IN4320: Machine Learning Assignment 2

Zheng Liu (4798406)

March 7, 2019

1 Question a.

The common points of the two descriptions of the algorithms are as follow:

- Both of them boost a weak classifier to a strong one.
- Both consider the mistakes from the previous step.
- Both predict the classification result for each object.

Formulation of Adaboost given in the slides of the course:

$$\begin{aligned} w^{'}_{k+1,i} &= e^{-y_i f_k(x)} \\ &= \mathrm{e}^{-y_i(f_{k-1}(x)+\alpha_k F_k(x))} \\ &= \mathrm{e}^{-y_i f_{k-1}(x)-y_i\alpha_k F_k(x)} \\ &= \mathrm{e}^{-y_i f_{k-1}(x)} e^{-y_i\alpha_k F_k(x)} \\ &= \mathrm{w}^{'}_{ki} e^{-y_i\alpha_k F_k(x)} \end{aligned}$$

The key idea here is to find classifier f_K which could minimise the error where each object is re-weighted by:

$$\omega_i = exp(-y_i \sum_{K-1}^{k=1} \alpha_k f_k(x_i))$$

when the object is correctly classified, the weight would be set lower; on the contrary, if the object is misclassified, its weight then grows larger, which is similar to the description with the paper:

$$\omega_i^{t+1} = \omega_i^t \beta_t^{1-|h_t(x_i)-y_i|}$$

when object is correctly classified, $|h_t(x_i) - y_i|$ is 0, then $\omega_i^{t+1} = \omega_i^t \times \beta_t^1 = \omega_i^t \beta_t$, if it is misclassified, the result is $\omega_i^{t+1} = \omega_i^t \times \beta_t^0 = \omega_i^{t+1} = \omega_i^t$, as β is smaller than 1, so it is obvious that when object is misclassified, it would obtain higher weight than when correctly classified, the same as what is expressed in the slide.

2 Question b.

I implemented weak learner and then extended the weighted version to find the optimal feature f and threshold θ, the code is show in Figure 1 and 2.

The basic idea is that to traverse all features, each time to choose one feature and select the value of the object of this current feature (this process could be done in a two layer *for loop*) as threshold θ and calculate the number of misclassified object, if a better result (lower error) appears, then store the current θ value and current feature *f*. The only difference between weighted version and the unweighted version is that every object has weight which is stored in a vector, each object would be treated differently in the weighted version.

3 Question c.

I generate a test dataset with two Gaussian distribution, one is $\mu_1 = [0, 0]^T$ and the other one is $\mu_2 = [2, 0]^T$. I made a scatterplot of the test dataset with the threshold I obtain which is shown as the decision boundary, shown in Figure 3. The optimal parameter I obtained is:

$$\theta = 1.1875,\ f = 1(index)$$

When rescaling one of the feature, it might change depending on which feature I choose to rescale. When I rescale feature 1, multiply it by a factor of 10, the optimal feature does not change but the θ value is 10 times as before. When I rescale feature 2, also multiply it by a factor of 10, both optimal feature and θ value do not change. The rescaling result is shown in Figure 4.

4 Question d.

I trained the decision stump on the training set *fashion57_train* and evaluated the result on the test set *fashion57_test*. The apparent error I obtained is 0.2 and the test error is 0.35.

5 Question e.

I learned the given paper especially the chapter 4 for implementing AdaBoost algorithm. The input are as follows:

- Dataset (sequence of N labelled examples)
- Initialised distribution D over the N examples
- WeakLearn (mine is WeightedWeakLearn) implemented before
- integer T specifying number of iterations (as required, here I set 100 for T)

Each step the weights are normalised based on

$$p^t = \frac{w^t}{\sum_{i=1}^{N} \omega_i^t}$$

then call WeakLearn() function to return a hypothesis and calculate the error and modify the weights vector, which is

```
function [feature, theta, sign] = WeakLearn(data, label)
% Implement a weak learner WeakLearn:
% the decision stump, that is minimising the apparent error.
% To find the optimal feature f and threshold θ.
% input: dataset
% output: the optimised feature index and theta value
% ******
[n, dim] = size(data);
% n is the number of objects/samples, dim is the number of features/dimensions
feature = 0;
error_min = 10000;
theta = 0;
% compare each dimension
th = min(data(:)) : 0.02 * (max(data(:)) - min(data(:))) : max(data(:));
step_y = size(th, 2);
for i = 1: step_y
    for j = 1: dim
        % traverse the value of objects on each dimension
        for k = 1: n
            % use each value itself as theta
            t = ones(n, 1) * th(i);
            % if the value is graeter or lower than threshold
            pred_lr = data(:, j) - t <= 0;
            pred_gr = data(:, j) - t >= 0;
            pred_lr = pred_lr + 1;
            pred_gr = pred_gr + 1;
            % calculate error in two situation (greater or lower)
            error_current_lr = sum(abs(pred_lr - label));
            error_current_gr = sum(abs(pred_gr - label));
            % to check the relationship with label '1'/'2' and threshold,
            % the smaller error should be the real error
            if error_current_lr >= error_current_gr
                error_current = error_current_gr;
                sign = 1;
            else
                error_current = error_current_lr;
                sign = 0;
            end
            % if current feature i can reach a lower error (rate)
            if error_current < error_min
                % assign the current error to 'error_min' for further comparision
                error_min = error_current;
                % assign current feature index to 'feature'
                feature = j;
                theta = th(i);
            end
        end
    end
end
end
```

Figure 1: MATLAB code for question b., unweighted version

```
function [feature, theta, sign] = WeightedWeakLearn(data, label, weight)
% Implement a weak learner WeakLearn,
% extend the implementation that it accepts a weight per object
% the decision stump, that is minimising the apparent error.
% To find the optimal feature f and threshold θ.
% input: dataset, label nad weight
% output: the optimised feature index and theta value
% %%%%%%
% initialise some parameter
error_min = 10000;
[n, dim] = size(data);
theta = 0;
% n is the number of objects/samples, dim is the number of features/dimensions
th = min(data(:)) : 0.02 * (max(data(:)) - min(data(:))) : max(data(:));
step_y = size(th, 2);
for i = 1: step_y
    for j = 1: dim
        % traverse the value of objects on each dimension
        for k = 1: n
            % use each value itself as theta
            t = ones(n, 1) * th(i);
            % if the value is graeter or lower than threshold
            pred_lr = data(:, j) - t <= 0;
            pred_gr = data(:, j) - t >= 0;
            pred_lr = pred_lr + 1;
            pred_gr = pred_gr + 1;
            % calculate error in two situation (greater or lower)
            error_current_lr = weight' * abs(pred_lr - label);
            error_current_gr = weight' * abs(pred_gr - label);
            % to check the relationship with label '1'/'2' and threshold,
            % the smaller error should be the real error
            if error_current_lr >= error_current_gr
                error_current = error_current_gr;
                sign = 1;
            else
                error_current = error_current_lr;
                sign = 0;
            end
            % if current feature i can reach a lower error (rate)
            if error_current < error_min
                % assign the current error to 'error_min' for further comparision
                error_min = error_current;
                % assign current feature index to 'feature'
                feature = j;
                theta = th(i);
            end
        end
    end
end
end
```

Figure 2: MATLAB code for question b., weighted version

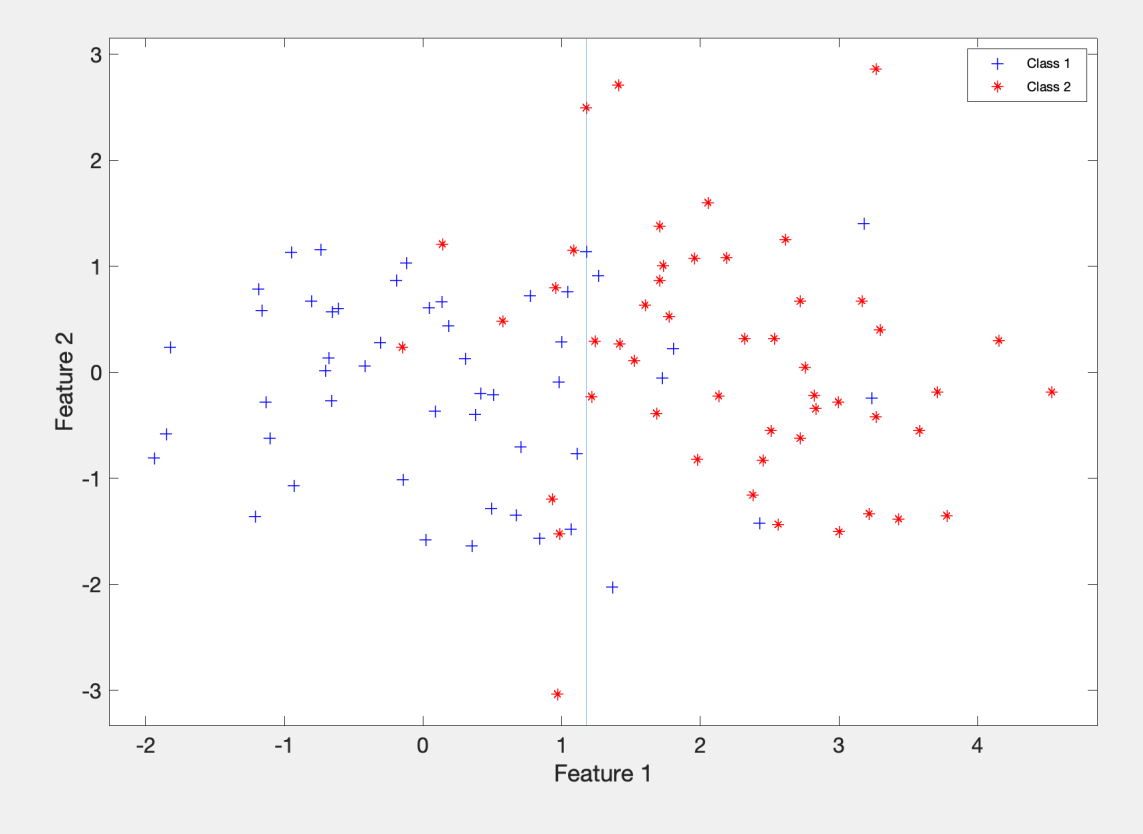

Figure 3: Scatter plot of test dataset with the boundary line (θ)

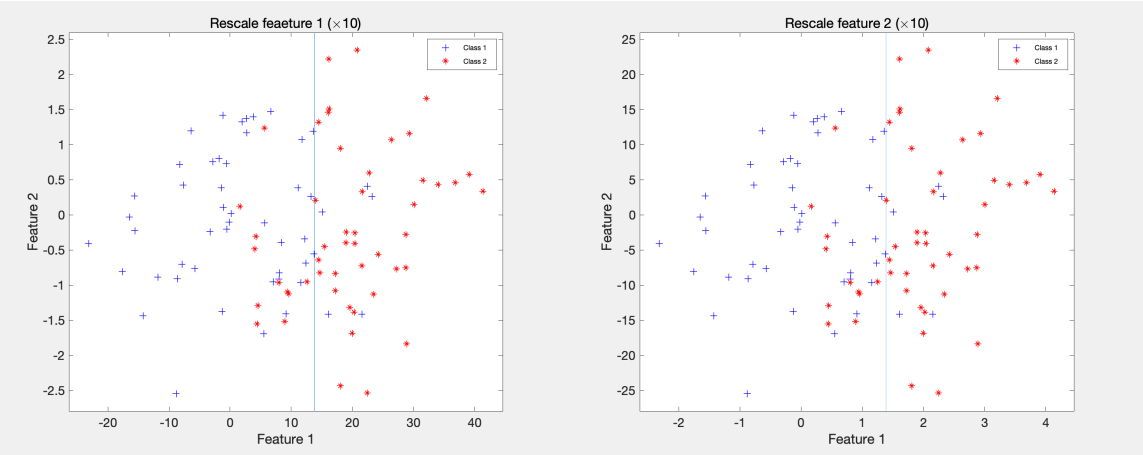

Figure 4: Rescaled dataset and the scatter plot with the boundary line (θ)

$$\omega_i^{t+1} = \omega_i^t \beta_t^{1-|h_t(x_i)-y_i|}$$

The implementation of AdaBoost and its relevant test function are shown in Figure 5 and 6.

6 Question f.

As the Adaboost algorithm I implemented, I stored the weights vectors in a matrix, and I draw the colorbar of the matrix, which shows the weights distribution. Use object 57 as an example, it belongs to the red class but stays closer to the blue class, which is easy to be misclassified. In AdaBoost algorithm, it is easy to access a higher weight, and, indeed, the colour value of object 57 is obviously higher than the background, the background I mentioned here is the objects which are easy to be classified, that is why their colour is deep and the weight value is low based on AdaBoost. The visualised results are shown in Figure 7 and 8.

7 Question g.

The test error I obtained is 27.50% when the iteration number is 100. When doing this question, I feel clearly that the code processing time is obviously longer. In the beginning, the test error decreases remarkably, then it stays in a certain level lower than 0.3. Based on the result I obtained shown in Figure 9, iteration number equalling to 60 shows the lowest error rate, then I plotted the weight distribution on each object, shown in Figure 10.

8 Question h.

I plotted the learning curve with both apparent error and true error. The apparent error goes to 0 in a small number of iteration. The true error decreases to a certain level and then stay around the level when the number of training objects is larger than 10, when training object is 2, 4 or 6, test error stays in a high level larger than 0.5. The result is shown in Figure 11

```
function [beta, para, w_matrix] = AdaBoost(data, label, T)
% input: dataset, label and number of iteration
% output: predicted label, beta and parameters for all base classifiers
[n, ~] = size(data);
% initialise weights
w = ones(n, 1) / n;
% to store T times iteration results
w_matrix = ones(n, T);
beta = zeros(T, 1);
para = zeros(T, 3);
% T times iteration (assignment requires 100 times)
for i = 1: T
    % p is current normalised weight
    p = w ./ sum(w);
    % store it in the matrix
    w_matrix(:, i) = p;
    % call WeightedWeakLearner() to access
    [feature, theta, sign] = WeightedWeakLearn(data, label, p);
    para(i, 1) = feature;
    para(i, 2) = theta;
    para(i, 3) = sign;
    [error, pred] = e(feature, theta, sign, data, label, p);
    % if error is already error, then break
    if error == 0
        break;
    end
    % set beta value
    beta(i) = error / (1 - error);
    % set the new weights vector
    w = w .* (beta(i) .^ (1 - abs(pred - label)));
end
end
```

Figure 5: MATLAB code for question e., AdaBoost implementation

```
function [AdaLabel] = AdaClassify(data, beta, para)
% input: beta, para (including feature, theta and sign) and data
% output: classification result
% initialise number of object and number of iterations
n = size(data, 1);
T = size(beta, 1);
hypo = zeros(n, T);
% apply the hypothesis to calculate the baseline to be compared
baseline = 0.5 * sum(log(1 ./ beta));
for i = 1: T
    % load feature, theta and sign
    feature = para(i, 1);
    theta = para(i, 2);
    sign = para(i, 3);
    Theta = ones(n, 1) * theta;
    % prevent error occuring
    if feature == 0
        break;
    end
    % to check the relationship with label '1'/'2' and threshold
    if sign == 1
        hypo(:, i) = data(:, feature) - Theta >= 0;
    else
        hypo(:, i) = data(:, feature) - Theta <= 0;
    end
    hypo(:, i) = hypo(:, i) * log(1 / beta(i));
end
% fulfill the hypothesis equation or otherwise, 0/1
AdaLabel = sum(hypo, 2) >= baseline;
% two class labels are 1 and 2 respectively, so need +1
AdaLabel = AdaLabel + 1;
end
```

Figure 6: MATLAB code for question e., AdaBoost classifier test

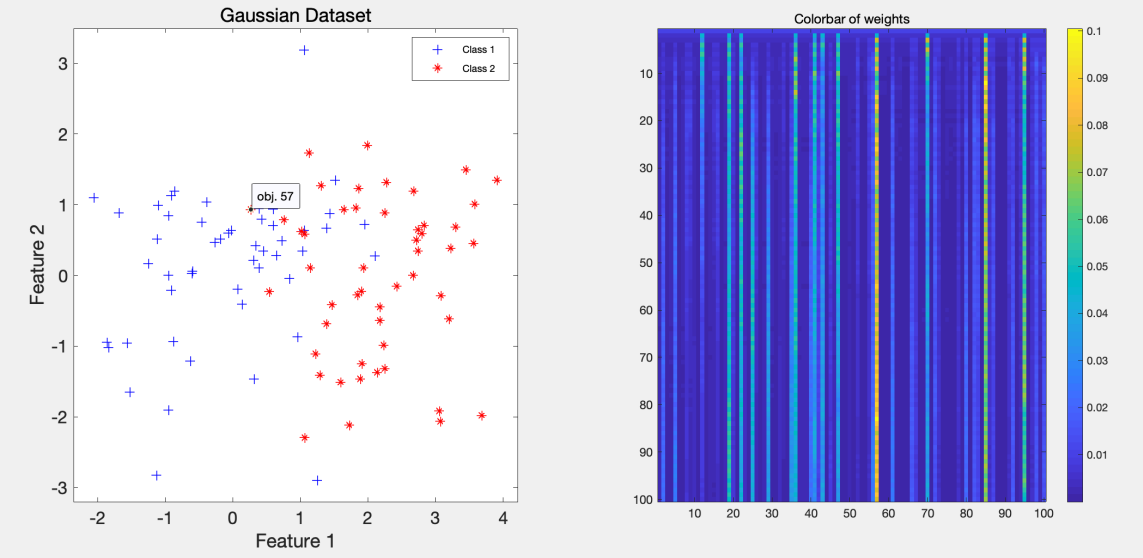

Figure 7: Scatterplot of Gaussian dataset in c. and its weights matrix colorbar

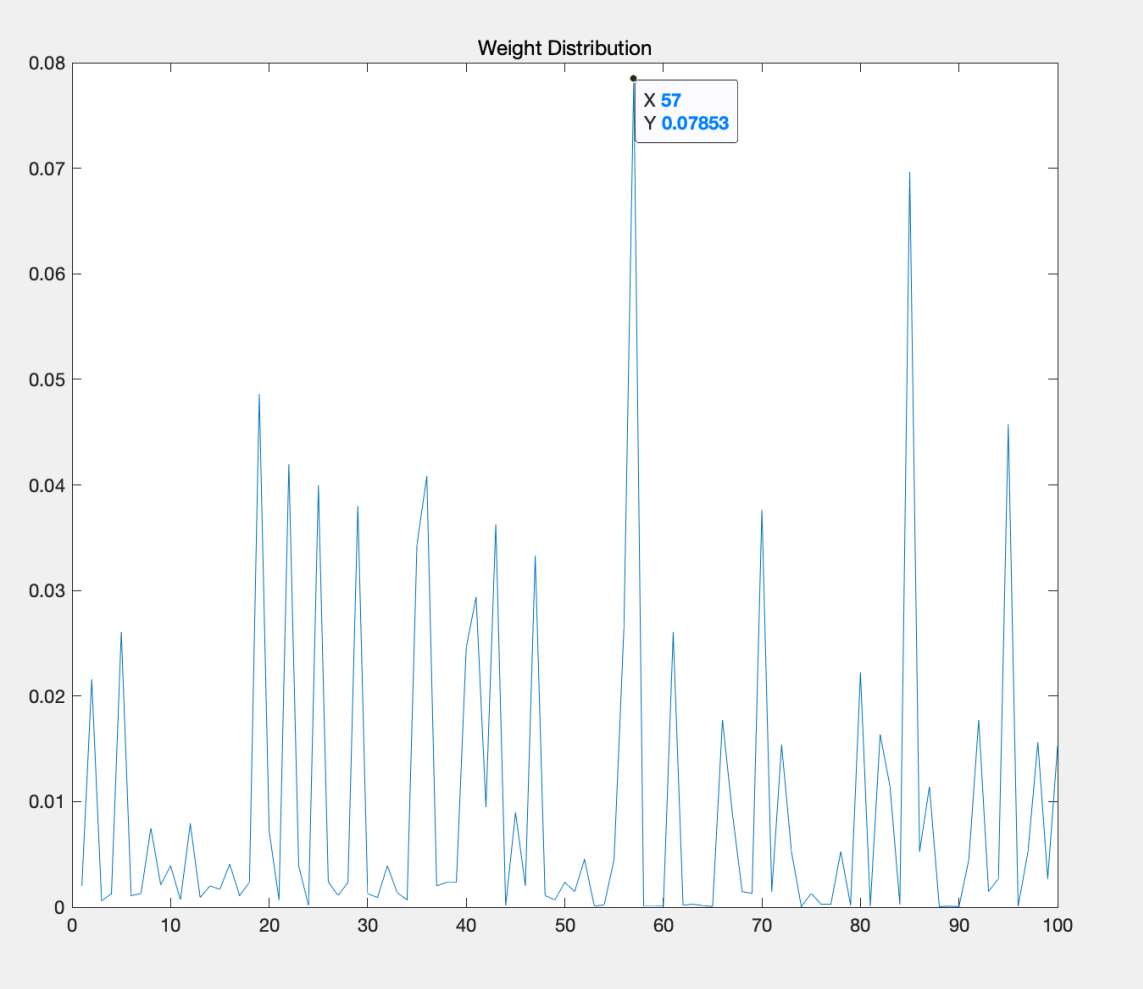

Figure 8: The distribution of weight

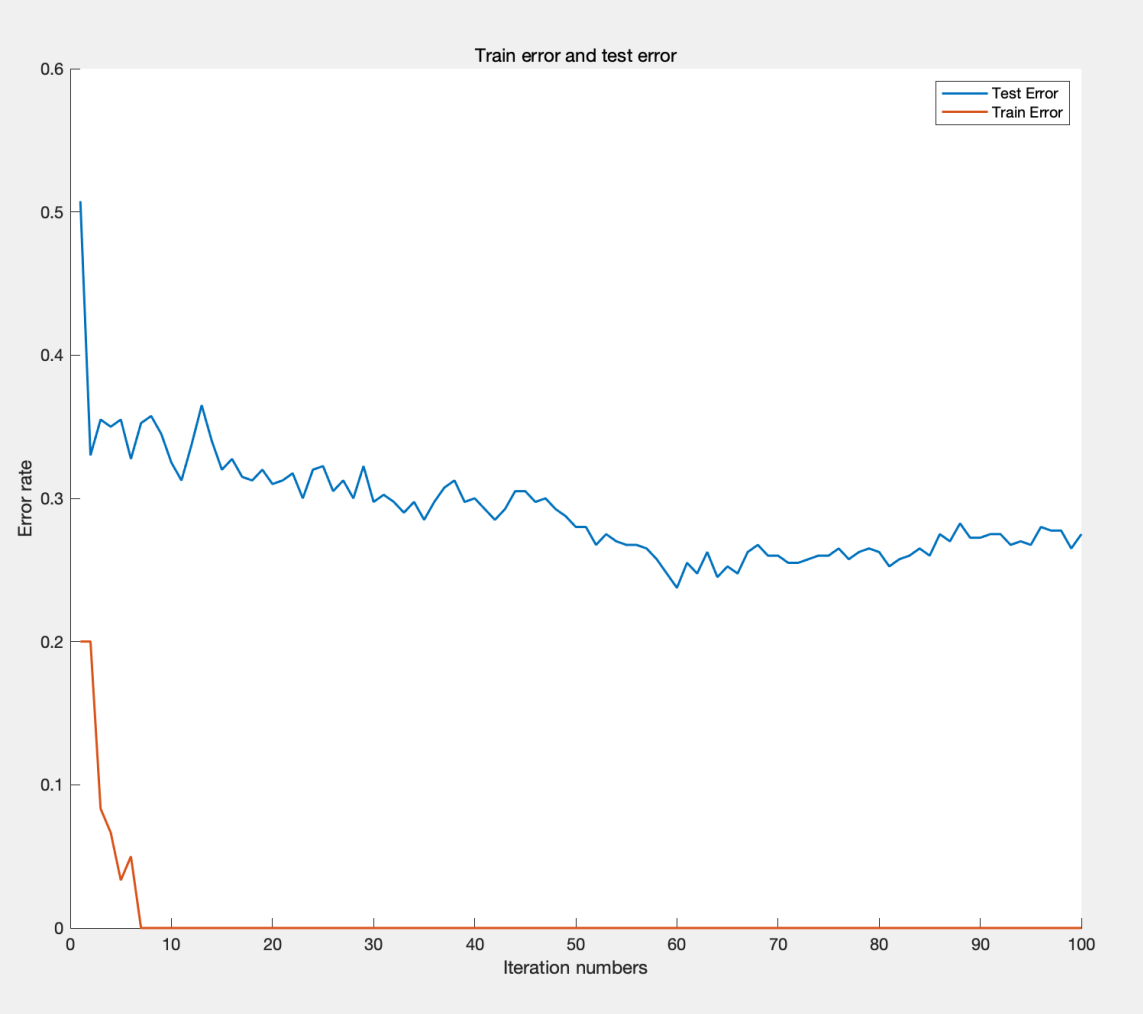

Figure 9: Apparent error and true error on Fashion dataset

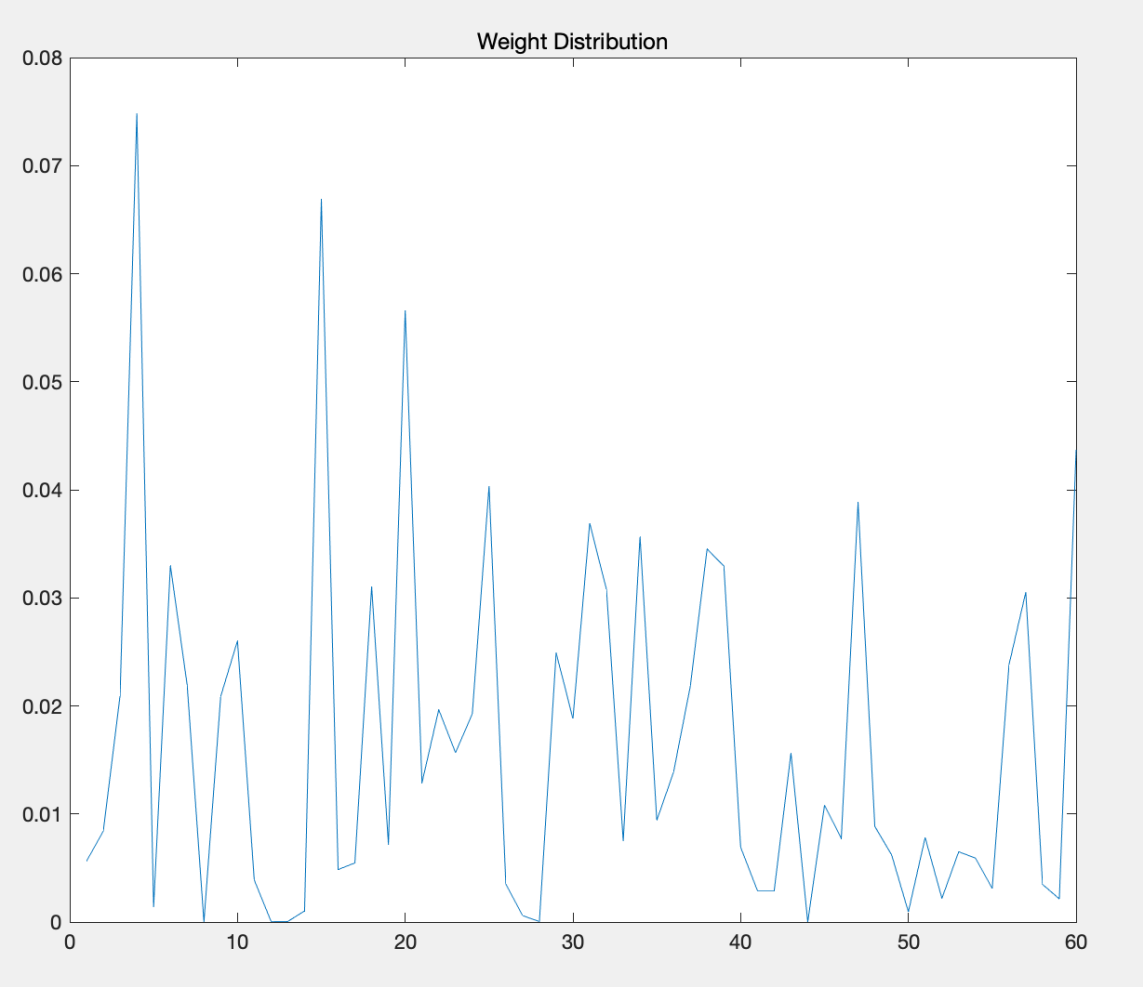

Figure 10: The distribution of weight

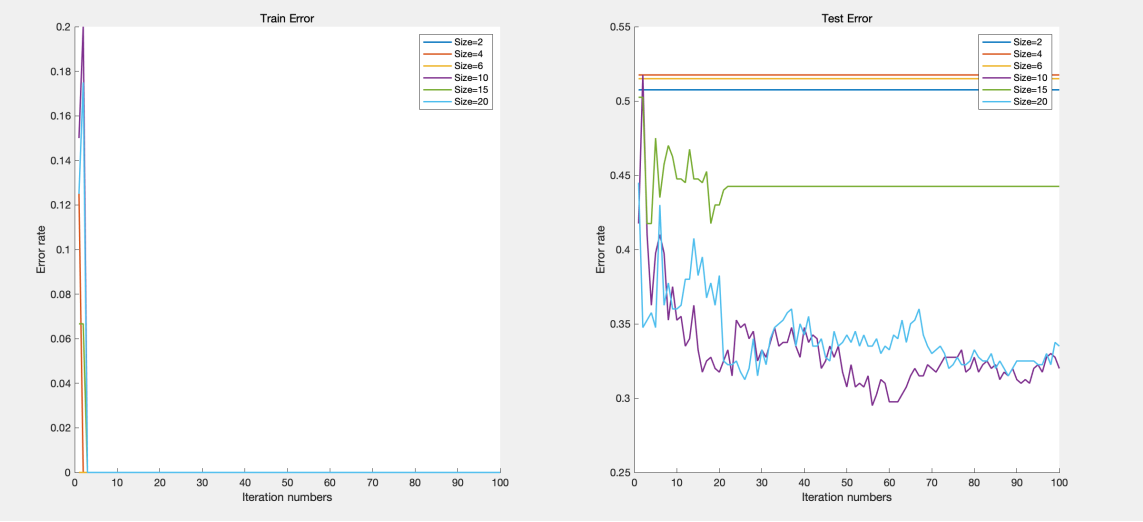

Figure 11: The learning curve for n=[2,4,6,10,15,20] training objects per class