CHEN Shuangrui

# Project Proposal

**L2 MI – Mini Projet - Xporters - SEGWAY**

Pre-processing
Model
Visualization

15/03/2020

# Contents

## 1. Background and description of the problem

We received a request from a young and disruptive entrepreuneur, he haa 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 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 informations in order to avoid waste of lemons.

For doing so, we would like to use this data to train a model that predicts the traffic volume. We also found the hourly meteo 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 classifier. We select and modify the data, in order to allow the classifier to be more efficient. We optimism the reading of the most relevant data which will allow a better treatment and a better result.

First of all, our algorithm will delete the data having the least influence on the final result, i.e. those having the lowest variance (the data having non-variant values on the result).
The objective of pre-processing will be partly to analyze each column and to note that some features will not affect the result, since they only contain 0s. This has the advantage of avoiding noise-related errors by deleting unnecessary data.

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 consistent data set. To do this we used a value filter method: outliers, which allows to keep the selected values in a data.

Next, we use the Principal component analysis (PCA) algorithm. This algorithm allows us to focus on a fragment of data.

# 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 its model performance is the Decision Tree Regressor. But after cross-valide, we find it a bit overfitting (91%) so we decide to use Voting Regressor to neutralize it's overfitting by adding the opinion of Random Forest Regressor(similar principle) and Gradient Boosting Regressor(more mathematical for solving such a multi feature problem). And the result is satisfing. 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 the overfitting of the Decision Tree Regressor.

As for why Decision Tree Regressor overfits, we think this might be caused by its own principle, because it tries to form a tree that exhaust 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 exhaustion. 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-Valide makes us reconsider how to choice a model without knowing its best hyperparameters and influences 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 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 hyperparameters.

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

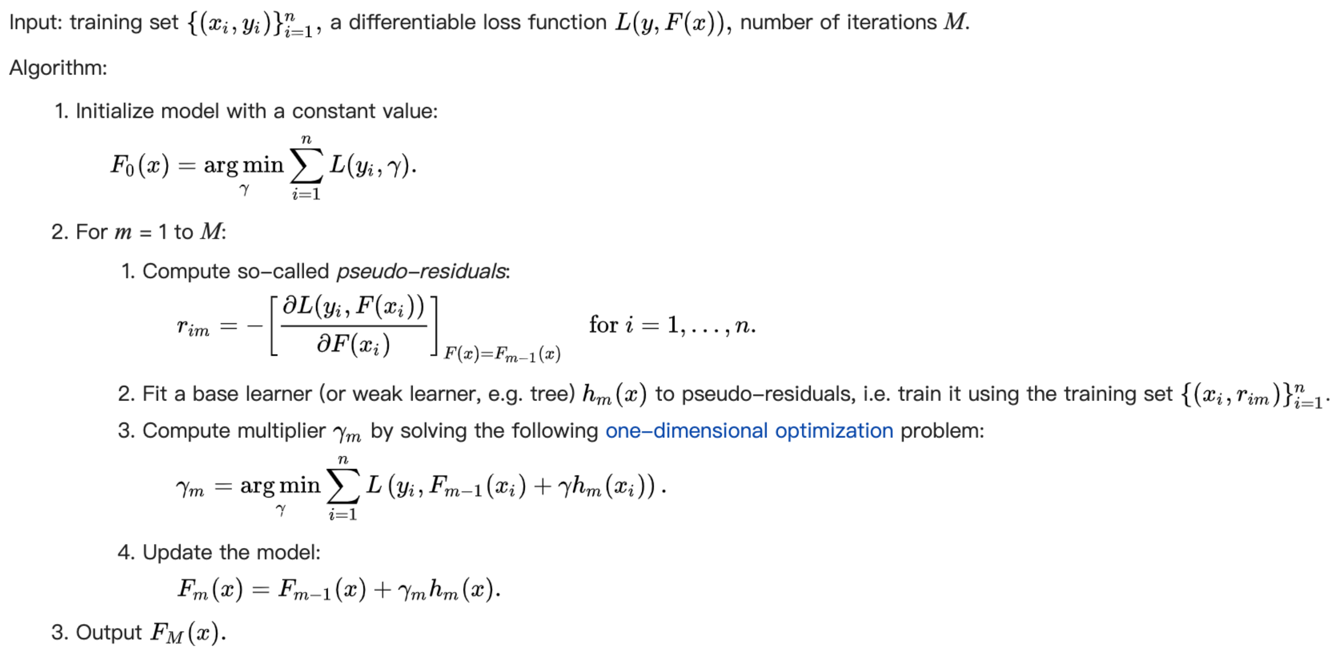
By analyzing the data, there are many irrelevant variables that we have not excluded. If we can filter the variables first, we might have more option on models and parameters.

## 3.2 Models Explanation

Here we dig in some articles from researchers or Wikipedia. Some we summarized by ourselves, some we quote from researchers, we have put our resources in 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 to evaluate the performance of the model. The smaller the better its performance. If the loss function is continuously reduced, the model can be continuously modified to improve its performance. For such a idea might suit our 58 features data set, in other word 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:
$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$
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 $\gamma_m$ by solving the following one-dimensional optimization problem:
   $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$
   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 full samples. 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 perform 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, so 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 decision in the tree structure.

### 3.2.4 Stacking Regression

Stacking regression is an ensemble 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 ensemble learning process is used as the meta-feature input for the meta-regressor. Multiple models can be combined by fitting these meta-features.
Further explanation in the figure:

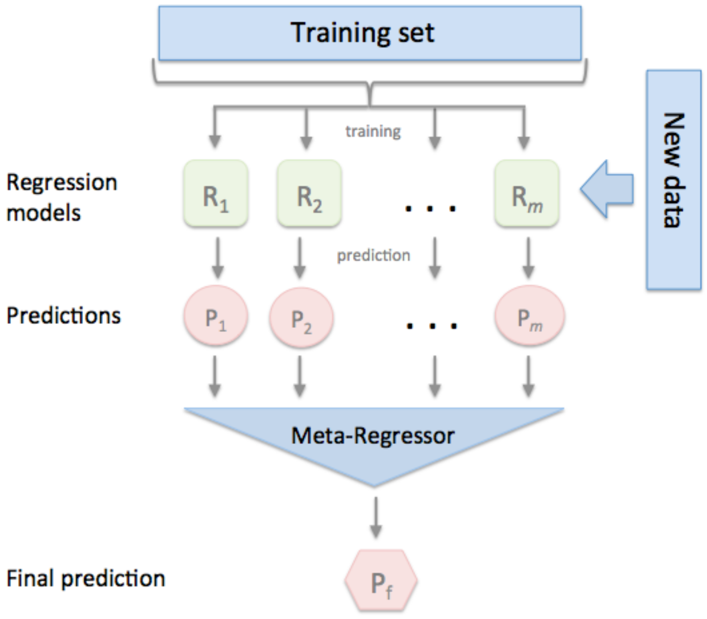


Figure 3.1 – Stacking Regression

### 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 exhaust all possible division of features, but it's also easy to get a over-fitting, makes our model not enough generalizing. But it suits our cases, as we have 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.

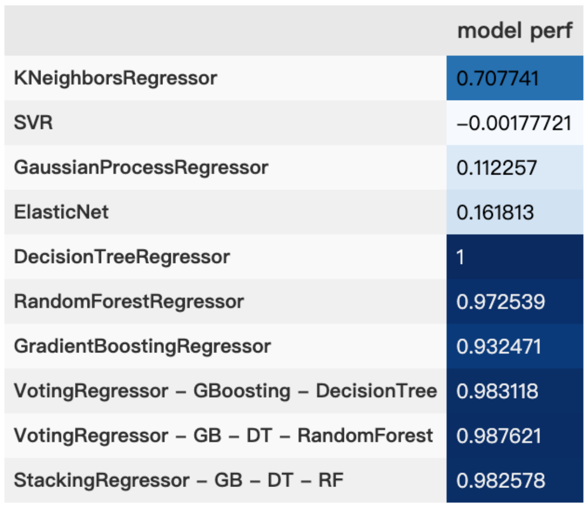
RandomizedSearchCV: The method of using RandomizedSearchCV is actually the same as GridSearchCV, but it replaces GridSearchCV's grid search for parameters by randomly sampling in the parameter space. For parameters with continuous variables, RandomizedSearchCV will treat it as a distribution. Sampling This is not possible with grid search, 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**: Limit 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 is very effective at high dimensions and 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-fitting. 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

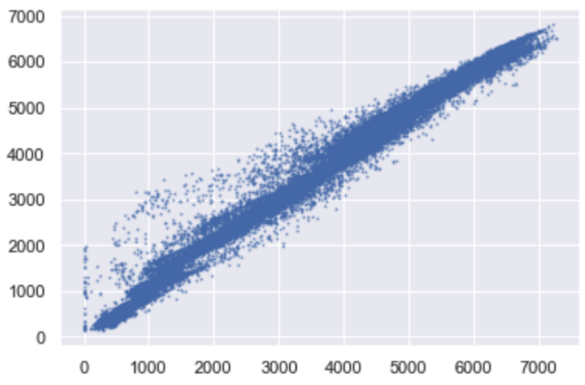
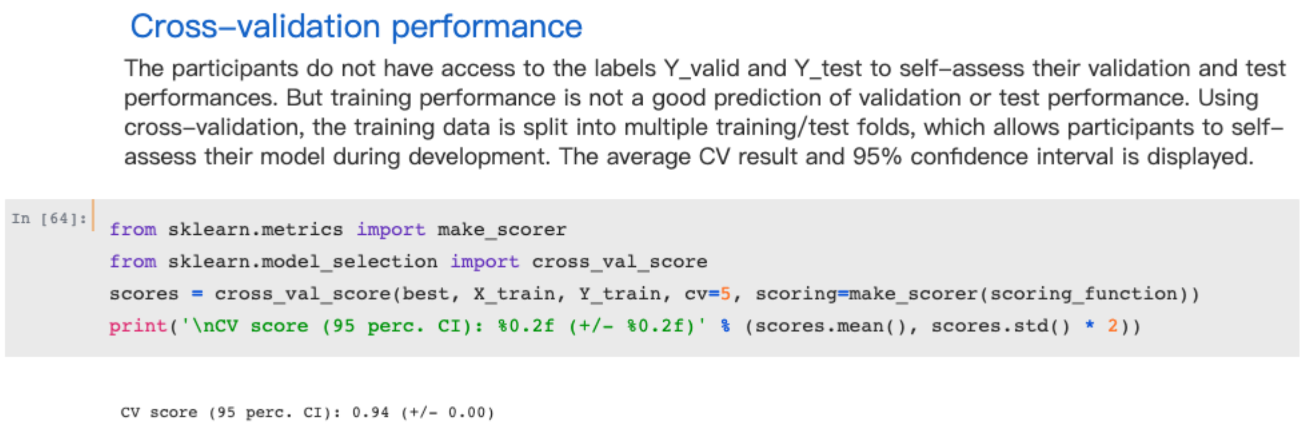


Figure 3.3 – Training Performance


## 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 interesting displays 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 training with the real values. The best way is to use "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 data, which seemed to us the most logical, it is to represent the data found as blocks distributed in time.

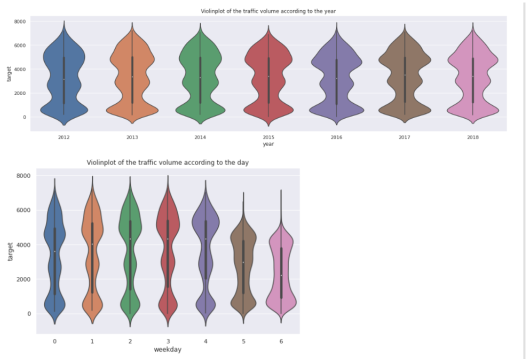


Figure 4.1 – Violin's curve of traffic level during a week and a year

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

From now, prediction results could be studied with three graphic curves: train, valid and test. (4.3 figure)

We can see those representations are helpful to compare them with our first data traffic level violin's graphics. As a result, we got a turned semi-violin curve with a 7000 data prediction. Orange's is the most interesting one which shows a heavy and punctual traffic at first, during the morning and a dragged one during the evening.

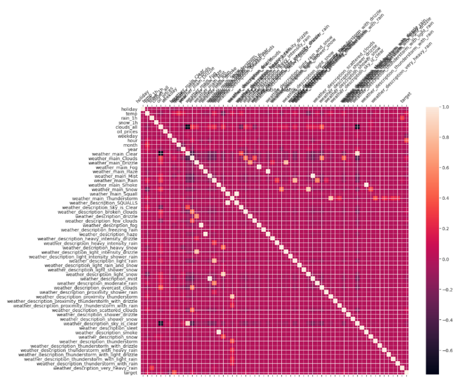
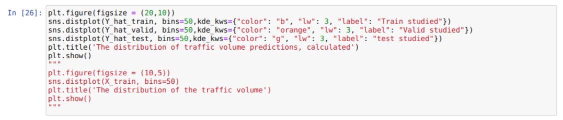


Figure 4.2 – Correlation curve



```
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()
         """
```

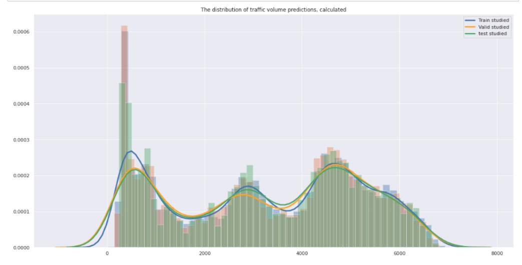


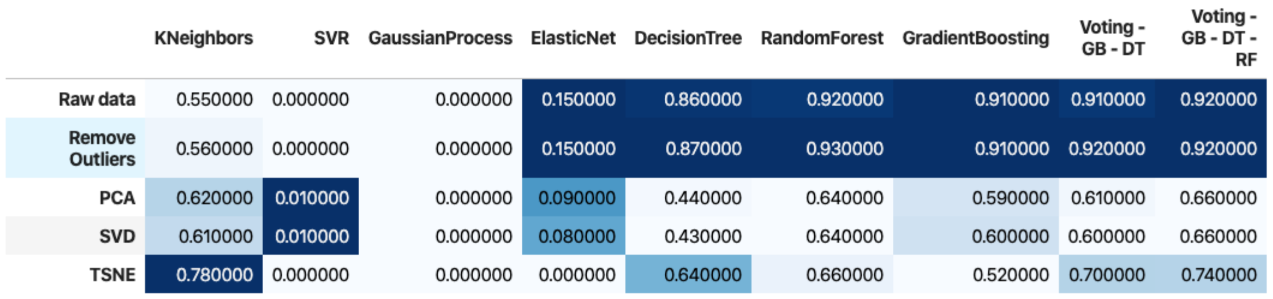
Figure 4.3 – Prediction results

## 5. Conclusions

As we have seen, in the cross valid, we gain a 94% for successfully predicting the traffic volume. We regard this a 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 choice several related models, find best parameters, cross valid, then we can get our result by 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, but if visualization subgroup can provide support for illustration of the whole training procedure, we can manually modify some parameters and find out what's best for what we expect.

# 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**



| | 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

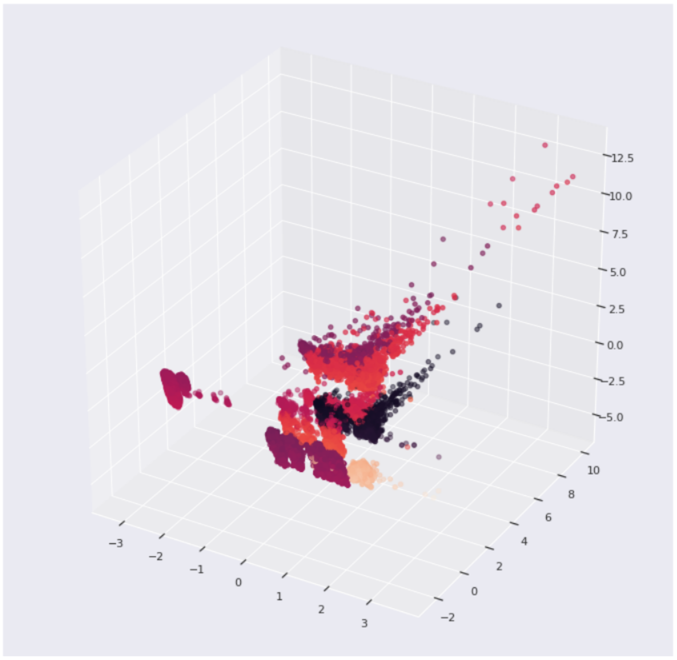


Figure 5.2 – Performance of PCA n_components = 4

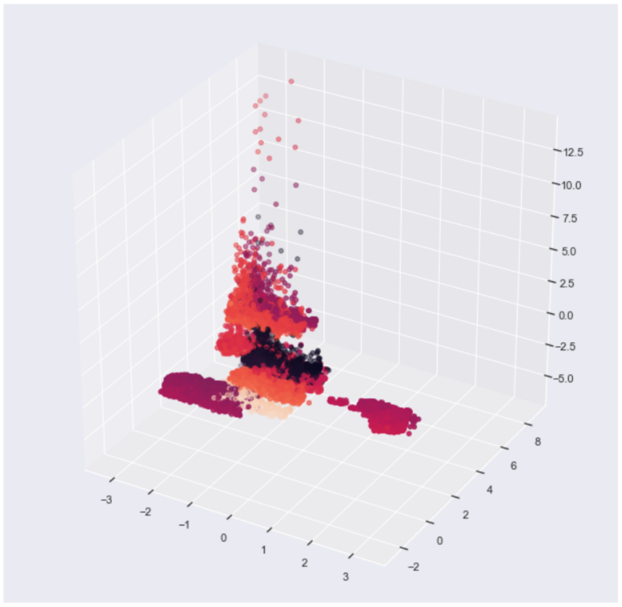


Figure 5.3 – Performance of SVD n_components = 4

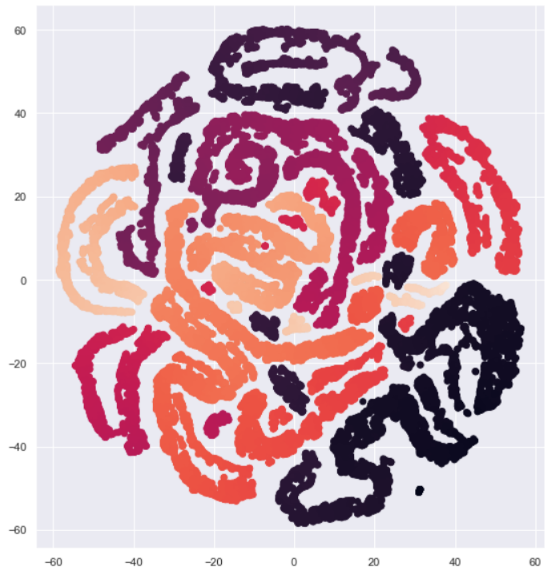


Figure 5.4 – Performance of TSNE