MO432 - Aprendizado Supervisionado

Exercício 2

- **Augusto Cesar | 265679**
- **Giulia Fazzi | 225270**
- **Lucas Peres | 265193**

Pré-processamento

In [18]:

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
```

In [39]:

```
data = pd.read_csv("Bias_correction_ucl.csv")
print(data.head())
```

```
   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]
```

In [40]:

```
data.drop(columns=["Next_Tmin", "Date"], inplace=True)
print(data.head())
```

```
   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]
```

In [5]:

```
# default: remover linha se houver pelo menos um NA
data.dropna(inplace=True)
print(f"No. linhas = {data.shape[0]}") # precisa ser =7588
```

```
No. linhas = 7588
```

In [6]:

```
# SAÍDA
y = data[["Next_Tmax"]]

# ENTRADA
X = data.drop(columns=["Next_Tmax"])
X_scaled = StandardScaler().fit_transform(X)

# só pra verificar
# pd.DataFrame(X_scaled, columns=X.columns)
```

Regressores

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from warnings import filterwarnings
import random
filterwarnings("ignore")

# instanciar dicionário para armazenar os RMSE dos modelos
rmse_dict = {}
```

Regressão Linear

In [21]:

```
rmse_dict["Regressão Linear"] = {}
```

In [22]:

```
cv = cross_validate(LinearRegression(), X_scaled, y.values.ravel(),
                    cv=5, scoring=["neg_root_mean_squared_error"])
scores = cv["test_neg_root_mean_squared_error"]
final_score = -scores.mean()

rmse_dict["Regressão Linear"]["Modelo com os parâmetros default"] = final_score

print(f"RMSEs = {scores}")
print(f"Média RMSE = {final_score}")
```

```
RMSEs = [-1.45728827 -1.63874781 -1.45413013 -1.62441096 -1.7131539 ]
Média RMSE = 1.5775462124225403
```

Regressão Linear com Regularização L2 (*Ridge*)

In [23]:

```
rmse_dict["Regressão Linear (Ridge)"] = {}
```

Com busca aleatória de hiperparâmetros

In [24]:

```
alpha_dist = np.logspace(start=-3, stop=3, num=10, base=10)
ridge_regressor = RandomizedSearchCV(Ridge(),
                                     param_distributions={"alpha": alpha_dist},
                                     n_iter=10,
                                     scoring="neg_root_mean_squared_error",
                                     cv=5)
ridge_regressor.fit(X_scaled, y.values.ravel())
```

```
final_score = -ridge_regressor.best_score_
final_params = ridge_regressor.best_params_

rmse_dict["Regressão Linear (Ridge)"][f"Modelo com os parâmetros {final_params}"] = final
_score

print(f"alpha (ótimo) = {final_params['alpha']}")
print(f"Média RMSE = {final_score}")
```

```
alpha (ótimo) = 215.44346900318823
Média RMSE = 1.5720609550493712
```

Com hiperparâmetros default

In [25]:

```
cv = cross_validate(Ridge(), X_scaled, y.values.ravel(),
                    cv=5, scoring=["neg_root_mean_squared_error"])
scores = cv["test_neg_root_mean_squared_error"]
final_scores = -scores.mean()

rmse_dict["Regressão Linear (Ridge)"]["Modelo com os parâmetros default"] = final_score

print(f"RMSEs = {scores}")
print(f"Média RMSE = {final_scores}")
```

```
RMSEs = [-1.4570365  -1.63894403 -1.4541832  -1.62418292 -1.71303988]
Média RMSE = 1.5774773058961011
```

Regressão Linear com Regularização L1 (*Lasso*)

In [26]:

```
rmse_dict["Regressão Linear (Lasso)"] = {}
```

Com busca aleatória de hiperparâmetros

In [27]:

```
alpha_dist = np.logspace(start=-3, stop=3, num=10, base=10)
lasso_regressor = RandomizedSearchCV(Lasso(),
                                     param_distributions={"alpha": alpha_dist},
                                     n_iter=10,
                                     scoring="neg_root_mean_squared_error",
                                     cv=5)
lasso_regressor.fit(X_scaled, y.values.ravel())
final_score = -lasso_regressor.best_score_
final_params = lasso_regressor.best_params_

rmse_dict["Regressão Linear (Lasso)"][f"Modelo com os parâmetros {final_params}"] = final
_score

print(f"alpha (ótimo) = {final_params['alpha']}")
print(f"Média RMSE = {final_score}")
```

```
alpha (ótimo) = 0.021544346900318832
Média RMSE = 1.5682906755319626
```

Com hiperparâmetros default

In [28]:

```
cv = cross_validate(Lasso(), X_scaled, y.values.ravel(),
                    cv=5, scoring=["neg_root_mean_squared_error"])
scores = cv["test_neg_root_mean_squared_error"]
final_score = -scores.mean()
```

```
rmse_dict["Regressão Linear (Lasso)"]["Modelo com os parâmetros default"] = final_score

print(f"RMSEs = {scores}")
print(f"Média RMSE = {final_score}")
```

```
RMSEs = [-1.82291874 -1.95382965 -1.7360366  -2.54635882 -2.0626362 ]
Média RMSE = 2.0243560014360105
```

SVM Linear

In [29]:

```
rmse_dict["SVM Linear"] = {}
```

Com busca aleatória de hiperparâmetros

Selecione 10 pares aleatórios ente:

- **Use epsilon =** 0.1 **ou** 0.3
- **Use C entre** 2^{-5} **e** 2^{15} **uniforme no expoente**

In [30]:

```
# n_iter = 10   para definir 10 pares de configurações
# cv = 5        para usar 5-fold cross validation
params = {"epsilon": [0.1, 0.3],
          "C": uniform(loc=2**(-5), scale=2**15)}
svr_regressor = RandomizedSearchCV(LinearSVR(),
                                   param_distributions=params,
                                   n_iter=10,
                                   cv=5,
                                   scoring = "neg_root_mean_squared_error",
                                   verbose=0,
                                   n_jobs = -1)
svr_regressor.fit(X_scaled, y.values.ravel())

final_params = svr_regressor.best_params_
final_score = -svr_regressor.best_score_

rmse_dict["SVM Linear"][f"Modelo com os parâmetros {final_params}"] = final_score

print(f"epsilon (ótimo) = {final_params['epsilon']}\nC (ótimo) = {final_params['C']}")
print(f"Média RMSE = {final_score}")
```

```
epsilon (ótimo) = 0.3
C (ótimo) = 20116.95370828616
Média RMSE = 1.994177658148694
```

Com hiperparâmetros default

In [31]:

```
cv = cross_validate(LinearSVR(), X_scaled, y.values.ravel(),
                    cv=5, scoring=["neg_root_mean_squared_error"])
scores = cv["test_neg_root_mean_squared_error"]
final_score = -scores.mean()

rmse_dict["SVM Linear"]["Modelo com os parâmetros default"] = final_score

print(f"RMSEs = {scores}")
print(f"Média RMSE = {final_score}")
```

```
RMSEs = [-1.50341705 -1.62690569 -1.42638171 -1.55527284 -1.6804331 ]
Média RMSE = 1.5584820774391304
```

SVM com kernel RBF

In [32]:

```
rmse_dict["SVM com kernel RBF"] = {}
```

Com busca aleatória de hiperparâmetros

Selecione 10 trincas aleatórias entre:

- **Use epsilon =** 0.1 **ou** 0.3
- **Use C entre** 2^{-5} **e** 2^{15} **uniforme no expoente**
- **Use gamma entre** 2^{-9} **e** 2^{3} **uniforme no expoente**

In [33]:

```
params = {"epsilon": [0.1, 0.3],
          "C": uniform(loc=2**(-5), scale=2**15),
          "gamma": uniform(loc=2**(-9), scale=2**3)}
# n_iter = 10   para definir 10 trincas de configurações
# cv = 5        para usar 5-fold cross validation
svm_rbf_regressor = RandomizedSearchCV(SVR(kernel='rbf', max_iter=5000),
                                       param_distributions=params,
                                       n_iter=10,
                                       cv=5,
                                       scoring='neg_root_mean_squared_error',
                                       verbose=0,
                                       n_jobs=-1)
svm_rbf_regressor.fit(X_scaled, y.values.ravel())

final_params = svm_rbf_regressor.best_params_
final_score = -svm_rbf_regressor.best_score_

rmse_dict["SVM com kernel RBF"][f"Modelo com os parâmetros {final_params}"] = final_score


print(f"epsilon (ótimo) = {final_params['epsilon']}\n",
      f"C (ótimo) = {final_params['C']}\n",
      f"gamma (ótimo) = {final_params['gamma']}")
print(f'Média RMSE = {final_score}')
```

```
epsilon (ótimo) = 0.3
C (ótimo) = 28758.630821570972
gamma (ótimo) = 0.0748710290684631
Média RMSE = 1.902051873372271
```

Com hiperparâmetros default

In [34]:

```
cv = cross_validate(SVR(kernel='rbf'), X_scaled, y.values.ravel(),
                    cv=5, scoring=['neg_root_mean_squared_error'])
scores = cv['test_neg_root_mean_squared_error']
final_score = -scores.mean()

rmse_dict["SVM com kernel RBF"][f"Modelo com os parâmetros default"] = final_score

print(f"RMSEs = {scores}")
print(f'Média RMSE = {final_score}')
```

```
RMSEs = [-1.5600696  -1.79768516 -1.5021843  -1.7832602  -1.79677649]
Média RMSE = 1.6879951487537077
```

KNN

In [35]:

In [35]:

```
rmse_dict["KNN"] = {}
```

Com busca aleatória de hiperparâmetros

Selecione 10 K aleatórios entre 1 e 1000

In [36]:

```
model = KNeighborsRegressor()
params = {"n_neighbors": range(1, 1000)}

# n_iter = 10   para definir 10 configurações
# cv = 5        para usar 5-fold cross validation
rand_search = RandomizedSearchCV(model, params, n_iter=10, cv=5,
                                 scoring = 'neg_root_mean_squared_error',
                                 verbose=0, n_jobs = -1)
rand_search.fit(X_scaled, y.values.ravel())

params_final = rand_search.best_params_
score_final = rand_search.best_score_

rmse_dict["KNN"][f"Modelo com os parâmetros {params_final}"] = score_final

print(f'Melhor RMSE = {-score_final:4f}')
```

```
Melhor RMSE = 1.951951
```

Com hiperparâmetros default

In [37]:

```
model = KNeighborsRegressor()
cv = cross_validate(model, X_scaled, y.values.ravel(), cv=5,
                    scoring=['neg_root_mean_squared_error'])
scores = cv['test_neg_root_mean_squared_error']

score_final = -scores.mean()

rmse_dict["KNN"][f"Modelo com os parâmetros default"] = score_final

print(f'Média RMSE = {score_final:4f}')
```

```
Média RMSE = 1.927051
```

MLP

Com hiperparâmetros default

In [8]:

```
rmse_dict["MPL"] = {}

model = MLPRegressor()
cv = cross_validate(model, X_scaled, y.values.ravel(), cv=5,
                    scoring=['neg_root_mean_squared_error'])
scores = cv['test_neg_root_mean_squared_error']

score_final = -scores.mean()

rmse_dict["MPL"][f"Modelo com os parâmetros default"] = score_final

print(f'Média RMSE = {score_final:4f}')
```

```
Média RMSE = 2.092381
```

Com busca aleatória de hiperparâmetros

hidden_layer_sizes: 5 a 20, de três em três

In [9]:

```
params = {"hidden_layer_sizes": range(5, 20, 3)}

MPL_regrssor = RandomizedSearchCV(MLPRegressor(), params, verbose=0)
MPL_regrssor.fit(X_scaled, y.values.ravel())

final_params = MPL_regrssor.best_params_
final_score = -MPL_regrssor.best_score_

rmse_dict["MPL"][f"Modelo com os parâmetros {final_params}"] = final_score

print(f"hidden_layer_sizes (ótimo) = {final_params['hidden_layer_sizes']}")
print(f"Média RMSE = {final_score}")
```

```
hidden_layer_sizes (ótimo) = 8
Média RMSE = -0.3347351013522
```

Árvore de Decisão

Com hiperparâmetros default

In [10]:

```
rmse_dict["Árvore de Decisão"] = {}

model = DecisionTreeRegressor()
cv = cross_validate(model, X_scaled, y.values.ravel(), cv=5,
                    scoring=['neg_root_mean_squared_error'])
scores = cv['test_neg_root_mean_squared_error']

score_final = -scores.mean()

rmse_dict["Árvore de Decisão"][f"Modelo com os parâmetros default"] = score_final

print(f'Média RMSE = {score_final:4f}')
```

```
Média RMSE = 2.187725
```

Com busca aleatória de hiperparâmetros

Usar prunning.

ccp_alpha: 10 números aleatórios entre 0.0 e 0.04

In [11]:

```
params = {"ccp_alpha": np.random.uniform(0.0, 0.04, 10)}

dt_regressor = RandomizedSearchCV(DecisionTreeRegressor(), params, verbose=0)

dt_regressor.fit(X_scaled, y.values.ravel())

final_params = dt_regressor.best_params_
final_score = -dt_regressor.best_score_

rmse_dict["Árvore de Decisão"][f"Modelo com os parâmetros {final_params}"] = final_score

print(f"ccp_alpha (ótimo) = {final_params['ccp_alpha']}")
print(f"Média RMSE = {final_score}")
```

```
ccp_alpha (ótimo) = 0.03780819524865303
Média RMSE = -0.617071022985362
```

Random Forest

Com hiperparâmetros default

In [12]:

```
from sklearn.ensemble import RandomForestRegressor

rmse_dict["Random Forest"] = {}

model = RandomForestRegressor()
cv = cross_validate(model, X_scaled, y.values.ravel(), cv=5, scoring=['neg_root_mean_squ
ared_error'])
scores = cv['test_neg_root_mean_squared_error']

score_final = -scores.mean()

rmse_dict["Random Forest"][f"Modelo com os parâmetros default"] = score_final

print(f'Média RMSE = {score_final:4f}')
```

```
Média RMSE = 1.658784
```

Com busca aleatória de hiperparâmetros

Usar todas as combinações dos valores abaixo.

- **n_estimators:** 10, 100 e 1000
- **max_features:** 5, 10, e 22

In [13]:

```
params = {"n_estimators": [10,100,1000],
          "max_features":  [5,10,22]}

dt_regressor = RandomizedSearchCV(RandomForestRegressor(), params,
                                  n_jobs=10)
dt_regressor.fit(X_scaled, y.values.ravel())

final_params = dt_regressor.best_params_
final_score = -dt_regressor.best_score_

rmse_dict["Random Forest"][f"Modelo com os parâmetros {final_params}"] = final_score

print(f"n_estimators (ótimo) = {final_params['n_estimators']}",
      f"\n max_features (ótimo) = {final_params['max_features']}")
print(f"Média RMSE = {final_score}")
```

```
n_estimators (ótimo) = 1000
 max_features (ótimo) = 10
Média RMSE = -0.7033300804480529
```

GBM

Com hiperparâmetros default

In [15]:

```
from sklearn.ensemble import GradientBoostingRegressor

rmse_dict["GBM"] = {}

model = GradientBoostingRegressor()
cv = cross_validate(model, X_scaled, y.values.ravel(),
```

```
                      cv=5, scoring=['neg_root_mean_squared_error'])
scores = cv['test_neg_root_mean_squared_error']

score_final = -scores.mean()

rmse_dict["GBM"][f"Modelo com os parâmetros default"] = score_final

print(f'Média RMSE = {score_final:4f}')
```

```
Média RMSE = 1.596377
```

Com busca aleatória de hiperparâmetros

Selecione 10 trinca aleatórias ente:

- **n_estimators:** 5 a 100
- **learning_rate:** 0.01 a 0.3
- **max_depth:** 2 ou 3

In [16]:

```
params = {"n_estimators": np.random.randint(5, 100, 10),
          "max_features":  np.random.uniform(0.0, 0.3, 10),
          "max_depth": [2, 3]}

dt_regressor = RandomizedSearchCV(GradientBoostingRegressor(), params, verbose=0)

dt_regressor.fit(X_scaled, y.values.ravel())

final_params = dt_regressor.best_params_
final_score = -dt_regressor.best_score_

rmse_dict["GBM"][f"Modelo com os parâmetros {final_params}"] = final_score

print(f"n_estimators (ótimo) = {final_params['n_estimators']}",
      f"\n max_features (ótimo) = {final_params['max_features']}",
      f"\n max_depth (ótimo) = {final_params['max_depth']}")
print(f"Média RMSE = {final_score}")
```

```
n_estimators (ótimo) = 67
 max_features (ótimo) = 0.19740513905356852
 max_depth (ótimo) = 2
Média RMSE = -0.697144197533261
```

Resultados

In [38]:

```
for key in rmse_dict:
  print(f"{key}:")

  for i in rmse_dict[key]:
    print(f"    {i}:\n      RMSE = {rmse_dict[key][i]:4f}")

  print("\n")
```

```
MPL:
    Modelo com os parâmetros default:
      RMSE = 2.092381
    Modelo com os parâmetros {'hidden_layer_sizes': 8}:
      RMSE = -0.334735


Árvore de Decisão:
    Modelo com os parâmetros default:
      RMSE = 2.187725
    Modelo com os parâmetros {'ccp_alpha': 0.03780819524865303}:
      RMSE = -0.617071
```

```
Random Forest:
    Modelo com os parâmetros default:
      RMSE = 1.658784
    Modelo com os parâmetros {'n_estimators': 1000, 'max_features': 10}:
      RMSE = -0.703330


GBM:
    Modelo com os parâmetros default:
      RMSE = 1.596377
    Modelo com os parâmetros {'n_estimators': 67, 'max_features': 0.19740513905356852, 'm
ax_depth': 2}:
      RMSE = -0.697144


Regressão Linear:
    Modelo com os parâmetros default:
      RMSE = 1.577546


Regressão Linear (Ridge):
    Modelo com os parâmetros {'alpha': 215.44346900318823}:
      RMSE = 1.572061
    Modelo com os parâmetros default:
      RMSE = 1.572061


Regressão Linear (Lasso):
    Modelo com os parâmetros {'alpha': 0.021544346900318832}:
      RMSE = 1.568291
    Modelo com os parâmetros default:
      RMSE = 2.024356


SVM Linear:
    Modelo com os parâmetros {'C': 20116.95370828616, 'epsilon': 0.3}:
      RMSE = 1.994178
    Modelo com os parâmetros default:
      RMSE = 1.558482


SVM com kernel RBF:
    Modelo com os parâmetros {'C': 28758.630821570972, 'epsilon': 0.3, 'gamma': 0.0748710
290684631}:
      RMSE = 1.902052
    Modelo com os parâmetros default:
      RMSE = 1.687995


KNN:
    Modelo com os parâmetros {'n_neighbors': 365}:
      RMSE = -1.951951
    Modelo com os parâmetros default:
      RMSE = 1.927051
```