ENGN 2520 Pattern Recognition and Machine Learning

Homework 4

Zhuo Wang

Problem 1

(a) Suppose $W_y^T x > W_{\hat{y}}^T x + 1$

$$\frac{dL((W_1,\dots,W_k),(x,y))}{dW_{j,l}} = 0$$

(b) Suppose $W_y^T x < W_{\hat{y}}^T x + 1$ and $j = y$

$$\frac{dL((W_1,\dots,W_k),(x,y))}{dW_{j,l}} = -x_l$$

(c) Suppose $W_y^T x < W_{\hat{y}}^T + 1$ and $j = \hat{y}$

$$\frac{dL((W_1,\dots,W_k),(x,y))}{dW_{j,l}} = x_l$$

(d) Suppose $W_y^T x < W_{\hat{y}}^T + 1$ and $j \neq y$ and $j \neq \hat{y}$

$$\frac{dL((W_1,\dots,W_k),(x,y))}{dW_{j,l}} = 0$$

Problem 2

(a)

The decision boundaries for these two classes are

$$W_1^T x + b_1 = 0 \quad \text{and} \quad W_2^T x + b_2 = 0$$

A single hyperplane:

$$(W_1^T - W_2^T)x + (b_1 - b_2) = 0$$

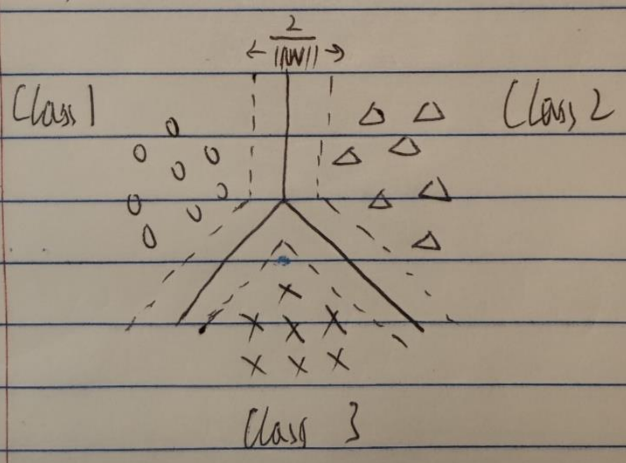
$$\Downarrow$$

$$W^T x + b = 0$$

$$\downarrow$$

$$W = W_1^T - W_2^T$$

(b)



## Problem 3

```
function [E, gradE] = E(w, ImgFlat, C)
    % -- initialization --
    %    w(10*748)
    E = 0;
    gradE = zeros(size(w));

    % Number of classes
    k = size(w, 1);

    % Loop over training examples
    for digitLabel = 9:-1:0
        n = ['train' num2str(digitLabel)];
        Img = ImgFlat.(n);
        for t = 1:size(Img,1)
            x = Img(t,:);
            % Compute scores for all classes
            scores = w * x';

            % Compute loss for each class
            for j = 1:k
                if j == digitLabel
                    continue; % Skip correct class
                end

                % Compute hinge loss
                loss = max(0, scores(j) - scores(digitLabel+1) + 1);

                % Update objective function
                E = E + 0.5 * norm(w(:, j))^2 + C * loss;

                % Update gradient
                if loss > 0
                    gradE(j, :) = gradE(j, :) + x;
                    gradE(digitLabel+1, :) = gradE(digitLabel+1, :) - x;
                end
            end
        end
    end

    reg_term = 0.5 * sum(sum(w.^2));

    E = E + reg_term;
```

```
    gradE = gradE + w;
end
```

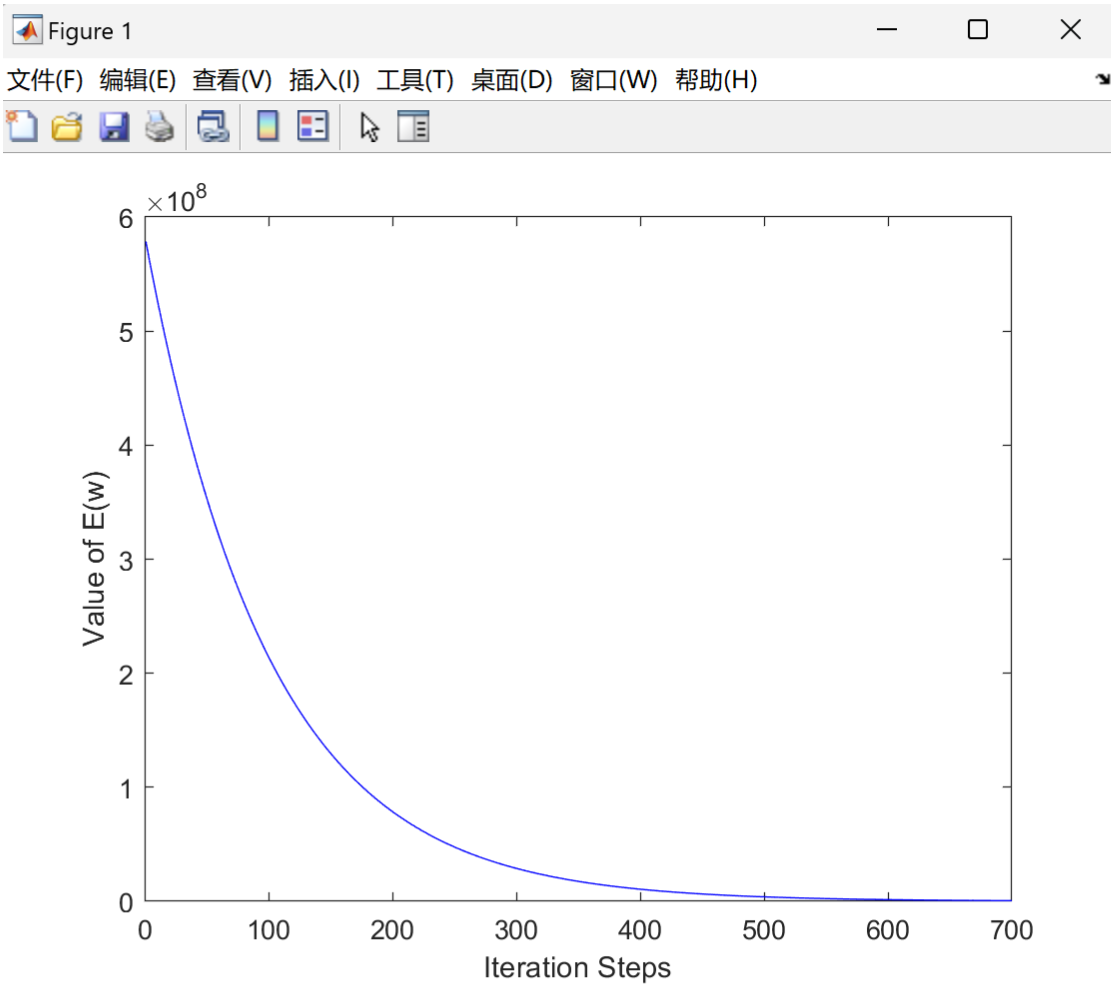
## Problem 4

(a)

```
function trained_w = GradientDescent_Multiclass_SVM(ImgFlat, c, r, T)
    % -- ImgFlat is n x 784 x k dataset, where n is # of imgs --
    % -- c is the constant used in the objective function --
    % -- r is learning rate used in the gradient descent algorithm --
    % -- T is the iterations of the gradient descent algorithm --

    % -- initialization --
    plotObjectiveFunc = 1;
    valOfE_overT = zeros(1, T);
    seqOfT = zeros(1, T);
    numOfClasses = length(fieldnames(ImgFlat))./2;
    numOfPixs = size(ImgFlat.train1, 2);
    w = 50*ones(numOfClasses, numOfPixs);

    % -- gradient descent algorithm --
    for iter = 1:T
        % -- find E(w) and gradient of E(w) --
        [valOfE, gradOfE] = E(w, ImgFlat, c);

        % -- update the trained model, w --
        w = w - r * gradOfE;

        % -- store the values of E(w) and the iteration indices --
        seqOfT(1, iter) = iter;
        valOfE_overT(1, iter) = valOfE;
    end

    trained_w = w;

    % -- plot the value of E(w) over all iterations --
    figure
    if plotObjectiveFunc
        plot(seqOfT, valOfE_overT, '-b');
        xlabel('Iteration Steps');
        ylabel('Value of E(w)');
    end
    set(gcf,'color','w');
end
```

(b)

The best performance is when C = 0.01.

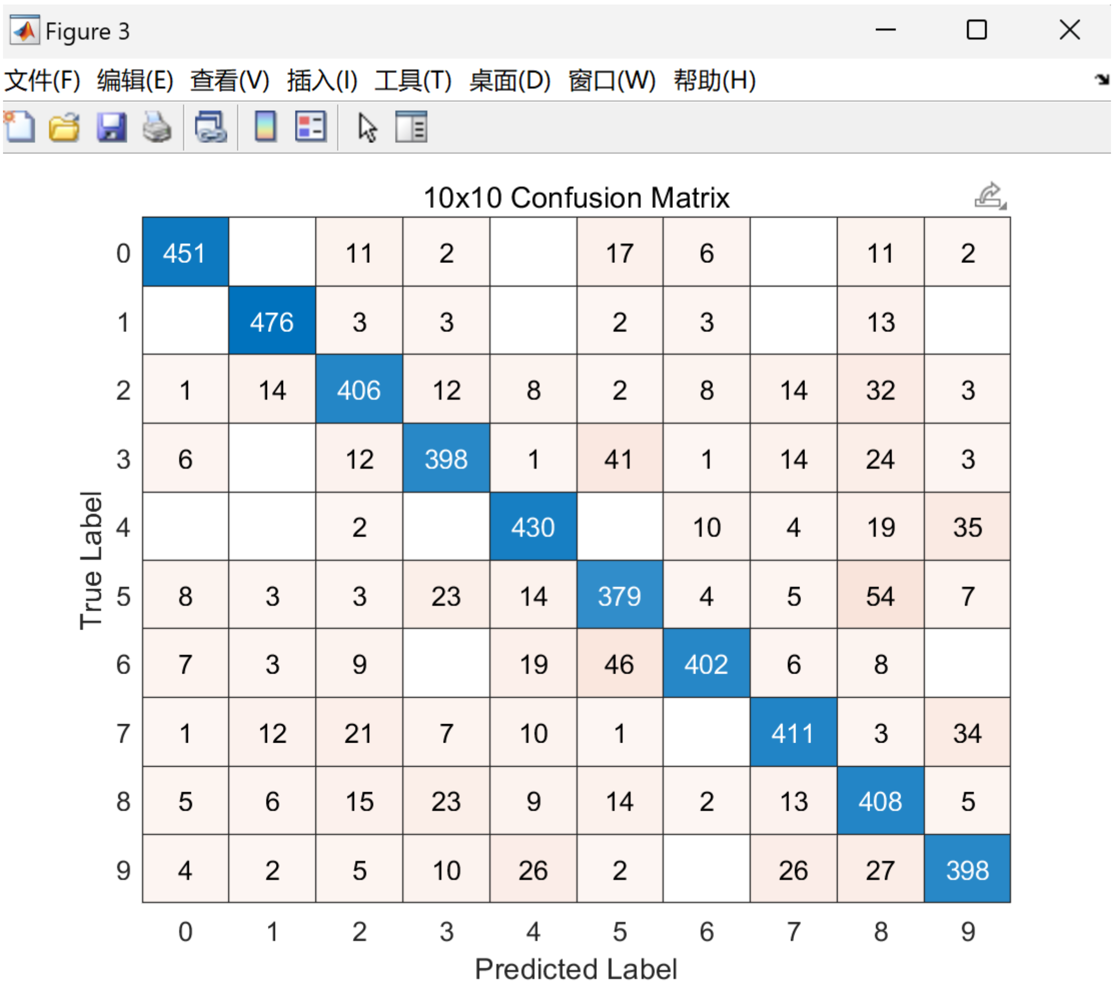



(r = 0.005, C = 0.01, T = 700)

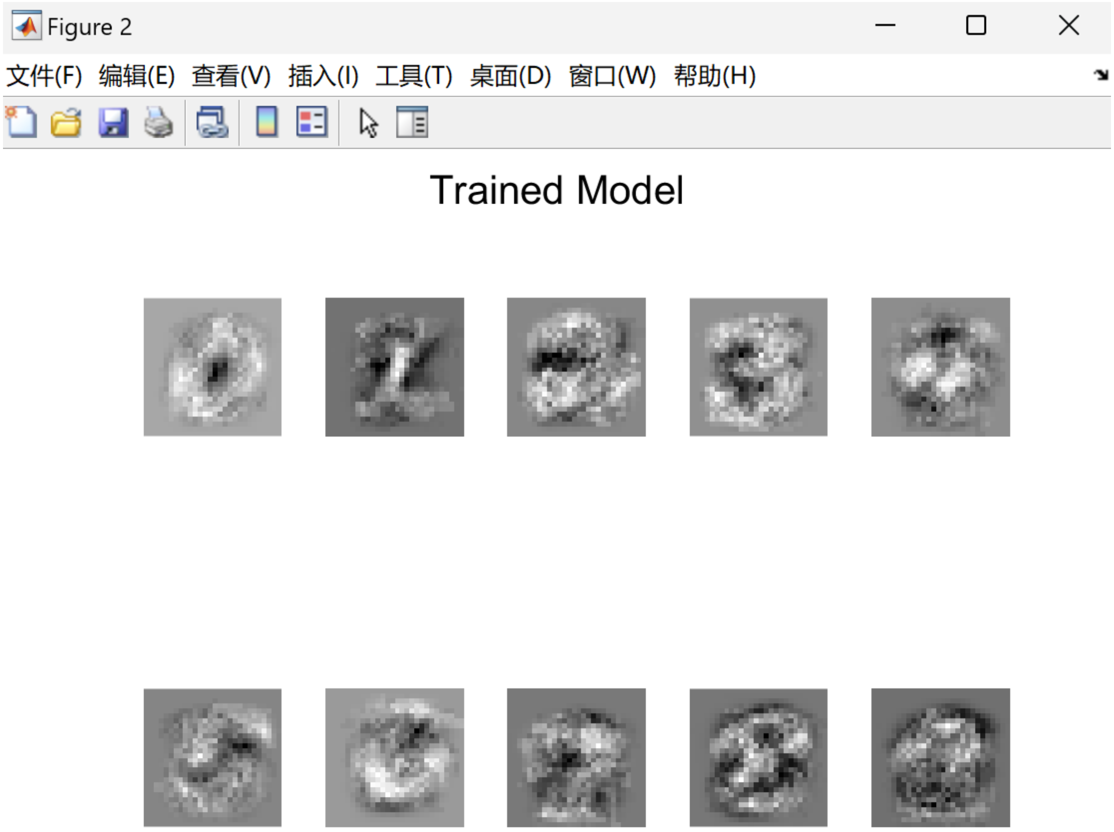
(c)

| C: | 0.01 | 0.1 | 1 | 10 | 100 |
|---|---|---|---|---|---|
| Accuracy Rate: | 83.18% | 83.18% | 83.18% | 83.18% | 83.18% |

(r = 0.005, T = 700)



(d)



## Source Code

```
clc;
clear;
close all;


% -- load data --
digitData = load('digits.mat');

% -- define parameters --
numOfTrainImgs = size(digitData.train0, 1);
numOfTestImgs = size(digitData.test0, 1);
imgSize = size(digitData.train0, 2);
showModel = 1;      % -- whether to show the model img or not --
r = 0.005;           % -- learning rate --
c = 0.01;           % -- constant c used in the objective function --
T = 700;            % -- # of iterations for gradient descent --

% -- 1) construct a trained model (weight vector) using multiclass SVM --
w = GradientDescent_Multiclass_SVM(digitData, c, r, T);

% -- 2) show the trained model for 10 handwritten classes --
modelSet = figure;
if showModel
    for s = 1:10
        mImg = reshape(w(s,:), 28, 28)';
        subplot(2, 5, s);
        valMin = min(mImg(:));
        valMax = max(mImg(:));
        norm_img = (mImg - valMin) / (valMax - valMin);
        imshow(uint8(255*mat2gray(norm_img)));
    end
    sgtitle('Trained Model');
    set(gcf,'color','w');
end

% -- 3) evaluate the accuracy of the model using test data --
digitProb = zeros(10, 1);       % -- probabiliy of the class given the input
image --
correctCnt = zeros(10, 1);      % -- count the correct predictions --
failDigit = zeros(10, 10, 1);   % -- collect the failed predictions --
for testDigitLabel = 0:9
    for testIndx = 1: numOfTestImgs
```

```
        n = ['test' num2str(testDigitLabel)];
        TImg = digitData.(n);
        x_test = TImg(testIndx,:);
        predict = w * x_test';
        [maxVal, digit] = max(predict);

        % -- if the prediciton of the model is successful --
        if digit == testDigitLabel+1
            correctCnt(testDigitLabel+1, 1) = correctCnt(testDigitLabel+1,
1) + 1;
            failDigit(testDigitLabel+1, testDigitLabel+1) =
failDigit(testDigitLabel+1, testDigitLabel+1) + 1;
        % -- otherwise it is failed --
        else
            failDigit(testDigitLabel+1, digit) = failDigit(testDigitLabel+1,
digit) + 1;
        end
    end
end

% -- plote the confusion matrix --
figure();
cm = confusionchart(failDigit, [0,1,2,3,4,5,6,7,8,9]);
cm.Title = '10x10 Confusion Matrix';
ylabel('True Label');
xlabel('Predicted Label');
set(gcf,'color','w');



% -- compute the test accuracies in percentage --
correctCnt = double(correctCnt) / double(500);
disp("Accuracy Rate:" + sum(correctCnt)/10);
```