GML AdaBoost Matlab Toolbox Manual

This manual describes the usage of GML AdaBoost matlab toolbox, and is organized as follows: in the first section will introduce you to the basic concept of the toolbox, then we give an example script that uses the toolbox, section 3 speaks about all available functions and classes and section 4 is Q and A.

Introduction

GML AdaBoost Matlab Toolbox is set of matlab functions and classes implementing a family of classification algorithms, known as Boosting.

Implemented algorithms

So far we have implemented 3 different boosting schemes: Real AdaBoost, Gentle AdaBoost and Modest AdaBoost.

Real AdaBoost (see [2] for full description) is the generalization of a basic AdaBoost algorithm first introduced by Fruend and Schapire [1]. Real AdaBoost should be treated as a basic "hardcore" boosting algorithm.

Gentle AdaBoost is a more robust and stable version of real AdaBoost (see [3] for full description). So far, it has been the most practically efficient boosting algorithm, used, for example, in Viola-Jones object detector [4]. Our experiments show, that Gentle AdaBoost performs slightly better then Real AdaBoost on regular data, but is considerably better on noisy data, and much more resistant to outliers.

Modest AdaBoost (see [5] for a full description) – regularized tradeoff of AdaBoost, mostly aimed for better generalization capability and resistance to overfitting. Our experiments show, that in terms of test error and overfitting this algorithm outperforms both Real and Gentle AdaBoost.

Available weak learners

Now a tree learner is available (there was only stumps in version 0.1). You can define the number of maximum splits that would be done during the training. You can still use a stump learner – it's just a tree with only one split.

CART

CART is an acronym for Classification and Regression Trees. Here, we will describe an algorithm for using and building a CART decision tree for classification task.

Decision tree is a tree graph, with leaves representing the classification result and nodes representing some predicate. Branches of the tree are marked true or false. Classification process in case of decision tree is a process of tree traverse. We start from root and descend further, until we reach the leaf – the value associated with the leaf is the class of the presented sample. At each step we compute the value of the predicate associated with current node. We choose next node (or leaf) that is connected with current by the branch with the value of current nodes predicate.

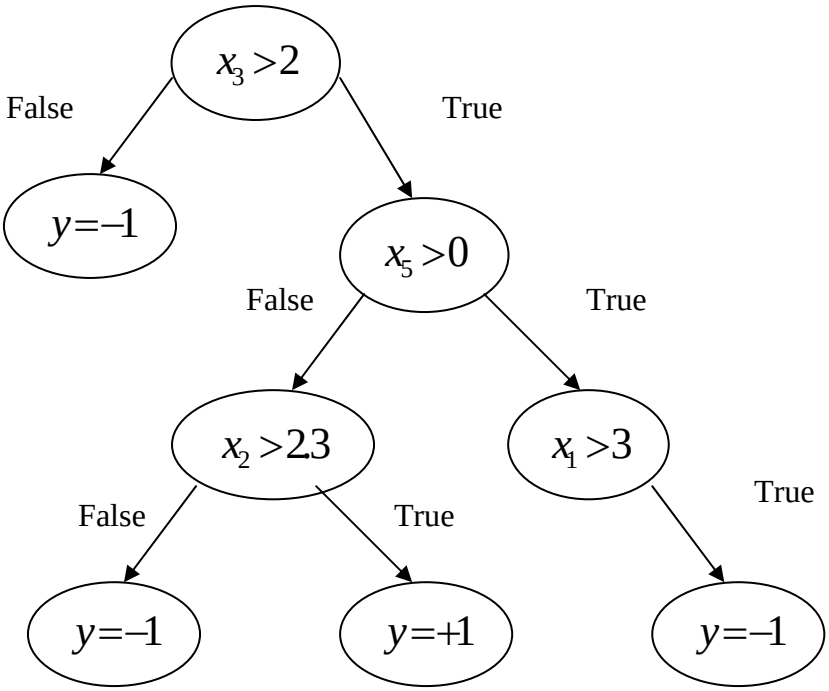

Figure 1. CART example.

Let $S = \langle (x^1, y^1), ..., (x^m, y^m) \rangle$ be a sequence of training examples, where each x^j belongs to the *domain* or *instance space* $X \in R^n$ (real valued vector with dimensionality n $x^j = (x_1^j, ..., x_n^j)$), and each *label* y^j belongs to a finite *label space* Y. We will consider binary classification task, where $Y = \{-1, +1\}$.

In toolbox we use the following algorithm for construction a node of CART:

1. For each and all n dimensions find the threshold, that separates S with least error;
2. Choose dimension *i* with least error, and construct the node:
    a. With predicate $x_i > \Theta$;
    b. Branches true/false, that are connected with leafs, that have respective classification.

Let "error of leaf" be the probability of a sample being misclassified if during the tree traverse we stop at this leaf. To construct the whole tree the following algorithm is used:

1. Construct root node;
2. Choose leave with largest error;
3. Construct node, using only those training samples, that are associated with chosen leaf;
4. Replace chosen leaf with constructed node;
5. Repeat 2-4 until all leafs have zero error, or predefined number of steps done.

To make CART able to learn on weighted training data we only have to evaluate all errors according to weights.

Additional functionalities

Alongside with 3 Boosting algorithms we also provide a class that should give you an easy way to make a cross-validation test.

Authors

This toolbox was implemented by Alexander Vezhnevets – an undergraduate student of Moscow State University. If you have any questions or suggestions, please mail me: vezhnick@gmail.com

Library structure and usage

Library provides a set of functions that implement classifier boosting procedures. Weak learners (classifiers) are implemented as class, while boosting procedures are implemented as global functions.

We provide CART (classification trees) as weak learners. Class "tree_node_w" implements CART. Number of maximum splits (tree depth) is passed as constructor parameter. User should create class object with desired number of splits and pass it to the boosting function. Boosting procedure (GentleAdaBoost, ModestAdaBoost, RealAdaBoost) constructs boosted classifier committee using training set represented by matrix of training samples and their respective labels (see function descriptions for more details).

Cell array of weak classifiers and a vector of their weights represent boosted committee. Actually, each node of CART trees constructed during the training process is represented as an individual weak classifier. Thus a tree with 4 nodes would be represented as a cell array of 4 nodes and 4 respective numbers in vector of weights.

Constructed committee can be saved to text file. This file can be used for analyzes of constructed committee C++ code provided that can load saved committee from file and perform classification with it. See TranslateToC function.

Functions and Classes

function [Learners, Weights, {final_hyp}] = RealAdaBoost(WeakLrn, Data, Labels, Max_Iter, {OldW, OldLrn, final_hyp})

Boosts a weak learner *WeakLrn* using Real AdaBoost algorithm with *Max_Iter* iterations on dataset given in *Data* and *Labels*.

Arguments:

- WeakLrn - weak learner
- Data - training data. Should be DxN matrix, where D is the dimensionality of data, and N is the number of training samples;
- Labels - training labels. Should be 1xN matrix, where N is the number of training samples, any label is either +1 or –1;
- Max_Iter - number of iterations;
- OldW - weights of already built committee (used for further training of already built commitee). Optional parameter;
- OldLrn - learners of already built committee (used for further training of already built committee). Optional parameter;
- final_hyp - output for training data of already built committee (used to speed up further training of already built committee). Optional parameter.

Return:

- Learners - cell array of constructed learners. Each learner is a node of CART tree represented by object of tree_node_w class;
- Weights - weights of learners. This vector has the same size as Learners and represents weight of each learner in final committee;

- final_hyp - output for training data.

function [Learners, Weights, {final_hyp}] = GentleAdaBoost(WeakLrn, Data, Labels, Max_Iter, {OldW, OldLrn})

Boosts a weak learner *WeakLrn* using Gentle AdaBoost algorithm with *Max_Iter* iterations on dataset given in *Data* and *Labels*. The parameters semantic is the same as in RealAdaBoost function.

function [Learners, Weights, {final_hyp}] = ModestAdaBoost(WeakLrn, Data, Labels, Max_Iter, {OldW, OldLrn})

Boosts a weak learner *WeakLrn* using Modest AdaBoost algorithm with *Max_Iter* iterations on dataset given in *Data* and *Labels*. The parameters semantics are the same as in RealAdaBoost function.

function Result = Classify(Learners, Weights, Data)

Classifies *Data* using boosted assembly of *Learners* with respective *Weights*. *Result* will contain real numbers; the signum of those numbers represents the class, and its absolute magnitude is the "confidence" of the decision. To obtain classification one should take signum of *Result.* To regulate the rate of false positive / false negative *Result* could be compared with some threshold. Increasing threshold would reduce false positive rate, but will also increase false negative.

Example:

```
Confidence = Classify(Learners, Weights, X); % obtaining real valued results
Tetta = 0.2;                                 % TP/FP regulating threshold
Y = sign(Confidence - Tetta);                % obtaining classification
```

function code = TranslateToC (Learners, Weights, fid)

Use this function to save constructed classifiers for further use in C++ applications. C++ codes that loads saved committee is provided (see C++ directory for code and example usage). File has the following format:

```
<TN>
<W> <N > <D> <T> <Ts> {<D> <T> <Ts> }
<W> <N > <D> <T> <Ts> {<D> <T> <Ts> }
…
<end>
```

Where:

TN – total number of weak classifiers;
W – weight of weak classifier;
N – number of thresholds representing weak classifier (in CART each node can be represented as the set of thresholds);
D – thresholds dimension;
T – threshold value;
Ts – threshold sing. It's either –1 or +1, resembling if the sample should be greater or lesser than threshold to be classified positive.

Arguments:

- Learners - learners of committee to be saved;
- Weights - weights of committee to be saved;
- fid - opened file id (use fopen to make one).

Return:

- code - equals 1 if everything was alright.

@tree_node_w :

A class that implements a classification tree weak learner. This is the most popular weak learner for the boosting algorithms. It splits data by a set of hyper planes orthogonal to coordinate axis. We use a greedy splitting rule – at each step we perform a split, which best lowers the total tree error.

Class methods:

function tree_node = tree_node_w(max_splits) – constructor. Call to make the object, max_splits specifies the maximum amount of tree splits during training.

function nodes = train(node, dataset, labels, weights) – trains a tree to fit *dataset* in to *labels*, with respect to *weights*. *nodes* – is a cell array that contains terminal tree nodes.
Arguments:

- node - object of tree_node_w class (initialized properly);
- dataset - training data;
- labels - training labels;
- weights - weights of training data. Needed for boosting procedure;

Return:

- nodes - tree is represented as a cell array of its nodes.

function y = calc_output(tree_node, XData) – classifies *XData* with *tree_node* and stores result in *y*.
Arguments:

- tree_node - classification tree node;
- XData - data that will be classified.

Return:

- y - +1, if XData belongs to tree node, -1 otherwise (y is a vector)

@crossvalidation :

A class that helps to perform a crossvalidation. It works like a storage class, you should pass the data alongside with labels and the class automatically splits it into the specified number of subsets. You can then access any fold you want.

Class methods:

function this = crossvalidation(folds) – constructor. Use to create an object with specified number of folds.

function this = Initialize(this, Data, Labels) – initializes the object. *Data* and *Labels* will be split in to the specified in constructor number of folds and stored within the class. *Data* should be a $K \times N$ matrix, where K is the instance space dimensionality and N is the number of training samples; *Labels* must be $1 \times N$ matrix.

function [Data, Labels] = GetFold(this, N) – returns *Data* and *Labels* of fold *N* stored in *this*.

function [Data, Labels] = CatFold(this, Data, Labels, N) – concatenates the fold *N* to *Data* and *Labels*.

Example scripts

Example_1 script

```
% Step1: reading Data from the file
file_data = load('Ionosphere.txt');
Data = file_data(:,1:end-1)';
Labels = file_data(:, end)';
Labels = Labels*2 - 1;

MaxIter = 200; % boosting iterations

% Step2: splitting data to training and control set
TrainData   = Data(:,1:2:end);
TrainLabels = Labels(1:2:end);

ControlData   = Data(:,2:2:end);
ControlLabels = Labels(2:2:end);

% Step3: constructing weak learner
weak_learner = tree_node_w(3); % pass the number of tree splits to the
constructor

% Step4: training with Gentle AdaBoost
[RLearners RWeights] = GentleAdaBoost(weak_learner, TrainData, TrainLabels,
MaxIter);

% Step5: training with Modest AdaBoost
[MLearners MWeights] = ModestAdaBoost(weak_learner, TrainData, TrainLabels,
MaxIter);

% Step6: evaluating on control set
ResultR = sign(Classify(RLearners, RWeights, ControlData));

ResultM = sign(Classify(MLearners, MWeights, ControlData));

% Step7: calculating error
ErrorR  = sum(ControlLabels ~= ResultR)

ErrorM  = sum(ControlLabels ~= ResultM)
```

Comments on the script

Step 1 – data is loaded from txt file. Each line of which is a data sample (feature vector). Last element of line is the class marker (it is 0/1 in example, so that is why "Labels = Labels*2 - 1;" line is required);
Step 2 – we split data in two subsets; half goes to control set, half to training set;
Step 3 – here we construct a tree weak learner, which would be used for boosting. We pass the max number of splits (1 = stump);
Step 4 and 5 – we boost weak learners with two different algorithms, using training set;
Step 6 – calculating classifiers output on control set;
Step 7 – calculating error.

Example_2 script

```
% Step1: reading Data from the file
file_data = load('Ionosphere.txt');
Data = file_data(:,1:end-1)';
```

```
Labels = file_data(:, end)';
Labels = Labels*2 - 1;

MaxIter = 100; % boosting iterations

% Step2: splitting data to training and control set
TrainData   = Data(:,1:2:end);
TrainLabels = Labels(1:2:end);

ControlData   = Data(:,2:2:end);
ControlLabels = Labels(2:2:end);

% and initializing matrices for storing step error
RAB_control_error = zeros(1, MaxIter);
MAB_control_error = zeros(1, MaxIter);
GAB_control_error = zeros(1, MaxIter);

% Step3: constructing weak learner
weak_learner = tree_node_w(3); % pass the number of tree splits to the
constructor

% and initializing learners and weights matices
GLearners = [];
GWeights = [];
RLearners = [];
RWeights = [];
NuLearners = [];
NuWeights = [];

% Step4: iterativly running the training

for lrn_num = 1 : MaxIter

    clc;
    disp(strcat('Boosting step: ', num2str(lrn_num),'/', num2str(MaxIter)));

    %training gentle adaboost
    [GLearners GWeights] = GentleAdaBoost(weak_learner, TrainData,
TrainLabels, 1, GWeights, GLearners);

    %evaluating control error
    GControl = sign(Classify(GLearners, GWeights, ControlData));

    GAB_control_error(lrn_num) = GAB_control_error(lrn_num) + sum(GControl ~=
ControlLabels) / length(ControlLabels);

    %training real adaboost
    [RLearners RWeights] = RealAdaBoost(weak_learner, TrainData, TrainLabels,
1, RWeights, RLearners);

    %evaluating control error
    RControl = sign(Classify(RLearners, RWeights, ControlData));

    RAB_control_error(lrn_num) = RAB_control_error(lrn_num) + sum(RControl ~=
ControlLabels) / length(ControlLabels);

    %training modest adaboost
    [NuLearners NuWeights] = ModestAdaBoost(weak_learner, TrainData,
TrainLabels, 1, NuWeights, NuLearners);
```

```
    %evaluating control error
    NuControl = sign(Classify(NuLearners, NuWeights, ControlData));

    MAB_control_error(lrn_num) = MAB_control_error(lrn_num) + sum(NuControl
~= ControlLabels) / length(ControlLabels);

end

% Step4: displaying graphs
figure, plot(GAB_control_error);
hold on;
plot(MAB_control_error, 'r');

plot(RAB_control_error, 'g');
hold off;

legend('Gentle AdaBoost', 'Modest AdaBoost', 'Real AdaBoost');
xlabel('Iterations');
ylabel('Test Error');
```

Comments on the script

This script implements iterative training of boosted committee. At step 4 we start a cycle in which training is done. For each iteration control error is stored and afterwards control error graphs are displayed. Note, that while training we pass committees constructed on previous steps to boosting function – this is done to speed up the process.

Example_3 script

```
file_data = load('Ionosphere.txt');

%transforming data to toolbox formats
FullData = file_data(:,1:end-1)';
FullLabels = file_data(:, end)';
FullLabels = FullLabels*2 - 1;


MaxIter = 100; % boosting iterations
CrossValidationFold = 5; % number of cross-validation folds

weak_learner = tree_node_w(2); % constructing weak learner

% initializing matrices for storing step error
RAB_control_error = zeros(1, MaxIter);
MAB_control_error = zeros(1, MaxIter);
GAB_control_error = zeros(1, MaxIter);

% constructing object for cross-validation
CrossValid = crossvalidation(CrossValidationFold);

% initializing it with data
CrossValid = Initialize(CrossValid, FullData, FullLabels);

NuWeights = [];

% for all folds
for n = 1 : CrossValidationFold
    TrainData = [];
    TrainLabels = [];
    ControlData = [];
```

```
    ControlLabels = [];

    % getting current fold
    [ControlData ControlLabels] = GetFold(CrossValid, n);

    % concatinating other folds into the training set
    for k = 1:CrossValidationFold
        if(k ~= n)
            [TrainData TrainLabels] = CatFold(CrossValid, TrainData,
TrainLabels, k);
        end
    end

    GLearners = [];
    GWeights = [];
    RLearners = [];
    RWeights = [];
    NuLearners = [];
    NuWeights = [];

    %training and storing the error for each step
    for lrn_num = 1 : MaxIter

        clc;
        disp(strcat('Cross-validation step: ',num2str(n), '/',
num2str(CrossValidationFold), '. Boosting step: ', num2str(lrn_num),'/',
num2str(MaxIter)));

        %training gentle adaboost
        [GLearners GWeights] = GentleAdaBoost(weak_learner, TrainData,
TrainLabels, 1, GWeights, GLearners);

        %evaluating control error
        GControl = sign(Classify(GLearners, GWeights, ControlData));

        GAB_control_error(lrn_num) = GAB_control_error(lrn_num) +
sum(GControl ~= ControlLabels) / length(ControlLabels);

        %training real adaboost
        [RLearners RWeights] = RealAdaBoost(weak_learner, TrainData,
TrainLabels, 1, RWeights, RLearners);

        %evaluating control error
        RControl = sign(Classify(RLearners, RWeights, ControlData));

        RAB_control_error(lrn_num) = RAB_control_error(lrn_num) +
sum(RControl ~= ControlLabels) / length(ControlLabels);

        %training modest adaboost
        [NuLearners NuWeights] = ModestAdaBoost(weak_learner, TrainData,
TrainLabels, 1, NuWeights, NuLearners);

        %evaluating control error
        NuControl = sign(Classify(NuLearners, NuWeights, ControlData));

        MAB_control_error(lrn_num) = MAB_control_error(lrn_num) +
sum(NuControl ~= ControlLabels) / length(ControlLabels);

    end
end
```

```
%saving results
%save(strcat(name,'_result'),'RAB_control_error', 'MAB_control_error',
'CrossValidationFold', 'MaxIter', 'name', 'CrossValid');

% displaying graphs
figure, plot(GAB_control_error / CrossValidationFold );
hold on;
plot(MAB_control_error / CrossValidationFold , 'r');

plot(RAB_control_error / CrossValidationFold, 'g');
hold off;

legend('Gentle AdaBoost', 'Modest AdaBoost', 'Real AdaBoost');
title(strcat(num2str(CrossValidationFold), ' fold cross-validation'));
xlabel('Iterations');
ylabel('Test Error');
```

Comments on the script

This script implements iterative training of boosted committee with cross-validation . The only difference from Example_2 is the use of @crossvalidation class.

TrainAndSave script

```
file_data = load('Ionosphere.txt');
Data = file_data(:,1:end-1)';
Labels = file_data(:, end)';
Labels = Labels*2 - 1;

% Data = Data';
% Labels = Labels';

weak_learner = tree_node_w(2);

% Step1: training with Gentle AdaBoost
[RLearners RWeights] = RealAdaBoost(weak_learner, Data, Labels, 200);

% Step2: training with Modest AdaBoost
[MLearners MWeights] = ModestAdaBoost(weak_learner, Data, Labels, 200);

fid = fopen('RealBoost.txt','w');
TranslateToC(RLearners, RWeights, fid);
fclose(fid);

fid = fopen('ModestBoost.txt','w');
TranslateToC(MLearners, MWeights, fid);
fclose(fid);
```

Comments on the script

This script illustrates TranslateToC function usage. This script is much similar to Example_1.

Q and A:

What version of Matlab should I have for the toolbox to work?

We don’t use any version specific functions, so it should work with most versions of Matlab. It will work on Matlab 6 and Matlab 7 for sure.

Is this toolbox free to use?

Yes. You can use it in any way you want and you don't have to pay anyone. Although you must mention that you used our toolbox, if you publish any results that were obtained using it.

I found a bug!

Please mail me, so I can fix it. avezhnevets@graphics.cs.msu.ru

How should I represent my data to use it in toolbox?

It should be a $K \times N$ matrix, where K is the instance space dimensionality and N is the number of training samples; and a vector ($1 \times N$ matrix) with labels (-1, +1).

How to load my data from txt file to use it in your toolbox?

You can use any method, that matlab provides. For most of txt files "`load('data.txt');`" should work fine.

Can I regulate false positive to false negative rate?

Yes. See description of "Classify" function.

What is the best way to analyze the resulting committee?

We advice you to save committee using "TranslateToC" function and analyze the txt file. Its format is described in "Library structure and usage" section.

Can I use constructed committee in C++ application?

Yes. See TranslateToC function and TrainAndSave example.

Reference

[1] Y Freund and R. E. Schapire. **Game theory, on-line prediction and boosting.** In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

[2] R.E. Schapire and Y. Singer **Improved boosting algorithms using confidence-rated predictions.** *Machine Learning*, 37(3):297-336, December 1999.

[3] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. **Additive logistic regression: A statistical view of boosting.** *The Annals of Statistics*, 38(2):337–374, April 2000.

[4] P. Viola and M. Jones. **Robust Real-time Object Detection.** *In Proc. 2nd Int'l Workshop on Statistical and Computational Theories of Vision -- Modeling, Learning, Computing and Sampling, Vancouver, Canada, July 2001.*

[5] A. Vezhnevets and V. Vezhnevets. **Modest AdaBoost – teaching AdaBoost to generalize better.** Graphicon 2005.

[6] Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). **UCI Repository of machine learning databases** [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.