Tarefa 04 - MO431

Patrick de Carvalho Tavares Rezende Ferreira - 175480

In [1]:

```
import cma
import hyperopt
import numpy as np
from hyperopt import fmin, tpe, hp

from pyswarm import pso
from scipy.optimize import dual_annealing
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV, ShuffleSplit
from sklearn.svm import SVR

%matplotlib inline
```

As funções abaixo são auxiliares para avaliar o erro "mean absolute error" (MAE) em cada algoritmo que requer a chaamda de uma função para tal, obedecendo às convenções impostas por cada um para passagem dos parâmetros.

In [2]:

```
# Retorna o MAE pra o SVR
def evaluate_svr(args):
    gamma, C, epsilon = args

    gamma = 2 ** gamma
    C = 2 ** C

    svr_object = SVR(kernel='rbf', gamma=gamma, C=C, epsilon=epsilon)
    svr_object.fit(x_treino, y_treino)
    return mean_absolute_error(svr_object.predict(x_teste), y_teste)


# Retorna o MAE tambem, mas recebendo argumentos como array
def erro(x):
    svr_object = SVR(kernel='rbf', gamma=2 ** x[0], C=2 ** x[1], epsilon=x[2])
    svr_object.fit(x_treino, y_treino)
    return mean_absolute_error(svr_object.predict(x_teste), y_teste)


# Como hp nao possui distribuicoes uniformes nos expoentes de base dois, fazemos
# este processo na funcao de avaliacao
def evaluate_svr_hp(args):
    gamma, C, epsilon = args

    gamma = 2 ** gamma
    C = 2 ** C

    svr_object = SVR(kernel='rbf', gamma=gamma, C=C, epsilon=epsilon)
    svr_object.fit(x_treino, y_treino)
    return mean_absolute_error(svr_object.predict(x_teste), y_teste)
```

Na célula abaixo, fazemos o carregamento dos dados a serem utilizados neste roteiro.

In [3]:

```
# Dados de treino
x_treino = np.load("Xtreino5.npy")
y_treino = np.load("ytreino5.npy")

# Dados de teste
x_teste = np.load("Xteste5.npy")
y_teste = np.load("yteste5.npy")
```

Random Search

O primiro algoritmo implmentado é o random search, que amostra aleatoriamente 125 valores para os hiperparâmetros C, gamma e epsilon dentre os valores no intervalo passado para procurar a melhor combinação.

Os ranges solicitados no roteiro são:

- C entre 2^{-5} e 2^{15} (uniforme nos expoentes);
- Gamma entre 2^{-15} e 2^{3} (uniforme nos expoentes);
- Epsilon entre 0.05 a 1.0 (uniforme neste intervalo).

O melhor conjunto encontrado e o MAE obtido são exibidos abaixo.

In [4]:

```
# Fixando a semente para garantir resultados aleatorios reprodutiveis.
np.random.seed(1234)

# Gera os parametros de entrada aleatoriamente. Alguns sao uniformes nos
# EXPOENTES.
c = 2 ** np.random.uniform(-5, 15, 125)
gamma = 2 ** np.random.uniform(-15, 3, 125)
epsilon = np.random.uniform(0.05, 1.0, 125)

# Une os parametros de entrada em um unico dicionario a ser passado para a
# funcao.
parametros = {'C': c, 'gamma': gamma, 'epsilon': epsilon}

svr_object = SVR()
# Montamos o objeto que realiza a busca
randomized_search_engine = \
    RandomizedSearchCV(
        estimator=svr_object,
        param_distributions=parametros,
        scoring="neg_mean_absolute_error",
        cv=ShuffleSplit(n_splits=1)
    )

# Realizamos a busca atraves do treinamento
randomized_search_engine.fit(x_treino, y_treino)

# Predizemos os valores de teste para avaliar o resultado.
mae = mean_absolute_error(randomized_search_engine.predict(x_teste), y_teste)

print("\n--------------------RANDOM_SEARCH_CV--------------------")

print("\nMelhor conjunto de parâmetros: \n", randomized_search_engine.best_estimato

print("\nMAE teste: \n", mae)
```

```
--------------------RANDOM_SEARCH_CV--------------------

Melhor conjunto de parâmetros:
 SVR(C=11563.99742567888, cache_size=200, coef0=0.0, degree=3,
    epsilon=0.4249541542729139, gamma=7.579089030110616e-05, kernel='r
bf',
    max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE teste:
 2.5777863091558473
```

Grid Search

Agora implementamos o grid search, que utiliza 125 combinações de hiperparâmetros dentre as 5 opções de cada passadas como intervalo, de acordo com o solicitado no roteiro.

Nota-se que o MAE obtido por este método é significativamente maior que o do random search, conforme comentado em aula. Talvez isto se deva a uma maior possibilidade de combinações para o random search, enquanto que o grid fica limitado às combinações predefinidas.

In [5]:

```
# Fixando a semente para garantir resultados aleatorios reprodutiveis.
np.random.seed(1234)

# Gera os parametros de entrada aleatoriamente. Alguns sao uniformes nos
# EXPOENTES.
c = 2 ** np.random.uniform(-5, 15, 5)
gamma = 2 ** np.random.uniform(-15, 3, 5)
epsilon = np.random.uniform(0.05, 1.0, 5)

# Une os parametros de entrada em um unico dicionario a ser passado para a
# funcao.
parametros = {'C': c, 'gamma': gamma, 'epsilon': epsilon}

svr_object = SVR()
# Montamos o objeto que realiza a busca
randomized_search_engine = \
    GridSearchCV(
        estimator=svr_object,
        param_grid=parametros,
        scoring="neg_mean_absolute_error",
        cv=ShuffleSplit(n_splits=1)
    )

# Realizamos a busca atraves do treinamento
randomized_search_engine.fit(x_treino, y_treino)

# Predizemos os valores de teste para avaliar o resultado.
mae = mean_absolute_error(randomized_search_engine.predict(x_teste), y_teste)

print("\n--------------------GRID_SEARCH_CV--------------------")

print("\nMelhor conjunto de parâmetros: \n", randomized_search_engine.best_estimato

print("\nMAE teste: \n", mae)
```

```
--------------------GRID_SEARCH_CV--------------------

Melhor conjunto de parâmetros:
 SVR(C=0.4445411902913089, cache_size=200, coef0=0.0, degree=3,
    epsilon=0.5259453692472857, gamma=0.0009153853954791652, kernel='r
bf',
    max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE teste:
 4.705358475374269
```

Otimização Bayesiana

Abaixo, a otimização bayesiana encontrou hiperparâmetros com valores diferentes dos anteriores e foi mais demorada também, obtendo porém um valor de MAE mais baixo que os demais.

In [6]:

```
# Fixando a semente para garantir resultados aleatorios reprodutiveis.
np.random.seed(1234)

# Definindo o espaco em que iremos trabalhar. Gamma e C serao elevados a 2
# na funcao de avaliacao do SVR
space = [hp.uniform('gamma', -15, 3),
         hp.uniform('C', -5, 15),
         hp.uniform('epsilon', 0.05, 1.0)]

# calling the hyperopt function
resultado = fmin(fn=evaluate_svr_hp, space=space, algo=tpe.suggest, max_evals=125)

print("\n--------------------OTIMIZAÇÃO-BAYESIANA-hyperopt--------------------")

print("\nMelhor conjunto de parâmetros: \n")
print("C: ", 2 ** resultado["C"])

print("gamma: ", 2 ** resultado["gamma"])

print("epsilon: ", resultado["epsilon"])

# Calculando o modelo encontrado para verificar seu erro mean_absolute_error
svr_object = SVR(kernel='rbf', gamma=2 ** resultado["gamma"], C=2 ** resultado["C"]
svr_object.fit(x_treino, y_treino)
mae = mean_absolute_error(svr_object.predict(x_teste), y_teste)

print("\nMAE teste: \n", mae)
```

```
100%|██████████| 125/125 [00:34<00:00,  3.67trial/s, best loss: 2.3462
210499902336]

--------------------OTIMIZAÇÃO-BAYESIANA-hyperopt-------------------
-

Melhor conjunto de parâmetros:

C:  5867.221372624267
gamma:  4.486331407697145e-05
epsilon:  0.095735808464466

MAE teste:
 2.3462210499902336
```

PSO

Utilizando a biblioteca pyswarm, implementamos o PSO abaixo e, embora seu MAE tenha sido maior que os anteriores, ele foi bem mais rápido e utilizou apenas 11 partículas e 11 iterações. O erro MAE decresceu rapidamente quando foram utilizadas mais partículas e mais iterações, porém com tempo de execução aumentando rapidamente conforme a quantidade de iterações e partículas.

In [7]:

```
# Fixando a semente para garantir resultados aleatorios reprodutiveis.
np.random.seed(1234)

lb = [-15, -5, 0.05]
ub = [3, 15, 1.0]

xopt, fopt = pso(erro, lb, ub, maxiter=11, swarmsize=11)

print("\n----------------------PSO----------------------")

# Calculando o modelo encontrado para verificar seu erro mean_absolute_error
svr_object = SVR(kernel='rbf', gamma=2 ** xopt[0], C=2 ** xopt[1], epsilon=xopt[2])
svr_object.fit(x_treino, y_treino)
mae = mean_absolute_error(svr_object.predict(x_teste), y_teste)

print("\nMAE teste: \n", mae)
print("C: ", 2 ** xopt[1])
print("epsilon: ", xopt[2])
print("gamma: ", 2 ** xopt[0])
```

```
Stopping search: maximum iterations reached --> 11

----------------------PSO----------------------

MAE teste:
 2.318512680271687
C:  9699.08244654229
epsilon:  0.29484562321646
gamma:  3.665232614765423e-05
```

Simulated Annealing

Abaixo implementamos o simulated annealing através do pacote scipy.optimize.dual-annealing, que implementa o tradicional simulated annealing quando "no_local_search=True". A escolha deste pacote é devido à documentação mais familiar.

O MAE obtido foi próximo do PSO e o tempo de execução foi de mesma ordem.

In [8]:

```
# Fixando a semente para garantir resultados aleatorios reprodutiveis.
np.random.seed(1234)

# Valores limite para a busca em cada um dos parametros
lw = [-15, -5, 0.05]
up = [3, 15, 1.0]

resultado = dual_annealing(evaluate_svr, bounds=list(zip(lw, up)), no_local_search=

print("\n--------------------SIMULATED-ANNEALING--------------------")

# Calculando o modelo encontrado para verificar seu erro mean_absolute_error
svr_object = SVR(kernel='rbf', gamma=2 ** resultado.x[0], C=2 ** resultado.x[1], ep
svr_object.fit(x_treino, y_treino)
mae = mean_absolute_error(svr_object.predict(x_teste), y_teste)

print("\nMAE teste: \n", mae)

print("C: ", 2 ** resultado.x[1])

print("gamma: ", 2 ** resultado.x[0])

print("epsilon: ", resultado.x[2])
```

```
--------------------SIMULATED-ANNEALING--------------------

MAE teste:
 2.3652112209709824
C:  4896.33391785218
gamma:  4.7463406469492094e-05
epsilon:  0.12451007990924375
```

CMA-ES

Por último, fazemos a implementação do CMA-ES, que obtém erro MAE próximo ao dobro do simulated-annealing e tempo de execução parecido.

In [9]:

```
# Fixando a semente para garantir resultados aleatorios reprodutiveis.
np.random.seed(1234)

opts = cma.CMAOptions()
opts.set("bounds", [[-15, -5, 0.05], [3, 15, 1.0]])
opts.set('maxfevals', 125)
parametros, es = cma.fmin2(evaluate_svr, [1, 1, 1], 0.1, opts)

# Calculando o modelo encontrado para verificar seu erro mean_absolute_error
svr_object = SVR(kernel='rbf', gamma=2 ** parametros[0], C=2 ** parametros[1], epsi
svr_object.fit(x_treino, y_treino)
mae = mean_absolute_error(svr_object.predict(x_teste), y_teste)

print("\nMAE teste: \n", mae)

print("C: ", 2 ** parametros[1])

print("gamma: ", 2 ** parametros[0])

print("epsilon: ", parametros[2])
```

```
(3_w,7)-aCMA-ES (mu_w=2.3,w_1=58%) in dimension 3 (seed=286190, Sun Ma
y 10 13:05:32 2020)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:
s]
    1      7 5.717387558832545e+00 1.0e+00 8.55e-02  7e-02  9e-02 0:0
0.1
    2     14 5.717166328903278e+00 1.5e+00 1.10e-01  1e-01  1e-01 0:0
0.1
    3     21 5.716953700733195e+00 1.6e+00 1.36e-01  1e-01  2e-01 0:0
0.2
   18    126 5.568481736477731e+00 5.4e+00 7.78e-01  3e-01  2e+00 0:0
1.0
termination on maxfevals=125 (Sun May 10 13:05:33 2020)
final/bestever f-value = 5.589905e+00 5.568482e+00
incumbent solution: [-3.7208355892031477, 0.0003926589019261906, 0.999
8367455494805]
std deviation: [1.5406137083084956, 0.6094208994881579, 0.274402780473
6519]

MAE teste:
 5.5684817364777315
C:  0.9461350310928665
gamma:  0.06449855391549174
epsilon:  0.9797012373039787
```