École Polytechnique de Louvain
Faculté de l'UCLouvain

LELEC2870 - Machine learning

Academic year 2020 – 2021

Project 1 Machine learning

Cools Hadrien - 04001700
Vanden Bulcke Colin - 21651600

December 20, 2020

1 Purpose

The project aims to determine the number of sharing of certain articles representing the image of visibility and attractiveness of certain posts on the internet.
In order to be able to exploit these data, several parameters are acquired to characterize each of these posts (each post is thus characterized by several features that are more or less relevant).
The goal of the project will be to implement several linear and non-linear models that will try to predict the number of shares of an article and to compare them by implementing some metrics.

Here are some words about the implementation. The code contains a lot of different tests, as you will see in the rest of this report, and thus the running time is pretty big (about 1 and a half hour) if you want to make all the tests in one run. Therefore, some booleans at the start of the code can manage which section you want to run. The first boolean *figure* allows the code to plot the figures that you can see in this report. Then, a boolean per section allows you to control whether or not you want this section to run, thus these tests to run (e.g. reg_test which control the regression tests section).

2 Preprocessing

The first step before implementing and processing data is to see the data, so we will first check the data and then we can try to understand it manually in order to be able to move towards better-adapted processing.

2.1 Covariance and data visualization

A good idea would be to look at the covariance between all the data, then we will see the data correlated with each other, another idea to see the correlation between the data would be to make tests of association between the data and the answer, for example, Chi-square tests.)
The idea will be to remove the correlated data first, this will be done using a more accurate PCA in the next step.

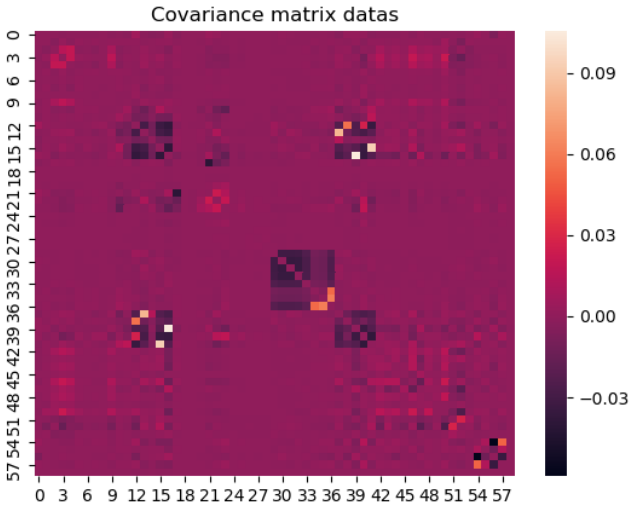

Figure 1: The correlation of the identity corresponding to the autocorrelation has been removed, the correlated features will be deleted)

2.2 Feature selection using PCA

Once the visual inspection is done, the data is normalized, we tested several different normalizations between 0 and 1 or centred in 0 according to a distribution in X, this does not fundamentally change the results. The main purpose of standardization is to help convergence and calculations by putting everything on the same scale, the disadvantage is a partial loss of information.

Once our data is readable we will try to reduce the number of features and to take the most relevant ones, for this we must select the classes giving the most information, a well-known algorithm allows us to do this step easily, it is the PCA, the PCA will help us to decompose the information into main components and allow us to choose the components bringing the most variance, that is to say bringing the most information. We therefore apply a BCP on our standardized data and we select the first N components until we reach a certain variance threshold.
It is possible to visualize the covariance between the information by making a covariance matrix (and by setting to 0 the autocovariance avoiding the effect of overwriting the other values) to see the link between them, technically the highly correlated information is redundant and can therefore be discarded.
Attention it is possible that the essential information is found in the components representing the remaining 5% variance, even if it is unlikely, the fact of choosing the main components mainly allows to reduce the number of dimensions and thus to drastically reduce the complexity at low cost.

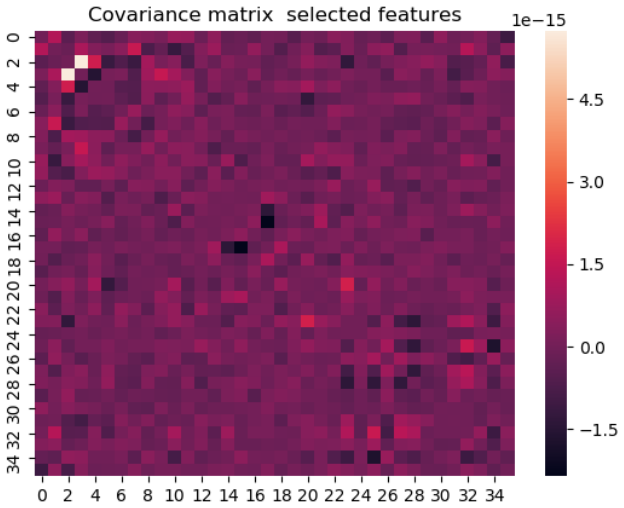

Figure 2: There are 22 features left and we can see the covariance)

On figure 2 we see that the covariances are low, which is normal because we discard the variables that do not bring any/low additional information. We have kept 36 of them, which are the data that will be sent in the different models.

3 Model

3.1 Metric

In order to evaluate our models, we will base ourselves mainly on the accuracy, it would also be possible to base ourselves on sensitivity or specificity to characterize the different algorithms and see if they are applicable to the field of application.
In our case the accuracy seems to be sufficient, another useful information can be the training time of the models, it is, however, variable according to the complexity and the linear and non-linear character of the algorithms.
To assess the accuracy of our models, It was decided to use a technique called Kfolds, the procedure is to separate the dataset into N folds using one fold for data validation after training and the n-1 folds to train the model. This algorithm can be iterated n times to obtain the accuracy for each fold used as validation data. The average of these accuracies gives us a better estimation of the performance of our model while mitigating the randomness of data validation.
There are other methods, such as the leave-out-one, which, when we have little data, allows us to simply keep only one data for the validation, the performance will then be calculated statistically unitarily. But since we have enough data, Kfolds cross-validation seems like a better option. In our case kfolds with n = 4 (so 4 folds) seems to be correct because we have a lot of data. However, taking a bigger number of folds increases drastically the running time.

3.2 Regression models

A regression model tries to predict the value of the Y variable given know values of the X variable. These kinds of model are perfect to use for such an application as predicting the number of shares of an article. The rest of this section will present the different regression models that have been used.

Linear regression To implement this model, we used the *LinearRegression* from the *scikit-learn* library from python. This is an ordinary least squares linear regression model and this is one of the simplest model that can be used to make regression on data. Therfore, its running time is low and there is not many parameters to tune. The default settings were used because these settings suited perfectly our data.

Lasso regression To implement this model, we used the *Lasso* from the *scikit-learn* library from python. This model is a linear model trained with a prior as regularizer (lasso). It is a little more complex than the previous linear regression model and thus has more parameters to tune to increase the performance of the machine. Nevertheless, the default parameters were the ones that gave the more accuracy.

KNN To implement this model, we used the *KNeighborsRegressor* from the *scikit-learn* library from python. This model regression is based on the k-nearest neighbors algorithm which is a non-parametric method. One important parameter to be tuned for this model is the number of neighbors to use for kneighbors queries. We observed, that setting this parameter to 7 gave us the best accuracy. An other important parameter is the weights of the different neighbors in a query. It can either be 'uniform', i.e. all the neighbors have the same weight in the result, or it can set to 'distance'. In that last case, the weight for each neighbor in a query is set to be the inverse of their distance. The model gains a little bit of accuracy when the weights parameter is set to 'distance'.

MLP To implement this model, we used the *MLPRegressor* from the *scikit-learn* library from python. This model uses multi-layer perceptron to build its regression on the data. There is a lot of different parameters that can be tuned, but the default ones seem to suit our data. No real significant improvements came with other parameters.

Decision Tree To implement this model, we used the *DecisionTreeRegressor* from the *scikit-learn* library from python. This regression model uses a decision tree as regressor. A decision tree is a structure in which each node represents a test on an attribute on the data, each branch represents the outcome of a test and the leaves represents the regressor. For this model, the default parameters suited this application.

SVM To implement this model, we used the *SVR* from the *scikit-learn* library from python. This model is an epsilon-Support Vector Regression. Once more, the default parameters of this model produced the strongest accuracy.

Results Figure 3 shows the accuracy of the different models presented above. All models have an accuracy below 50%. The first 4 models used have similar performance, varying around 48 to 49%. The strongest model is the KNN regression model that is accurate 49.16% of the time. Then, we can also observe on this figure that the two last models have lower accuracy: 46% for the decision tree and 41% for the SVM model. This can be explained by the fact that these two models are more designed to classify data than to perform regression.

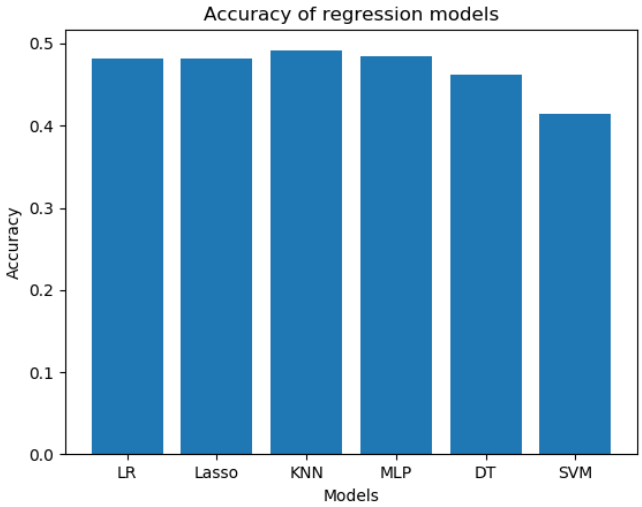

Figure 3: Accuracy of regression models

3.3 Classification models

In this application, instead of wanting to predict the exact number of shares that an article will have, another point of view on this question would be to place the article in a certain class which would represent the range of shares. For example, the article with less than 500 shares could be all associated in one class. This idea comes from the way we score the regression model. In fact, we check to see if the number of shares predicted by one of the regression models is in the same range as its true value. Therefore, the articles are separated into 5 classes: flop ($shares < 500$), mild success ($shares \in [500; 1400[$), success ($shares \in [1400; 5000[$), great success($shares \in [5000; 10000[$), viral hit ($shares > 10000$). Therefore, by changing the target dataset Y1 from the number of shares to a number corresponding to one of this class (0 to 4), classification algorithm can now be used to maybe have better results. The same algorithm as for the regression has been used to classify the data, except for the linear regression and lasso regression which have not a similar model to classify data.

KNN To implement this model, we used the *KNeighborsClassifier* from the *scikit-learn* library from python. This model uses the k-nearest neighbors algorithm to classify data. The same parameters were tuned as for the regression. The weights parameter has been set to 'distance' and the number of neighbors has been set to 30. These set of parameters produces the strongest accuracy.

MLP To implement this model, we used the *MLPClassifier* from the *scikit-learn* library from python. As for the regression, this model uses a multi-layer perceptron to classify the data. The default parameters were adapted to this application.

Decision Tree To implement this model, we used the *DecisionTreeClassifier* from the *scikit-learn* library from python. A decision tree is used to classify the data and the default parameters suited this application.

SVM To implement this model, we used the *SVC* from the *scikit-learn* library from python. This model uses support vector to classify the data. Once more, the default parameters produces the best accuracy.

Results Figure 4 shows the results for the classification algorithms, which are more disparate. Except for the decision tree, all the models have better performance when they are used as a classifier. Especially the SVM model for which it reaches an accuracy of 54%.

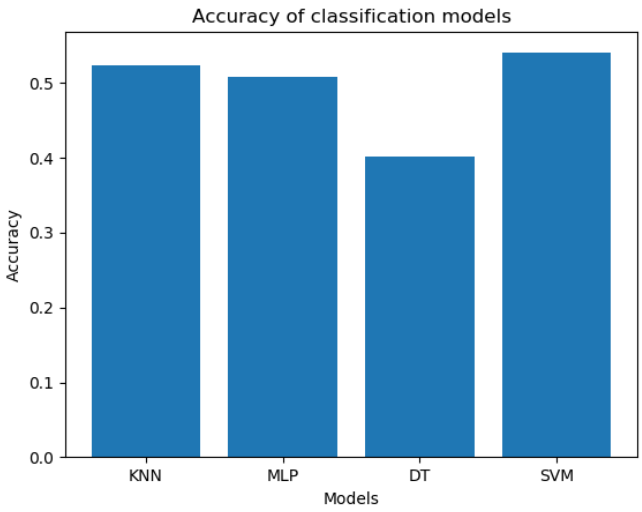

Figure 4: Accuracy of classification models

One problem that we observe for this classification was that the dataset was imbalanced, i.e. there was a huge difference in the number of training data in each class. This can be seen in Figure 5. This problem can bias the model. To correct that, one solution is to modify the dataset by undersampling or oversampling it. Undersampling means removing data from the over-represented class to better balance the dataset while oversampling means creating new synthetic data in the under-represented class. In this case, we chose no undersample our dataset to avoid losing data and oversampling required using non-standard python library. Thus, we did not try to modify the number of data to work with.

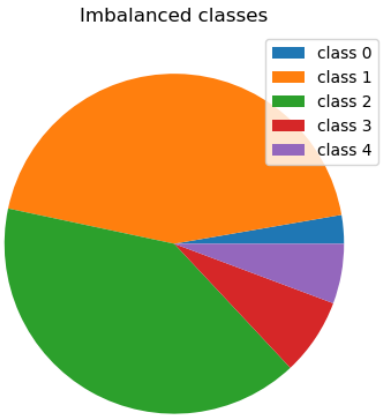

Figure 5: Imbalanced classes

4 Improvements

In this section, we will present the different method that we use to try to improve the accuracy of the machine in order to have the best results possible. First of all, we try to use ensemble learning. Ensemble learning is the art of combining several individual models to enhance the predictive power of the model. To do so, we used some commonly used ensemble learning techniques: Bagging and Boosting. Bagging tries to implement similar learners on small sample populations and then takes a mean of all the predictions. Boosting is an iterative technique which adjusts the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. Here, we used the *GradientBoostingRegressor* and the *BaggingClassifier* from *scikit-learn* library from python. The Boosting technique was used to try to improve the regression problem while the Bagging technique was used for the classification problem. Unfortunately, neither of these techniques improved the results, on the contrary, as it can be seen in Figure 6.

The last idea of improvement that we had is to combine the best classification algorithm with the best regression model. For the classification algorithm, each class for the target dataset corresponds to a range of shares (e.g. 0 corresponds to shares below 500). Therefore, the idea was to use the KNN regression model and the SVM classifier at the same time and then compare the prediction. If the number of shares predicted by the regression was in the same range as the class predicted by the classifier, nothing else was done. On the other hand, if the number of shares was not in the same class predicted, then the prediction was updated to a weighted mean between the number of shares predicted by the regression and the centre of the class predicted by the classifier. The weights corresponding to the accuracy of each model. Figure 6 shows the accuracy of the improved model explained in this section. Unfortunately, no ideas of improvements actually have a positive effect.

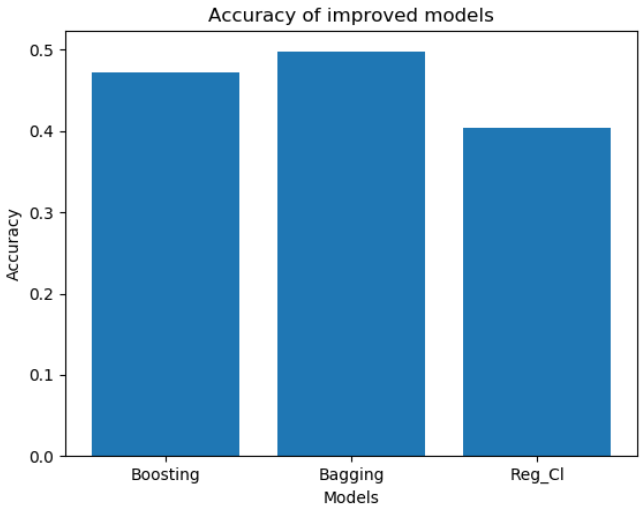

Figure 6: Accuracy of improved models

5 Discussion

This section will be a small discussion on the different models and tests that we perform. The goal of this section is to choose which model we are going to use to predict the number of shares of the articles present

in the "X2.csv" file. Given that we need to predict a specific number of shares for an article and not a class, we decided to use the KNN model regressor, which is the best model for regression, to predict the data from the "X2.csv" file.

However, we can observe that for this kind of application, the use of a classifier and classes for the data may be better to make prediction. Indeed, at the end the use of classes for different range of shares is still good to have an idea of the popularity of the articles.