L2 MI – Mini Projet
Challenge Xporters – Groupe MOTO
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.

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.

These information are and as there was thousands of numbers, we used the IsolationForest and VarianceThreshold methods to make a selection among the useful variables.

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.

First results

Model:

Cross validation performance:

Code :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)

Best Hyper-parameters:

Code: 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 estimatorestimator 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

Result:Fitting 3 folds for each of 500 candidates, totalling 1500 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed: 24.0min
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed: 60.5min
[Parallel(n_jobs=-1)]: Done 776 tasks      | elapsed: 123.8min
[Parallel(n_jobs=-1)]: Done 1226 tasks      | elapsed: 224.0min
[Parallel(n_jobs=-1)]: Done 1500 out of 1500 | elapsed: 282.4min finished

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)

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

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,