4.3

4.3

a) Softmax cost function

$$g_2(\tilde{w}) = \sum_{p=1}^{P} \log(1+e^{-y_p \tilde{x}_p^T \tilde{w}})$$

$$\nabla g_2(\tilde{w}) = \sum_{p=1}^{P} \frac{1}{1+e^{-y_p \tilde{x}_p^T \tilde{w}}} e^{-y_p \tilde{x}_p^T \tilde{w}} \cdot (-y_p \tilde{x})$$

$$= \sum_{p=1}^{P} \frac{1}{1+e^{y_p \tilde{x}_p^T \tilde{w}}} (-y_p \tilde{x}) \quad \leftarrow \sigma(t) = \frac{1}{1+e^{-t}}$$

$$= \boxed{-\sum_{p=1}^{P} \sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p \tilde{x}_p}$$

b). $\nabla g(\tilde{w}) = \tilde{X} \bar{r}$

$$-\sum_{p=1}^{P} \sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p \tilde{x}_p = [\tilde{x}_1, \tilde{x}_2, \cdots \tilde{x}_p] \begin{bmatrix} r_1 \\ \vdots \\ r_p \end{bmatrix} = \begin{bmatrix} \sum_{p=1}^{P} x_{p1} r_p \\ \vdots \\ \sum_{p=1}^{P} x_{p(N+1)} r_p \end{bmatrix}_{(N+1)\times 1}$$

$$\Downarrow$$

$$\begin{bmatrix} -\sum_{p=1}^{P} \sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p x_{p1} \\ -\sum_{p=1}^{P} \sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p x_{p2} \\ \vdots \\ -\sum_{p=1}^{P} \sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p x_{p(N+1)} \end{bmatrix}_{(N+1)\times 1} = \begin{bmatrix} \sum_{p=1}^{P} x_{p1} r_p \\ \vdots \\ \sum_{p=1}^{P} x_{p(N+1)} r_p \end{bmatrix}_{(N+1)\times 1}$$

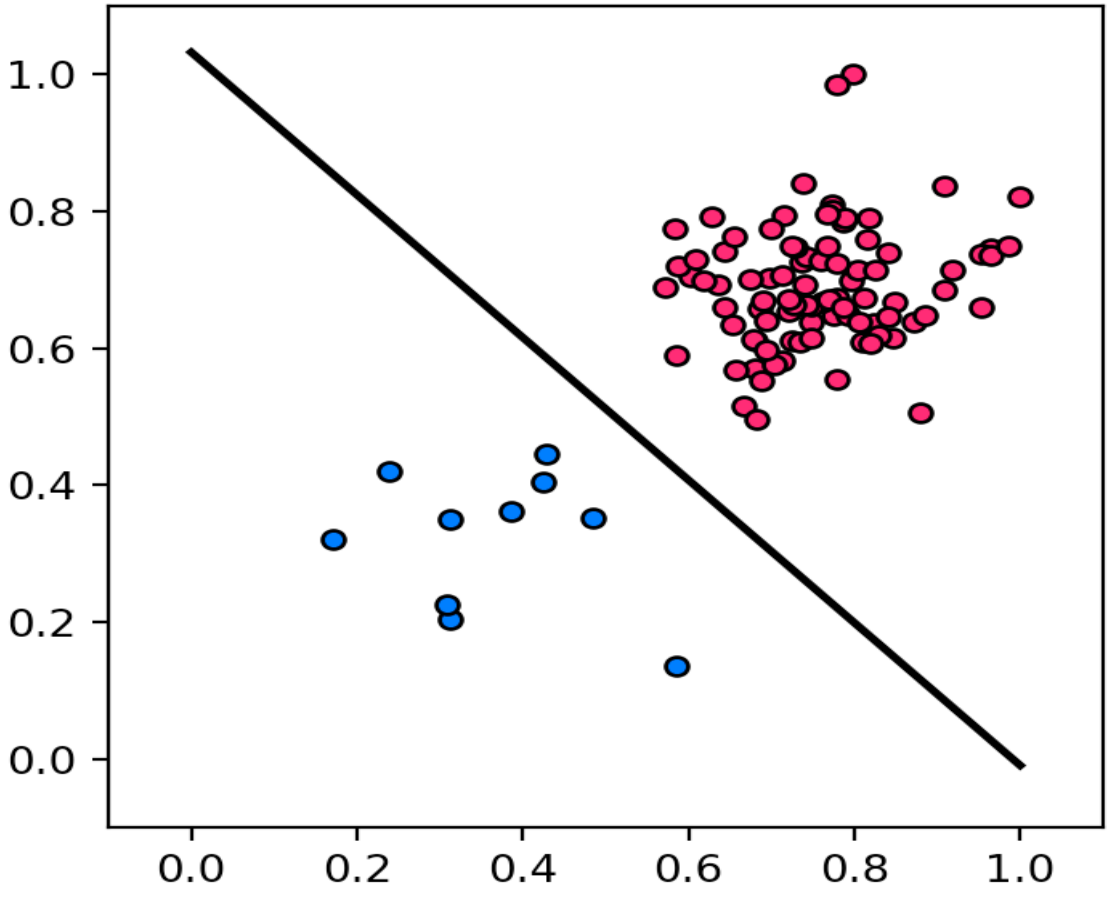
Then we can get $r_p = -\sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p$.

$$\therefore \boxed{\bar{r} = \begin{bmatrix} -\sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_1 \\ \vdots \\ -\sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p \end{bmatrix}_{p\times 1}}$$

(c)

```python
# YOUR CODE GOES HERE - create a gradient descent function for softmax cost/logistic regression
def softmax_grad(X,y):
    alpha = 2
    w0 = array([2,X[1][10],X[2][10]])
    w0.shape = (3,1)
    w = w0
    # x_p = np.hsplit(X,len(X[0]))
    # start gradient descent loop
    grad = 1
    iter = 1
    max_its = 500
    while iter <= max_its:
        # take gradient step
        r = -y * sigmoid(-y * dot(X.T,w))
        grad = np.dot(X,r)
        w = w - alpha*grad
        iter += 1

    # print w
    return w
```



4.5

a) The equation of separating hyperplane

$$b + \bar{x}_p^T \bar{w} = 0 \quad \text{with pair } (b, \bar{w}).$$

when we multiply a positive constant, it becomes

$$Cb + C\bar{x}_p^T \bar{w} = 0 \quad \text{with pair } (C \cdot b,\ C \cdot \bar{w}).$$

$$\boxed{b + \bar{x}_p^T \bar{w} = 0} \Rightarrow \text{it doesn't alter the equation of the separating hyperplane}$$

$$g(b, \bar{w}) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p(b + \bar{x}_p^T \bar{w})}\right)$$

All point is classified correctly

Because $-y_p(b + \bar{x}_p^T \bar{w}) < 0 \overset{C>1}{\Rightarrow} -y_p(C \cdot b + C \cdot \bar{x}_p^T \bar{w}) < -y_p(b + \bar{x}_p^T \bar{w})$

$$\Downarrow$$

$$1 + e^{-y_p(C \cdot b + C \cdot \bar{x}_p^T \bar{w})} < 1 + e^{-y_p(b + \bar{x}_p^T \bar{w})}$$

$$\Downarrow$$

$$\sum_{p=1}^{P} \log\left(1 + e^{-y_p(C \cdot b + C\bar{x}_p^T \bar{w})}\right) < \sum_{p=1}^{P} \log\left(1 + e^{-y_p(b + \bar{x}_p^T \bar{w})}\right)$$

$$\Downarrow$$

$$\boxed{g(C \cdot b, C \cdot \bar{w}) < g(b, \bar{w})}$$

b). This is a problem, because when $b, \bar{w}$ increase same rate, which will not change the hyperplane, but it will decrease the softmax cost (proved in part a))

In fact, this problem is the softmax cost can't present the good or bad of the classification, and it makes no sense.

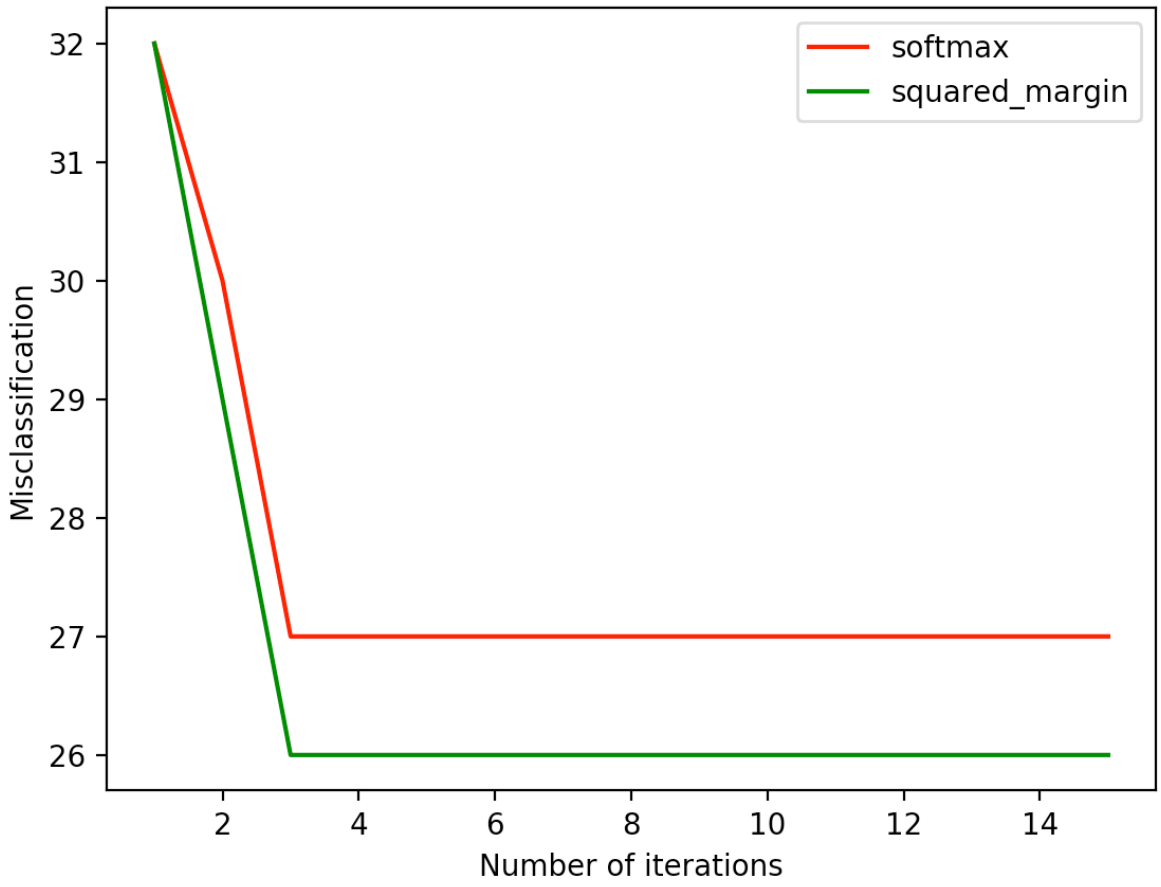
4.9

```
def softmax(X,y):
    # alpha = 0.2
    w0 = array([0,0,0,0,0,0,0,0,0])
    w0.shape = (9,1)
    # w1 = w0
    w1 = w0         # for softmax
    # start gradient descent loop
    # grad = 1
    iter = 1
    max_its = 15
    mismatch_soft_list = []
    # x = np.hsplit(X,len(X[0]))
    while iter <= max_its:
        # take gradient step to softmax
        r = -sigmoid(- y * dot(X.T,w1)) * y
        r_2 = sigmoid(- y * dot(X.T,w1)) * (1 - sigmoid(- y * dot(X.T,w1)))
        grad_soft = np.dot(X,r)
        # grad_soft_2 = dot(dot(X,np.diagflat(r_2)),X.T)
        grad_soft_2 = dot(dot(X,diagflat(r_2)),X.T)
        w1 = w1 - dot(np.linalg.pinv(grad_soft_2),grad_soft)
        soft_mis = count_mis(X,y,w1)
        mismatch_soft_list.append(soft_mis)

        iter += 1
    return mismatch_soft_list
```

```
def square_margin(X,y):
    # alpha = 0.2
    w0 = array([0,0,0,0,0,0,0,0,0])
    w0.shape = (9,1)
    # w1 = w0
    w2 = w0         # for square
    # start gradient descent loop
    # grad = 1
    iter = 1
    max_its = 15
    mismatch_square_list = []
    # x = np.hsplit(X,len(X[0]))
    while iter <= max_its:
        # print grad_soft
        # print grad_soft_2
        # take gradient step to square
        grad_square = -2 * dot((X * y.T), np.maximum(0, 1 - y * dot(X.T,w2)))
        temp = 1 - y * dot(X.T,w2)
        temp = np.reshape(temp, len(y))          # change into one dimension array
        idx = np.where(temp <= 0)                # find the index of element = 0
        _X = X.T
        _X[idx] = 0                              # do not consider that -y(b+xw) < 0
        grad_square_2 = 2 * dot(_X.T, _X)
        w2 = w2 - dot(np.linalg.pinv(grad_square_2),grad_square)
        square_mis = count_mis(X,y,w2)
        mismatch_square_list.append(square_mis)

        iter += 1
    return mismatch_square_list
```



Comparing with the softmax and square_margin, there is no much difference, and both them perform similarly well.

For more detail, please run the python file in the zip.

4.12

4.12.

cost function:

$$h(b,\vec{w}) = -\sum_{p=1}^{P} \bar{y}_p \log \sigma(b+\vec{x}_p^T\vec{w}) + (1-\bar{y}_p)\log(1-\sigma(b+\vec{x}_p^T\vec{w}))$$

$$= -\sum_{p=1}^{P} \bar{y}_p \log\left(\frac{1}{1+e^{-b-\vec{x}_p^T\vec{w}}}\right) + (1-\bar{y}_p)\log\left(1-\frac{1}{1+e^{-b-\vec{x}_p^T\vec{w}}}\right)$$

$$= -\sum_{p=1}^{P} \bar{y}_p \log\left(\frac{e^{b+\vec{x}_p^T\vec{w}}}{1+e^{b+\vec{x}_p^T\vec{w}}}\right) + (1-\bar{y}_p)\log\left(\frac{1}{1+e^{b+\vec{x}_p^T\vec{w}}}\right)$$

when $y_p = -1$, $\bar{y}_p = 0$.

$$h(b,\vec{w}) = -\sum_{p=1}^{P} \log\left(\frac{1}{1+e^{b+\vec{x}_p^T\vec{w}}}\right)$$

$$= -\sum_{p=1}^{P} \log 1 - \log\left(1+e^{b+\vec{x}_p^T\vec{w}}\right)$$

$$= \sum_{p=1}^{P} \log\left(1+e^{b+\vec{x}_p^T\vec{w}}\right)$$

$$g(b,\vec{w}) = \sum_{p=1}^{P} \log\left(1+e^{-y_p(b+\vec{x}_p^T\vec{w})}\right) \xRightarrow{y_p=-1} \boxed{h(b,\vec{w}) = g(b,\vec{w})}$$

when $y_p = 1$, $\bar{y}_p = 1$

$$h(b,\vec{w}) = -\sum_{p=1}^{P} \log\left(\frac{1}{1+e^{-b-\vec{x}_p^T\vec{w}}}\right)$$

$$= -\sum_{p=1}^{P} \left(\log 1 - \log\left(1+e^{-b-\vec{x}_p^T\vec{w}}\right)\right)$$

$$= \sum_{p=1}^{P} \log\left(1+e^{-b-\vec{x}_p^T\vec{w}}\right)$$

$$g(b,\vec{w}) = \sum_{p=1}^{P} \log\left(1+e^{-y_p(b+\vec{x}_p^T\vec{w})}\right) \xRightarrow{y_p=1} \boxed{h(b,\vec{w}) = g(b,\vec{w})}$$

Therefore, the cross-entropy cost for logistic regression, is equivalent to the softmax cost function.