Machine Learning	Prof. Dr. Bernt Schiele
	Dr. Paul Swoboda & Dr. Gerard Pons-Moll
Exercise Sheet 9 - 17.12.2018	due: 08.01.2019, 14:15

Exercise 17 - GentleBoost

You are supposed to implement GentleBoost as discussed in the lecture using a real-valued decision stump as the weak base classifier.

a. **(3 Points)** In the base step of the boosting procedure one has to fit the weak classifier to the weighted training data. In GentleBoost this is done via weighted least squares. As the weak classifier f we use a decision stump:

$$f(x) = a\mathbb{1}_{\langle w,x\rangle+b>0} + c,$$

with $a, b, c \in \mathbb{R}$ and $w \in \mathbb{R}^d$.

Assume that w is fixed. Derive the optimality condition for a and c for minimizing the weighted least squares loss $L(f)$,

$$L(f) = \sum_{i=1}^{n} \gamma_i \big(Y_i - f(X_i)\big)^2,$$

where $\gamma \in \mathbb{R}^n$ are the weights.

b. **(3 Points)** Write a Matlab-function

`[a,b,c,minError]=FitStump(X,Y,w,gamma)`

which given the fixed vector $w \in \mathbb{R}^d$ and the weights $\gamma \in \mathbb{R}^n$ (n is the number of training points) derives the optimal decision stump, that is derive the optimal parameters a, b, c, of

$$f(x) = a\mathbb{1}_{\langle w,x\rangle+b>0} + c.$$

(as usual $X \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^n$).

c. **(4 Points)** Write a Matlab-function

`[W,aparam,bparam,cparam]=GentleBoost(X,Y,MaxIter)`

which given the training data X, Y and the number of maximal iterations `MaxIter` returns

$$\texttt{W} \in \mathbb{R}^{d\times k}, \texttt{aparam} \in \mathbb{R}^k, \texttt{bparam} \in \mathbb{R}^k, \texttt{cparam} \in \mathbb{R}^k,$$

where k is the number of used weak classifiers ($k \leq$ `MaxIter`).

As the weak learner use the decision stump of b).

d. **(2 Points)** Apply the GentleBoost classifier with `MaxIter` = 100 to the USPS data using one-versus-all classification. Save your predictions `Pred` and the corresponding test error `TestError` in the file `USPSResults`. Plot the training and test error as a function of the number of iterations done. Save the plot as `PlotTrainTestError`. How does the test error compare to the one of the support vector machine ?

Hints:

a. For the implementation of the function `FitStump`

- You do not need to check all possible thresholds b - think about how many different possible thresholds b exist which yield different results for the weighted least squares error ? Compute for each possible threshold first the optimal parameters a and b and then the corresponding weighted least squares error. Take the threshold which yields the smallest error.
- The function `cumsum` which computes the cumulative sum of a vector might be useful.
- One can very efficiently implement this function using vectorization. There is no need for any for loop !

b. For the implementation of the function `GentleBoost` draw the weight vector w uniformly from the unit sphere (`w=randn(dim,1); w=w/norm(w);`)

As usual, please zip all files into one and name it using the filename convention and then send it to your respective tutor.

Solution:

a. For fixed $w \in \mathbb{R}^d$ we have to derive the parameters a, b, c which optimize the weighted least squares loss,

$$L(f) = \sum_{i=1}^{n} \gamma_i \big(Y_i - a\mathbb{1}_{\langle w, X_i\rangle + b > 0} - c\big)^2.$$

We get as conditions which hold at the optimum of $L(f)$ (note that $L(f)$ is convex in the parameters a, c so that the first order condition is sufficient for a minimum),

$$\frac{\partial}{\partial a} L(f) = 2\sum_{i=1}^{n} \gamma_i (a\mathbb{1}_{\langle w, X_i\rangle + b > 0} + c)\mathbb{1}_{\langle w, X_i\rangle + b > 0} - 2\sum_{i=1}^{n} \gamma_i Y_i \mathbb{1}_{\langle w, X_i\rangle + b > 0} = 0,$$
$$\frac{\partial}{\partial c} L(f) = 2\sum_{i=1}^{n} \gamma_i (a\mathbb{1}_{\langle w, X_i\rangle + b > 0} + c) - 2\sum_{i=1}^{n} \gamma_i Y_i = 0.$$

We can solve the last equation for c and get,

$$c = \frac{\sum_{i=1}^{n} \gamma_i Y_i}{\sum_{i=1}^{n} \gamma_i} - a\frac{\sum_{i=1}^{n} \gamma_i \mathbb{1}_{\langle w, X_i\rangle + b > 0}}{\sum_{i=1}^{n} \gamma_i}.$$

Solving the first equation for a yields,

$$a = \frac{\sum_{i=1}^{n} \gamma_i Y_i \mathbb{1}_{\langle w, X_i\rangle + b > 0}}{\sum_{i=1}^{n} \gamma_i \mathbb{1}_{\langle w, X_i\rangle + b > 0}} - c.$$

Plugging the last equation into the one for c, we get

$$c = \frac{\sum_{i=1}^{n} \gamma_i Y_i (1 - \mathbb{1}_{\langle w, X_i\rangle + b > 0})}{\sum_{i=1}^{n} \gamma_i (1 - \mathbb{1}_{\langle w, X_i\rangle + b > 0})},$$

and plugging this result into the equation for a, we get

$$a = \frac{\sum_{i=1}^{n} \gamma_i Y_i \mathbb{1}_{\langle w, X_i\rangle + b > 0}}{\sum_{i=1}^{n} \gamma_i \mathbb{1}_{\langle w, X_i\rangle + b > 0}} - \frac{\sum_{i=1}^{n} \gamma_i Y_i (1 - \mathbb{1}_{\langle w, X_i\rangle + b > 0})}{\sum_{i=1}^{n} \gamma_i (1 - \mathbb{1}_{\langle w, X_i\rangle + b > 0})}.$$

b. The key trick to compute the best decision stump is that for n training points there are only $n + 1$ possible thresholds b. Note, that for every possible threshold we can compute the optimal parameters a, c as derived in a). Thus we compute for each possible threshold the optimal parameters a, c for this threshold and compute the corresponding weighted least

squares error. Then the parameters a, b, c are selected which yield the minimal weighted least squares error. Note, that the weighed least squares error can be computed as

$$L(f) = \sum_{i=1}^{n} \gamma_i Y_i^2 + (a^2 + 2ac) \sum_{i=1}^{n} \gamma_i \mathbb{1}_{\langle w, X_i \rangle + b > 0} + c^2 \sum_{i=1}^{n} \gamma_i - 2(a+c) \sum_{i=1}^{n} \gamma_i Y_i \mathbb{1}_{\langle w, X_i \rangle + b > 0}$$
$$= 1 + (a^2 + 2ac) \sum_{i=1}^{n} \gamma_i \mathbb{1}_{\langle w, X_i \rangle + b > 0} + c^2 - 2a \sum_{i=1}^{n} \gamma_i Y_i \mathbb{1}_{\langle w, X_i \rangle + b > 0} - 2c \sum_{i=1}^{n} \gamma_i Y_i,$$

where we have used that $Y_i^2 = 1$ and $\sum_{i=1}^{n} \gamma_i = 1$. All the operations can be vectorized (written as matrix vector multiplications) and thus are very fast in Matlab. The drawback is that the code is slightly harder to understand.

```
function [a,b,c,minerror]=FitStump(X,Y,w,gamma)
LinOut=X*w;
[SortLinOut,Idx]=sort(LinOut,'descend'); % order the projected data in descending
order

Product = Y.*gamma;
Product=Product(Idx); % reorder the product of weights and gamma as LinOut
Gamma=gamma(Idx); % reorder the weights gamma as LinOut
CumProduct = cumsum(Product); CumProduct=CumProduct(1:end-1); % cumulative sum
of Product - since a,c are not defined if b<min(LinOut) or b>max(LinOut) we cut
the possible values
CumGamma = cumsum(Gamma); CumGamma = CumGamma(1:end-1); % same procedure for gamma

c = (sum(Product)-CumProduct)./(sum(Gamma)-CumGamma); % computes optimal coefficient
c for every possible threshold (there are at most n+1 different ones)
a = CumProduct./CumGamma - c; % computes optimal coefficient a for every possible
threshold

% compute the error for the optimal parameters for all thresholds
%error = sum(gamma.*(Y- a*( LinOut + b)>0 - c).^2);
error = (1+c.^2).*sum(gamma) + (a.^2 + 2.*a.*c).*CumGamma -2*a.*CumProduct - 2*c.*sum(Product

minerror = min(error);
thresh = find(error == minerror); % find indices which achieve minimal error
numTresh = length(thresh); IdxThresh = ceil(numTresh/2); % if there is more than
one take the middle one
thresh = thresh(IdxThresh);
b=-(SortLinOut(thresh)+ SortLinOut(thresh+1))*0.5; % threshold is placed in the
middle
a=a(thresh); c=c(thresh);
```

c. The code for GentleBoost.

```
function [W,aparam,Offset,cparam] = GentleBoost(X,Y,MaxIter)
% implements GentleBoost (page 353) from
% Friedman, Hastie, Tibshirani:  Additive logistic boosting, a statistical
% view of boosting
% input:  Designmatrix X
% Outputvector Y
% Maximal number of iterations MaxIter
% output:  an array of weight vectors W
% the parameters of the decision stumps a*(X*w+Offset>0)+c
```

```
dim = size(X,2); % dimension of the training data
num = size(X,1); % number of training points

gamma = 1/num*ones(num,1); % weight vector - initialize uniformly (has always
to sum up to one)

W = zeros(dim,MaxIter); Gamma=zeros(num,MaxIter);
ExpLoss =zeros(MaxIter,1); ZeroOneLoss=zeros(MaxIter,1);
CurF=zeros(num,1);

counter=1; Gamma(:,1)=gamma;
while(counter<=MaxIter)
% generate random weight vector
w = randn(dim,1); w=w/norm(w); W(:,counter)=w;
[aparam(counter),Offset(counter),cparam(counter),Loss] = FitStump(X,Y,w,gamma);

FOut = aparam(counter)*double(X*w+Offset(counter)>0)+cparam(counter); % the decision
stump

gamma = gamma.*exp(-Y.*FOut); % update the weight vector
gamma = gamma/sum(gamma); % normalize the weights
Gamma(:,counter)=gamma;

CurF = CurF + FOut; % update function

% compute exponential and zero-one loss
ExpLoss(counter) = sum(exp(-Y.*CurF))/num;
ZeroOneLoss(counter) = sum(Y.*CurF <0)/num;
disp(['Iteration:  ',num2str(counter),' - ExpLoss:  ',num2str(ExpLoss(counter)),'
- ZeroOneLoss:  ',num2str(ZeroOneLoss(counter))]);

counter=counter+1;
end
```

d. After 100 iterations we achieve a test error of 15.4% (see left plot in Figure 1). Compared to the SVM with a Gaussian kernel with which we achieved a test error of 6.53% this is quite bad. However, one observes that the test error is still decreasing which suggests that more iterations would be helpful. In practice one would use cross-validation in order to find the best stopping point for the iterations. After 2000 iterations (see the right plot in Figure 1) we achieve a test error of 8.57%. This is still worse than the SVM but is at least comparable. The required high number of iterations suggest that the true decision boundary is highly nonlinear so that we need a lot of simple decision stumps to approximate it well. A fair comparison with the SVM has to consider that the GentleBoost has only one free parameter (the number of iterations), whereas we have to select two parameters for the SVM (kernel width and regularization parameter). The training and test error is plotted in Figure 1. Nevertheless boosting is often used in computer vision. In particular, since the boosting classifier can be evaluated very fast which is important for realtime classification. One tries to eliminate performance problems by adding many features, which are then selected sequentially (weight vector of the form $w_i = 1$ for selected feature and otherwise zero.

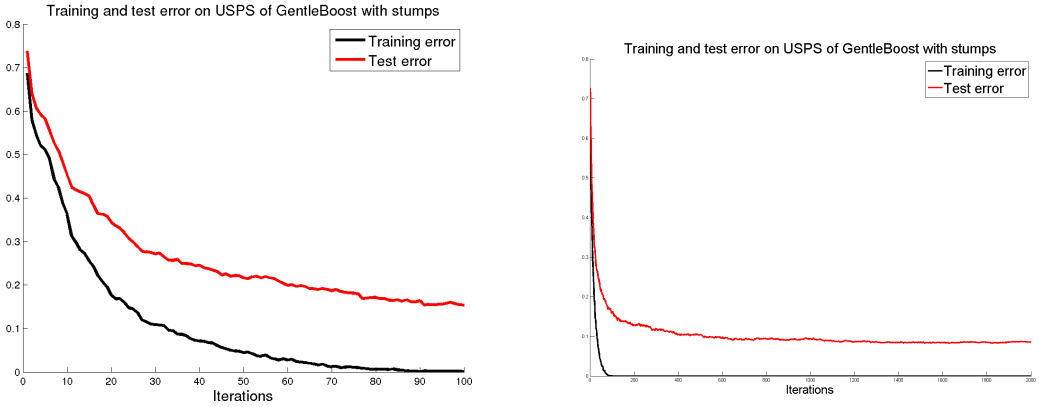

Figure 1: Left: after 100 iterations, Right: after 2000 iterations. The training and test error of GentleBoost with decision stumps on the USPS dataset. There is no sign of overfitting. On the other hand the performance compared to the SVM is not competitive.