Tarefa#4

May 3, 2021

Unicamp - MO431 - Álgebra linear e otimização para aprendizado de maquina - Jacques Wainer

Felipe Marinho Tavares

RA: 265680

Tarefa #4

---

[26]:
```
import numpy as np
import matplotlib.pyplot as plt
import time
import pandas as pd

from sklearn.svm import SVR
#from sklearn.model_selection import KFold
#from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
#from sklearn.metrics import mean_squared_error
from skopt.space import Real
from skopt import BayesSearchCV
import pyswarm as ps
from sklearn.model_selection import cross_val_score

verbose=False

seed=1082141
np.random.seed(seed)
```

[2]:
```
X = np.load("tarefa#4_dados/X.npy")
print(f"X shape: {X.shape}")

y = np.load("tarefa#4_dados/y.npy")
print(f"y shape: {y.shape}")
```

```
X shape: (506, 13)
y shape: (506,)
```

0.1 1) Random search

[4]:
```
n_folds = 5
n_iter = 125
param_range=5
print(f"Random search com n_iter={n_iter}\n")

# sum parameter ranges for rbf kernel
C_range = np.logspace(-5, 15, param_range, base=2)
gamma_range = np.logspace (-15, 3, param_range, base=2)
epsilon = np.linspace(0.05, 1.0, param_range)
param_grid = dict(C=C_range, gamma=gamma_range, epsilon=epsilon)
if verbose:
    print(f"param_grid:\n{param_grid}\n")

# regr definition and fit
regr = SVR(kernel='rbf')
clf = RandomizedSearchCV(regr, param_grid, cv=n_folds, n_iter=n_iter,
                         random_state=seed, scoring='neg_mean_squared_error',
                         n_jobs=-1)

print("Params:")
print(clf.get_params())
print("")

s_time = time.time()
clf.fit(X, y)
e_time = time.time()

print(f"Tempo total de execução: \n{e_time - s_time} seconds\n")
```

```
Random search com n_iter=125
{'cv': 5, 'error_score': nan, 'estimator__C': 1.0, 'estimator__cache_size': 200,
'estimator__coef0': 0.0, 'estimator__degree': 3, 'estimator__epsilon': 0.1,
'estimator__gamma': 'scale', 'estimator__kernel': 'rbf', 'estimator__max_iter':
-1, 'estimator__shrinking': True, 'estimator__tol': 0.001, 'estimator__verbose':
False, 'estimator': SVR(), 'iid': 'deprecated', 'n_iter': 125, 'n_jobs': -1,
'param_distributions': {'C': array([3.1250e-02, 1.0000e+00, 3.2000e+01,
1.0240e+03, 3.2768e+04]), 'gamma': array([3.05175781e-05, 6.90533966e-04,
1.56250000e-02, 3.53553391e-01,
       8.00000000e+00]), 'epsilon': array([0.05  , 0.2875, 0.525 , 0.7625, 1.
])}, 'pre_dispatch': '2*n_jobs', 'random_state': 1082141, 'refit': True,
'return_train_score': False, 'scoring': 'neg_mean_squared_error', 'verbose': 0}
Tempo total de execução:
16.327448844909668 seconds
```

[5]:
```
df = pd.DataFrame(clf.cv_results_)
cols = ['rank_test_score', 'mean_test_score', 'std_test_score',
        'param_C', 'param_gamma', 'param_epsilon']
n_top = 5

# Coluna mean_test_score com valores negativos por conta da padronização de␣
 ↪RandomizedSearchCv
df['mean_test_score'] = df['mean_test_score']*-1

print(f'Top {n_top} melhores valores para os hiperparametros e o RMSE para␣
 ↪esses valores (coluna mean_test_score):')

display(df.sort_values('rank_test_score')[:n_top][cols].
 ↪set_index('rank_test_score'))
```

```
Top 5 melhores valores para os hiperparametros e o RMSE para esses valores
(coluna mean_test_score):
```

rank_test_score	mean_test_score	std_test_score	param_C	param_gamma	param_epsilon
1	14.812549	5.866103	32768.0	0.000031	0.2875
2	14.850936	6.118843	32768.0	0.000031	0.05
3	15.349063	5.557397	32768.0	0.000031	0.525
4	15.521831	5.333289	32768.0	0.000031	0.7625
5	15.542174	4.902948	32768.0	0.000031	1.0

0.2 2) Grid search

[6]:
```
n_folds = 5
n_iter = 125
param_range=5
print(f"Grid search para parametros [C, gamma, epsilon] em grid 5x5x5\n")

# sum parameter ranges for rbf kernel
C_range = np.logspace(-5, 15, param_range, base=2)
gamma_range = np.logspace (-15, 3, param_range, base=2)
epsilon = np.linspace(0.05, 1.0, param_range)
param_grid = dict(C=C_range, gamma=gamma_range, epsilon=epsilon)
if verbose:
```

```
    print(f"param_grid:\n{param_grid}\n")

# regr definition and fit
regr = SVR(kernel='rbf')
clf = GridSearchCV(regr, param_grid, cv=n_folds,
                   scoring='neg_mean_squared_error', n_jobs=-1)

print("Params:")
print(clf.get_params())
print("")

s_time = time.time()
clf.fit(X, y)
e_time = time.time()

print(f"Tempo total de execução: \n{e_time - s_time} seconds\n")
```

```
Grid search para parametros [C, gamma, epsilon] em grid 5x5x5
Params:
{'cv': 5, 'error_score': nan, 'estimator__C': 1.0, 'estimator__cache_size': 200,
'estimator__coef0': 0.0, 'estimator__degree': 3, 'estimator__epsilon': 0.1,
'estimator__gamma': 'scale', 'estimator__kernel': 'rbf', 'estimator__max_iter':
-1, 'estimator__shrinking': True, 'estimator__tol': 0.001, 'estimator__verbose':
False, 'estimator': SVR(), 'iid': 'deprecated', 'n_jobs': -1, 'param_grid':
{'C': array([3.1250e-02, 1.0000e+00, 3.2000e+01, 1.0240e+03, 3.2768e+04]),
'gamma': array([3.05175781e-05, 6.90533966e-04, 1.56250000e-02, 3.53553391e-01,
       8.00000000e+00]), 'epsilon': array([0.05  , 0.2875, 0.525 , 0.7625, 1.
])}, 'pre_dispatch': '2*n_jobs', 'refit': True, 'return_train_score': False,
'scoring': 'neg_mean_squared_error', 'verbose': 0}

Tempo total de execução:
8.299102306365967 seconds
```

[7]:
```
df = pd.DataFrame(clf.cv_results_)
cols = ['rank_test_score', 'mean_test_score', 'std_test_score',
        'param_C', 'param_gamma', 'param_epsilon']
n_top = 5

# Coluna mean_test_score com valores negativos por conta da padronização de␣
 ↪GridSearchCV
df['mean_test_score'] = df['mean_test_score']*-1

print(f'Top {n_top} melhores valores para os hiperparametros e o RMSE para␣
 ↪esses valores (coluna mean_test_score):')
```

```
display(df.sort_values('rank_test_score')[:n_top][cols].
 ↪set_index('rank_test_score'))
```

Top 5 melhores valores para os hiperparametros e o RMSE para esses valores (coluna mean_test_score):

rank_test_score	mean_test_score	std_test_score	param_C	param_gamma	param_epsilon
1	14.812549	5.866103	32768.0	0.000031	0.2875
2	14.850936	6.118843	32768.0	0.000031	0.05
3	15.349063	5.557397	32768.0	0.000031	0.525
4	15.521831	5.333289	32768.0	0.000031	0.7625
5	15.542174	4.902948	32768.0	0.000031	1.0

0.3 3) Otimização bayesiana

[8]:
```
n_folds = 5
n_iter = 125
param_range=5
print(f"Otimização bayesiana com n_iter=125\n")

# sum parameter ranges for rbf kernel
C_range = (2e-5, 2e15, 'log-uniform')
gamma_range = (2e-15, 2e3, 'log-uniform')
epsilon = (0.05, 1.0, 'uniform')
param_grid = dict(C=C_range, gamma=gamma_range, epsilon=epsilon, degree=[3],␣
 ↪kernel=['rbf'])
if verbose:
    print(f"param_grid:\n{param_grid}\n")

# regr definition and fit
regr = SVR(kernel='rbf')
clf = BayesSearchCV(regr, param_grid, cv=n_folds, n_iter=n_iter,
                    random_state=seed, scoring='neg_mean_squared_error',␣
 ↪n_jobs=-1)

print("Params:")
print(clf.get_params())
print("")
```

```
s_time = time.time()
clf.fit(X, y)
e_time = time.time()

print(f"Tempo total de execução: \n{e_time - s_time} seconds\n")
```

```
Otimização bayesiana com n_iter=125
Params:
{'cv': 5, 'error_score': 'raise', 'estimator__C': 1.0, 'estimator__cache_size':
200, 'estimator__coef0': 0.0, 'estimator__degree': 3, 'estimator__epsilon': 0.1,
'estimator__gamma': 'scale', 'estimator__kernel': 'rbf', 'estimator__max_iter':
-1, 'estimator__shrinking': True, 'estimator__tol': 0.001, 'estimator__verbose':
False, 'estimator': SVR(), 'fit_params': None, 'iid': True, 'n_iter': 125,
'n_jobs': -1, 'n_points': 1, 'optimizer_kwargs': None, 'pre_dispatch':
'2*n_jobs', 'random_state': 1082141, 'refit': True, 'return_train_score': False,
'scoring': 'neg_mean_squared_error', 'search_spaces': {'C': (2e-05,
2000000000000000.0, 'log-uniform'), 'gamma': (2e-15, 2000.0, 'log-uniform'),
'epsilon': (0.05, 1.0, 'uniform'), 'degree': [3], 'kernel': ['rbf']}, 'verbose':
0}

Tempo total de execução:
317.4695568084717 seconds
```

[9]:
```
df = pd.DataFrame(clf.cv_results_)
cols = ['rank_test_score', 'mean_test_score', 'std_test_score',
        'param_C', 'param_gamma', 'param_epsilon']
n_top = 5

# Coluna mean_test_score com valores negativos por conta da padronização de␣
 ↪RandomizedSearchCv
df['mean_test_score'] = df['mean_test_score']*-1

print(f'Top {n_top} melhores valores para os hiperparametros e o RMSE para␣
 ↪esses valores (coluna mean_test_score):')

display(df.sort_values('rank_test_score')[:n_top][cols].
 ↪set_index('rank_test_score'))
```

```
Top 5 melhores valores para os hiperparametros e o RMSE para esses valores
(coluna mean_test_score):
```

rank_test_score	mean_test_score	std_test_score	param_C	param_gamma
1	37.261907	10.854608	2.403701e+06	0.002173
2	39.198389	10.006766	3.354724e+06	0.001306
3	41.868005	12.298985	4.299913e+12	0.003294

```
4                      41.885996       12.314357  4.619252e+12      0.003324
5                      41.958508       12.291757  4.164697e+12      0.003218

                  param_epsilon
rank_test_score
1                      0.859128
2                      0.858417
3                      0.159628
4                      0.154686
5                      0.154007
```

0.4 4) PSO

[31]:
```
# boundaries for hyperparams
params = ['C' , 'gamma', 'epsilon']
lower_b = [0.0, 0.0, 0.05]
upper_b = [1.0, 1.0, 1.0]

def eval_svr_rmse(hparams):
    # hyperparams calc
    C_exp = 20 * hparams[0] - 5
    C = 2 ** C_exp
    gamma_exp = 18 * hparams[1] - 15
    gamma = 2 ** gamma_exp
    epsilon = hparams[2]

    regr = SVR(kernel='rbf', C=C, gamma=gamma, epsilon=epsilon)

    errs = cross_val_score(regr, X, y, cv=5,␣
 ↪scoring='neg_root_mean_squared_error')
    rmse = np.absolute(np.average(errs))
    return rmse
```

[32]:
```
s_time = time.time()
params, rmse = ps.pso(func=eval_svr_rmse, lb=lower_b, ub=upper_b,
                      swarmsize=11, maxiter=11)
e_time = time.time()

print(f"Tempo total de execução: \n{e_time - s_time} seconds\n")
```

```
Stopping search: maximum iterations reached --> 11
Tempo total de execução:
219.6600785255432 seconds
```

[34]:
```
print("Melhores hyperparametros encontrados:")
print(f"C: {params[0]}")
```

```
print(f"gamma: {params[1]}")
print(f"epsilon: {params[2]}\n")
print(f"RMSE: {rmse}")
```

```
Melhores hyperparametros encontrados:
C: 0.9223555069009095
gamma: 0.0
epsilon: 0.7500774206675289

RMSE: 3.7992387559168215
```
