M4-L2-P3

October 1, 2023

1 Problem 6 (5 points)

In this problem, we will investigate kernel selection and regularization strength in support vector regression for a 1-D problem.

Run each cell below, then try out the interactive plot to answer the questions.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR

xs = np.array([0.094195,0.10475,0.12329,0.12767,0.1343,0.11321,0.16134,0.
 ↪16622,0.15704,0.16892,0.1707,0.19564,0.18697,0.20818,0.22071,0.21833,0.
 ↪23029,0.23398,0.25217,0.25168,0.2538,0.25143,0.27121,0.27319,0.28675,0.
 ↪29971,0.30451,0.32319,0.32141,0.33977,0.35378,0.37053,0.35916,0.36534,0.
 ↪3807,0.38696,0.41073,0.41095,0.41302,0.42177,0.42517,0.43633,0.42191,0.
 ↪45198,0.4606,0.4838,0.4664,0.48132,0.49296,0.51028,0.51747,0.499,0.49948,0.
 ↪53049,0.53986,0.55444,0.54966,0.56389,0.5544,0.56139,0.58974,0.59864,0.
 ↪59467,0.6122,0.61911,0.62601,0.63302,0.63993,0.65452,0.64038,0.67782,0.
 ↪66911,0.67807,0.68518,0.68705,0.70398,0.72397,0.71793,0.72931,0.76366,0.
 ↪75441,0.73797,0.7741,0.77121,0.77784,0.7816,0.79257,0.80469,0.82256,0.
 ↪82495,0.83913,0.8226,0.84766,0.83838,0.8493,0.89643,0.86783,0.89621,0.
 ↪90823,0.90054,])
ys = np.array([0.51123,0.50881,0.50546,0.50756,0.51653,0.50797,0.49658,0.
 ↪50899,0.50218,0.50242,0.50906,0.50466,0.48063,0.49306,0.48622,0.51558,0.
 ↪50493,0.48378,0.518,0.49348,0.51459,0.53657,0.54106,0.54207,0.56463,0.
 ↪56601,0.61192,0.61208,0.63699,0.64194,0.67329,0.70949,0.74668,0.77664,0.
 ↪82362,0.84736,0.89991,0.91268,0.92689,0.93635,0.94732,0.95202,0.94112,0.
 ↪92713,0.89726,0.88055,0.83289,0.78465,0.75197,0.71588,0.64221,0.58237,0.
 ↪52391,0.45466,0.37946,0.31505,0.25479,0.18915,0.14154,0.084572,0.058735,0.
 ↪027538,0.013328,0.0098045,0.068816,0.094916,0.10225,0.16912,0.21646,0.
 ↪27493,0.33072,0.40278,0.48282,0.53813,0.63165,0.69685,0.74494,0.8089,0.
 ↪8693,0.89515,0.92841,0.94583,0.93489,0.91862,0.92811,0.90047,0.86258,0.
 ↪85054,0.82246,0.83096,0.78313,0.74352,0.71369,0.69591,0.65134,0.65297,0.
 ↪61356,0.59983,0.57448,0.56923,])
```

```
x_gt = np.array([0.0,0.010101,0.020202,0.030303,0.040404,0.050505,0.060606,0.
 ↪070707,0.080808,0.090909,0.10101,0.11111,0.12121,0.13131,0.14141,0.15152,0.
 ↪16162,0.17172,0.18182,0.19192,0.20202,0.21212,0.22222,0.23232,0.24242,0.
 ↪25253,0.26263,0.27273,0.28283,0.29293,0.30303,0.31313,0.32323,0.33333,0.
 ↪34343,0.35354,0.36364,0.37374,0.38384,0.39394,0.40404,0.41414,0.42424,0.
 ↪43434,0.44444,0.45455,0.46465,0.47475,0.48485,0.49495,0.50505,0.51515,0.
 ↪52525,0.53535,0.54545,0.55556,0.56566,0.57576,0.58586,0.59596,0.60606,0.
 ↪61616,0.62626,0.63636,0.64646,0.65657,0.66667,0.67677,0.68687,0.69697,0.
 ↪70707,0.71717,0.72727,0.73737,0.74747,0.75758,0.76768,0.77778,0.78788,0.
 ↪79798,0.80808,0.81818,0.82828,0.83838,0.84848,0.85859,0.86869,0.87879,0.
 ↪88889,0.89899,0.90909,0.91919,0.92929,0.93939,0.94949,0.9596,0.9697,0.9798,0.
 ↪9899,1.0,])
y_gt = np.array([0.46193,0.47566,0.48699,0.49609,0.50315,0.50836,0.51189,0.
 ↪51393,0.51467,0.51428,0.51294,0.51085,0.50818,0.50512,0.50186,0.49856,0.
 ↪49542,0.49263,0.49035,0.48878,0.4881,0.4885,0.49015,0.49323,0.49794,0.
 ↪50446,0.51298,0.52376,0.53706,0.55316,0.57231,0.59478,0.62084,0.65075,0.
 ↪68477,0.72317,0.76529,0.80864,0.85051,0.88819,0.91898,0.94015,0.94917,0.
 ↪94553,0.93,0.90339,0.86651,0.82017,0.76518,0.70233,0.63243,0.5563,0.47475,0.
 ↪38966,0.3049,0.22456,0.15274,0.093526,0.051005,0.028929,0.027469,0.044659,0.
 ↪078502,0.127,0.18816,0.25999,0.34048,0.42761,0.51845,0.60913,0.69574,0.
 ↪77438,0.84113,0.89208,0.92416,0.93858,0.93795,0.92487,0.90197,0.87185,0.
 ↪83712,0.80039,0.76426,0.73054,0.69893,0.66883,0.63963,0.61072,0.5815,0.
 ↪55136,0.51968,0.48587,0.44931,0.40939,0.36551,0.31706,0.26344,0.20402,0.
 ↪13821,0.065402,])
```

[ ]:
```
%matplotlib inline
from ipywidgets import interact, interactive, fixed, interact_manual, Layout,␣
 ↪FloatSlider, Dropdown



def plotting_function(kernel, log_C, log_epsilon):
    C = np.power(10.,log_C)
    epsilon = np.power(10.,log_epsilon)

    model = SVR(kernel=kernel,C=C,epsilon=epsilon)
    model.fit(xs.reshape(-1,1),ys)

    xfit = np.linspace(0,1,200)
    yfit = model.predict(xfit.reshape(-1,1))

    plt.figure(figsize=(12,7))
    plt.scatter(xs,ys,s=10,c="k",label="Data")
    plt.plot(xfit,yfit,linewidth=3, label="SVR")
    plt.plot(x_gt,y_gt,"--",label="Ground Truth")
    title = f"Kernel: {kernel},  C = {C:.1e}, eps = {epsilon:.1e}"
    plt.legend(loc="lower left")
    plt.xlabel("$x_1$")
```

```python
    plt.ylabel("$y$")
    plt.title(title)
    plt.show()

slider1 = FloatSlider(
    value=0,
    min=-5,
    max=5,
    step=.5,
    description='C',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

slider2 = FloatSlider(
    value=-1,
    min=-7,
    max=-1,
    step=.5,
    description='epsilon',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

dropdown = Dropdown(
    options=['linear', 'rbf','sigmoid'],
    value='linear',
    description='kernel',
    disabled=False,
)


interactive_plot = interactive(
    plotting_function,
    kernel = dropdown,
    log_C = slider1,
    log_epsilon = slider2
    )
output = interactive_plot.children[-1]
output.layout.height = '500px'
```

```
interactive_plot
```

```
[ ]: interactive(children=(Dropdown(description='kernel', options=('linear', 'rbf',
'sigmoid'), value='linear'), Fl…
```

1.1 Questions

1. Which kernel produced the best fit overall? (Assume this kernel for subsequent questions.)

   Linear produced the best overall fit

2. As 'C' increases, does model performance on in-sample data generally improve or worsen?

   It generally improves until a certain point where it starts to overfit

3. As 'C' increases, does model performance on out-of-sample data (on the intervals [0.0, 0.1] and [0.9, 1.0]) generally improve or worsen?

   It doesn't necessarily improve all that much past a certaian point but compared to very low C values it does imporve.

4. What 'C' value would you recommend for this kernel?

   I would recommend a C value of 3,200 because it has a good balance of fitting the inner data well while also not overfitting too much

5. What 'epsilon' value would you recommend?

   I would recommend an epsilon of 3.2e-5.