Midterm-Exam

October 7, 2021

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

[1]:
```
NAME = "Solution"
ID = "999999"
```

---

1 AT82.03 Machine Learning Aug 2021: Midterm Examination

Happy Tuesday! This is the midterm for Machine Learning in the August 2021 semester.

This exam is 2.5 hours long. Once the exam starts, you will have exactly 2.5 hours to finish your work and upload your notebook to Google Classroom.

Please fill in this notebook with your code and short answers. Be sure to put all of your code in the cells marked with

`# YOUR CODE GOES HERE`

and please put your answers to the short answer questions exactly where you see the remark

You answer goes here.

Be complete and precise in your answers! Be sure to answer the question that's being asked. Don't dump random information in the hope that it'll give you partial credit. I give generous partial credit, but I will deduct points for answers that are not on point.

Also beware that if I discover any cheating, I will give you a 0 for the entire exam, or worse, and you will likely fail the class. Just don't do it!

OK, that's all for the advice. Relax, take a deep breath, and good luck!

1.1 Question 1: Data exploration plots (10 points)

Consider the dataset below consisting of three input variables/featues and a single target variable:

[2]:
```
import numpy as np
```

```
X = np.array([[9.83092837e+00, 1.67509768e+01, 1.23855771e+01],
       [4.03614729e+00, 1.80169669e+01, 8.06272264e+00],
       [7.70145250e+00, 1.36710606e+01, 1.05755787e+01],
       [5.98117158e+00, 1.62830715e+01, 1.00402754e+01],
       [1.94291149e+00, 1.62641406e+01, 7.52260946e+00],
       [1.27635255e+00, 1.49446295e+01, 5.76041859e+00],
       [5.31574941e+00, 1.86027465e+01, 1.30434320e+01],
       [7.55070867e+00, 1.15022410e+01, 9.42081313e+00],
       [3.92820508e+00, 1.95483646e+01, 1.29023644e+01],
       [1.20427581e+00, 1.47983221e+01, 1.02299886e+01],
       [7.75314731e+00, 1.38830186e+01, 1.49441614e+01],
       [8.85235584e+00, 1.77079150e+01, 1.30544425e+01],
       [4.29234973e+00, 1.82022109e+01, 9.61365318e+00],
       [3.31364724e+00, 1.68188662e+01, 1.10725579e+01],
       [2.04684374e+00, 1.20017403e+01, 1.26607296e+01],
       [7.91508194e+00, 1.79939098e+01, 9.39757613e+00],
       [1.23484809e-01, 1.45218900e+01, 1.35420537e+01],
       [5.46922996e+00, 1.22967460e+01, 1.01504941e+01],
       [4.97022929e+00, 1.88868168e+01, 1.09789947e+01],
       [8.19988602e+00, 1.47238616e+01, 5.63969548e+00],
       [4.14897797e+00, 1.66984663e+01, 8.84492342e+00],
       [6.44725802e+00, 1.32299519e+01, 1.23707962e+01],
       [4.98194444e+00, 1.40451500e+01, 7.70572003e+00],
       [4.69950491e+00, 1.13108938e+01, 6.77282758e+00],
       [4.28696627e+00, 1.88956709e+01, 1.09697748e+01],
       [7.05651388e+00, 1.33809648e+01, 1.25978064e+01],
       [2.06401712e+00, 1.38374555e+01, 7.75850610e+00],
       [4.48241443e-01, 1.79835736e+01, 7.53627938e+00],
       [7.49374789e+00, 1.32325666e+01, 9.65612884e+00],
       [2.63376612e+00, 1.15261271e+01, 1.06653866e+01],
       [9.81054401e+00, 1.13463620e+01, 9.34722090e+00],
       [9.29502712e-01, 1.64592690e+01, 1.01194213e+01],
       [6.48761839e+00, 1.45603342e+01, 1.02997781e+01],
       [9.29203220e+00, 1.71729462e+01, 7.03638037e+00],
       [4.12295373e+00, 1.91500605e+01, 1.05667378e+01],
       [4.62980075e-01, 1.35574763e+01, 7.07920487e+00],
       [9.29550667e+00, 1.73016923e+01, 9.30620166e+00],
       [8.29721651e+00, 1.95354335e+01, 6.45636339e+00],
       [9.94808948e+00, 1.68477890e+01, 1.22861737e+01],
       [6.27792564e+00, 1.48149754e+01, 1.32384140e+01],
       [8.54398376e+00, 1.55892961e+01, 8.13192467e+00],
       [6.98114165e+00, 1.71852569e+01, 5.77530621e+00],
       [3.69789925e+00, 1.11596092e+01, 7.72993093e+00],
       [5.03790372e+00, 1.25968787e+01, 1.29494950e+01],
       [9.16043295e+00, 1.29479548e+01, 1.33775703e+01],
       [9.22137886e+00, 1.99688712e+01, 9.21916508e+00],
       [1.59494432e+00, 1.82294417e+01, 8.70491054e+00],
```

```
      [2.83879103e+00, 1.16692190e+01, 1.43313576e+01],
      [1.69539330e+00, 1.99738893e+01, 5.28089740e+00],
      [8.54822953e+00, 1.91223908e+01, 1.21291573e+01],
      [7.62680677e+00, 1.44945031e+01, 1.49301908e+01],
      [9.40783840e+00, 1.77026133e+01, 1.20622606e+01],
      [5.57981298e+00, 1.83281562e+01, 1.43050772e+01],
      [5.88081688e+00, 1.57629362e+01, 6.95646689e+00],
      [7.11495481e+00, 1.53450144e+01, 5.71493227e+00],
      [7.11593216e-01, 1.25905431e+01, 7.89798333e+00],
      [8.27120951e+00, 1.95090723e+01, 5.21464055e+00],
      [9.21290157e-01, 1.59734601e+01, 1.11409613e+01],
      [9.07056471e+00, 1.63279809e+01, 1.47734248e+01],
      [7.18794670e-03, 1.18291412e+01, 1.31903051e+01],
      [9.35941118e+00, 1.99433966e+01, 9.68693942e+00],
      [3.44442560e+00, 1.66530133e+01, 7.95495815e+00],
      [7.27551152e+00, 1.41576442e+01, 8.80797674e+00],
      [7.46221178e+00, 1.62362239e+01, 6.28911680e+00],
      [7.08661129e+00, 1.61626221e+01, 1.46363769e+01],
      [5.39287788e+00, 1.25220828e+01, 9.10842456e+00],
      [1.62769044e+00, 1.45564597e+01, 5.03433435e+00],
      [5.28596013e+00, 1.94825963e+01, 8.41390551e+00],
      [1.41639283e+00, 1.08884253e+01, 7.35718575e+00],
      [6.58341711e+00, 1.80852093e+01, 1.49904334e+01],
      [1.24227822e+00, 1.51047322e+01, 1.22796470e+01],
      [9.34331975e+00, 1.26719594e+01, 5.51627738e+00],
      [1.44709921e+00, 1.44004673e+01, 8.94329738e+00],
      [6.92853440e+00, 1.20741178e+01, 9.80557157e+00],
      [7.82234104e-01, 1.43396685e+01, 9.28063288e+00],
      [5.61610934e+00, 1.78530052e+01, 1.11637880e+01],
      [2.77708941e-01, 1.16429832e+01, 9.44624545e+00],
      [4.88877830e+00, 1.41157080e+01, 8.80317463e+00],
      [6.86978818e+00, 1.99038496e+01, 1.35418929e+01],
      [2.26069573e+00, 1.39243137e+01, 9.76417260e+00],
      [7.77110165e+00, 1.09336500e+01, 1.39962042e+01],
      [7.63407633e+00, 1.94623222e+01, 1.00322235e+01],
      [9.30474248e+00, 1.19691553e+01, 9.94788556e+00],
      [9.98939593e+00, 1.65537889e+01, 1.44286341e+01],
      [7.39657751e+00, 1.28079496e+01, 1.46583652e+01],
      [5.35387846e+00, 1.91660558e+01, 9.76650573e+00],
      [7.54721108e-01, 1.92017358e+01, 6.89080751e+00],
      [8.27335897e-01, 1.59047374e+01, 1.00299121e+01],
      [1.65250995e+00, 1.68476479e+01, 9.09680889e+00],
      [7.31438138e+00, 1.01438173e+01, 1.24529620e+01],
      [2.51801436e+00, 1.92847091e+01, 1.46632223e+01],
      [7.16495494e-01, 1.86992005e+01, 1.48722171e+01],
      [2.50312593e+00, 1.42556725e+01, 8.18741033e+00],
      [3.85304815e+00, 1.14408305e+01, 1.37686620e+01],
```

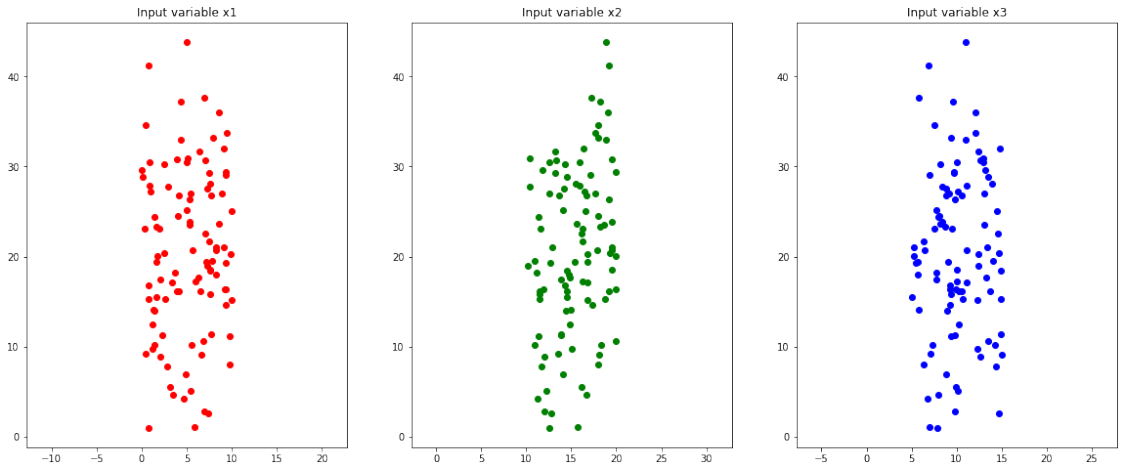
```
       [2.87673080e+00, 1.03567107e+01, 8.36238089e+00],
       [3.15067426e+00, 1.61821175e+01, 9.91867548e+00],
       [1.38909322e+00, 1.13796955e+01, 7.98078803e+00],
       [5.11005209e+00, 1.03587331e+01, 1.29151747e+01],
       [7.60555388e+00, 1.55084779e+01, 1.38989301e+01],
       [9.71567171e+00, 1.79458679e+01, 6.32041574e+00]])

y = np.array([[20.29896928],
       [24.47724934],
       [26.74708398],
       [17.2248154 ],
       [23.13286619],
       [14.09463833],
       [23.59515478],
       [15.84098025],
       [30.77113328],
       [12.50175531],
       [11.4195351 ],
       [27.04778259],
       [37.20803262],
       [17.14502953],
       [ 8.92537813],
       [33.22880958],
       [28.89227239],
       [ 5.09467182],
       [43.8649981 ],
       [18.04077426],
       [26.75304796],
       [31.71811405],
       [25.13329768],
       [ 4.26289467],
       [32.93627027],
       [30.71161576],
       [17.41308545],
       [34.64315345],
       [29.30098044],
       [15.28333795],
       [11.14603983],
       [27.17939011],
       [16.13118072],
       [29.08023432],
       [16.13820366],
       [ 9.23277024],
       [14.67358752],
       [20.69813466],
       [15.15638862],
       [17.66392457],
```

```
       [23.61569563],
       [37.60202046],
       [18.27759216],
       [30.50890892],
       [21.01374315],
       [16.33990792],
       [23.29450204],
       [ 7.80710697],
       [20.08179847],
       [36.04881458],
       [18.40114371],
       [33.75739846],
       [10.21408149],
       [ 1.06382363],
       [19.45829197],
       [ 0.96990427],
       [21.04848444],
       [27.84163775],
       [31.99162413],
       [29.65192432],
       [29.42621668],
       [ 4.62279871],
       [27.5623428 ],
       [21.69214303],
       [22.53534372],
       [26.97286588],
       [15.50295176],
       [23.85545761],
       [10.14939993],
       [ 9.14653525],
       [ 9.72070554],
       [19.32854681],
       [14.00123843],
       [ 2.85334935],
       [16.7990375 ],
       [20.70539768],
       [23.06167152],
       [ 6.98480062],
       [10.5859626 ],
       [11.33267849],
       [19.50723977],
       [18.54757964],
       [16.35452906],
       [25.08136229],
       [ 2.62156583],
       [26.31153527],
       [41.25559818],
```

```
       [30.44672428],
       [19.39257511],
       [18.96857452],
       [20.37584738],
       [15.29878455],
       [30.27918842],
       [16.13049868],
       [27.79965137],
       [ 5.48576703],
       [24.45576599],
       [30.94831078],
       [28.12926875],
       [ 8.05769326]])
```

In the cell below, write code to make three scatterplots, each showing an input variable x_j on the X axis and the target variable y on the Y axis.

[3]:
```
import matplotlib
from matplotlib import pyplot as plt

fig = plt.figure(figsize=(20,8))
ax = plt.subplot(1, 3, 1)
ax.plot(X[:,0],y[:,0],'ro')
plt.title('Input variable x1')
plt.axis('equal')
ax = plt.subplot(1, 3, 2)
ax.plot(X[:,1],y[:,0],'go')
plt.title('Input variable x2')
plt.axis('equal')
ax = plt.subplot(1, 3, 3)
ax.plot(X[:,2],y[:,0],'bo')
plt.title('Input variable x3')
plt.axis('equal')
plt.show()
```



From your graphs, which of the three input variables has the strongest relationship with the target variable?

Answer: It is a little difficult to tell, as there is a big spread in y for all three inputs, but it looks like x_2 has the strongest relationship with $y - y$ seems to increase the most with x_2.

1.2 Question 2: Linear regression model (10 points)

Use the normal equations to find the optimal linear regression model for the data in Question 1. Note that you do NOT need to split the data into training and test sets. Place your code below:

[4]:
```
Xa = np.insert(X, 0, 1, axis=1)
theta_opt = np.linalg.inv(Xa.T @ Xa) @ Xa.T @ y
print('Optimal parameters: %f, %f, %f, %f' % (theta_opt[0], theta_opt[1],␣
 ↪theta_opt[2], theta_opt[3]))
```

```
Optimal parameters: 4.320927, 0.032668, 0.940561, 0.144414
```

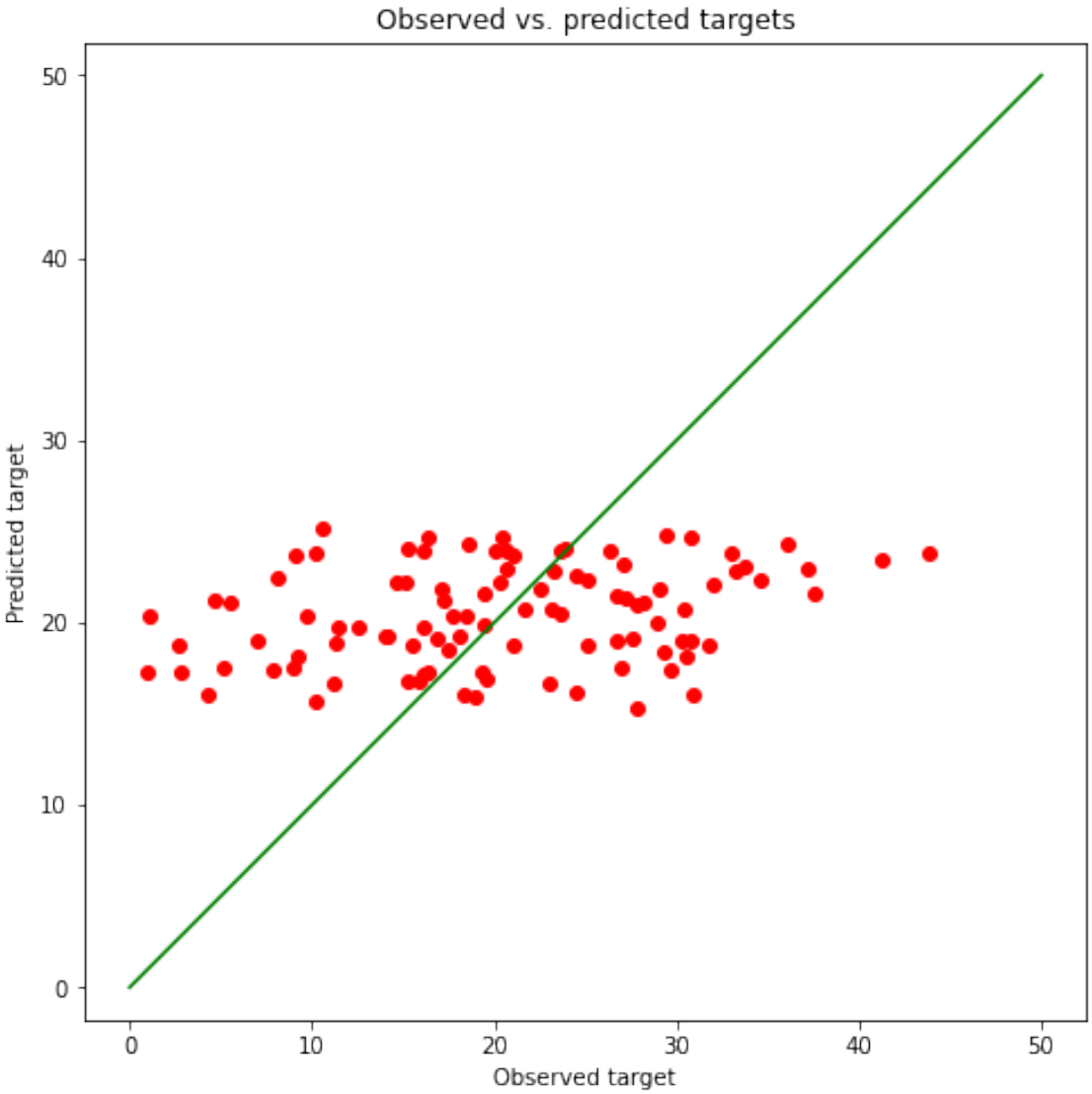
Explain the meaning of each of the parameters in the model you got. How is your answer to Question 1 about the input variable having the strongest relationship with the target evident in the set of parameters you got?

Answer: The first parameter, θ_0, is an intercept or constant term. It serves to shift the regression line to the mean of the data. The remaining three terms are the coefficients θ_1, θ_2, and θ_3 corresponding to the input variables x_1, x_2, and x_3. They give the strength of the contribution of each input variable in predicting the target. We saw earlier that x_2 had the strongest association with the target y, and indeed, we see that in the model, $\theta_2 = 0.94$ is the largest of the three coefficients corresonding to input variables, confirming that x_2 has the strongest relationship.

1.3 Question 3: Linear model prediction (10 points)

In the space below, provide a scatter plot plotting observed and predicted y values for the dataset. For each point in the dataset from Question 1, you should plot a point whose position on the X axis is the observed value of y and whose position on the Y axis is the the value of y predicted by your model ($h_\theta(\mathbf{x}^{(i)})$).

[5]:
```
y_pred = Xa @ theta_opt
fig = plt.figure(figsize=(8,8))
plt.plot(y, y_pred, 'ro')
plt.plot([0, 50], [0, 50], 'g-')
plt.xlabel('Observed target')
plt.ylabel('Predicted target')
plt.title('Observed vs. predicted targets')
plt.axis('equal')
plt.show()
```



1.4 Question 4: Cost function (10 points)

Assuming θ is the optimal parameter vector you obtained in Question 2, what is the value of the cost function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^{\top}\mathbf{x}^{(i)})^2?$$

[6]:
```
def J(theta, X, y):
    errs = y - X @ theta
    return 0.5 * errs.T @ errs

print('Cost J=%f' % J(theta_opt, Xa, y))
```

```
Cost J=4016.923044
```

Answer: The sum squared cost is 4017, which is pretty high!

If you changed the sum squared error cost function above to the mean squared error cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \theta^\top \mathbf{x}^{(i)})^2,$$

would you expect to get a different optimal parameter vector θ? Why or why not?

Answer: I would expect not, because the optimum occurs where $\frac{\partial J}{\partial \theta_j} = 0$ for all j. Multiplying J by any constant c will not change the optimum, as we would have $c\frac{\partial J}{\partial \theta_j} = 0$, which would be satisfied by the same θ.

If you changed the cost function to negative log likelihood under the assumption of Gaussian error, would you expect to get a different optimal parameter vector θ? Why or why not?

Answer: No, we should not expect any difference, as we already found (seel lecture notes) that the log likelihood is maximized by the same theta that minimizes squared error. Maximum likelihood under a Gaussian error assumption is equivalent to minimum squared error.

1.5 Question 5: Normalization (10 points)

If you were to normalize the input variables by subtracting their means and dividing by their standard deviations before estimating a linear regression model using the normal equations, would it affect the optimal parameter vector and value of the cost function? If so, how? If not, why not?

Answer (optimal parameter vector): It would change the optimal parameter vector. With just a single feature x_1, we can see that replacing it with $\hat{x}_1 = (x_1 - \mu_1)/\sigma_1$ means that to get the same hypothesis, we would need to have

$$\begin{aligned} h_{\hat{\theta}}(\mathbf{x}) &= \hat{\theta}_0 + \hat{\theta}_1 \hat{x}_1 && (1)\\ &= \hat{\theta}_0 + \hat{\theta}_1 (x_1 - \mu_1)/\sigma_1 && (2)\\ &= (\hat{\theta}_0 - \mu_1/\sigma_1) + (\hat{\theta}_1/\sigma_1) x_1 && (3) \end{aligned}$$

After substituting θ_0 and θ_1 for the constant term and coefficient of x_1, we see that $\hat{\theta}_0 = \theta_0 + \mu_1/\sigma_1$ and $\hat{\theta}_1 = \sigma_1\theta_1$.

Answer (cost function): It would not change the value of the cost function. As observed above, we obtain the same hypothesis even with the transformation of the input variables, and since the cost function only depends on the hypothesis values and observed target values, the cost function will be unchanged by the transformation.

1.6 Question 6: Classification dataset exploration plot (10 points)

Consider the dataset below consisting of two input features/variables and a Boolean (binary classification) target.

[7]:
```
X = np.array([[ 4.01433912e+00,  1.41968138e+01],
       [ 4.83561647e+00,  9.64522949e+00],
       [ 4.96629140e+00,  1.86333926e+01],
       [ 1.56738655e+00,  2.41698012e+01],
       [-1.08870754e+01,  3.49867260e+01],
       [-8.36109021e+00,  2.13485100e+01],
       [ 1.33123911e+01,  2.99762508e+01],
       [-3.62486684e+00,  2.53478211e+01],
       [ 5.93420611e+00,  1.66431345e+01],
       [-7.05281585e+00, -1.19838010e+00],
       [ 1.37966106e+01,  2.08842291e+01],
       [-2.90339542e+00,  1.46382186e+01],
       [ 1.01901590e+01,  2.21332053e+01],
       [ 6.46906160e+00,  2.78224849e+01],
       [ 1.10369419e+00,  2.22994701e+01],
       [-8.31526042e+00,  3.01137364e+01],
       [ 1.35519633e+01,  1.10275513e+01],
       [ 1.44183466e+01,  1.99870438e+00],
       [ 1.26741669e+01,  1.89593846e+01],
       [ 2.55429914e+01,  1.17008906e+01],
       [ 2.51593140e+01,  1.93079901e+01],
       [ 1.42722115e+01,  1.08806141e+01],
       [-3.08420891e+00,  1.63117053e+01],
       [-4.61424657e+00,  1.85142935e+01],
       [ 9.67601214e+00,  3.77090098e+00],
       [ 3.64884330e+00,  1.13306534e+01],
       [ 9.14690294e+00,  2.88084031e+01],
       [ 4.05941062e+00,  2.65877911e+01],
       [-2.79687217e+00,  1.15301774e+01],
       [ 2.15514629e-02,  2.91028777e+01],
       [-3.19788011e+00,  2.77633468e+01],
       [ 2.38113707e-01,  2.51189907e+01],
       [ 6.02347208e-01,  1.49391218e+01],
       [-1.27049129e+01,  3.13077790e+01],
       [-1.47673523e+00,  2.71266370e+01],
       [ 4.68246749e+00,  1.19225405e+01],
       [ 1.63578213e+01, -9.60719519e-01],
       [ 1.62476381e+00,  2.08291653e+01],
       [ 1.48188118e+01,  3.12940553e+01],
       [ 2.90835573e+00,  7.45858681e+00],
       [ 3.50703444e+00,  1.18861509e+01],
       [-6.11493760e+00,  1.73599954e+01],
       [ 3.35472301e+00,  3.07795965e+01],
       [ 6.23662160e+00,  2.53157756e+01],
       [ 3.02600812e+01,  1.44789320e+01],
       [-1.14717907e+00,  3.26481356e+01],
       [-4.08113245e+00,  3.39088088e+01],
```

```
       [ 9.61414050e+00,  3.19202977e+01],
       [ 5.42540459e+00, -1.11045056e+00],
       [ 1.06676734e+01,  1.40082251e+01],
       [ 1.30719407e+01,  3.52630724e+01],
       [ 7.27133455e+00,  3.62319289e+01],
       [ 2.92468400e+01,  5.02567534e+01],
       [-6.14992838e-01,  2.48522484e+01],
       [ 9.27998582e+00,  6.70597928e+00],
       [-1.40621635e+00, -3.11178512e-01],
       [-1.02720668e+01,  2.63573500e+01],
       [-2.70894672e+01,  7.08107370e+00],
       [ 2.49551569e+01,  1.88054177e+01],
       [-1.58813844e+00,  2.95710660e+00],
       [ 1.77468995e+01,  2.73084335e+01],
       [ 2.17715365e+01,  3.49823655e+01],
       [-2.60426318e-01,  2.05112098e+01],
       [ 7.49689584e+00,  2.65984714e+01],
       [ 1.08249922e+01,  2.30236886e+01],
       [-8.19324700e-01,  2.70146110e+00],
       [ 9.95483231e+00,  1.63738713e+01],
       [ 1.37958527e+01,  1.74233797e+01],
       [ 6.73724858e+00,  3.65761730e+01],
       [-1.25509489e+01,  1.95811860e+01],
       [ 3.32642994e+00,  2.20737851e+01],
       [-8.65237678e+00,  2.08807077e+01],
       [ 3.39613458e-01,  1.15784753e+01],
       [ 1.14869051e+01,  1.59071025e+01],
       [ 1.85948018e+01,  1.44967186e+01],
       [ 1.84332810e+01,  2.83099862e+01],
       [ 1.34633808e+01,  1.41673370e+01],
       [-1.69908227e+00,  6.75303841e+00],
       [-9.95568772e+00,  9.38570622e+00],
       [ 1.91465588e+00,  8.95449724e+00],
       [ 4.95914661e+00,  1.81001713e+01],
       [ 2.67147131e+01,  1.26037732e+01],
       [-9.09342338e+00,  3.70426756e+01],
       [ 3.54938413e+00,  1.48097147e+01],
       [-2.34064926e+01,  3.05828380e+01],
       [-2.44110013e+00,  1.78419041e+01],
       [ 3.19829731e+00,  1.37361794e+01],
       [ 1.05258867e+01,  2.09401469e+01],
       [ 2.19967319e+00,  1.79785061e+01],
       [-6.84740158e+00,  1.76894752e+01],
       [ 5.43156688e+00,  1.47748713e+01],
       [-8.49399418e+00,  8.75609059e+00],
       [ 3.78743399e+00,  1.90297680e+01],
       [ 6.21549380e+00,  2.67786293e+01],
```

```
       [ 1.78797840e+00,  3.69050059e+01],
       [ 1.11607521e+01,  1.09485168e+01],
       [ 5.57974253e+00,  8.58072306e+00],
       [ 9.96921202e+00,  2.69120552e+01],
       [ 2.25922359e+00,  2.47983130e+01],
       [-3.36878945e+00,  2.08206005e+01],
       [ 1.84625387e+01,  3.44210058e+00],
       [ 2.07066979e+01,  7.23583362e+00],
       [ 2.23181797e+01, -6.07536012e+00],
       [ 2.71948249e+01,  1.75266040e+00],
       [ 2.32810860e+01,  2.29892444e+00],
       [ 2.03047011e+01,  1.18016602e+01],
       [ 1.58162493e+01, -3.46316914e+00],
       [ 1.87815063e+01,  5.95277450e-01],
       [ 1.69219832e+01,  5.86649093e+00],
       [ 1.09121327e+01, -4.35188950e+00],
       [ 2.28830093e+01,  9.16374006e+00],
       [ 2.60747811e+01,  1.82067288e-01],
       [ 2.32471125e+01,  1.00382551e+01],
       [ 1.71806748e+01,  8.55343692e+00],
       [ 2.50807252e+01,  5.61903484e+00],
       [ 2.00590644e+01,  7.81510317e+00],
       [ 2.14684437e+01,  1.23885892e+01],
       [ 1.53579574e+01,  1.11065796e+00],
       [ 2.44389191e+01,  1.64536381e-02],
       [ 2.12250748e+01,  5.23085550e-01],
       [ 1.88126704e+01,  5.35177274e+00],
       [ 1.72505440e+01,  1.06034317e+01],
       [ 2.50799928e+01,  1.29212620e+01],
       [ 2.80373674e+01,  7.77412294e-01],
       [ 1.21687744e+01,  2.72942262e+00],
       [ 2.32391040e+01, -6.18057506e+00],
       [ 2.06614254e+01,  8.09625390e+00],
       [ 1.57301479e+01,  7.24119936e+00],
       [ 2.22308886e+01,  5.48650532e+00],
       [ 2.02979437e+01,  8.57777106e+00],
       [ 1.92051876e+01,  2.44911204e+00],
       [ 1.83044229e+01,  1.71192903e+00],
       [ 1.56397565e+01,  3.57785030e+00],
       [ 2.39013728e+01,  6.05944237e+00],
       [ 1.96455217e+01,  9.78419211e+00],
       [ 2.20182637e+01,  4.65533477e+00],
       [ 2.24978681e+01,  2.20718718e+00],
       [ 2.22543088e+01,  1.02295127e+01],
       [ 1.31849911e+01,  7.16882296e+00],
       [ 2.12575235e+01,  6.27694459e+00],
       [ 1.70150147e+01,  3.60866781e+00],
```

```
       [ 2.18756895e+01,  6.60779268e+00],
       [ 2.21163547e+01,  2.49386620e+00],
       [ 1.54524774e+01, -7.13755001e+00],
       [ 2.54065358e+01,  1.82095762e+00],
       [ 1.81289296e+01,  2.86813742e+00],
       [ 2.05919778e+01,  1.54545650e+01],
       [ 1.14931704e+01,  4.65673839e+00],
       [ 2.46033947e+01,  7.70031528e+00],
       [ 1.38665840e+01,  3.25975684e+00],
       [ 2.71011177e+01, -8.72894726e+00],
       [ 1.86525250e+01,  8.84535854e+00],
       [ 2.35651899e+01, -5.21462679e+00],
       [ 2.10724831e+01, -5.43661781e+00],
       [ 2.75946155e+01,  1.69743616e+00],
       [ 1.06112171e+01,  7.89890142e+00],
       [ 1.50597592e+01, -7.47718137e+00],
       [ 2.35906380e+01,  2.44435817e+00],
       [ 2.02998810e+01,  9.33960978e+00],
       [ 1.90532650e+01,  5.27580771e+00],
       [ 2.04598935e+01,  6.30007880e+00],
       [ 1.68069589e+01,  5.46138135e+00],
       [ 1.50621834e+01,  1.00914500e+01],
       [ 2.74143491e+01, -1.03960109e+00],
       [ 1.99750841e+01,  2.01247506e+00],
       [ 1.59144752e+01, -6.50091210e-01],
       [ 1.95640520e+01,  7.36635146e+00],
       [ 2.71253115e+01,  5.64373695e+00],
       [ 1.92951146e+01,  5.76347472e+00],
       [ 2.45597575e+01,  5.62731737e+00],
       [ 2.11337256e+01,  1.15127579e+01],
       [ 2.22332220e+01,  4.42877548e+00],
       [ 1.83017952e+01,  8.12710908e+00],
       [ 2.01120729e+01,  4.92978157e+00],
       [ 8.74330074e+00, -7.94490715e-01],
       [ 2.00540992e+01,  1.71930972e+00],
       [ 5.16805892e+00, -2.54757582e+00],
       [ 2.87164908e+01,  4.27709942e-01],
       [ 2.48698486e+01,  1.87081221e+00],
       [ 2.02794328e+01,  4.35024129e+00],
       [ 2.08219329e+01,  2.78067569e+00],
       [ 8.72136806e+00,  1.33872193e+01],
       [ 1.84360871e+01,  1.08154727e+01],
       [ 1.78071157e+01, -7.43466330e-01],
       [ 1.65860919e+01,  1.09916697e+01],
       [ 1.83278552e+01,  7.24234833e-01],
       [ 1.70434511e+01,  4.68472900e+00],
       [ 1.95376056e+01,  4.65159995e+00],
```

```
       [ 1.57068086e+01,  3.77701296e+00],
       [ 1.50697915e+01,  6.82645204e+00],
       [ 2.22384588e+01, -1.52789633e-01],
       [ 2.12540901e+01,  1.31779245e+01],
       [ 2.43768043e+01,  2.08216510e+00],
       [ 1.76616073e+01,  1.92644403e+00],
       [ 1.47497977e+01,  1.56867443e+01],
       [ 1.84581498e+01,  5.98989507e+00],
       [ 3.25083813e+01,  1.25781150e+01],
       [ 2.76047122e+01,  6.05211580e+00],
       [ 2.86386135e+01,  2.89452070e+00],
       [ 1.47660078e+01,  1.59544804e+01]])

y = np.array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
```

```
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
```

```
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
```

```
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
```

```
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

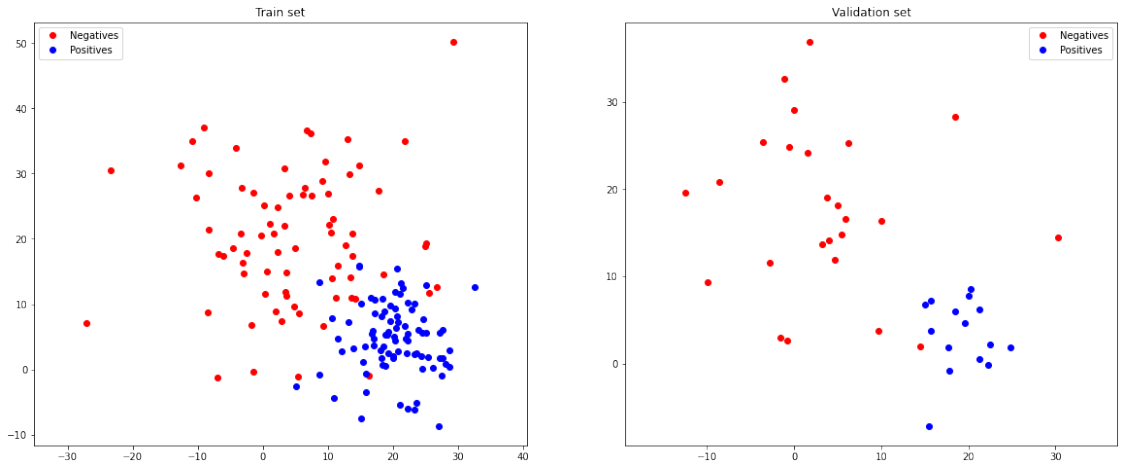
In the space below, write code to split the dataset into separate training and validation datasets in an 80%/20% ratio, then provide two scatter plots, one for the training set and one for the validation set, in which the positive and negative examples are plotted in different colors.

[8]:
```
m, n = X.shape
m_train = m * 8 // 10
m_test = m - m_train
indices = np.random.permutation(m)
train_indices = indices[:m_train]
val_indices = indices[m_train:]
X_train = X[train_indices,:]
y_train = y[train_indices,:]
X_val = X[val_indices,:]
y_val = y[val_indices,:]
```

[9]:
```
fig = plt.figure(figsize=(20,8))
ax = plt.subplot(1, 2, 1)
X_train_neg = X_train[y_train.squeeze()==0,:]
X_train_pos = X_train[y_train.squeeze()==1,:]
ax.plot(X_train_neg[:,0],X_train_neg[:,1],'ro', label='Negatives')
ax.plot(X_train_pos[:,0],X_train_pos[:,1],'bo', label='Positives')
```

```
plt.title('Train set')
plt.axis('equal')
plt.legend()
ax = plt.subplot(1, 2, 2)
X_val_neg = X_val[y_val.squeeze()==0,:]
X_val_pos = X_val[y_val.squeeze()==1,:]
ax.plot(X_val_neg[:,0],X_val_neg[:,1],'ro', label='Negatives')
ax.plot(X_val_pos[:,0],X_val_pos[:,1],'bo', label='Positives')
plt.title('Validation set')
plt.axis('equal')
plt.legend()
plt.show
```

[9]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



1.7 Question 7: Logistic regression model (10 points)

In the space below, write code to train a logistic regression model on the training set.

[10]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

theta = np.zeros((3, 1))
Xa_train = np.insert(X_train, 0, 1, axis=1)
alpha = 0.1
initial_cost = None
epochs = 50
for epoch in range(epochs):
    y_pred = sigmoid(Xa_train @ theta)
    cost = - (np.multiply(y_train, np.log(y_pred)) + np.multiply(1 - y_train,␣
 ↪np.log(1 - y_pred))).mean()
```

```
    if initial_cost is None:
        initial_cost = cost
    theta = theta + alpha * np.mean((y_train - y_pred) * Xa_train, axis=0,␣
↪keepdims=True).T

print('Cost reduced from %f to %f in %d epochs.' % (initial_cost, cost, epochs))
print('Optimal theta: %f, %f, %f' % (theta[0], theta[1], theta[2]))
```

```
Cost reduced from 0.693147 to 0.219830 in 50 epochs.
Optimal theta: -0.030912, 0.222309, -0.343737
```

Give the values of the parameters in the optimal parameter vector you got, and explain the meaning of each of the parameters.

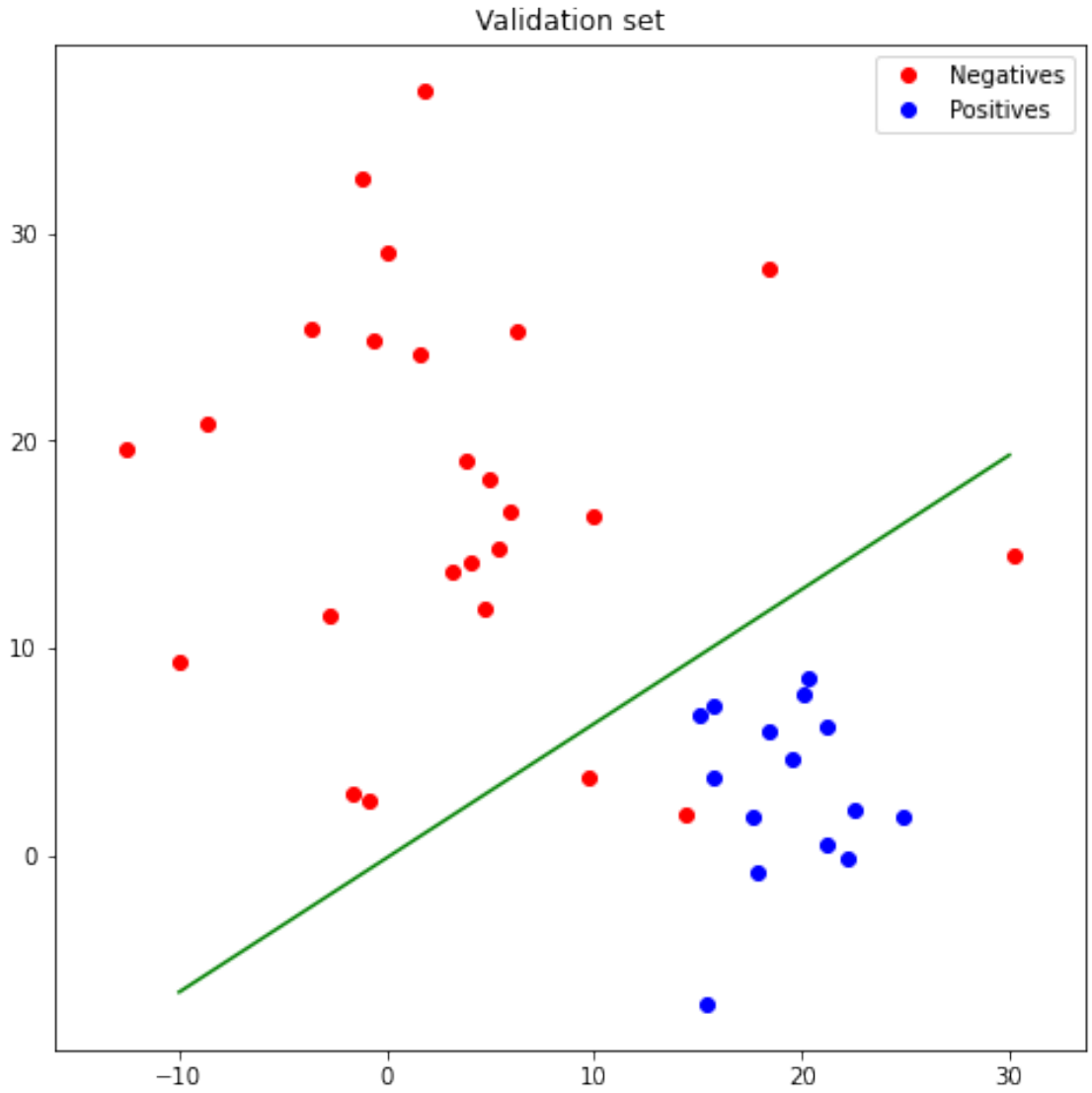
Answer: the optimal values of -0.039863, 0.250906, and -0.334765 represent the scaled signed distance of the origin from the decision boundary (the first term, i.e., θ_0) and the decision boundary's normal vector orientation (the second two terms, i.e., θ_1 and θ_2). If we normalize by $\|[\theta_1, \theta_2]\|$ we get -0.09528496, 0.59974335, -0.80019243, indicating that the origin is at a Euclidean distance of 0.095 units behind (to the left of/above) the decision boundary.

1.8 Question 8: Logistic regression decision boundary (10 points)

Provide a plot of the validation dataset from Question 6 with the positive and negative examples plotted in different colors, along with the optimal decision boundary obtained in Question 7.

[11]:
```
fig = plt.figure(figsize=(8,8))
plt.plot(X_val_neg[:,0],X_val_neg[:,1],'ro', label='Negatives')
plt.plot(X_val_pos[:,0],X_val_pos[:,1],'bo', label='Positives')
x1 = -10
y1 = - (theta[0] + theta[1] * x1) / theta[2]
x2 = 30
y2 = - (theta[0] + theta[1] * x2) / theta[2]
plt.plot([x1, x2], [y1, y2], 'g-')
plt.title('Validation set')
plt.axis('equal')
plt.legend()
plt.show
```

[11]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



Do you have reason to believe that a SVM could obtain a better decision boundary than the one you got with the logistic regression? Why or why not? If your answer is yes, indicate what type of SVM you would use and what hyperparameters would be involved.

Answer: It should be possible to get a better decision boundary using a SVM with a nonlinear kernel such as the Gaussian kernel and an appropriate setting for the hyperparameters C and σ (or γ). However, in this specific layout of the validation set, it is not clear that we would get better classification accuracy, as there are two negatives fairly deep in the positive region. A different random split may have benefitted more from a nonlinear kernel.

1.9 Question 9: SVM with Gaussian kernel (10 points)

Write code to construct a SVM with the Gaussian kernel and slack variables for the dataset in Question 6. Use the quadratic programming solution to the dual optimization problem with cvxopt. Attempt solution with two different values of the hyperparameters. For each solution, give the validation set accuracy and number of support vectors found.

[12]:
```
!https_proxy=http://192.41.170.23:3128 http_proxy=http://192.41.170.23:3128 pip␣
↪install cvxopt
#!pip install cvxopt
import cvxopt
```

```
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Collecting cvxopt
  Downloading
cvxopt-1.2.7-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.0 MB)
     |                  | 13.0 MB 10.4 MB/s eta 0:00:01
Installing collected packages: cvxopt
Successfully installed cvxopt-1.2.7
```

[27]:
```
def cvxopt_solve_qp(Q, c, A=None, b=None, E=None, d=None):
    # Fill your code value in 'None'
    # Some 'None' can be avoided.
    Q_new = 0.5 * (Q + Q.T)
    args = [cvxopt.matrix(Q_new), cvxopt.matrix(c) ]
    if A is not None:
        args.extend([cvxopt.matrix(A), cvxopt.matrix(b)])
        if E is not None:
            args.extend([cvxopt.matrix(E), cvxopt.matrix(d)])
    sol = cvxopt.solvers.qp(*args)
    if sol is not None and 'optimal' not in sol['status']:
        return None
    x = np.array(sol['x']).reshape((Q.shape[0], ))
    return x

def gauss_kernel(X, sigma=0.2):
    m = X.shape[0];
    K = np.matrix(np.zeros([m,m]));
    for i in range(0,m):
        for j in range(0,m):
            K[i,j] = (X[i,:] - X[j,:]).reshape(1,-1) @ (X[i,:] - X[j,:]).
↪reshape(-1,1)
    K = np.exp(-K/(2*sigma*sigma))
    return K

def svm_predict(x, X, y, alpha, sigma):
    s = []
    b = 0
```

```
    for i in range(X.shape[0]):
        b += y[i]
        for j in range(X.shape[0]):
            b -= alpha[j]*y[j]*np.exp((-(X[i,:]-X[j,:])@((X[i,:]-X[j,:]).T))/
↪(2*sigma*sigma))
    b /= X.shape[0]
    for j in range(x.shape[0]):
        ss = 0
        for i in range(X.shape[0]):
            ss += alpha[i]*y[i]*np.exp((-(X[i,:]-x[j,:])@((X[i,:]-x[j,:]).T))/
↪(2*sigma*sigma))
        s.append(ss)
    s = np.array(s) + b
    s[s >= 0] = 1
    s[s < 0] = -1
    return s

# Convert 0/1 targets to -1/+1
y_train_svm = y_train
y_train_svm[y_train==0] = -1
y_val_svm = y_val
y_val_svm[y_val==0] = -1

def svm_train(X, y, C, sigma):
    m = X.shape[0]
    ydiag = np.diag(np.array(y).squeeze())
    K = gauss_kernel(X_train, sigma=sigma)
    Q = ydiag @ K @ ydiag
    c = -np.ones((m, 1))
    A1 = -np.eye(m)
    b1 = np.zeros((m, 1))
    A2 = np.eye(m)
    b2 = np.ones((m, 1)) * C
    A = np.concatenate((A1, A2), 0)
    b = np.concatenate((b1, b2), 0)
    E = y_train_svm.T
    d = np.zeros(1)
    alpha_star = cvxopt_solve_qp(Q, c, A, b, E, d)
    return alpha_star

# Hyperparameter set 1: C=1, sigma=1

C1 = 1
sigma1 = 1
cvxopt.solvers.options['show_progress'] = True
alpha1 = svm_train(X_train, y_train_svm, C1, sigma1)
y_pred1 = svm_predict(X_val, X_train, y_train_svm, alpha1, sigma1)
```

```
acc1 = np.sum(y_pred1 == y_val_svm) / len(y_val_svm)
print('Accuracy with C=1, sigma=1:', acc1)

# Here's how to do a hyperparameter search in a 21x21 grid
#
# cvxopt.solvers.options['show_progress'] = False
# for i in range(-10, 11):
#     C = 2**i
#     for j in range(-10, 11):
#         sigma = 2**j
#         alpha = svm_train(X_train, y_train_svm, C, sigma)
#         y_pred = svm_predict(X_val, X_train, y_train_svm, alpha, sigma)
#         acc = np.sum(y_pred == y_val_svm) / len(y_val_svm)
#         print('C = %f, sigma = %f, acc = %f' % (C, sigma, acc))

# Best accuracy (1 error) occurred with C=8, sigma=2

C2 = 8
sigma2 = 2
alpha2 = svm_train(X_train, y_train_svm, C2, sigma2)
y_pred2 = svm_predict(X_val, X_train, y_train_svm, alpha2, sigma2)
acc2 = np.sum(y_pred2 == y_val_svm) / len(y_val_svm)
print('Accuracy with C=8, sigma=2:', acc2)
```

```
     pcost       dcost       gap    pres   dres
 0: -4.9525e+01 -2.7346e+02  2e+02  3e-15  5e-16
 1: -5.1042e+01 -6.5393e+01  1e+01  3e-16  3e-16
 2: -5.2725e+01 -5.4827e+01  2e+00  2e-16  1e-16
 3: -5.3070e+01 -5.3352e+01  3e-01  1e-15  1e-16
 4: -5.3122e+01 -5.3151e+01  3e-02  8e-16  1e-16
 5: -5.3127e+01 -5.3130e+01  3e-03  2e-15  1e-16
 6: -5.3128e+01 -5.3128e+01  4e-05  2e-15  2e-16
Optimal solution found.
Accuracy with C=1, sigma=1: 0.925
     pcost       dcost       gap    pres   dres
 0:  3.2926e+02 -3.6955e+03  4e+03  5e-14  2e-15
 1:  2.5091e+01 -4.1083e+02  4e+02  9e-15  2e-15
 2: -5.3047e+01 -1.3681e+02  8e+01  1e-14  8e-16
 3: -6.4615e+01 -7.9523e+01  1e+01  9e-16  4e-16
 4: -6.6559e+01 -6.8599e+01  2e+00  2e-16  4e-16
 5: -6.6857e+01 -6.7285e+01  4e-01  2e-15  4e-16
 6: -6.6925e+01 -6.7005e+01  8e-02  2e-15  4e-16
 7: -6.6940e+01 -6.6950e+01  1e-02  4e-15  4e-16
 8: -6.6942e+01 -6.6943e+01  7e-04  9e-16  5e-16
 9: -6.6942e+01 -6.6942e+01  2e-05  2e-16  5e-16
Optimal solution found.
Accuracy with C=8, sigma=2: 0.975
```

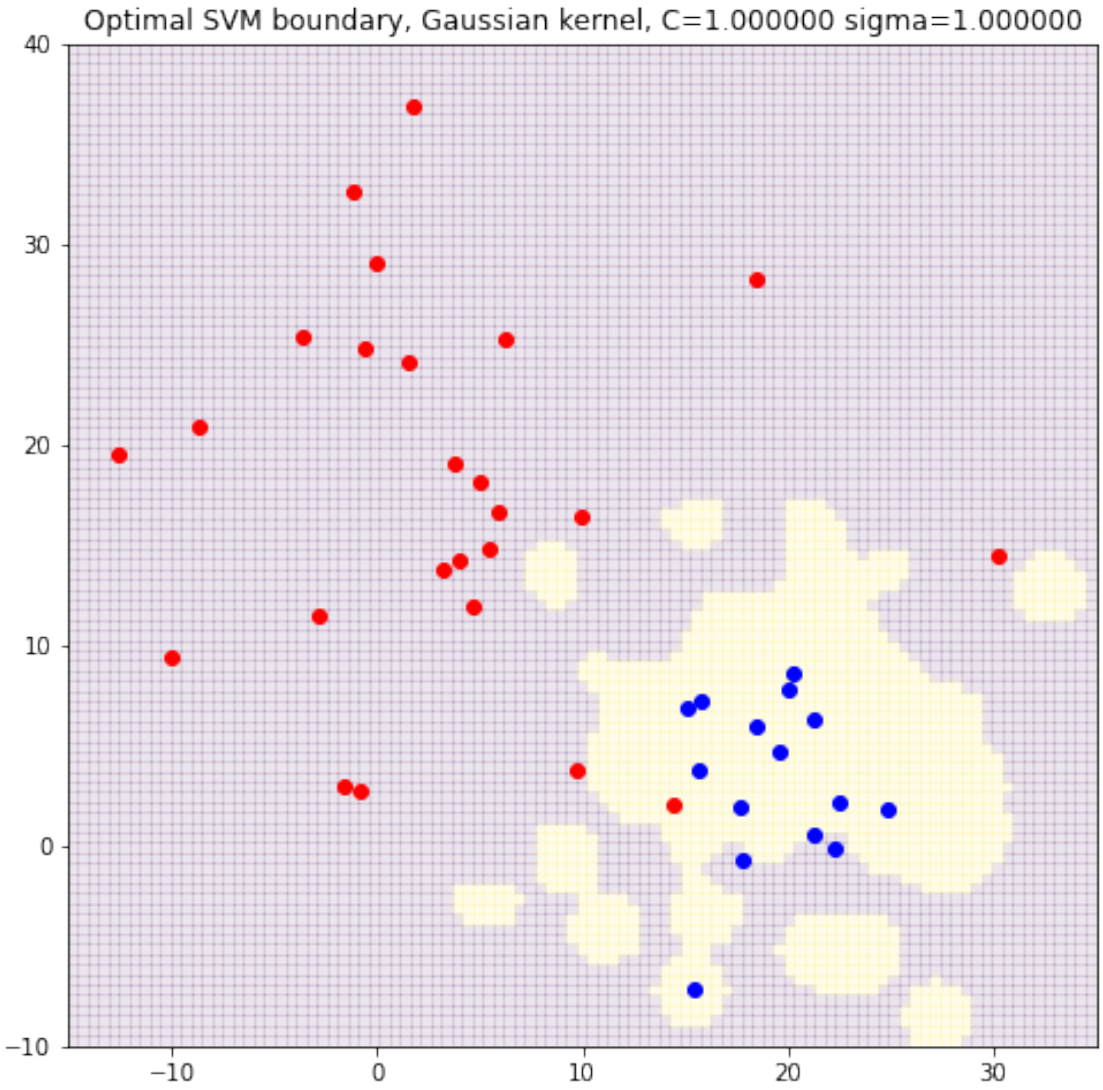
1.10 Question 10: SVM decision boundaries (10 points)

For each of the two solutions you obtained in Question 9, show the validation set with the decision boundary under the optimal parameters.

[33]:
```
def plot_svm_boundary(X_train, y_train, X_val, y_val, alpha, sigma, title):
    x_series = np.linspace(-15, 35, 100)
    y_series = np.linspace(-10, 40, 100)

    x_mesh, y_mesh = np.meshgrid(x_series, y_series)

    x_mesh = x_mesh.reshape(-1, 1)
    y_mesh = y_mesh.reshape(-1, 1)

    mesh = np.append(x_mesh, y_mesh, axis=1)
    y_pred = predict(mesh, X_train, y_train, alpha, sigma)

    x_mesh = x_mesh.reshape(100, 100)
    y_mesh = y_mesh.reshape(100, 100)
    y_pred = y_pred.reshape(100, 100)

    fig1 = plt.figure(figsize=(8,8))
    X_val_neg = X_val[y_val.squeeze()==-1]
    X_val_pos = X_val[y_val.squeeze()==1]
    plt.plot(X_val_neg[:,0], X_val_neg[:,1], 'ro', label='Negatives')
    plt.plot(X_val_pos[:,0], X_val_pos[:,1], 'bo', label='Positives')
    plt.title(title)
    plt.pcolormesh(x_mesh, y_mesh, y_pred, cmap='viridis', shading='auto',␣
 ↪alpha=0.1)
    plt.show()
```

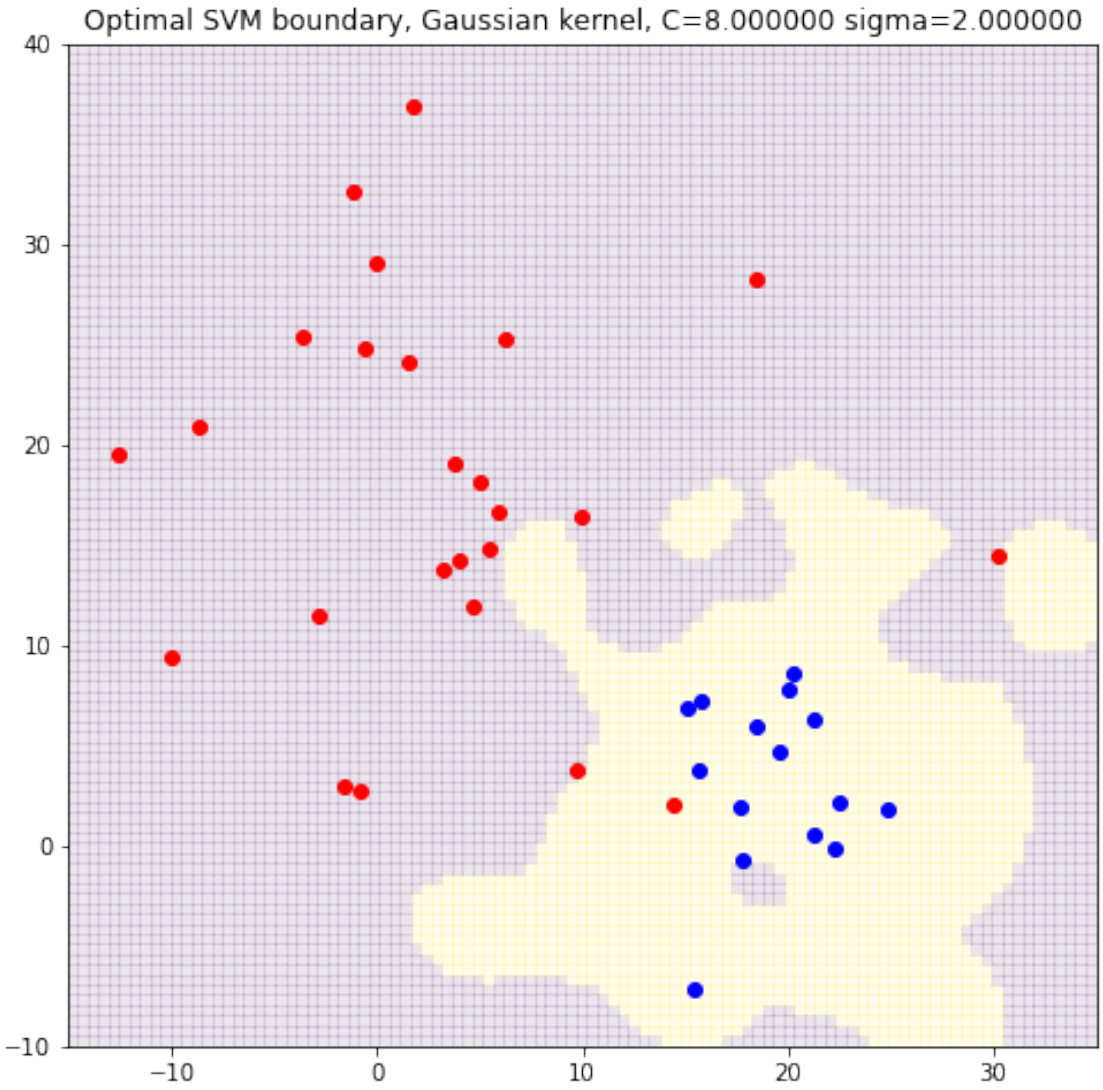
[34]:
```
plot_svm_boundary(X_train, y_train_svm, X_val, y_val_svm, alpha1, sigma1,␣
 ↪'Optimal SVM boundary, Gaussian kernel, C=%f sigma=%f' % (C1, sigma1))
```

```
[-0.20398441]
[[-0.20398441]
 [-0.20398441]
 [-0.20398441]
 …
 [-0.20398441]
 [-0.20398441]
 [-0.20398441]]
```



[35]:
```
plot_svm_boundary(X_train, y_train_svm, X_val, y_val_svm, alpha2, sigma2,␣
↪'Optimal SVM boundary, Gaussian kernel, C=%f sigma=%f' % (C2, sigma2))
```

```
[-0.34714056]
[[-0.34714058]
 [-0.34714061]
 [-0.34714067]
 …
 [-0.34714065]
 [-0.34714061]
 [-0.34714059]]
```



In conclusion, which solution is more appropriate according to validation set accuracy and the boundaries you observe?

Answer: We can see that both boundaries are highly irregular, trying to pull in the positive (blue) training set examples that overlapped with the negative (red) examples. However, the second set of hyperparameters $C = 8$, $\sigma = 2$ is more smooth and obtains better validation set accuracy. We could look further for a hyperparameter setting that didn't lead to those amoebic arms or the red-classified hole in the middle of the blue region.

[ ]: