MS High Performance Computing & Artificial Intelligence

Introduction to Machine Learning : Kaggle in class Challenge

Student
Full name : Ismaïl Haris
Kaggle username : Ismail Haris
Kaggle team name : I.Haris

Teacher :
Dr. Chloé-Agathe Azencott

2019-2020

1 Introduction

The goal of the challenge is to predict how many times an online article is going to be shared using variables that describe the article such as the topic, the keywords, the sentiment analysis etc ... I present you through this report some insights that I made about the data exploratory analysis, the feature engineering that I thought useful, the expected results through the cross validation framework of the different models that I've tried, and finally the performance of these models on the validation sets.

2 Exploratory data analysis

— (Fig.1) There are no NaNs to fill in all the data provided. All data is numerical.

```
In [17]: data_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 44 columns):
identifier                    5000 non-null int64
nb_words_title                5000 non-null int64
nb_words_content              5000 non-null int64
pp_uniq_words                 5000 non-null float64
pp_stop_words                 5000 non-null float64
pp_uniq_non-stop_words        5000 non-null float64
nb_links                      5000 non-null float64
nb_outside_links              5000 non-null int64
nb_images                     5000 non-null int64
nb_videos                     5000 non-null int64
ave_word_length               5000 non-null int64
nb_keywords                   5000 non-null int64
category                      5000 non-null int64
nb_mina_mink                  5000 non-null int64
nb_mina_maxk                  5000 non-null int64
nb_mina_avek                  5000 non-null int64
nb_maxa_mink                  5000 non-null int64
nb_maxa_maxk                  5000 non-null int64
nb_maxa_avek                  5000 non-null float64
nb_avea_mink                  5000 non-null int64
nb_avea_maxk                  5000 non-null int64
nb_avea_avek                  5000 non-null float64
nb_min_linked                 5000 non-null int64
nb_max_linked                 5000 non-null int64
nb_ave_linked                 5000 non-null float64
weekday                       5000 non-null int64
dist_topic_0                  5000 non-null float64
dist_topic_1                  5000 non-null float64
dist_topic_2                  5000 non-null float64
dist_topic_3                  5000 non-null float64
dist_topic_4                  5000 non-null float64
subj                          5000 non-null float64
polar                         5000 non-null float64
pp_pos_words                  5000 non-null float64
pp_neg_words                  5000 non-null float64
pp_pos_words_in_nonneutral    5000 non-null float64
ave_polar_pos                 5000 non-null float64
min_polar_pos                 5000 non-null float64
```

— (Fig.2) There is a lot of disparities between features means, which imply to standardize all to have comparable features. Means and medians are sometimes different and sometimes two or three percentiles (25%, 50% and 75%) are the same, which suggest some skewed data (either right or left) and the need for feature engineering.

```
In [20]: data_train.describe()
```

Out[20]:

	identifier	nb_words_title	nb_words_content	pp_uniq_words	pp_stop_words	pp_uniq_non-stop_words	nb_links	nb_outside_links	nb_images	nb_videos	..
count	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03	5000.000000	5000.000000	5000.00000	5000.00000	5000.000000	..
mean	4499.500000	10.380000	562.619200	0.530260	3.020000e-02	0.672933	10.709400	7.42240	4.36960	1.234200	..
std	1443.520003	2.104777	465.785259	0.136349	1.711544e-01	0.152879	10.700498	9.79755	7.95729	3.952781	..
min	2000.000000	4.000000	0.000000	0.000000	2.430000e-10	0.000000	0.000000	0.00000	0.00000	0.000000	..
25%	3249.750000	9.000000	265.000000	0.472275	2.379500e-09	0.627575	4.000000	1.00000	1.00000	0.000000	..
50%	4499.500000	10.000000	427.000000	0.539700	4.082000e-09	0.691700	7.000000	4.00000	1.00000	0.000000	..
75%	5749.250000	12.000000	724.250000	0.606725	6.667000e-09	0.754275	14.000000	10.00000	3.00000	1.000000	..
max	6999.000000	19.000000	7775.000000	1.000000	1.000000e+00	1.000000	162.000000	159.00000	128.00000	75.000000	..

8 rows × 44 columns

— We do have categorical features : 'category' Category of the article : 0-Lifestyle, 1-Entertainment, 2-Business, 3-Web, 4-Tech, 5-World and 'weekday' Day of the week : 0-Monday, 1-Tuesday, 2-Wednesday, until 6-Sunday

— I was surprised to see that 0 was the minimum of 'nb_words_content', but if we look at that line of the dataframe, it is normal since the article contains only a video.

— In the training set, I have noticed that there are some data (151) where the average word length is 0, which I don't understand. Sometimes it is the fact that the article contains only images or videos, but there are also some samples with no video, no image, a number of word content non null, but an average word lenght null (fig.7).

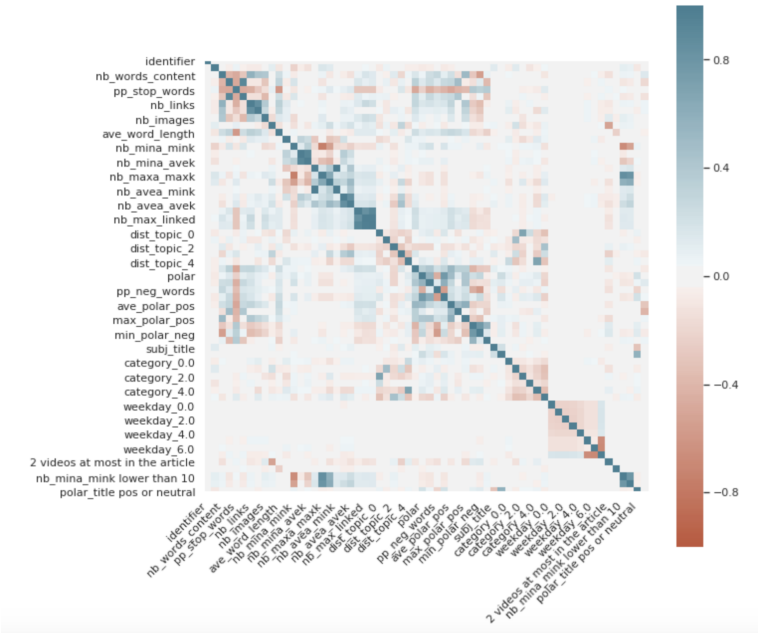
```
In [57]: X_train[X_train['ave_word_length']==0]
```

Out[57]:

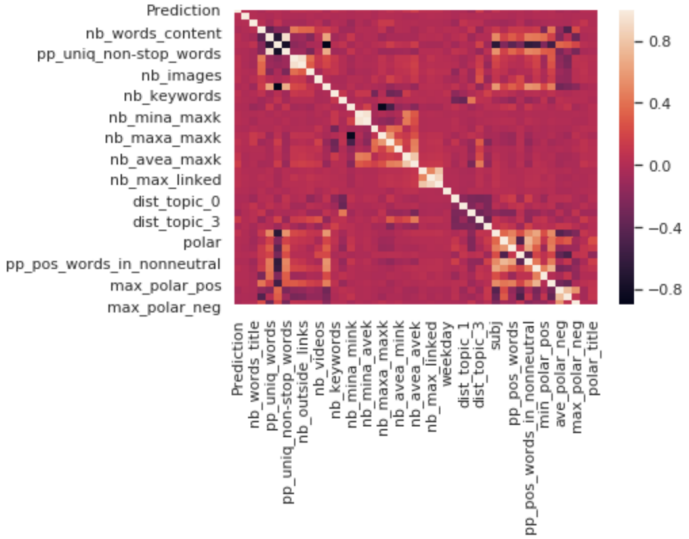
	identifier	nb_words_title	nb_words_content	pp_uniq_words	pp_stop_words	pp_uniq_non-stop_words	nb_links	nb_outside_links	nb_images	nb_videos	...	category_
37	2037.0	12.0	22.0	0.0	1.0	0.0	0.0	0.0	14.0	0.0	...	
60	2060.0	12.0	13.0	0.0	1.0	0.0	0.0	0.0	12.0	0.0	...	
74	2074.0	8.0	31.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	...	
146	2146.0	11.0	7.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	
154	2154.0	12.0	22.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
4892	6892.0	11.0	6.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	
4897	6897.0	11.0	21.0	0.0	1.0	0.0	0.0	0.0	11.0	0.0	...	
4955	6955.0	10.0	18.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	
4965	6965.0	9.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	
4998	6998.0	12.0	23.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	

51 rows × 57 columns

— When looking at the correlation matrix (fig.12), it may be interesting to do some dimensionality reduction. When I have looked at the correlation between the features and the training target, I haven't seen any remarkable correlation (nothing bigger than 0.5) to push me to consider the most correlated features to the output. I will consider thus the most varying features.



No correlation between features and target superior to 0.5



In [94]:
```
corrcolumns = []
for i in range(datacorr.shape[0]):
    if np.abs(datacorr.iloc[0,i]) >= 0.5:
            corrcolumns.append(result.iloc[:,i].name)
selected_columns = np.asarray(corrcolumns)
```

In [95]:
```
result[selected_columns]
```

Out[95]:

	Prediction
0	882
1	1102
2	1102

— I check if there are no duplicates in the training and the test sets using 'duplicate_test = X_test.duplicated() and duplicate_test[duplicate==True] '. There are no duplicates.

3 Feature Processing

— Encoding categorical features :
We use it because there is no reason why a point should be further from category 6 than category 1 for example, it should be the same. We use One hot Encoding both on training and testing sets. We drop the 'category' and 'weekday' columns and create 6 and 7 new columns (fig.3) with binary representation.

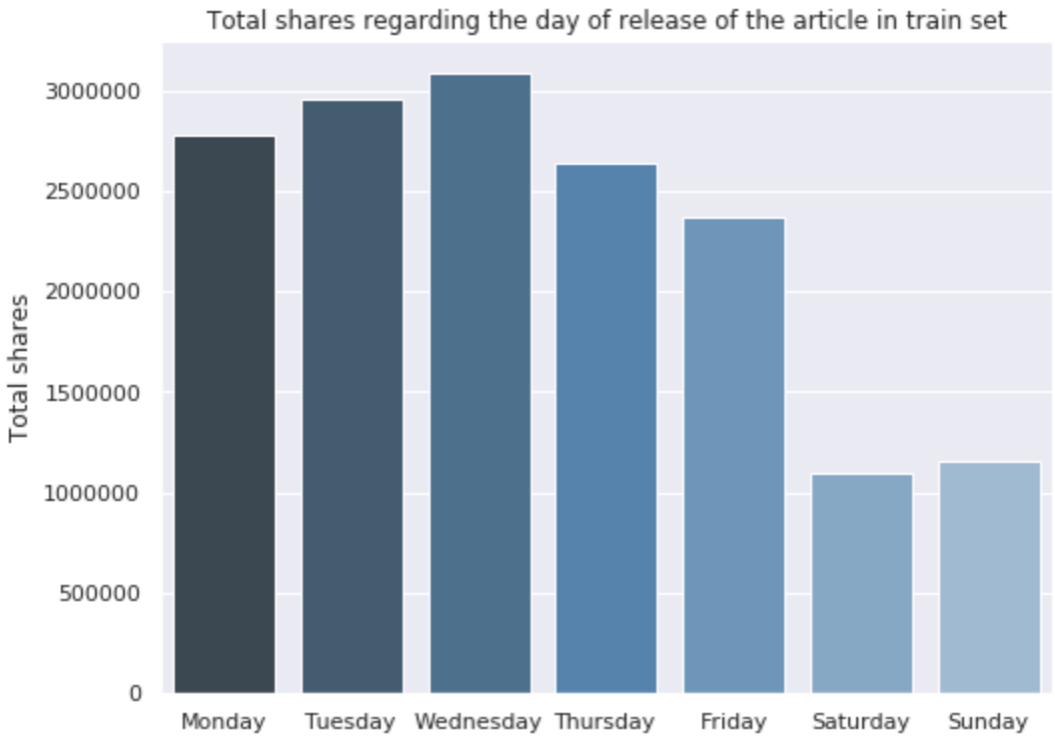
In [26]:
```
#we do it for 'category' feature
# Get the category data and encode it using a dummy categorical encoding
category_data = pd.get_dummies(X_train['weekday'], prefix='weekday', drop_first=False, dtype= int)
category_data_test = pd.get_dummies(X_test['weekday'], prefix='weekday', drop_first=False, dtype= int)
# Get the rest of the data
other_data = X_train.drop(['weekday'], axis = 1)
other_data_test = X_test.drop(['weekday'], axis = 1)

# Create a new data set by concatenation of the new category data and the old rest of the data
X_train = pd.concat([other_data,category_data], axis=1)
X_test = pd.concat([other_data_test,category_data_test], axis=1)
# Print the created training data.
X_train.head(5)
```

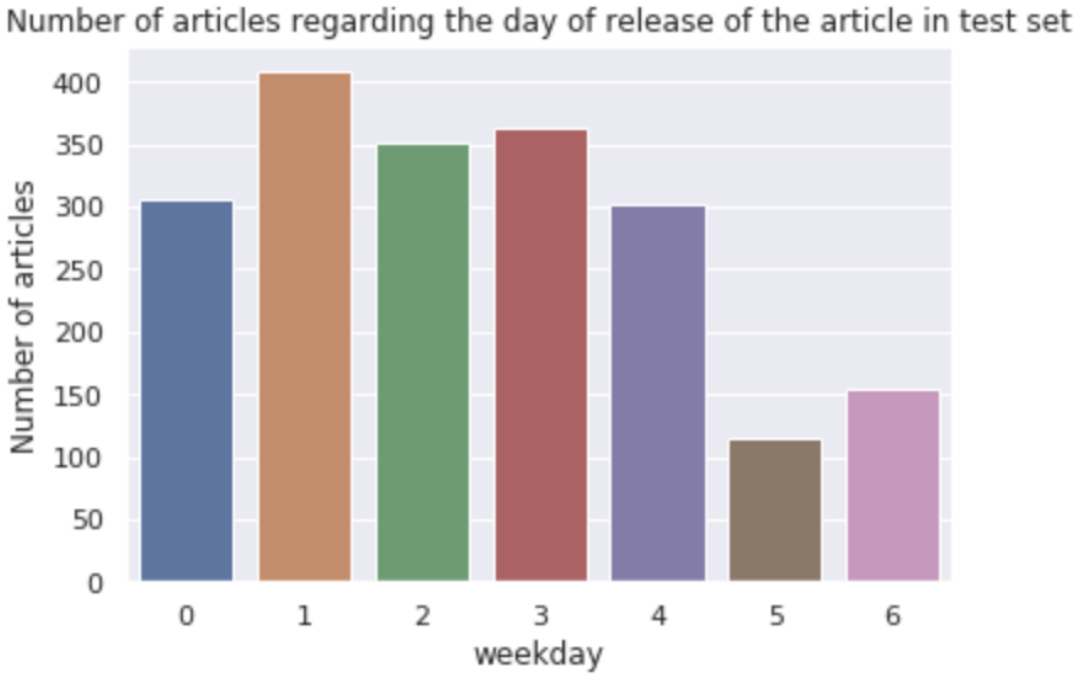
Out[26]:

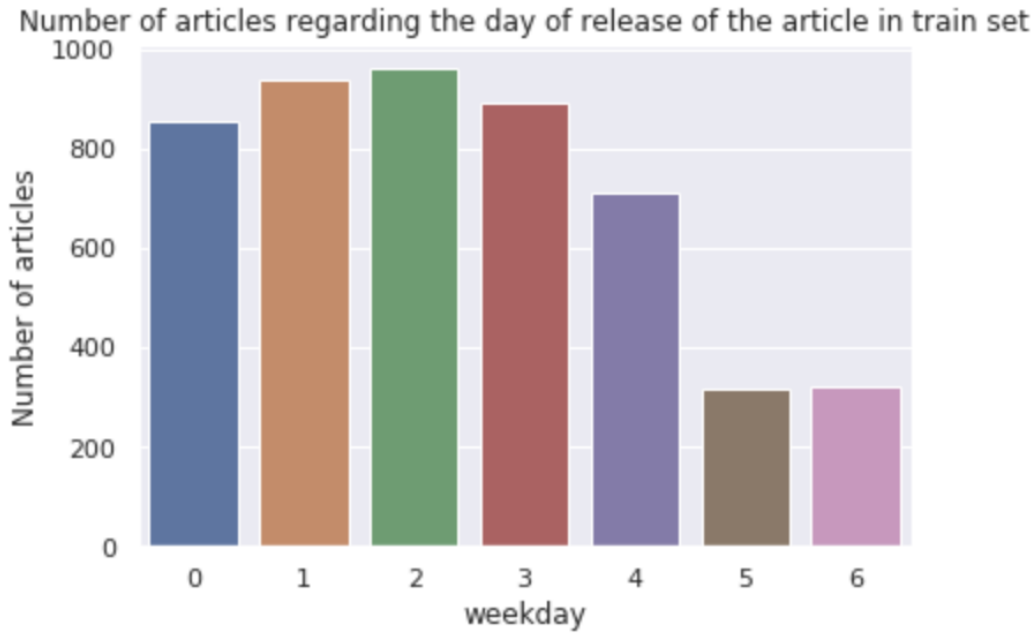
mages	nb_videos	...	category_3.0	category_4.0	category_5.0	weekday_0.0	weekday_1.0	weekday_2.0	weekday_3.0	weekday_4.0	weekday_5.0	weekday_6.0
11.0	1.0	...	0	0	0	0	1	0	0	0	0	0
1.0	0.0	...	0	0	1	0	0	1	0	0	0	0
0.0	2.0	...	0	1	0	1	0	0	0	0	0	0
0.0	0.0	...	0	1	0	0	0	1	0	0	0	0
1.0	0.0	...	0	0	1	0	0	1	0	0	0	0

— If we look closer to the weekdays (fig.4), it seems judicious to binarize this features because there are clearly much less articles published on the weekends than during the week (2 to 3 times less) either in train or test sets. I created a feature that qualifies whether the day is a working day or the weekend

```
weekday_0.0  weekday_1.0  weekday_2.0  weekday_3.0  weekday_4.0  weekday_5.0  weekday_6.0
0            0            0            0            0            0            1            323
                                                                  1            0            315
                                                    1            0            0            712
                                         1          0            0            0            891
                                1        0          0            0            0            961
                     1          0        0          0            0            0            941
1                    0          0        0          0            0            0            857
dtype: int64
```


Total shares regarding the day of release of the article in train set
Total shares
3000000
2500000
2000000
1500000
1000000
500000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday


Number of articles regarding the day of release of the article in test set
Number of articles
400
350
300
250
200
150
100
50
0
0
1
2
3
4
5
6
weekday



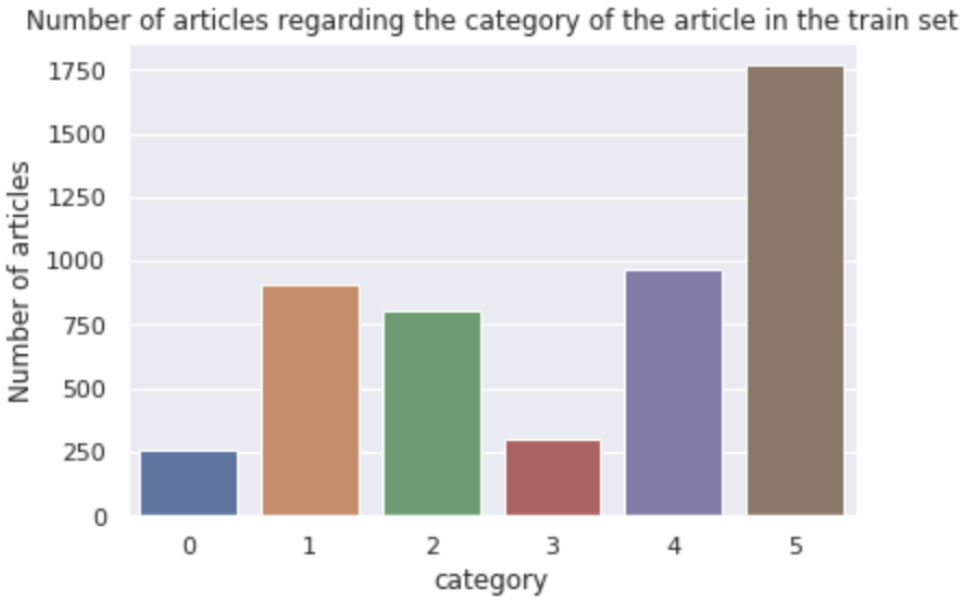
NB : I have noticed some disparities among the different categories (ie category 0 and 3 are quiet underrepresented , see fig.5) but I prefer to not add any change concerning the article categories.

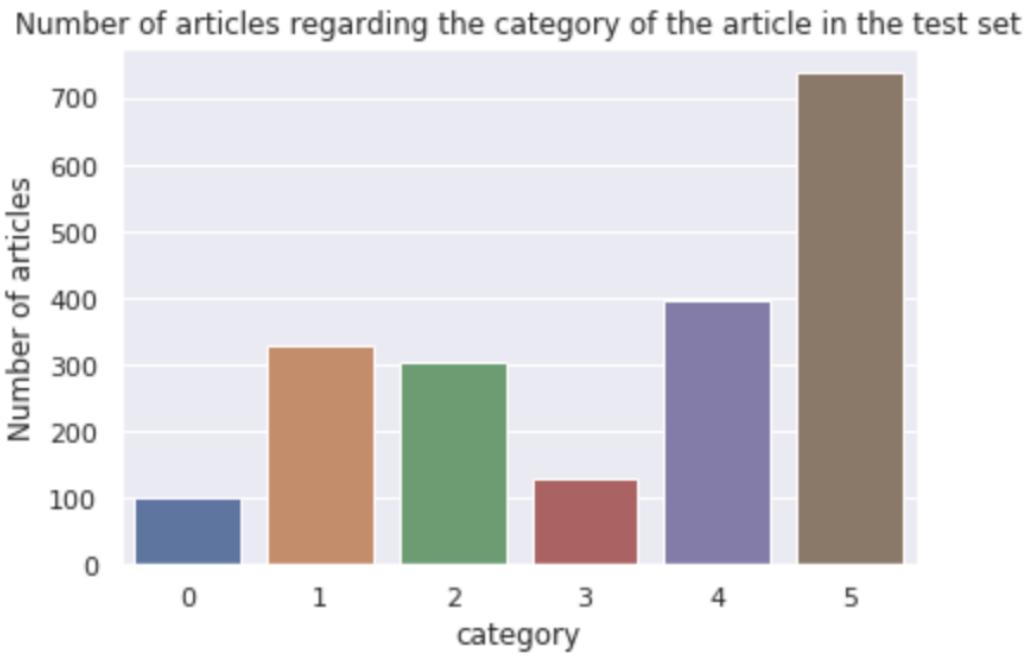
```
In [36]: X_train.groupby(['category_0.0','category_1.0','category_2.0','category_3.0','category_4.0','category_5.0']).size()
```

```
Out[36]: category_0.0  category_1.0  category_2.0  category_3.0  category_4.0  category_5.0
         0             0             0             0             1             1767
                                                                   0              966
                                                   1             0             0   305
                                     1             0             0             0   803
                       1             0             0             0             0   904
         1             0             0             0             0             0   255
         dtype: int64
```

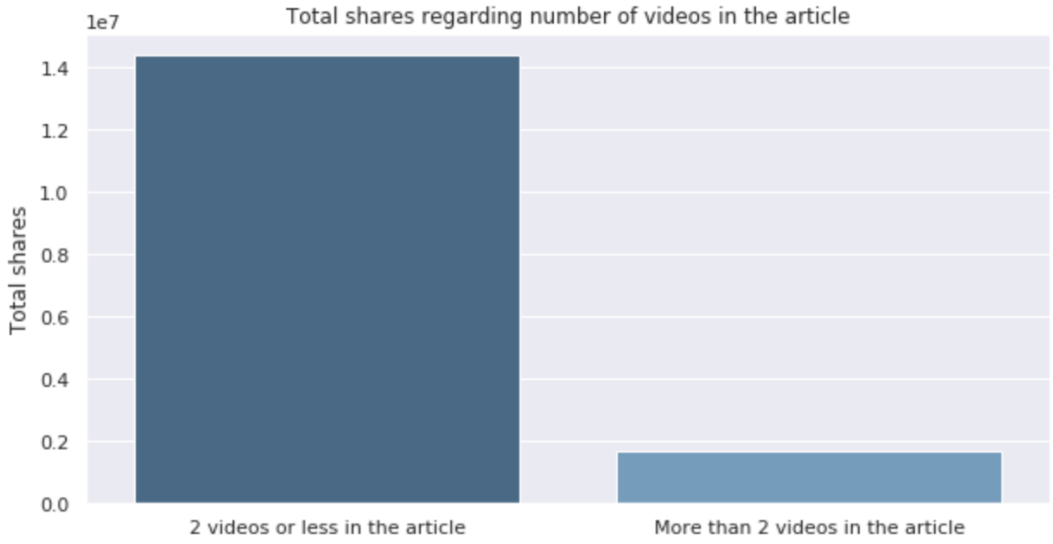
```
In [37]: X_test.groupby(['category_0.0','category_1.0','category_2.0','category_3.0','category_4.0','category_5.0']).size()
```

```
Out[37]: category_0.0  category_1.0  category_2.0  category_3.0  category_4.0  category_5.0
         0             0             0             0             1             1739
                                                                   0              396
                                                   1             0             0   129
                                     1             0             0             0   305
                       1             0             0             0             0   331
         1             0             0             0             0             0   100
         dtype: int64
```



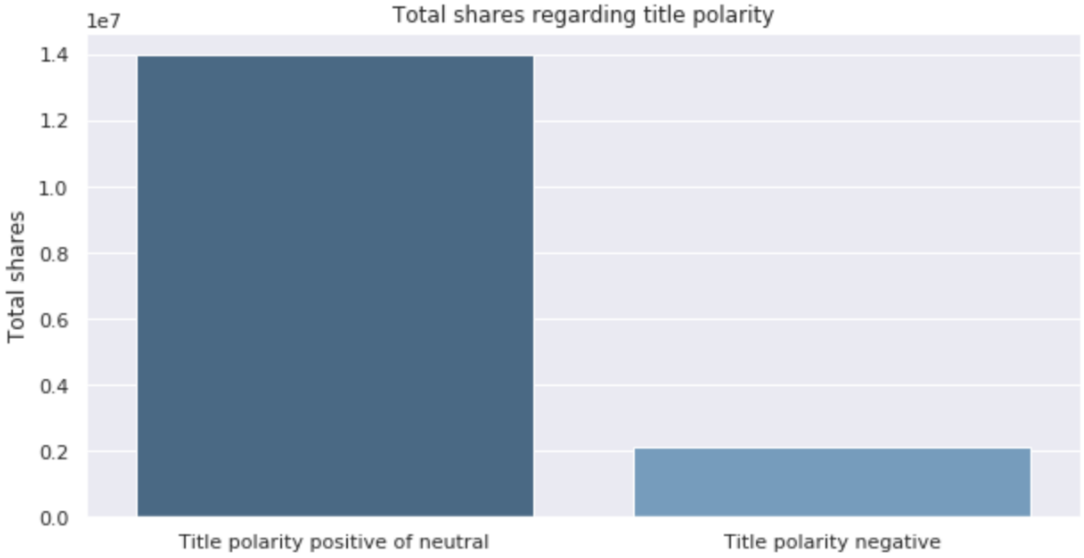


— If we have a closer look to the number of videos, we can see that the extreme majority or aticles and the total shares contains 2 videos or less in them. I decided to create a binary feature '2 videos at most in the article' to illustrate this insight (fig.6).



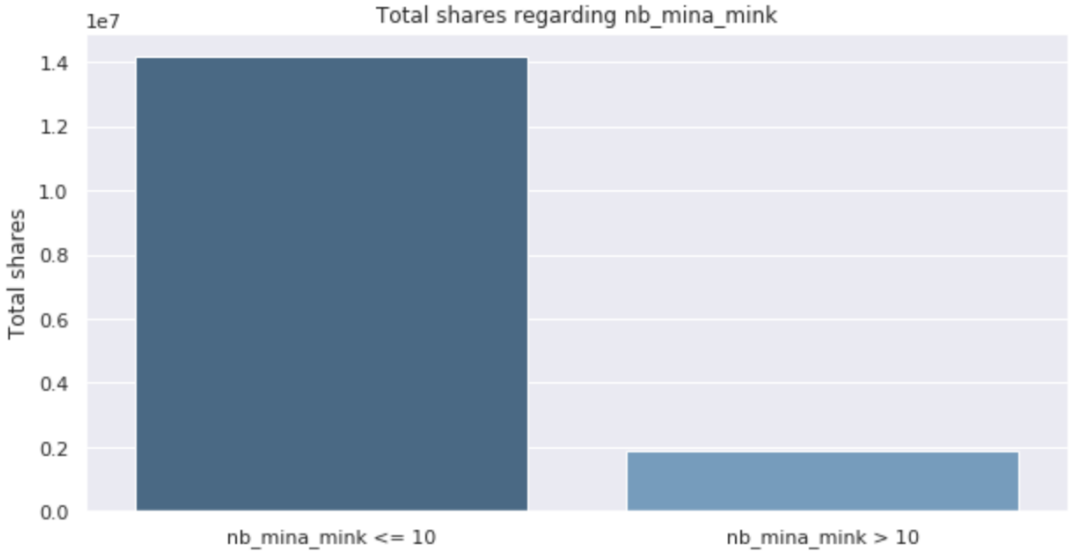
```
In [44]: X_train.groupby('nb_videos').size()

Out[44]: nb_videos
         0.0     3154
         1.0     1189
         2.0      286
         3.0       62
         4.0       33
         5.0       25
         6.0       12
         7.0       12
         8.0       13
         9.0       13
         10.0      26
         11.0      30
         12.0      10
         13.0      10
         14.0      10
         15.0      10
         16.0      12
         17.0       5
         18.0       6
         20.0      10
         21.0      25
         22.0       1
         23.0       4
         24.0       2
         25.0       8
         26.0      22
         27.0       1
         28.0       2
         31.0       1
         33.0       2
         38.0       1
         58.0       1
         65.0       1
         75.0       1
         dtype: int64
```

— (fig.8) binarization of title polarity : positive or neutral / negative

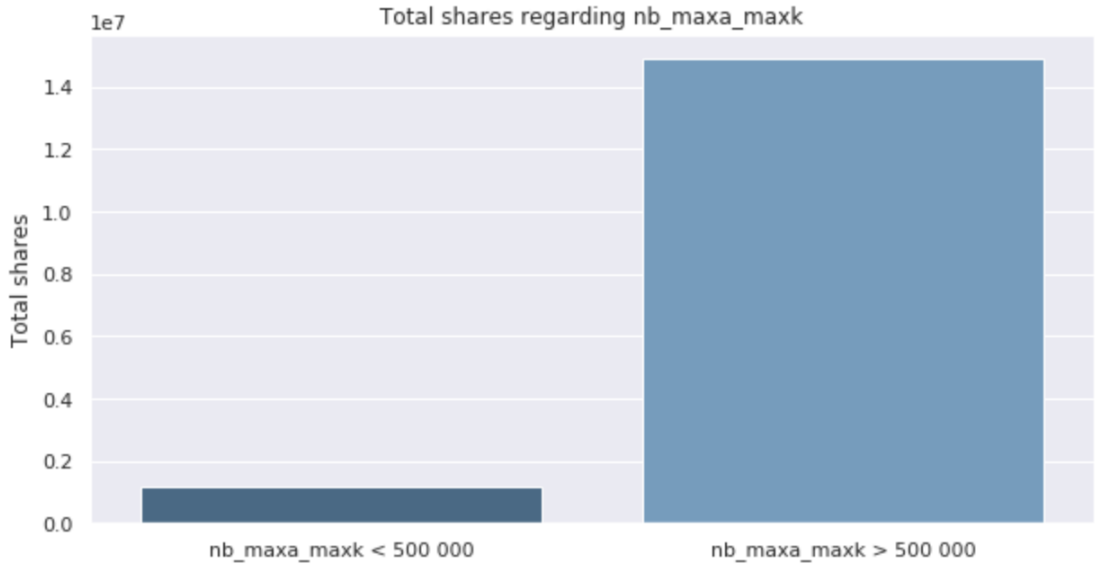


— binarization of average word length : more of less than 5

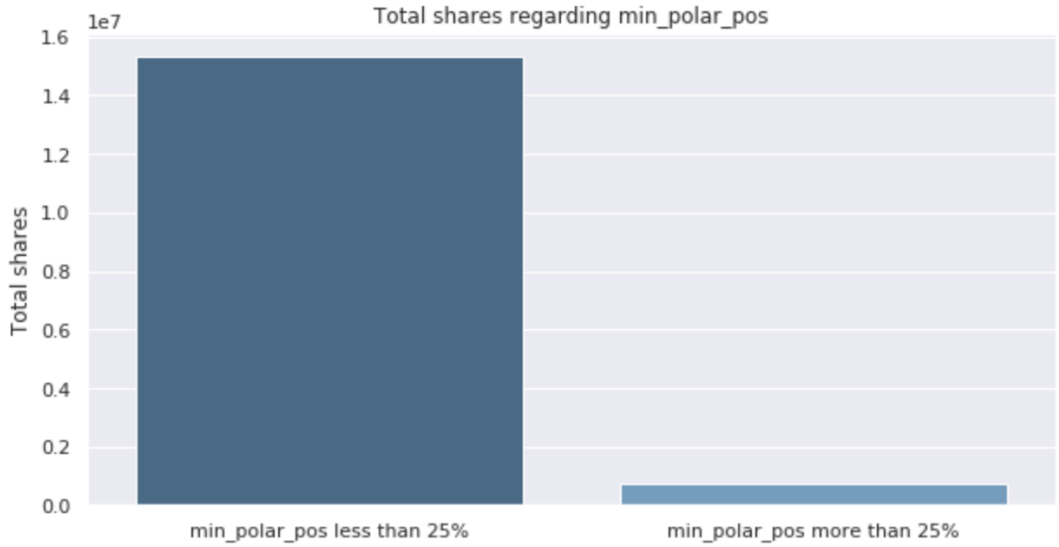
— (fig.9) binarization of nb_mina_mink : more of less than 10



— (fig.10) binarization of nb_maxa_maxk : more of less than 500 000



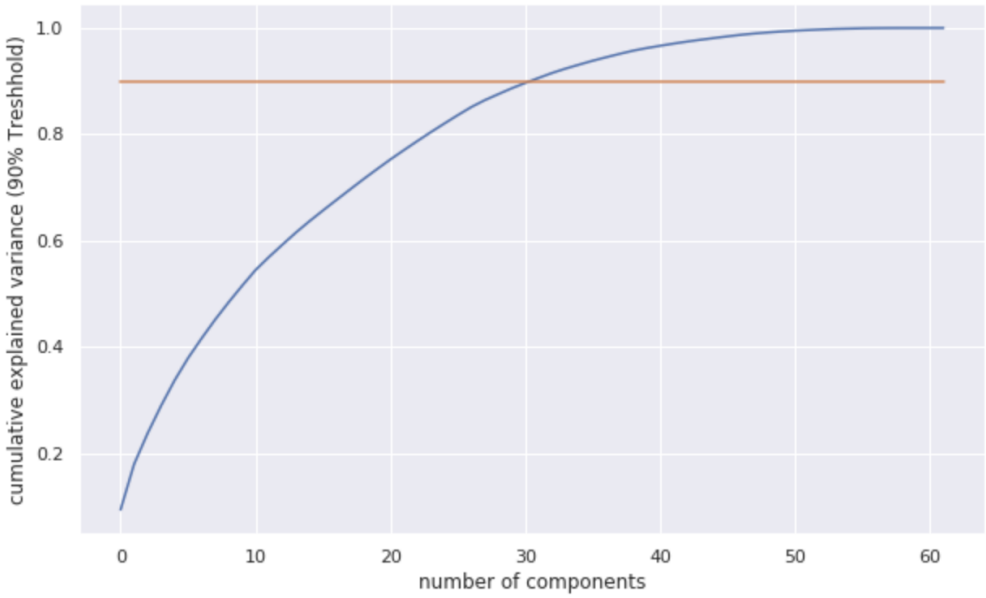
— (fig.11) binarization of min_polar_pos : more of less than 25%



— (fig.14)I decided to standardize my data, to have a 0 mean and a standard deviation of 1. I used the PowerTransform scaler of sklearn. We apply a power transform featurewise to make data more Gaussian-like with the Yeo-Johnson method which supports both positive or negative data. By default, zero-mean, unit-variance normalization is applied to the transformed data.

```
In [83]: from sklearn import preprocessing

         # Create a standardizer object and fit it to the training data.
         scaler = preprocessing.PowerTransformer().fit(X_train_scaled)

         # Apply the standardization to the training and the test data.
         X_train_scaled = scaler.transform(X_train_scaled)
         X_test_scaled = scaler.transform(X_test_scaled)

/opt/tljh/user/lib/python3.6/site-packages/numpy/core/_methods.py:121: RuntimeWarning: overflow encountered in multip
ly
  x = um.multiply(x, x, out=x)
/opt/tljh/user/lib/python3.6/site-packages/numpy/core/_methods.py:122: RuntimeWarning: overflow encountered in reduce
  ret = umr_sum(x, axis, dtype, out, keepdims)

In [84]: print('mean: {}'.format(X_train_scaled.mean(axis=0)))
         print('std: {}'.format(X_train_scaled.std(axis=0)))

mean: [ 9.09494702e-17  9.04520903e-16 -5.89750471e-16 -1.09423581e-16
  1.70530257e-17 -2.55795385e-16  2.44426701e-16 -7.95807864e-17
 -1.40687462e-16 -6.67910172e-17 -1.14397380e-16 -3.90798505e-16
  7.10542736e-19  3.69482223e-17  4.97379915e-17  5.89750471e-17
 -2.36610731e-16  7.17648163e-17  4.76063633e-17 -2.38742359e-16
 -8.07887091e-16  2.39808173e-16  2.20268248e-17 -1.76925141e-16
 -4.97379915e-17 -2.27373675e-17  3.01980663e-17  1.35713663e-16
 -3.55271368e-18 -4.79793982e-16  1.82076576e-16  1.05160325e-16
 -1.56319402e-17  2.29505304e-16 -4.40536496e-17  4.93827201e-17
  5.40012479e-17 -2.23110419e-16  3.37152528e-16 -1.77280413e-16
 -1.82254212e-16 -1.13686838e-17  1.70530257e-17  3.97903932e-17
 -3.23296945e-17 -3.12638804e-17 -2.70006240e-17 -8.02913291e-17
 -1.10134124e-17  2.84217094e-18 -9.23705556e-18 -1.17239551e-16
  6.03961325e-17 -5.68434189e-17  9.94759830e-18 -2.55795385e-17
  2.61124455e-16  3.80282472e-15  1.17950094e-16 -3.29691829e-16
 -4.87432317e-16 -7.70938868e-16]
std: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

In [85]: print('mean: {}'.format(X_test_scaled.mean(axis=0)))
         print('std: {}'.format(X_test_scaled.std(axis=0)))
```

— I run a PCA decomposition to reduce the dimensionality of my features (initially 62). Regarding the cumulative explained variance, I prefered to keep 90% of the variance explained in the training set, which lead us to keep 32 features.(fig.13) Number of features has been reduced by half.



4 Models explored and their performance on the validation set

I will explore several machine learning algorithms from the different families and get the time of hyperparameters optimization, the time of fitting the training set, and the performance on the public validation set on Kaggle. I decided to explore the following algorithms :

- KNN
- Ensemble methods :
  - Random Forest
  - Adaboost with decision tree regressor as base estimator
  - XGBoost
- Linear models
  - Lasso regression
  - Ridge regression
  - Bayesian Ridge regression
- SVM regression
- Deep Learning : MultiLayer Perceptron MLP and Convolutional Neural Nets CNN

For each model, I explore the hyperparameters and their meaning and importance for our model. I then run a GridSearch cross validation to find the optimal hyperparameters. I define a grid with all the parameters to optimize and different possible values (generally with a power 10 step). The scoring metric chosen is the mean squared error (I wanted to take the log like in kaggle but the features contains negative values not accepted). The cross validation uses 10 folds. I set n_jobs to -1 to use all the cores that I have at disposal (16). I usually run two to three gridsearches and reduce the intervals at each time to find the best hyperparameters. I measure the time for optimization, the time for fitting the training set, and check the score on Kaggle on the public validation set.

Results are summed up on the table below :

Out[129]:

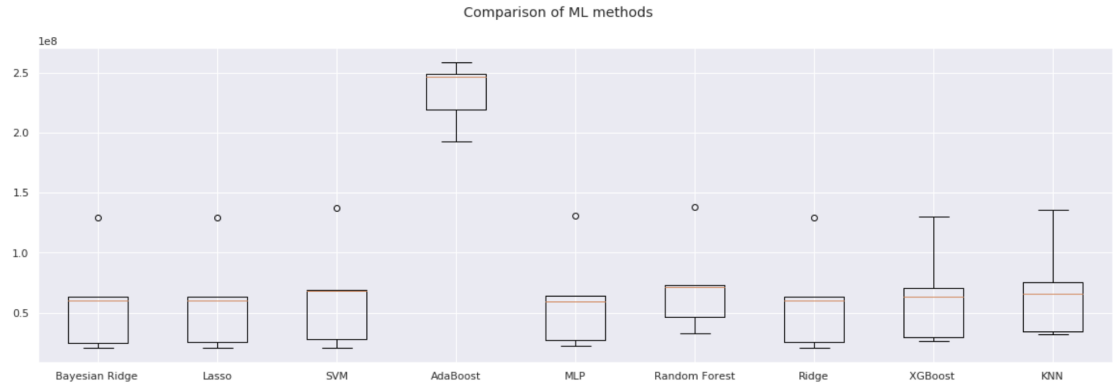
	Optimization time (s)	Fit time (s)	Validation score on Public board (RMSLE)
CNN	248.149073	0.226897	0.86561
KNN	4.534795	0.028484	0.94571
XGBOOST	40.683350	0.188223	0.88563
Adaboost	226.732800	21.569312	0.90370
Random Forest	58.142441	4.443001	0.99948
MLP	33.510419	0.924138	0.85891
LASSO	2.507907	0.011343	0.99126
Bayesian Ridge	14.746755	0.059647	1.01341
Ridge	1.480207	0.014011	1.05879
SVM	332.590388	5.685370	0.90596

(I will talk about CNN and linear models in the section 'additional models tested below).
The multilayer perceptron seems to give the best performance with a great time efficiency. The MLP explored was with Keras not Scikit Learn. It consists of two dense layers of 30 neurons each. The batch size chosen is 100. I chosed to put a large number of epochs and set an earlystopping according to mean squared logarithmic error MSLE scoring metric. Relu activation function gives the best results and avoids Gradient vanishing. I allowed Shuffling to avoid overfitting (the model won't learn the order). A 25% Dropout layer was added to also avoid overfitting. I used the Adam optimizer with a learning rate of 0.01. Learning rate always impacts the outcome of training. For model optimization I used again the Gridsearch from Scikit learn by importing from keras.wrappers.scikit_learn the KerasRegressor and run the classic sk GridSearchCV. The number of neurons is always somekind of lottery, (30,30) seems to give great accuracy, so I tuned the rest of hyperparameters according to that.
The K Nearest neighbours method is very fast in terms of optimization and fitting but is not very accurate.
Ensemble methods (XGBoost, Adaboost with decision tree as base estimator, and Random Forest) give globally the same performance around 0.9, but XGBoost is the best among these ensemble methods regarding time efficiency and random forest method is the worst in term of accuracy. The difficulty here remains finding the right number of estimators and the maximum depth.

5 Cross-Validated performance

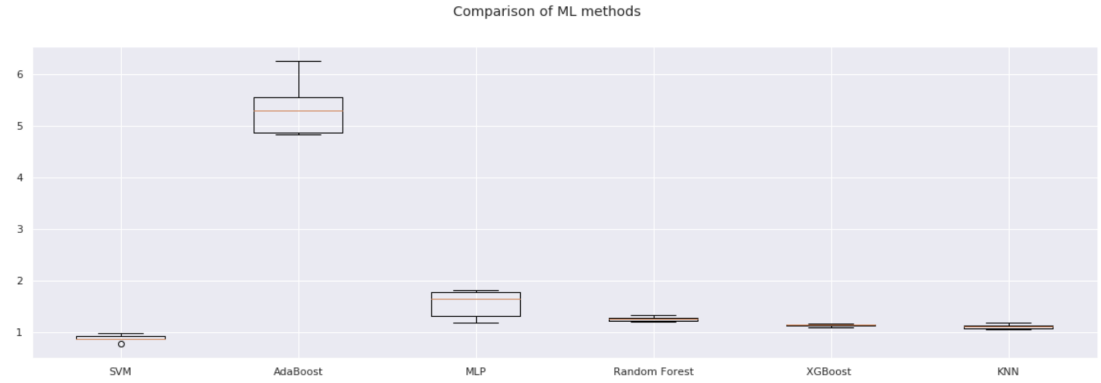
I set up a cross-validation framework for model selection. All models mentioned above are added to the framework. The scoring metric is the mean squared error. I used scikit learn Kfold method (and not the stratified Kfolds because it's a regression problem not a classification one). Number of folds is set to 5. The results are basically the mean and the standard deviation. The results are shown on a boxplot.

Please note that the MLP results below concern the scikit learn MLPRegressor not the one that I used with Keras(it explains its bad results..) I first run the cross validation framework on all "naked" models, which mean with no parameters tuning. We can see that Adaboost is clearly considered as the worse model. Most of the models contain outliers (mainly linear models and SVM). Data is clearly skewed. I cannot really make a preference from this first "naked" cross validation.



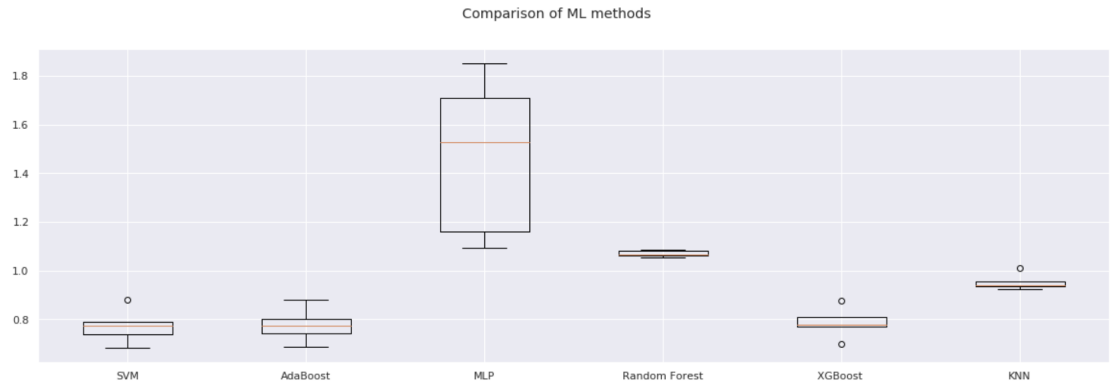
Below we can find the results of the cross validation framework on still on naked models, but here I removed the linear models so that I can use the MSLE scoring metric (no negative values). Adaboost is still the worst model. SVM seems to be the best model. All the others giving more or less the same performance above 1. Note again that the MLP here is the scikit learn one not the Keras one that i finally used.

```
SVM 0.874363 0.071775
AdaBoost 5.361984 0.523836
MLP 1.542764 0.257818
Random Forest 1.254849 0.040296
XGBoost  1.128995 0.021421
KNN 1.102084 0.046369
```



Below we can find the final cross validation framework with the tuned models. I again removed the linear models to use the MSLE. Results are clearly different once we optimize

the hyperparameters of all the models. I expect SVM, Adaboost (wich was the worse), and XGBoost to perform the most. Random Forest and KNN give poor results.




```
SVM 0.772658 0.065763
AdaBoost 0.777202 0.064503
MLP 1.468521 0.297982
Random Forest 1.070134 0.012719
XGBoost  0.786763 0.057163
KNN 0.952821 0.030943
```

When I compare these expectations to the final results on the public validation set (please have a look at the table of section 4), the expectations are the same. I mean that SVM, Adaboost and XGBoost give very similar results (0.9 in Kaggle vs 0.78 expected). KNN (0.95 both expected and in reality) and random forest (0.999 vs 1.07 expected) follow in the same order than expected.
But I have seen some additional models that sometimes perfomed better, and sometimes worse.

6 Additional models tried

As you have seen, I have looked at different linear machine learning models (Lasso, Ridge, Bayesian Ridge). For all of them, I obtained negative predicted values which is understandable from linear models dealing with non linear data. To get their performance on Kaggle, I have set all the negative values to zero, which maybe can explain the bad results for these type of models. We also have to notice that they all are very fast to train and optimize despite less accuracy.

I wanted to explore some Deep Learning models. So I have run, as you read above the Keras MLP which is our winner of this work since it has delivered the best performance in great timing.
I also explored the convolutional neural networks. I used a CNN made by 2 Conv1D layers of 16 and 32 depth and 10x10 kernel size. MaxPooling and Dropout layers (0.1 as dropout

rate) were also added. It also gave remarkable performance on public validation set but needed way more time to compute. Deep learning models perform better but are really hard to design since I wasn't able to find the best combination of number of layers and number of neurons in each layer.
I choose them because they more able to catch non linearities, I still have big amount of data and a lot of features, and DL performs better when the amount of data is bigger.

There are still parts of uncertainty in this work alongside DL and ML parameters. I think about the dimensionality reduction to drop half of the features using PCA. PCA decomposition is a linear dimensionality reduction so maybe it is not the best thing to do. Also the choice of the number of features is not so obvious regarding the explained variance curve, and I preferred to choose 90% of explained variance in the training set.

7 Final models & results on the private validation set

I will present here the final results of all the models explored in the private validation set in Kaggle.
Definitely, all results expected from the cross validation results and from the public validation set performances are respected.
My two winners are still CNN and MLP.
XGBoost, Adaboost and SVM give similar scores.
Linear models are really bad(above 1). The order between the models is respected and maintained : MLP, CNN, (SVM/Adabbost/XGBoost), KNN, Random Forest, Linear models.

Out[131]:

	Validation score on Private board (RMSLE)
CNN	0.87462
KNN	0.96009
XGBOOST	0.89800
Adaboost	0.88954
Random Forest	1.03881
MLP	0.87438
LASSO	1.03210
Bayesian Ridge	1.11563
Ridge	1.30869
SVM	0.88598

#KNN