task

April 28, 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: 87.143273
gamma: 0.000423
epsilon: 1.002995
----- RMSE for given values -----
RMSE: 5.22651
```

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']
g = 2** best['gamma']
e = best['epsilon']
```

```
100%|                          | 125/125 [03:52<00:00,
1.86s/trial, best loss: -0.8247652336636048]
```

1.7.1 Resultados obtidos

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

```
----- Best values of hyperparameters -----
C: 14637.801447
gamma: 3.1e-05
epsilon: 0.579594
----- RMSE for given values -----
RMSE: 3.991933
```

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: 17082.149431
gamma: 3.1e-05
epsilon: 0.775143
----- RMSE for given values -----
RMSE: 3.981901
```

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          0.03   100.00%     0.00%     0:02:49     0:00:00
```

1.9.1 Resultados obtidos

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

```
----- Best values of hyperparameters -----
C: 2718.172374
gamma: 9.8e-05
epsilon: 0.121457
----- RMSE for given values -----
RMSE: 4.31529
```

1.10 CMA-ES

[22]:
```
def objetive_function_CMA_ES(x):
    C, gamma, epsilon = x
    kernel =  'rbf'

    # compute cross val score with normalized hyperparams
    acc = hyperopt_train_test({'C': 2** (c_lb + C*20),
                               'gamma': 2** (g_lb + gamma*18),
                               'epsilon': e_lb + epsilon*0.95 ,
                               'kernel': kernel})
    return -acc
```

[24]:
```
# 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 normalizing it
c = 2 ** (c_lb + result.best.x[0] * 20)
g = 2 ** (g_lb + result.best.x[1] * 18)
e = e_lb + result.best.x[2]*0.95
```

```
(3_w,7)-aCMA-ES (mu_w=2.3,w_1=58%) in dimension 3 (seed=747261, Wed Apr 28
11:55:18 2021)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      7 -3.161849215501714e-01 1.0e+00 2.42e-01  2e-01  3e-01 0:01.5
    2     14 -5.906189742695303e-01 1.3e+00 2.79e-01  2e-01  4e-01 0:03.4
    3     21 -6.706682171259113e-01 1.7e+00 2.94e-01  2e-01  4e-01 0:06.0
    4     28 -8.213829486441921e-01 2.3e+00 3.55e-01  3e-01  6e-01 0:34.8
    5     35 -6.780610386013299e-01 2.6e+00 4.27e-01  3e-01  7e-01 1:17.9
    6     42 -7.402443128022795e-01 3.0e+00 4.16e-01  3e-01  7e-01 1:36.5
    7     49 -6.895381527236459e-01 2.5e+00 3.70e-01  3e-01  5e-01 1:56.7
    9     63 -6.429845008328915e-01 2.6e+00 3.16e-01  2e-01  4e-01 3:22.8
   10     70 -7.416259106937801e-01 2.6e+00 3.14e-01  2e-01  3e-01 4:06.6
   12     84 -7.746931352354519e-01 1.9e+00 2.32e-01  1e-01  2e-01 4:34.7
   13     91 -8.080529677968983e-01 2.0e+00 1.95e-01  1e-01  2e-01 5:54.0
   14     98 -8.137512756742638e-01 2.0e+00 1.92e-01  1e-01  1e-01 7:05.0
   15    105 -7.799164633435346e-01 1.8e+00 2.19e-01  1e-01  2e-01 9:25.4
   16    112 -7.762640531971401e-01 2.2e+00 1.81e-01  9e-02  1e-01 9:46.8
   17    119 -8.260046180390657e-01 2.3e+00 1.94e-01  1e-01  2e-01 10:19.6
   18    126 -8.239612761920622e-01 2.8e+00 1.78e-01  8e-02  2e-01 11:17.8
   19    133 -8.281600007277561e-01 3.0e+00 1.53e-01  6e-02  1e-01 12:00.7
   20    140 -8.267519832444725e-01 2.7e+00 1.87e-01  7e-02  1e-01 13:31.8
   21    147 -8.324574114284491e-01 2.8e+00 2.32e-01  7e-02  2e-01 15:02.5
   22    154 -8.295438957489510e-01 4.2e+00 2.11e-01  6e-02  2e-01 16:50.5
   23    161 -8.270783822415473e-01 4.8e+00 2.43e-01  5e-02  3e-01 17:54.1
   24    168 -8.315710679777943e-01 5.9e+00 2.22e-01  4e-02  2e-01 19:50.7
   25    175 -8.304685379093746e-01 6.6e+00 1.81e-01  3e-02  2e-01 21:16.7
   26    182 -8.325488373535830e-01 6.9e+00 1.65e-01  3e-02  2e-01 23:08.2
   27    189 -8.326409726264019e-01 7.7e+00 1.50e-01  2e-02  1e-01 26:23.8
   28    196 -8.315053820810648e-01 9.3e+00 1.43e-01  2e-02  1e-01 28:22.6
   29    203 -8.320219923451045e-01 1.0e+01 1.33e-01  2e-02  1e-01 30:52.1
   30    210 -8.313363240016198e-01 1.1e+01 1.28e-01  2e-02  1e-01 32:36.4
   31    217 -8.319472959442255e-01 1.2e+01 1.18e-01  2e-02  1e-01 34:56.3
   32    224 -8.322636810973110e-01 1.0e+01 1.16e-01  1e-02  1e-01 37:19.1
   33    231 -8.318724261876749e-01 1.2e+01 1.13e-01  1e-02  9e-02 39:29.5
   34    238 -8.324551766191937e-01 9.2e+00 1.45e-01  2e-02  1e-01 41:17.9
   35    245 -8.328123308043700e-01 7.0e+00 1.40e-01  2e-02  1e-01 43:55.2
   36    252 -8.330384117533282e-01 7.0e+00 1.21e-01  2e-02  7e-02 45:57.5
   37    259 -8.324946017289137e-01 6.2e+00 1.24e-01  1e-02  7e-02 48:37.9
   38    266 -8.329022547179026e-01 5.5e+00 1.05e-01  1e-02  6e-02 51:06.6
   39    273 -8.322015811156632e-01 5.6e+00 7.75e-02  8e-03  4e-02 52:58.3
   40    280 -8.330057873465607e-01 5.0e+00 6.21e-02  6e-03  3e-02 54:32.7
   41    287 -8.329951102497107e-01 5.3e+00 4.89e-02  4e-03  2e-02 55:46.1
```

42	294	-8.332001014967088e-01	5.6e+00	6.06e-02	5e-03	4e-02	57:09.8
43	301	-8.333172563559801e-01	7.9e+00	1.03e-01	8e-03	7e-02	58:36.7
44	308	-8.332672862032264e-01	9.0e+00	1.24e-01	1e-02	8e-02	60:08.3
45	315	-8.332199178557961e-01	8.4e+00	1.12e-01	8e-03	6e-02	61:35.9
46	322	-8.332711416506111e-01	7.3e+00	9.49e-02	7e-03	5e-02	63:02.3
47	329	-8.332076755666373e-01	7.3e+00	8.75e-02	5e-03	5e-02	64:39.2
48	336	-8.332452044330318e-01	9.1e+00	8.93e-02	6e-03	5e-02	66:16.5
49	343	-8.332441620407796e-01	7.3e+00	9.96e-02	8e-03	5e-02	67:48.5
50	350	-8.330584953829325e-01	6.0e+00	1.03e-01	8e-03	5e-02	69:18.6
51	357	-8.331798361657121e-01	5.9e+00	8.32e-02	6e-03	4e-02	70:52.6
52	364	-8.331872355712620e-01	6.7e+00	6.85e-02	5e-03	3e-02	72:28.3
53	371	-8.333735796783273e-01	6.6e+00	6.12e-02	4e-03	2e-02	74:05.1
54	378	-8.333473504286582e-01	6.8e+00	5.88e-02	4e-03	2e-02	75:47.0
56	392	-8.333308543341609e-01	5.9e+00	5.45e-02	4e-03	2e-02	77:14.8
58	406	-8.334048860094349e-01	5.2e+00	4.76e-02	3e-03	2e-02	78:43.3
60	420	-8.334326074288210e-01	6.6e+00	4.53e-02	3e-03	1e-02	80:16.7
61	427	-8.334053017974259e-01	7.8e+00	3.67e-02	2e-03	1e-02	81:12.5
63	441	-8.333993718474753e-01	6.9e+00	2.70e-02	1e-03	7e-03	82:52.4
65	455	-8.334030003470403e-01	7.8e+00	2.97e-02	2e-03	7e-03	84:35.1
67	469	-8.334239253568347e-01	6.4e+00	2.41e-02	1e-03	4e-03	86:12.3
69	483	-8.334106542772919e-01	6.1e+00	1.97e-02	8e-04	3e-03	87:47.7
71	497	-8.333829358196161e-01	6.0e+00	1.66e-02	7e-04	2e-03	89:26.2
73	511	-8.334530536228737e-01	4.0e+00	1.54e-02	5e-04	2e-03	91:06.6
75	525	-8.333040202462536e-01	5.2e+00	1.17e-02	3e-04	1e-03	92:46.5
77	539	-8.333561104277484e-01	4.8e+00	1.03e-02	2e-04	1e-03	94:27.4
79	553	-8.333268735400189e-01	4.8e+00	8.36e-03	2e-04	7e-04	96:06.2
81	567	-8.334731816265112e-01	5.1e+00	8.99e-03	2e-04	7e-04	97:46.7
83	581	-8.333903937113047e-01	4.8e+00	7.96e-03	2e-04	7e-04	99:26.6
85	595	-8.334264429671236e-01	6.5e+00	6.12e-03	1e-04	5e-04	101:39.1
86	602	-8.333738779303065e-01	6.6e+00	6.63e-03	1e-04	6e-04	102:49.2
87	609	-8.334759092132001e-01	6.3e+00	8.87e-03	2e-04	7e-04	104:12.0
88	616	-8.333537481159123e-01	4.8e+00	7.12e-03	2e-04	6e-04	106:27.1
89	623	-8.333434938891504e-01	4.5e+00	6.06e-03	1e-04	4e-04	109:06.2
90	630	-8.333747778666586e-01	4.6e+00	5.02e-03	9e-05	3e-04	111:32.2
91	637	-8.334017037250556e-01	5.3e+00	5.66e-03	1e-04	5e-04	113:58.5
92	644	-8.333394794143224e-01	6.2e+00	6.37e-03	2e-04	5e-04	116:09.8
93	651	-8.334099624760309e-01	6.1e+00	5.91e-03	1e-04	5e-04	118:38.2
94	658	-8.333923362009698e-01	6.5e+00	4.68e-03	1e-04	3e-04	120:40.5
95	665	-8.333588683495814e-01	6.0e+00	5.40e-03	1e-04	4e-04	123:06.6
96	672	-8.333532761552190e-01	5.7e+00	5.38e-03	1e-04	3e-04	125:20.1
97	679	-8.334374637464714e-01	5.0e+00	4.95e-03	1e-04	3e-04	127:44.6
98	686	-8.333650789574953e-01	5.0e+00	4.32e-03	9e-05	2e-04	130:03.5
99	693	-8.334113257699322e-01	6.1e+00	3.94e-03	8e-05	2e-04	132:34.7
100	700	-8.333683675906280e-01	5.9e+00	3.98e-03	8e-05	2e-04	134:54.9
101	707	-8.334319109171536e-01	5.4e+00	4.55e-03	1e-04	2e-04	137:12.2
102	714	-8.333932517834567e-01	5.8e+00	5.03e-03	1e-04	2e-04	139:23.4
103	721	-8.333421494540343e-01	5.7e+00	4.61e-03	1e-04	2e-04	141:55.1
104	728	-8.334696474219075e-01	5.4e+00	4.16e-03	9e-05	2e-04	144:09.5

```
 105    735 -8.334476533583078e-01 5.5e+00 4.20e-03  1e-04  2e-04 145:49.2
 107    749 -8.333581296496183e-01 5.0e+00 4.38e-03  1e-04  2e-04 148:03.8
 109    763 -8.334230704587849e-01 5.3e+00 6.38e-03  2e-04  3e-04 149:42.3
 111    777 -8.333878278366275e-01 9.8e+00 6.28e-03  2e-04  3e-04 151:27.2
 113    791 -8.334360667020935e-01 1.1e+01 5.57e-03  2e-04  3e-04 153:24.9
 115    805 -8.334081688003154e-01 1.1e+01 5.90e-03  2e-04  3e-04 155:06.4
 117    819 -8.333917604478561e-01 1.3e+01 5.42e-03  1e-04  3e-04 156:52.7
 119    833 -8.333930953656334e-01 1.1e+01 6.14e-03  2e-04  3e-04 158:27.6
 122    854 -8.333720450301655e-01 9.8e+00 7.47e-03  2e-04  4e-04 160:48.0
 124    868 -8.333165388888139e-01 9.9e+00 1.16e-02  3e-04  7e-04 162:25.0
```

1.10.1 Resultados obtidos

[25]:
```
compute_SVM_result(c, g, e)
```

```
----- Best values of hyperparameters -----
C: 22610.006226
gamma: 3.1e-05
epsilon: 0.050666
----- RMSE for given values -----
RMSE: 4.183407
```