task

April 27, 2021

1 Tarefa 4: Álgebra Linear e Otimização para ML - MO431A

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

Prof. Jacques Wainer, 2021s1

[1]:
```
# RA & Name
print('265673: ' + 'Gabriel Luciano Gomes')
print('192880: ' + 'Lucas Borges Rondon')
print('265674: ' + 'Paulo Júnio Reis Rodrigues')
```

```
265673: Gabriel Luciano Gomes
192880: Lucas Borges Rondon
265674: Paulo Júnio Reis Rodrigues
```

1.1 Imports necessários para a tarefa

[2]:
```
import numpy as np

from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score, KFold, RandomizedSearchCV,␣
 ↪GridSearchCV
from sklearn.metrics import mean_squared_error

from scipy.stats import loguniform, uniform

from hyperopt import hp, tpe, fmin, STATUS_OK

from pyswarm import pso

from cma import CMAEvolutionStrategy

from simanneal import Annealer
```

1.2 Leitura da base de dados

[3]:
```
X = np.load('db/X.npy')
Y = np.load('db/y.npy')
```

1.3 Variáveis Globais

[4]:
```
# C lower and upper bounds
c_lb = -5
c_ub = 15

# C lower and upper bounds
g_lb = -15
g_ub = 3

#epsilon lower and upper bounds
e_lb = 0.05
e_ub = 1.0

# Base SVM model
base_model = SVR(kernel = 'rbf')
```

1.4 Funções úteis

1.4.1 Computar RMSE

[5]:
```
def compute_rmse(scores):
    # Compute RMSE
    return np.sqrt(np.mean(np.absolute(scores)))
```

1.4.2 Calcular cross val score

[6]:
```
def hyperopt_train_test(params):
    ''' Computes the cross validation score
    to be compared in order to identify
    the best value
    @params: list of params to SVR (C, gamma, epsilon and Kernel)
    '''
    clf = SVR(**params)
    return cross_val_score(clf, X, Y).mean()
```

1.4.3 SVM Regressor

[7]:
```
def compute_SVM_result(c, gamma, epsilon):
    # define cross validation score
    cv = KFold(n_splits = 5, random_state = 1, shuffle = True)

    # Compute SVM
    svr = SVR(kernel = 'rbf', C = c, gamma = gamma, epsilon = epsilon)

    # SVM scores
    scores = cross_val_score(svr, X, Y, scoring = ('neg_mean_squared_error'),␣
 ↪cv = cv)

    show_results(c, gamma, epsilon, compute_rmse(scores))
```

1.4.4 Exibir resultados

[8]:
```
def show_results(c, gamma, epsilon, rmse) :
    print('----- Best values of hyperparameters ----- \n' +
      f'C: {round(c, 6)}\ngamma: {round(gamma, 6)} \nepsilon: {round(epsilon,␣
 ↪6)} \n' +
      '----- RMSE for given values ----- \n' +
      f'RMSE: {round(rmse, 6)}')
```

1.5 Random Search

[9]:
```
# Search space
space = dict()
space['C'] = loguniform(2**c_lb, 2**c_ub)
space['gamma'] = loguniform(2**g_lb, 2**g_ub)
space['epsilon'] = uniform(e_lb, e_ub)

# define search
search = RandomizedSearchCV(base_model,
                            space,
                            n_iter = 125,
                            scoring = 'neg_mean_squared_error',
                            n_jobs = -1,
                            cv = 5,
                            random_state = 1)
result = search.fit(X, Y)
c = result.best_params_['C']
g = result.best_params_['gamma']
e = result.best_params_['epsilon']
```

1.5.1 Resultados obtidos

[10]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 8584.928547
gamma: 3.2e-05
epsilon: 0.623679
```

```
----- RMSE for given values -----
RMSE: 4.023489
```

1.6 Grid Search

[11]:
```
# grid size
g_size = 5

# Search space
space = dict()
space['C'] = loguniform.rvs(2**c_lb, 2**c_ub, size = g_size)
space['gamma'] = loguniform.rvs(2**g_lb, 2**g_ub, size = g_size)
space['epsilon'] = uniform.rvs(e_lb, e_ub, size = g_size)

# define search
search = GridSearchCV(base_model,
                      space,
                      scoring = 'neg_mean_squared_error',
                      n_jobs = -1,
                      cv = 5)

result = search.fit(X, Y)
c = result.best_params_['C']
e = result.best_params_['epsilon']
g = result.best_params_['gamma']
```

1.6.1 Resultados obtidos

[12]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 2145.208037
gamma: 5.9e-05
epsilon: 0.276161
----- RMSE for given values -----
RMSE: 4.326304
```

1.7 Bayesian Optimization

[13]:
```
def objective_function_bo(params):
    ''' Callable function to compare SVR scores.
    For this example, loss will be used.
    @params: list of params to SVR (C, gamma, epsilon and Kernel)
    '''
    C =  params['C']
    gamma = params['gamma']
    epsilon = params['epsilon']
    acc = hyperopt_train_test({'C': 2**C, 'gamma': 2**gamma, 'epsilon':␣
 ↪epsilon})

    return {'loss': -acc, 'status': STATUS_OK}
```

[14]:
```
space = {
    'C': hp.uniform('C', c_lb, c_ub),
    'gamma': hp.uniform('gamma', g_lb, g_ub),
    'epsilon': hp.uniform('epsilon', e_lb, e_ub)
}

best = fmin(objective_function_bo, space, algo = tpe.suggest, max_evals = 125)
c = 2** best['C']
e = 2** best['epsilon']
g = 2** best['gamma']
```

```
100%|                        | 125/125 [03:33<00:00,
1.71s/trial, best loss: -0.8276213557206253]
```

1.7.1 Resultados obtidos

[15]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 20081.964026
gamma: 3.2e-05
epsilon: 1.309902
----- RMSE for given values -----
RMSE: 3.976988
```

1.8 PSO

[16]:
```
def objective_function_pso(x):
    C, gamma, epsilon = x
    kernel =  'rbf'
    acc = hyperopt_train_test({'C': 2**C, 'gamma': 2**gamma, 'epsilon':␣
 ↪epsilon, 'kernel': kernel})
    return -acc
```

[17]:
```
# upper and lower bounds for C, gamma and epsilon respectively
lb = [c_lb, g_lb, e_lb]
ub = [c_ub, g_ub, e_ub]

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

c = 2** xopt[0]
g = 2** xopt[1]
e = xopt[2]
```

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

1.8.1 Resultados obtidos

[18]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 22121.916813
gamma: 3.1e-05
epsilon: 0.180888
----- RMSE for given values -----
RMSE: 4.101285
```

1.9 Simulated Annealing

Classe Filha do Annealing, necessária para funcionamento

[19]:
```
class SimulatedAnnealing(Annealer):
    """Test annealer to objetctive function"""

    def __init__(self, state):
        super(SimulatedAnnealing, self).__init__(state)

    def move(self):
        """Swaps params of SVM."""
        self.state[0] = 2 ** np.random.uniform(low = c_lb, high = c_ub)
        self.state[1] = 2 ** np.random.uniform(low = g_lb, high = g_ub)
        self.state[2] = np.random.uniform(low = e_lb, high = e_ub)

    def energy(self):
        """Calculates cross validation score"""
        C, gamma, epsilon = self.state[0], self.state[1], self.state[2]
        kernel =  'rbf'

        return self.objective_function_sa({
            'C': C,
            'gamma': gamma,
            'epsilon': epsilon,
            'kernel': kernel
        })

    def objective_function_sa(self, x):
        acc = hyperopt_train_test(x)
        return -acc
```

[20]:
```
initial_state = [
      2 ** np.random.uniform(low = c_lb, high = c_ub),
      2 ** np.random.uniform(low = g_lb, high = g_ub),
      np.random.uniform(low = e_lb, high = e_ub)
]

sa = SimulatedAnnealing(initial_state)
sa.steps = 125

xopt, fopt = sa.anneal()
c = xopt[0]
g = xopt[1]
e = xopt[2]
```

```
 Temperature        Energy    Accept    Improve     Elapsed   Remaining
     2.50000         -0.79     0.00%      0.00%     0:00:28     0:00:00
```

1.9.1 Resultados obtidos

[21]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 2715.058986
gamma: 6.6e-05
epsilon: 0.393088
----- RMSE for given values -----
RMSE: 4.30675
```

1.10 CMA-ES

[22]:
```
def objetive_function_CMA_ES(x):
    C, gamma, epsilon = x
    kernel =  'rbf'
    acc = hyperopt_train_test({'C': 2** (c_lb + C*20),
                               'gamma': 2** (g_lb + gamma*18),
                               'epsilon': abs(epsilon),
                               'kernel': kernel})
    return -acc
```

[23]:
```
# Define initial bounds
lw = [0.0, 0.0, 0.0]
up = [1.0, 1.0, 1.0]

# Initial values
x0 = 3 * [0.05]
sigma = 0.25

result = CMAEvolutionStrategy(x0, sigma, {'bounds': [lw, up]})
result.optimize(objetive_function_CMA_ES, iterations = 125)

# extract best hyperparameters values
```

```python
c = 2 ** (c_lb + result.best.x[0] * 20)
g = 2 ** (g_lb + result.best.x[1] * 18)
e = abs(result.best.x[2])
```

```
(3_w,7)-aCMA-ES (mu_w=2.3,w_1=58%) in dimension 3 (seed=692023, Tue Apr 27
21:22:40 2021)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      7 -2.315199708422314e-01 1.0e+00 2.08e-01  2e-01  2e-01 0:00.6
    2     14 -3.320906359970631e-01 1.2e+00 2.14e-01  2e-01  2e-01 0:01.1
    3     21 -6.616681538101122e-01 1.5e+00 2.40e-01  2e-01  3e-01 0:01.9
    4     28 -4.394857983716051e-01 1.7e+00 2.65e-01  2e-01  3e-01 0:10.7
    5     35 -8.039246495667275e-01 1.6e+00 4.26e-01  3e-01  5e-01 0:34.0
    6     42 -7.403545289565399e-01 1.7e+00 4.57e-01  3e-01  5e-01 0:47.5
    7     49 -8.196302081259074e-01 1.8e+00 4.89e-01  3e-01  5e-01 1:00.1
    9     63 -8.233516654906345e-01 2.5e+00 5.97e-01  3e-01  8e-01 1:25.2
   10     70 -7.586400226082639e-01 3.1e+00 5.58e-01  3e-01  7e-01 1:38.4
   11     77 -6.634747558410108e-01 3.2e+00 5.08e-01  2e-01  6e-01 1:52.1
   12     84 -8.198820025351490e-01 3.2e+00 5.06e-01  2e-01  6e-01 2:09.0
   13     91 -8.295974114681991e-01 3.2e+00 4.99e-01  2e-01  6e-01 2:24.9
   14     98 -8.138837583418976e-01 3.2e+00 4.76e-01  2e-01  6e-01 2:37.9
   15    105 -8.196443479403580e-01 4.0e+00 3.87e-01  1e-01  5e-01 3:11.5
   16    112 -8.276502570702018e-01 4.9e+00 3.58e-01  1e-01  5e-01 3:55.2
   17    119 -8.282884659162324e-01 5.4e+00 2.99e-01  8e-02  4e-01 4:34.8
   18    126 -8.297361520489789e-01 6.0e+00 2.51e-01  6e-02  3e-01 5:08.7
   19    133 -8.240430506975756e-01 6.1e+00 2.01e-01  4e-02  2e-01 5:52.3
   20    140 -8.288105243544004e-01 5.9e+00 1.67e-01  4e-02  1e-01 6:27.7
   21    147 -8.292921700362971e-01 4.7e+00 1.32e-01  2e-02  1e-01 6:59.4
   22    154 -8.298784682262872e-01 4.7e+00 1.12e-01  2e-02  9e-02 7:43.8
   23    161 -8.312632365136265e-01 4.6e+00 1.12e-01  2e-02  1e-01 8:23.6
   24    168 -8.310541353048487e-01 5.9e+00 1.05e-01  2e-02  9e-02 9:11.0
   25    175 -8.324004245997431e-01 5.9e+00 9.21e-02  1e-02  7e-02 9:44.5
   26    182 -8.329107206436044e-01 5.6e+00 9.85e-02  1e-02  9e-02 10:24.8
   27    189 -8.338952779813829e-01 7.6e+00 1.10e-01  1e-02  1e-01 11:05.7
   28    196 -8.330245120877310e-01 8.9e+00 1.23e-01  2e-02  1e-01 12:05.1
   29    203 -8.330840041671852e-01 9.9e+00 9.88e-02  1e-02  9e-02 12:57.1
   30    210 -8.336916894944840e-01 9.2e+00 1.00e-01  1e-02  1e-01 13:43.7
   31    217 -8.338528197596025e-01 9.7e+00 9.13e-02  1e-02  8e-02 14:40.8
   32    224 -8.338676368147739e-01 8.7e+00 8.02e-02  9e-03  6e-02 15:51.9
   33    231 -8.337940485229380e-01 8.4e+00 6.91e-02  7e-03  5e-02 17:07.7
   34    238 -8.338454470263843e-01 8.3e+00 6.47e-02  6e-03  5e-02 18:13.8
   35    245 -8.336482249155767e-01 8.1e+00 5.81e-02  5e-03  4e-02 19:19.8
   36    252 -8.338511774292681e-01 7.5e+00 4.43e-02  3e-03  3e-02 20:24.8
   37    259 -8.339403507527837e-01 8.7e+00 4.04e-02  3e-03  2e-02 21:28.2
   38    266 -8.338890394533396e-01 8.8e+00 4.53e-02  4e-03  2e-02 22:27.6
   39    273 -8.339660945864326e-01 6.0e+00 3.82e-02  3e-03  2e-02 23:25.2
   40    280 -8.339012786707819e-01 6.4e+00 3.27e-02  3e-03  2e-02 24:23.7
   41    287 -8.339113640474632e-01 6.4e+00 3.08e-02  2e-03  1e-02 25:24.7
   42    294 -8.339612526494264e-01 5.6e+00 2.87e-02  2e-03  1e-02 26:26.6
```

```
 43    301 -8.339772334409261e-01 5.9e+00 2.64e-02  2e-03  1e-02 27:32.3
 44    308 -8.339017570021845e-01 6.5e+00 2.63e-02  2e-03  1e-02 28:28.6
 45    315 -8.339161236774103e-01 8.0e+00 3.15e-02  2e-03  1e-02 29:28.0
 46    322 -8.340864847659452e-01 7.2e+00 2.91e-02  2e-03  1e-02 30:27.4
 47    329 -8.339908162773710e-01 7.4e+00 2.65e-02  2e-03  1e-02 31:26.7
 48    336 -8.340247310720701e-01 8.3e+00 2.64e-02  2e-03  1e-02 32:28.9
 49    343 -8.339706757686258e-01 7.8e+00 2.81e-02  2e-03  1e-02 33:28.9
 50    350 -8.339910494771956e-01 7.8e+00 2.62e-02  1e-03  1e-02 34:28.8
 51    357 -8.339462095181496e-01 8.7e+00 2.22e-02  1e-03  1e-02 35:33.6
 52    364 -8.339215890052806e-01 9.3e+00 2.57e-02  1e-03  1e-02 36:34.9
 53    371 -8.339035745294918e-01 1.2e+01 2.39e-02  1e-03  1e-02 37:33.4
 54    378 -8.339089860690626e-01 1.2e+01 2.35e-02  1e-03  1e-02 38:34.5
 55    385 -8.339515766497179e-01 1.1e+01 2.39e-02  1e-03  1e-02 39:34.9
 56    392 -8.340141763255258e-01 1.0e+01 2.15e-02  1e-03  1e-02 40:32.9
 57    399 -8.339929821362009e-01 1.1e+01 1.74e-02  8e-04  7e-03 41:30.7
 59    413 -8.339306319761366e-01 1.1e+01 1.75e-02  8e-04  7e-03 43:30.7
 60    420 -8.338735781843611e-01 1.3e+01 1.38e-02  6e-04  6e-03 44:41.3
 61    427 -8.339814701176573e-01 1.6e+01 1.12e-02  4e-04  4e-03 45:50.2
 62    434 -8.339744702736205e-01 1.4e+01 1.18e-02  5e-04  4e-03 46:52.0
 63    441 -8.339514680970584e-01 1.4e+01 1.16e-02  4e-04  4e-03 47:54.9
 64    448 -8.339355297377370e-01 1.4e+01 1.13e-02  4e-04  4e-03 48:57.2
 66    462 -8.339389850754262e-01 1.4e+01 9.39e-03  3e-04  3e-03 51:04.1
 67    469 -8.339814871029698e-01 1.3e+01 1.05e-02  4e-04  3e-03 52:07.8
 68    476 -8.339413053346357e-01 1.2e+01 1.04e-02  4e-04  3e-03 53:15.2
 70    490 -8.340404016169962e-01 1.2e+01 8.97e-03  3e-04  2e-03 55:16.9
 72    504 -8.339456067503515e-01 1.0e+01 8.72e-03  3e-04  2e-03 57:30.0
 74    518 -8.339570637100742e-01 7.9e+00 7.56e-03  2e-04  2e-03 59:34.3
 76    532 -8.339928395365461e-01 7.0e+00 5.93e-03  2e-04  1e-03 61:35.8
 78    546 -8.339360340940395e-01 8.8e+00 4.81e-03  2e-04  8e-04 63:42.2
 80    560 -8.340093381203697e-01 8.5e+00 4.69e-03  1e-04  8e-04 65:45.1
 82    574 -8.339920259474475e-01 9.5e+00 4.44e-03  1e-04  8e-04 67:50.7
 84    588 -8.339761550613118e-01 1.0e+01 6.62e-03  2e-04  1e-03 69:55.1
 86    602 -8.339249046545995e-01 1.1e+01 5.43e-03  1e-04  1e-03 71:55.3
 88    616 -8.339486717224622e-01 1.5e+01 3.76e-03  8e-05  7e-04 73:53.5
 90    630 -8.340173131411195e-01 1.4e+01 2.56e-03  4e-05  4e-04 75:55.7
 92    644 -8.339961098024264e-01 1.5e+01 2.44e-03  4e-05  3e-04 77:59.0
 94    658 -8.339349389074446e-01 1.4e+01 3.66e-03  5e-05  6e-04 80:00.8
 96    672 -8.339342244560008e-01 1.6e+01 2.72e-03  4e-05  4e-04 82:08.8
 98    686 -8.339801310255759e-01 2.1e+01 4.19e-03  5e-05  8e-04 84:36.8
100    700 -8.339668936875541e-01 1.8e+01 3.92e-03  5e-05  5e-04 86:49.4
102    714 -8.340025792357771e-01 2.1e+01 3.05e-03  3e-05  4e-04 89:18.7
104    728 -8.339863468388551e-01 2.1e+01 3.27e-03  4e-05  4e-04 91:35.2
106    742 -8.340023531448537e-01 2.3e+01 2.92e-03  3e-05  4e-04 93:59.7
108    756 -8.339611267704488e-01 2.2e+01 2.07e-03  2e-05  2e-04 96:15.9
110    770 -8.340263582420210e-01 1.9e+01 1.98e-03  2e-05  2e-04 98:34.6
112    784 -8.339572518880312e-01 1.8e+01 1.62e-03  1e-05  1e-04 100:59.8
114    798 -8.339888703136917e-01 1.6e+01 1.45e-03  9e-06  1e-04 103:19.8
116    812 -8.339348474856715e-01 1.2e+01 1.38e-03  9e-06  8e-05 105:39.3
```

```
  118    826 -8.339501741135770e-01 8.3e+00 1.14e-03  7e-06  5e-05 107:41.3
  120    840 -8.339858527303674e-01 8.1e+00 1.36e-03  8e-06  7e-05 109:45.9
  122    854 -8.340007084254794e-01 9.6e+00 1.59e-03  8e-06  7e-05 111:52.5
  124    868 -8.339417148773597e-01 1.0e+01 1.77e-03  8e-06  1e-04 113:58.7
```

1.10.1 Resultados obtidos

[24]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 23682.252806
gamma: 3.1e-05
epsilon: 0.001736
----- RMSE for given values -----
RMSE: 4.210726
```