Problem 3.19

a) as we can see $\phi(x)$ projected X into $\mathbb{R}^n$ space, for PLA in $\mathbb{R}^n$ $d_{vc} = n+1$. thus is to able to shetter n point. which means this trasformation works.

b)

$$\phi(x, x_n) = \left(\exp\left(-\frac{\|x - x_n\|^2}{2\gamma^2}\right)\right)$$ (γ^2 is just adding scalar to X and x_n, thus could ignore it)

$$\Rightarrow \exp(-(x^2))\exp(-(x_n)^2)\exp(2x x_n)$$

Taylor expansion

$$= \exp(-(x^2))\exp(-(x_n)^2)\sum_{i=1}^{\infty}\frac{(2x x_n)^i}{i!}$$

$$= \sum_{i=1}^{\infty}\left(\exp(-(x)^2)\cdot\sqrt{\frac{2^i}{i!}}(x)^i\right)\left(\exp(-(x_n)^2)\frac{\sqrt{2^i}}{\sqrt{i!}}(x_n)^i\right)$$

for known. x_n and x.

x is transformed into finity dimensional with.

$$\left[1, \frac{\sqrt{2}}{\sqrt{1!}}, \frac{\sqrt{2}}{\sqrt{2!}}, \cdots\right]\begin{bmatrix} 1 \\ x^1 \\ x^2 \\ \vdots \\ \vdots \end{bmatrix}$$

for infinity dimension. $d_{vc} = \infty$, and can shatter all X. thus this transformation works

c) is just a special case of b)

To calculate the eculidean distance, we need to project X to (i,j)'s space

When we project X to $\mathbb{R}^{100}$ $\triangleq$ proj(X)

proj(X) is just linear combination of elements of X

thus we can use the taylor expansion similar in part b) respect to proj(X). and trasform proj(X) into infinity dimension. Which also means X is transformed to infinity dimension. Thus the transformation works.

Exercise , 4.5

a) $w'\Gamma'\Gamma w = (\Gamma w)'(\Gamma w) = \sum_{q=0}^{Q} w_q^2$

thus $\Gamma w = \begin{bmatrix} w_0 \\ \vdots \\ w_Q \end{bmatrix}_{(Q+1)\times 1} \Rightarrow \Gamma = I_{Q+1}$

b) $(\Gamma w)'(\Gamma w) = (\sum_{q=0}^{Q} w_q)^2$

$$\Gamma w = \begin{bmatrix} \frac{1}{\sqrt{Q}} \sum_{i=0}^{Q} w_i \\ \vdots \\ \frac{1}{\sqrt{Q}} \sum_{i=0}^{Q} w_i \end{bmatrix}_{(Q+1)\times 1} \Rightarrow \Gamma = \frac{1}{\sqrt{Q+1}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \ddots & & \\ \vdots & & \ddots & \\ 1 & & \cdots & 1 \end{bmatrix}_{(Q+1)\times(Q+1)}$$

problem 4.8

from book Equation (4.6)

$$E_{Aug}(w) = E_{in}(w) + \lambda w^T w$$

take derivative respect to w on both side.

$$\nabla E_{aug}(w) = \nabla E_{in}(w) + 2\lambda w$$

$$\Rightarrow w(t) - \eta \nabla E_{aug}(w(t)) = w(t) - \eta(\nabla E_{in}(w(t)) + 2\lambda w(t))$$

$$= w(t) - \eta \nabla E_{in}(w(t)) - 2\lambda\eta w(t)$$

$$= (1-2\eta\lambda) w(t) - \eta \nabla E_{in}(w(t))$$

CSE 417 homework 3

Muzhou Liu

Problem 1

- The generalization of the model looked okay, as there is small difference between the training error and the test error in each case, which shows the model is well generalized.
  - The E_in we got from the 10k, 100k and 1 million iterations are 0.5847, 0.4937 and 0.4354
  - The training error we got from 10k, 100k and 1 million iterations are 0.3092, 0.2237, and 0.1513
  - The test error we got from 10k, 100k and 1 million iterations are 0.3172,0.2069, and 0.1310
  - As the three interactions all used up maxim iteration steps we set up, the more the algorithm iterates, the closer the gradient to its minimum state. Which is the reason that we can see that in all criteria, the output is improving as the number of iteration increases.

- Comparing with the results we got form last part, glm model, it has the lowest E_in (0,4074), little bit higher training error (0.1711) and the lowest testing error (0.1103)
  - Compare the running time using the “timeit” function, it returns 5.1616*10^-4 second for the ‘glmfit’ function and 7.5506 seconds for the one-million-time nitration which returns an even worse output comparing to the ‘glmfit’ function.

- After standardization, we tried five different learning rates (from 10^-1 to 10^-5) to input into the model, they took 2333, 23368, 233708, 2337116 and 23371191 steps to final terminate.
  - The e_in we got from all the five learning are all 0.4074.

Appendix 1 :Codes

```
function [ w, e_in ] = logistic_reg( X, y, w_init, max_its, eta )
%LOGISTIC_REG Learn logistic regression model using gradient descent
%   Inputs:
%       X : data matrix (without an initial column of 1s)
%       y : data labels (plus or minus 1)
%       w_init: initial value of the w vector (d+1 dimensional)
%       max_its: maximum number of iterations to run for
%       eta: learning rate

%   Outputs:
%       w : weight vector
%       e_in : in-sample error (as defined in LFD)

X = [ones(size(X,1),1) X];
y = 2*y -1;
old_w = w_init;
n_itr = 0;
tol = 0.001*2;
n = size(X,1);
while tol > 0.001 && n_itr < max_its
 g = -sum(y .* X ./ (1+exp(y.*X*old_w)))/n;
 new_w = old_w-eta*transpose(g);
 old_w = new_w;
 n_itr = n_itr+1;
 tol = max(abs(g)) ;
end
w = new_w;
e_in = sum(log(1+exp(-y.*X*w)))/n;
end
```

```
function [ test_error ] = find_test_error( w, X, y )
%FIND_TEST_ERROR Find the test error of a linear separator
%   This function takes as inputs the weight vector representing a linear
%   separator (w), the test examples in matrix form with each row
%   representing an example (X), and the labels for the test data as a
%   column vector (y). X does not have a column of 1s as input, so that
%   should be added. The labels are assumed to be plus or minus one.
%   The function returns the error on the test examples as a fraction. The
%   hypothesis is assumed to be of the form (sign ( [1 x(n,:)] * w )



y = 2*y-1;
temp = ones(size(X,1),1);
X = [temp X];
test_label = sign(X*w);

test_error = sum(y ~= test_label)/size(X,1);
```

```
end
```

```
function [ w, e_in, n_itr ] = logistic_reg2( X, y, w_init, eta )
%LOGISTIC_REG Learn logistic regression model using gradient descent
%   Inputs:
%       X : data matrix (without an initial column of 1s)
%       y : data labels (plus or minus 1)
%       w_init: initial value of the w vector (d+1 dimensional)
%       max_its: maximum number of iterations to run for
%       eta: learning rate

%   Outputs:
%       w : weight vector
%       e_in : in-sample error (as defined in LFD)

X = [ones(size(X,1),1) X];
y = 2*y -1;
old_w = w_init;
n_itr = 0;
tol = 0.001*2;
n = size(X,1);

while tol > 10^(-6)
 g = -sum(y .* X ./ (1+exp(y.*X*old_w)))/n;
 new_w = old_w-eta*transpose(g);
 old_w = new_w;
 n_itr = n_itr+1;
 tol = max(abs(g)) ;
end

n_itr= n_itr;
w = new_w;
e_in = sum(log(1+exp(-y.*X*w)))/n;
end
```

```
train = csvread('clevelandtrain.csv',1,0);
train_x = train(:,1:13);
train_y =  train(:,14);

test = csvread('clevelandtest.csv',1,0);
test_x = test(:,1:13);
test_y = test(:,14);


[w_10k,e_10k] = logistic_reg(train_x,train_y,zeros(14,1),10000,0.00001);
[w_100k,e_100k] = logistic_reg(train_x,train_y,zeros(14,1),100000,0.00001);
[w_1m,e_1m] = logistic_reg(train_x,train_y,zeros(14,1),1000000,0.00001);
```

```
train_error_10k = find_test_error(w_10k,train_x,train_y)
test_error_10k = find_test_error(w_10k,test_x,test_y)

train_error_100k = find_test_error(w_100k,train_x,train_y)
test_error_100k = find_test_error(w_100k,test_x,test_y)

train_error_1m = find_test_error(w_1m,train_x,train_y)
test_error_1m = find_test_error(w_1m,test_x,test_y)




glm_w = glmfit(train_x,train(:,14) , 'binomial');

e_glm = sum(log(1+exp(-(2*train_y-1).*[ones(size(train_x,1),1)
train_x]*glm_w)))/size(train_x,1)
glm_train_error = find_test_error(glmfit(train_x,train(:,14) ,
'binomial'),train_x,train_y)
glm_test_error = find_test_error(glmfit(train_x,train(:,14) ,
'binomial'),test_x,test_y)

glm_time = @() glmfit(train_x,train(:,14) , 'binomial');
glm_time = timeit(glm_time)
my_time = @() logistic_reg(train_x,train_y,zeros(14,1),1000000,0.00001);
my_time = timeit(my_time)

 stand_train_x = zscore(train_x);



[w_std1,e_std1, n_std1] = logistic_reg2(stand_train_x, train_y,
zeros(14,1),0.1)

[w_std2,e_std2, n_std2] = logistic_reg2(stand_train_x, train_y,
zeros(14,1),0.01)

[w_std3,e_std3, n_std3] = logistic_reg2(stand_train_x, train_y,
zeros(14,1),0.001)

[w_std4,e_std4, n_std4] = logistic_reg2(stand_train_x, train_y,
zeros(14,1),0.0001)

[w_std5,e_std5, n_std5] = logistic_reg2(stand_train_x, train_y,
zeros(14,1),0.00001)

%[w_std6,e_std6, n_std6] = logistic_reg2(stand_train_x, train_y,
zeros(14,1),0.000001);


train_error_std1 = find_test_error(w_std1,stand_train_x,train_y)
train_error_std2 = find_test_error(w_std2,stand_train_x,train_y)
train_error_std3 = find_test_error(w_std3,stand_train_x,train_y)
```

```
train_error_std4 = find_test_error(w_std4,stand_train_x,train_y)
train_error_std5 = find_test_error(w_std5,stand_train_x,train_y)
```

Appendix 2 : Outputs

Workspace

Name	Value
e_100k	0.4937
e_10k	0.5847
e_1m	0.4354
e_glm	0.6062
e_std1	0.4074
e_std2	0.4074
e_std3	0.4074
e_std4	0.4074
e_std5	0.4074
glm_test_error	0.1103
glm_time	5.1616e-04
glm_train_error	0.1103
glm_w	*14x1 double*
my_time	7.5506
n_std1	2333
n_std2	23368
n_std3	233708
n_std4	2337116
n_std5	23371191
stand_train_x	*152x13 double*
test	*145x14 double*
test_error_100k	0.2069
test_error_10k	0.3172
test_error_1m	0.1310
test_x	*145x13 double*
test_y	*145x1 double*
train	*152x14 double*
train_error_100k	0.2237
train_error_10k	0.3092
train_error_1m	0.1513
train_error_std1	0.1711
train_error_std2	0.1711
train_error_std3	0.1711
train_error_std4	0.1711
train_error_std5	0.1711
train_x	*152x13 double*
train_y	*152x1 double*
w_100k	*14x1 double*
w_10k	*14x1 double*
w_1m	*14x1 double*
w_std1	*14x1 double*
w_std2	*14x1 double*
w_std3	*14x1 double*
w_std4	*14x1 double*
w_std5	*14x1 double*