COMPSCI 371D Homework 5

Shuyi Gin Wang

Problem 0 (3 points)

Part 1: The Logistic-Regression Classifier in One Dimension

In [686]:
```
from urllib.request import urlretrieve
from os import path as osp


def retrieve(file_name, semester='fall21', course='371d', homework=5):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci{}/homework/
        url = fmt.format(semester, course, homework, file_name)
        urlretrieve(url, file_name)
        print('Downloaded file {}'.format(file_name))
```

In [687]:
```
import pickle


file_name = 'data1d.pkl'
retrieve(file_name, homework=5)
with open(file_name, 'rb') as file:
    t = pickle.load(file)
tx, ty = t['x'], t['y']
```

```
Using previously downloaded file data1d.pkl
```

Problem 1.1 (Exam Style)

$$l(y, f(a)) = -ylog\frac{1}{1+e^{-a}} - (1-y)log(1 - \frac{1}{1+\frac{1}{e}^{-a}})$$

$$= -ylog\frac{1}{1+e^{-a}} - log(1 - \frac{1}{1+e^{-a}}) + ylog(1 - \frac{1}{1+e^{-a}})$$

$$= -ylog\frac{1}{1+e^{-a}} - (y-1)log(\frac{e^{-a}}{1+e^{-a}})$$

$$= ylog(1+e^{-a}) + (y-1)(-a - log(1+e^{-a}))$$

$$= ylog(1+e^{-a}) - ya - ylog(1+e^{-a}) + a + log(1+e^{-a})$$

$$= a(1-y) + log(1+e^{-a})$$

Problem 1.2 (Exam Style)

The second derivative of $l(y, f(a))$ is:

$$= \frac{e^{-a}}{(e^{-a}+1)^2}$$

Because this expression is positive at all points where $a \in R$, $l(y, f(a))$ is strictly convex.

Problem 1.3 (Exam Style)

The combined function of L_T is a convex function because it's a combination of many sample losses $l(y, f(a))$, which are convex functions due to linearity of derivatives. The sum of individually convex functions lead to another convex function.

Problem 1.4 (Exam Style)

$$l(y, f(a) = b) = (1-y)(b) + log(1+e^{-b})$$
$$L_T(b,0) = \frac{F(b + log(1+e^{-b})) + (N-F)log(1+e^{-b}))}{N}$$

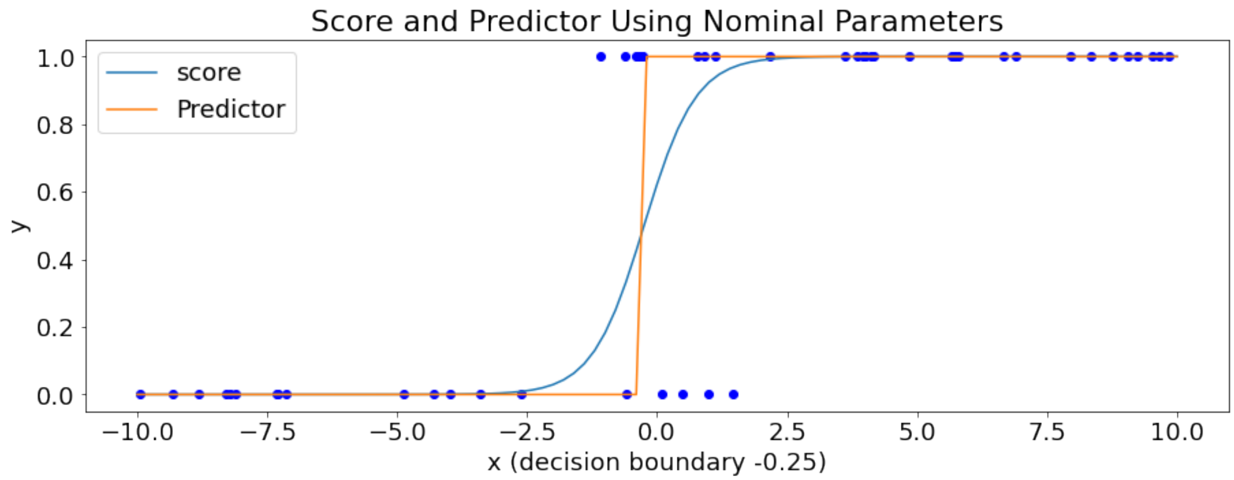
$$l(y, f(a) = 0) = (1-y)(0) + log(1+e^{0})$$
$$L_T(0,0) = log(2)$$

Problem 1.5

In [688]:
```
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

def logistic(a):
    return 1/(1+np.exp(-a))

def affine(x,v):
    return v[0]+v[1]*x

def score(x,v):
    return logistic(affine(x,v))

def h(x,v):
    if isinstance(x, float):
        return helper(score(x,v))
    else:
        return list(map(helper, score(x,v)))


def helper(x):
    if x > 1/2:
        return 1
    else:
        return 0
```

In [689]:
```
def plot_score(x,y,v,loss_name,x_bound,n_points,type_size):
    plt.figure(figsize = (15,5))
    plt.rcParams['font.size'] = type_size
    plt.plot(x, y, 'o', color='blue')
    xs = np.linspace(-x_bound, x_bound, n_points)
    plt.plot(xs, score(xs,v), label = 'score')
    plt.plot(xs, h(xs,v), label = 'Predictor')
    plt.title('Score and Predictor Using ' + loss_name.title())
    plt.xlabel('x (decision boundary ' + str(round(-v[0]/v[1],3))+')')
    plt.ylabel('y')
    plt.legend()
```

In [690]:
```
v_hat_test = np.array((0.5, 2.))
plot_score(tx, ty, v_hat_test, 'nominal parameters',
    x_bound=10., n_points=101, type_size=18)
```



Problem 1.6

In [691]:
```
from scipy import linalg, optimize
```

In [692]:
```
def cross_entropy(y,p):
    pp = y*p + (1-y)*(1-p)
    return -np.log(pp)

def sample_loss(x,y,v,loss=cross_entropy):
    return loss(y,score(x,v))

def risk(v,x,y,loss=cross_entropy):
    n, total = len(x), 0
    for i, case in enumerate (x):
        total += sample_loss(case,y[i],v,loss)
    return total/n

def train(x,y,loss=cross_entropy):
    x_0 = np.array([0,0])
    return optimize.minimize(risk, x_0, args=(x,y,loss), method='CG')
```

In [693]:
```
v_hat_ce = train(tx, ty)
print(v_hat_ce)
```

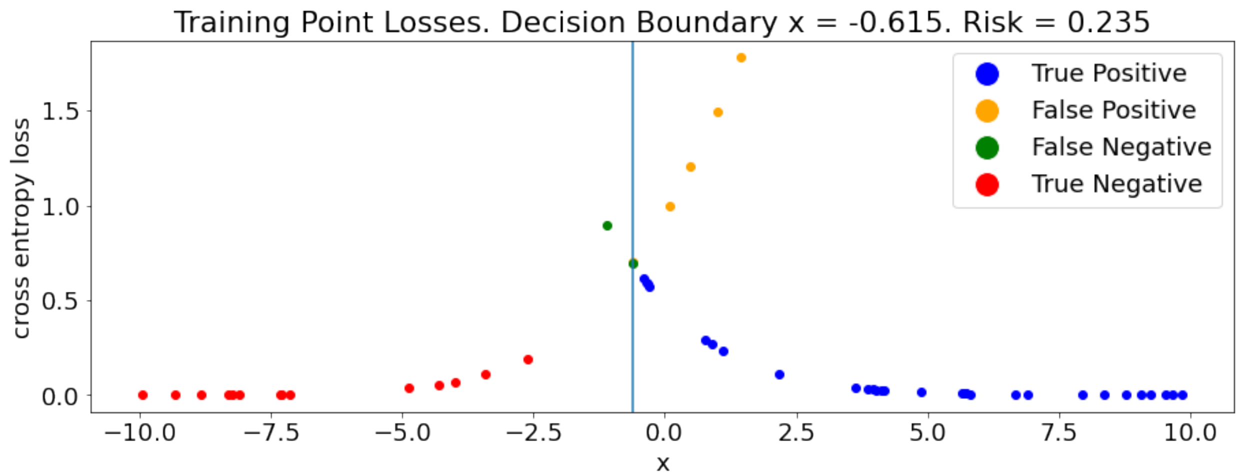
```
     fun: 0.2352997251244322
     jac: array([-9.31322575e-09,  1.58324838e-07])
 message: 'Optimization terminated successfully.'
    nfev: 51
     nit: 7
    njev: 17
  status: 0
 success: True
       x: array([0.47471946, 0.7714649 ])
```

Problem 1.7

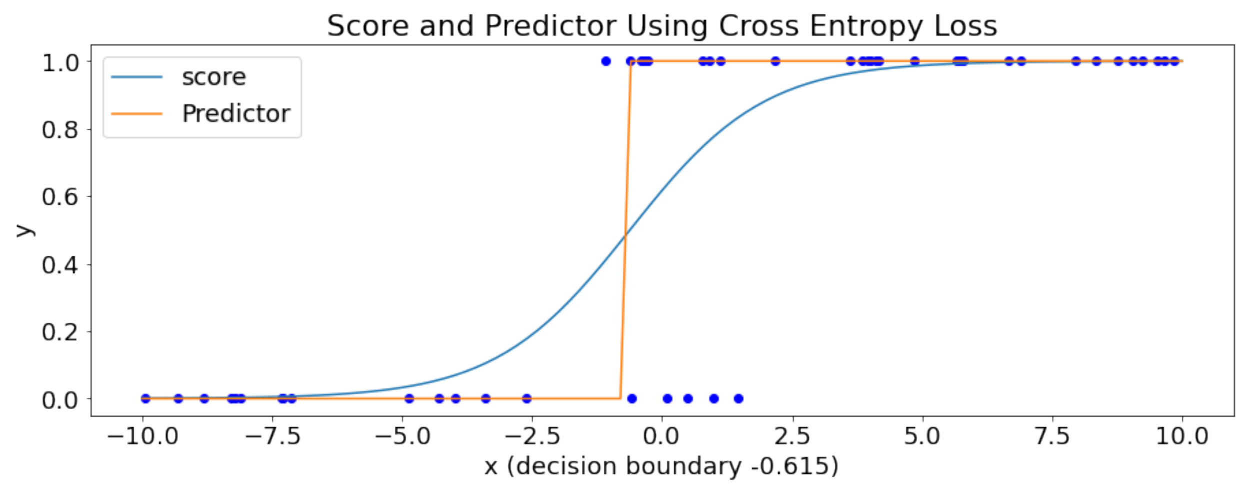
In [694]:
```
ce_name = 'cross entropy loss'
x_bound, n_points = 10., 101
type_size = 18
```

In [695]:
```python
from matplotlib.lines import Line2D
def plot_losses(x,y,v,loss_fct, loss_name, type_size):
    plt.figure(figsize = (15,5))
    plt.rcParams['font.size'] = type_size
    labels = ['True Positive','False Positive',
              'False Negative','True Negative']
    colors = ['blue', 'orange', 'green', 'red']
    loss = []

    for i, _x in enumerate (x):
        if h(_x,v) == 1:
            if y[i] == 1:
                c = 'blue'
                l = 'True Positive'
            else:
                c = 'orange'
                l = 'False Positive'
        else:
            if y[i] == 1:
                c = 'green'
                l = 'False Negative'
            else:
                c = 'red'
                l = 'True Negative'

        plt.plot(_x, loss_fct(int(y[i]),score(_x,v)),
                 'o', color=c, label = l)
    plt.axvline(-v[0]/v[1])
    dots =  [Line2D([0], [0], marker='o', color = 'w',
                          markerfacecolor = c,
                    markersize=type_size) for c in colors]
    plt.legend(dots,labels)
    plt.title('Training Point Losses. Decision Boundary x = '
              + str(np.round(-v[0]/v[1],3))
              + '. Risk = ' + str(np.round(risk(v,x,y,loss_fct),3)))
    plt.xlabel('x')
    plt.ylabel(loss_name)
```

In [696]:
```
plot_losses(tx, ty, v_hat_ce['x'], cross_entropy, ce_name, type_size)
```

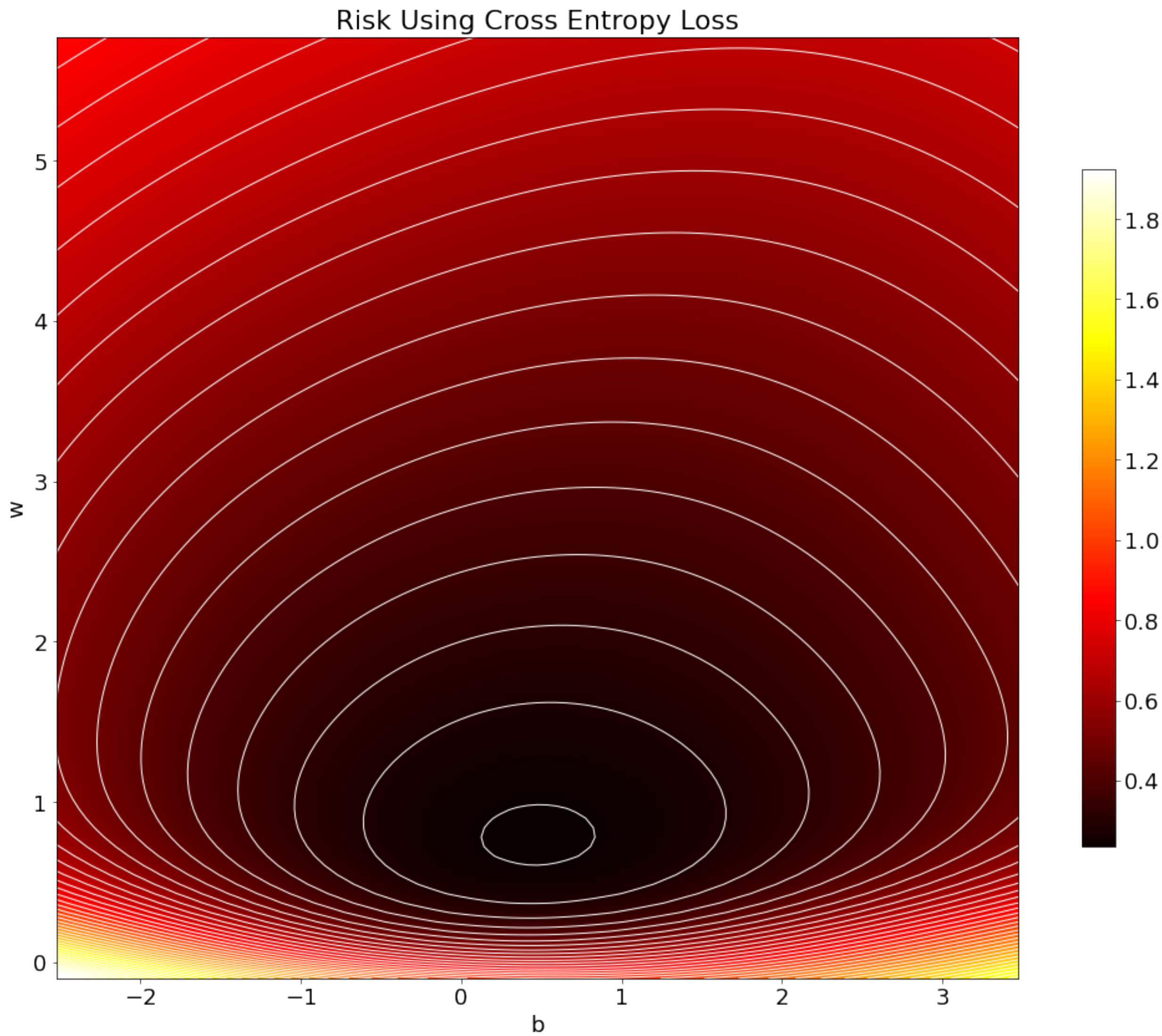


In [697]:
```
plot_score(tx, ty, v_hat_ce['x'], ce_name, x_bound, n_points, type_siz
```



In [698]:
```
def plot_contours(x, y, v, loss_fct, loss_name, n_points, type_size, f
    b_hat, w_hat = v[0], v[1]
    box = (b_hat-3.,b_hat+3, -0.1, w_hat+5.)
    fig = plt.figure(figsize = fig_size, tight_layout=True)
    b = np.linspace(b_hat-3., b_hat + 3., n_points)
    w = np.linspace(-0.1, w_hat + 5., n_points)
    b_grid, w_grid = np.meshgrid(b, w)
    v_grid = np.stack((b_grid, w_grid), axis=0)
    fct_grid = []
    for i in range(v_grid.shape[1]):
        for j in range(v_grid.shape[2]):
            fct_grid.append(risk(v_grid[:,i,j],x,y,loss_fct))
    f_grid = np.reshape(fct_grid,b_grid.shape)
    img = plt.imshow(f_grid, interpolation='bilinear',
            origin='lower', extent=box, cmap=plt.cm.hot)
    bar = fig.colorbar(img, shrink=0.72)
    plt.contour(b_grid, w_grid, f_grid, 50, colors='w', linewidths=1)
    plt.xlabel('b')
    plt.ylabel('w')
    plt.title('Risk Using '+loss_name.title())
```

In [699]:
```
plot_contours(tx, ty, v_hat_ce['x'], cross_entropy, ce_name, n_points,
```



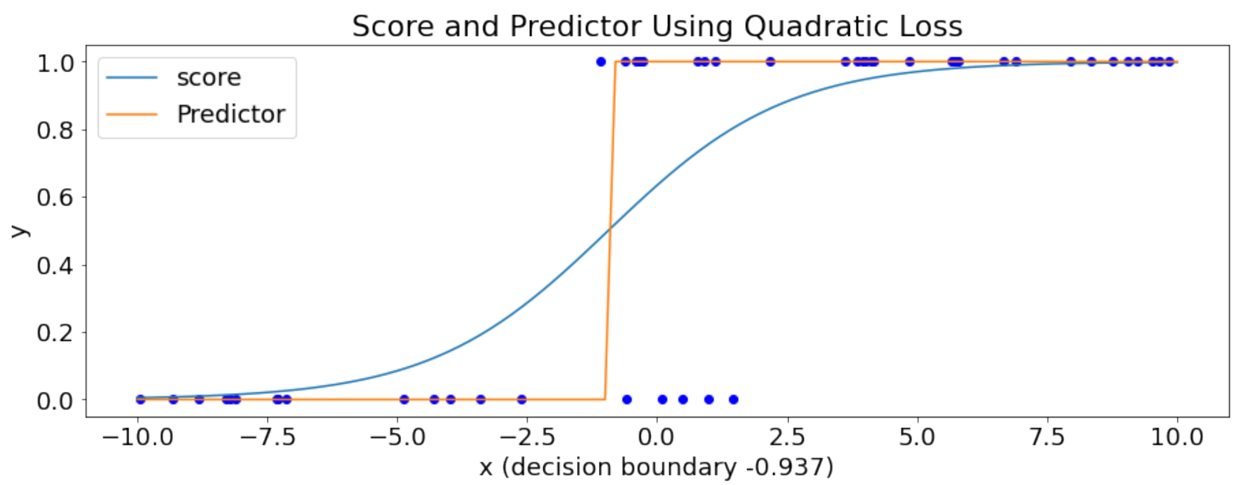
Problem 1.8

In [700]:
```
def quadratic(y,p):
    return (y-p)**2
```

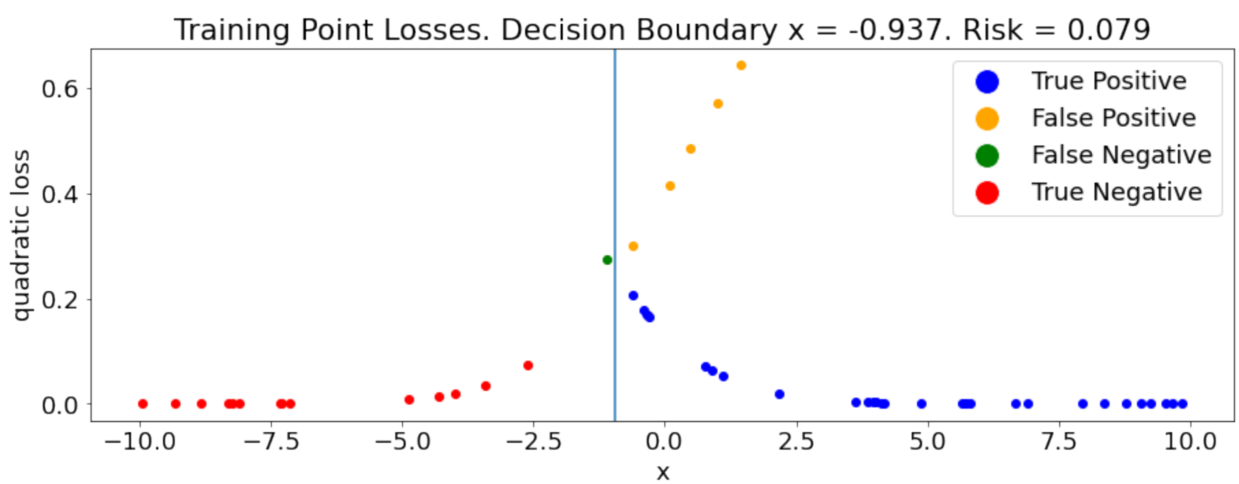
In [701]:
```
v_hat_q = train(tx, ty, loss=quadratic)
print(v_hat_q)
q_name = 'quadratic loss'
```

```
     fun: 0.07929000928722678
     jac: array([-7.45058060e-09, -4.93600965e-08])
 message: 'Optimization terminated successfully.'
    nfev: 102
     nit: 11
    njev: 34
  status: 0
 success: True
       x: array([0.54866042, 0.58562302])
```

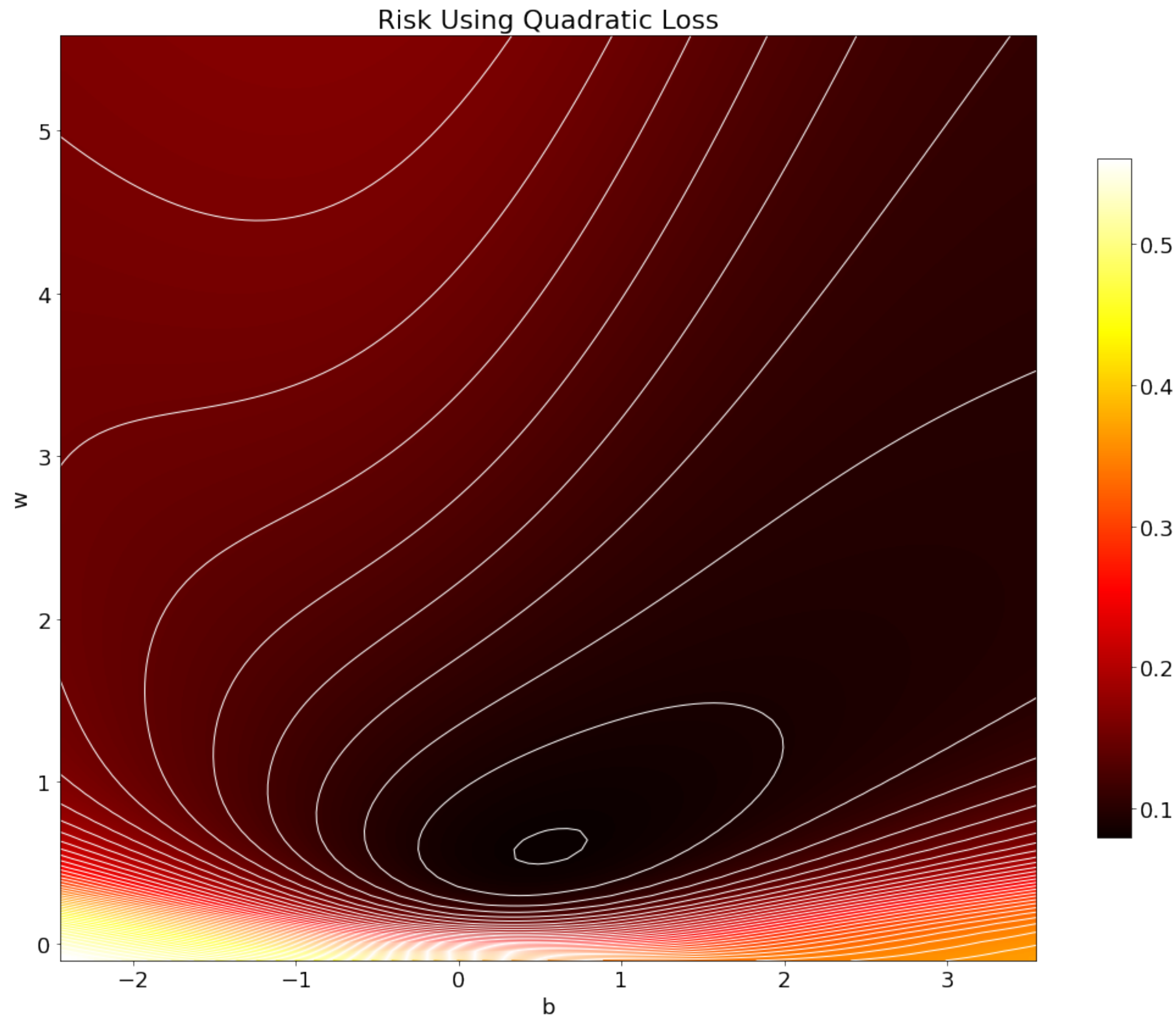
In [702]:
```
plot_score(tx, ty, v_hat_q['x'], q_name, x_bound, n_points, type_size)
```



In [703]:
```
plot_losses(tx, ty, v_hat_q['x'], quadratic, q_name, type_size)
```



```
In [704]: plot_contours(tx, ty, v_hat_q['x'], quadratic, q_name, n_points, type_
```



Problem 1.9 (Exam Style)

The risk with the quadratic loss is not convex everywhere because we can see from the top left quadrant of the plot, the red gradient color goes from darker red to light red and then back to darker red along the isocontour. This means that if we have two points (b, w) and (b', w'), the risk using quadratic loss with $(\frac{b+b'}{2}, \frac{w+w'}{2})$ is greater than both the risk of using quadratic loss with (b, w) and (b', w').

Problem 1.10

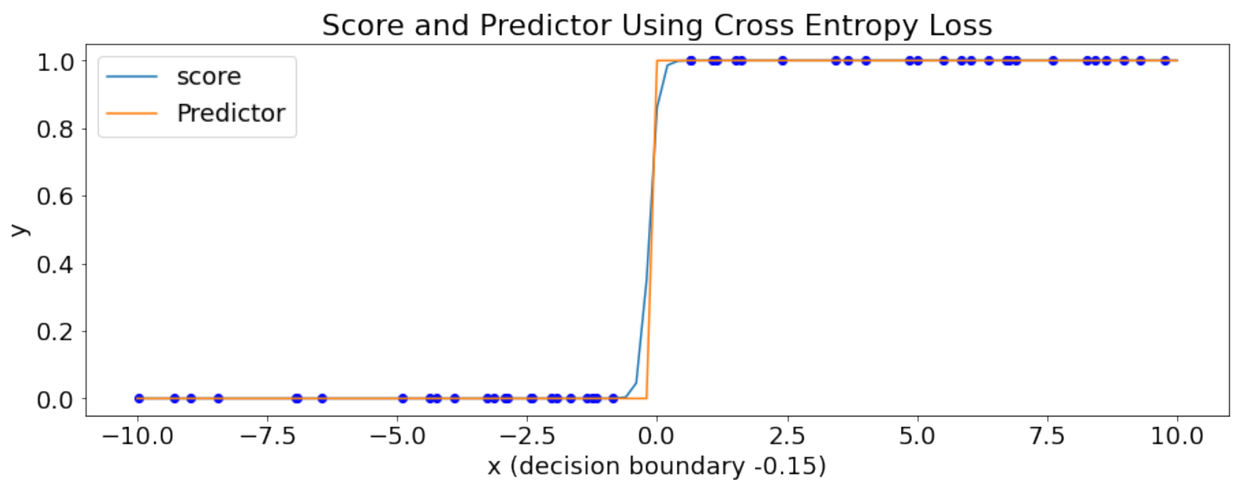
In [705]:
```
file_name_sep = 'data1d_sep.pkl'
retrieve(file_name_sep)
with open(file_name_sep, 'rb') as file:
    t_sep = pickle.load(file)
tx_sep, ty_sep = t_sep['x'], t_sep['y']
```

```
Using previously downloaded file data1d_sep.pkl
```

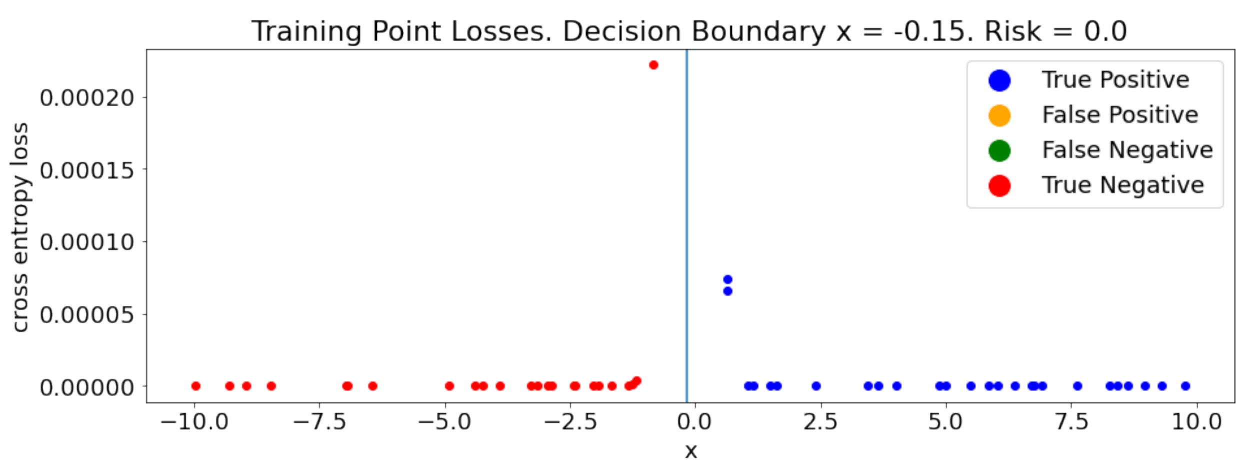
In [706]:
```
v_hat_ce_sep = train(tx_sep, ty_sep)
print(v_hat_ce_sep)
```

```
     fun: 7.372209049550898e-06
     jac: array([ 1.76255384e-06, -5.68232025e-06])
 message: 'Optimization terminated successfully.'
    nfev: 120
     nit: 7
    njev: 40
  status: 0
 success: True
       x: array([ 1.82222152, 12.15671404])
```

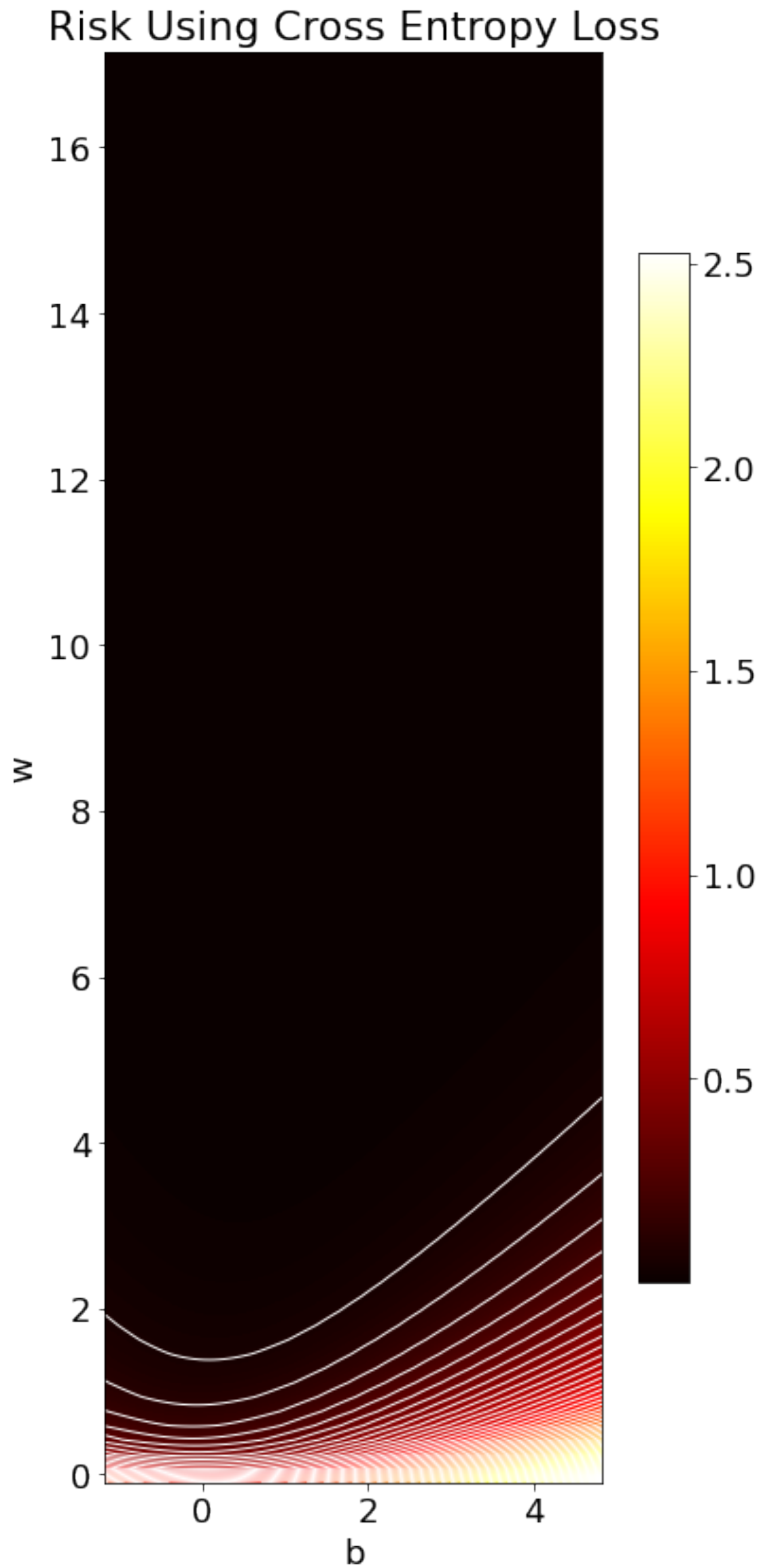
In [707]:
```
plot_score(tx_sep, ty_sep, v_hat_ce_sep['x'], ce_name, x_bound, n_poir
```



In [708]:
```
plot_losses(tx_sep, ty_sep, v_hat_ce_sep['x'], cross_entropy, ce_name,
```



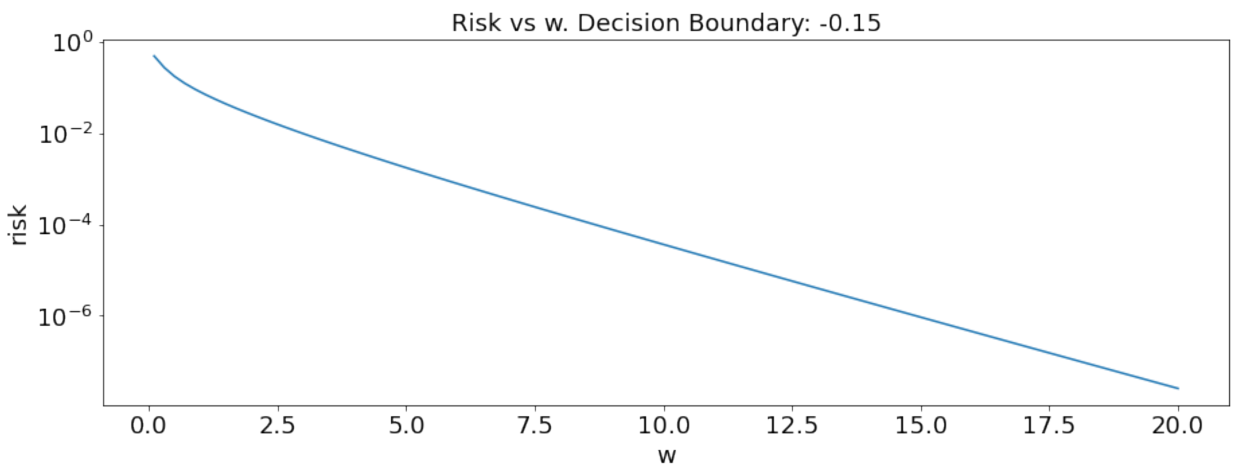
In [709]:
```
plot_contours(tx_sep, ty_sep, v_hat_ce_sep['x'], cross_entropy, ce_nam
```



Problem 1.11

In [710]:
```
def decision_boundary(v):
    return -v[0]/v[1]
```

In [711]:
```
db_hat = decision_boundary(v_hat_ce_sep['x'])
slopes = np.linspace(0.1,20,n_points)
bs = slopes*-db_hat
rs = []
for i in range(len(slopes)):
    v = (bs[i],slopes[i])
    rs.append(risk(v,tx_sep,ty_sep,cross_entropy))
plt.figure(figsize = (15,5))
plt.semilogy(slopes,rs)
plt.xlabel('w', fontsize=18)
plt.ylabel('risk', fontsize=18)
plt.title('Risk vs w. Decision Boundary: '
          + str(round(db_hat,3)), fontsize = 18)
plt.show()
```



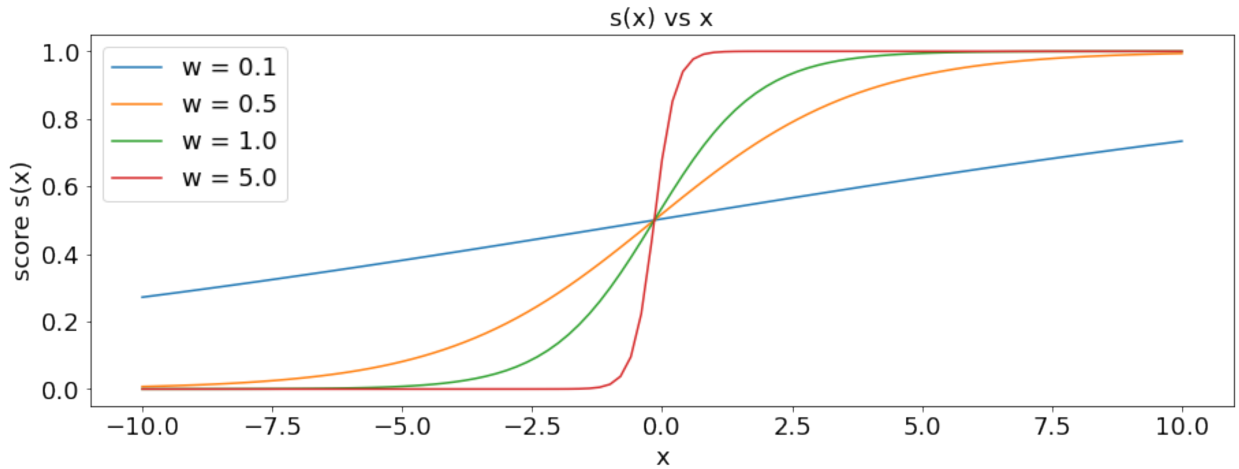
The risk appears to at least monotonically decreases as w increases.

Problem 1.12 (Exam Style)

Some plausible reasons why the algorithm in function train stopped and returned a value even though risk is at least monotonically decreasing may be because (1) the optimization algorithm is written to stop when it observes very tiny changes of risk or very tiny changes in w. (2) the algorithm may be written to stop when it experiences a set number of max iterations.

Problem 1.13

In [712]:
```
plt.figure(figsize = (15,5))
_x = np.linspace(-10,10,n_points)
_w = np.array([0.1,0.5,1.,5.])
_b = _w*-db_hat
for i in range(len(_w)):
    v = (_b[i],_w[i])
    plt.plot(_x, score(_x, v), label = 'w = '+str(_w[i]))
plt.xlabel('x', fontsize=18)
plt.ylabel('score s(x)', fontsize=18)
plt.title('s(x) vs x', fontsize = 18)
plt.legend()
plt.show()
```



The score function increases faster on the positive side (vice versa decreases faster on the negative side) and reaches the limites faster when x increases as a result of increases in w, getting closer to a step function.

Problem 1.14 (Exam Style)

$$b = -x_0 w$$

$$a(x) = w(x - x_0)$$

$$l(x, 1) = -log(p)$$

$$l(x, 0) = -log(1 - p)$$

$$\text{When } y = 0, x - x_0 < 0 :$$
$$l(x, 0) = -log(1 - \frac{1}{1 + e^{-a}}) = log(1 + e^{a})$$
$$lim_{w->\infty} a = -\infty, lim_{w->\infty} l(x, 0) = log(1) = 0$$

$$\text{When } y = 1, x - x_0 > 0 :$$
$$l(x, 1) = log(1 + e^{-a})$$
$$lim_{w->\infty} a = \infty, lim_{w->\infty} l(x, 1) = log(1) = 0$$

Problem 1.15

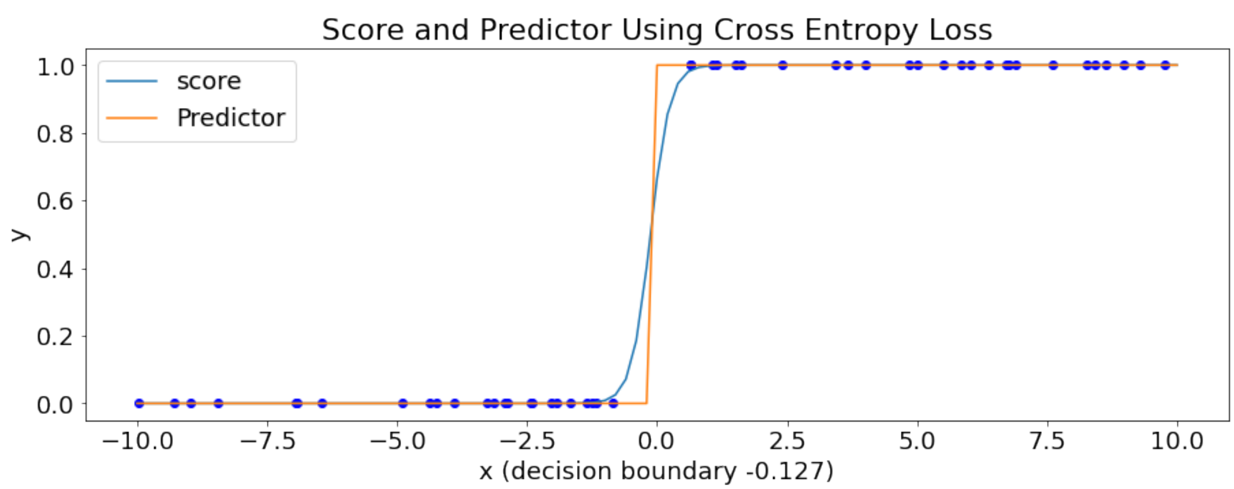
In [713]:
```
def reg_risk(v,x,y,loss, mu):
    return risk(v,x,y,loss) + mu*np.linalg.norm(v)
```

In [714]:
```
def train_reg(x, y, loss = cross_entropy, mu=1.e-3):
    x_0 = np.array([0,0])
    return optimize.minimize(reg_risk, x_0,
                             args=(x,y,loss, mu), method='CG')
```

In [715]:
```
v_hat_train_reg = train_reg(tx_sep, ty_sep)
print(v_hat_train_reg)
```

```
     fun: 0.006711895596208915
     jac: array([ 3.59519618e-06, -3.36387893e-06])
 message: 'Optimization terminated successfully.'
    nfev: 102
     nit: 7
    njev: 34
  status: 0
 success: True
       x: array([0.68743076, 5.41742621])
```

In [716]:
```
plot_score(tx_sep, ty_sep, v_hat_train_reg['x'], ce_name, x_bound, n_p
```



The decision boundary of the regularized predictor is slightly closer to the origin (-.127) compared to that of the non-regularized predictor (0.15). This doesn't change how the points are categorized, however. But looking at the score function curve in the plot above, points closer to the decision boundary are assigned a higher score which leads to it probably getting assigned a higher risk, which would probably be slightly better at handling datasets it has never seen (when dataset is no longer linearly separable).