Lab 07: Support Vector Machines

Today we'll look at the SVM maximum margin classification problem and how we can implement the optimization in Python.

We'll use the cvxopt quadratic programming optimizer in Python.

Later in the lectures, we'll see that more specialized algorithms such as Sequential Minimal Optimization implemented by the machine learning libraries are more effective for large SVM problems.

Linearly separable case: direct solution using quadratic programming

If we assume that the data are linearly separable, we can use the following setup for the optimization:

- The data are pairs $(\mathbf{x}^{(i)}, y^{(i)})$ with $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and $y^{(i)} \in \{-1, 1\}$.
- The hypothesis is
$$h_{\mathbf{w},b}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$
- The objective function is
$$\mathbf{w}^*, b^* = \text{argmax}_{\mathbf{w},b} \gamma,$$
where γ is the minimum geometric margin for the training data:
$$\gamma = \min_i \gamma^{(i)}$$
and $\gamma^{(i)}$ is the geometric margin for training example i, i.e., the signed distance of $\mathbf{x}^{(i)}$ from the decision boundary, with positive distances indicating that the point is on the correct side of the boundary and negative distances indicating that the point is on the incorrect side of the boundary:
$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \right)^\top \mathbf{x}^{(i)} + \frac{b}{\|\mathbf{w}\|} \right).$$
- To find the optimal $\mathbf{w}, b$ according to the objective function above, we can solve the constrained optimization problem $$ \begin{array}{rl} \min_{\mathbf{w},b} & |\mathbf{w}| \

```
                \text{subject to} & y^{(i)}(\mathbf{w}^\top\mathbf
{x}^{(i)}+b)\ge 1, i \in 1..m
```

\end{array} $$

In [41]:
```
# in case of there is not cvxopt
!pip install cvxopt
```

```
Requirement already satisfied: cvxopt in /home/alisa/anaconda3/li
b/python3.8/site-packages (1.2.7)
```

In [42]:
```
import numpy as np
import matplotlib.pyplot as plt
import cvxopt
```

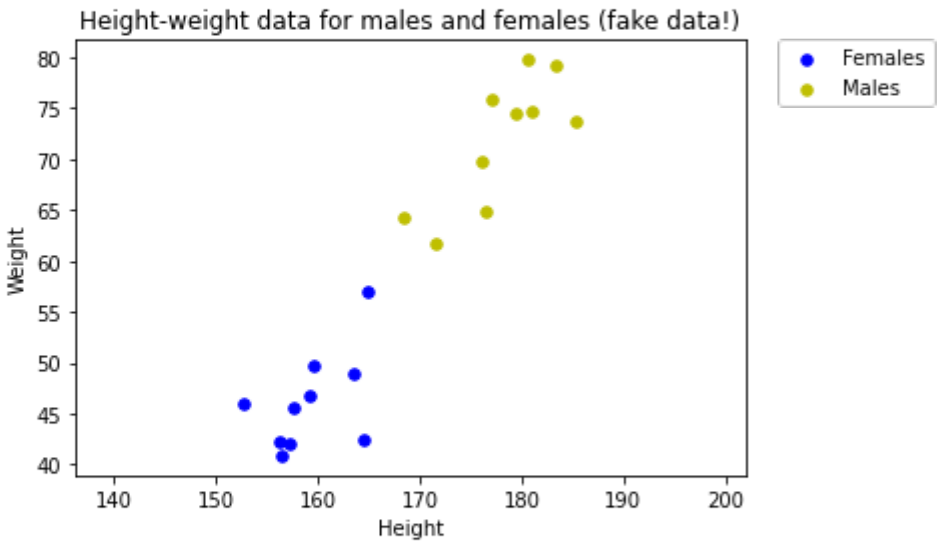
The example data

In [43]:
```
Xf = np.matrix([[ 164.939, 163.431, 157.554, 152.785, 156.385, 159.242, 156.281, 164.411, 157.308, 159.579 ],
                [  56.927,  48.945,  45.678,  45.969,  40.896,  46.848,  42.225,  42.380,  42.150,  49.739 ]]).T;
Xm = np.matrix([[ 168.524, 171.597, 179.469, 176.063, 180.939, 177.011, 183.284, 180.549, 176.502, 185.392 ],
                [  64.353,  61.793,  74.552,  69.851,  74.730,  75.871,  79.170,  79.753,  64.923,  73.665 ]]).T;
X = np.concatenate([Xf, Xm],0);
y = np.concatenate([-np.matrix(np.ones([10,1])),np.matrix(np.ones([10,1]))]);
```

In [44]:
```
print(X.shape)
print(y.shape)
```

```
(20, 2)
(20, 1)
```

In [45]:
```
# Plot the data
def plot_mf(Xf,Xm):
    axes = plt.axes()
    females_series = plt.scatter(np.array(Xf[:,0]), np.array(Xf
[:,1]), s=30, c='b', marker='o', label='Females')
    males_series = plt.scatter(np.array(Xm[:,0]), np.array(Xm[:,
1]), s=30, c='y', marker='o', label='Males')
    axes.set_aspect('equal', 'datalim')
    plt.xlabel('Height')
    plt.ylabel('Weight')
    plt.title('Height-weight data for males and females (fake dat
a!)')
    plt.legend(handles=[females_series, males_series], bbox_to_an
chor=(1.05, 1), loc=2, borderaxespad=0.)
    return axes

def plot_w(w,b):
    ylim = plt.axes().get_ylim()
    xlim = plt.axes().get_xlim()
    p1 = (xlim[0], - (w[0,0] * xlim[0] + b) / w[1,0])
    p2 = (xlim[1], - (w[0,0] * xlim[1] + b) / w[1,0])
    plt.plot((p1[0],p2[0]), (p1[1],p2[1]), 'r-')

plot_mf(Xf,Xm)
plt.show()
```



Exercise 1 (in lab): linearly separable data (Total 25 points)

Take the example data and SVM optimization code using cvxopt from the exercise in lecture. Verify that you can find the decision boundary for such "easy" cases. Show your results in your lab report.

Exercise 1.1 Create SVM function using cvopt (5 points)

In [46]:
```
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 = None
    args = [None, None]
    if A is not None:
        args.extend([None, None])
        if E is not None:
            args.extend([None, None])
    sol = None
    if sol is not None and 'optimal' not in sol['status']:
        return None
    x = None
    ### BEGIN SOLUTION
    Q_new = .5 * (Q + Q.T)  # make sure Q is symmetric
    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 'optimal' not in sol['status']:
        return None
    x = np.array(sol['x']).reshape((Q.shape[1],))
    ### END SOLUTION
    return x
```

In [47]:
```
# Test function: Do not remove
Q_test = np.array([[1, 0],[0, 0]])
c_test = np.zeros([2])
A_test = np.array([[15., 1.],[14., 1.], [13., 1], [-5.,-1.],[-
6.,-1.],[-8,-1]])
B_test = -np.ones([6])
try:
    x_test = cvxopt_solve_qp(Q_test, c_test, A_test, B_test)
    print('x_test:', x_test)
except:
    assert False, "cvxopt_solve_qp is incorrect"

print("success!")
# End Test function
```

```
     pcost       dcost       gap    pres   dres
 0:  2.8800e-02  1.0464e+00  1e+01  1e+00  3e+01
 1:  1.8859e-01 -8.5852e-01  1e+00  2e-01  3e+00
 2:  1.6523e-01  3.7852e-02  1e-01  4e-16  4e-16
 3:  8.2300e-02  7.2465e-02  1e-02  6e-16  3e-15
 4:  8.0043e-02  7.9915e-02  1e-04  7e-16  2e-16
 5:  8.0000e-02  7.9999e-02  1e-06  2e-16  1e-15
 6:  8.0000e-02  8.0000e-02  1e-08  8e-16  1e-15
Optimal solution found.
x_test: [-0.40000001  4.20000012]
success!
```

Expect result (or look-alike):\ pcost dcost gap pres dres\ 0: 2.8800e-02 1.0464e+00 1e+01 1e+00 3e+01\ 1: 1.8859e-01 -8.5852e-01 1e+00 2e-01 3e+00\ 2: 1.6523e-01 3.7852e-02 1e-01 4e-16 4e-16\ 3: 8.2300e-02 7.2465e-02 1e-02 6e-16 3e-15\ 4: 8.0043e-02 7.9915e-02 1e-04 7e-16 2e-16\ 5: 8.0000e-02 7.9999e-02 1e-06 2e-16 1e-15\ 6: 8.0000e-02 8.0000e-02 1e-08 8e-16 1e-15\ Optimal solution found.\ x_test: [-0.40000001 4.20000012]\

Exercise 1.2: Find Q, c, A, B for input into cvxopt_solve_qp function (10 points)

In [48]:

```
Q = None
c = None
A = None
B = None
### BEGIN SOLUTION
Q = np.eye(3);
Q[2,2] = 0;
c = np.zeros([3])
A = np.multiply(np.tile(-y,[1, 3]), np.concatenate([X, np.ones([2
0,1])],1))
B = -np.ones([20])
### END SOLUTION

x = cvxopt_solve_qp(Q, c, A, B)
```

```
     pcost       dcost       gap    pres   dres
 0:  1.4721e-03  6.5053e+00  5e+01  2e+00  4e+02
 1:  1.0012e-02 -4.7161e+00  1e+01  6e-01  1e+02
 2:  2.6180e-02 -4.8172e+00  7e+00  2e-01  6e+01
 3:  3.9767e-02 -4.5363e-01  5e-01  1e-02  2e+00
 4:  3.5404e-02  1.8200e-02  2e-02  4e-15  2e-13
 5:  3.1392e-02  3.0877e-02  5e-04  6e-15  2e-12
 6:  3.1250e-02  3.1245e-02  5e-06  4e-15  6e-13
 7:  3.1249e-02  3.1248e-02  5e-08  4e-15  2e-13
Optimal solution found.
```

In [49]:
```
print('Q:\n', Q)
print('c:\n', c)
print('A:\n', A[7:13])
print('B:\n', B)
print('x:\n', x)
# Test function: Do not remove
assert Q.shape == (3, 3) and Q[2,2] == Q[0,1] and Q[2,0] == 0 and Q[0,0] == Q[1,1] and Q[0,0] == 1, 'Q value is incorrect'
assert c.shape == 3 or c.shape == (3,) or c.shape == (3,1), 'Size of c is incorrect'
assert np.all((c == 0)), 'c value is incorrect'
assert A.shape == (20,3), 'Size of A is incorrect'
assert np.max(A[:,2]) == 1 and np.min(A[:,2]) == -1, 'A value is incorrect'
assert not np.array_equal(np.round(A[:,0:2],1), np.round(X,1)), 'A value is incorrect'
assert np.array_equal(np.round(x,1), np.round([0.16001143, 0.19207647, -38.32646165],1)), 'x value is incorrect'

print("success!")
# End Test function
```

```
Q:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 0.]]
c:
 [0. 0. 0.]
A:
 [[ 164.411   42.38     1.   ]
 [ 157.308   42.15     1.   ]
 [ 159.579   49.739    1.   ]
 [-168.524  -64.353   -1.   ]
 [-171.597  -61.793   -1.   ]
 [-179.469  -74.552   -1.   ]]
B:
 [-1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
-1. -1.
 -1. -1.]
x:
 [  0.16001143   0.19207647 -38.32646165]
success!
```

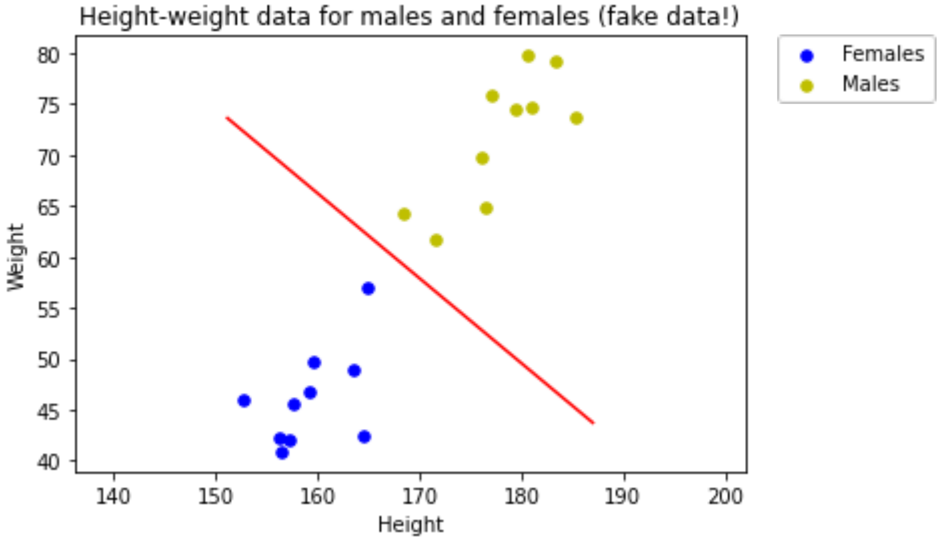
Expect Result (or look-alike):\ Q:\ [[1. 0. 0.]\ [0. 1. 0.]\ [0. 0. 0.]]\ c:\ [0. 0. 0.]\ A:\ [[164.411 42.38 1.]\ [157.308 42.15 1.]\ [159.579 49.739 1.]\ [-168.524 -64.353 -1.]\ [-171.597 -61.793 -1.]\ [-179.469 -74.552 -1.]]\ B:\ [-1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]\ x:\ [0.16001143 0.19207647 -38.32646165]\

Exercise 1.3: Use x from above to find w and b (5 points)

In [50]:
```
w = None
b = None
### BEGIN SOLUTION
w = np.matrix([[x[0]],[x[1]]])
b = x[2]
scale = np.linalg.norm(w)
w = w / scale
b = b / scale
### END SOLUTION
```

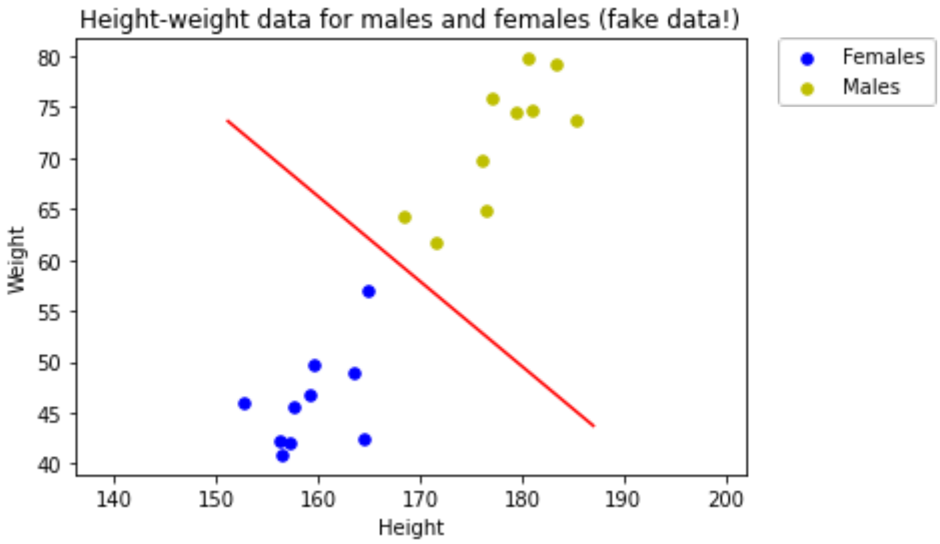
In [51]:
```
print('Optimal w: [%f %f] b: %f' % (w[0,0],w[1,0],b))

plot_mf(Xf,Xm)
plot_w(w,b)
plt.show()

# Test function: Do not remove
assert w.shape == 2 or w.shape ==(2,) or w.shape == (2,1), 'Size of w is incorrect'
assert w[0] > 0 and w[1] > 0 and w[0] <= 1 and w[1] <= 1, 'w value is incorrect'
assert isinstance(b, (float, int)), 'Type of b is incorrect'
assert b < 0, 'b value is incorrect'

print("success!")
# End Test function
```

```
Optimal w: [0.640061 0.768324] b: -153.309495

<ipython-input-45-a17db04aa7fd>:14: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  ylim = plt.axes().get_ylim()
<ipython-input-45-a17db04aa7fd>:15: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  xlim = plt.axes().get_xlim()
```




```
success!
```

Expect result (Or look-alike):\ Optimal w: [0.640061 0.768324] b: -153.309495



In [52]:
```
def predict_linear(X,w,b):
    s = X@w+b
    s[s >= 0] = 1
    s[s < 0] = -1
    return s
y_pred = predict_linear(X,w,b)
accuracy = np.sum(y_pred==y)/y.size
print(accuracy)
```

```
1.0
```

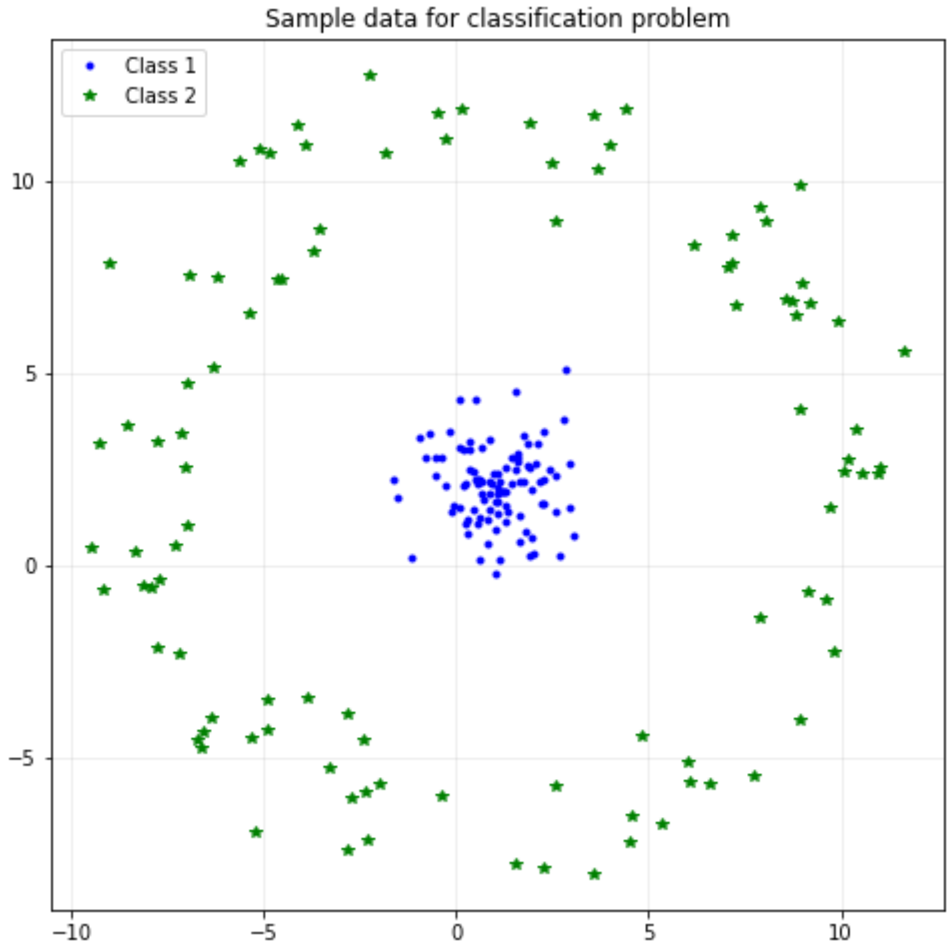
Exercise 2 (in lab): non-separable data (5 points)

Take the example of the annulus from the logistic regression lab. Verify that cvxopt cannot find a decision boundary for this case. Show your results in your lab report.

Note: You don't need to separate data to train/test values

In [53]:
```
# Generate data for class 1
mu_1 = np.array([1.0, 2.0])
sigma_1 = 1
num_sample = 100
cov_mat = np.matrix([[sigma_1,0],[0,sigma_1]])
X1 = np.random.multivariate_normal(mean= mu_1, cov=cov_mat, size
= num_sample)

# Generate data for class 2
angle = np.random.uniform(0, 2*np.pi, num_sample)
d =  np.random.normal(np.square(3*sigma_1),np.square(.5*sigma_1),
num_sample)
X2 = np.array([X1[:,0] + d*np.cos(angle), X1[:,1] + d*np.sin(angl
e)]).T

# Combine X1 and X2 into single dataset
X_annulus = np.concatenate([X1, X2],axis = 0)
y_annulus = np.append(-np.ones(num_sample),np.ones(num_sampl
e))[:,np.newaxis]
```

In [54]:
```
fig1 = plt.figure(figsize=(8,8))
ax = plt.axes()
plt.title('Sample data for classification problem')
plt.grid(axis='both', alpha=.25)
plt.plot(X1[:,0],X1[:,1],'b.', label = 'Class 1')
plt.plot(X2[:,0],X2[:,1],'g*', label = 'Class 2')
plt.legend(loc=2)
plt.axis('equal')
plt.show()
```



In [55]:
```
# Try to use try/catch to get output
get_error = False
try:
    Q = None
    c = None
    A = None
    B = None
    x = None
    w = None
    b = None
    ### BEGIN SOLUTION
    Q = np.eye(3);
    Q[2,2] = 0;
    c = np.zeros([3])
    A = np.multiply(np.tile(-y_annulus,[1, 3]), np.concatenate([X
_annulus, np.ones([200,1])],1))
    b = -np.ones([200])
    x = cvxopt_solve_qp(Q, c, A, b)
    w = np.matrix([[x[0]],[x[1]]])
    b = x[2]
    scale = np.linalg.norm(w)
    w = w / scale
    b = b / scale
    ### END SOLUTION
    output_str = 'Optimal w: [%f %f] b: %f' % (w[0,0],w[1,0],b)
    get_error = False
except Exception as e:
    output_str = e
    get_error = True
```

```
     pcost       dcost       gap    pres   dres
 0:  8.3375e-05  1.9930e+02  2e+02  2e+00  6e-14
 1:  7.6693e-09  4.6646e+02  5e+00  1e+00  2e-13
 2:  7.2606e-13  4.1550e+04  5e+00  1e+00  6e-12
 3:  7.2562e-17  3.6072e+08  4e+02  1e+00  2e-07
 4:  7.2562e-21  3.1308e+14  4e+06  1e+00  3e-02
 5:  7.2564e-25  2.7173e+22  3e+12  1e+00  3e+06
 6:  2.1134e-26  2.3230e+32  7e+20  1e+00  3e+16
 7:  2.2488e-26  2.7866e+40  8e+28  1e+00  1e+25
 8:  2.9744e-26  1.0338e+47  3e+35  1e+00  2e+31
 9:  6.1706e-26  4.9003e+54  1e+43  1e+00  2e+39
10:  5.3970e-26  1.4914e+62  4e+50  1e+00  9e+46
11:  5.1908e-26  6.2056e+69  2e+58  1e+00  3e+54
12:  4.6299e-26  1.5907e+77  5e+65  1e+00  6e+61
13:  5.5269e-26  4.1712e+84  1e+73  1e+00  3e+69
14:  4.2903e-26  1.9751e+92  6e+80  1e+00  4e+76
15:  5.7310e-26  2.2560e+99  6e+87  1e+00  7e+83
16:  4.8119e-26  5.2541e+106  1e+95  1e+00  3e+91
17:  5.4391e-26  1.4981e+114  4e+102  1e+00  6e+98
18:  4.2148e-26  6.9060e+121  2e+110  1e+00 1e+106
19:  5.6850e-26  1.0980e+129  3e+117  1e+00 5e+113
20:  4.5908e-26  4.0391e+136  1e+125  1e+00 2e+121
21:  5.5264e-26  1.2168e+144  3e+132  1e+00 9e+128
22:  4.9144e-26  4.0387e+151  1e+140  1e+00 1e+136
23:  5.7176e-26  1.2269e+159  3e+147  1e+00 7e+143
24:  5.6123e-26  2.5414e+166  7e+154  1e+00 3e+150
25:  5.6119e-26  7.2104e+173  2e+162  1e+00    inf
26:  4.6759e-26  1.8358e+181  5e+169  1e+00    inf
27:  5.4745e-26  5.0501e+188  1e+177  1e+00    inf
28:  4.3312e-26  2.3436e+196  7e+184  1e+00    inf
29:  5.7894e-26  2.6939e+203  8e+191  1e+00    inf
30:  4.8851e-26  5.9572e+210  2e+199  1e+00    inf
31:  5.3756e-26  1.7786e+218  5e+206  1e+00    inf
32:  4.2038e-26  7.8614e+225  2e+214  1e+00    inf
33:  5.6018e-26  1.5303e+233  4e+221  1e+00    inf
34:  4.1044e-26  6.7578e+240  2e+229  1e+00    inf
35:  5.5555e-26  1.3360e+248  4e+236  1e+00    inf
36:  4.1419e-26  5.9255e+255  2e+244  1e+00    inf
37:  5.5857e-26  1.0975e+263  3e+251  1e+00    inf
38:  4.1216e-26  4.7003e+270  1e+259  1e+00    inf
39:  5.5217e-26  1.0802e+278  3e+266  1e+00    inf
40:  4.4281e-26  4.9675e+285  1e+274  1e+00    inf
```

In [56]:
```
print(output_str)

# Test function: Do not remove
assert Q.shape == (3, 3) and Q[2,2] == Q[0,1] and Q[2,0] == 0 and Q[0,0] == Q[1,1] and Q[0,0] == 1, 'Q value is incorrect'
assert c.shape == 3 or c.shape == (3,) or c.shape == (3,1), 'Size of c is incorrect'
assert np.all((c == 0)), 'c value is incorrect'
assert A.shape == (200,3), 'Size of A is incorrect'
assert str(output_str) == 'domain error' or "'NoneType' object is not subscriptable" or get_error, 'Output incorrect'

print("success!")
# End Test function
```

```
domain error
success!
```

Expect result: Show something error calculation

Generalized Lagrangian optimization for SVMs

Now we consider the generalized Lagrangian for the SVM. This technique is suitable for solving problems of the form

$$\begin{array}{rl} \min_{\mathbf{w}} & f(\mathbf{w}) \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, i \in 1..k \\ & h_i(\mathbf{w}) = 0, i \in 1..l \end{array}$$

The generalized Lagrangian is

$$\mathcal{L}(\mathbf{w}, \alpha, \beta) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{l} \beta_i h_i(\mathbf{w}),$$

which has been cleverly arranged to be equal to $f(\mathbf{w})$ whenever $\mathbf{w}$ satisfies the constraints and ∞ otherwise.

Primal and dual Lagrangian problems

The primal problem is to find

$$p^* = \min_{\mathbf{w}} \theta_{\mathcal{P}}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\alpha, \beta, \alpha_i \geq 0} \mathcal{L}(\mathbf{w}, \alpha, \beta)$$

and the dual problem is to find

$$d^* = \max_{\alpha, \beta, \alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha, \beta, \alpha_i \geq 0} \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \alpha, \beta).$$

If f is convex, the g_i's are affine, the h_i's are convex, and the g_i's are strictly feasible, it turns out that the solutions to the primal and dual problem are the same, and the KKT conditions hold:

$$\begin{array}{rcl} \frac{\partial}{\partial w_i} \mathcal{L}(\mathbf{w}^*, \alpha^*, \beta^*) & = & 0, i \in 1..n \\ \frac{\partial}{\partial \beta_i} \mathcal{L}(\mathbf{w}^*, \alpha^*, \beta^*) & = & 0, i \in 1..l \\ \alpha_i^* g_i(\mathbf{w}^*) & = & 0, i \in 1..k \\ g_i(\mathbf{w}^*) & \leq & 0, i \in 1..k \\ \alpha_i^* & \geq & 0, i \in 1..k \end{array}$$

Solving the dual Lagrangian problem

The dual problem turns out to be easiest to solve.

We first solve for $\mathbf{w}$ assuming fixed α and β (we don't have equality constraints though, so no need for β).

We need to rewrite the SVM constraints in the necessary form with $g_i(\mathbf{w}) = 0$.obtain, for the SVM, constraints

$$g_i(\mathbf{w}, b) = -y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) + 1 \geq 0.$$

Using that definition of $g_i(\mathbf{w}, b)$, we obtain the Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - 1 \right]$$

Taking the gradient of $\mathcal{L}$ with respect to $\mathbf{w}$ and setting it to 0, we obtain

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b, \alpha) = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y^{(i)} \mathbf{x}^{(i)} = 0,$$

which gives us

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \mathbf{x}^{(i)}.$$

From $\frac{\partial \mathcal{L}}{\partial b} = 0$, we obtain

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

which is interesting (think about what it means also considering that $\alpha_i > 0$ only for examples on the margin. Unfortunately it doesn't help us find b! In any case, we plug this definition for the optimal $\mathbf{w}$ into the original Lagrangian, to obtain

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)} - b \sum_{i=1}^{m} \alpha_i y^{(i)}.$$

We already know that the last term is 0, so we get

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle.$$

OK! We've eliminated $\mathbf{w}$ and b from the optimization. Now we just need to maximize $\mathcal{L}$ with respect to α. This gives us the final (dual) optimization problem

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \textstyle\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle \\ \text{such that} \quad & \alpha_i \geq 0, i \in 1..m \\ & \textstyle\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

This turns out to be QP again!

Aside: once we solve for α, we obtain $\mathbf{w}$ according to the equation above, then it turns out that the optimal b can be obtained as in the lecture notes.

QP solution to dual problem

We need to negate our objective function to turn the max (SVM formulation) into a min (QP formalation).

For the second term of $W(\alpha)$, first let $\mathtt{K}$ be the kernel matrix with $\mathtt{K}_{ij} = \left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle$. Then $\alpha^\top \mathrm{diag}(\mathbf{y}) \mathtt{K} \mathrm{diag}(\mathbf{y}) \alpha$ gives us the summation in the second term ($\mathrm{diag}(\mathbf{y})$ is just the square diagonal matrix with $\mathbf{y}$ as its diagonal).

The (negated) first term of $W(\alpha)$ can be written in QP form with $\mathbf{c} = \begin{bmatrix} -1 & -1 & \ldots \end{bmatrix}^\top$.

So that gives us our QP setup:

$$\mathtt{Q} = \mathrm{diag}(\mathbf{y}) \mathtt{K} \mathrm{diag}(\mathbf{y}) \quad \mathbf{c} = \begin{bmatrix} -1 \\ -1 \\ \vdots \end{bmatrix}$$
$$\mathtt{A} = -\mathtt{I}_{m \times m} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$$
$$\mathtt{E} = \mathbf{y}^\top \quad \mathbf{d} = \begin{bmatrix} 0 \end{bmatrix}.$$

OK, now the code:

In [94]:
```
m = X.shape[0];
n = X.shape[1];

# Transform data set so that each attribute has a
# mean of 0 and a standard deviation of 1

def preprocess(X):
    means = X.mean(0);
    scales = 1/np.std(X,0);
    Xh = np.concatenate([X.T,np.ones([1,20])],0);
    Tm = np.matrix(np.eye(3));
    Tm[0:2,2:3] = -X.mean(0).T;
    Ts = np.matrix(np.eye(3));
    Ts[0:2,0:2] = np.diagflat(scales);
    T = Ts*Tm;
    XX = (T * Xh);
    XX = XX[0:2,:].T;
    return XX, T;

# RBF/Gaussian kernel

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 linear_kernel(X):
    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,:].reshape(1,-1)@X[j,:].reshape(-1,1))
    return K;
```

Exercise 3 (in lab): linearly separable data (15 points)

Take the example data from the exercise in lecture. Verify that you can use the dual optmization to find the decision boundary for such "easy" cases. Show your results in your lab report.

Exercise 3.1: Find Q, c, A, b, E, d for input into cvxopt_solve_qp function (10 points)

In [95]:
```
Q = None
c = None
A = None
b = None
E = None
d = None

K = linear_kernel(X)
### BEGIN SOLUTION
Q = np.multiply(y * y.T, K)
c = -np.ones([m]);
A = -np.eye(m);
B = np.zeros([m]);
E = y.T;
d = np.zeros(1);
### END SOLUTION
alpha_star = cvxopt_solve_qp(Q, c, A, B, E, d)
```

```
     pcost       dcost       gap    pres   dres
 0: -2.7646e+00 -4.9725e+00  5e+01  6e+00  2e+00
 1: -6.4101e+00 -3.8299e+00  1e+01  2e+00  6e-01
 2: -5.0055e+00 -1.2719e+00  7e+00  8e-01  2e-01
 3: -5.1552e-02 -4.0648e-02  5e-01  3e-02  1e-02
 4: -1.8200e-02 -3.5404e-02  2e-02  2e-17  2e-13
 5: -3.0877e-02 -3.1392e-02  5e-04  8e-18  2e-13
 6: -3.1245e-02 -3.1250e-02  5e-06  4e-18  1e-13
 7: -3.1248e-02 -3.1249e-02  5e-08  3e-18  2e-13
Optimal solution found.
```

In [96]:
```
print('Q rank: %d' % np.linalg.matrix_rank(Q))
print("Optimal alpha:\n", alpha_star)

# Test function: Do not remove
assert Q.shape == (20, 20), 'Size of Q is incorrect'
assert np.linalg.matrix_rank(Q) == 2, 'Q rank is incorrect'
assert np.all((c == -1)), 'c value is incorrect'
assert A.shape == (20,20), 'Size of A is incorrect'
assert np.all((B == 0)), 'b value is incorrect'
assert np.array_equal(np.round(E,1), np.round(y.T,1)), 'E value i
s incorrect'
assert d.shape == (1,) or d.shape == 1 or d.shape == (1,1) , 'Siz
e of d is incorrect'
assert np.all((d == 0)), 'd value is incorrect'
assert alpha_star.shape == (20,) or alpha_star.shape == 20 or alp
ha_star.shape == (20,1), 'Size of alpha_star is incorrect'

print("success!")
# End Test function
```

```
Q rank: 2
Optimal alpha:
 [3.12484796e-02 1.13821985e-09 7.68004003e-10 6.22346942e-10
 6.43114906e-10 8.93974184e-10 6.60695980e-10 4.31409827e-10
 6.68557407e-10 1.19689394e-09 1.56332821e-02 1.56151999e-02
 5.00606721e-10 6.71834910e-10 4.89168003e-10 4.93113729e-10
 4.90935291e-10 4.85894509e-10 9.54348935e-10 4.42078646e-10]
success!
```

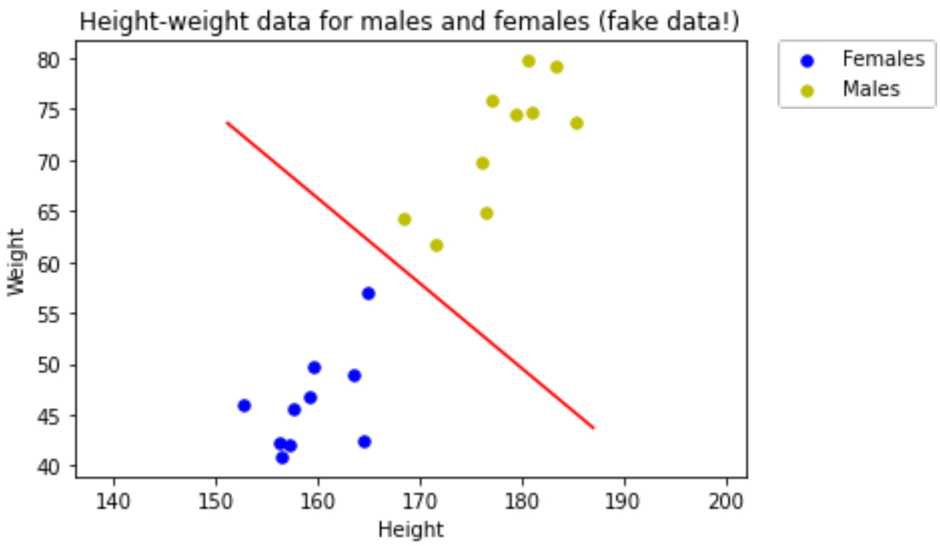
Expect Result (or look a like):\ Q rank: 2\ Optimal alpha:\ [3.12484796e-02 1.13821985e-09 7.68004003e-10 6.22346942e-10\ 6.43114906e-10 8.93974184e-10 6.60695980e-10 4.31409827e-10\ 6.68557407e-10 1.19689394e-09 1.56332821e-02 1.56151999e-02\ 5.00606721e-10 6.71834910e-10 4.89168003e-10 4.93113729e-10\ 4.90935291e-10 4.85894509e-10 9.54348935e-10 4.42078646e-10]

Exercise 3.2: write get_wb function (5 points)

In [97]:
```
def get_wb(X, y, alpha, K):
    # Find the support vectors
    S = alpha > 1e-6
    XS = None
    yS = None
    alphaS = None
    alphaSyS = None
    w = None
    # Find b
    KS = None
    NS = None
    b = None
    # Normalize w,b
    scalef = None
    w = None
    b = None

    ### BEGIN SOLUTION
    S = alpha > 1e-6
    XS = X[S,:]
    yS = y[S]
    alphaS = alpha[S]
    alphaSyS = np.tile(np.multiply(yS.T, alphaS).T, n)
    w = sum(np.multiply(alphaSyS, XS)).T
    # Find b
    KS = K[S,:][:,S]
    NS = yS.shape[0]
    b = (np.sum(yS) - np.sum(np.multiply(alphaS,yS.T)*KS))/NS
    # Normalize w,b
    scalef = np.linalg.norm(w)
    w = w / scalef
    b = b / scalef
    ### END SOLUTION
    return w,b
```

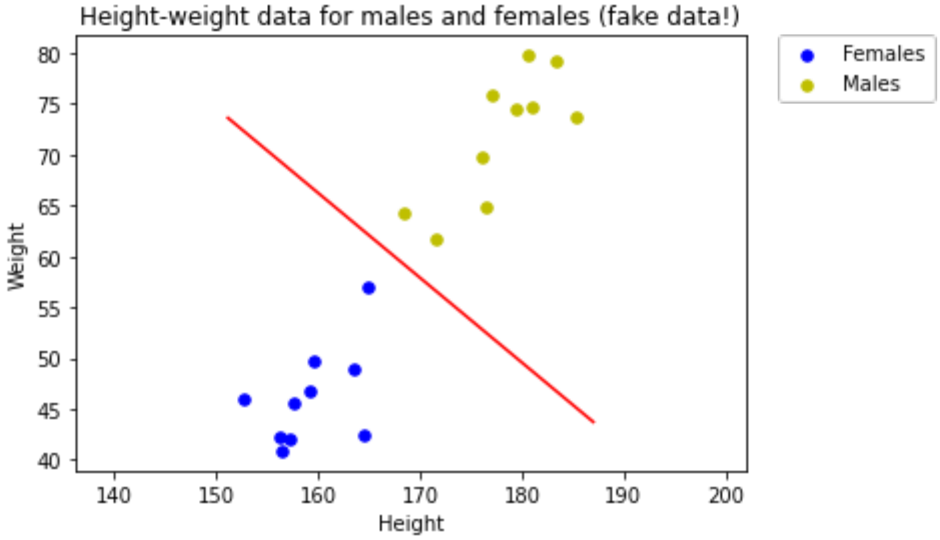
In [98]:
```
# Test function: Do not remove
w,b = get_wb(X, y, alpha_star, K)

print("Optimal w: [%f,%f] b: %f" % (w[0],w[1],b))
plot_mf(Xf,Xm)

plot_w(w,b)
plt.show()

print("success!")
# End test function
```

```
Optimal w: [0.640062,0.768324] b: -153.309583

<ipython-input-45-a17db04aa7fd>:14: MatplotlibDeprecationWarning:
Adding an axes using the same arguments as a previous axes curren
tly reuses the earlier instance.  In a future version, a new inst
ance will always be created and returned.  Meanwhile, this warnin
g can be suppressed, and the future behavior ensured, by passing
a unique label to each axes instance.
  ylim = plt.axes().get_ylim()
<ipython-input-45-a17db04aa7fd>:15: MatplotlibDeprecationWarning:
Adding an axes using the same arguments as a previous axes curren
tly reuses the earlier instance.  In a future version, a new inst
ance will always be created and returned.  Meanwhile, this warnin
g can be suppressed, and the future behavior ensured, by passing
a unique label to each axes instance.
  xlim = plt.axes().get_xlim()
```




```
success!
```

Expect Result (Or look-alike):\ Optimal w: [0.640062,0.768324] b: -153.309583



In [99]:
```
y_pred = predict_linear(X,w,b)
accuracy = np.sum(y_pred==y)/y.size
print(accuracy)
```

```
1.0
```

Exercise 4 (in lab): non-separable data, linear kernel (15 points)

Again, take the example of the annulus from the logistic regression lab. Verify that the dual optimization with the linear kernel still cannot find a decision boundary for this case. Show your results in your lab report.

In [118]:
```
(m, n) = X_annulus.shape
print(X_annulus.shape)
K_annulus = linear_kernel(X_annulus);

Q_annulus = None
c = None
A = None
B = None
E = None
d = None

### BEGIN SOLUTION
Q_annulus = np.multiply(y_annulus * y_annulus.T, K_annulus)

c = -np.ones([m]);
A = -np.eye(m);
B = np.zeros([m]);
E = y_annulus.T;
d = np.zeros(1);
### END SOLUTION
```

```
(200, 2)
```

In [119]:
```
alpha_star_annulus = cvxopt_solve_qp(Q_annulus, c, A, B, E, d)

print('Q rank: %d' % np.linalg.matrix_rank(Q_annulus))
print("Optimal alpha:", alpha_star_annulus)
# Test function: Do not remove
assert np.linalg.matrix_rank(Q_annulus) > 2, "Q rank is incorrec
t"
assert alpha_star_annulus is None, "alpha_star_annulus cannot be
calculated."

print("success!")
# End test function
```

```
     pcost       dcost       gap    pres   dres
 0: -1.9930e+02 -4.2459e+02  2e+02  8e-15  2e+00
 1: -4.6646e+02 -4.7177e+02  5e+00  6e-14  1e+00
 2: -4.1550e+04 -4.1555e+04  5e+00  4e-11  1e+00
 3: -3.5661e+08 -3.5661e+08  9e+02  5e-07  1e+00
 4: -4.3418e+08 -4.3418e+08  1e+03  3e-07  1e+00
Terminated (singular KKT matrix).
Q rank: 2
Optimal alpha: None
success!
```

In [120]:
```
get_error = False
try:
    w,b = get_wb(X_annulus, y_annulus, alpha_star_annulus, K_annu
lus)

    output_str = "Optimal w: [%f,%f] b: %f" % (w[0],w[1],b)
    plot_mf(Xf,Xm)
    plot_w(w,b)
    get_error = False
except Exception as e:
    output_str = str(e)
    get_error = True

print(output_str)
# Test function: Do not remove
assert str(output_str) == 'domain error' or "'NoneType' object is
not subscriptable" or get_error, 'Output incorrect'

print("success!")
# End Test function
```

```
'>' not supported between instances of 'NoneType' and 'float'
success!
```

Expect Result: Any error

Exercise 5 (in lab): "easy" non-separable data, Gaussian (RBF) kernel with non-overlapping data (10 points)

Now use the Gaussian (radial basis function) kernel instead of the linear kernel implemented in the code above and verify that you can correctly solve the problem.

In [129]:
```
(m, n) = X_annulus.shape
K_annulus = None
Q_annulus = None
c = None
A = None
B = None
E = None
d = None

### BEGIN SOLUTION
K_annulus = gauss_kernel(X_annulus)

Q_annulus = np.multiply(y_annulus * y_annulus.T, K_annulus)

c = -np.ones([m]);
A = -np.eye(m);
B = np.zeros([m]);
E = y_annulus.T;
d = np.zeros(1);
### END SOLUTION
```

In [130]:
```
alpha_star_annulus = cvxopt_solve_qp(Q_annulus, c, A, B, E, d)
w,b = get_wb(X_annulus, y_annulus, alpha_star_annulus, K_annulus)
w = w.reshape(-1,1)

print('Q rank: %d' % np.linalg.matrix_rank(Q))
print("Optimal alpha:")
print(alpha_star_annulus[:5])
print(w.shape)
print("Optimal w: [%f,%f] b: %f" % (w[0],w[1],b))

# Test function: Do not remove
assert np.linalg.matrix_rank(Q_annulus) > 2, "Q rank is incorrec
t"
assert alpha_star_annulus is not None, "alpha_star_annulus cannot
be calculated."
assert w.shape == (2,1), 'Size of w is incorrect'
assert np.all(w <= 1) and np.all(w >= -1), 'w value is incorrect'

print("success!")
# End test function
```

```
     pcost       dcost       gap    pres   dres
 0: -5.0795e+01 -1.3928e+02  9e+01  1e-15  2e+00
 1: -5.9073e+01 -6.6466e+01  7e+00  2e-14  3e-01
 2: -6.2076e+01 -6.3563e+01  1e+00  1e-14  4e-02
 3: -6.2184e+01 -6.2423e+01  2e-01  8e-15  5e-03
 4: -6.2205e+01 -6.2232e+01  3e-02  3e-14  5e-04
 5: -6.2209e+01 -6.2209e+01  7e-04  4e-14  6e-06
 6: -6.2209e+01 -6.2209e+01  1e-05  1e-14  7e-08
Optimal solution found.
Q rank: 2
Optimal alpha:
[5.77224260e-01 1.31648362e+00 1.17475881e+00 4.38419640e-01
 8.16578932e-07]
(2, 1)
Optimal w: [-0.974636,0.223795] b: 0.008878
success!
```

Expect result (or look-alike):\ pcost dcost gap pres dres\ 0: -5.0795e+01 -1.3928e+02 9e+01 1e-15 2e+00\ 1: -5.9073e+01 -6.6466e+01 7e+00 2e-14 3e-01\ 2: -6.2076e+01 -6.3563e+01 1e+00 1e-14 4e-02\ 3: -6.2184e+01 -6.2423e+01 2e-01 8e-15 5e-03\ 4: -6.2205e+01 -6.2232e+01 3e-02 3e-14 5e-04\ 5: -6.2209e+01 -6.2209e+01 7e-04 4e-14 6e-06\ 6: -6.2209e+01 -6.2209e+01 1e-05 1e-14 7e-08\ Optimal solution found.\ Q rank: 2\ Optimal alpha:\ [5.77224260e-01 1.31648362e+00 1.17475881e+00 4.38419640e-01 8.16578932e-07]\ (2, 1)\ Optimal w: [-0.974636,0.223795] b: 0.008878

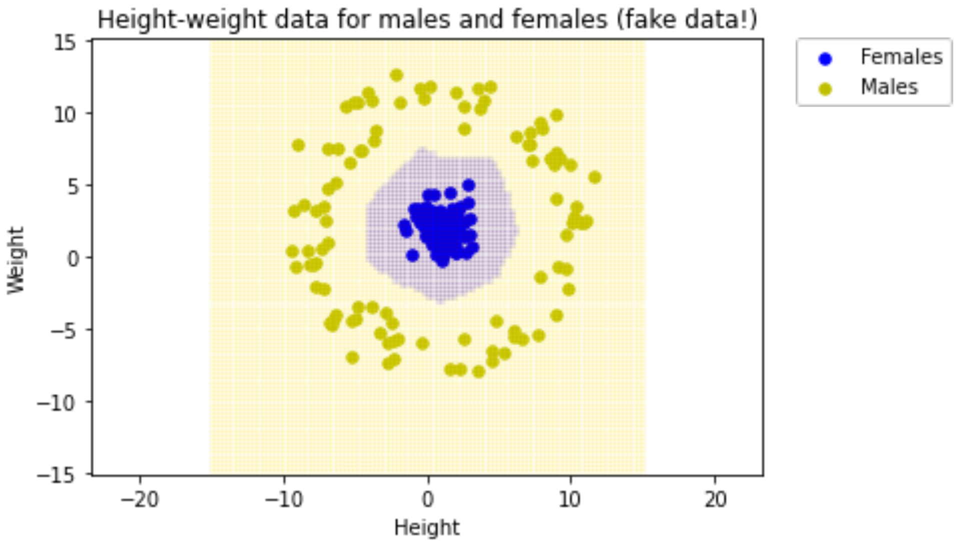
In [136]:
```
def predict(x,X,y,alpha):
    s = []
    sigma = 0.2
    for j in range(x.shape[0]):
        ss = 0
        for i in range(X.shape[0]):
            ss += alpha[i]*y[i]*np.exp((-(X[i]-x[j])@(X[i]-x
[j]))/(2*sigma*sigma))
        s.append(ss)
    s = np.array(s)
    s[s >= 0] = 1
    s[s < 0] = -1
    return s

y_pred = predict(X_annulus,X_annulus,y_annulus,alpha_star_annulu
s)
np.sum(y_annulus == y_pred)/y_annulus.size
```

Out[136]:
```
1.0
```

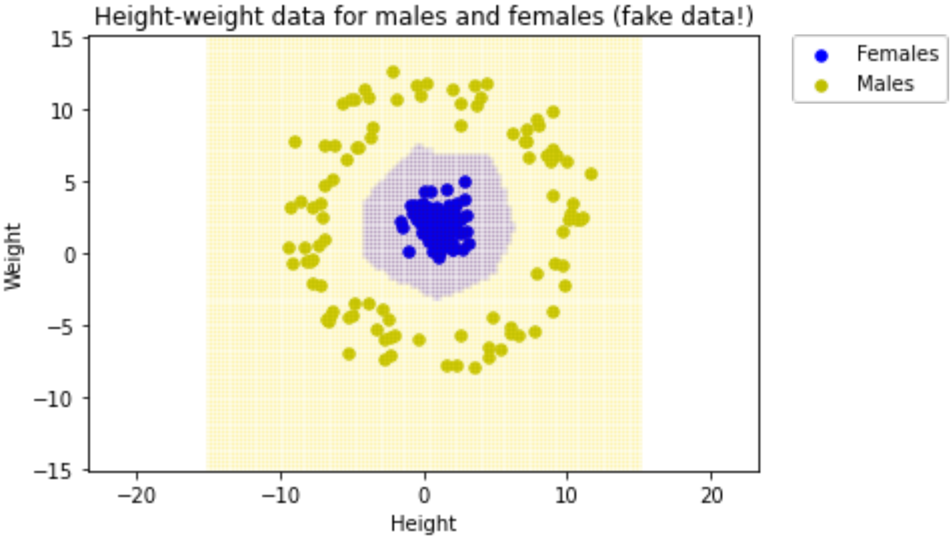
See your graph of classification

In [137]:
```
x_series = np.linspace(-15,15,100)
y_series = np.linspace(-15,15,100)

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_annulus,y_annulus,alpha_star_annulus)

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


#plt.scatter(mesh[:,0],mesh[:,1],c=y_pred)
plot_mf(X1,X2)
plt.pcolormesh(x_mesh,y_mesh,y_pred,cmap='viridis',shading='auto
',alpha=0.1)
```

Out[137]: <matplotlib.collections.QuadMesh at 0x7fb4664480d0>



Expect Result:



Exercise 6 (take home): more difficult non-separable data

Now find or generate a dataset in which the decision boundary is nonlinear AND the data overlap along that nonlinear boundary. Show that the result.

In [ ]:

In [ ]:

In [ ]:

In [ ]: