# 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

```
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:

```
pipeKNN = make_pipeline(StandardScaler(), KNeighborsRegressor(p=1))
pipeKNN.fit(train_train,train_target_train)
```

```
## Pipeline(steps=[('standardscaler', StandardScaler()),
##                 ('kneighborsregressor', KNeighborsRegressor(p=1))])
```

```
knn_pred=pipeKNN.predict(train_validation)

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

```
## DecisionTreeRegressor()
```

```
tree_pred=modelotree.predict(train_validation)

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

```
## Pipeline(steps=[('standardscaler', StandardScaler()), ('svr', SVR())])
```

```
SVR_pred=pipeSVR.predict(train_validation)

metrics.mean_absolute_error(knn_pred,train_target_validation)
```

```
## 2530306.7342465753
```

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

```
## 3102145.0684931506
```

```
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

```
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 = ['auto', '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")
```

```


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(), n_iter=100, n_jobs=-1,
##                    param_distributions={'max_depth': [10, 20, 30, 40, 50, 60,
##                                                       70, 80, 90, 100, 110,
##                                                       120],
##                                         'max_features': ['auto', '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)
```

```
## 3102145.0684931506
```

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

```
## 1453808.607131932
```

```
# 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")

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
```

```
#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")





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))
```