Trabalho 2

June 13, 2021

1 Tarefa 2 - MO432A

Alunos: * Andreza - RA: 164213 * Gil - RA: 225323 * Yan - RA: 118982

2 Tabela de Resultados

Abaixo apresentamos as tabelas com os resultados obtidos. Podemos verificar que, na maioria das vezes, os melhores hiperparâmetros retornaram RSME menores se comparado com os hiperparâmetros default. Entretanto, o melhor classificador, o SVM Linear, apresentou o resultado mais baixo do RMSE pra o dafault.

Classificador	RSME otimizado	RMSE default
Regressão Linear	**1.2560**	**1.2560**
Regressão Linear com L2	**1.2539**	1.2559
Regressão Linear com L1	**1.2752**	1.4227
SVM linear	1.2498	**1.2494**
SVM RBF	**1.2843**	1.2992
KNN	**1.3832**	1.3881
MLP	1.5234	**1.4411**
Árvore de Decisão	**1.3588**	**1.3588**
Random Forest	**1.2757**	1.2859
GBM	1.2660	**1.2633**

2.1 Ler o arquivo

Trata-se do arquivo Bias_correction_ucl.csv, devendo ser removida a coluna Next_Tmin, a coluna Date,e as linhas que tem valor faltante.

[1]:
```
import sklearn
import pandas as pd
import numpy as np
import random

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, cross_validate
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import LinearSVR, SVR
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
df_original = pd.read_csv('Bias_correction_ucl.csv')
df_original.head()
```

[2]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	\
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	
2	3.0	2013-06-30	31.6	23.3	48.690479	83.973587	
3	4.0	2013-06-30	32.0	23.4	58.239788	96.483688	
4	5.0	2013-06-30	31.4	21.9	56.174095	90.155128	

	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	…	LDAPS_PPT2	\
0	28.074101	23.006936	6.818887	69.451805	…	0.0	
1	29.850689	24.035009	5.691890	51.937448	…	0.0	
2	30.091292	24.565633	6.138224	20.573050	…	0.0	
3	29.704629	23.326177	5.650050	65.727144	…	0.0	
4	29.113934	23.486480	5.735004	107.965535	…	0.0	

	LDAPS_PPT3	LDAPS_PPT4	lat	lon	DEM	Slope	\
0	0.0	0.0	37.6046	126.991	212.3350	2.7850	
1	0.0	0.0	37.6046	127.032	44.7624	0.5141	
2	0.0	0.0	37.5776	127.058	33.3068	0.2661	
3	0.0	0.0	37.6450	127.022	45.7160	2.5348	
4	0.0	0.0	37.5507	127.135	35.0380	0.5055	

	Solar radiation	Next_Tmax	Next_Tmin
0	5992.895996	29.1	21.2
1	5869.312500	30.5	22.5
2	5863.555664	31.1	23.9
3	5856.964844	31.7	24.3
4	5859.552246	31.2	22.5

[5 rows x 25 columns]

[3]:
```
df = df_original.drop(['Date', 'Next_Tmin'], axis=1)
print("Número de linhas: {}\nNúmero de colunas: {}".format(df.shape[0], df.
 ↪shape[1]))
df.head()
```

```
Número de linhas: 7752
Número de colunas: 23
```

[3]:
```
   station  Present_Tmax  Present_Tmin  LDAPS_RHmin  LDAPS_RHmax  \
0      1.0          28.7          21.4    58.255688    91.116364
1      2.0          31.9          21.6    52.263397    90.604721
2      3.0          31.6          23.3    48.690479    83.973587
3      4.0          32.0          23.4    58.239788    96.483688
4      5.0          31.4          21.9    56.174095    90.155128

   LDAPS_Tmax_lapse  LDAPS_Tmin_lapse  LDAPS_WS    LDAPS_LH  LDAPS_CC1  …  \
0         28.074101         23.006936  6.818887   69.451805   0.233947  …
1         29.850689         24.035009  5.691890   51.937448   0.225508  …
2         30.091292         24.565633  6.138224   20.573050   0.209344  …
3         29.704629         23.326177  5.650050   65.727144   0.216372  …
4         29.113934         23.486480  5.735004  107.965535   0.151407  …

   LDAPS_PPT1  LDAPS_PPT2  LDAPS_PPT3  LDAPS_PPT4      lat      lon       DEM  \
0         0.0         0.0         0.0         0.0  37.6046  126.991  212.3350
1         0.0         0.0         0.0         0.0  37.6046  127.032   44.7624
2         0.0         0.0         0.0         0.0  37.5776  127.058   33.3068
3         0.0         0.0         0.0         0.0  37.6450  127.022   45.7160
4         0.0         0.0         0.0         0.0  37.5507  127.135   35.0380

    Slope  Solar radiation  Next_Tmax
0  2.7850      5992.895996       29.1
1  0.5141      5869.312500       30.5
2  0.2661      5863.555664       31.1
3  2.5348      5856.964844       31.7
4  0.5055      5859.552246       31.2

[5 rows x 23 columns]
```

[4]:
```
df.dropna(inplace = True)
print("Número de linhas: {}\nNúmero de colunas: {}".format(df.shape[0], df.
 ↪shape[1]))
df.head()
```

```
Número de linhas: 7588
Número de colunas: 23
```

[4]:
```
   station  Present_Tmax  Present_Tmin  LDAPS_RHmin  LDAPS_RHmax  \
0      1.0          28.7          21.4    58.255688    91.116364
1      2.0          31.9          21.6    52.263397    90.604721
2      3.0          31.6          23.3    48.690479    83.973587
3      4.0          32.0          23.4    58.239788    96.483688
4      5.0          31.4          21.9    56.174095    90.155128
```

```
   LDAPS_Tmax_lapse  LDAPS_Tmin_lapse  LDAPS_WS    LDAPS_LH  LDAPS_CC1  …  \
0         28.074101         23.006936  6.818887   69.451805   0.233947  …
1         29.850689         24.035009  5.691890   51.937448   0.225508  …
2         30.091292         24.565633  6.138224   20.573050   0.209344  …
3         29.704629         23.326177  5.650050   65.727144   0.216372  …
4         29.113934         23.486480  5.735004  107.965535   0.151407  …

   LDAPS_PPT1  LDAPS_PPT2  LDAPS_PPT3  LDAPS_PPT4      lat      lon       DEM  \
0         0.0         0.0         0.0         0.0  37.6046  126.991  212.3350
1         0.0         0.0         0.0         0.0  37.6046  127.032   44.7624
2         0.0         0.0         0.0         0.0  37.5776  127.058   33.3068
3         0.0         0.0         0.0         0.0  37.6450  127.022   45.7160
4         0.0         0.0         0.0         0.0  37.5507  127.135   35.0380

    Slope  Solar radiation  Next_Tmax
0  2.7850      5992.895996       29.1
1  0.5141      5869.312500       30.5
2  0.2661      5863.555664       31.1
3  2.5348      5856.964844       31.7
4  0.5055      5859.552246       31.2

[5 rows x 23 columns]
```

[5]:
```
x = df.drop(['Next_Tmax'], axis=1)
x = StandardScaler().fit_transform(x)
x = pd.DataFrame(x)
print("Número de linhas: {}\nNúmero de colunas: {}".format(x.shape[0], x.
 ↪shape[1]))
x.head()
```

```
Número de linhas: 7588
Número de colunas: 22
```

[5]:
```
          0         1         2         3         4         5         6   \
0 -1.664607 -0.353318 -0.748029  0.104660  0.382768 -0.525269 -0.215525
1 -1.526052  0.725138 -0.664721 -0.305052  0.311697  0.078334  0.223368
2 -1.387498  0.624033  0.043400 -0.549344 -0.609425  0.160080  0.449896
3 -1.248943  0.758840  0.085054  0.103573  1.128335  0.028710 -0.079238
4 -1.110389  0.556630 -0.539758 -0.037665  0.249244 -0.171981 -0.010803

          7         8         9   …        12        13        14        15  \
0 -0.126423  0.206603 -0.513123  … -0.660441 -0.305589 -0.275777 -0.239969
1 -0.644133 -0.313359 -0.545304  … -0.673074 -0.305589 -0.275777 -0.239969
2 -0.439100 -1.244497 -0.606944  … -0.616249 -0.305589 -0.275777 -0.239969
3 -0.663353  0.096026 -0.580143  … -0.647336 -0.305589 -0.275777 -0.239969
4 -0.624327  1.349989 -0.827872  … -0.506152 -0.305589 -0.275777 -0.239969
```

```
         16        17        18        19        20        21
0 -0.224971  1.186076 -0.005302  2.769091  1.111162  1.510565
1 -0.224971  1.186076  0.512280 -0.315828 -0.543220  1.222997
2 -0.224971  0.650626  0.840503 -0.526719 -0.723891  1.209602
3 -0.224971  1.987268  0.386040 -0.298272  0.928888  1.194265
4 -0.224971  0.117159  1.812547 -0.494848 -0.549485  1.200286

[5 rows x 22 columns]
```

[6]:
```
y = df.get(['Next_Tmax'])
print("Número de linhas: {}\nNúmero de colunas: {}".format(y.shape[0], y.
 ↪shape[1]))
y.head()
```

```
Número de linhas: 7588
Número de colunas: 1
```

[6]:
```
   Next_Tmax
0       29.1
1       30.5
2       31.1
3       31.7
4       31.2
```

2.2 Cross validation, medida de erro e busca de hiperparametros

- Usar 5-fold cross validation
- Usar RMSE como medida de erro
- Criar distribuição uniforme para a busca aleatório de hiperparametro, dnetro do intervalo especificado em cada problema/regressor

2.3 Para cada um dos regressores abaixo

- Reportar o RMSE da melhor combinação de hiperparametros e o valor dos hiperparametros encontrados
- Reportar o RMSE do uso dos valores default do SKLearn para os hiperparametros que buscamos

2.3.1 Linear

[7]:
```
cross_val = cross_validate(
    LinearRegression(),
    x,
    y,
    scoring=('neg_root_mean_squared_error')
)
rmse_medio =  np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
```

```
print("RMSE = {}".format(rmse_medio))
```

```
RMSE = 1.2560040654482534
```

2.3.2 Regularização L2

[8]:
```
alpha_list = 10**(np.random.uniform(-3, 3, 10))
final_rmse = np.inf
for alpha_value in alpha_list:
    cross_val = cross_validate(
        Ridge(alpha_value),
        x, y,
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_alpha = alpha_value
print("Resultado da busca por hiperparâmetros:")
print("Melhor RMSE = {}".format(final_rmse))
print("Melhor alpha = {}".format(final_alpha))
```

```
Resultado da busca por hiperparâmetros:
Melhor RMSE = 1.2539586160813418
Melhor alpha = 339.52984406057004
```

[9]:
```
cross_val = cross_validate(
            Ridge(),
            x, y,
            scoring=('neg_root_mean_squared_error'))
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.2559766342954395
```

2.3.3 Regularização L1

[10]:
```
alpha_list = 10**(np.random.uniform(-3, 3, 10))
final_rmse = np.inf
for alpha_value in alpha_list:
    cross_val = cross_validate(
        Lasso(alpha_value),
        x,
        y,
        scoring=('neg_root_mean_squared_error')
```

```
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_alpha = alpha_value
print("Melhor RMSE = {}".format(final_rmse))
print("Valor do alpha = {}".format(final_alpha))
```

```
Melhor RMSE = 1.275238319727372
Valor do alpha = 0.17118486715846912
```

[11]:
```
cross_val = cross_validate(
            Lasso(),
            x, y,
            scoring=('neg_root_mean_squared_error'))
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.4227986510522177
```

2.3.4 SVM Linear

[12]:
```
epsilon_list = [0.1, 0.3]
C_list = 2**(np.random.uniform(-5, 15, 10))
final_rmse = np.inf
for C_value in C_list:
    epsilon_value = random.choice(epsilon_list)
    cross_val = cross_validate(
        LinearSVR(epsilon = epsilon_value, C = C_value, max_iter = 3000),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_epsilon = epsilon_value
        final_C = C_value
print("Melhor RMSE = {}".format(final_rmse))
print("Valor do epsilon = {}".format(final_epsilon))
print("Valor do C = {}".format(final_C))
```

```
Melhor RMSE = 1.2498223491872777
Valor do epsilon = 0.1
Valor do C = 0.049891309692652284
```

[13]:
```
cross_val = cross_validate(
        LinearSVR(),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
    )
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.249403604548239
```

2.3.5 SVM com kernel RBF

[14]:
```
epsilon_list = [0.1, 0.3]
C_list = 2**(np.random.uniform(-5, 15, 10))
gamma_list = 2**(np.random.uniform(-9, 3, 10))
final_rmse = np.inf

for C_value in C_list:
    epsilon_value = random.choice(epsilon_list)
    gamma_value = random.choice(gamma_list)
    cross_val = cross_validate(
        SVR(gamma = gamma_value,
            C = C_value,
            epsilon = epsilon_value,
            kernel='rbf'),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_epsilon = epsilon_value
        final_C = C_value
        final_gamma = gamma_value

print("Melhor RMSE = {}".format(final_rmse))
print("Valor do epsilon = {}".format(final_epsilon))
print("Valor do C = {}".format(final_C))
print("Valor do gamma = {}".format(final_gamma))
```

```
Melhor RMSE = 1.2843368535684645
Valor do epsilon = 0.1
Valor do C = 1.6559911674631167
Valor do gamma = 0.027627878514778933
```

[15]:
```
cross_val = cross_validate(
        SVR(),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
    )
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.299228674531788
```

2.3.6 KNN

[16]:
```
K_list = random.sample(range(1, 1001), 10)
final_rmse = np.inf
for K_value in K_list:
    cross_val = cross_validate(
        KNeighborsRegressor(K_value),
        x,
        y,
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_K = K_value
print("Melhor RMSE = {}".format(final_rmse))
print("Valor do K = {}".format(final_K))
```

```
Melhor RMSE = 1.383205978661817
Valor do K = 230
```

[17]:
```
cross_val = cross_validate(
        KNeighborsRegressor(),
        x,
        y,
        scoring=('neg_root_mean_squared_error')
    )
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.3881824886378122
```

2.3.7 MLP

[18]:
```
final_rmse = np.inf
for Hlayers_value in range(5, 21, 3):
    cross_val = cross_validate(
        MLPRegressor(Hlayers_value),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_Hlayers = Hlayers_value
print("Melhor RMSE = {}".format(final_rmse))
print("Camadas escondidas = {}".format(final_Hlayers))
```

```
Melhor RMSE = 1.523411446769098
Camadas escondidas = 20
```

[19]:
```
cross_val = cross_validate(
        MLPRegressor(),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
)
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.4411338085156025
```

2.3.8 Árvore de decisão

[20]:
```
ccp_alpha_list = np.random.uniform(0, 0.04, 10)
final_rmse = np.inf
for ccp_alpha_value in ccp_alpha_list:
    cross_val = cross_validate(
        DecisionTreeRegressor(ccp_alpha = ccp_alpha_value),
        x,
        y,
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_ccp_alpha = ccp_alpha_value
```

```
print("Melhor RMSE = {}".format(final_rmse))
print("Valor do ccp_alpha = {}".format(final_ccp_alpha))
```

```
Melhor RMSE = 1.3588060137522588
Valor do ccp_alpha = 0.03695600724680763
```

[21]:
```
cross_val = cross_validate(
        DecisionTreeRegressor(),
        x,
        y,
        scoring=('neg_root_mean_squared_error')
    )
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.3588060137522588
```

2.3.9 Random Forest

[22]:
```
n_estimators_list = [10, 100, 1000]
max_features_list = [5, 10, 22]
final_rmse = np.inf
for n_estimators_value in n_estimators_list:
    for max_features_value in max_features_list:
        cross_val = cross_validate(
            RandomForestRegressor(n_estimators = n_estimators_value,␣
 ↪max_features = max_features_value),
            x,
            np.ravel(y),
            scoring=('neg_root_mean_squared_error')
        )
        rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
        if rmse_medio < final_rmse:
            final_rmse = rmse_medio
            final_n_estimators = n_estimators_value
            final_max_features = max_features_value
print("Melhor RMSE = {}".format(final_rmse))
print("Valor de estimadores = {}".format(final_n_estimators))
print("Valor máximo de atributos = {}".format(final_max_features))
```

```
Melhor RMSE = 1.2757094678467147
Valor de estimadores = 1000
Valor máximo de atributos = 10
```

[23]:
```
cross_val = cross_validate(
    RandomForestRegressor(),
    x,
```

```
    np.ravel(y),
    scoring=('neg_root_mean_squared_error')
)
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.285901400528785
```

2.3.10 GBM

[24]:
```
n_estimators_list = random.sample(range(5, 101), 10)
learning_rate_list = np.random.uniform(0.01, 0.3, 10)
max_depth_list = [2, 3]
final_rmse = np.inf

for n_estimators_value in n_estimators_list:
    learning_rate_value = random.choice(learning_rate_list)
    max_depth_value = random.choice(max_depth_list)
    cross_val = cross_validate(
        GradientBoostingRegressor(
            learning_rate = learning_rate_value,
            n_estimators = n_estimators_value,
            max_features = max_features_value
        ),
        x,
        np.ravel(y),
        scoring=('neg_root_mean_squared_error')
    )
    rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
    if rmse_medio < final_rmse:
        final_rmse = rmse_medio
        final_n_estimators = n_estimators_value
        final_learning_rate = learning_rate_value
        final_max_depth = max_depth_value

print("Melhor RMSE = {}".format(final_rmse))
print("Valor de estimadores = {}".format(final_n_estimators))
print("Valor da taxa de aprendizado = {}".format(final_learning_rate))
print("Valor da profundidade máxima = {}".format(final_max_depth))
```

```
Melhor RMSE = 1.266012329398822
Valor de estimadores = 29
Valor da taxa de aprendizado = 0.21844340584688535
Valor da profundidade máxima = 2
```

[25]:
```
cross_val = cross_validate(
    GradientBoostingRegressor(),
    x,
    np.ravel(y),
    scoring=('neg_root_mean_squared_error')
)
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
rmse_medio = np.sqrt(np.mean(np.absolute(cross_val['test_score'])))
print("Resultado usando valores default:")
print("RMSE = {}".format(rmse_medio))
```

```
Resultado usando valores default:
RMSE = 1.2633919172119057
```