**ENGN2520 Homework 4**

**Ming Xu (Banner ID: B01532164)**

**Problem1**

***(a)***

Suppose :

Therefore:

***(b)***

Suppose and :

Therefore:

***(c)***

Suppose and :

Therefore:

***(d)***

Suppose and and:

Therefore:

**Problem2**

***(a)***

For multiclass SVM, the classifer ***y*** is defined as below:

When there are 2 class, y can be written as：

Equivalently, y can be written as:

Let , then y becomes:

, which is a linear perceptron classifier.

***(b)***

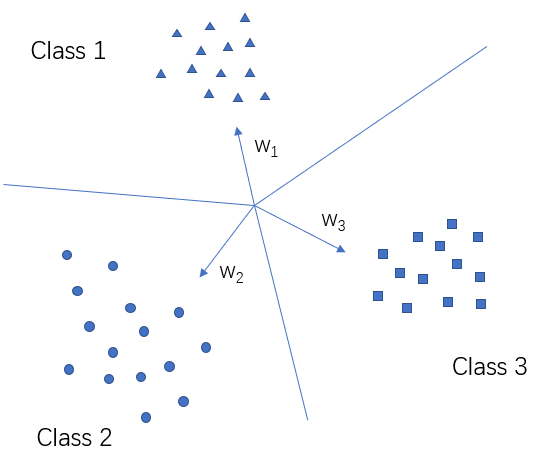
For multiclass SVM, the classifer ***y*** is defined as below:

For K classes, the classifier can be written as below:

If , then .

Therefore, the boundaries between each decision regions are like linear perceptions.

Below is an example of 3 class problem in the plane(D=2) where a multiclass SVM is used. The boundary between decision regions are lines determined by , and .



**Problem3**

The matlab function to calculate and :

function [Ew,gradientEw] = calculateEw(w,x,y,C)

%k classes

[~,k] = size(w);

%n training datas

[n,~] = size(x);

%initialization

Ew = 0;

gradientEw = w;

%loop all training datas

for index = 1:n

temp = x(index,:)\*w;

y\_train = y(index,1);

%find yhat, make sure yhat = argmax(temp) where yhat!=y

temp(y\_train) = -Inf;

[~,y\_hat] = max(temp);

%calculate w\_y\_x and w\_y\_hat\_t

w\_y\_x = x(index,:)\*w(:,y\_train);

w\_y\_hat\_x = x(index,:)\*w(:,y\_hat);

if w\_y\_x<w\_y\_hat\_x+1

%update E

Ew = Ew+C\*(w\_y\_hat\_x+1-w\_y\_x);

%update gradient E

gradientEw(:,y\_hat) = gradientEw(:,y\_hat) + C\*x(index,:)';

gradientEw(:,y\_train) = gradientEw(:,y\_train)-C\*x(index,:)';

end

end

for i = 1 : k

Ew = Ew + w(i,:)\*w(i,:)'/2;

end

end

**Problem4**

***(a)***

The matlab function for gradient descent algorithm:

function [w,Loss] = gradientDescent(x,y,C,r,T)

%generate a random w matrix

w = rand(785,10,'double');

Loss = [];

d = 1.01;

%iteration for T times

for i = 1:T

%calculate Loss and gradient of the loss

[E, G] = calculateEw(w,x,y,C);

%save loss for further visulization

Loss = [Loss, E];

%update w

w = w - r \* G;

%decrease stepsize

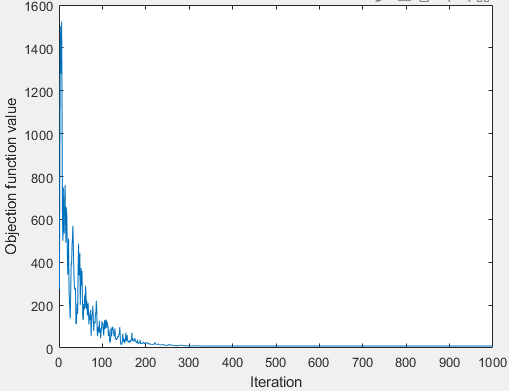
r = r / d;

end

end

***(b)***

Below is a figure of objection function value over time, when C = 0.01, r = 0.01, and T = 1000:



***(c)***

The different values of C and its corresponding correct fraction is shown in the table below. In this experiment, r = 0.01 and T=1000.

|  |  |  |
| --- | --- | --- |
| Index | C | Correct fraction |
| 1 | 0.0001 | 74.66% |
| 2 | 0.001 | 84.40% |
| 3 | 0.01 | 87.32% |
| 4 | 0.1 | 85.16% |
| 5 | 1 | 84.26% |
| 6 | 10 | 83.90% |
| 7 | 100 | 84.02% |

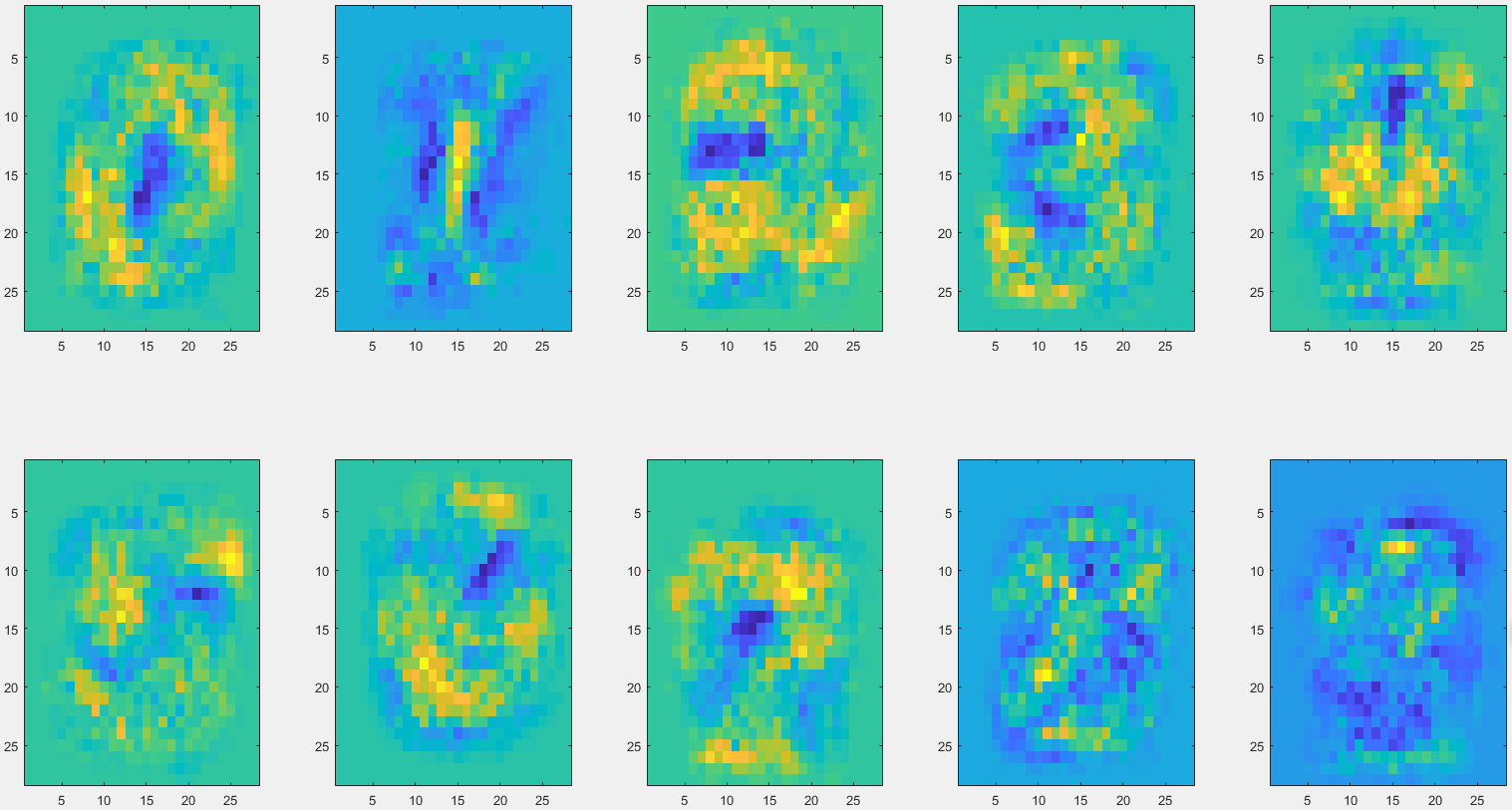
From the table, we can find when C=0.01, the model has minimum number of erros.

The confusion matrix, when C = 0.01, is show as below:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 464 | 0 | 13 | 2 | 0 | 7 | 8 | 0 | 4 | 2 |
| **1** | 0 | 478 | 6 | 6 | 1 | 3 | 4 | 0 | 2 | 0 |
| **2** | 5 | 11 | 410 | 17 | 5 | 2 | 8 | 20 | 17 | 5 |
| **3** | 4 | 0 | 13 | 406 | 0 | 35 | 3 | 15 | 16 | 8 |
| **4** | 2 | 0 | 0 | 4 | 440 | 1 | 9 | 7 | 11 | 26 |
| **5** | 12 | 2 | 11 | 27 | 16 | 379 | 9 | 7 | 30 | 7 |
| **6** | 8 | 3 | 13 | 0 | 17 | 18 | 434 | 5 | 1 | 1 |
| **7** | 0 | 5 | 22 | 17 | 7 | 0 | 0 | 417 | 0 | 32 |
| **8** | 5 | 5 | 21 | 31 | 12 | 18 | 2 | 11 | 384 | 11 |
| **9** | 3 | 2 | 4 | 12 | 43 | 4 | 1 | 31 | 11 | 389 |

***(d)***

When C = 0.01, the visualization of each digit is shown as below:



***(e)*** Source code

**1. Code to prepare training set:**

clear;clc;

load('digits');

x = [];

y = [];

for i = 1 : 10

x = [x; ones(500,1), eval(['train' num2str(i-1)])];

y = [y; ones(500,1) \* i];

end

**2. Code to plot figure of objection function value vs iteration time**

%% Objective function value vs iteration time

r = 0.1;

T = 1000;

C = 0.01;

[w,Loss] = gradientDescent(x,y,C,r,T);

plot(Loss);

xlabel('Iteration');

ylabel('Objection function value');

**3. Function to do experiments on different values of C:**

%% Do experiments of different values of C and find minimum classification erros

figure();

hold on;

acc\_max = 0;

C = [0.0001,0.001,0.01, 0.1, 1, 10, 100];

r = 0.1;

T = 1000;

for i = 1 : size(C, 2)

[w,Loss] = gradientDescent(x,y,C(i),r,T);

[acc, classification\_map] = Test(w, 10, 500);

fprintf('C = %f, accuracy = %f \n', C(i), acc);

if acc > acc\_max

acc\_max = acc;

c\_best = C(i);

W\_best = w;

result = classification\_map;

end

end

**4. Function to visualize digits of w:**

figure();

for i = 1 : 10

subplot(2,5,i);

w\_i = normalize(W\_best(2 : end, i), 'range');

imagesc(reshape(w\_i,28,28)');

end