PROPOSAL

Membres : BENARD Mathieu (chef), REBRAB Tassadit, MOUDEN Hugo, KHALDI Haithem TEBIB Mazen, THERET Tom
URL du challenge: https://codalab.lri.fr/competitions/652
Repo GitHub du projet : https://github.com/gojoise/MOTO/

Challenge presentation

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.

TEAM 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 outlier detection and removal methods that we found here :
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

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

Indeed, the Isolation Forest is highly adapted to high-dimensional datasets to make recursive random splittings. The 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 and 0.05 for the IsolationForest), as shown in the code below :

Methods import
```
from sklearn import preprocessing
from sklearn.feature_selection import VarianceThreshold
```

```
from sklearn.ensemble import IsolationForest
X = data
```

Transformation within the parameters and model fitting

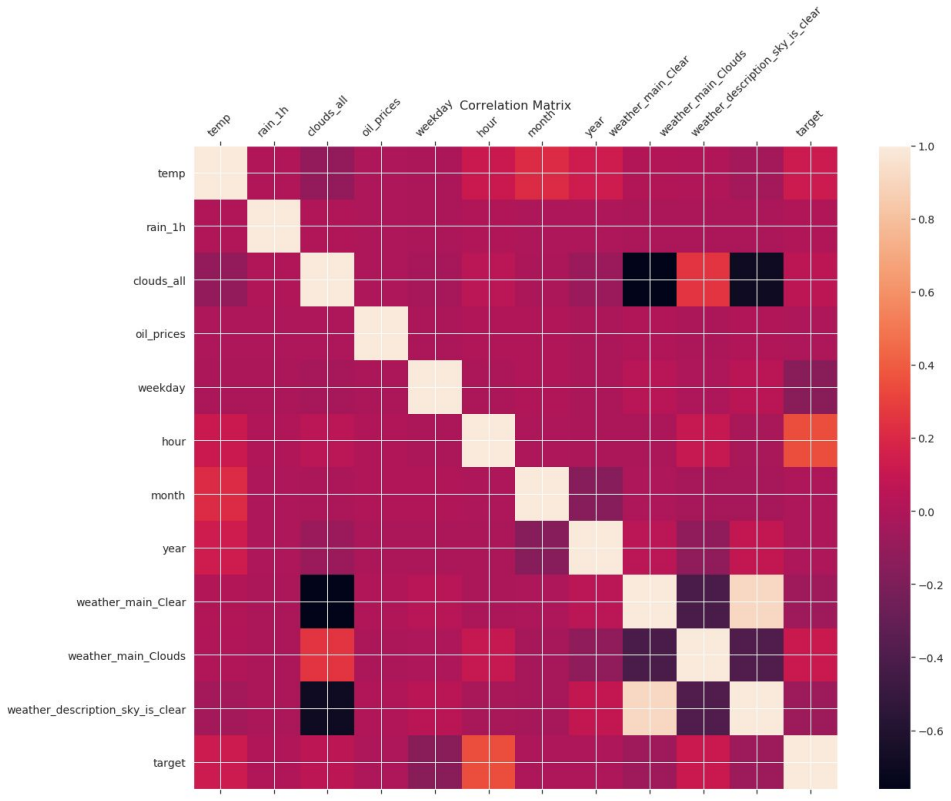
```
model = IsolationForest(max_samples = 100, n_estimators = 20, contamination = 0.05)
model.fit(X)
?model.fit

sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
X_slect = sel.fit_transform(X)
scaler = preprocessing.StandardScaler().fit(X_slect)
scaler.mean_
scaler.scale_
X_slect=scaler.transform(X_slect)
```

Extraction of a DataFrame

```
X_slect
data=pd.DataFrame(X_slect, columns=X.columns[sel.get_support()])
```

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 RESUL

TEAM MODEL:

KHALDI Haithem et TEBIB Mazen

First results:
Model:
Cross validation performance:
Here we tried to find the best performing model by testing their cross validation performance.

Code :

```
#RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

#KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor

#DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor

#LinearRegression
from sklearn.linear_model import LinearRegression


#ExtraTreesRegressor
from sklearn.ensemble import ExtraTreesRegressor

#GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor

#BaggingRegressor
from sklearn.ensemble import BaggingRegressor
```

```
modelName = [ "KNeighborsRegressor","RandomForestRegressor", "DecisionTreeRegressor", "LinearRegression",
         "ExtraTreesRegressor","GradientBoostingRegressor","BaggingRegressor"]

modelList = [
    KNeighborsRegressor(),
    RandomForestRegressor(max_depth=20, random_state=0,  n_estimators=100 ),
    DecisionTreeRegressor(),
    LinearRegression(),
    ExtraTreesRegressor(),
    GradientBoostingRegressor(),
    BaggingRegressor()
]
```

```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
for i in range(len(modelList)):
    scores = cross_val_score(modelList[i], X_train, Y_train, cv=5, scoring=make_scorer(scoring_function))
    print('\n'+modelName[i]+' CV score (95 perc. CI): %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))
```

Cross_val_score: Evaluate a score by cross-validation.
Make_scorer: This factory function wraps scoring functions for use in GridSearchCV and cross_val_score. It takes a score function, such as accuracy_score, mean_squared_error, adjusted_rand_index or average_precision and returns a callable that scores an estimator's output.

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)

RandomForestRegressor :

the best performing model is **RandomForestRegressor** with a score 95% :

Intoroduction:
Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision *trees*, resulting in a *forest of trees*, hence the name "Random Forest"

Algorithm 1: Pseudo code for the random forest algorithm

To generate c classifiers:
for i = 1 to c **do**
  Randomly sample the training data D with replacement to produce D_i
  Create a root node, N_i containing D_i
  Call BuildTree(N_i)
end for

BuildTree(N):
if N contains instances of only one class **then**
  return
else
  Randomly select x% of the possible splitting features in N
  Select the feature F with the highest information gain to split on
  Create f child nodes of N, N_1,..., N_f, where F has f possible values (F_1, … , F_f)
  for i = 1 to f **do**
    Set the contents of N_i to D_i, where D_i is all instances in N that match F_i
    Call BuildTree(N_i)
  end for
end if

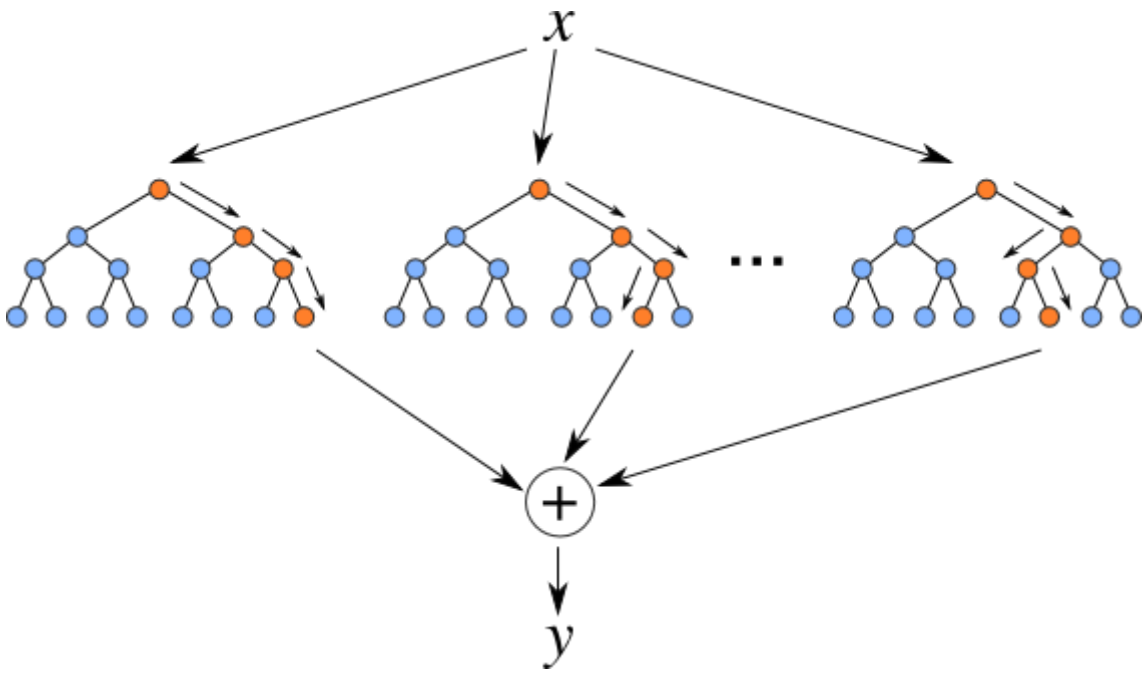
default :
sklearn.ensemble.RandomForestRegressor(n_estimators=100, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None)

Feature and Advantages of Random Forest :

1. It is one of the most accurate learning algorithms available. For many data sets, it produces a **highly accurate classifier**.
2. It runs efficiently on large databases.
3. It can **handle thousands of input variables** without variable deletion.
4. It gives estimates of what variables that are important in the classification.
5. It generates an internal **unbiased estimate of the generalization error** as the forest building progresses.
6. It has an **effective method for estimating missing data** and maintains accuracy when a large proportion of the data are missing.

Disadvantages of Random Forest :

1. Random forests have been observed to **overfit for some datasets** with noisy classification/regression tasks.
2. For data including categorical variables with different number of levels, **random forests are biased in favor of those attributes with more levels**. Therefore, the variable importance scores from random forest are not reliable for this type of data.



the importance of hyper-parameter:

1. n_estimators: The n_estimators parameter specifies the number of trees in the forest of the model, Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably, and it will not overfit when increasing the number of trees in the forest
2. max_depth: The max_depth parameter specifies the maximum depth of each tree. The default value for max_depth is None, which means that each tree will expand until every leaf is pure. A pure leaf is one where all of the data on the leaf comes from the same class. and model overfits for large depth values. The trees perfectly predicts all of the train data, however, it fails to generalize the findings for new data
3. min_samples_split: The min_samples_split parameter specifies the minimum number of samples required to split an internal leaf node. The default value for this parameter is 2, which means that an internal node must have at least two samples before it can be split to have a more specific classification. Increasing this value can cause underfitting
4. min_samples_leaf: The min_samples_leaf parameter specifies the minimum number of samples required to be at a leaf node. The default value for this parameter is 1, which

means that every leaf must have at least 1 sample that it classifies. Increasing this value can cause underfitting

5. max_features: represents the number of features to consider when looking for the best split

Best Hyper-parameters:

Here we try searching for best hyper-parameters for RandomForestRegressor by using GridSearchCV and RandomizedSearchCV, we used both to see how different the results could be.
GridSearchCV :

Code:

```
from sklearn.model_selection import GridSearchCV
```

```
hyperF = {
    'max_depth': [5, 8, 15, 25, 30],
    'min_samples_leaf': [1, 2, 5, 10] ,
    'min_samples_split': [2, 5, 10, 15, 100],
    'n_estimators': [100, 300,500,800, 1000]
}

forest = RandomForestRegressor()
gridF = GridSearchCV(forest, hyperF, cv = 3, verbose = 1,
                     n_jobs = -1)
```

```
gridF.fit(X_train,Y_train)
```

Cv : Determines the cross-validation splitting strategy.

Verbose : Controls the verbosity: the higher, the more messages.

n_jobs : Number of jobs to run in parallel.

Forest : is an estimator object.This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.

hyperf : list of dictionaries

Dictionary with parameters names (string) as keys(in our case :max_depth,min_samples_leaf,min_samples_split,n_estimators) and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

Gridf.fit : Calling the fit() method will fit the model at each grid point, keeping track of the scores along the way

```
GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                        max_depth=None,
                                        max_features='auto',
                                        max_leaf_nodes=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1,
                                        min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators='warn', n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False),
        iid='warn', n_jobs=-1,
        param_grid={'max_depth': [5, 8, 15, 25, 30],
                    'min_samples_leaf': [1, 2, 5, 10],
                    'min_samples_split': [2, 5, 10, 15, 100],
                    'n_estimators': [100, 300, 500, 800, 1000]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
        scoring=None, verbose=1)
```

Code:

```
gridF.best_params_
```

Parameter setting that gave the best results on the hold out data

```
Result: {'max_depth': 30,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 1000}
```

Code:

```
gridF.best_estimator_
```

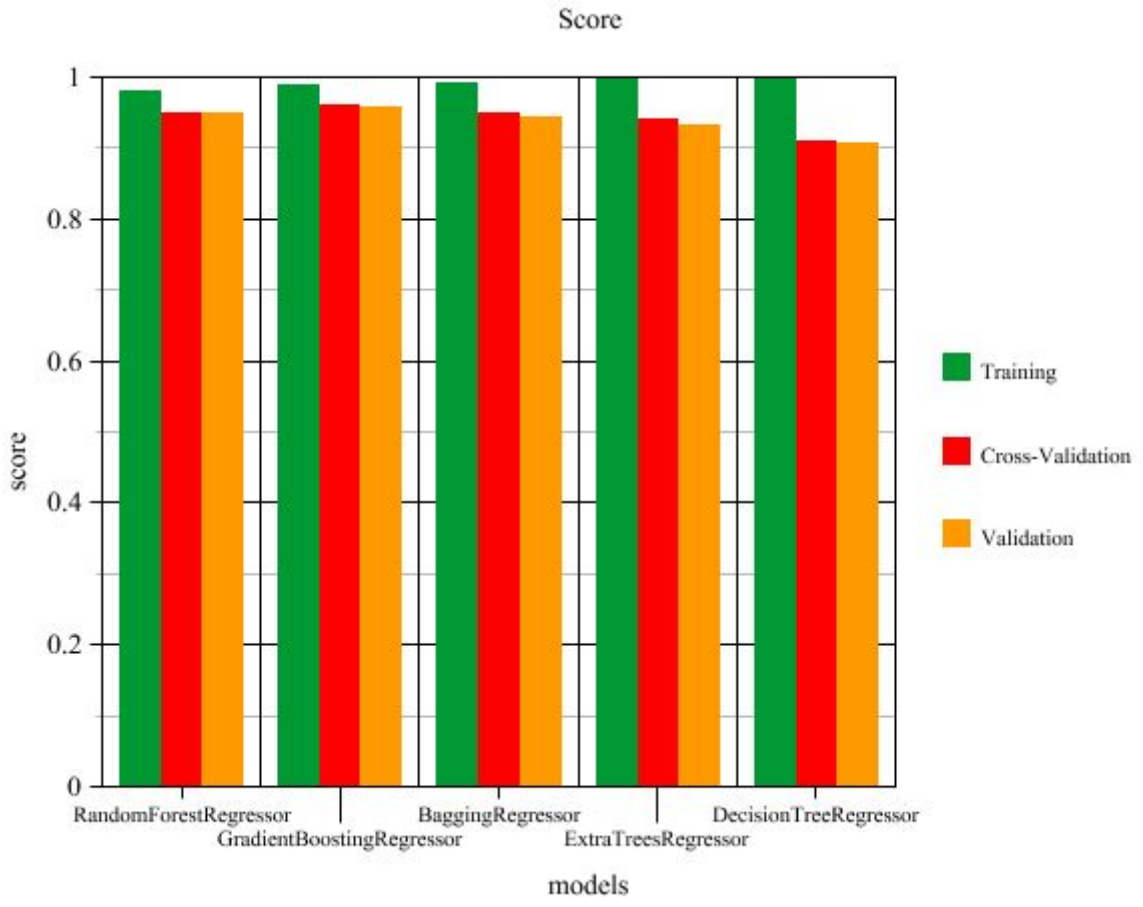
Estimator that was chosen by the search, i.e. estimator which gave highest score (or smallest loss if specified) on the left out data. Not available if refit=False

```
Result:RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=30,
                             max_features='auto', max_leaf_nodes=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=2, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, n_estimators=1000,
                             n_jobs=None, oob_score=False, random_state=None,
                             verbose=0, warm_start=False)
```

And for the RandomizedSearchCV we basically used the same method to search for the best hyper-parameter and as a result we got :

{'n_estimators': 1200,
 'min_samples_split': 2,
 'min_samples_leaf': 3,
 'max_depth': 31}

And for the score result both gave us the same score there was a difference between them of 0.001+ for the RandomizedSearchCV



We display here the scores for each dataset.
We notice that there's not a big difference in scores in these models.
We notice here a low error rate for most of the models and on the validation Weaker score,
And for better visualization we put the scores here in a table :

Dataset	**RandomForest Regressor**	**Gradient -Boosting Regressor**	**Bagging Regressor**	**Extra-Trees Regressor**	**Decision Tree Regressor**
Training	0.9802	0.9170	0.9904	1.000	1.0000
Cross-Validation	0.95	0.92(+/-0.01)	0.95(+/-0.01)	0.94	0.91 (+/-0.01)
Validation	0.948575034	0.9132339611	0.9430651582	0.93393788	0.908399749827

Sources :
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html