A Study of AdaBoost in 3D Gesture Recognition

Jia Sheng
Department of Computer Science
University of Toronto
Toronto, Ontario
jsheng@dgp.toronto.edu

Abstract

In this report, the AdaBoost algorithm is applied to multi-class 3D gesture recognition problem. The performance of AdaBoost is compared across different base classifiers and between different data sets. One method of improving AdaBoost by regularizing the distribution weights is also presented and discussed.

1 Introduction

Boosting is one type of meta learning methods that try to build a ”good” learning algorithm based on a group of ”weak” classifiers, where ”weak” comes from Valiant’s PAC (probably approximately correct) framework [8]. According to Schapire [4], the first provable polynomial-time boosting algorithm was given by Schapire [3] and later improved by Freund [1]. Since then, boosting has been explored by many researchers theoretically and empirically. The most popular algorithm *AdaBoost*, which was introduced by Freund and Schapire in 1995 [2], has successfully solved many practical problems of previous boosting approaches. AdaBoost is also extended to multi-class classification problems and regression problems in [2] and [5].

Many experiments have been carried out using AdaBoost and its variants (refer to [4]), including OCR, text filtering, image retrieval, medical diagnosis, etc. In this report, we apply the idea of boosting on *3D Gesture Recognition*, which is of practical importance in many areas, such like human-computer interaction, computer vision and computer graphics. A good introduction to hand gesture recognition can be found in [6]. The human hand gesture can provide a free and natural alternative to today’s cumbersome interface devices so as to improve the efficiency and effectiveness of human-computer interaction. Almost all researches on gesture recognition are based on computer vision techniques, thus suffering the difficulties of image processing and feature extraction. In our experiment, however, we benefit from the Vicon motion tracking system [9] and can solve the problem of ”finding” the hand relatively more easily and more accurately. Few have been done to try to integrate the boosting approach with the gesture recognition problem and we hope our experiments would arose more interest in this area.

This report is organized as follows: Section 2 talks about the algorithm of AdaBoost and its variant AdaBoost.M1 for multi-class problems. Section 3 describes the process of data capturing and feature extraction of gestures. In Section 4, experiment methods and results are presented. Some thoughts on the AdaBoost algorithm and the experiment results are shown in Section 5. The last section includes the conclusion and future work.

2 The AdaBoost Algorithm

Boosting algorithm works by calling a weak classifier several times, each time providing it with a different distribution over the domain X, and in the end combine the hypotheses from all iterations into one hypothesis. The basic idea of boosting is that it tries to focus more on the "hard" part of the data space by increasing the probability of those data, thus hoping to decrease the mistakes made by the weak classifier. The pseudo-code of AdaBoost is as follows:

AdaBoost
Input: N labelled examples $< (x_1, y_1), ...(x_N, y_N) >$, $x_i \in X$, $y_i \in \{-1, 1\}$
weak learning algorithm **WeakLearn**
integer T specifies the iteration number
Initialize: initial weights $\mathbf{w}$, $w_i = 1/N$, for $i = 1, ..., N$.
$t = 0$, $err^0 = 0$
Do while $t \leq T$ and $err^t < 0.5$
1. Normalize $\mathbf{w^t}$, so that $\sum_{1 \leq i \leq N} w_i^t = 1$.
2. Call **WeakLearn**, providing with the weight $\mathbf{w}^t$; get hypothesis $h^t : X \rightarrow \{-1, 1\}$.
3. Compute $err^t = \sum_{1 \leq i \leq N} w_i^t e_i^t$, where $e_i^t = 1$, if $h^t(x_i) \neq y_i$, and 0 otherwise.
4. Set $\alpha^t = 0.5log[(1 - err^t)/err^t]$.
5. Update the weights to be: $w_i^{t+1} = w_i^t exp(2\alpha^t e_i^t)$.
6. $t = t + 1$.

Output: the hypothesis $h(x_i) = sign[\sum_{j=1}^{t} \alpha^j h^j(x_i)]$

Above is slightly different than in [2] in that we quit the iteration when the weighted error rate of the weak classifier is higher than 0.5. The theoretical lower bounds are given in [5] and [2], which states that the training error is bounded by $exp(-2\sum_{j=1}^{t}(1/2 - err^j)^2)$.

It is worth to note that in step 2, when the $WeakLearn$ is provided with the weight vector $\mathbf{w}^t$, it has two options. Some classifiers can make use of weights directly, such as gradient based algorithms to change the update step size based on the example weights. Other algorithms, however, cannot make direct use of the weights, such as KNN [1]. In the latter case, we can re-sample the training data to generate a new set of training examples that is distributed according to the weights. The examples with larger weight values have more chances of being chosen in the new set, even multiple times, while the less favored might get lost.

AdaBoost can be extended to multiclass classification problems. There are some variants, including AdaBoost.M1, AdaBoost.M2 [2] and AdaBoost.MH [5]. In this report we use AdaBoost.M1 for our multiple gesture recognition. The main difference between AdaBoost.M1 with the above procedure is that the final combination step in AdaBoost.M1 considers only the correctly labelled instances in each iteration and maximize the sum of the weights on these instances, like following:

Output: the hypothesis $h(x_i) = argmax_{y \in Y} \sum_{j=1}^{t} \alpha^j [h^t(x_i) = y_i]$
where $Y = \{1, ..., k\}$ and $[x] = 1$, if x is $true$; $[x] = 0$ otherwise.

[1]Here KNN stands for the *un-weighted K-Nearest Neighbor algorithm*, where all data have the same relevance for classification. We are not talking about its variant *weighted KNN* where the data have different voting weights

3 Features for 3D Gesture Recognition

Vicon [9] is a motion capture system composing of a group of infrared cameras that tracks the reflective markers attached to the subject body. The 2D image seen from each camera is combined to reconstruct the 3D scene in the data station and the 3D position information of the markers can be obtained at a rate of 20-30 fps by the user's application from the workstation and real-time server.

Different to traditional image-based approaches, Vicon possesses the superiority of being able to get the position information accurately: the precision is in millimeter. On the other hand, however, it has no idea of other parts of the hand except for those reflective points. Besides, there might be some noise in the stream due to the missing of markers or some "ghost points". So we decide to use it on the recognition of gestures, which are the dynamic movements of hands within a certain time interval, instead of postures, which are represented by static shapes of hands, e.g. the American Sign Language.

Let p_t^i denote the position of marker i at time t, a gesture G can be represented by the set of all positions of all markers within the period of that gesture. For simplicity, we assume all markers move in parallel, so we have $G = \{p_1, ..., p_T\}$, $p_t = (x_t, y_t, z_t)$ denotes 3D coordinates and T is the duration of the gesture. To reduce the amount of data, we need to extract features out of the 3D point sequences. The features are chosen in a similar way as [7], but extended from 2D to 3D. Totally we use 16-dimension features in our experiment, defined as below:

$$f_1 = (x_3 - x_1)/|p_3 - p_1|$$
$$f_2 = (y_3 - y_1)/|p_3 - p_1|$$
$$f_3 = (z_3 - z_1)/|p_3 - p_1|$$
$$f_4 = |p_T - p_1|$$
$$f_5 = (x_T - x_1)/|p_T - p_1|$$
$$f_6 = (y_T - y_1)/|p_T - p_1|$$
$$f_7 = (z_T - z_1)/|p_T - p_1|$$
$$f_8 = \sqrt{((x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2)}$$
$$f_9 = (x_{max} - x_{min})/f_8$$
$$f_{10} = (y_{max} - y_{min})/f_8$$
$$f_{11} = (z_{max} - z_{min})/f_8$$
$$f_{12} = \sum_{t=2}^{T} |p_t - p_{t-1}|$$
$$\text{let } \theta_t \text{ be the angle between } \overrightarrow{p_{t-1}p_t} \text{ and } \overrightarrow{p_t p_{t+1}}$$
$$f_{13} = \sum_{t=2}^{T-1} \theta_t$$
$$f_{14} = \sum_{t=2}^{T-1} \theta_t^2$$
$$f_{15} = max_{t=1}^{T-1} |p_{t+1} - p_t|$$
$$f_{16} = T$$

The features are defined according to some criterion: each feature should be incrementally computable in constant time per input point, thus allowing effectively handling of arbitrarily slow gestures and permitting online recognition; small changes in the input should result only small changes in the features; each feature should be meaningful so as to be used semantically. Clearly the above features satisfy such requirements.

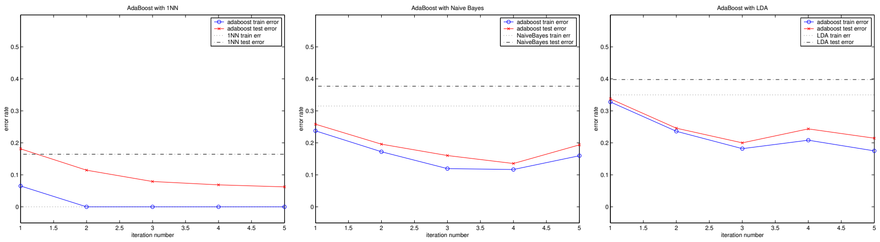

Figure 1: Comparison of AdaBoost.M1 and the base classifiers. From left to right, the base classifiers are 1NN, NaiveBayes and LDA. The x-axis is the iteration number and the y-axis shows the error rate. Black dot line is the training error of the base classifier, black dash-dot line is the testing error of the base classifier; red circle is the training error of AdaBoost.M1, blue cross is the testing error of AdaBoost.M1.

4 Experiments

4.1 Data Sets Preparation

We define 6 gestures in our experiment: ”line”, ”angle”, ”cross”, ”circle”, ”triangle”, ”rectangle”, whose meaning can be interpreted from the name, for example, ”circle” stands for the gesture given by the user when he draws a circle in the air. For each gesture, totally 200 instances are captured. Those data are then split: 120 are used for training and 80 for testing.

In order to study the performance of AdaBoost under noise, we prepare 2 data sets: Set1 is the original data; in Set2, a certain portion of training data is replaced by random noise.

3 weak classifiers are used in the experiment: 1NN, NaiveBayes and LDA. The covariance matrix in LDA is simplified to be diagonal. We use re-sampling to make use of the weights in each iteration of AdaBoost.M1. Since 1NN measures the similarity between data points based on their Euclidean distances, normalization is carried on each dimension to place the training data in $[0, 1]$, then on the testing data.

4.2 Experiments on Adaboost.M1

First, with the original data Set1, we compare the performances of weak classifiers and AdaBoost.M1 using different iterations.

From Figure 1, we can see that for all three cases, AdaBoost.M1 outperforms the base classifiers, both for training and testing sets. We only show 5 iterations here because the weak classifiers using NaiveBayes and LDA will perform worse than random guessing, i.e. $err^t \leq 0.5$, after several iterations. For binary classification, we can just flip the hypothesis if error rate is less than 0.5, but for multi-class problems, the algorithm will quit.

Then we examine the performance of AdaBoost.M1 on noisy data Set2. We replace parts of the training set with random noises and compare the result of base classifier and AdaBoost.M1, as shown in Figure 2. The result is interesting: for 1NN based AdaBoost, the performance drops more slowly than the base classifier, even when the training data is all random noises, the error rate is only 50% ! for NaiveBayes and LDA based AdaBoost, however, the performance breaks down after the noise percentage is over certain threshold. These indicate contradictorily the sensitivity of AdaBoost to noise. We prefer the latter two cases based on NaiveBayes and LDA, and think AdaBoost is more sensitive to noise, because in the nearest neighbor algorithm, the data is so dense that the chance of meeting

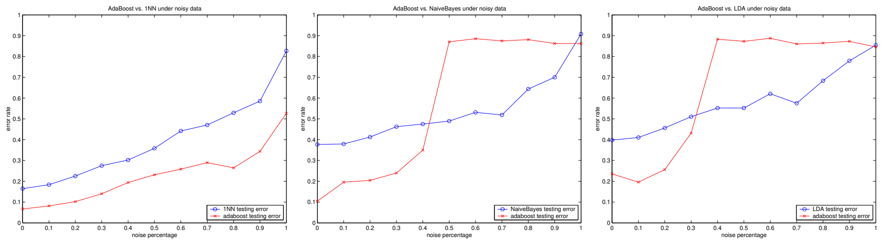

Figure 2: Comparison of AdaBoost.M1 and the base classifiers under noisy data. The x-axis is the percentage of random noise in the training data, ranging from 0 to 1; the y-axis the error rate of testing error. From left to right, the base classifiers are 1NN, NaiveBayes and LDA. Red line is the testing error of AdaBoost.M1, while blue line is the testing error of the base classifier.

the data from the same class as the test data point in its neighborhood is high even when the data is totally noise. However, it is open to further investigation.

5 Revising Adaboost.M1

The idea of AdaBoost is that it tries to put more emphasis on the "harder" data by adjusting the distribution in every iteration. Let's first take a look at the change of weights with the iteration number.

We can see in Figure 3 that AdaBoost.M1 will correctly classify a large percentage of data in most iterations, but struggle for a better estimates on a small group of instances. Moreover, we find that the weight values for the hard instances grow as the iteration number increases, which means those data get more chances to be selected in the re-sampling process, and means that AdaBoost.M1 is "trapped" in those data. It also fits our conjecture that AdaBoost.M1 is sensitive to noises. So we propose our first method to modify AdaBoost.M1:

Revise 1: set the instances with the highest weights to be 0 after c iterations.

By setting the weights zero, we actually discard those hard examples. Alternatively, we may set those weights to the mean value instead of zero, and we have

Revise 2: set the instances with the highest weights to have the mean weight value after c iterations.

We hope these methods would give AdaBoost more chances of jumping out of those instances. The comparison of AdaBoost.M1, AdaBoost.M1 Revise(1) and AdaBoost.M1 Revise(2) is shown in Figure 4 ($c = 5$).

The result show that Revise 1 and Revise 2 do not show significant improvements than AdaBoost.M1, if any. But considering the fact that we actually discard the hard data by setting their weights zero in Revise 1, such method has potential of reducing the amount of data. It is worthy of further study in the future.

6 Conclusion and Future Work

So far, we have reported our experiments of AdaBoost algorithm on the application of 3D gesture recognition. For all the three base classifiers we have tried, AdaBoost.M1 has shown significant improvement in performance. On the other hand, however, AdaBoost

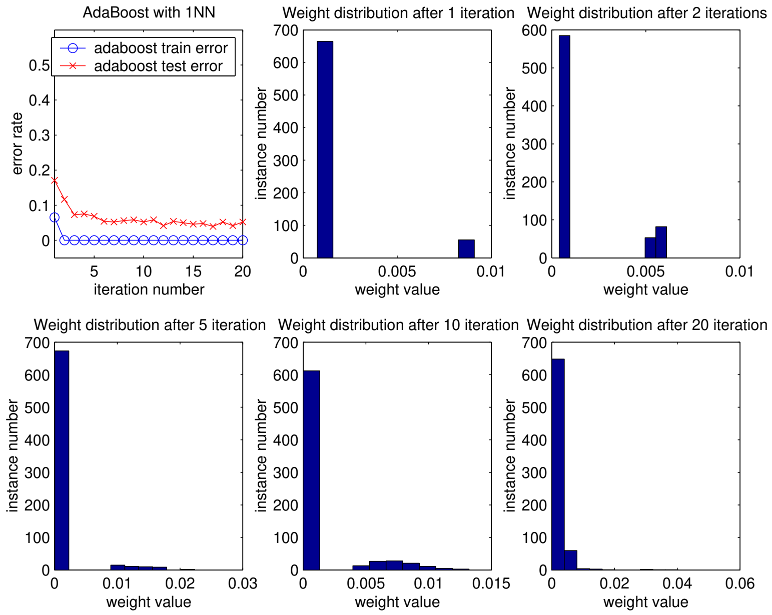

Figure 3: The training and testing error of 1NN-based AdaBoost.M1 for 20 iterations and the weight distribution after 1, 2, 5, 10, 20 iterations, respectively from left to right.

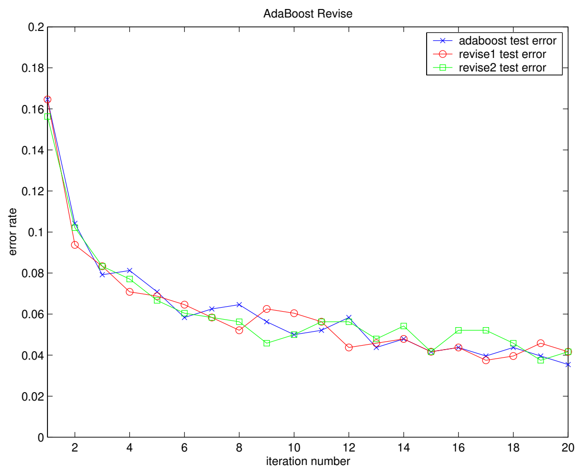

Figure 4: The comparison of AdaBoost.M1 and two revising methods. The blue line is the testing error of 1NN-based AdaBoost.M1 for 20 iterations. The red line is for Revise 1 and the cyan line is for Revise 2.

seems more sensitive to noisy data. We further propose some of our thoughts on revising the AdaBoost algorithm, which impose some kind of regularization on the distribution weights during the iteration in order to prevent the possible detriment caused by hard-to-classify examples. Although the results don't show significant improvement, we think it is important to impose regularization to AdaBoost algorithms.

As to the specific task of 3D gesture recognition, one of the key problems is how to distinguish intentional and unintentional movements. The data in our experiments is captured by explicitly triggering the starting and ending point of each gesture. In real life, however, it is obtrusive or even impossible to ask the subject to send a signal before and after each gesture. Basically, it involves the problem of binary classification based the movement within a certain time interval. Considering that the speed of movements may vary a lot, such classification would pose a challenge on traditional approaches, while AdaBoost might give different results.

Another potential usage of AdaBoost in 3D gesture recognition is the selection of features. In our experiments, we propose the 16 dimension feature, which come almost empirically from our experience. Other researchers may suggest other definition of features. How to judge the quality of different feature sets, combine these sets, and finally distill an optimal subset is an interesting and important problem. AdaBoost, again, might be a good option.

Implementation

The algorithm is written with Matlab and source code and data can be downloaded from http://www.dgp.toronto.edu/~jsheng/ml

References

[1] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256-285, 1995.

[2] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119-139, Auguest 1997.

[3] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197-227, 1990.

[4] R.E. Schapire. The boosting approach to machine learning: an overview. *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[5] R. Schapire and Y. Singer. Improved boosting algorithms using confidentce-rated predictions. *Machine Learning*, 39(3):297-336, December 1999.

[6] V. Pavlovic, R. Sharma, T.S. Huang. Visual interpretation to hand gestures for human computer interaction: a review, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997

[7] D. Rubine. Specifying gestures by examples. *SIGGRAPH '91 Proceedings*, 25(4):329–337, July 1991.

[8] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134-1142, November 1984.

[9] http://www.vicon.com