MOTO - REPORT

Members: BENARD Mathieu (leader), REBRAB Tassadit (Gr.2), MOUDEN Hugo (Gr.3), KHALDI Haithem (Gr.4), TEBIB Mazen (Gr.2bis), THERET Tom
Challenge link: https://codalab.lri.fr/competitions/652
GitHub repository: https://github.com/gojoise/MOTO/
Last submission on Codalab: 11
Video link: https://drive.google.com/open?id=1yhKHrkJ5oR7z5lvSBFtC8anW2GzNHmVR

Lemonade sales are dependent on car traffic near your place of business. Your mission, should you decide to accept it, is to predict the number of cars that will pass by at a given date, hour, and additional meteorological informations, near the lemonade stand.

In this regression problem, our task is to predict highway traffic volume using a dataset of 58 features. The work is divided in three steps : preprocessing, visualization and model prediction.

Task distribution

Preprocessing (REBRAB Tassadit - BENARD Mathieu)

This step is made to optimize the dataset by preparing, sorting and organizing all the data. The purpose is to get a lighter and more efficient dataset to make the analysis faster and closer to reality.
Indeed, we have begun with a starting kit including all the results of the experiments, listed in several variables such as the temperature, the weather description, the time, etc.

Model (KHALDI Haithem - TEBIB Mazen)

We are facing a regression problem and we are looking to find the best machine learning model.We will use 5 models: RandomForestRegressor, KNeighborsRegressor, DecisionTreeRegressor, LinearRegression, ExtraTreesRegressor, GradientBoostingRegressor.The cross validation will permit us to choose the best model.

Visualization (MOUDEN Hugo - THERET Tom)

After having processed and found the model to interpret our results, we need to find an interesting and accurate way to visualize them. Therefore, it makes it possible to understand better all the amount of data by everyone.
To visualize the data and create plots, we will use the the matplotlib and also more specifically matplotlib.pyplot library.

PREPROCESSING:

REBRAB Tassadit - BENARD Mathieu

We starter our research with the scikit library available here :
https://scikit-learn.org/stable/modules/preprocessing.html

Then, we focused on feature selection methods that we found here :
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

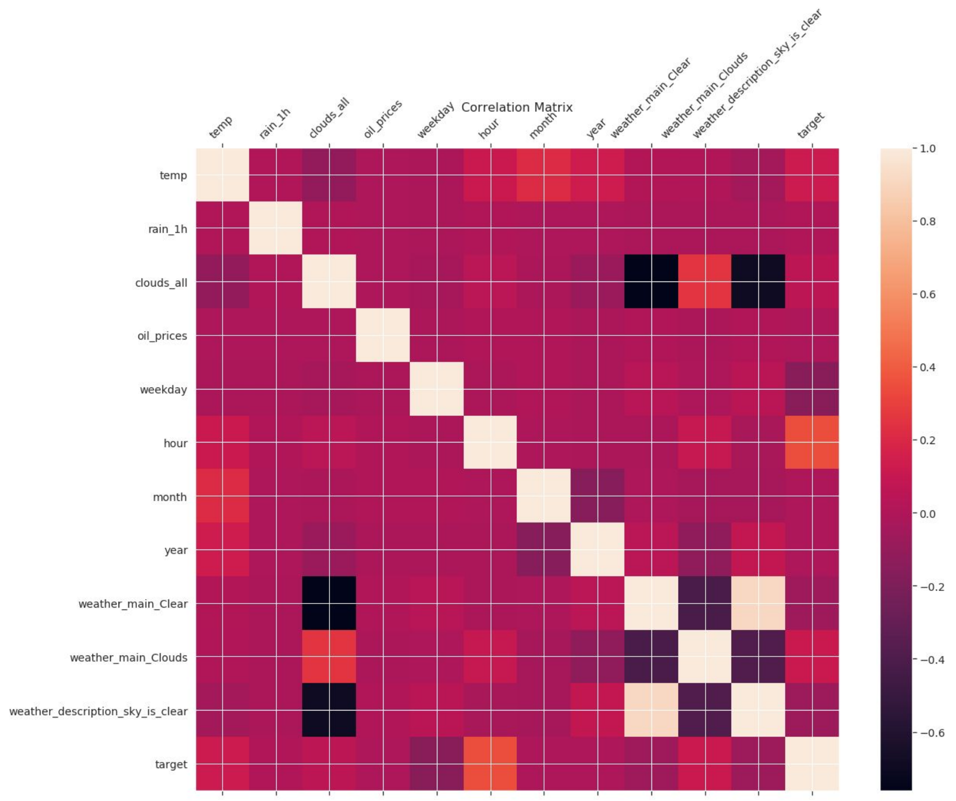
Our main idea is to reduce the number of features because there are a lot of features which contains too many zeros
We use scaler to normalise our data but the effect on prediction are not significant

The dataset is very rich and as there was thousands of numbers, we used VarianceThreshold methods to make a feature selection among the useful variables.

Indeed Variance Threshold is a feature selection algorithm that looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

The main advantage of these methods is the capacity to reduce the number of data while avoiding the loss of the important ones, thanks to the error threshold (we have set the value a 0.2 for the VarianceThreshold

Here is the result of the pd.DataFrame call :



As we can see here, it is easier to see the most influent parameters, the off-center values and the distance to the target. We went from 58 features to only 11 left to work on.

In this respect, we chose to keep the Random Forest methods to continue the model prediction.

VISUALIZATION RESULTS:

README Github link : https://github.com/gojoise/MOTO/blob/master/starting_kit/README_Visualization.ipynb

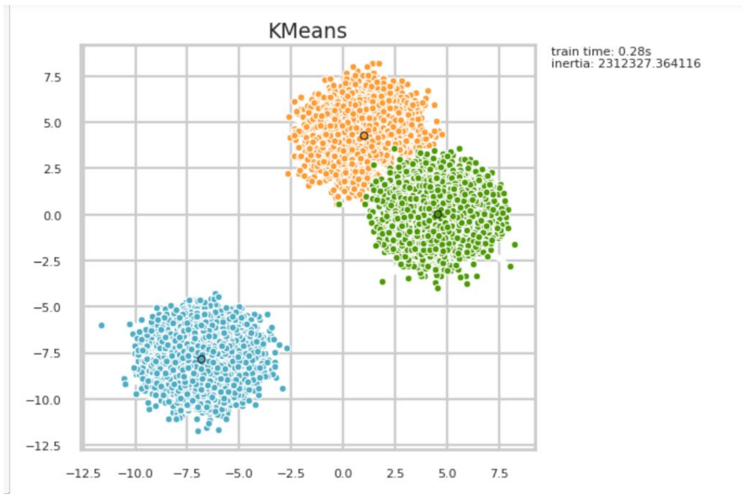
MOUDEN Hugo - THERET Tom

Function used to display the plots using matplotlib library :

To visualize the different clusters in the data, we chose to use the Kmeans cluster generator from scikit learn library.

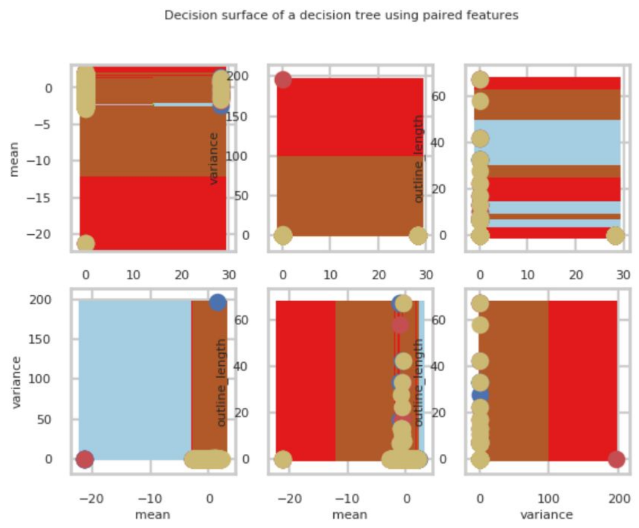
kmeans = KMeans(n_clusters=nb_cluster,random_state=rdm_state).fit(X)

Result :



We also managed to investigate the error in prediction using the decision tree classifier :

Here we can see the classification of the sum, mean, variance and outline length of the most important values chosen by the preprocessing team :



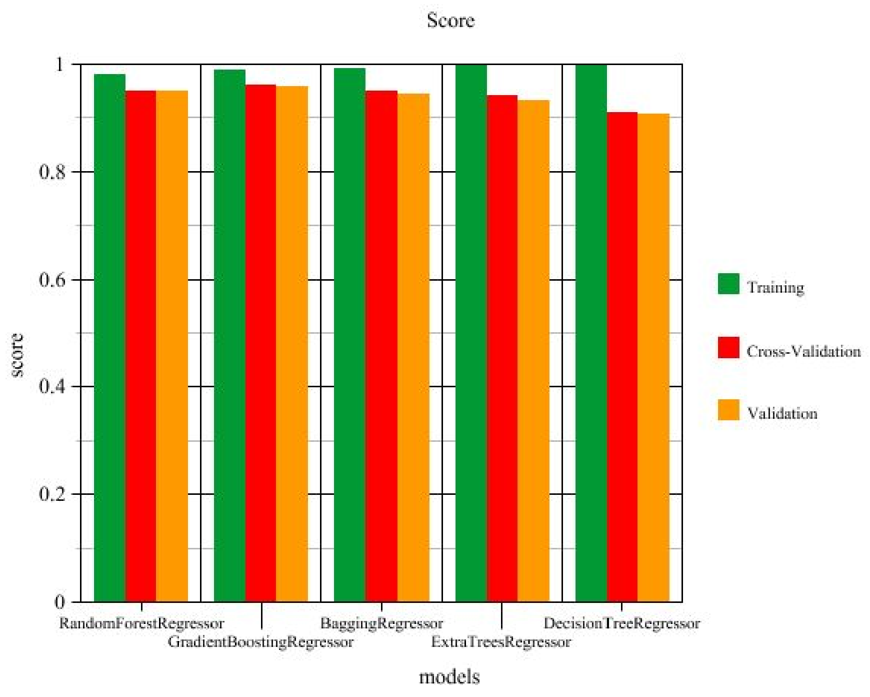
To test the visualization, for not having to wait for the preprocessing results, we used the data given at the beginning, and created the methods to be able to visualize the results on the train data. We made sure to have general variables names so that when we increment it on the final code, we wouldn't have to change what we did.

Code of the decision boundary plot for the decision tree : we used the "`most_important_features`" as target so it could be adapted to any features selected after the preprocessing.

MODEL:

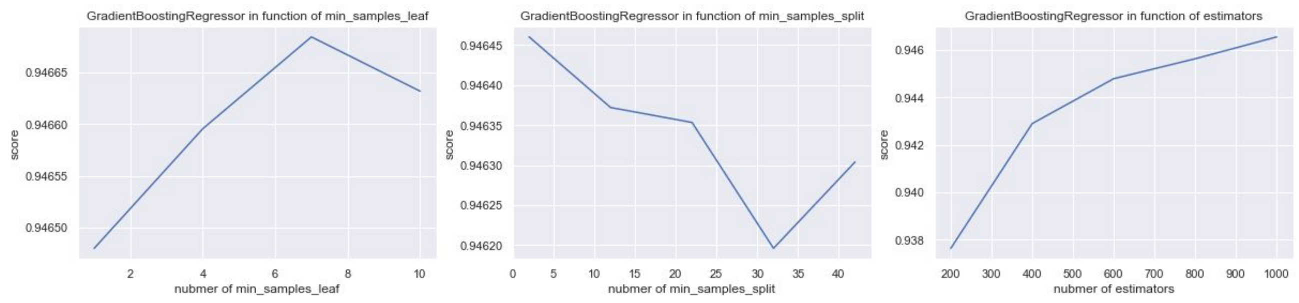
Cross validation performance:
Here we tried to find the best performing model by testing their cross validation performance. And you could see our code in our repository in our file README_model.ipynb
Then for each machine learning model we evaluate the score and choose the best model :
Result:KNeighborsRegressor CV score (95 perc. CI): 0.77 (+/- 0.01)
RandomForestRegressor CV score (95 perc. CI): 0.95 (+/- 0.00)
DecisionTreeRegressor CV score (95 perc. CI): 0.91 (+/- 0.01)
LinearRegression CV score (95 perc. CI): -0.11 (+/- 1.09)
ExtraTreesRegressor CV score (95 perc. CI): 0.94 (+/- 0.01)
GradientBoostingRegressor CV score (95 perc. CI): 0.92 (+/- 0.01)

At first we found that RandomForestRegressor had the best cross validation score but after finding the best hyperparameters for each model with GridSearchCV and RandomizedSearchCV we found out that the best performing model is **GradientBoostingRegressor** with a score 96% :

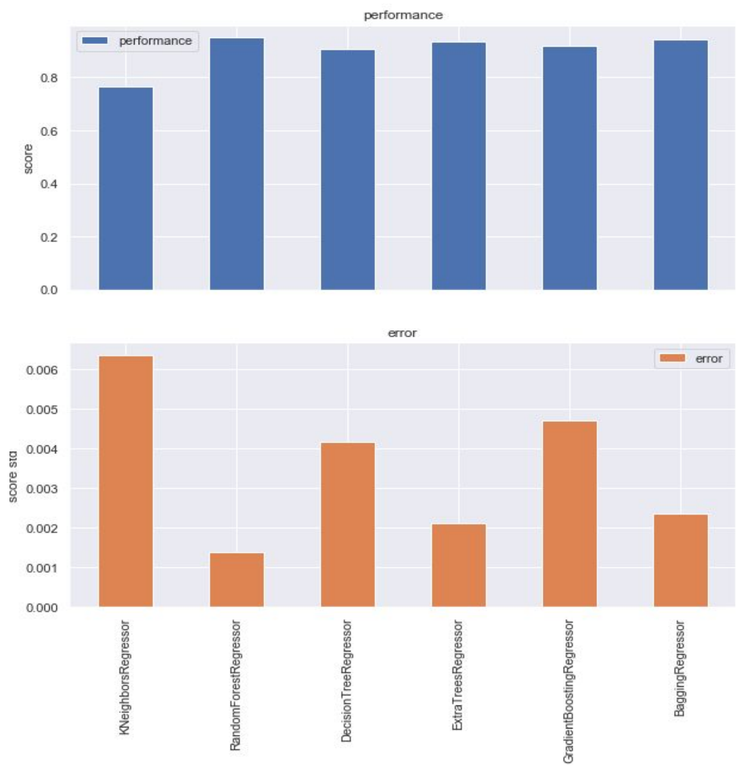


Voting: After searching for best hyperparameters, we tried ensemble learning by combining prediction from different model and we left the weights=None so every model will have same uniform weights and we got as a CV score : CV score (95 perc. CI): 0.95 (+/- 0.00)

hyperparameters : As we know tuning hyperparameters is the most important thing in choosing our model and getting the best score. So to display the importance of each hyperparameter, we present the score for our model in function of the evolution of each hyperparameter:



- The estimators we notice that the score gets better the bigger the number of estimators but we don't want increase it more so we don't have overfitting case.
- For the min_samples_split, it's used to control over-fitting we notice that the score gets better the bigger the number of estimators but we don't want increase more so we don't have under-fitting case.
- For the min_samples_leaf, it's also used to control over-fitting similar the min_samples_split we notice that the score gets better the bigger the number of leafs but we don't want increase more it can cause under-fitting as you can see in the graph.



we can also display the errors and performance to decide which model is better performing because we can have a great performance but we can have error rates that may be very low and seem to be indicative of a high-performing model, but one must be careful, as this may be due to overfitting as, which would result in a model that is unable to generalise well to new data.

Sources :

Preprocessing
https://scikit-learn.org/stable/modules/preprocessing.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html#sklearn.ensemble.IsolationForest

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

Model
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
https://scikit-learn.org/stable/supervised_learning.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

Visualization
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html?highlight=decisiontreeregressor#sklearn.tree.DecisionTreeRegressor