Project 2 Part 1

Andres Mejia

12/27/2021

Data preprocessing

We first separate the training data in training and validation data, note that the data is time ordered and the construction of the validation set must respect that ordering

Split the train data into train__train (10 years) and train__validation (2 years). The validation set (train__validation) will be used to compare all six approaches in this section.

```
my_NIA =80762238
np.random.seed(my_NIA)
train = pd.read_pickle('/home/andres/AdprogSKlearn/traintestdata_pickle/trainst1ns16.pkl')
test = pd.read_pickle('/home/andres/AdprogSKlearn/traintestdata_pickle/testst1ns16.pkl')
train_target=train.loc[:,"energy"]
test_target=test.loc[:,"energy"]
train_train=train.iloc[:3650,0:75]
train_validation=train.iloc[3650:,0:75]
test1=test.iloc[:,0:75]
train_target_train=train_target.iloc[:3650]
train_target_validation=train_target.iloc[3650:]
```

Models using default parameters

Doing the models with the default parameters and using the validation set to evaluate them:

Train and evaluate KNN, SVM, and Regression Trees with default hyper-parameters.

Remember that KNN and SVM require scaling.

KNN model

```
pipeKNN = make_pipeline(StandardScaler(), KNeighborsRegressor())
pipeKNN.fit(train_train,train_target_train)
```

```
knn_pred=pipeKNN.predict(train_validation)
```

Tree model

```
modelotree=DecisionTreeRegressor(random_state=1)
modelotree.fit(train_train,train_target_train)
```

```
tree_pred=modelotree.predict(train_validation)
```

SVR model

```
pipeSVR = make_pipeline(StandardScaler(), SVR())
pipeSVR.fit(train_train,train_target_train)
```

```
SVR_pred=pipeSVR.predict(train_validation)
```

```
metrics.mean_absolute_error(knn_pred,train_target_validation)
```

```
## 2588455.571506849
```

```
metrics.mean_absolute_error(tree_pred,train_target_validation)
```

```
## 3068353.430136986
```

```
metrics.mean_absolute_error(SVR_pred,train_target_validation)
```

```
## 6380505.183755267
```

From this it seems that the model that has the lowest mean absolute error is k-nearest neighbors regression.

Tuning Hyperparameters

Tree model * Tree model only change mae es criterion (to be consistant with evaluation)

```
train_cv_index=np.zeros(train.shape[0])

train_cv_index[:3650] = -1
train_cv_index = PredefinedSplit(train_cv_index)
n_estimators = [int(x) for x in np.linspace(start = 1, stop = 20, num = 20)] # number of trees in the r
max_features = ['log2', 'sqrt'] # number of features in consideration at every split
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)] # maximum number of levels allowed in each
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False] # method used to sample data points
random_grid = {
    #'n_estimators': n_estimators,
'max_features': max_features,
'max_depth': max_depth,
'min_samples_split': min_samples_split,
'min_samples_leaf': min_samples_leaf
}

rf_random = RandomizedSearchCV(estimator = modelotree,  param_distributions =
    random_grid, n_iter = 100, cv = train_cv_index, verbose=2, random_state=35,
    n_jobs = -1, scoring="neg_mean_absolute_error")
```

Comparison tree default vs optimized in validation set!

```
rf_random.fit(train.iloc[:,0:75],train_target)
```

```
## Fitting 1 folds for each of 100 candidates, totalling 100 fits
## RandomizedSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
##                    estimator=DecisionTreeRegressor(random_state=1), n_iter=100,
##                    n_jobs=-1,
##                    param_distributions={'max_depth': [10, 20, 30, 40, 50, 60,
##                                                       70, 80, 90, 100, 110,
##                                                       120],
##                                         'max_features': ['log2', 'sqrt'],
##                                          'min_samples_leaf': [1, 3, 4],
##                                          'min_samples_split': [2, 6, 10]},
##                    random_state=35, scoring='neg_mean_absolute_error',
##                    verbose=2)
```

```
rf_random_pred=rf_random.predict(train_validation)

metrics.mean_absolute_error(tree_pred,train_target_validation)
```

```
## 3068353.430136986
```

```
metrics.mean_absolute_error(rf_random_pred,train_target_validation)
```

```
## 1363064.270440196
```

We found no way to align the training metric with mean absolute error

```
# USing SVMs
kernel=["linear","poly","rbf","sigmoid"]
degree=[1,2,3,4]
C=[x for x in np.linspace(start = 0.1, stop = 10, num = 10)]
shrinking=[True]

random_grid = {
   "kernel":kernel,
   "degree":degree,
   "C":C,
   "shrinking":shrinking}

scaler1=StandardScaler().fit(train_train,train_target_train)
train_train_st=scaler1.transform(train_train)
train_st=scaler1.transform(train.iloc[:,0:75])

SVR_estimatorS=SVR()
rs_svr = RandomizedSearchCV(estimator = SVR_estimatorS,
    param_distributions = random_grid,
    n_iter = 100,
    cv = train_cv_index,
    verbose=2, random_state=35, n_jobs = -1,scoring="neg_mean_absolute_error")
```

Comparison knn default vs optimized in validation set!

```
rs_svr.fit(train_st,train_target)
```

```
## Fitting 1 folds for each of 100 candidates, totalling 100 fits
## RandomizedSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
##                    estimator=SVR(), n_iter=100, n_jobs=-1,
##                    param_distributions={'C': [0.1, 1.2000000000000002,
##                                               2.3000000000000003,
##                                               3.4000000000000004, 4.5, 5.6, 6.7,
##                                               7.800000000000001, 8.9, 10.0],
##                                         'degree': [1, 2, 3, 4],
##                                         'kernel': ['linear', 'poly', 'rbf',
##                                                    'sigmoid'],
##                                         'shrinking': [True]},
##                    random_state=35, scoring='neg_mean_absolute_error',
##                    verbose=2)
```

```
SVR_pred_2=rs_svr.predict(scaler1.transform(train_validation))

metrics.mean_absolute_error(SVR_pred,train_target_validation)
```

```
## 6380505.183755267
```

```
metrics.mean_absolute_error(SVR_pred_2,train_target_validation)
```

```
## 5674228.961291356
```

- knn p=1 minkwoski distance equal one so it is consinstrant wieht mean absolute error.

```
#Using KNN
n_neighbors=[3,4,5,6,7]
weight=["uniform","distance"]
algorithm=["ball_tree","kd_tree","brute"]
leaf_size=[10,20,30,40]
p=[1]


param_grid={
    "n_neighbors":n_neighbors,
    #"weight":weight,
    "algorithm":algorithm,
    "leaf_size":leaf_size,
    "p":p
    }


knn_estimator_cv=KNeighborsRegressor()


rs_knn = GridSearchCV(estimator =knn_estimator_cv, param_grid=param_grid,
                       cv = train_cv_index, verbose=2,
                      n_jobs = -1,scoring="neg_mean_absolute_error")
```

Compraring knn default vs optimized

```
rs_knn.fit(train_st,train_target)
```

```
## Fitting 1 folds for each of 60 candidates, totalling 60 fits
## GridSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
##              estimator=KNeighborsRegressor(), n_jobs=-1,
##              param_grid={'algorithm': ['ball_tree', 'kd_tree', 'brute'],
##                          'leaf_size': [10, 20, 30, 40],
##                          'n_neighbors': [3, 4, 5, 6, 7], 'p': [1]},
##              scoring='neg_mean_absolute_error', verbose=2)
```

```
Knn_pred_2=rs_knn.predict(scaler1.transform(train_validation))

metrics.mean_absolute_error(knn_pred,train_target_validation)
```

```
## 2588455.571506849
```

```
metrics.mean_absolute_error(Knn_pred_2,train_target_validation)
```

```
## 2043643.6062622308
```

Tablita de resumen para mostrar que el mejor es knn con tuning 1743230.4429920174

Model evaluation: the method that was selected in model selection will be evaluated

on the test set

```
tree_test=rf_random.predict(scaler1.transform(test.iloc[:,0:75]))
metrics.mean_absolute_error(tree_test,test_target)
```

```
## 9614809.857402716
```

final model training

```

totaldata=train.append(test)
final_model_params=rf_random.best_params_
#final_scaling=StandardScaler().fit(X=totaldata.iloc[0:75])
#final_data_transformed=final_scaling.transform(X=totaldata.iloc[0:75])
modelofinal=DecisionTreeRegressor(**final_model_params,random_state=1)
modelofinal.fit(totaldata.iloc[:,0:75],totaldata.loc[:,"energy"])
```

```
## DecisionTreeRegressor(max_depth=10, max_features='log2', random_state=1)
```

```
modelofinal
```

```
## DecisionTreeRegressor(max_depth=10, max_features='log2', random_state=1)
```