CODES FILES

PROBLEM 2

1. **linear_least_square_regression.m**

```
function [w] = linear_least_squares_learner(train,label_train)
[m n] = size(train);
train2 = train(:,1);
for i=2:n
    train2 = [train2 , train(:,i)];
end
train = [train2 , ones(m,1)];
w = (train'*train);
w = inv(w);
w = w*train';
w = w*label_train;
save('2a.mat','w');
end
```

2. **linear_predictor.m**

```
function [ypred] = linear_predictor(test)
load('2a.mat');
[m n] = size(test);
size(test)
test = [test  ones(m,1)]
size(test)
size(w)
ypred = (w')*(test') ;
save('2d2.mat','ypred');
end
```

3. **linear_ridge_learner.m**

```
function [w] = linear_ridge_learner(train,label_train,lambda)
[m n] = size(train);
train = [train , ones(m,1)];
w = (train'*train);
w = w + lambda*eye(size(w,1),size(w,2));
w = inv(w);
w = w*train';
w = w*label_train;
save('2d.mat','w');
end
```

4. **linear_predictor_d.m**

```
function [ypred] = linear_predictor_d(test)
load('2d.mat');
[m n] = size(test);
test = [test , ones(m,1)]
ypred = w'*test' ;
save('2d2.mat','ypred');
end
```

5. **kernel_ridge_learner.m**

```
function [w] = kernel_ridge_learner(train,label_train,lambda,kernel,d)
[m n] = size(train);
K = computeKernel(train,train,kernel,d);
w = K;
w = w + lambda*eye(size(w,1),size(w,2));
w = inv(w);
w = w*label_train;
save('2e.mat','w');
end
```

6. **kernel_predictor.m**

```
function [ypred] = kernel_predictor(test,train,kernel,d)
load('2e.mat');
[m n] = size(test);
ypred = zeros(size(test,1),1);
for i=1:size(test,1)
    K = computeKernel(test(i,:),train,kernel,d);
size(K);
size(w);
    ypred(i) = K*w;
end
save('2e2.mat','ypred');
end
```

7. **squared_error.m**

```
function [err] = squared_error(y_pred,y_true)
% This function computes the average squared error between the predicted
% value vector and the actual value vector.
% y_pred - predicted real value vector of size  (n*1)
% y_true - actual real value vector of size (n*1)

%Output
%err - mean squared error between y_pred and y_true.

err = mean((y_pred-y_true).^2);
end
```

PROBLEM 3

8. SVM_learner.m

```
function[model]  = SVM_learner(traindata,trainlabels,kernel,d,C,eps)
1.    [m n] = size(traindata);
2.
3.    %compute the Kernel
4.    K = computeKernel(traindata,traindata,kernel,d);
5.
6.    %compute the Hessian Matrix
7.    H = [K -K; -K K];
8.    alphas = zeros(2*m,1);
9.
10.   %compute f
11.   f1 = eps - trainlabels ;
12.   f2 = eps + trainlabels ;
13.   f = [f1 ; f2] ;
14.
15.   %put lower and upper bounds
16.   lb = zeros(2*m,1);
17.   ub = C*ones(2*m,1);
18.
19.   %put equality constraints
20.   Aeq = [ones(m,1) ; (-1)*ones(m,1)]';
21.   beq = 0;%zeros(2*m,1);
22.
23.   %set options
24.   options = optimset('quadprog');
25.   options.MaxIter = 20;
26.   options.Algorithm = 'interior-point-convex' ;
27.
28.  %call quadprog
29.  [alphas fvals] = quadprog(H,f,[],[],Aeq,beq,lb,ub,[],options);
30.
31.  save mydata
32.
33.  x = [traindata ; traindata];
34.  w = Aeq * x ;
35.
36.  model.alphas = alphas;
37.  model.fval = fvals;
38.  model.w = w;
39. save alldata.mat
40. end
```

9. kernel_predictor.m

```
function [ypred] = kernel_predictor(test,train,kernel,d,model)
[m n] = size(test);
ypred = zeros(size(test,1),1);
for i=1:size(test,1)
    K = computeKernel(test(i,:),train,kernel,d);
size(K)
ypred(i) = (model.alphas)' * ([K' ; -K']);
end
save('2e2.mat','ypred');
end
```

PROBLEM 4

10. kNN.m

```
function pred = kNN( X_train, y_train, k, r, x_test)
    % getting size of training set
    [m n] = size( X_train);

    mp = zeros( m, 1);
    kmap = zeros( m, 1);
    indices = zeros( m, 1);

    % calculating distances
    for i = 1 : m
        diff = ( x_test - X_train( i, :)) .^ 2;
        dist = sum( diff);
        mp( i) = sqrt( dist);
    end

    % sorting based on distances
    [ kmap, indices] = sort( mp);

    decision = zeros( r, 1);

    %selecting nearest k neighbours
    for i = 1 : k
        decision( y_train( indices( i)) + 1) = decision( y_train( indices( i)) + 1) + 1;
    end

    [ digit pos] = max( decision);

    pred = pos - 1;
end
```

11. multiclass_error.m

```
function [ err] = multiclass_error(y_pred,y_true)
%This function computes the multiclass error between predicted lables
%y_pred and actual lables y_true.

%y_pred - Predicted lables (m*1 vector whose each entry belongs to the set
{0,1,2,3,4,5,6,7,8,9}
%y_true - True lables (m*1 vector whose each entry belongs to the set
{0,1,2,3,4,5,6,7,8,9}

%Output
%err - Fraction of data instances where y_pred and y_true do not match.

len = (length(y_pred));
err = length(find(y_pred ~= y_true))/len;
end
```