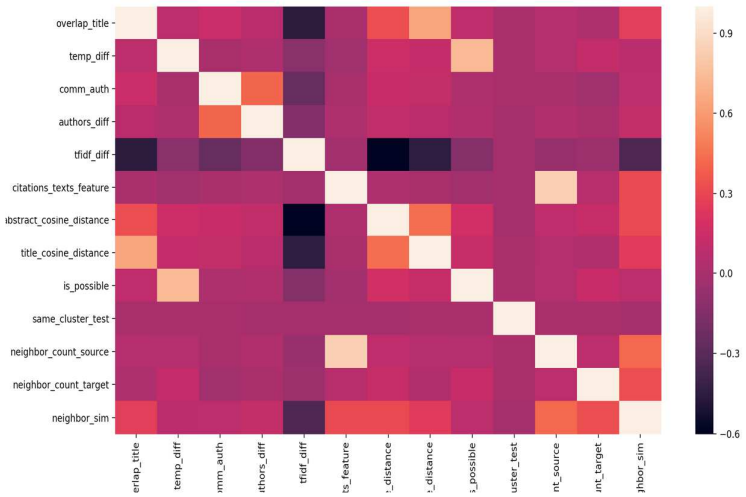
The more an article was quoted in the training set, the more likely the article will be quoted in the testing set. So we thought of computing a feature with the number of citations by article in the training set.

Distance between authors

We wanted to implement a feature that would give model the distance between the authors of the articles. At first, we started by computing an adjacency matrix with all the authors in the training set. The element on row i column j would be the number of times an article written by author i is linked to an article written by author j. This has been implemented easily.

Then, using this matrix, we were able to build a graph, using the igraph library, so we could access the actual distance between authors. We chose to use the sigmoid function. Therefore, the distance between two authors, with non-null number of citations, is sigmoid(1/number of citations). For authors with zero citation, we computed the shortest path to get a distance. The last step was to choose the best way to take into account this distance with multiple authors on the same article. We did not really settle for a solution and did not have time to implement this feature.

→ Correlation Matrix heatmap of our features



The heatmap matrix shows that all the features are correlated with the category (the feature we want to predict), and that the correlation between the features is for the major part of them not to high. This observation confirms that all the features catch some information about the prediction we are trying to make.

Model tuning and comparison

The public baseline script came with a Linear SVM model for the prediction. We began the preprocessing using this algorithm, then tried different predictions models.

Boosting Algorithms

We had a previous experience in Kaggle competitions where the boosting algorithms were quite performing. So we tried the XGBoost and LightGBM algorithms, but they appeared to be less efficient than the SVM algorithms.

→ Score : 0.90

Random Forest

This classifier is the one which gave us the best results. Therefore, we focused on its optimization in order to get the best possible score.
→ Score : 0.97

Non linear Model

As the SVM brought satisfying results, we decided to try a SVM algorithm with a radial basis function kernel, which improved a lot our score.

→ Score : 0.96

Grid Search and Final Model
We tried a Voting classifier made of SVC, RandomForestClassifier, AdaBoostClassifier and GradientBoostingClassifier. We did a grid search on these classifier parameters.
The conclusion was that the best classifier is a random forest, with n_estimators=50 and max_depth=7.

→ Score : 0.973