Pre-processing : SOLTANI Hicham et LIU Yongjie
Model : CHEN Shuangrui et XU Zhenghai
Visualization : YACHOUTI Mouad et Alcantara Hernandez Ursula

Project Proposal

L2 MI – Mini Projet - Xporters - SEGWAY

Pre-processing
Model
Visualization

universite PARIS-SACLAY

15/03/2020

Contents

1. Background and description of the problem

We received a request from a young and disruptive entrepreneur, he has just acquired a small lemonade stand located next to an highway. The previous owner of the stand told him: "I noticed that, on average, 1 car out of 100 stops at my stand".

During the last years, he also kept track of the hourly number of cars that used this highway, our mission is to predict the number of cars that will pass by at a given date, hour, and additional meteorological information in order to avoid waste of lemons.

To do this, we would like to use these data to train a model that predicts the traffic volume. We also found the hourly weather forecast records of the last years.

The data set contains 58 features and the solution is the number of cars registered in an hour ranging from 0 to 7280.

2. Pre-processing

Pre-processing consists of preparing data for the Regressor.
We select and modify the data, in order to make the Regressor more efficient.
We optimize reading of the most relevant data which will allow a better training and therefore better results.
First of all, our algorithm will delete the data which has the least influence on the final result, i.e. those which differ significantly from the others (data indicate the most unusual situation).
The objective of pre-processing will be to analyse every feature and to notice that some features will not affect the results. This has the advantage of avoiding noise-related errors by deleting unnecessary data.
So, we used 6 methods of Pre-processing, these are Shuffle data, Remove Outliers, Scaler, PCA, SVD, TSNE.

2.1 Algorithms

2.1.1 Shuffle data

We leave shuffle = data.copy (), to throw confusion in index, we use np.random.permutation, and we extract the last result; because the last result is the conclusion of all the data that is analyzed.

2.1.2 Remove Outliers

We call Outliers which is the non-correlation data, in order to get the last result for the same reason.
Some external and / or inconsistent data are said to be outliers. Our goal is to detect them and get rid of them in order to have a more valuable and more standard data-set. To do this we used a value filter method: outliers, which allows to keep the selected values in our data.

2.1.3 Scaler

It consist of normalize the data by subtracting the mean, then dividing by the variance (or standard deviation). This method of data normalization (StandardScaler) after processing the data conforms to the standard normal distribution, that is, the mean is 0, the standard deviation is 1.

Applies to: If the data distribution itself obeys the normal distribution, this method can be used.
The standard score of a sample x is calculated as:

$$z = (x - u)/s$$

where u is the mean of the training samples or zero if $with\_mean = False$, and s is the standard deviation of the training samples or one if $with\_std = False$.
Usually, this method can be used mainly when there are outliers, but outliers will always affect the operation when calculating the variance and the mean. Therefore, in the outlier case, the distribution of the characteristics of the converted numbers can be completely different.
So here we use $StandardScaler().Fit_transform(outliers)$ to Improve the accuracy of the classifier.

2.1.4 PCA

Next, we use the Principal component analysis (PCA) algorithm. This algorithm allows us to focus on a fragment of our data.
A matrix compression algorithm which reduces the dimensions of the matrix while preserving the information of the original matrix as much as possible. In simple terms, it converts an $n * m$ matrix into an $n * k$ matrix, with $k < m$, keeping only the main characteristics existing in the matrix, which can save a lot of space and data:
1) Form the original data into n rows and m columns matrix X by column.
2) Zero each line of X (representing an attribute field), i.e. subtract the average of this line.
3) Find the co-variance matrix.
4) Find the eigenvalues of the co-variance matrix and the corresponding eighth-vectors.
5) The characteristic vectors are arranged in a matrix from top to bottom according to the corresponding characteristic value, and the first k lines are taken to form a matrix P.
6) is the data after reduction of dimension in k-dimension.
There are 4 dimensions which is the best under the condition of efficiency.
Using PCA, we concentrate tens of thousands data given on the spelling with 4 dimensions, it gets more readable.

2.1.5 SVD

We use SVD to decompose a linear transformation into two linear transformations, a linear transformation represents rotations and a linear transformation represents stretching.
TruncatedSVD(n_components = 4)

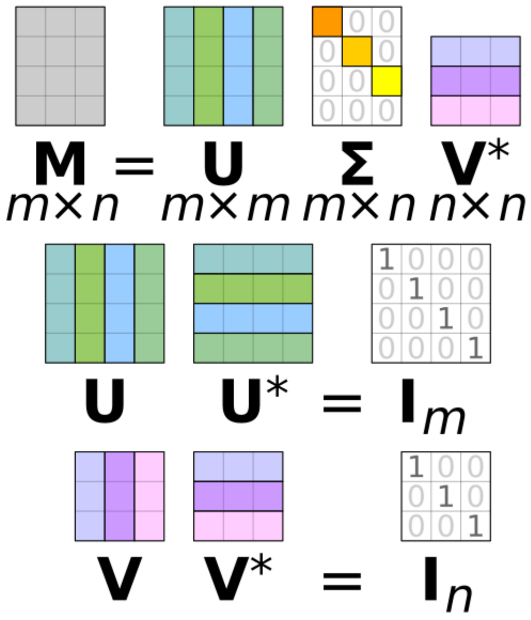


Figure 2.1 – SVD Matrix

2.1.6 TSNE

It is a question of making the matrices symmetric of probability distribution in the high and low dimensions, which can be easily calculated, but there is no improvement on the problem of congestion. The TSNE formula:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)},$$

TSND has a very general problem and takes a lot of time and work. When there are many samples, it is difficult to build the network and the gradient descent is too slow.

Using SVD decomposition will consume a lot of memory and uses many slow operation.

In conclusion, I think PCA is the best method for processing data.

2.2 Conclusion

TSND has a very general problem and takes a lot of time and work. When there are many samples, it is difficult to build the network and the gradient descent is too slow.
Using SVD decomposition will consume a lot of memory and uses many slow operation.
PCA (Principal Component Analysis) is a commonly used data analysis method. PCA transforms the original data into a class of linearly independent representations through linear

transformations, which can be used to extract the main characteristic components of the data. Thanks to SVD's data processing, we can use a much smaller data set to represent the original data set, which actually removes noise and redundant information to achieve the goal of optimizing data and improving results.
SVD is different from PCA: PCA is a matrix decomposition of the data co-variance matrix, while SVD is a matrix decomposition, directly on the original matrix.
To conclude, we think PCA is the best method for this case as there are 58 features to be processed.

3. Model

3.1 Analysis

Model selection We have chosen 8 models, one of them is a Voting Regression which takes opinion from 3 others. As you can see, our best model according to model performances is the Decision Tree Regressor. But after cross-validation, we find it a bit over-fitting (91%) so we decide to use Voting Regressor to neutralize over-fitting by adding the opinion of Random Forest Regressor(similar principle) and Gradient Boosting Regressor(more mathematical for solving such multi features problem). And the result is more satisfying. We gain a 4% improvement in the validation set.

Best model principle The best model we chose is VotingRegressor, it averages the individual predictions to form a final prediction. Which is why we think this might neutralize overfitting of the Decision Tree Regressor.

We think that the reason for the Decision Tree Regressor to overfit might be caused by its own principle, because it tries to form a tree that shows every possible situation, that's why it gains 100% correction in Training set. But we don't want that because weather-traffic problem is not a problem of describing every situation. It's about choosing the most influential feature and judge on that. So that's why we think Gradient Boosting Regressor suits such a problem.

Cross-Valide Cross-Validation makes us reconsider how to choose a model without knowing its best hyper-parameters influenced by the training set. The answer might be that we don't know. Decision Tree Regressor does have the best performance, and it's 100% obviously we can't expect that for Validation set even test set. And our best hyperparameter might not fit the validation set or even for the future usage. So even though we have GridSearch and other APIs that provide us a easier way, we might need a better way to find best model and best hyper-parameters.

Further improvement The range of parameters we use when we looks for the best parameters doesn't necessarily include best hyper-parameters. And it suits only the training set, how do we make sure it's best for all sets?

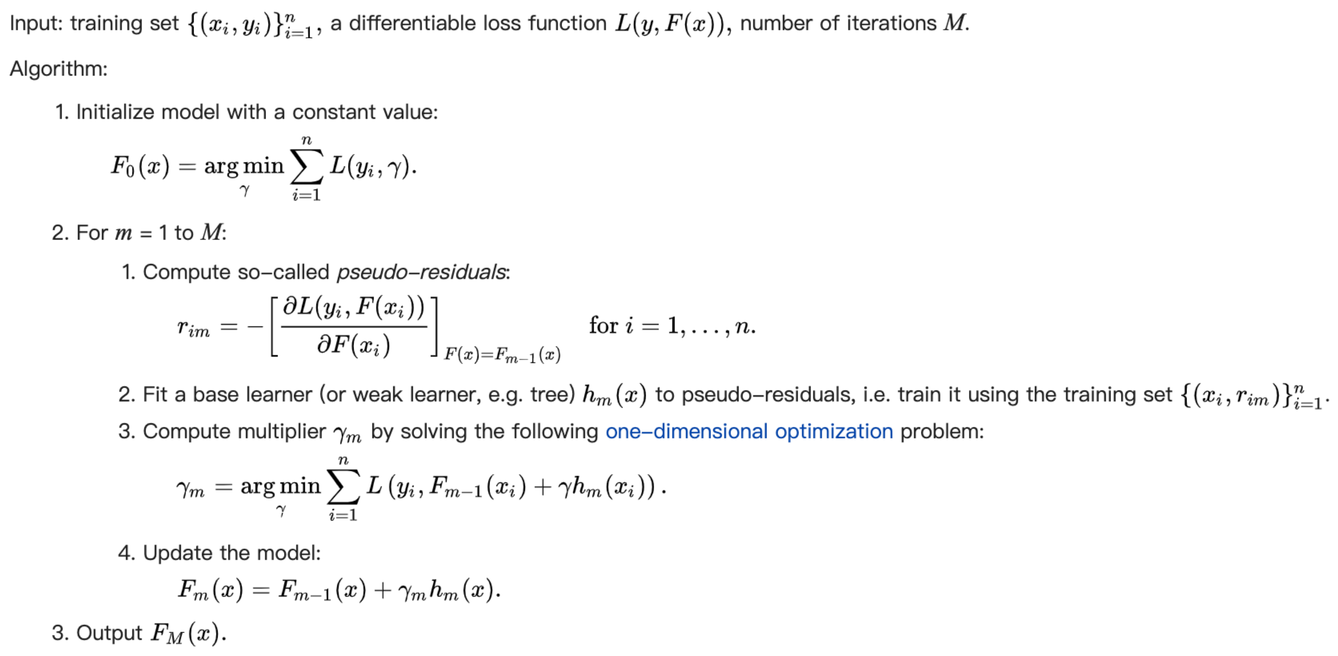
By analyzing the data, there are many irrelevant variables that we have not excluded. If we can filter the variables first, we might have more options on models and parameters.

3.2 Models Explanation

Here we dig in some articles from researchers or Wikipedia. We have summarized some of them by ourselves, we have quoted some of them from researchers, we have put our resources at the end.

3.2.1 Gradient Boosting Model

Gradient Boosting is a Boosting method that each time the model try to converge to the direction of the gradient descent of the loss function. The loss function is used to evaluate the performance of the model. The smaller the loss function the better its performance are. If the loss function is continuously reduced, the model can be continuously modified to improve its performance. Such an idea might suit our 58 features data set, in other words 58 variables equation.
The main idea of Gradient Boosting's algorithm is as follows: (From wiki)



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

3.2.2 Classification And Regression Trees (CART)

Quote from a plosone article:
Classification And Regression Trees (CART) algorithm is a classification algorithm for building a decision tree based on Gini's impurity index(indicates how things are properly separated in binary) as splitting standard. CART is a binary tree build by splitting node into two child nodes repeatedly. The algorithm works repeatedly in three steps:
1. Find each feature's best split. For each feature with K different values there exist K-1 possible splits. Find the split, which maximizes the splitting criterion. The resulting set of splits contains best splits .
2. Find the best split for a node. In the best segmentation of step i, find a segmentation that maximizes the segmentation criteria.
3. Split the node using best node split from Step ii and repeat from Step i until stopping criterion is satisfied.
As splitting criterion, we used Gini's impurity index, which is defined for node t as:

$$i(t) = \sum_{i,j} C(i \mid j) p(i|t) p(j \mid t)$$

where $C(i \mid j)$ is cost of misclassifying a class j case as a class i case (in our case $C(i \mid j) = 1$, if $i \neq j$ and $C(i \mid j) = 0$ if $i = j$), $p(i \mid t)(p(j \mid t)$ respectively) is probability of case in class i (j) given that falls into node t.

3.2.3 Random forests

In view of the shortcomings of decision trees that are easy to overfit, random forest uses a voting mechanism of multiple decision trees to improve the decision tree. We assume that random forest uses m decision trees. For a tree, it is obviously not desirable to train m decision trees with a full sample every time. Full sample training ignores the rules of local samples, which is harmful to the generalization ability of the model.
Random forests de-correlate all trees through random interference, so random forests performs better than Bagging. Random Bags are not like Bagging. When constructing each tree, a random sample predictor is used before each node is split. Because in the core idea, the random forest is still the same as the Bagging tree, its variance is reduced. In addition, random forests can consider using a large number of predictors, not only because this method reduces bias, but also the local feature predictor plays an important role in the tree structure.

3.2.4 Stacking Regression

Stacking regression is a set of learning technique that combines multiple regression models through a meta-regressor. Moreover, each base regression model (described above) must use the complete training set during training. The output of each base regression model during the whole learning process is used as meta-feature input for the meta-regressor. Multiple models can be combined by fitting these meta-features.
Further explanation in the figure:

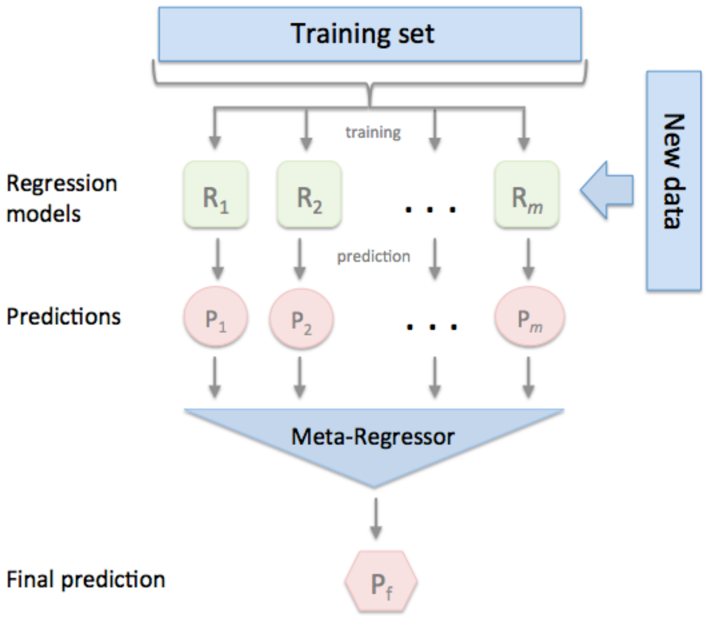


Figure 3.1 – Stacking Regression

3.2.5 Best model

Without doubt, Decision tree Regression is the best model, because it can generate a clear tree structure based on feature selection of different prediction results. But it's also easy to be influenced, because the final decision of the decision tree is usually based on a single condition, we need only to change a few features to get a different detection. Then it is indeed capable in training procedure, for having the ability to describe all possible division of features, it's also easy to get over-fitting results, which makes our model not enough generalizing. But it suits our cases, as we have only 58 features, and dozen of them are quite influential.

3.3 Hyper Parameters Search

GridSearchCV: It goes through all permutations and combinations of the parameters passed in, and returns the highest-scoring parameter in all parameter combinations through cross-validation.
Advantages: automatic parameter adjustment, high parameter accuracy
Disadvantages: It takes huge computing power and calculation time.

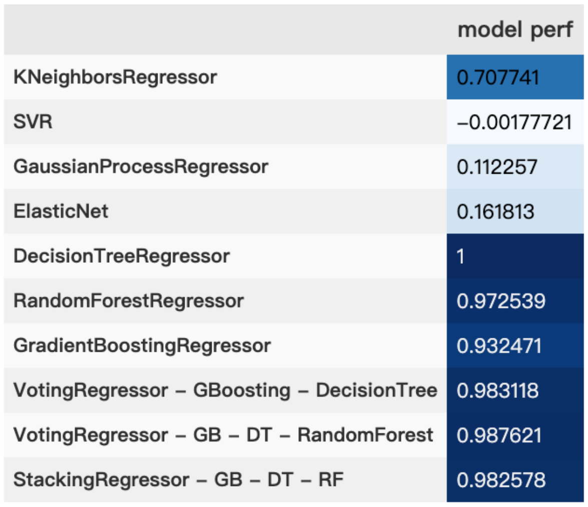
RandomizedSearchCV: The method of using RandomizedSearchCV is actually the same as GridSearchCV, but it replaces GridSearchCV's grid search for parameters by randomly sampling the parameter space. For parameters with continuous variables, RandomizedSearchCV will treat it as a distribution. Sampling this is not possible with grid research, and its search

ability depends on the n_iter (number of training) parameters
Advantages: high operating efficiency, suitable for big data samples
Disadvantages: The accuracy rate is relatively low compared to GridSearchCV

3.4 Explanation of some parameters

max_depth: Limits the maximum depth of the tree, all branches that exceed the set depth are cut off. This is the most widely used pruning parameter and it is very effective at high dimension and with low sample sizes. The decision tree grows one more layer, and the demand for sample size will double. Therefore, limiting the tree depth can effectively limit over-fitting.

n_estimators: It is the maximum number of weak learners. Generally speaking, n_estimators is too small, which is easy to under-fit. If n_estimators is too large, the amount of calculation will be too large.



	model perf
KNeighborsRegressor	0.707741
SVR	−0.00177721
GaussianProcessRegressor	0.112257
ElasticNet	0.161813
DecisionTreeRegressor	1
RandomForestRegressor	0.972539
GradientBoostingRegressor	0.932471
VotingRegressor – GBoosting – DecisionTree	0.983118
VotingRegressor – GB – DT – RandomForest	0.987621
StackingRegressor – GB – DT – RF	0.982578

Figure 3.2 – Models Performance

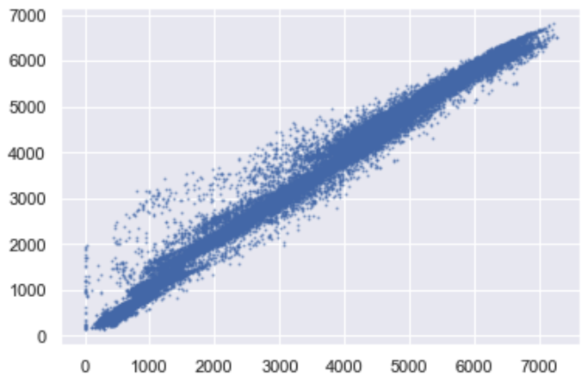
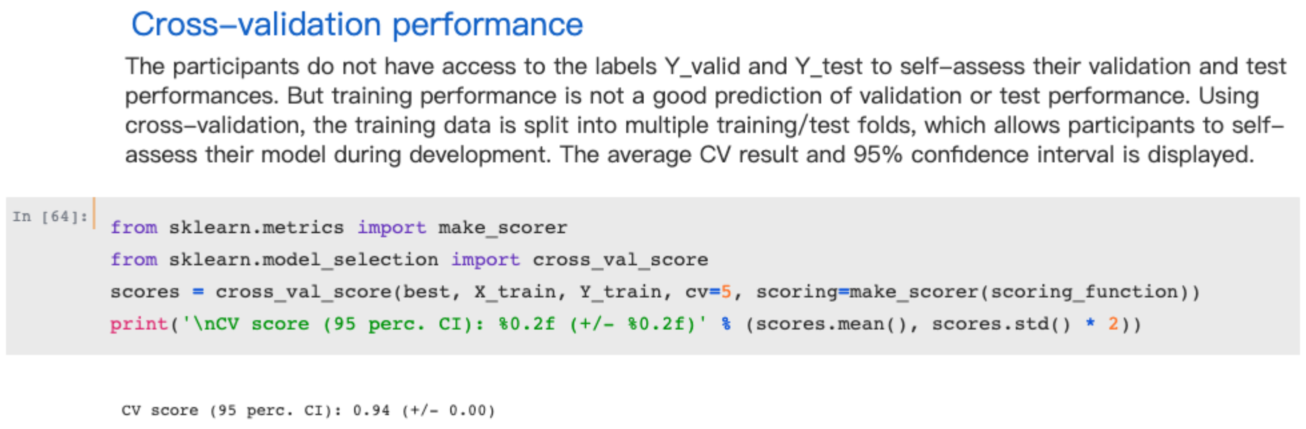


Figure 3.3 – Training Performance



Cross–validation performance

The participants do not have access to the labels Y_valid and Y_test to self–assess their validation and test performances. But training performance is not a good prediction of validation or test performance. Using cross–validation, the training data is split into multiple training/test folds, which allows participants to self–assess their model during development. The average CV result and 95% confidence interval is displayed.

```
In [64]:
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
scores = cross_val_score(best, X_train, Y_train, cv=5, scoring=make_scorer(scoring_function))
print('\nCV score (95 perc. CI): %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))
```

```
CV score (95 perc. CI): 0.94 (+/- 0.00)
```

Figure 3.4 – Cross Validation Performance

4. Visualization

The aim of this part is to produce an interesting display for the study of our problem and the interpretation of the results obtained, so that they are easily understandable by all.

For this, we reasoned as follows. Regression learning involves predicting events, in our case, predicting traffic in a city. After the learning transition, we thought about comparing the values predicted in the training with real values. The best way is to use the method "scatter" from matplotlib because, like a confusion matrix for a classification problem, this tool allows us to compare predicted results and "in real life" values. However, we work with more than 7000 pieces of data, what seemed the most logical for us is to represent the data found as distributed blocks in time.

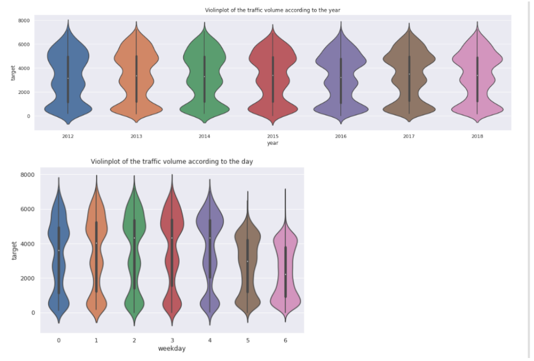


Figure 4.1 – Violin's curve of traffic level during a week and a year

4.1 General features

In addition, in order to determine the most important parameters, we used the correlation matrix which allows us to have percentages representing how closely one parameter is linked to one another. (4.2 figure)

4.2 Traffic results

From now, prediction results could be studied with three graphic curves: train, valid and test. (4.3 figure). We can see that those representations are helpful to compare with our first data

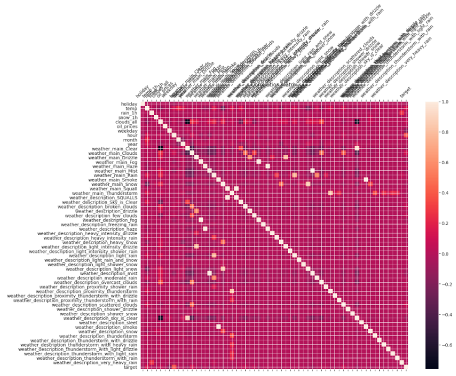
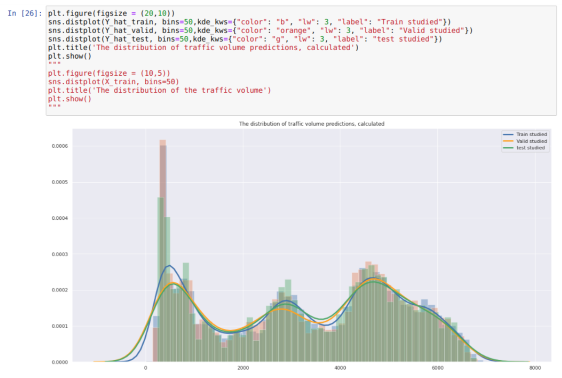


Figure 4.2 – Correlation curve

traffic level violin's graphics. As a result, we have a turned semi-violin curve with a prediction of 7000 pieces of data. In orange, we have the most interesting ones. It shows a heavy and punctual traffic at first, during the morning and a dragged one during the evening.



```
In [26]: plt.figure(figsize = (20,10))
         sns.distplot(Y_hat_train, bins=50,kde_kws={"color": "b", "lw": 3, "label": "Train studied"})
         sns.distplot(Y_hat_valid, bins=50,kde_kws={"color": "orange", "lw": 3, "label": "Valid studied"})
         sns.distplot(Y_hat_test, bins=50,kde_kws={"color": "g", "lw": 3, "label": "test studied"})
         plt.title('The distribution of traffic volume predictions, calculated')
         plt.show()
         """
         plt.figure(figsize = (10,5))
         sns.distplot(X_train, bins=50)
         plt.title('The distribution of the traffic volume')
         plt.show()
         """
```

Figure 4.3 – Train Test and Valid sets

4.3 Comparison of regressor models

The first thing is to be clear about label choices. After selection of many pre-processing options and model regressors, the visualization group charged to compare easily score results (Figure 4.4). Score curves are useful to evaluate the evolution of score according to every processing method. Nevertheless, Bar plot (Figure 4.5 4.6) may be interesting to measure the best score. There are many ways to compare models graphically, a visual one is Correlation matrix of score levels.

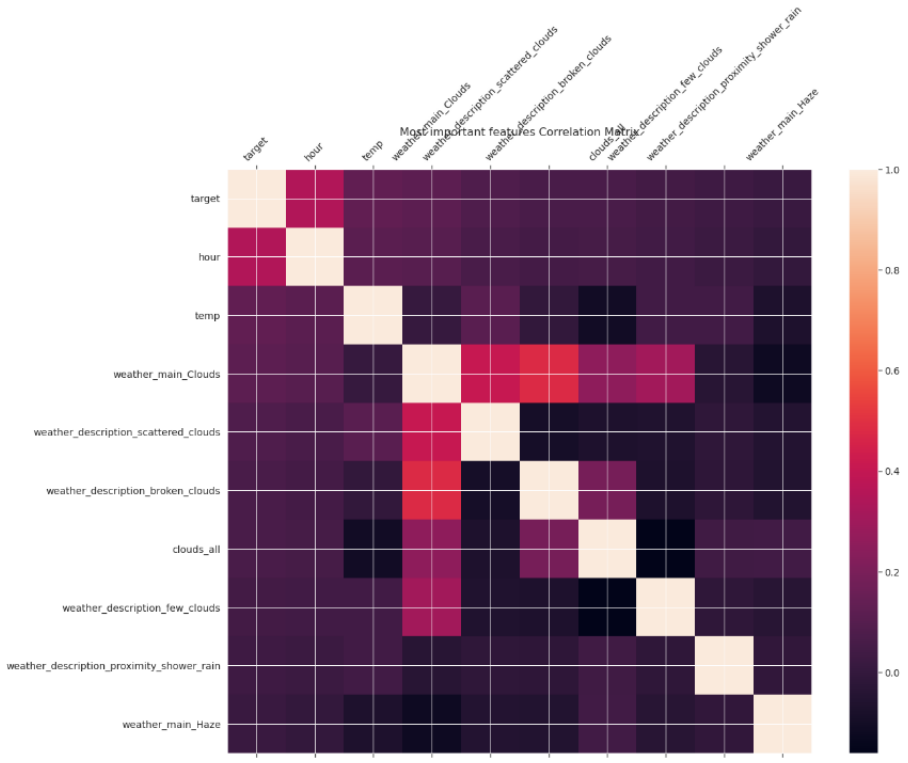


Figure 4.4 – Pertinent features - correlation matrix

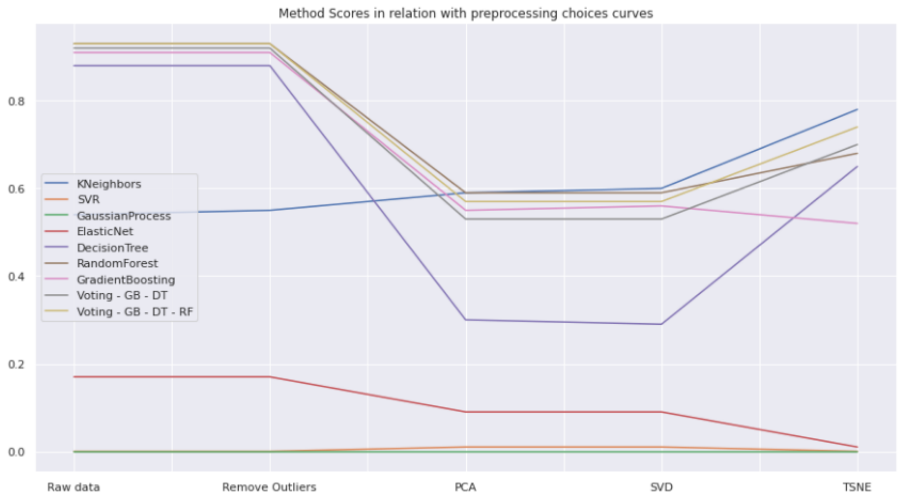


Figure 4.5 – Score evolution in function of processed features for each model regressor

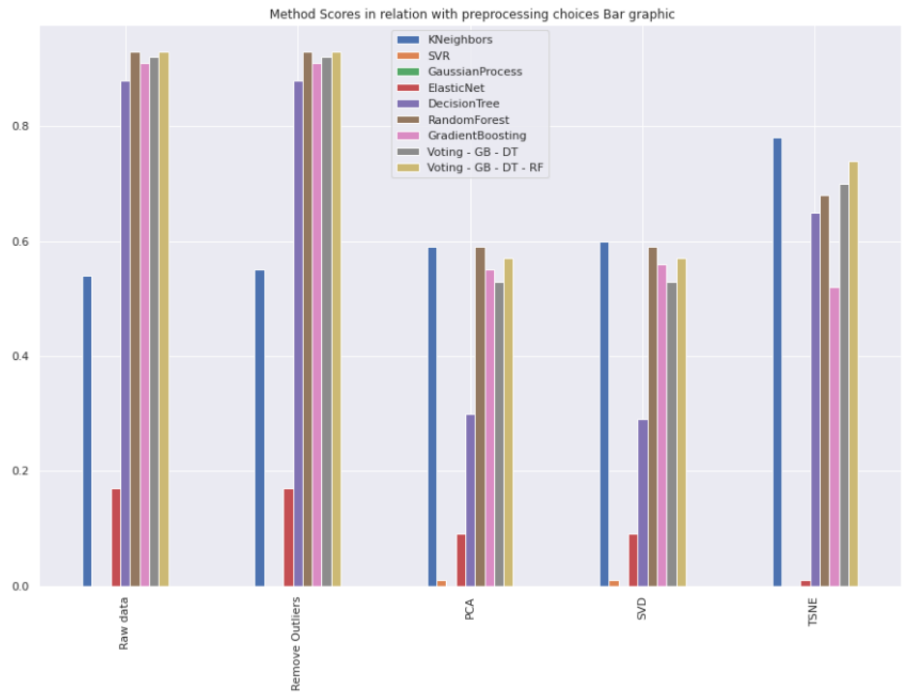


Figure 4.6 – Method Score bar plot

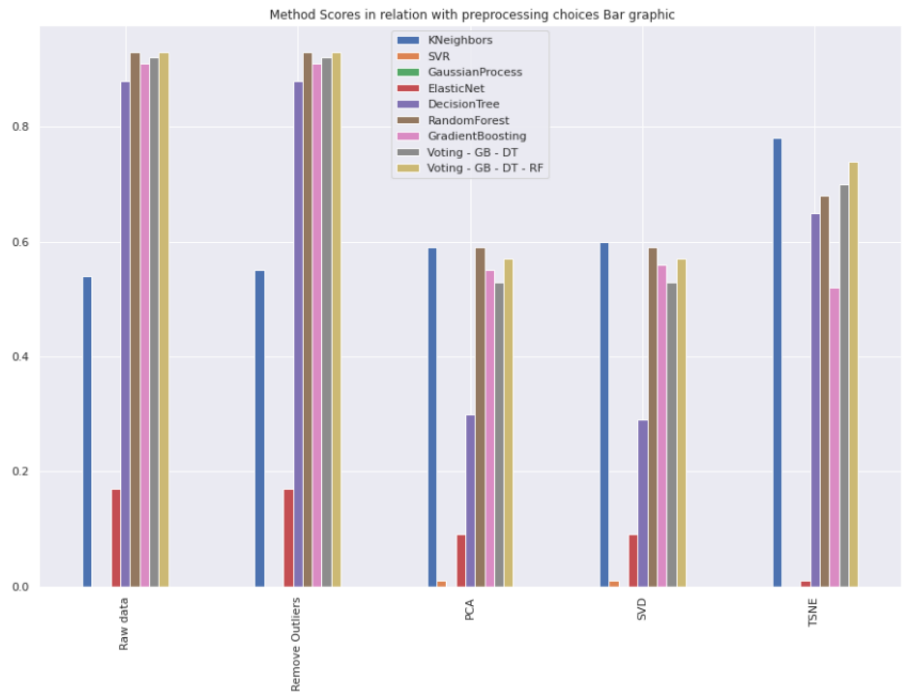


Figure 4.7 – Method Score bar plot

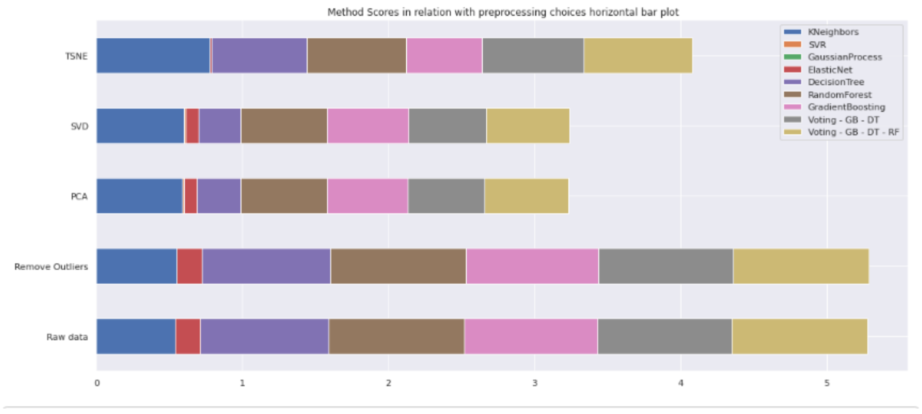


Figure 4.8 – Method Score horizontal bar plot

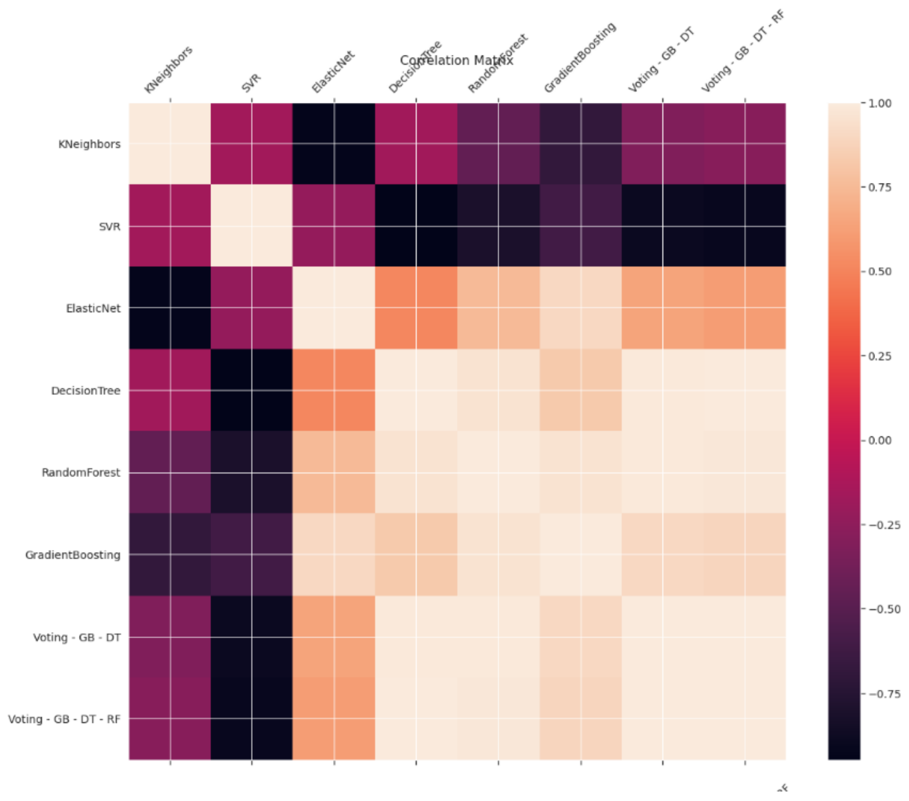


Figure 4.9 – Correlation matrix score levels

5. Conclusions

As we have seen, in the cross validation method, we gain 94% score for successfully predicting the traffic volume. We consider this an acceptable result. But further improvement is still possible.
Here, we can summary that with such a multi features data set, it's important to eliminate unnecessary features first, then choose several related models, finding best parameters, cross validation, then we can get our result using graphics.
Every part is as important as each other, and if possible, cooperate with each other like model subgroup could propose a model and pre-processing subgroup can eliminate the specific useless feature for this model. And for choosing the best parameters, GridSearchCV might be useful. Results of score visualization can help us to conclude that Voting regressor represents the best option. We can manually modify some parameters and find out what's best for what we expect and how to improve score group.

Appendix : Resources

1. Breiman L (1984) Classification and regression trees. The Wadsworth and Brooks-Cole statistics probability series. Chapman & Hall.

2. Wiki - Gradient Boosting https://en.wikipedia.org/wiki/Gradient_boosting

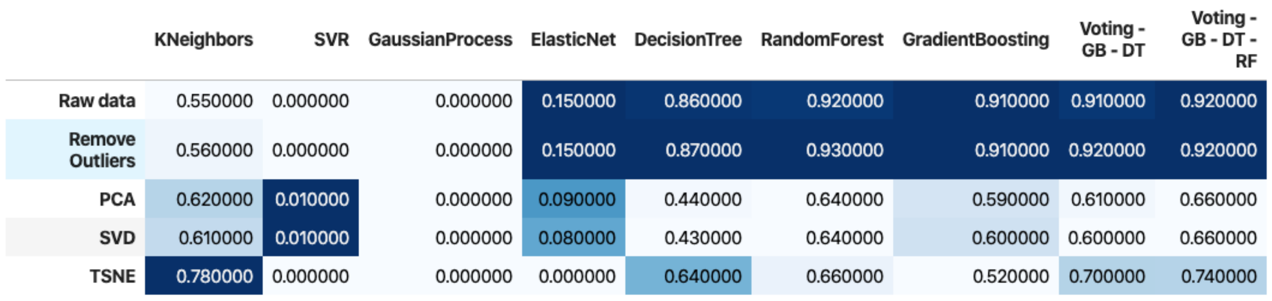
3. Mlxtend Sebastian Raschka Assistant Professor of Statistics at the University of Wisconsin-Madison
http://rasbt.github.io/mlxtend/user_guide/regressor/StackingRegressor/

4. Understanding Random Forest
https://towardsdatascience.com/understanding-random-forest-58381e0602d2

5. A TUTORIAL ON PRINCIPAL COMPONENT ANALYSIS
https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf

6. Visualizing Data using t-SNE
http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

7. Outlier detection between statistical reasoning and data mining algorithms
https://findresearcher.sdu.dk:8443/ws/files/153197807/There_and_Back_Again.pdf



	KNeighbors	SVR	GaussianProcess	ElasticNet	DecisionTree	RandomForest	GradientBoosting	Voting - GB - DT	Voting - GB - DT - RF
Raw data	0.550000	0.000000	0.000000	0.150000	0.860000	0.920000	0.910000	0.910000	0.920000
Remove Outliers	0.560000	0.000000	0.000000	0.150000	0.870000	0.930000	0.910000	0.920000	0.920000
PCA	0.620000	0.010000	0.000000	0.090000	0.440000	0.640000	0.590000	0.610000	0.660000
SVD	0.610000	0.010000	0.000000	0.080000	0.430000	0.640000	0.600000	0.600000	0.660000
TSNE	0.780000	0.000000	0.000000	0.000000	0.640000	0.660000	0.520000	0.700000	0.740000

Figure 5.1 – Performance of combination between models and pre-processing

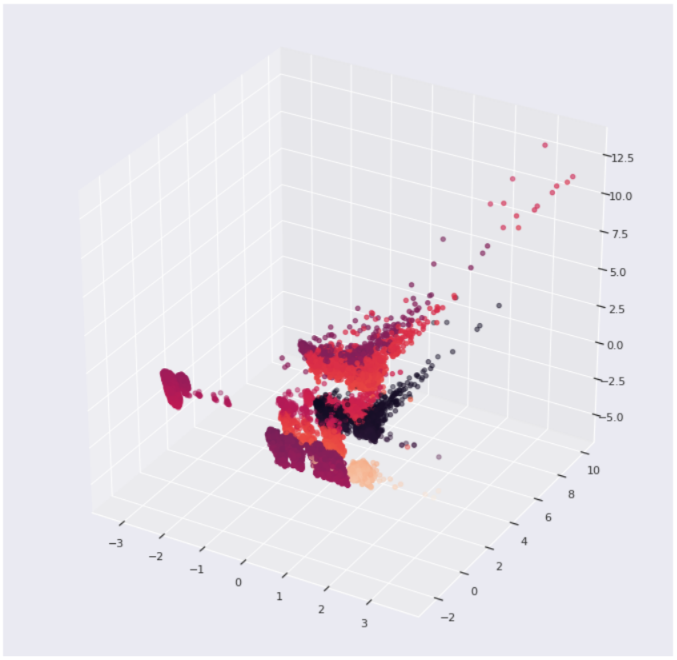


Figure 5.2 – Performance of PCA n_components = 4

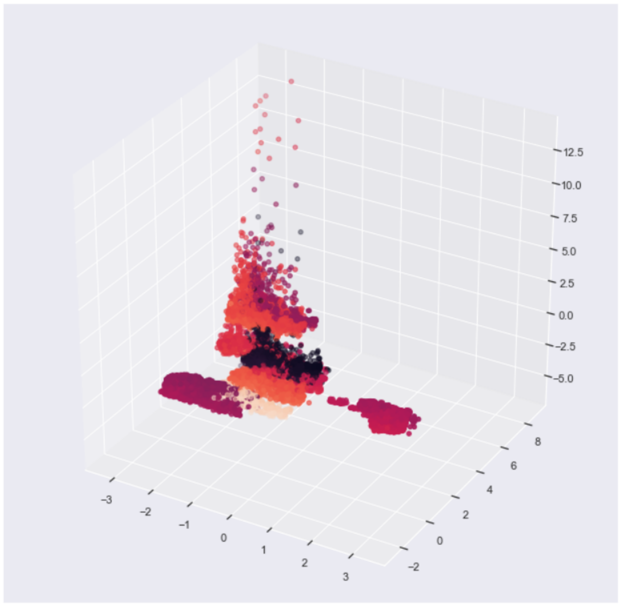


Figure 5.3 – Performance of SVD n_components = 4

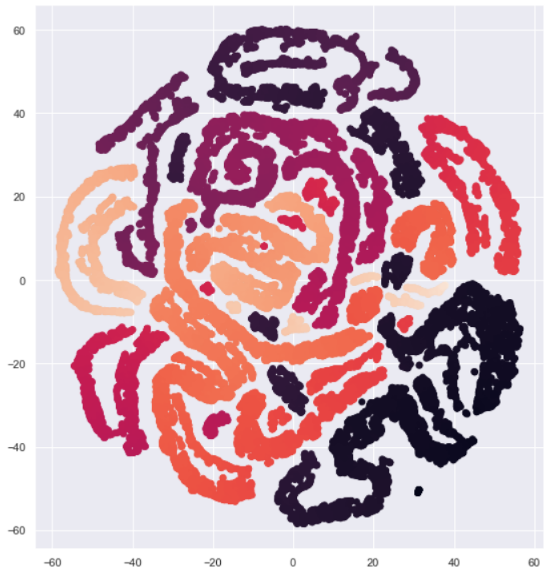


Figure 5.4 – Performance of TSNE