BIKE COUNTERS PROJECT NOTES

**Standard kit submission - Ridge :**

Result :

RMSE : 1.043 / 1.042 (with the temperature)

Train time : 3.795162

Validation time : 6.102223

It’s a ridge regression with the basic hyperparameters, it is using the date features, the site and counter name (+ the temperature).

**RandomForestRegressor (n\_estimators=50, min\_samples\_leaf=10, n\_jobs=4) :**

Result :

RMSE : 0.926

Train time : 2020.985890

Validation time : 15.326111

We heard about the efficiency of the RandomForestRegressor as a decision tree. The principle seems to be that by maximum voting from a panel of independent judges, we get the final prediction better than the best judge.

We tried to play on two parameters :

* n\_estimators : which is the number of trees to build before taking the decision, the higher the best the prediction but the higher the longer the computing time

# Tune random forest parameters

def RFR\_tuning\_n\_estimators(parameters):

score\_stock = np.zeros(parameters.shape)

score\_test = np.zeros(parameters.shape)

for i, parameter in enumerate(parameters):

regressor = RandomForestRegressor(random\_state=0, n\_estimators=parameter)

print(regressor)

pipe = make\_pipeline(date\_encoder, preprocessor, regressor)

pipe.fit(X\_train, y\_train)

print(pipe)

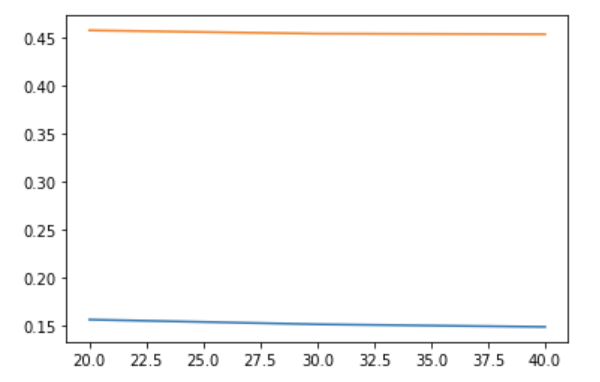
score\_stock[i] = mean\_squared\_error(y\_train, pipe.predict(X\_train), squared=False)

score\_test[i] = mean\_squared\_error(y\_test, pipe.predict(X\_test), squared=False)

print(score\_stock)

return score\_stock, score\_test

Number of estimators / RMSE (blue : train set / orange : test set)



* min\_sample\_leaf : this is the minimum number of samples required to be at a leaf node (means an external node, one end of the tree). A small sample leaf overfit the train set and takes a lot of time as we can see on the graph below. We need to get a good equilibrium between both.

def RFR\_tuning\_min\_sample\_leaf(parameters):

score\_stock = np.zeros(parameters.shape)

score\_test = np.zeros(parameters.shape)

for i, parameter in enumerate(parameters):

regressor = RandomForestRegressor(random\_state=0, min\_samples\_leaf=parameter)

print(regressor)

pipe = make\_pipeline(date\_encoder, preprocessor, regressor)

pipe.fit(X\_train, y\_train)

print(pipe)

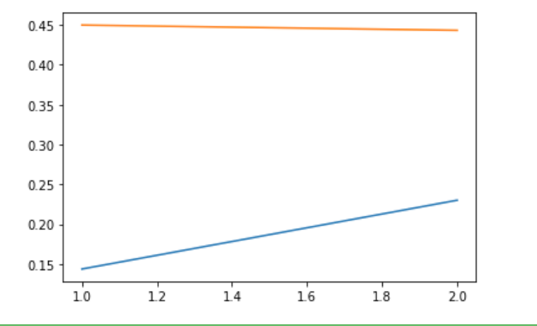
score\_stock[i] = mean\_squared\_error(y\_train, pipe.predict(X\_train), squared=False)

score\_test[i] = mean\_squared\_error(y\_test, pipe.predict(X\_test), squared=False)

print(score\_stock)

return score\_stock, score\_test

Number of estimators / RMSE (blue : train set / orange : test set)



* n\_jobs : the number of processor the computer is authorized to use to compute it. This parameter reduces the time taken for computing.

The RandomForestRegressor is better than the RidgeRegressor for the RMSE but here the problem is that it takes a lot of time to compute, thus it is even difficult to test the parameters because of the time it takes.

Help : https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

**LightGBM :**

To use LightGBM :

pip install lightgbm

Light GBM Ne semble pas bien fonctionner sur jupyter mais OK sur VS

A noter, toujours vérifier la conda list de l’environnement et si un truc marche pas il faut essayer de désinstaller sur conda et pip (uninstall) et réessayer sur conda

Mes params sont corrects là, demain je fais un grid search dessus

<https://ichi.pro/fr/qu-est-ce-que-lightgbm-comment-le-mettre-en-oeuvre-comment-affiner-les-parametres-105778469517308>

<https://www.kaggle.com/paweljankiewicz/lightgbm-with-sklearn-pipelines>

Premier Grid Search :

Trois paramètres – le reste par défaut :

Sur le training test :

* min\_data : Gère l’ajustement, plus petit = plus overfitté (10 entre 10 et 40)
* max\_depth : Profondeur maximum de l’arbre, gère également l’ajustement (70 entre [70 et 150]
* learning\_rate : Impact de chaque arbre sur le résultat final (0,1 entre [0.001, 0.01, 0.1])

Sur le testing set :

* Même résultat
* {'lgbmregressor\_\_learning\_rate': 0.1, 'lgbmregressor\_\_max\_depth': 70, 'lgbmregressor\_\_min\_data': 10} {'lgbmregressor\_\_learning\_rate': 0.1, 'lgbmregressor\_\_max\_depth': 70, 'lgbmregressor\_\_min\_data': 10}

1er ramp test :

params['learning\_rate'] = 0.1

params['boosting\_type'] = 'gbdt'

params['objective'] = 'regression'

params['metric']: ['l2', 'auc']

params['num\_leaves'] = 200

params['min\_data'] = 10

params['max\_depth'] = 70

params['n\_estimators'] = 200

params['task']: 'train'

params["max\_bin"]: 512

score rmse

valid 0.894

test 0.665

Assez rapide

Test avec up learning rate 0.15 est mieux

Test avec max\_depth :

{'lgbmregressor\_\_learning\_rate': 0.15, 'lgbmregressor\_\_max\_depth': 30, 'lgbmregressor\_\_min\_data': 10} {'lgbmregressor\_\_learning\_rate': 0.15, 'lgbmregressor\_\_max\_depth': 30, 'lgbmregressor\_\_min\_data': 10}

Test de descendre min\_dat  2 – 5 -10 : le mieux c’est 5

{'lgbmregressor\_\_learning\_rate': 0.15, 'lgbmregressor\_\_max\_depth': 30, 'lgbmregressor\_\_min\_data': 5} {'lgbmregressor\_\_learning\_rate': 0.15, 'lgbmregressor\_\_max\_depth': 30, 'lgbmregressor\_\_min\_data': 5}

score rmse

valid 0.892

test 0.664

Une image contenant texte

Description générée automatiquement

Next step : Add & Improve data set

Deal with external data : after test on the data that makes sens according to me temperature (t) humidity (u), rr1 (precipitations dernières heure) and rf10 (rafales 10 dernières minutes) semlent pertinentes.

Mises dans numerical.

Test 1

valid 0.891

test 0.666

Deal with external data : after test on every features : humidity, temperature, tendance atmo 24h, néulosité totale et point de rosée

valid 0.890

test 0.664

***Next step : Encodage des données***