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.

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

Example script

```
% Step1: reading Data from the file
load diabetes

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

ControlData   = FullData(:,2:2:end);
ControlLabels = FullLabels(:,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 Real AdaBoost
[RLearners RWeights] = RealAdaBoost(weak_learner, TrainData, TrainLabels, 50);

% Step5: training with Modest AdaBoost
[MLearners MWeights] = ModestAdaBoost(weak_learner, TrainData, TrainLabels, 50);

% 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 – we load data file, containing a UCI [6] Diabetes dataset
Step 2 – we split data in two subset, 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.

Other Examples

There is also another example in "TestCompare.m" file. It uses cross-validation to compare Modest and Gentle AdaBoost on UCI Ionosphere dataset. You can change cross-validation folds, boosting iterations, use different training schemes or load your own data simply modifying the script.

Functions and Classes

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

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

WeakLrn – a classification tree weak learner should be passed. You can create it with this line: WeakLrn = tree_node_w(3);

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, any label is either +1 or -1.

Max_Iter specifies the number of iteration that would be used to train the boosted classifier.

OldW, OldLrn an optional parameters, you can pass a trained classifiers committee for additional training. It can be used to train a committee for few more steps if current performance is unacceptable.

Learners, Weights are the variables in which the resulting boosted classifier is stored. *Final_hyp* is the output of the resulting strong classifier on the training set.

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 semantic is 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 it's absolute magnitude is the "confidence" of the decision.

@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.

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

@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*.

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. vezhnick@gmail.com

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.