Web And Text Analytics, AMAZON review projects

Report

Pierre HOCKERS
engineering studen

November 2018

1 About the code

The code is organized in three files : The main one, called WebAndText, contains the main function and is the center of the project. We will there clean and preprocess the data, thanks to custom functions. We will construct there our reviews and collection objects, and call them for most of the work in Features Selection. In Training & Predictions we will treat a lot of data before being ready to use the predicting algorithms there. Finally, in the extra point, the same methods as for the previous section will be used in order to create the missing data for the added feature. We will go through each section in details

2 Cleaning & Preprocessing

I didn't notice the methods provided on the amazon webPage to use the JSON format until the end of the project, so since I couldn't find a way, I simply copy pasted the JSON file in a txt file and worked from there. A first function $ExtractHelpful$ is used to scan a line and see if the review has to be excluded (no helpfulness score) or not. If there is a score, we keep the line and start processing it, by first extracting the actual review, then cleaning it through 2 functions that remove stop words($CleanStopWord$) and punctuation signs ($RemovePunct$)

The cleaned review is then added in the form of a custom $Review$ object to a custom $Collection$ Object. A review object is mainly a representation of review, containing diverses way of representing the words inside of it, the helpfulness score associated, etc, plus a few handful methods. A collection is a collection of reviews, coming too with a lot of helpful methods.

3 Features selection

Using said methods, it was easy to compute the TFIDF scores of every word for every reviews. I gathered all words encountered in any review in a dictionnary called tfidf and associated with each word the mean of its tfidf scores. I also, before dividing the sum of the tfidf scores by the number of document having it, mutliplied the tfidf score of each word for a review by the helpfulness score of said review, so that the helpfulness would be impactful later. Ordering the words by their new score allows to select the x best one, which is done by a proportion variable. These words will from now one be referred to as "relevant words". I selected a proportion of 0.5 = 50% because it seemed to be a point where useless noise words started to get rarer but also where more sophisticated words started to be used, such as "artifacts", or words implying a constructed thought as "beside".

tfidf scores of 10 words :

— 'expensive' : 2.569311141031141
— 'well' : 1.4335739662091251
— 'arrived' : 3.512475721431468
— 'clamp' : 6.201357205709944
— 'popping' : 6.261014167159737

— 'blocks' : 8.45160421885235
— 'goose' : 8.416548868270192
— 'enough' : 2.410764074665813
— 'coaxing' : 7.454430555641825
— 'put' : 2.4325425699715217

4 Training & Predictions

I took a 10th of the reviews to make a testing set. the remaining reviews were kept as the training set. After some data processing to be ready for the algorithms, we can apply them. As a linear regression I used the *linear_model* from sklearn. I also used the *polyfit* method from numpy as I wasn't sure that the regression was supposed to be strictly linear or if we could use a polynome too. For the SVR, the *svm.SVR* method from sklearn was used.

Custom functions were used to make the code more readable. They basically create the model, train it and predict both the train and test samples. They also plot the results. The plots are visible in annex.

The results of the linear models were quite good : - the linear simple regression had a MSE of 0.1085 on the training set, and of 0.1173 on the testing set. Since the range of scores is between 0 and 1 this is a 10% mean error, which is rather good.
- the polynomial regression had similar results : on the training set 0.0998 and on the testing set : 0.1048
- The SVR was rather close too : error (MSE) of the SVR on the training set : 0.1057 error (MSE) of the SVR on the testing set : 0.1148
The SVR overfits a bit too much which works against it for this problem.

5 EXTRA

After preprocessing the data in the same way we did earlier, we apply the exact same algorithms (minus the polynomial one to not overcharge the rest of the results) after adding the number of words in the review as an input. The results are therefore also plotted and will be available in annex. Here are the error results :
error (MSE) of the linear regression on the training set : 0.11714683261342791
error (MSE) of the linear regression on the testing set : 0.12296236554221941
error (MSE) of the SVR on the training set : 0.09143681452547699
error (MSE) of the SVR on the testing set : 0.13093535883225862

The results are worse than before. I am not sure if this is supposed to be the case, I might have made a mistake. However, it seems like the number of words has no correlation with the helpfulness of the review. A review can be short and very helpful, or short and simply useless. Same goes for long detailled reviews or long vague and unclear reviews. If this feature is indeed not relevant, it might explain the slightly less good performances, since this would basically be adding noise.

For the classifier, I chose the K Nearest Neighbors classifier, simply because I am used to it and it is a recommended classifier for small numbers of features.

MSE of the KNN on the train set : 0.16816720257234727 MSE of the KNN on the test set : 0.2579710144927536

It is hard to compare these results to older ones, because the scale is not the same. the difference between a good value and a bad one will be of 1, while the older one could be closer since they were dispersed. Since the error is systematically of one, we can still see that we have an error rate of 1 out of 4, which is not that bad but clearly improvable. This could probably be fixed with other relevant features. The results are, as usual, available in annex.

The confusion matrix is the following :
|245 , 14|
|76 , 10| We can see that there were a lot of mistake (76) on the classification of the 1st class, 76 mistakes for 245 correct guesses. It's even worse for the second class, where there were more mistakes (14) than good guesses. The average is still way higher than half good answer half bad answer, but we could definitely hope for better.

6 Annex

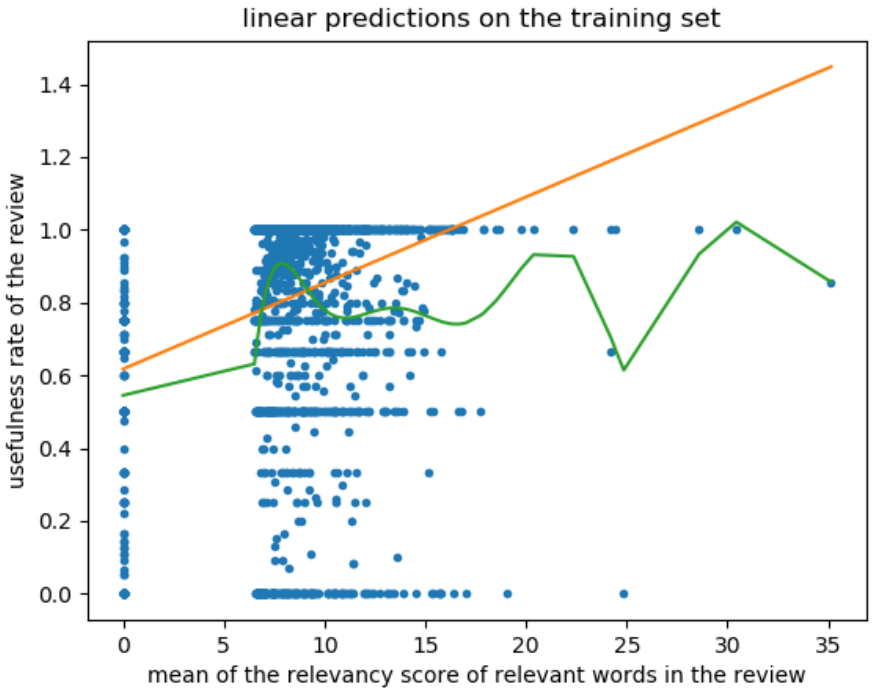

FIGURE 1 – Predictions of linear regression on the train sample

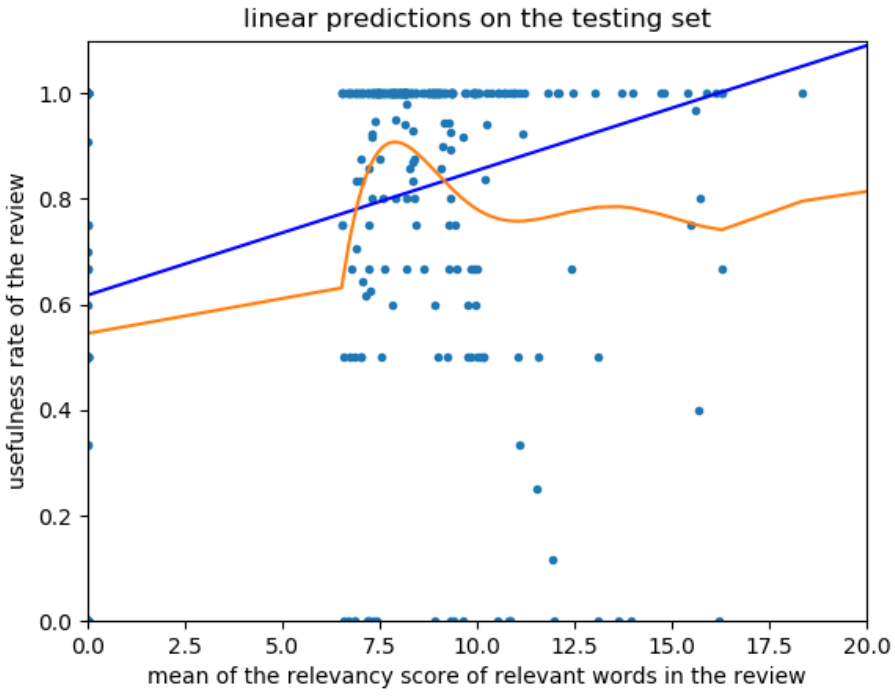

FIGURE 2 – Predictions of linear regression on the test sample

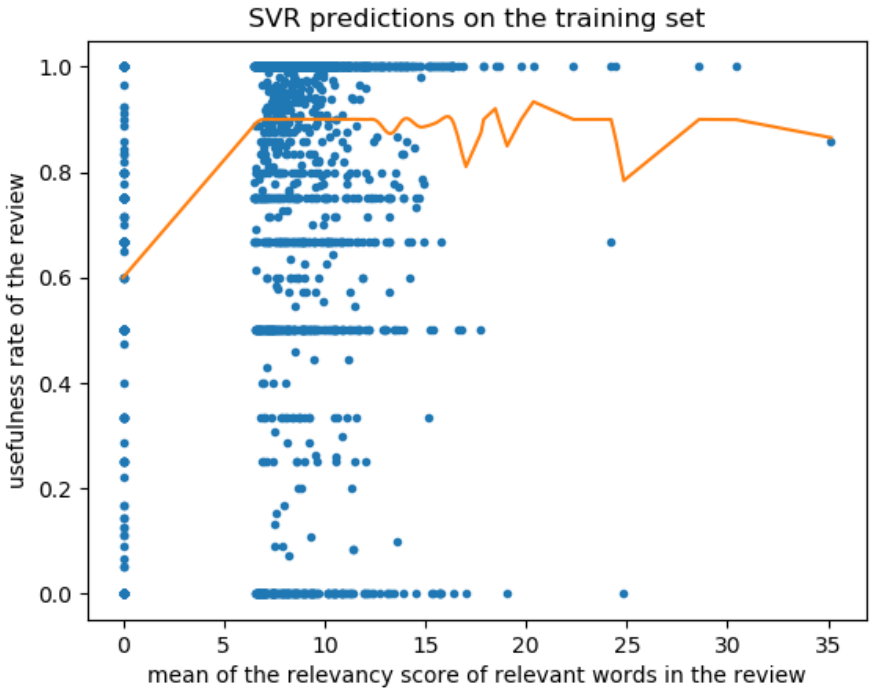

FIGURE 3 – Predictions SVR on the train sample

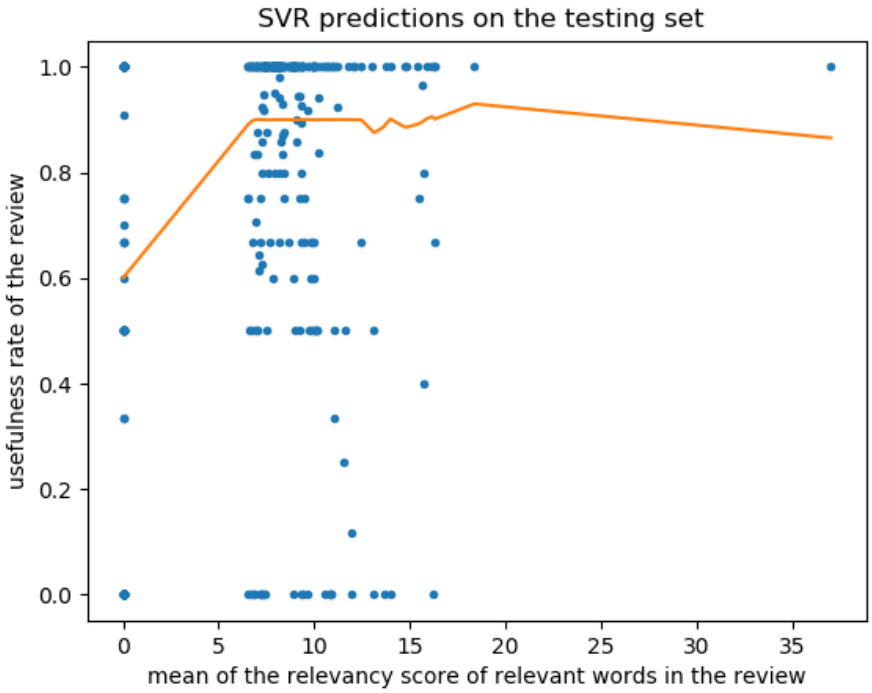

FIGURE 4 – Predictions of SVR on the test sample

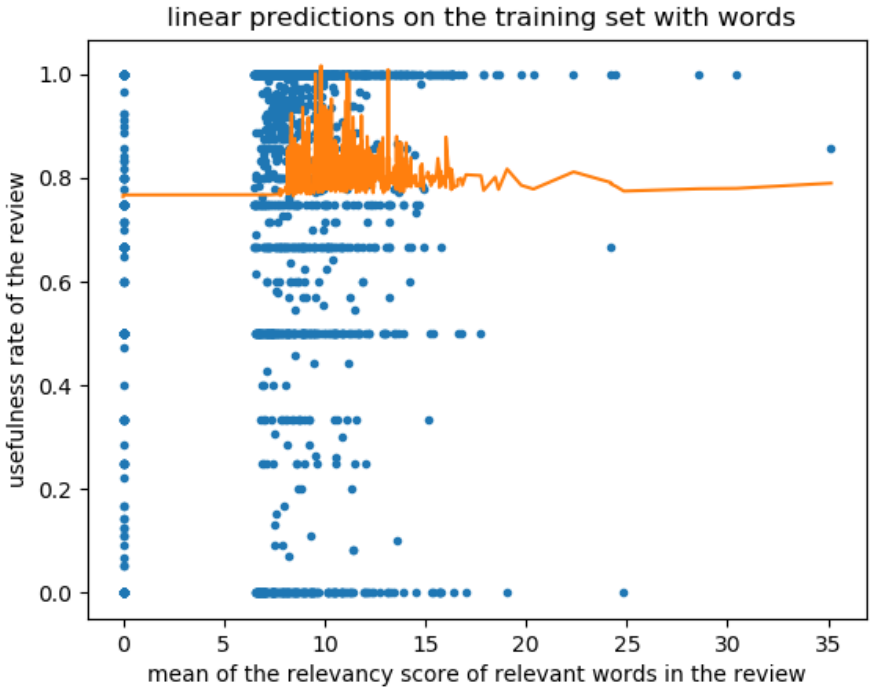

FIGURE 5 – Predictions of linear regression with the number of words as an additionnal feature on the train sample

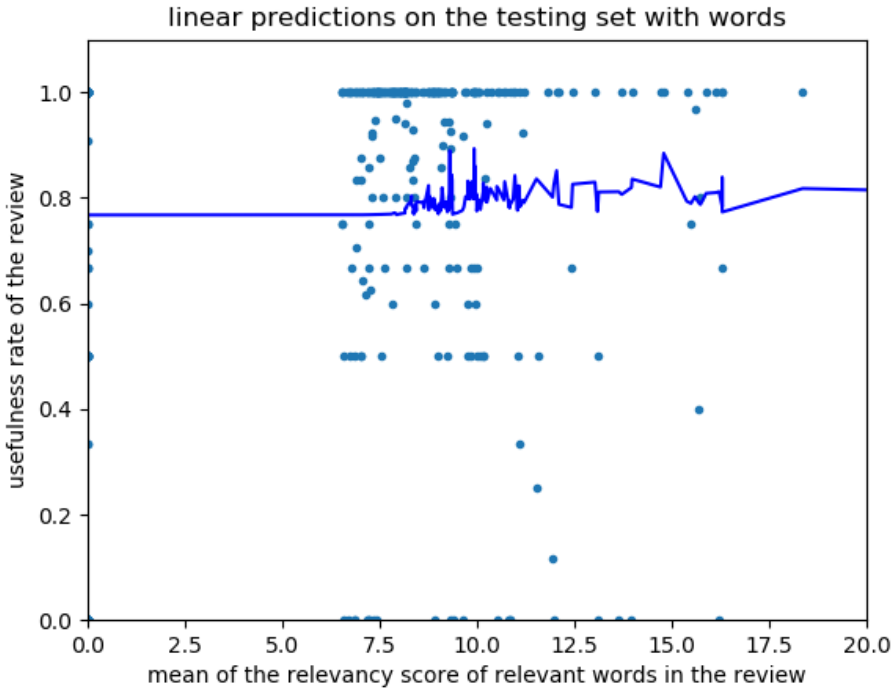

FIGURE 6 – Predictions of linear regression with the number of words as an additionnal feature on the test sample

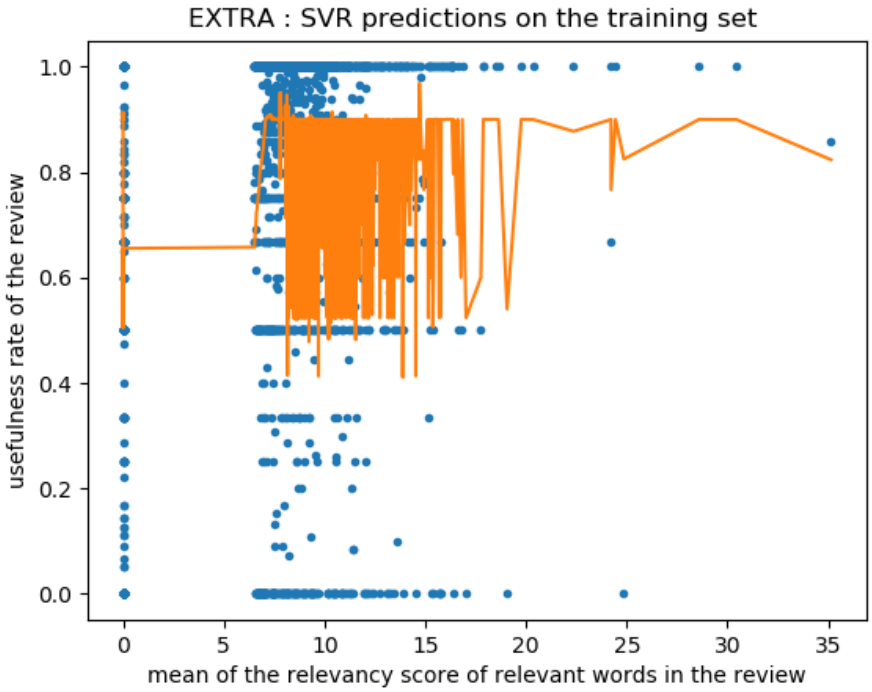

FIGURE 7 – Predictions of SVR with the number of words as an additionnal feature on the train sample

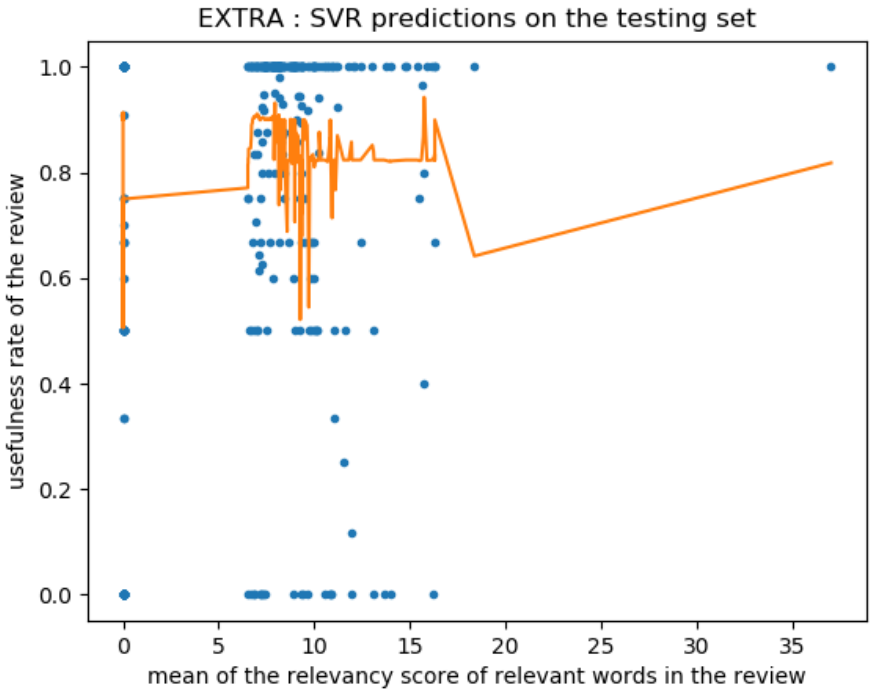

FIGURE 8 – Predictions of SVR with the number of words as an additionnal feature on the test sample

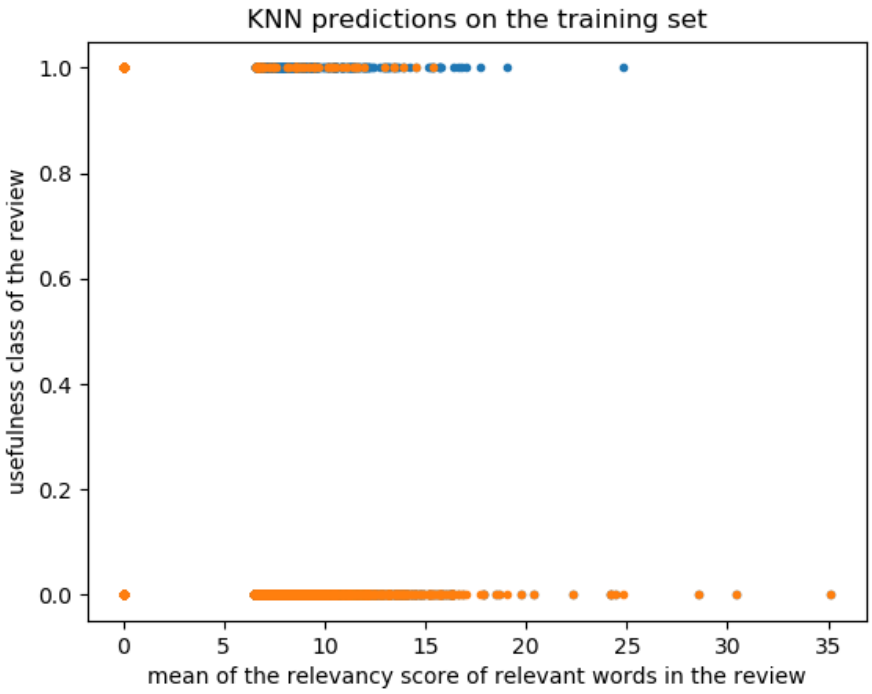

FIGURE 9 – Predictions of KNN with the number of words as an additionnal feature on the train sample as a classification problem

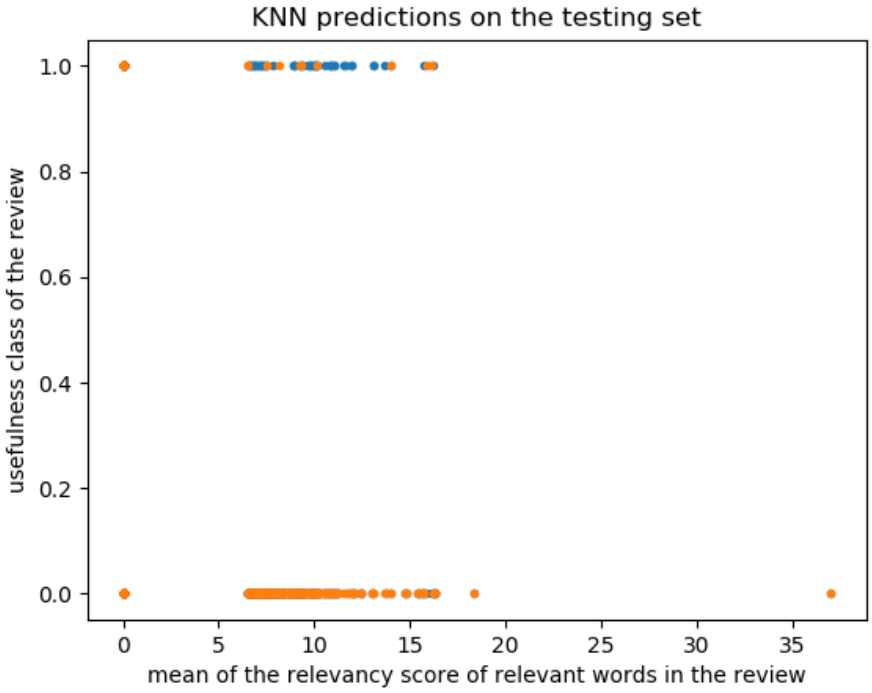

FIGURE 10 – Predictions of KNN with the number of words as an additionnal feature on the test sample as a classification problem