Tarefa#2 - Aprendizado supervisionado - MO432

June 8, 2021

Universidade Estadual de Campinas (UNICAMP), Instituto de Computação (IC)

Prof. Jacques Wainer, 2021s1

[1]:
```
# RA & Name
print('265673: ' + 'Gabriel Luciano Gomes')
print('264965: ' + 'Décio Luiz Gazzoni Filho')
print('192880: ' + 'Lucas Borges Rondon')
```

```
265673: Gabriel Luciano Gomes
264965: Décio Luiz Gazzoni Filho
192880: Lucas Borges Rondon
```

1 Leitura da base de dados

[2]:
```
import pandas as pd
import requests
import io

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00514/
 ↪Bias_correction_ucl.csv"
s = requests.get(url).content

db = pd.read_csv(io.StringIO(s.decode('utf-8')))
db.head()
```

[2]:
```
   station        Date  Present_Tmax  …  Solar radiation  Next_Tmax  Next_Tmin
0      1.0  2013-06-30          28.7  …      5992.895996       29.1       21.2
1      2.0  2013-06-30          31.9  …      5869.312500       30.5       22.5
2      3.0  2013-06-30          31.6  …      5863.555664       31.1       23.9
3      4.0  2013-06-30          32.0  …      5856.964844       31.7       24.3
4      5.0  2013-06-30          31.4  …      5859.552246       31.2       22.5

[5 rows x 25 columns]
```

[3]:
```
print(f'A base de dados possui {db.shape[0]} instâncias com {db.shape[1]}␣
 ↪atributos.')
```

```
A base de dados possui 7752 instâncias com 25 atributos.
```

2 Pré-processamento dos dados

2.1 Remoção das colunas “Next_Tmin” e “Date”

```
[4]: db = db.drop(columns=['Next_Tmin', 'Date'])

     db.head()
```

```
[4]:    station  Present_Tmax  Present_Tmin  …   Slope  Solar radiation  Next_Tmax
     0      1.0          28.7          21.4  …  2.7850      5992.895996       29.1
     1      2.0          31.9          21.6  …  0.5141      5869.312500       30.5
     2      3.0          31.6          23.3  …  0.2661      5863.555664       31.1
     3      4.0          32.0          23.4  …  2.5348      5856.964844       31.7
     4      5.0          31.4          21.9  …  0.5055      5859.552246       31.2

     [5 rows x 23 columns]
```

2.2 Identificação de valores nulos

```
[5]: db.isna().sum() # Verificar se existem valores desconhecidos na base de dados
```

```
[5]: station              2
     Present_Tmax        70
     Present_Tmin        70
     LDAPS_RHmin         75
     LDAPS_RHmax         75
     LDAPS_Tmax_lapse    75
     LDAPS_Tmin_lapse    75
     LDAPS_WS            75
     LDAPS_LH            75
     LDAPS_CC1           75
     LDAPS_CC2           75
     LDAPS_CC3           75
     LDAPS_CC4           75
     LDAPS_PPT1          75
     LDAPS_PPT2          75
     LDAPS_PPT3          75
     LDAPS_PPT4          75
     lat                  0
     lon                  0
     DEM                  0
     Slope                0
     Solar radiation      0
     Next_Tmax           27
     dtype: int64
```

2.2.1 Remoção dos valores nulos

[6]:
```
db = db.dropna()
db.shape
```

[6]:
```
(7588, 23)
```

2.2.2 Separação da base de dados (entrada e saída)

[7]:
```
X = db.drop(columns=['Next_Tmax'])
y = db['Next_Tmax']
```

2.2.3 Centering e Scaling dos dados de entrada

[8]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = scaler.fit_transform(X)
```

3 Processamento do conjunto de dados

3.1 Imports necessários

[9]:
```
from random import sample
from scipy.stats import loguniform, uniform
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.exceptions import ConvergenceWarning
from sklearn.linear_model import Lasso, LinearRegression, Ridge
from sklearn.model_selection import cross_val_score, GridSearchCV,␣
 ↪RandomizedSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from statistics import mean
```

[10]:
```
def fit_values(model, X, y):
  cvs = cross_val_score(model,
                        X,
                        y,
                        cv=5,
                        scoring='neg_root_mean_squared_error')
  return cvs

def find_hyperparams(model, space, X, y, n_iter):
  search = RandomizedSearchCV(model,
                        space,
                          n_iter = n_iter,
                          scoring = 'neg_root_mean_squared_error',
                          n_jobs = -1,
                          cv = 5,
                          random_state = 1)
  return search.fit(X,y)

def best_results(model, space, X, y, hparams_name, n_iter = 10):
  hparams = find_hyperparams(model, space, X, y, n_iter)

  best_hparams = hparams.best_params_
  best_rmse    = -hparams.best_score_

  string_result = f'Melhor RMSE: {best_rmse:5f} para'

  for x in hparams_name:
    string_result += f' {x}: {best_hparams[x]:5f} \t'


  print(string_result)

  model_default = fit_values(model, X, y)

  print(f'RMSE para modelo default: {mean(-model_default):5f}')

  return (best_hparams, best_rmse, model_default)
```

3.2 Modelo Linear

[11]:

```
linear_ = fit_values(LinearRegression(), X, y)

print('*' *10 + ' Modelo Linear ' + '*' *10)
print(f'Melhor RMSE: {mean(-linear_)}')
```

```
********** Modelo Linear **********
Melhor RMSE: 1.5775462124225403
```

3.3 Modelo Linear com regularização L2

[12]:

```
space = dict()
space['alpha'] = loguniform(10**-3, 10**3)


print('*' *10 + ' Modelo Ridge ' + '*' *10)
ridge_ = best_results(Ridge(),
                      space,
                      X,
                  y,
                  ['alpha'])
```

```
********** Modelo Ridge **********
Melhor RMSE: 1.576248 para alpha: 20.986836
RMSE para modelo default: 1.577477
```

3.4 Modelo Linear com regularização L1

[13]:

```
space = dict()
space['alpha'] = loguniform(10**-3, 10**3)


print('*' *10 + ' Modelo Lasso ' + '*' *10)
lasso_ = best_results(Lasso(),
                      space,
                      X,
                      y,
                      ['alpha'])
```

```
********** Modelo Lasso **********
Melhor RMSE: 1.570089 para alpha: 0.013109
RMSE para modelo default: 2.024356
```

3.5 Modelo SVM Linear

[14]:

```
space = dict()
space['epsilon'] = [0.1, 0.3]
space['C'] = loguniform(2**-5, 2**15)

print('*' *10 + ' Modelo SVM Linear ' + '*' *10)
svm_linear_ = best_results(SVR(kernel = 'linear', max_iter = 3000),
                           space,
                           X,
                           y,
                           ['epsilon', 'C'])
```

```
********** Modelo SVM Linear **********

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)

Melhor RMSE: 1.788912 para epsilon: 0.300000     C: 0.824608

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

```
RMSE para modelo default: 1.843715
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

3.6 Modelo SVM kernel RBF

[15]:

```
space = dict()
space['epsilon'] = [0.1, 0.3]
space['C'] = loguniform(2**-5, 2**15)
space['gamma'] = loguniform(2**-9, 2**3)

print('*' *10 + ' Modelo SVM Linear ' + '*' *10)
svm_rbf_ = best_results(SVR(kernel = 'rbf', max_iter = 3000),
                        space,
                        X,
                        y,
                        ['epsilon', 'C', 'gamma'])
```

```
********** Modelo SVM Linear **********
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

```
Melhor RMSE: 1.566807 para epsilon: 0.100000     C: 14.611758    gamma: 0.002453
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=3000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

```
RMSE para modelo default: 1.685369
```

3.7 KNN

[16]:

```
space = dict()
space['n_neighbors'] = list(range(1, 1001))

print('*' *10 + ' Modelo KNN ' + '*' *10)
svm_rbf_ = best_results(KNeighborsRegressor(),
                        space,
                        X,
                        y,
                        ['n_neighbors'])
```

```
********** Modelo KNN **********
Melhor RMSE: 1.845414 para n_neighbors: 38.000000
RMSE para modelo default: 1.927051
```

3.8 MLP

[17]:

```
space = dict()
space['hidden_layer_sizes'] = list(range(5, 21, 3))

print('*' *10 + ' Modelo MLP ' + '*' *10)
mlp_ = best_results(MLPRegressor(),
                    space,
                    X,
                    y,
                    ['hidden_layer_sizes'],
                    6)
```

```
********** Modelo MLP **********
```

```
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
```

```
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Melhor RMSE: 2.383525 para hidden_layer_sizes: 11.000000
```

```
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
RMSE para modelo default: 1.931995
```

```
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

[18]:

```
mlp_
```

[18]:

```
({'hidden_layer_sizes': 11},
 2.3835245841742254,
 array([-2.00656122, -1.79094716, -2.02333629, -1.97491476, -1.8642152 ]))
```

3.9 Árvore de Decisão

[19]:

```
space = dict()
space['ccp_alpha'] = uniform(0.0, 0.04)

print('*' *10 + ' Modelo de Árvore de Decisão ' + '*' *10)

model = DecisionTreeRegressor()
model.cost_complexity_pruning_path(X, y)
```

```
decision_tree_ = best_results(model,
                              space,
                              X,
                              y,
                              ['ccp_alpha'])
```

```
********** Modelo de Árvore de Decisão **********
Melhor RMSE: 1.861153 para ccp_alpha: 0.021553
RMSE para modelo default: 2.208033
```

3.10 Random Forest

[20]:

```
space = dict()
space['n_estimators'] = [10, 100, 1000]
space['max_features'] = [5, 10, 22]

print('*' *10 + ' Modelo Random Forest ' + '*' *10)

random_forest_ = best_results(RandomForestRegressor(),
                              space,
                              X,
                              y,
                              ['n_estimators', 'max_features'],
                              9)
```

```
********** Modelo Random Forest **********
```

```
/usr/local/lib/python3.7/dist-
packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

```
Melhor RMSE: 1.627018 para n_estimators: 1000.000000     max_features: 10.000000
RMSE para modelo default: 1.658443
```

3.11 GBM

[21]:

```
space = dict()
space['n_estimators'] = list(range(5, 101))
space['learning_rate'] = uniform(0.01, 0.3)
space['max_depth'] = [2,3]

print('*' *10 + ' Modelo GBM ' + '*' *10)
gbm_ = best_results(GradientBoostingRegressor(),
                              space,
                              X,
                              y,
                              ['n_estimators', 'learning_rate', 'max_depth'])
```

```
********** Modelo GBM **********
Melhor RMSE: 1.585280 para n_estimators: 77.000000       learning_rate: 0.135107
max_depth: 2.000000
RMSE para modelo default: 1.596553
```

3.12 Tabela Comparativa

Modelo RMSE	Default	Melhores Hyperparâmetros
Linear	1.577546	-
Linear com L1	2.024356	1.570089
Linear com L2	1.577477	1.576248
SVM Linear	1.843715	1.788912
SVM com brf	1.685369	1.566807
KNN	1.927051	1.845414
MLP	1.931995	2.383525
Árvore de Decisão	2.208033	1.861153
Random Forest	1.658443	1.627018
GBM	1.596553	1.585280