PREDICTING SOLAR ENERGY PRODUCTION WITH SCIKIT-LEARN PIPELINE PART II

To start with we have selected the attributes associated to the first four blue points, and randomly selected 10% of the input attributes using Pandas. Then we put 10% of missing value at random places in the selected attributes.

1) The best imputation method among mean, median, and best scaler between minmaxscaler and robustscaler.

The machine learning method that is going to be used is KNN. First we create the two scalers: MInmaxScaler() and RobustScaler(). And for each scaler we create a pipeline with preprocessing and training methods: imputer, scaler and knn, second we define the parameters spaces, in this case the numbers of neighbors and impute strategy [mean, median]. Later we use GridSearchCV for hyper-parameter tuning and find the best imputation method and the number of neighbors.

For the comparisons we will use the train_validation dataset.

Imputer, Scalers and Knn creation.

```
53  #Imputation
54  from sklearn.impute import SimpleImputer
55  imputer = SimpleImputer()
56
57  #Scalers
58  from sklearn.preprocessing import MinMaxScaler, RobustScaler
59  minmax_scaler = MinMaxScaler()
60  minmax_scaler.fit(train_train_x)
61
62  robust_scaler = RobustScaler()
63  robust_scaler.fit(train_train_x)
64
65  #Model
66  knn = neighbors.KNeighborsRegressor()
```

Parameter grid for the Grid Search: imputer strategy and the number of neighbors

```
74  param_grid = {
75      'Model__n_neighbors': list(range(1,20,1))
76      , 'Impute__strategy': ['mean', 'median']
77      }
```

Pipeline with MinMax Scaler and GridSearchCV:

```
79  minmax_pipeline = Pipeline([
80      ('Impute', imputer),
81      ('Scale', minmax_scaler),
82      ('Model', knn)
83      ])
84
85  min_max_search = GridSearchCV(
86      minmax_pipeline
87      , param_grid
88      , scoring='neg_mean_absolute_error'
89      , cv = validation_partition
90  )
91
92  min_max_search.fit(train_x, train_y)
```

Pipeline with Robust Scaler and GridSearchCV:

```
robust_pipeline = Pipeline([
    ('Impute', imputer),
    ('Scale', robust_scaler),
    ('Model', knn)
    ])

robust_search = GridSearchCV(
    robust_pipeline
    , param_grid
    , scoring='neg_mean_absolute_error'
    , cv = validation_partition
)

robust_search.fit(train_x, train_y)
```

For the comparisons we will use the train_validation dataset, and we can visualize the results in the following table:

SCALER	Best Impute strategy	Neighbors	Best score (MAE)
MinMaxScaler	mean	19	**-2.319.305,878010094**
RobustScaler	median	17	-3.205.493,7775987107

The best possible pipeline is a combination of an imputation of missing values using the mean strategy, then a scaling process with a minmaxscaler, ending with a knn model using the 19 closest neighbors.

2) Feature selection by means of a pipeline

Given that we want to use both SelectKBest for feature selection and PCA for selecting the number of PCA components, we use FeatureUnions which allow us to define a step in the pipeline that combines features obtained from different sources.

Next we use a pipeline which contains the best imputation method, the optimal number of neighbors for KNN, the scaler, knn and the feature union, and we define the search space for variable selection and pca components.

We use the GridSearch for tuning the number of attributes selected by feature selection and the number of PCA components. Finally we obtain that the optimal number of attributes is 205 out of 300 and the best number of PCA components is 2.

```
#Select K Best Feeatures
imputer =  SimpleImputer(strategy='mean')
scaler = MinMaxScaler()
knn = neighbors.KNeighborsRegressor(n_neighbors=19)

selector = SelectKBest()
pca = PCA()


combined_feat = FeatureUnion([
    ("pca", pca),
    ("selector", selector)
    ])

model_pipe = Pipeline([
    ('imputer', imputer),
    ('scaler', scaler),
    ('features', combined_feat),
    ('knn_reg', knn)
    ])

selector_grid = {
    'features__selector__k': list(range(5,300,10))
    , 'features__pca__n_components': list(range(2,5,1))
    }
```

```
model_search = GridSearchCV(
     model_pipe
     , selector_grid
     , scoring='neg_mean_absolute_error'
     , cv = validation_partition
     , verbose=1
)

model_search.fit(train_x, train_y)
```

3) Extract the names of the attributes selected by feature selection pipeline

We extracted the names of the 205 selected attributes using the following code:

```
name, obj = model_pipe['features'].transformer_list[1]
train_x_new = train_x.iloc[:, obj.get_support(indices= True)]
kbest_vars = train_x_new.columns.tolist()
```

Selected Variables

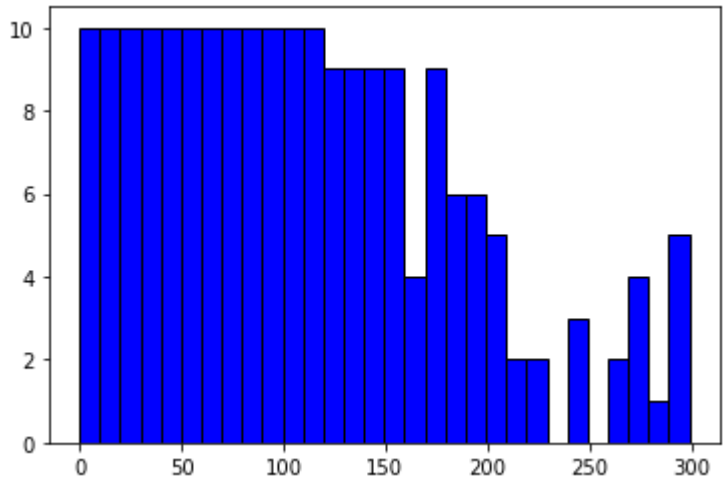
apcp_sf2_1	tmp_sfc4_1	tmin_2m2_2	dswrf_s2_3	tmp_sfc5_3	tcolc_e3_4
apcp_sf3_1	tmp_sfc5_1	tmin_2m3_2	dswrf_s3_3	ulwrf_s1_3	tcolc_e5_4
dlwrf_s5_1	ulwrf_s1_1	tmin_2m4_2	dswrf_s4_3	ulwrf_s2_3	tmax_2m1_4
dswrf_s1_1	ulwrf_s2_1	tmin_2m5_2	dswrf_s5_3	ulwrf_s3_3	tmax_2m2_4
dswrf_s2_1	ulwrf_s3_1	tmp_2m_1_2	pres_ms3_3	ulwrf_s4_3	tmax_2m3_4
dswrf_s3_1	ulwrf_s4_1	tmp_2m_2_2	tcdc_ea1_3	ulwrf_s5_3	tmax_2m4_4
dswrf_s4_1	ulwrf_s5_1	tmp_2m_3_2	tcdc_ea2_3	ulwrf_t1_3	tmax_2m5_4
dswrf_s5_1	ulwrf_t2_1	tmp_2m_4_2	tcdc_ea3_3	ulwrf_t2_3	tmin_2m1_4
spfh_2m5_1	ulwrf_t3_1	tmp_2m_5_2	tcdc_ea4_3	ulwrf_t3_3	tmin_2m2_4
tcdc_ea1_1	ulwrf_t4_1	tmp_sfc1_2	tcdc_ea5_3	ulwrf_t4_3	tmin_2m3_4
tcdc_ea2_1	ulwrf_t5_1	tmp_sfc2_2	tcolc_e1_3	ulwrf_t5_3	tmin_2m4_4
tcdc_ea3_1	uswrf_s2_1	tmp_sfc3_2	tcolc_e2_3	uswrf_s1_3	tmin_2m5_4
tcdc_ea4_1	uswrf_s3_1	tmp_sfc4_2	tcolc_e3_3	uswrf_s2_3	tmp_2m_1_4
tcolc_e1_1	uswrf_s4_1	tmp_sfc5_2	tcolc_e4_3	uswrf_s3_3	tmp_2m_2_4
tcolc_e2_1	uswrf_s5_1	ulwrf_s1_2	tcolc_e5_3	uswrf_s4_3	tmp_2m_3_4
tcolc_e3_1	dlwrf_s4_2	ulwrf_s2_2	tmax_2m1_3	uswrf_s5_3	tmp_2m_4_4
tmax_2m1_1	dswrf_s1_2	ulwrf_s3_2	tmax_2m2_3	apcp_sf1_4	tmp_2m_5_4

tmax_2m2_1	dswrf_s2_2	ulwrf_s4_2	tmax_2m3_3	apcp_sf2_4	tmp_sfc1_4
tmax_2m3_1	dswrf_s3_2	ulwrf_s5_2	tmax_2m4_3	apcp_sf3_4	tmp_sfc2_4
tmax_2m4_1	dswrf_s4_2	ulwrf_t2_2	tmax_2m5_3	dswrf_s1_4	tmp_sfc3_4
tmax_2m5_1	dswrf_s5_2	ulwrf_t3_2	tmin_2m1_3	dswrf_s2_4	tmp_sfc4_4
tmin_2m1_1	spfh_2m5_2	ulwrf_t4_2	tmin_2m2_3	dswrf_s3_4	tmp_sfc5_4
tmin_2m2_1	tcdc_ea2_2	ulwrf_t5_2	tmin_2m3_3	dswrf_s4_4	ulwrf_s1_4
tmin_2m3_1	tcdc_ea3_2	uswrf_s1_2	tmin_2m4_3	dswrf_s5_4	ulwrf_s2_4
tmin_2m4_1	tcdc_ea4_2	uswrf_s2_2	tmin_2m5_3	spfh_2m2_4	ulwrf_s3_4
tmin_2m5_1	tcolc_e2_2	uswrf_s3_2	tmp_2m_1_3	spfh_2m3_4	ulwrf_s4_4
tmp_2m_1_1	tcolc_e3_2	uswrf_s4_2	tmp_2m_2_3	spfh_2m4_4	ulwrf_s5_4
tmp_2m_2_1	tcolc_e4_2	uswrf_s5_2	tmp_2m_3_3	spfh_2m5_4	ulwrf_t2_4
tmp_2m_3_1	tmax_2m1_2	apcp_sf1_3	tmp_2m_4_3	tcdc_ea1_4	ulwrf_t3_4
tmp_2m_4_1	tmax_2m2_2	apcp_sf2_3	tmp_2m_5_3	tcdc_ea2_4	ulwrf_t4_4
tmp_2m_5_1	tmax_2m3_2	apcp_sf3_3	tmp_sfc1_3	tcdc_ea3_4	ulwrf_t5_4
tmp_sfc1_1	tmax_2m4_2	dlwrf_s4_3	tmp_sfc2_3	tcdc_ea5_4	uswrf_s2_4
tmp_sfc2_1	tmax_2m5_2	dlwrf_s5_3	tmp_sfc3_3	tcolc_e1_4	uswrf_s3_4
tmp_sfc3_1	tmin_2m1_2	dswrf_s1_3	tmp_sfc4_3	tcolc_e2_4	uswrf_s4_4
					uswrf_s5_4

Then we compared these attributes against the correlation of each feature with the *energy* variable.

```
linear_corr = train_x.corrwith(train_y)
ordered_corr = linear_corr[linear_corr.abs().sort_values(ascending = False).index]
ordered_index = ordered_corr.index.tolist()
kbest_index = []
for vn in kbest_vars:
    kbest_index.append(ordered_index.index(vn))


import matplotlib.pyplot as plt
plt.hist(kbest_index
         , color = 'blue'
         , edgecolor = 'black'
         , bins = 30
         )
```



This histogram shows the presence of a feature in the model and its index of correlation, being 0 the highest correlated variable, and 300 the least correlated variable. We can see that the 120 highest correlated variables are all in the model, but we also have some of the least correlated variables in it (variables 298 and 299 are included).

4) Evaluate the best pipeline on the test data.

In this part we are going to use the best pipeline and evaluate it on the test data. This evaluation gives us a MAE of **2,213,012.2303** over the test data.

```
In [145]: y_hat = model_pipe.predict(test_x)
     ...: print(metrics.mean_absolute_error(test_y, y_hat))
2213012.2303439365
```

At the end, we defined the best pipeline as follows:

```
156     imputer =  SimpleImputer(strategy='mean')
157     scaler = MinMaxScaler()
158     selector = SelectKBest(k = 205)
159     pca = PCA(n_components=2)
160     knn = neighbors.KNeighborsRegressor(n_neighbors=19)
161
162     combined_feat = FeatureUnion([
163          ("pca", pca),
164          ("selector", selector)
165          ])
166
167     model_pipe = Pipeline([
168          ('imputer', imputer),
169          ('scaler', scaler),
170          ('features', combined_feat),
171          ('knn_reg', knn)
172          ])
173
```

When comparing these results over the test set, with the results obtained in the part I of this assignment (*MAE over the test set of 2,271,745.778600663*) we do see a slight improvement between both models. This improvement might be attributed to the fact that this model uses almost 2.7 times more features than the last one, a different scaling approach, and a smarter selection of attributes over all.