exercicio2

June 8, 2021

1 1. Pré processamento

- Leia o arquivo Bias_correction_ucl.csv

[1]:
```
import pandas as pd
import warnings

warnings.filterwarnings('ignore')

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00514/
↪Bias_correction_ucl.csv"

original_ko_temp_forecast_df = pd.read_csv(url)
```

- Remova a coluna "Next_Tmin".
- Remova a coluna Date

[2]:
```
ko_temp_forecast_df = original_ko_temp_forecast_df.drop(['Next_Tmin', 'Date'],␣
↪axis=1)
```

- Remova as linhas que tem valor faltante. Das 7752 linhas originais sobram 7588

[3]:
```
ko_temp_forecast_df.shape
```

[3]:
```
(7752, 23)
```

[4]:
```
ko_temp_forecast_df.dropna(inplace=True)
```

[5]:
```
ko_temp_forecast_df.shape
```

[5]:
```
(7588, 23)
```

- O atributo de saída é Next_Tmax (a temperatura máxima no próximo dia). Vamos removê-lo do conjunto de dados para evitar contaminação durante o processo de normalização.

[6]:
```
next_tmax = ko_temp_forecast_df['Next_Tmax']
ko_temp_forecast_df.drop('Next_Tmax', axis=1, inplace=True)
```

- Centralize e normalize cada atributo de entrada

[7]:
```
from sklearn import preprocessing

prep_ko_temp_forecast_np = preprocessing.scale(ko_temp_forecast_df)
prep_ko_temp_forecast_np.shape
```

[7]: (7588, 22)

Verificando visualmente o resultado do pré processamento comparando os dados originais com os pré processados

[8]:
```
original_ko_temp_forecast_df
```

[8]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	\
0	1.0	2013-06-30	28.7	21.4	58.255688	
1	2.0	2013-06-30	31.9	21.6	52.263397	
2	3.0	2013-06-30	31.6	23.3	48.690479	
3	4.0	2013-06-30	32.0	23.4	58.239788	
4	5.0	2013-06-30	31.4	21.9	56.174095	
…	…	…	…	…	…	
7747	23.0	2017-08-30	23.3	17.1	26.741310	
7748	24.0	2017-08-30	23.3	17.7	24.040634	
7749	25.0	2017-08-30	23.2	17.4	22.933014	
7750	NaN	NaN	20.0	11.3	19.794666	
7751	NaN	NaN	37.6	29.9	98.524734	

	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	\
0	91.116364	28.074101	23.006936	6.818887	69.451805	
1	90.604721	29.850689	24.035009	5.691890	51.937448	
2	83.973587	30.091292	24.565633	6.138224	20.573050	
3	96.483688	29.704629	23.326177	5.650050	65.727144	
4	90.155128	29.113934	23.486480	5.735004	107.965535	
…	…	…	…	…	…	
7747	78.869858	26.352081	18.775678	6.148918	72.058294	
7748	77.294975	27.010193	18.733519	6.542819	47.241457	
7749	77.243744	27.939516	18.522965	7.289264	9.090034	
7750	58.936283	17.624954	14.272646	2.882580	-13.603212	
7751	100.000153	38.542255	29.619342	21.857621	213.414006	

	…	LDAPS_PPT2	LDAPS_PPT3	LDAPS_PPT4	lat	lon	DEM	\
0	…	0.000000	0.000000	0.000000	37.6046	126.991	212.3350	
1	…	0.000000	0.000000	0.000000	37.6046	127.032	44.7624	
2	…	0.000000	0.000000	0.000000	37.5776	127.058	33.3068	
3	…	0.000000	0.000000	0.000000	37.6450	127.022	45.7160	
4	…	0.000000	0.000000	0.000000	37.5507	127.135	35.0380	
…	…	…	…	…	…	…	…	
7747	…	0.000000	0.000000	0.000000	37.5372	126.891	15.5876	
7748	…	0.000000	0.000000	0.000000	37.5237	126.909	17.2956	
7749	…	0.000000	0.000000	0.000000	37.5237	126.970	19.5844	

```
7750  …    0.000000    0.000000    0.000000  37.4562  126.826   12.3700
7751  …   21.621661   15.841235   16.655469  37.6450  127.135  212.3350

         Slope  Solar radiation  Next_Tmax  Next_Tmin
0     2.785000      5992.895996       29.1       21.2
1     0.514100      5869.312500       30.5       22.5
2     0.266100      5863.555664       31.1       23.9
3     2.534800      5856.964844       31.7       24.3
4     0.505500      5859.552246       31.2       22.5
…        …              …          …          …
7747  0.155400      4443.313965       28.3       18.1
7748  0.222300      4438.373535       28.6       18.8
7749  0.271300      4451.345215       27.8       17.4
7750  0.098475      4329.520508       17.4       11.3
7751  5.178230      5992.895996       38.9       29.8

[7752 rows x 25 columns]
```

[9]:
```
prep_ko_temp_forecast_df = pd.DataFrame(data=prep_ko_temp_forecast_np,␣
↪columns=ko_temp_forecast_df.columns)
prep_ko_temp_forecast_df
```

[9]:
```
       station  Present_Tmax  Present_Tmin  LDAPS_RHmin  LDAPS_RHmax  \
0    -1.664607     -0.353318     -0.748029     0.104660     0.382768
1    -1.526052      0.725138     -0.664721    -0.305052     0.311697
2    -1.387498      0.624033      0.043400    -0.549344    -0.609425
3    -1.248943      0.758840      0.085054     0.103573     1.128335
4    -1.110389      0.556630     -0.539758    -0.037665     0.249244
…        …            …            …           …            …
7583  1.106481     -2.240616     -2.247578    -2.190405    -1.402898
7584  1.245036     -2.442826     -2.414195    -1.820788    -0.648815
7585  1.383590     -2.173212     -2.539157    -2.050081    -1.318376
7586  1.522145     -2.173212     -2.289232    -2.234735    -1.537141
7587  1.660699     -2.206914     -2.414195    -2.310467    -1.544257

      LDAPS_Tmax_lapse  LDAPS_Tmin_lapse  LDAPS_WS  LDAPS_LH  LDAPS_CC1  …  \
0            -0.525269         -0.215525 -0.126423  0.206603  -0.513123  …
1             0.078334          0.223368 -0.644133 -0.313359  -0.545304  …
2             0.160080          0.449896 -0.439100 -1.244497  -0.606944  …
3             0.028710         -0.079238 -0.663353  0.096026  -0.580143  …
4            -0.171981         -0.010803 -0.624327  1.349989  -0.827872  …
…               …                 …        …         …          …  …
7583         -0.614083         -2.223667 -0.225479 -1.569844  -1.203497  …
7584         -0.990460         -2.432415 -0.609132  0.583486  -1.151793  …
7585         -1.110333         -2.021883 -0.434188  0.283984  -1.290699  …
7586         -0.886737         -2.039881 -0.253241 -0.452772  -1.268430  …
7587         -0.570995         -2.129768  0.089654 -1.585402  -1.218554  …
```

```
      LDAPS_CC4  LDAPS_PPT1  LDAPS_PPT2  LDAPS_PPT3  LDAPS_PPT4       lat  \
0     -0.660441   -0.305589   -0.275777   -0.239969   -0.224971  1.186076
1     -0.673074   -0.305589   -0.275777   -0.239969   -0.224971  1.186076
2     -0.616249   -0.305589   -0.275777   -0.239969   -0.224971  0.650626
3     -0.647336   -0.305589   -0.275777   -0.239969   -0.224971  1.987268
4     -0.506152   -0.305589   -0.275777   -0.239969   -0.224971  0.117159
…           …           …           …           …           …         …
7583  -1.159890   -0.305589   -0.275777   -0.239969   -0.224971  0.117159
7584  -1.107407   -0.305589   -0.275777   -0.239969   -0.224971 -0.686016
7585  -1.177177   -0.305589   -0.275777   -0.239969   -0.224971 -0.150566
7586  -1.177177   -0.305589   -0.275777   -0.239969   -0.224971 -0.418291
7587  -1.174034   -0.305589   -0.275777   -0.239969   -0.224971 -0.418291

            lon       DEM     Slope  Solar radiation
0    -0.005302  2.769091  1.111162         1.510565
1     0.512280 -0.315828 -0.543220         1.222997
2     0.840503 -0.526719 -0.723891         1.209602
3     0.386040 -0.298272  0.928888         1.194265
4     1.812547 -0.494848 -0.549485         1.200286
…          …         …         …              …
7583  0.613271 -0.655747 -0.500966        -2.065599
7584  1.193973 -0.735482 -0.820711        -2.098689
7585 -1.267697 -0.852919 -0.804538        -2.095175
7586 -1.040466 -0.821476 -0.755800        -2.106671
7587 -0.270405 -0.779341 -0.720103        -2.076487

[7588 rows x 22 columns]
```

Aparentemente está ok, vamos então usar o conjunto nparray de predictors "prep_ko_temp_forecast_np" para a próxima fase pois o sklearn lida melhor com ele do que com pandas DataFrames.

2 2. Busca de hiperparâmetros e modelos utilizando:

- 5 fold cross validation como técnica de resampling separando os dados em conjuntos de 70% de treino e 30% de teste;
- RMSE como medida de erro;
- Busca aleatória de hiperparametros (nos modelos aplicáveis)

[10]:
```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
import numpy as np
import random
```

```
# Numero de vezes em que serão gerados valores aleatórios de hiperparâmetros
HYPERPARAM_SAMPLING_N = 10

def rmse_positive(model):
    # Calcula o valor de RMSE no conjunto de dados pré processado dado o modelo
    # especificado
    scores = cross_val_score(model, \
                             prep_ko_temp_forecast_np,
                             next_tmax, \
                             cv = ShuffleSplit(n_splits=5, test_size=0.3), \
                             scoring=('neg_root_mean_squared_error'))
    return np.mean(- scores)
```

2.1 2.1 Linear

[11]:
```
from sklearn.linear_model import LinearRegression

traditional_lr_rmse = rmse_positive(LinearRegression())
```

2.2 2.2 Linear com regularização L2

Sendo alpha escolhido com: 10 números aleatórios entre 10-3 e 103

[13]:
```
from sklearn.linear_model import Ridge

#-------------------------------------------------------------------------
# Default hyperparameters
default_ridge_lr_RMSE = rmse_positive(Ridge())

#-------------------------------------------------------------------------
# Tunning hyperparams
best_ridge_lr_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):
    ridge = Ridge(alpha=random.uniform(10**-3, 10**3))

    mean_iter_scores = rmse_positive(ridge)

    if best_ridge_lr_RMSE > mean_iter_scores:
        best_ridge_lr_RMSE = mean_iter_scores
```

2.3 2.3 Linear com regularização L1

Sendo alpha escolhido com: 10 números aleatórios entre 10-3 e 103

[16]:
```
from sklearn.linear_model import Lasso

#-------------------------------------------------------------------------
```

```
# Default hyperparameters
default_lasso_lr_RMSE = rmse_positive(Lasso())

#------------------------------------------------------------------------
# Tunning hyperparams

best_lasso_lr_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):
    lasso = Lasso(alpha=random.uniform(10**-3, 10**3))

    mean_iter_scores = rmse_positive(lasso)

    if best_lasso_lr_RMSE > mean_iter_scores:
        best_lasso_lr_RMSE = mean_iter_scores
```

2.4 2.4 SVM Linear

Sendo: * epsilon = 0.1 ou 0.3 * 2 -5 >= C >= 2 15

[18]:
```
from sklearn.svm import LinearSVR

#------------------------------------------------------------------------
# Default hyperparameters
default_svr_linear_RMSE = rmse_positive(LinearSVR())

#------------------------------------------------------------------------
# Tunning hyperparams

best_svr_linear_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):
    svr = LinearSVR(C = random.uniform(2**-5, 2**15), \
                    epsilon = random.choice([0.1, 0.3]))

    mean_iter_scores = rmse_positive(svr)

    if best_svr_linear_RMSE > mean_iter_scores:
        best_svr_linear_RMSE = mean_iter_scores
```

2.5 2.5 SVM com kernel RBF

Sendo: * epsilon = 0.1 ou 0.3 * 2 -5 >= C >= 2 15 * 2 -9 >= gamma >= 2 3

[21]:
```
from sklearn.svm import SVR

#------------------------------------------------------------------------
# Default hyperparameters
```

```
default_svr_rbf_RMSE = rmse_positive(SVR(kernel='rbf'))

#-------------------------------------------------------------------------
# Tunning hyperparams

best_svr_rbf_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):

    svr = SVR(epsilon = random.choice([0.1, 0.3]), \
              C = random.uniform(2**-5, 2**15), \
              gamma = random.uniform(2**-9, 2**3), \
              kernel='rbf')

    mean_iter_scores = rmse_positive(svr)

    if best_svr_rbf_RMSE > mean_iter_scores:
        best_svr_rbf_RMSE = mean_iter_scores
```

2.6 2.6 KNN

Sendo K: 10 números aleatórios entre 1 e 1000

[23]:
```
from sklearn.neighbors import KNeighborsRegressor

#-------------------------------------------------------------------------
# Default hyperparameters
default_knn_regr_RMSE = rmse_positive(KNeighborsRegressor())

#-------------------------------------------------------------------------
# Tunning hyperparams

best_knn_regr_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):
    knn_regressor = KNeighborsRegressor(n_neighbors=random.randint(1, 1000))

    mean_iter_scores = rmse_positive(knn_regressor)

    if best_knn_regr_RMSE > mean_iter_scores:
        best_knn_regr_RMSE = mean_iter_scores
```

2.7 2.7 MLP

Neurônios na camada do meio: de 5 a 20, de três em três

[24]:
```
from sklearn.neural_network import MLPRegressor
```

```
#-------------------------------------------------------------------------
# Default hyperparameters
default_mlp_regr_RMSE = rmse_positive(MLPRegressor())

#-------------------------------------------------------------------------
# Tunning hyperparams

best_mlp_regr_RMSE = 10**3

neurons = 5

while neurons <= 20:
    mlp_regressor = MLPRegressor(hidden_layer_sizes=neurons)

    mean_iter_scores = rmse_positive(mlp_regressor)

    if best_mlp_regr_RMSE > mean_iter_scores:
        best_mlp_regr_RMSE = mean_iter_scores

    neurons +=3
```

2.8 2.8 Arvore de decisão

- Usando prunning com ccp_alpha sendo 10 números aleatórios entre 0.0 e 0.04.

[26]:
```
from sklearn.tree import DecisionTreeRegressor

#-------------------------------------------------------------------------
# Default hyperparameters
default_dec_tree_RMSE = rmse_positive(DecisionTreeRegressor())

#-------------------------------------------------------------------------
# Tunning hyperparams

best_dec_tree_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):
    dec_tree_regressor = DecisionTreeRegressor(ccp_alpha=random. \
                                               uniform(0.0, 0.04))

    mean_iter_scores = rmse_positive(dec_tree_regressor)

    if best_dec_tree_RMSE > mean_iter_scores:
        best_dec_tree_RMSE = mean_iter_scores
```

2.9 2.9 Random Forest

Usando todas as combinações dos valores abaixo. * n_estimators: use os valores: 10, 100 e 1000; * max_features: use os valores 5, 10, e 22.

[27]:
```
from sklearn.ensemble import RandomForestRegressor

#-------------------------------------------------------------------------
# Default hyperparameters
default_rand_for_RMSE = rmse_positive(RandomForestRegressor())

#-------------------------------------------------------------------------
# Tunning hyperparams

best_rand_for_RMSE = 10**3
n_estimators = [10, 100, 1000]
max_features = [5, 10, 22]

for estimator in n_estimators:
    for max_feature in max_features:
        dec_tree_regressor = RandomForestRegressor(n_estimators = estimator, \
                                                   max_features = max_feature)
        mean_iter_scores = rmse_positive(dec_tree_regressor)

        if best_rand_for_RMSE > mean_iter_scores:
            best_rand_for_RMSE = mean_iter_scores
```

2.10 2.10 GBM

Selecionando 10 trinca aleatórias ente: * n_estimators: de 5 a 100; * learning_rate: de 0.01 a 0.3; * max_depth: 2 ou 3.

[28]:
```
from sklearn.ensemble import GradientBoostingRegressor

#-------------------------------------------------------------------------
# Default hyperparameters
default_gbm_RMSE = rmse_positive(GradientBoostingRegressor())

#-------------------------------------------------------------------------
# Tunning hyperparams

best_gbm_RMSE = 10**3

for i in range(HYPERPARAM_SAMPLING_N):
    gbm = GradientBoostingRegressor(\
            max_depth = random.choice([2,3]), \
            learning_rate = random.uniform(0.01, 0.3), \
            n_estimators = random.randint(5, 100)
```

```
            )

    mean_iter_scores = rmse_positive(gbm)

    if best_gbm_RMSE > mean_iter_scores:
        best_gbm_RMSE = mean_iter_scores
```

3 3. Tabela final

Tabela final com cada classificador, os valores do RMSE com valore default para os hiperparametros, e o valor do RMSE com o melhor valor dos hiperparametros.

[29]:
```
columns = ['Regressor', 'RMSE default', 'RMSE hiperparametros']
data = [
        ['Linear',            traditional_lr_rmse,    'N/A'], \
        ['Ridge',             default_ridge_lr_RMSE,   best_ridge_lr_RMSE], \
        ['Lasso',             default_lasso_lr_RMSE,   best_lasso_lr_RMSE], \
        ['SVR Linear',        default_svr_linear_RMSE, best_svr_linear_RMSE], \
        ['SVR RBF',           default_svr_rbf_RMSE,    best_svr_rbf_RMSE], \
        ['KNN',               default_knn_regr_RMSE,   best_knn_regr_RMSE], \
        ['MLP',               default_mlp_regr_RMSE,   best_mlp_regr_RMSE], \
        ['Arvore de decisao', default_dec_tree_RMSE,   best_dec_tree_RMSE], \
        ['Random Forest',     default_rand_for_RMSE,   best_rand_for_RMSE], \
        ['GBM',               default_gbm_RMSE,        best_gbm_RMSE], \
       ]

final_table = pd.DataFrame(data, columns = columns)
```

[30]:
```
final_table
```

[30]:

	Regressor	RMSE default	RMSE hiperparametros
0	Linear	1.454240	N/A
1	Ridge	1.466257	1.452185
2	Lasso	1.988330	3.08591
3	SVR Linear	1.465918	1.946544
4	SVR RBF	1.183737	2.746445
5	KNN	1.275011	1.63229
6	MLP	1.280703	1.954181
7	Arvore de decisao	1.500880	1.407038
8	Random Forest	1.000110	0.940968
9	GBM	1.219926	1.095765