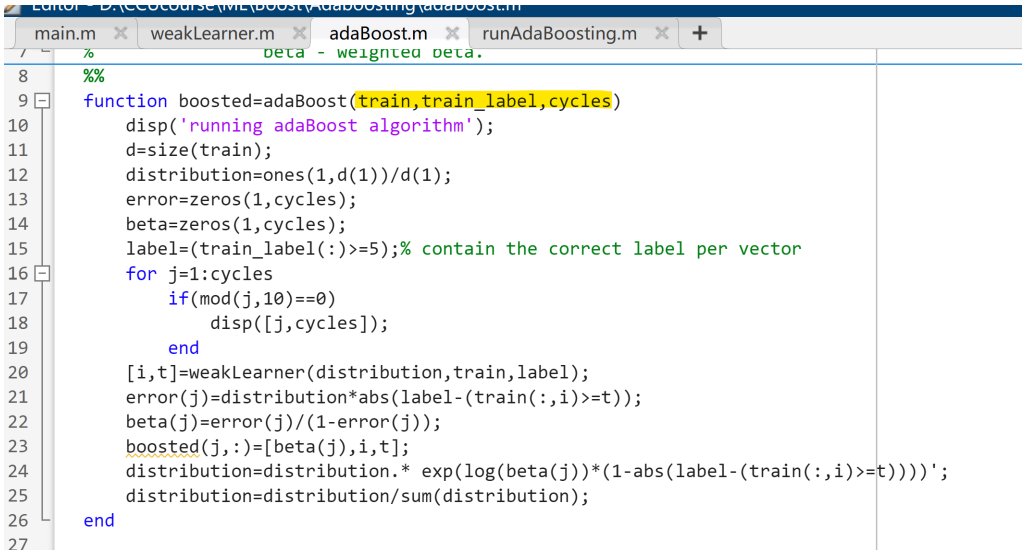
Adative Boosting 報告

資工四 408410098 蔡嘉祥

1.



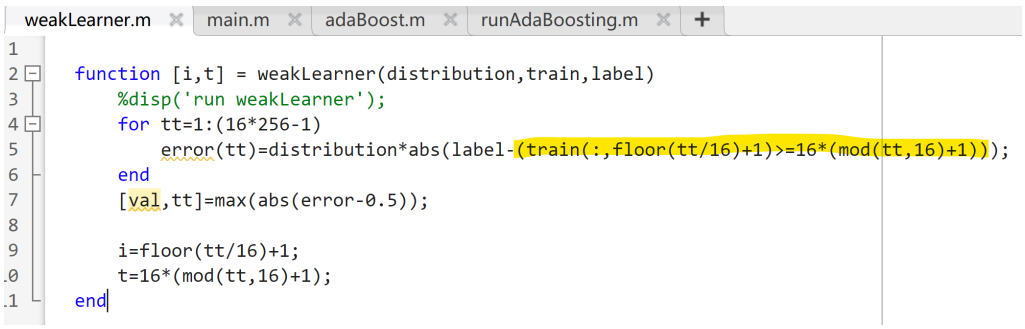
Parameters:

1. `train` ：training data
2. `train_label` ：每筆 training data 的答案
3. `cycle` ：要做多少 iterations (課程投影片的 T)

2.

資料的形式應該是一個 row 就是一筆資料，每個 column 就是他的各個 feature。

How to decide :



使用16個 $h_k(x) = s \cdot (x_j - \theta), \text{where } \theta_k = 16*1, 16*2, 16*3, ..., 16*16; s = 1$。

在 256*16-1 = 4095 個產生的 predictions 中，每 16 個一組，共256組，每組代表一個 feature，會依據該 feature 產生 16 個 predictions

但是他這樣寫 `for i=1:(16*256)-1` ，那第一個 feature 好像只會產生 15 個 predictions ?

$$\text{predictions for a single data train}[j] = \begin{bmatrix} & h_2(x_{j,1}) & \dots & h_{16}(x_{j,1}) \\ h_1(x_{j,2}) & h_2(x_{j,2}) & \dots & h_{16}(x_{j,2}) \\ \dots & \dots & \dots & .... \\ h_1(x_{j,256}) & h_2(x_{j,256}) & \dots & h_{16}(x_{j,256}) \end{bmatrix}$$

($i = 1 \rightarrow 15$ for `floor(i/16)+1` =1, $\theta = 16 * 2, ...16 * 16$)

Learning Algorithm A (.i.e How to train the weaklearner ?)

weakLearner.m | main.m | adaBoost.m | runAdaBoosting.m

```
function [i,t] = weakLearner(distribution,train,label)
    %disp('run weakLearner');
    for tt=1:(16*256-1)
        error(tt)=distribution*abs(label-(train(:,floor(tt/16)+1)>=16*(mod(tt,16)+1)));
    end
    [val,tt]=max(abs(error-0.5));

    i=floor(tt/16)+1;
    t=16*(mod(tt,16)+1);
end
```

16*256-1=4095 個 errors，16個一組 (第一組只有15個)，為每個 feature_i 使用不同的 h_k (16個) 的 predictions 與 groundtruth 的異(1)同(0) $\times \text{distribution}_j^t$ 相加 (他使用 dot 的做法加速)

(distributuion 為上課程投影片的 u)。

0.5的錯誤率代表隨機，所以與 0.5 差距最大就代表使用 $h_k(x_i)$ 最為精準(不管是正面的還是負面的最準)。

所以，就去找哪一個被 tt index 到的 $\theta_t = 16 * (tt\%16 + 1)$ 以及其對應的 feature index i 所產生出來的 error 與 0.5 差距最大的，傳回去，為更新 distribution (u) 做準備。

Does it use bootstrapped dataset? & How D^t is obtain for each iteraion?

```
for j=1:cycles
    %if(mod(j,10)==0)
    %    disp([j,cycles]);
    %end
    [i,t]=weakLearner(distribution,train,label);
    error(j)=distribution*abs(label-(train(:,i)>=t));
    beta(j)=error(j)/(1-error(j));
    boosted(j,:)=[beta(j),i,t];
    distribution=distribution.* exp(log(beta(j))*(1-abs(label-(train(:,i)>=t))))';
    distribution=distribution/sum(distribution);
```

他沒有使用 boostrapped dataset。 每一次的 iteration 他是直接把傳進來的 `train` 全部傳入 `weakLearner` 裡面。

3.

其中綠色的部分可以看到，他並沒有把 $x_i - \theta_i$ 的結果再去乘以其他數字 ，所以應該 s 都 = 1

$$h = [\text{train}(:,i) >= t]$$
$$\Rightarrow h = 1 * \text{sign}(\text{train}(:,i) - t)$$

main.m | weakLearner.m | adaBoost.m * | runAdaBoosting.m | test.m | getError.m

```
%%
function boosted=adaBoost(train,train_label,cycles)
    disp('running adaBoost algorithm');
    d=size(train);
    distribution=ones(1,d(1))/d(1); %u
    error=zeros(1,cycles);
    beta=zeros(1,cycles);
    label=(train_label(:)>=5);% contain the correct label per vector
    for j=1:cycles
        %if(mod(j,10)==0)
        %    disp([j,cycles]);
        %end
        [i,t]=weakLearner(distribution,train,label);
        error(j)=distribution*abs(label-(train(:,i)>=t));
        beta(j)=error(j)/(1-error(j));
        boosted(j,:)=[beta(j),i,t];

        distribution=distribution.* exp(log(beta(j))*(1-abs(label-(train(:,i)>=t ))))';
        distribution=distribution/sum(distribution); %normalize to 0~1
        if j < 4
           fprintf("%d turns\n, featureID=%d, theta=%d\n",j,i,t);
           fprintf("blending weight = %f\n",log(beta(jX)));
        end

end
```

他的實驗有兩部分:

- 在相同 iterations(100)下 ，training 使用 100 , 200, ... 1000 (whole data)
- iterations 不同 : 20, 40, ... ,100

其中，若使用 data 數量一樣，則前3個iterations 情況都相同。

#data	3 iterations (i, θ)
100	(170,160), (26,192), (74,144)
200	(170,128), (10,64), (29,192)
300	(11,16), (170,128), (26,160)
400	(11,16), (170,128), (74,16)
500	(11,16), (170,128), (74,16)
600	(11,16), (170,80), (74,16)
700	(11,16), (170,80), (74,16)
800	(11,80), (170,80), (74,16)
900	(11,80), (170,80), (58,16)

#data	3 iterations (i, θ)
1000(whole)	(11,80), (170,80), (58,16)

4.

blending weight:

main.m | weakLearner.m | adaBoost.m | runAdaBoosting.m | test.m | getError.m

```
%%
% file: getError.m
% This function calculates the error returned by the current run of the weak learner.
%%
function [errorTrain,errorTest]=getError(boost,train,train_label,test,test_label)
    disp('run getError');
    d=size(boost);
    num=size(train);
    prediction=zeros(num(1),1);
    % geting the train error
    for h=1:d(1)
        prediction=prediction-log(boost(h,1))*(train(:,boost(h,2))>=boost(h,3));
    end
    temp=-sum(log(boost(:,1)))/2;
    errorTrain=sum(abs((train_label>=5)-(prediction>=temp)))/num(1);
    prediction=zeros(1000,1);
    % geting the test error
    for h=1:d(1)
        prediction=prediction-log(boost(h,1))*(test(:,boost(h,2))>=boost(h,3));
    end
    errorTest=sum(abs((test_label>=5)-(prediction>=temp)))/1000;

```

可以看到 blending weight 在程式中是使用 `log(boost(h,1))` ，而回追回去 boost:

main.m | weakLearner.m | adaBoost.m | runAdaBoosting.m | test.m | getError.m

```
%%
function boosted=adaBoost(train,train_label,cycles)
    disp('running adaBoost algorithm');
    d=size(train);
    distribution=ones(1,d(1))/d(1); %u
    error=zeros(1,cycles);
    beta=zeros(1,cycles);
    label=(train_label(:)>=5);% contain the correct label per vector
    for j=1:cycles
        %if(mod(j,10)==0)
        %    disp([j,cycles]);
        %end
        [i,t]=weakLearner(distribution,train,label);
        error(j)=distribution*abs(label-(train(:,i)>=t));
        beta(j)=error(j)/(1-error(j));
        boosted(j,:)=[beta(j),i,t];

        distribution=distribution.* exp(log(beta(j))*(1-abs(label-(train(:,i)>=t ))))';
        distribution=distribution/sum(distribution); %normalize to 0~1
        if j < 4
            fprintf("%d turns\n, featureID=%d, theta=%d\n",j,i,t);
            fprintf("blending weight = %f\n",log(beta(j)));
        end

end
```

boosted(j, 1) 就是 $\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}$ ，所以 blending weight = $\ln(\sqrt{\frac{\epsilon_t}{1-\epsilon_t}})$

#data	3 iterations blending weight
100	-1.098612, -1.227230, 1.007370

#data	3 iterations blending weight
200	-0.969401, -1.143421, -0.826497
300	-1.028715, -0.935677, -0.716969
400	-0.944462, -0.832696, 0.739329
500	-0.914891, -0.790367, 0.823993
600	-0.952744, -0.822025, 0.819820
700	-0.930333, -0.742270, 0.774582
800	-0.950670, -0.716032, 0.712811
900	-0.944462, -0.764451, 0.706830
1000(whole)	-0.974422, -0.732568, 0.701601