CSE 446: Homework 2

ayush29f@cs.washington.edu
Ayush Saraf

Problem 1.

(a) False. Since $\hat{w}$ is the optimal value for the model which means it fits really well to the data set - in some cases maybe over-fits. However, a sparse $\hat{w}$ means there are less features being used and the model is not complex. Since this contradicts with an optimally fitting model, the statement is false.

(b) True. If the data is linearly separable and higher values of the weights are not penalized then if we increase all the weights by a factor > 1 then we will have a higher likelihood - therefore as we do gradient descent weights will eventually go to ∞

(c) False. With a smaller value of λ we are penalizing the weights less, which means that the model will fit better on the training data which means higher likelihood. However, as we increase the λ the training likelihood would decrease because now the model is not fitting the data well.

(d) False. As we increase the λ we are penalizing larger values of weights and making the model less complex to fit the training data-set - not over-fit but maybe underfit. This would suggest we are generating an optimal model for the test set somewhere in between but probably getting worse on either ends. This means there is no monotonic behaviour and the $l(\hat{w}, D_{test})$ will not increase all the time.

(e) i

$$l(\boldsymbol{w}, D_{train}) = \sum_j \Big(-(1 - \mathbb{1}_{y^j=1})(w_0 + x_1^j w_1) - ln(1 + e^{w_0 + x_1^j w_1}) \Big)$$

$$l(\boldsymbol{w}, D_{mod}) = \sum_j \Big(-(1 - \mathbb{1}_{y^j=1})(w_0' + x_1^j(w_1' + w_2')) - ln(1 + e^{w_0' + x_1^j(w_1' + w_2')}) \Big)$$

ii $w_0 = w_0'$ and $w_1 = w_1' + w_2'$

iii Yes, it will change. If we use L2 regularization then the loss function would include the l2 value of weights. Since in case of D_{train} L2(weight) $= w_1^2$ and in case of D_{mod} L2(weight) $= w_1'^2 + w_2'^2$ which will change the partials when optimizing.

Problem 2.a

Given,

$$f(x) = \sum_t^T \alpha_t h_t(x)$$

$$H(x) = sgn(f(x))$$

$$\epsilon_{training} = \frac{1}{N}\sum_{j}^{N} 1_{H(x^j)\neq y^j}$$

Therefore, $\epsilon_{training}$ is essentially equal to number of incorrect predictions. And, we know that if $f(x^j) > 0$ then we will predict 1 and -1 otherwise. So $f(x^j)y^j > 0$ for correct predictions and $f(x^j)y^j < 0$ for incorrect predictions.

Now, since $e^{-f(x^j)y^j} < 1$ for correct predictions and $e^{-f(x^j)y^j} > 1$ for incorrect predictions.
Now Let,

$$\delta_1 \geq 1$$

$$\delta_2 \in (0, 1)$$

Therefore,

$$N\epsilon_{training}\delta_1 \geq N\epsilon_{training}$$

Now since $\delta_2 \in (0, 1)$ you can do the following,

$$(N - N\epsilon_{training})\delta_2 + N\epsilon_{training}\delta_1 \geq N\epsilon_{training}$$

Notice that the $(N - N\epsilon_{training})\delta_2 + N\epsilon_{training}\delta_1$ is same as $\sum_{j=1}^{N} e^{-f(x^j)y^j}$ as we explained earlier,

$$N\epsilon_{training} \leq \sum_{j=1}^{N} e^{-f(x^j)y^j}$$

$$\epsilon_{training} \leq \frac{1}{N}\sum_{j=1}^{N} e^{-f(x^j)y^j}$$

Hence Proved!

Problem 2.b

Given,

$$f(x) = \sum_{t}^{T} \alpha_t h_t(x)$$

Starting from left hand side,

$$\frac{1}{N}\sum_{j=1} Ne^{-f(x^j)y^j} = \frac{1}{N}\sum_{j=1}^{N} e^{-\sum_t^T \alpha_t h_t(x^j)y^j}$$

$$= \frac{1}{N}\sum_{j=1}^{N}\prod_{t}^{T} e^{-\alpha_t h_t(x^j)y^j}$$

Given,

$$w_j^{(t+1)} = \frac{w_j^{(t)} e^{-\alpha_t h_t(x^j) y^j}}{Z^{(t)}}$$

Therefore,

$$\frac{1}{N}\sum_{j=1} N e^{-f(x^j)y^j} = \frac{1}{N}\sum_{j=1}^{N}\prod_{t}^{T}\frac{w_j^{(t+1)} Z^{(t)}}{w_j^{(t)}}$$

$$= \frac{1}{N}\sum_{j=1}^{N}\frac{w_j^{(T+1)}}{w_j^{(1)}}\prod_{t}^{T} Z^{(t)}$$

We know $w_j^{(1)} = \frac{1}{N}$ as an initial condition for normalized weights.

$$= \frac{1}{N}\sum_{j=1}^{N}\frac{w_j^{(T+1)}}{\frac{1}{N}}\prod_{t}^{T} Z^{(t)}$$

$$= \sum_{j=1}^{N} w_j^{(T+1)}\prod_{t}^{T} Z^{(t)}$$

Since weights are normalized $\sum_{j=1}^{N} w_j^{(T+1)} = 1$

$$\frac{1}{N}\sum_{j=1} N e^{-f(x^j)y^j} = \prod_{t}^{T} Z^{(t)}$$

Hence Proved!

Problem 2.c

---

i Given,

$$Z_y = (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$$

Take the derivative w.r.t α_t,

$$\frac{\partial Z_y}{\partial \alpha_t} = -(1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$$

Set the derivative to 0,

$$-(1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 0$$

$$(1-\epsilon_t)e^{-\alpha_t} = \epsilon_t e^{\alpha_t}$$

$$\frac{1-\epsilon_t}{\epsilon_t} = e^{2\alpha_t}$$

$$\alpha_t = \frac{1}{2} ln\Big(\frac{1-\epsilon_t}{\epsilon_t}\Big)$$

Therefore,

$$Z_y^{opt} = (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

$$Z_y^{opt} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

Hence Proved!

ii Given,

$$Z_y = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

$$\epsilon_t = \frac{1}{2} - \gamma_t$$

Therefore,

$$Z_y = 2\sqrt{(\frac{1}{2}-\gamma_t)(1-(\frac{1}{2}-\gamma_t))}$$

$$Z_y = 2\sqrt{\frac{1}{4}-\gamma_t^2}$$

$$Z_y = \sqrt{1-(2\gamma_t)^2}$$

$$ln(Z_y) = \frac{1}{2}ln(1-(2\gamma_t)^2)$$

Now since, we know Z_t is a real number and $\gamma_t > 0$; $0 < \gamma_t \leq 1/2$, and by using the fact that $log(1-x) \leq -x$ for $0 \leq x \leq 1$

$$2ln(Z_y) = ln(1-(2\gamma_t)^2) \leq -(2\gamma_t)^2$$

$$ln(Z_y) \leq -2\gamma_t^2$$

$$Z_y \leq e^{-2\gamma_t^2}$$

Hence Proved!

iii Given,

$$\epsilon_{training} \leq \prod_{t=1}^{T} Z_y \leq e^{-2\sum_{t=1}^{T}\gamma_t^2}$$

$$\gamma_t \geq \gamma$$

Therefore,

$$\gamma_t^2 \geq \gamma^2$$

$$\sum_{t=1}^{T}\gamma_t^2 \geq T\gamma^2$$

$$-2\sum_{t=1}^{T}\gamma_t^2 \leq -2T\gamma^2$$

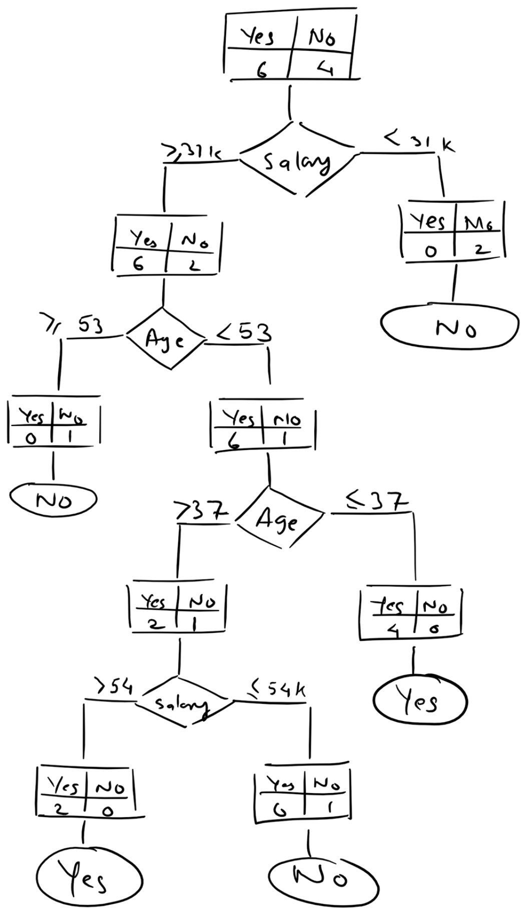
$$e^{-2\sum_{t=1}^{T}\gamma_t^2} \leq e^{-2T\gamma^2}$$

$$\epsilon_{training} \leq \prod_{t=1}^{T} Z_y \leq e^{-2\sum_{t=1}^{T}\gamma_t^2} \leq e^{-2T\gamma^2}$$

$$\epsilon_{training} \leq e^{-2T\gamma^2}$$

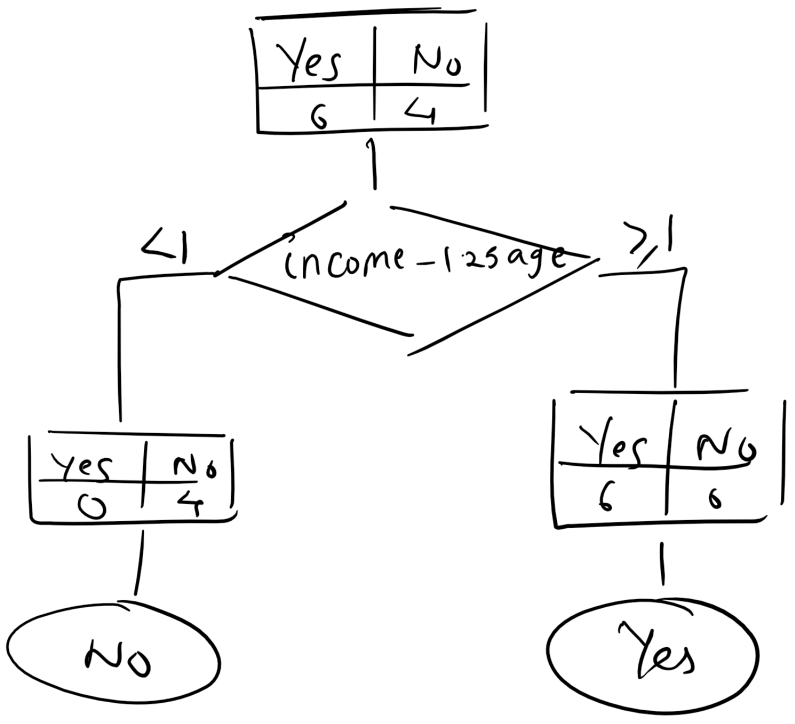
Hence Proved!

Problem 3.a



classification error = 0
depth = 4

Problem 3.b



$\alpha = -1.25 \quad \beta = \underline{1}$

depth = 1

Correction error = 0

Problem 3.c

Advantages

1. Multivariate allows the ability to generate more complex decision boundaries v/s orthogonal boundaries in uni-variate decision trees.
2. Multivariate decision trees are generally relatively less depth than uni-variate decision trees.

Disadvantages

1. Multivariate decision trees are relatively harder to find since you might have to use regression to find boundaries which is computationally expensive
2. Multivariate decision trees don't work very well with discrete value features unlike univariate decision trees

Problem 4.2.a

For Linear Regression,

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + 2\eta\Big(h_j(x^t)(y^t - \sum_i (h_i(x^t)w_i^{(t)}))\Big)$$

For Logistic Regression,

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta\Big(h_j(x^t)(y^t - sigmoid(\sum_i (h_i(x^t)w_i^{(t)})))\Big)$$

Problem 4.2.b and 4.2.c

Please turn to next page for all the plots

Problem 4: Linear & Logistic Regression

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from script import load_data, logistic_loss, linear_loss, train

X_train, y_train, features = load_data('dataset/train.csv')
X_test, y_test, _ = load_data('dataset/test.csv')

X_train = np.append(X_train, np.ones((X_train.shape[0], 1)), axis=1)
X_test = np.append(X_test, np.ones((X_test.shape[0], 1)), axis=1)

X = {'train': X_train, 'test': X_test}
y = {'train': y_train, 'test': y_test}

N, D = X_train.shape
etas = [0.8, 1e-3, 1e-5]
```

```
/Users/ayush/anaconda2/lib/python2.7/site-packages/matplotlib/font_mana
ger.py:273: UserWarning: Matplotlib is building the font cache using fc
-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. T
his may take a moment.')
```

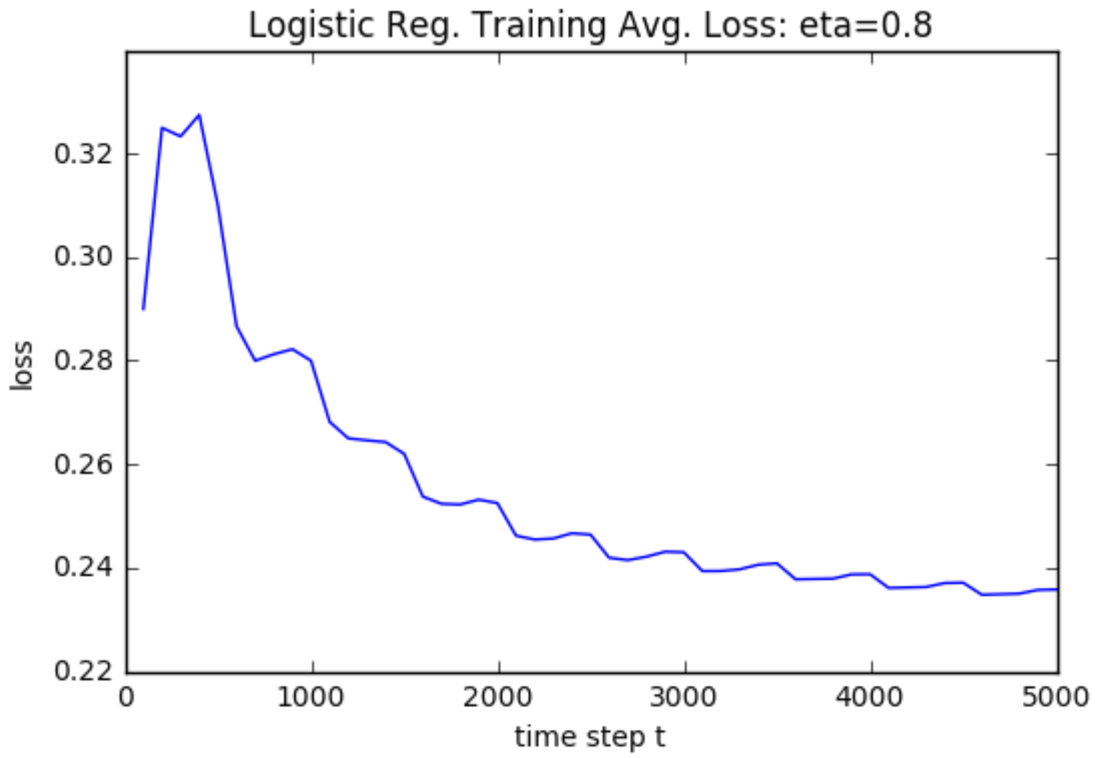
Logistic Regression

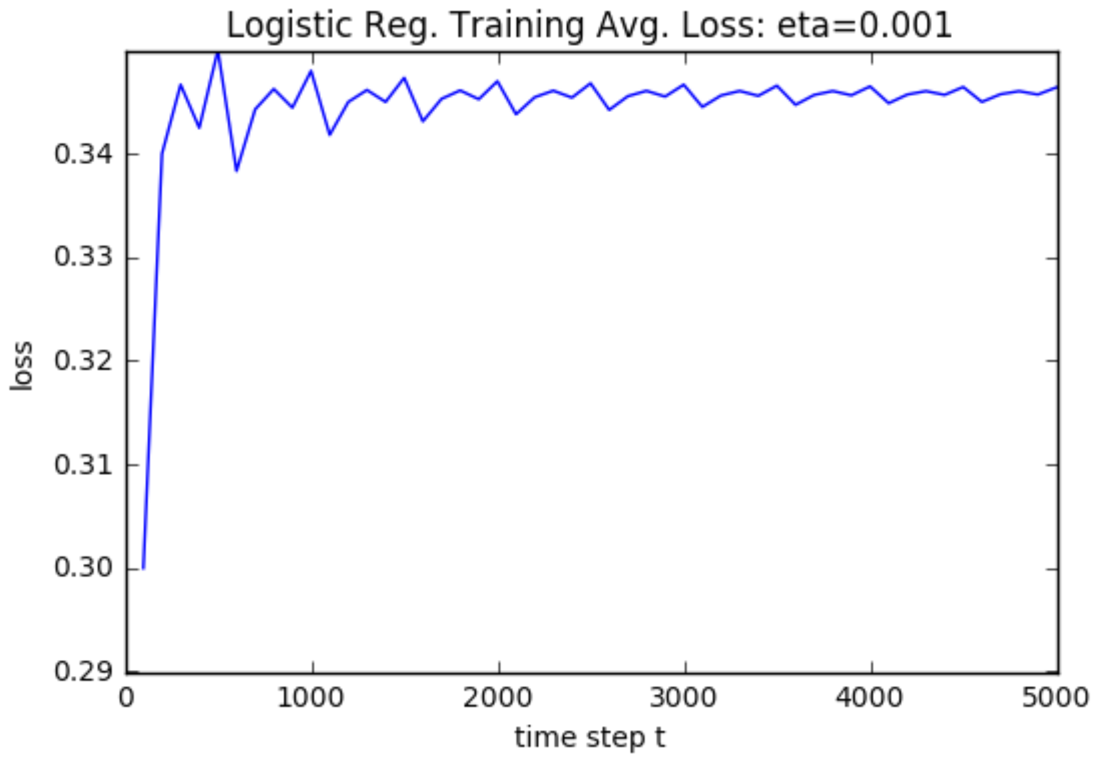
In [2]:
```
logistic_logs = {}
for eta in etas:
    logistic_logs[eta] = {}
    train(X, y, np.zeros(D), logistic_loss, eta, 10 * N, logistic_logs[e
ta])
```

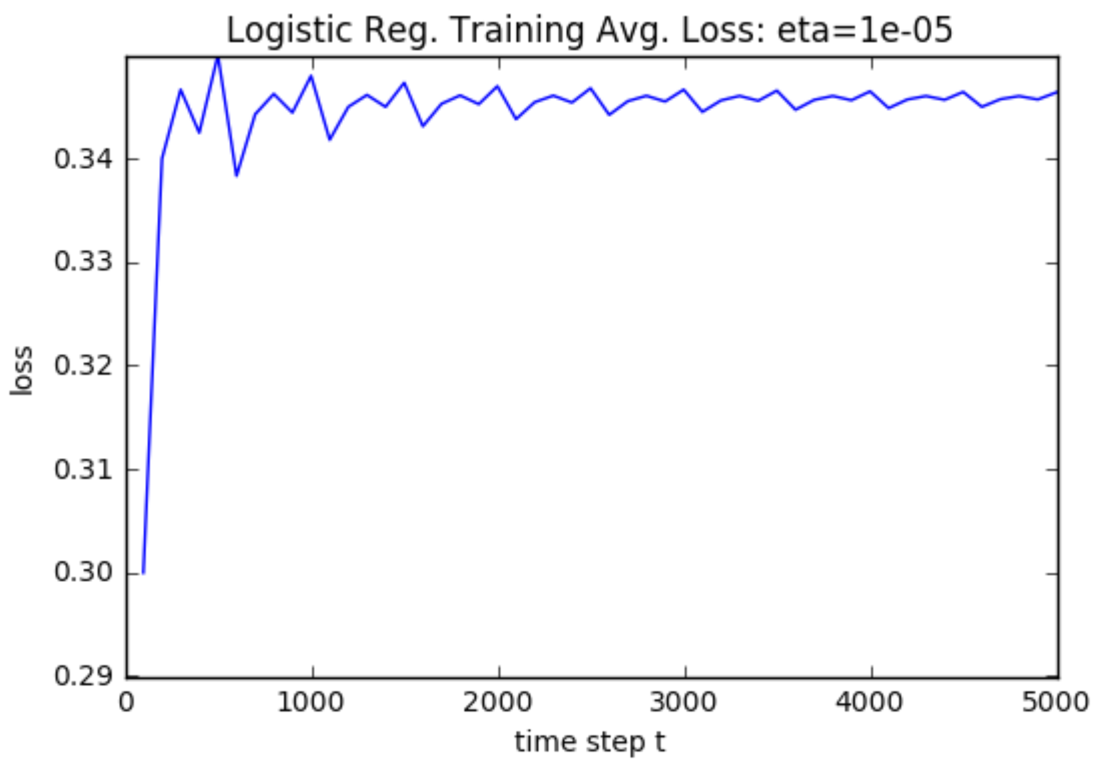
4.2.b.i

Plot the average loss L as a function of the number of steps T

In [3]:
```
for eta in etas:
    plt.plot(logistic_logs[eta]['train_losses']['x'], logistic_logs[eta]
['train_losses']['y'])
    plt.title('Logistic Reg. Training Avg. Loss: eta=' + str(eta))
    plt.xlabel('time step t')
    plt.ylabel('loss')
    plt.show()
```


Logistic Reg. Training Avg. Loss: eta=0.8
loss
time step t
0.22
0.24
0.26
0.28
0.30
0.32
0
1000
2000
3000
4000
5000


Logistic Reg. Training Avg. Loss: eta=0.001
loss
time step t
0.29
0.30
0.31
0.32
0.33
0.34
0
1000
2000
3000
4000
5000

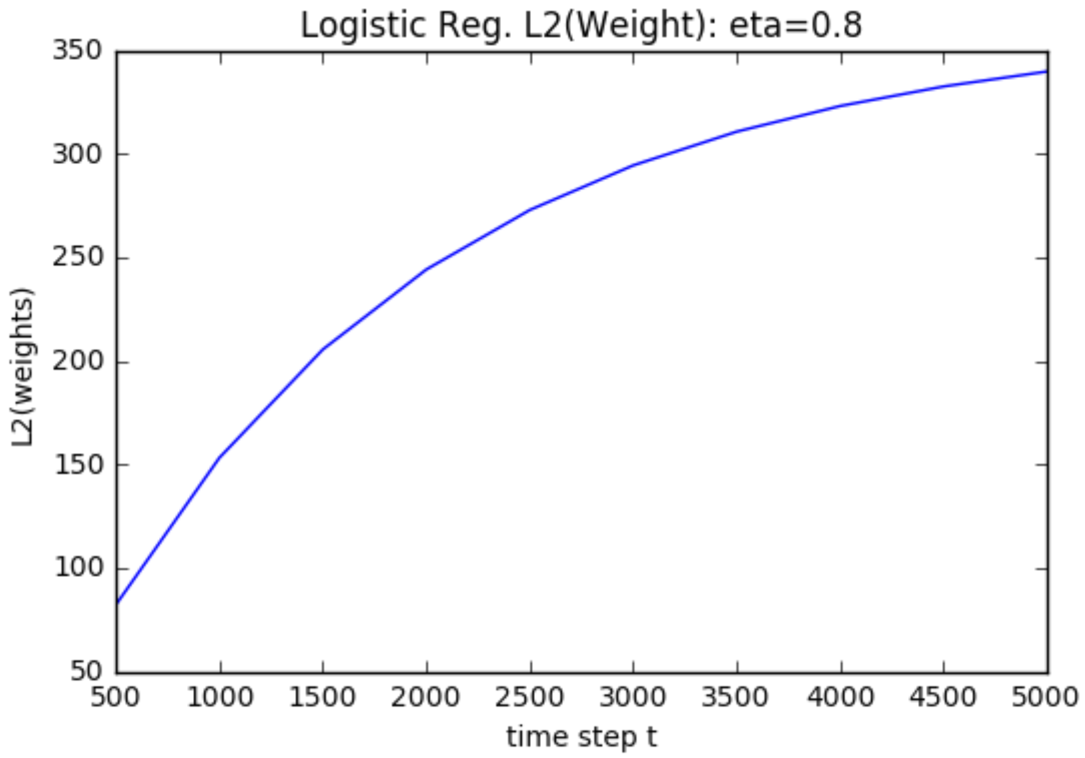


4.2.b.ii

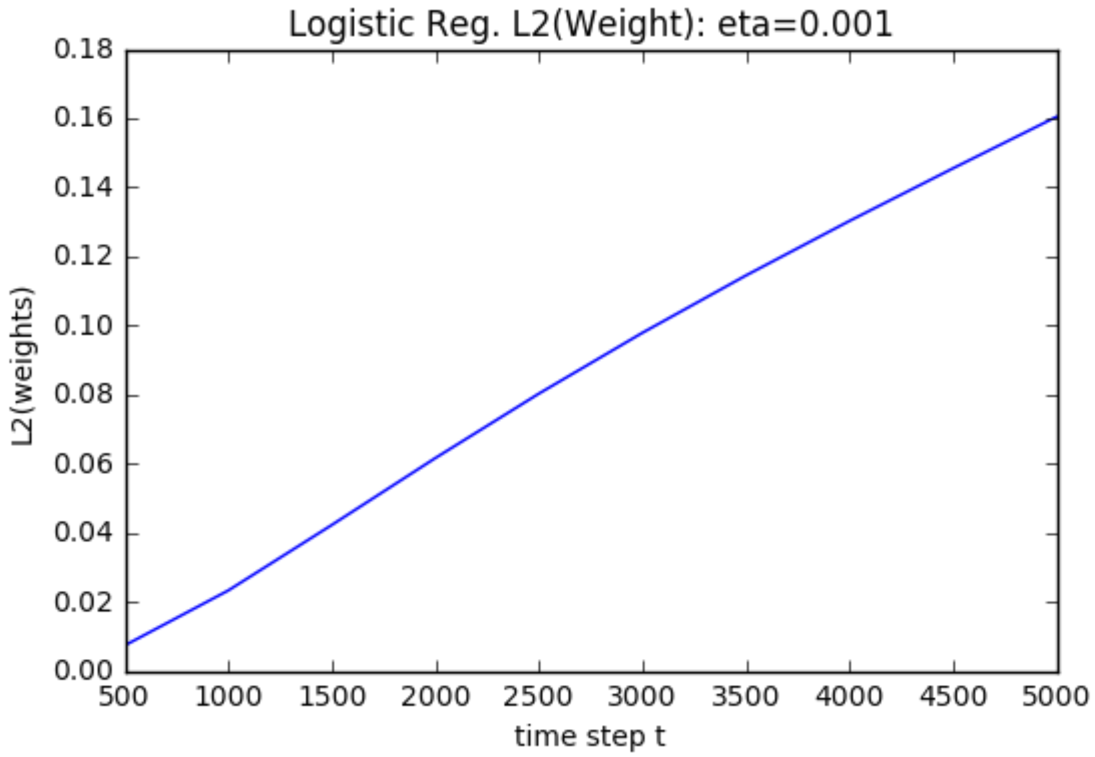
Report the l2 norm of the weights at the end of the pass. In general, l2 regularization can be useful if the norm of these weights grows very large over the course of model training. In this assignment, however, you are not expected to perform l2 regularization.

In [4]:

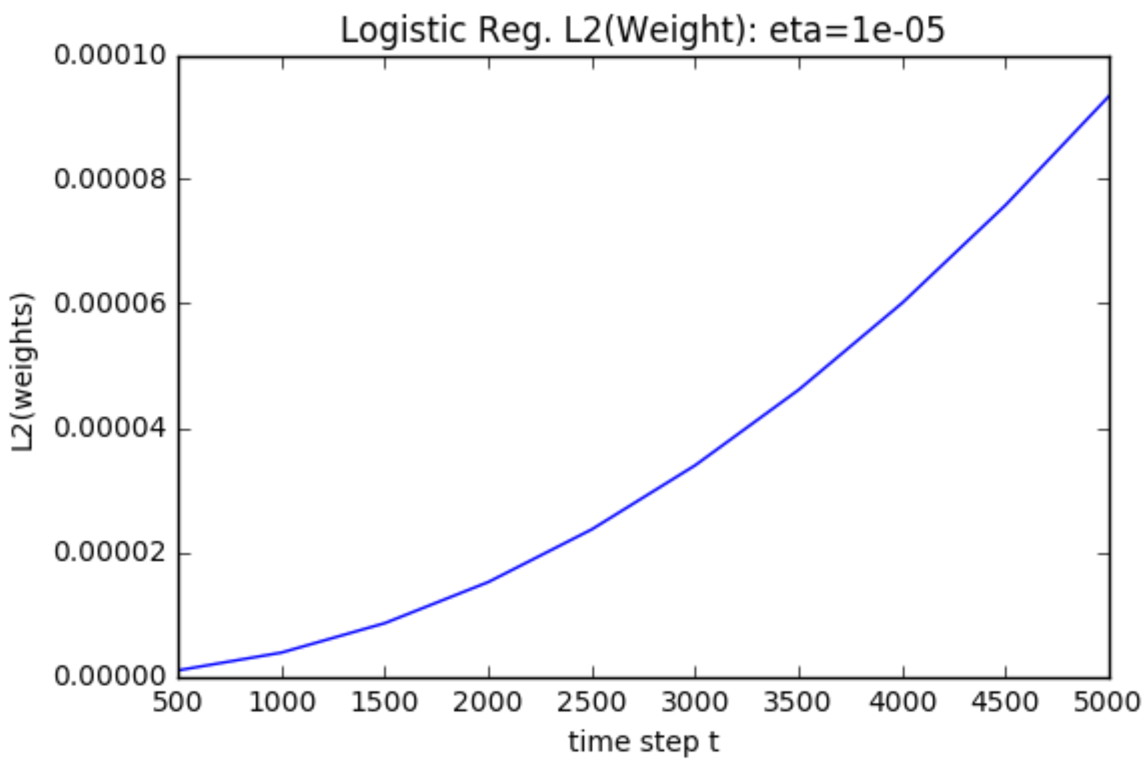
```
for eta in etas:
    plt.plot(logistic_logs[eta]['l2s']['x'], logistic_logs[eta]['l2s']
['y'])
    plt.title('Logistic Reg. L2(Weight): eta=' + str(eta))
    plt.xlabel('time step t')
    plt.ylabel('L2(weights)')
    plt.show()
```



Logistic Reg. L2(Weight): eta=0.8
L2(weights)
time step t
350
300
250
200
150
100
50
500
1000
1500
2000
2500
3000
3500
4000
4500
5000



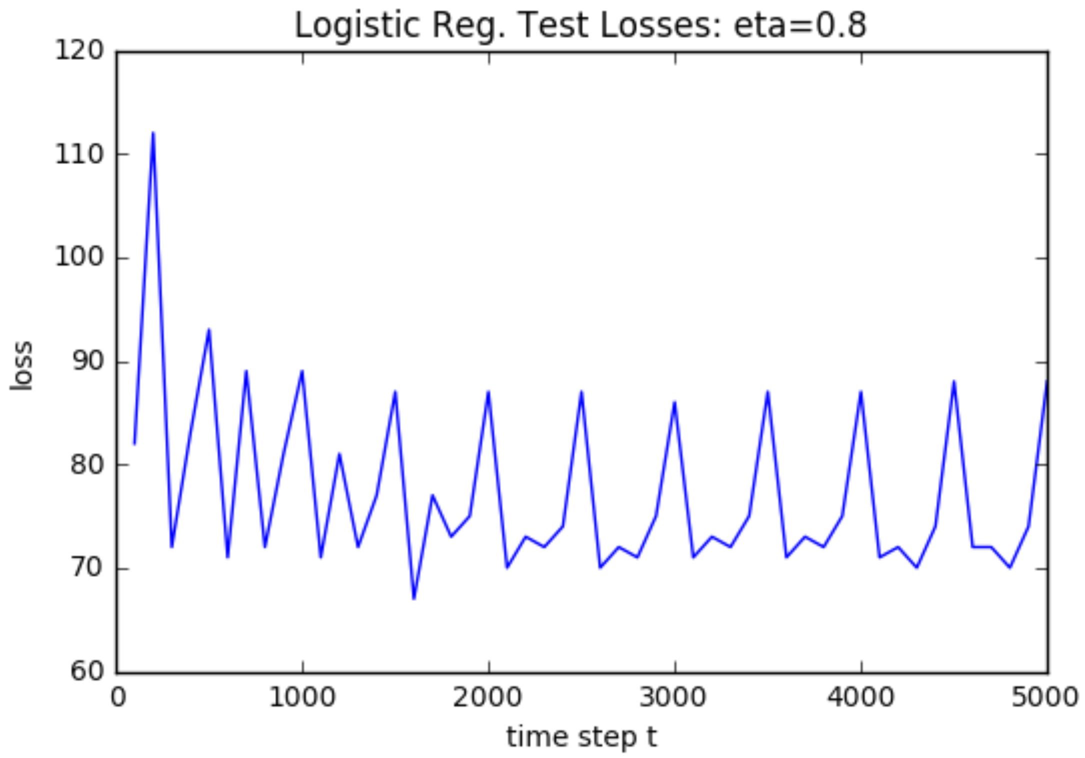
Logistic Reg. L2(Weight): eta=0.001
L2(weights)
time step t
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

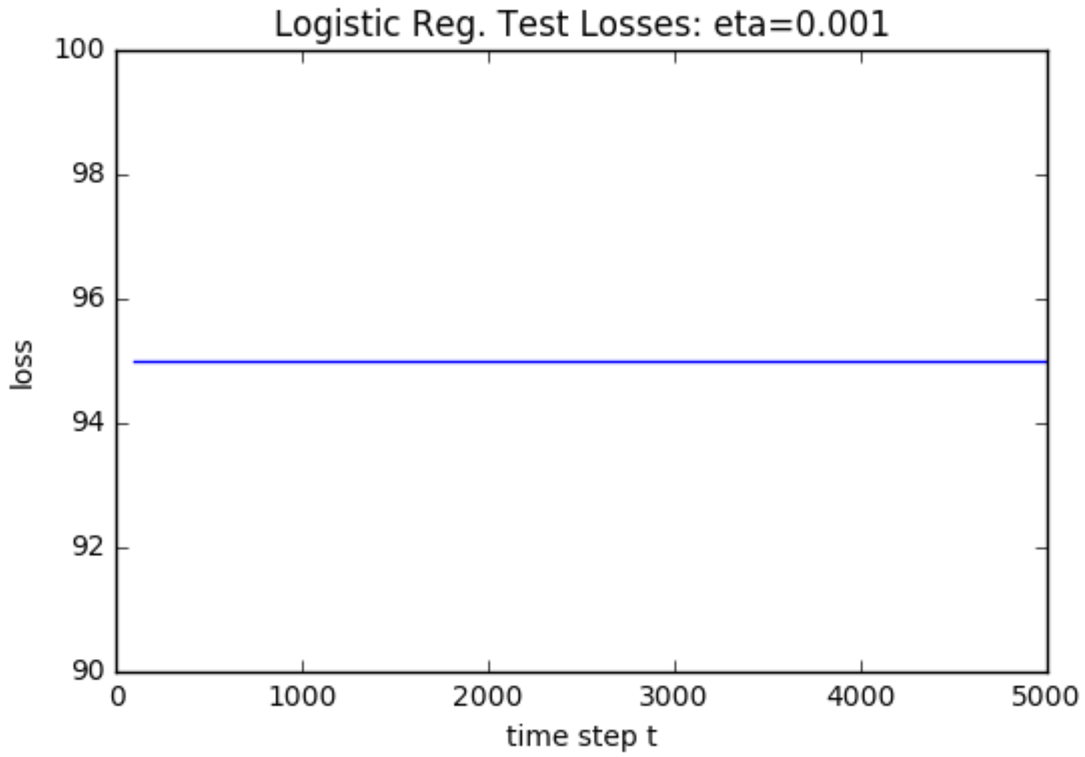


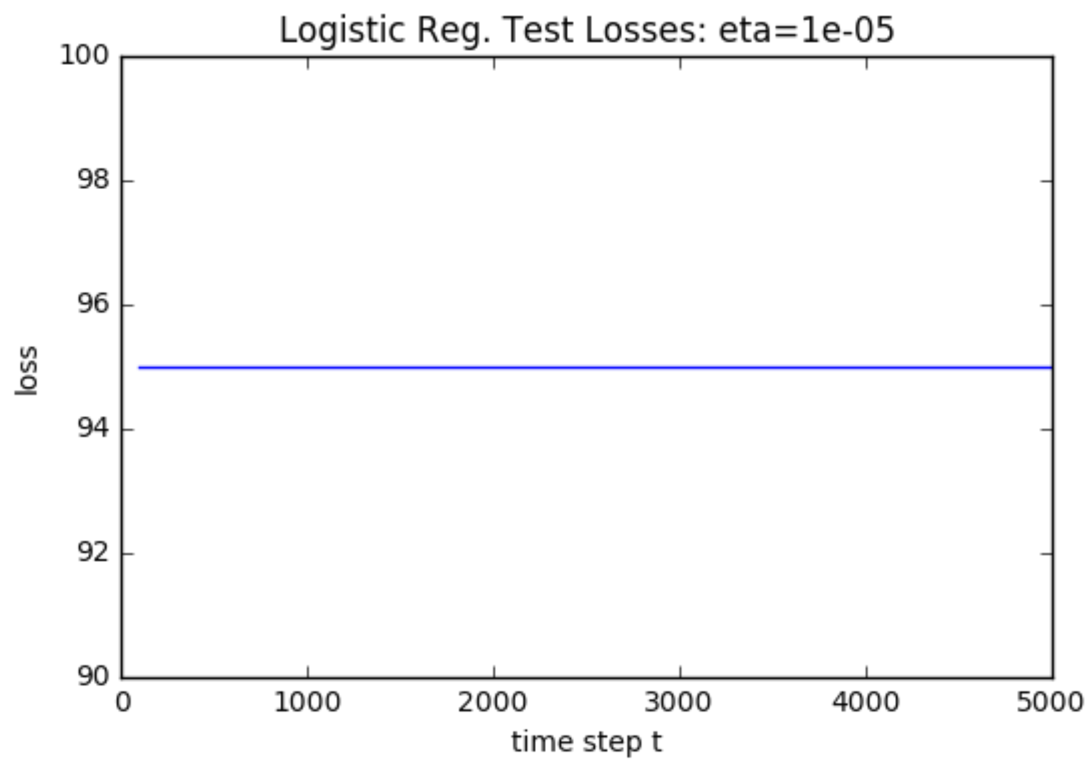
4.2.b.iii

Use the model weights to predict whether each patient in the test set has diabetes, for every 100 steps. Recall that "test label.txt" contains the labels for the test data. Report the SSE (sum of squared errors) of your prediction. Make sure to use the SSE for the 0/1 prediction.

```
In [5]: for eta in etas:
            plt.plot(logistic_logs[eta]['test_losses']['x'], logistic_logs[eta]
        ['test_losses']['y'])
            plt.title('Logistic Reg. Test Losses: eta=' + str(eta))
            plt.xlabel('time step t')
            plt.ylabel('loss')
            plt.show()
```


Logistic Reg. Test Losses: eta=0.8
120
110
100
90
80
70
60
loss
0
1000
2000
3000
4000
5000
time step t


Logistic Reg. Test Losses: eta=0.001
100
98
96
94
92
90
loss
0
1000
2000
3000
4000
5000
time step t



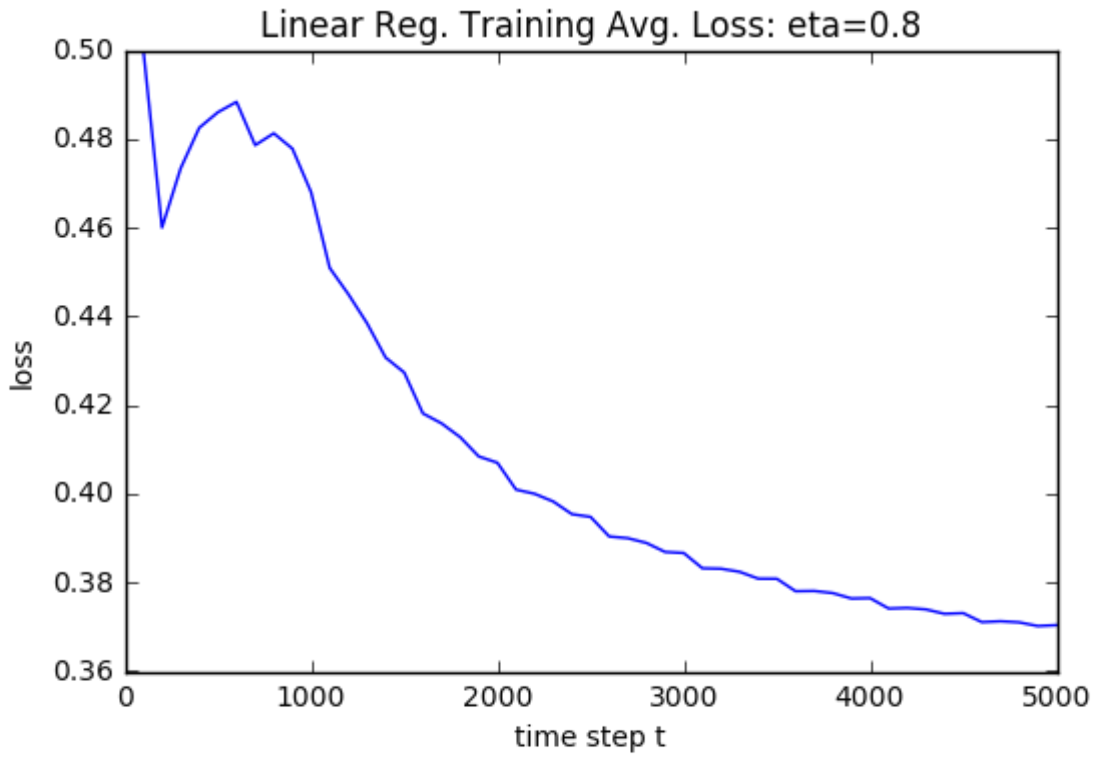
Linear Regression

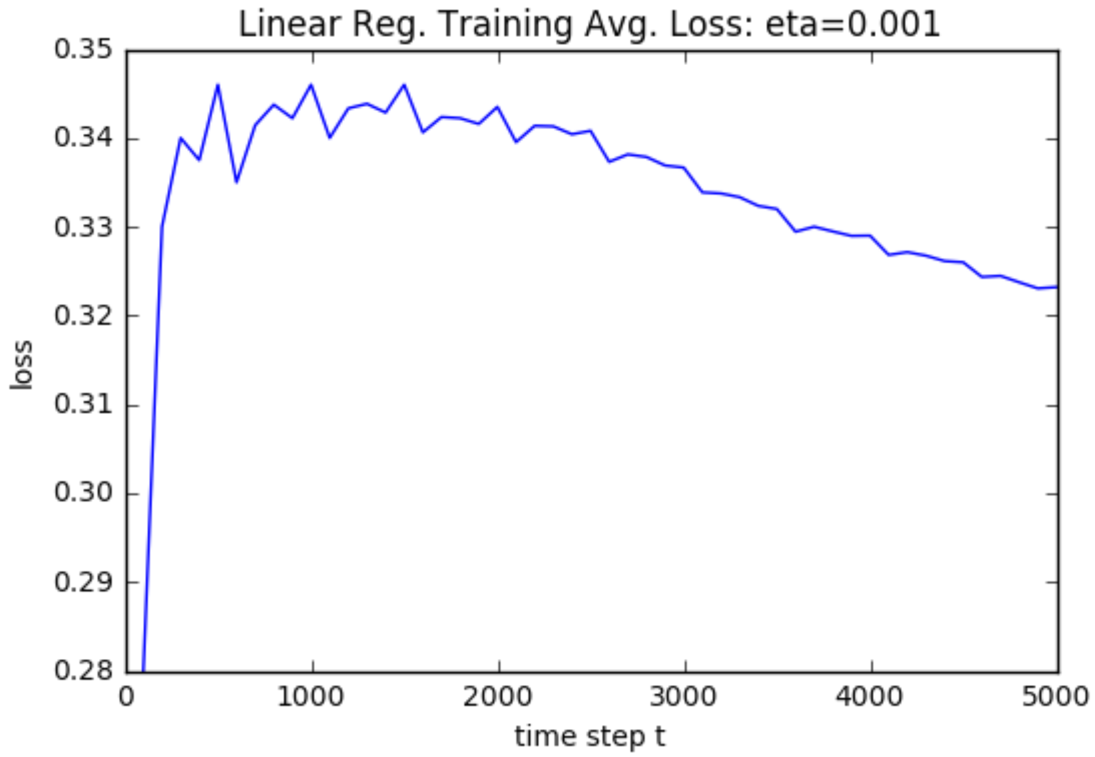
```
In [6]: linear_logs = {}
        for eta in etas:
            linear_logs[eta] = {}
            train(X, y, np.zeros(D), linear_loss, eta, 10 * N, linear_logs[eta])
```

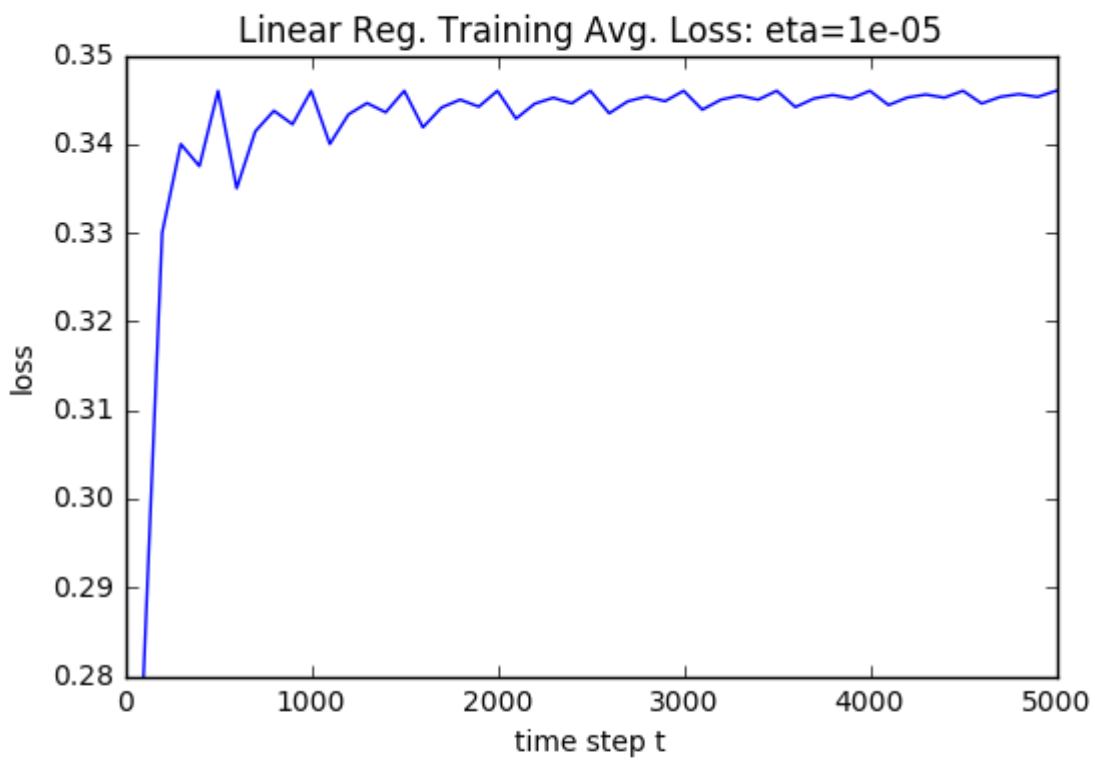
4.2.b.i

Plot the average loss L as a function of the number of steps T

In [7]:
```
for eta in etas:
    plt.plot(linear_logs[eta]['train_losses']['x'], linear_logs[eta]['tr
ain_losses']['y'])
    plt.title('Linear Reg. Training Avg. Loss: eta=' + str(eta))
    plt.xlabel('time step t')
    plt.ylabel('loss')
    plt.show()
```


Linear Reg. Training Avg. Loss: eta=0.8
loss
time step t
0.50
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0
1000
2000
3000
4000
5000


Linear Reg. Training Avg. Loss: eta=0.001
loss
time step t
0.35
0.34
0.33
0.32
0.31
0.30
0.29
0.28
0
1000
2000
3000
4000
5000

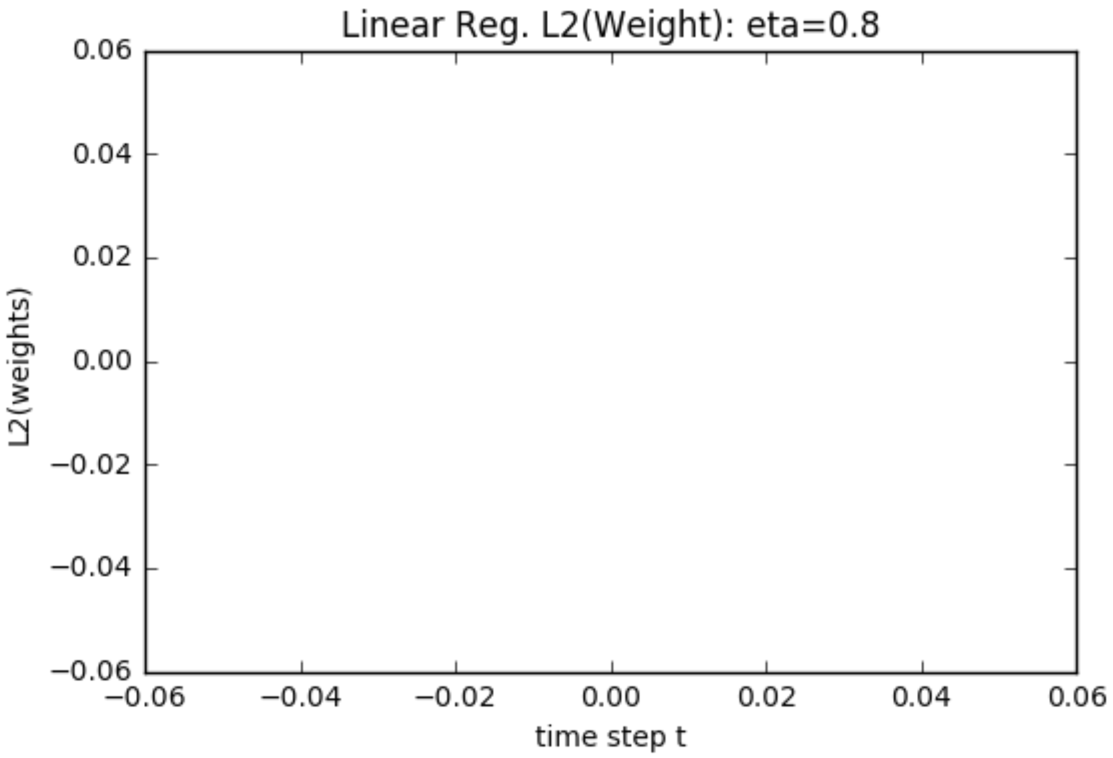


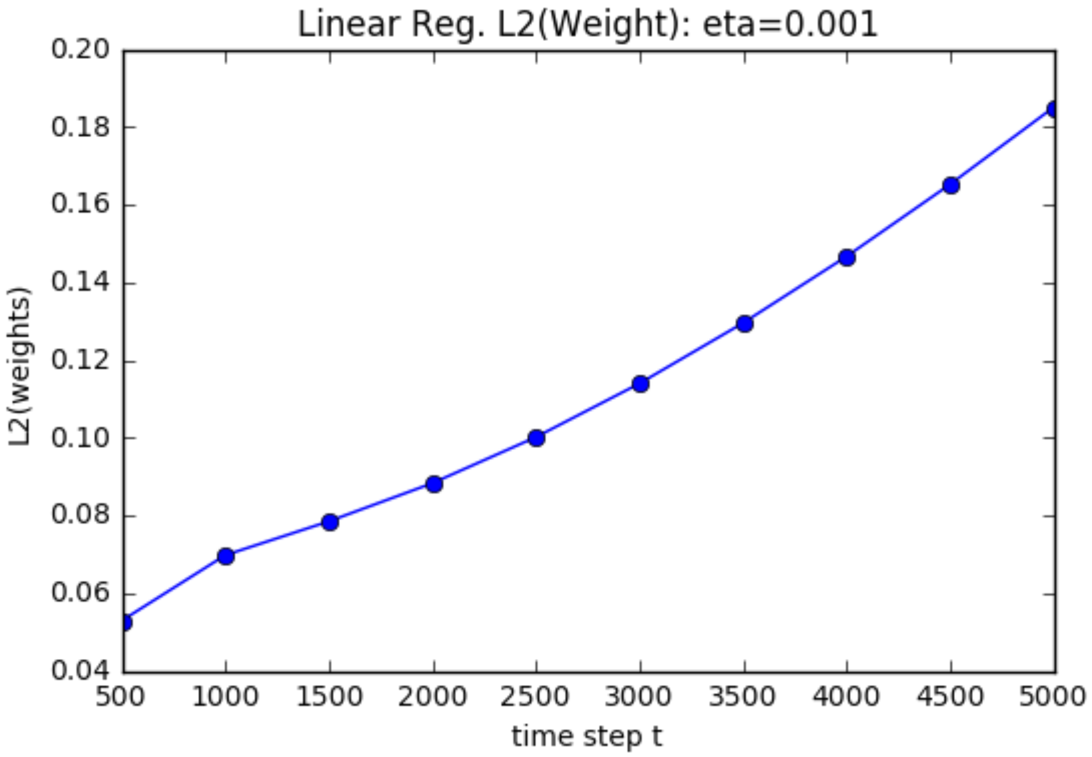
4.2.b.ii

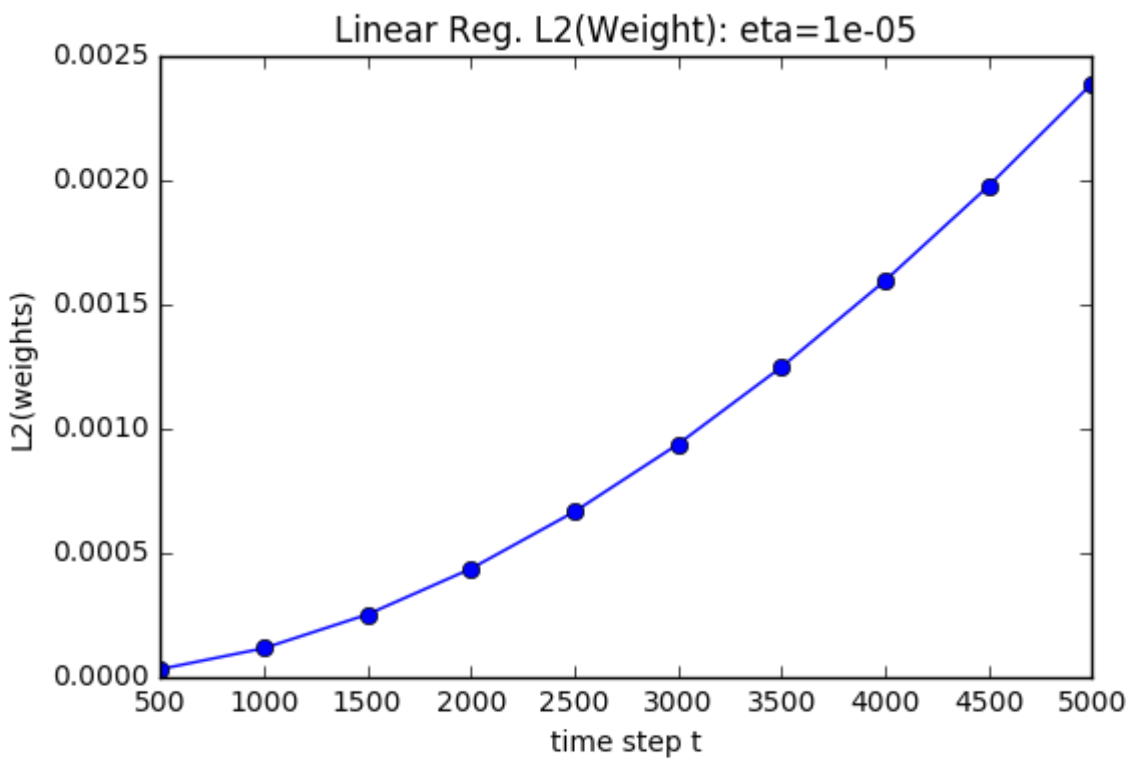
Report the l2 norm of the weights at the end of the pass. In general, l2 regularization can be useful if the norm of these weights grows very large over the course of model training. In this assignment, however, you are not expected to perform l2 regularization.

In case of et=0.8 the weights overshoot the maximum possible number and end up in infty or nan and thats why the L2 plot is empty

```
In [8]: for eta in etas:
            plt.plot(linear_logs[eta]['l2s']['x'], linear_logs[eta]['l2s']['y'],
         marker='o')
            plt.title('Linear Reg. L2(Weight): eta=' + str(eta))
            plt.xlabel('time step t')
            plt.ylabel('L2(weights)')
            plt.show()
```


Linear Reg. L2(Weight): eta=0.8
L2(weights)
time step t
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06

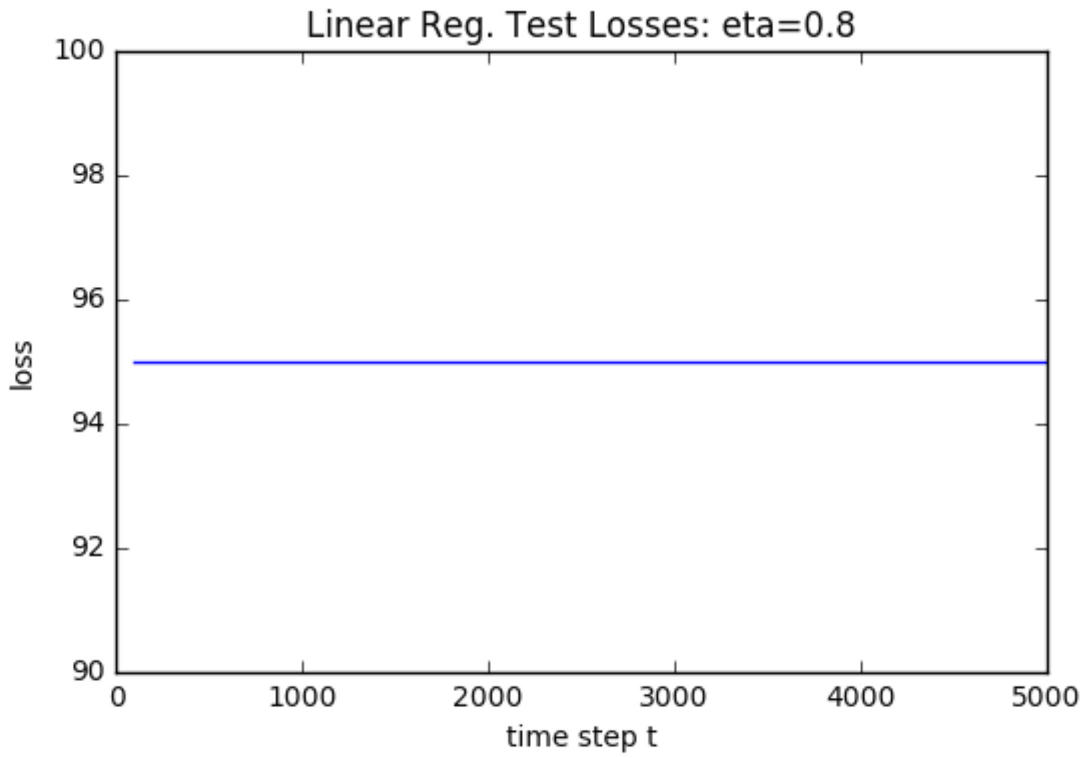

Linear Reg. L2(Weight): eta=0.001
L2(weights)
time step t
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

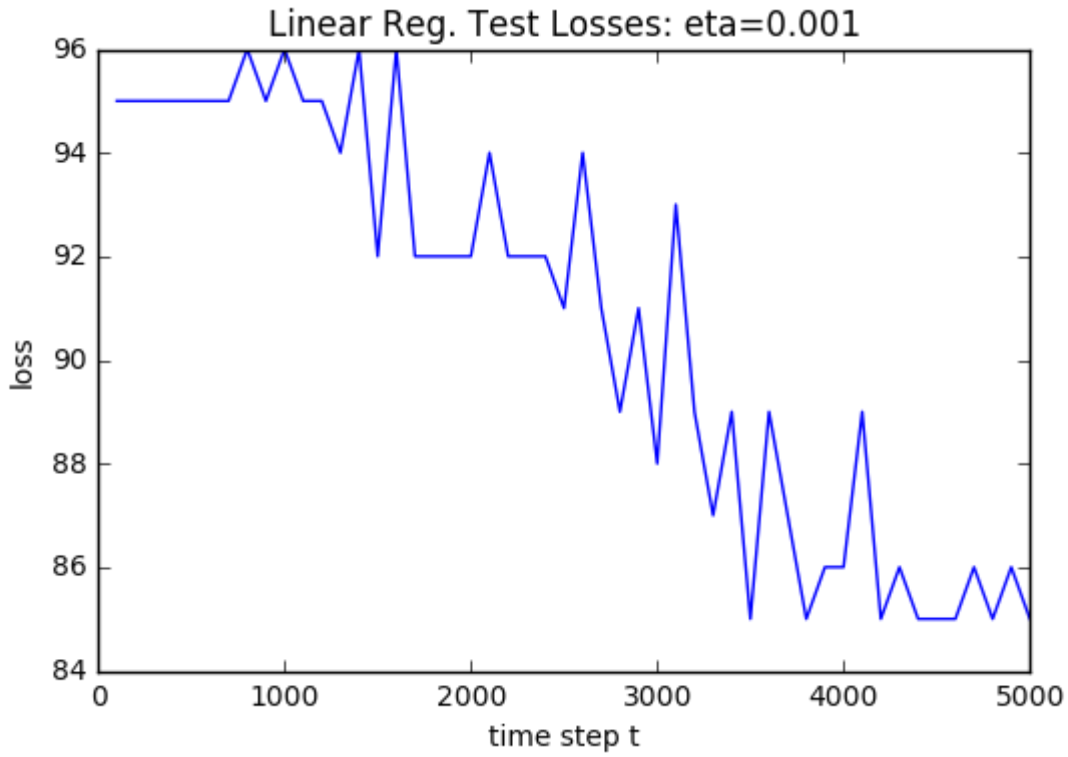


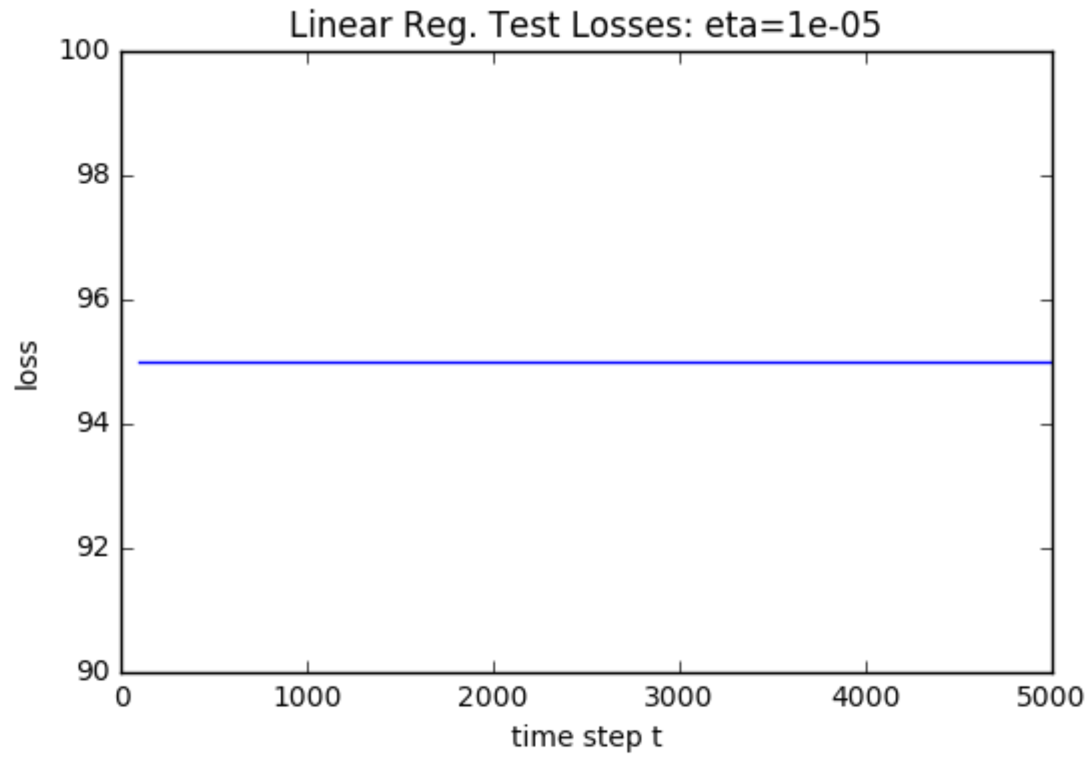
4.2.b.iii

Use the model weights to predict whether each patient in the test set has diabetes, for every 100 steps. Recall that "test label.txt" contains the labels for the test data. Report the SSE (sum of squared errors) of your prediction. Make sure to use the SSE for the 0/1 prediction.

```
In [9]: for eta in etas:
            plt.plot(linear_logs[eta]['test_losses']['x'], linear_logs[eta]['tes
        t_losses']['y'])
            plt.title('Linear Reg. Test Losses: eta=' + str(eta))
            plt.xlabel('time step t')
            plt.ylabel('loss')
            plt.show()
```


Linear Reg. Test Losses: eta=0.8
loss
time step t
100
98
96
94
92
90
0
1000
2000
3000
4000
5000


Linear Reg. Test Losses: eta=0.001
loss
time step t
96
94
92
90
88
86
84
0
1000
2000
3000
4000
5000



Best Logistic Regression Model

In order to find the best logistic regression model, we will train the network for 100,000 steps for all eta values and then take the average of the last **N=500** weights for each eta. Then finally we will eta value which resulted in the least loss and then use the respective weights as our final weights for the model.

In [10]:
```
logistic_w = {}
logistic_logs = {}
for eta in etas:
    logistic_logs[eta] = {}
    logistic_w[eta] = train(X, y, np.zeros(D), logistic_loss, eta, 10000
0, logs=logistic_logs[eta])
```

In [11]:
```
best_eta = etas[0]
best_loss, _ = logistic_loss(X['train'], y['train'],
logistic_w[best_eta])
for eta in etas[1:]:
    loss, _ = logistic_loss(X['train'], y['train'], logistic_w[eta])
    if loss < best_loss:
        best_eta = eta
        best_loss = loss

print 'best eta=' + str(best_eta) + ' with loss=' + str(best_loss)
features_ = np.array(['BMI', 'insulin', 'PGC'])
ids = [np.argmax(features == f) for f in features_]
print str(features_) + " = " + str(logistic_w[best_eta][ids])
```

```
best eta=0.8 with loss=107
['BMI' 'insulin' 'PGC'] = [  7.30162095  -0.92481426  11.74163817]
```

In [12]:
```
y_ = logistic_loss(X['test'], None, logistic_w[best_eta])
print str(round(100.0 * np.sum(y_ == y['test']) / y['test'].shape[0],
2)) + '% accuracy on test'
```

```
72.76% accuracy on test
```

Interpretation of The Weights

W['BMI'] = 7.30162095 W['insulin'] = -0.92481426 W['PGC'] = 11.74163817

The above weights for the features BMI, 2-Hour serum insulin level, and Plasma glucose concentration make sense because:

1. A higher BMI suggests higher chances for diabities.
2. Lack of insulin in the body suggests higher changes for diabities
3. A higher Plasma glucose concentration suggests higher changes of diabities.