Reverse AdaBoost Algorithm to do Segmental Linear Regression

Xunxian Wang, David Brown
Intelligent systems and fault diagnostic group, University of Portsmouth, UK
Email: xunxian.wang@port.ac.uk

Abstract: An algorithm is proposed to separate different linear part from each other. It can be used in the linear dynamic systems with small non-linear features such as dead zone and saturation, which is very popular in DC motor systems and vector converter controlled AC motor systems. With the proposed algorithm, the working linear part can be modelled without the influence of the nonlinear features.

Keywords: Linear regression, Statistical learning, Boosting, Clustering, Function approximation; System modelling; Data mining

1. Introduction

As an easy and useful tool in data analysis and data mining, linear regression occupies an important position in data analysis. This paper shows a research result on multi-segmental linear regression by using boosting method.

Motor systems, both of DC and AC motors, have a very important position in the industry field. Due to its linear feature, DC motors have been used a lot in high accurate speed control systems. In this kind of systems, precisely modelling the motor system becomes a key technique, which can determine the performance of the whole system. The development of the microprocessor technique and power electronic technique make the availability of the vector converter. Based on the converter, the combined system of vector controller and AC motor becomes a linear system. So modelling linear system precisely is still a useful technique.

As an important boosting algorithm, AdaBoost has obtained lot of attention in these days. Not only this algorithm can be used in combining thumb rules into a better one, but also it can be used in many different areas as shown in [1]. In system modelling, by given different samples different weights, the boosting algorithm can adjust the regression line goes through the middle of the data.

In this paper, reversed AdaBoost algorithm is used in multi-segmentation linear system modelling. While normal AdaBoost algorithm give the data points near to the current regression line less weight, far points more weight, the reverse algorithm give near more weight and far less and by this the regression line can be controlled to go through only one part of the multi-segmented system.

The whole paper is consisted of five parts. In section 2 the basic algorithm of the methods is shown; while in section 3 the result is given and the section 4 give some improvement methods for the algorithm. A conclusion is given in section 5.

2. Description of boosting linear regression

2.1 The basic principle of boosting

Boosting is a method that can be used to combine a number of available rules into one single rule. One of the popular boosting algorithms is AdaBoost[1], which can be described in pseudo-code as follows

Input:

A sequence of N labelled examples $< (x_1, y_1),...,(x_N, y_N) >$

Distribution D over the N examples.

Weak learning algorithm **Weaklearn**[1]

Integer T specifying number of iterations

Initialise:

The weight vector $w_i^1 = D(i)$ for $i = 1,..., N$

Do for $t = 1,...,T$

Set

1. $\vec{p}^t = \dfrac{\vec{w}^t}{\sum_{i=1}^{N} w_i^t}$
2. Call Weaklearn, and provide it with the distribution $\vec{p}^t$; get back a hypothesis $h_t : X \to [0,1]$.

3. Calculate the error of $h_t : \varepsilon_t = \sum_{i=1}^{N} p_i^t \left| h_t(x_i) - y_i \right|$
4. Set $\beta_t = \dfrac{\varepsilon_t}{1-\varepsilon_t}$
5. Set the new weight vector to be

$$w_i^{t+1} = w_i^t \beta_t^{1-\left|h_t(x_i)-y_i\right|} \quad (1)$$

The AdaBoost algorithm above is used to boost the performance of weak learning algorithms. The advantage of this method is that the accuracy of the final hypothesis produced depends on the accuracy of all the hypotheses returned by WeakLearn[1] and so is able to fully exploit the power of the weak learning algorithm. The goal of the algorithm is to find a final hypothesis with low error relative to a given distribution D over the training examples. Initially this D is appointed as $D(i) = 1/N$. The algorithm maintains a set of weights w^t over the training examples. On iteration t a distribution p^t is computed by normalising these weights. This distribution is fed to the weak learner WeakLearn which generates a hypothesis h_t that has small error with respect to the distribution. Using the new hypothesis h_t, the boosting algorithm generates the next weight vector w^{t+1}, and process repeats.
In above algorithm, the weight related to the big loss example will increase while the low loss example will decrease.
By using the above in an approximation problem, the above algorithm can force the approximation function to go through the middle of the training data [1].

2.2 The algorithm of linear regression with boosting

In linear system, the system input and output relationship can be represented by

$$f(x) = \vec{a}'\vec{x} + b \quad (2)$$

Where $\vec{a} = (a_1, a_2, \cdots, a_n)^T$, $\vec{x} = (x_1, x_2, \cdots, x_n)^T$. To do the linear regression by using least square method as follows

$$E = \sum_{i=1}^{N} (f(\vec{x}^i) - y^i)^2 \quad (3)$$

Where $(\vec{x}^i, y^i), i = 1, \cdots, N$ is the training data.

The following formula can be obtained

$$\begin{pmatrix} \vec{a} \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} \vec{x}^i (\vec{x}^i)' & \sum_{i=1}^{N} \vec{x}^i \\ \sum_{i=1}^{N} (\vec{x}^i)' & 1 \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^{N} \vec{x}^i y_i \\ \sum_{i=1}^{N} y_i \end{pmatrix} \quad (4)$$

However, many linear systems especially linear control system aren't ideal linear. Normally there are three linear parts in the system; they are working part, saturation part and head zero. To represent the whole system feature, all the three parts needs to be modelled which needed three linear sub-systems.
Linear regression method shown above can't give a satisfied answer to the above question. By using boosting algorithm shown in 2.1, a good result can be obtained.
First, the cost function (3) is modified as

$$E = \sum_{i=1}^{N} \delta^i (f(\vec{x}^i) - y^i)^2 = \sum_{i=1}^{N} \delta^i [(\vec{a}'\vec{x}^i + b) - y^i]^2 \quad (5)$$

Where $\delta^i, i = 1, \cdots, N$ with $\sum_{i=1}^{N} \delta^i = 1$ will be determined by the boosting method.
By using

$$\begin{cases} \dfrac{\partial E}{\partial a} = 0 \\ \dfrac{\partial E}{\partial b} = 0 \end{cases} \quad (6)$$

The following results can be obtained

$$\begin{pmatrix} \vec{a} \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} \delta^i \vec{x}^i (\vec{x}^i)' & \sum_{i=1}^{N} \delta^i \vec{x}^i \\ \sum_{i=1}^{N} \delta^i (\vec{x}^i)' & \sum_{i=1}^{N} \delta^i \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^{N} \delta^i \vec{x}^i y_i \\ \sum_{i=1}^{N} \delta^i y_i \end{pmatrix} \quad (7)$$

2.3 The determination of the weight by using boosting method

For a set of given training data, which consists of two or more different linear parts, ordinary linear regression will give a linear function representing neither of the linear functions as shown in Fig.1. Anyway, if the points which are nearer to the available regression line are given more weights than the far data, the next straight line generated by the new weighted training data will nearer to the first part of the straight segments in Fig. 1. The weights given to each data are determined by the modified boosting algorithm, which is shown below.

- Step 1: Preparation

1. Set $\delta_0^1 = \delta_0^2 = \cdots = \delta_0^N = \frac{1}{N}$.
2. k = 0

- Step 2: Linear regression
  1. Using (7) to calculate the parameter $\vec{a}, b$ noted as $\vec{a}, b$.
  2. Obtain the linear function $y(x) = \vec{a}\vec{x} + b$
- Step 3: Boosting
  1. Calculate the distance between each data and the function $y(x) = \vec{a}\vec{x} + b$.
     That is
     $$dist(i) = \left| y(x^i) - \hat{y}^i \right|, i = 1, \cdots, N$$
  2. Normalise the loss
     $$ndist(i) = \frac{dist(i)}{\sum_{i=1}^{N} dist(i)}$$
  3. Use the following formula to calculate β_t
     $$\varepsilon_k = \sum_{i=1}^{N} \delta_k^i ndist(i)$$
     $$\beta_k = \frac{\varepsilon_k}{(1 - \varepsilon_k)}$$
  4. Update the weight vector
     $$\delta_{k+1}^i = \delta_k^i \beta_k^{ndist(i)}, i = 1, \cdots, N \quad (8)$$
  5. Normalise the weight vector
     $$\delta_{k+1}^i = \frac{\delta_{k+1}^i}{\sum_{i=1}^{N} \delta_{k+1}^i}, i = 1, \cdots, N \quad (9)$$
- Step 4: Repeat from Step 2 $\max_{i \in [1,N]} \left| \delta_{k+1}^i - \delta_k^i \right| < \varepsilon$, where ε is a small value.
- Stop

There are one modifications in the above algorithm compared to Adaboost. First, In AdaBoost, weights are updated by $w_{k+1}^i = w_k^i \beta_k^{1-ndist(i)}$ but here we use $\delta_{k+1}^i = \delta_k^i \beta_k^{ndist(j)}$. (in Adaboost, $0 < \beta_k < 1$) This means that the nearer data points will get more weight on the next round and the far data points will get less weight in the next round.

3 Simulation results

3.1 Simulation results in scalar system

The training data is generated by

$$y(x) = \begin{cases} 0.2x + \omega & x \le 20 \\ 0.3x + \omega & 20 < x \le 40 \\ 0.1x + \omega & x > 40 \end{cases} \quad (10)$$

Where ω is Gaussian noise with 0 mean and 0.1 variance. Fig.1 shows the result obtained by the ordinary linear regression. The regression line can't represent the real data well. Fig.2 is the result by using the modified boosting algorithm and the regression line can be used to represent the first part of the training data.

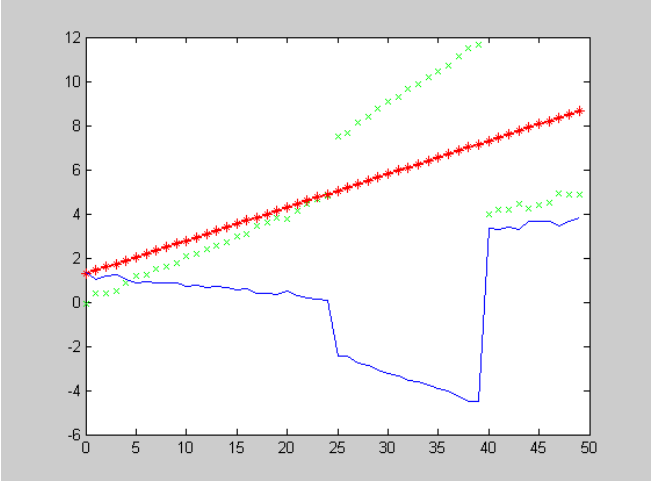

Fig.1. The regression result obtained by ordinary linear regression

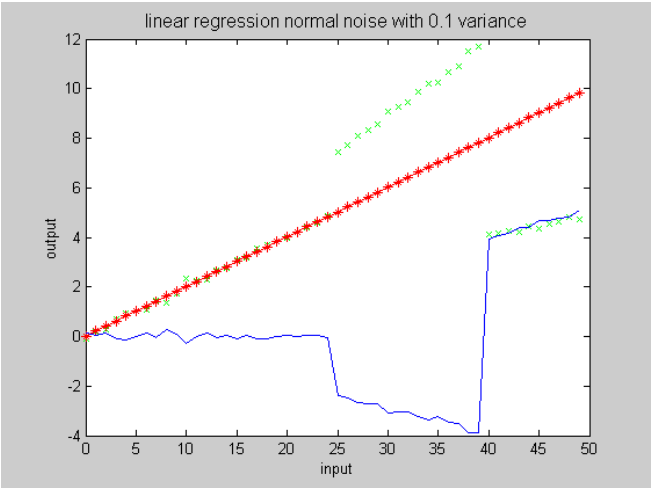

Fig.2. The regression result obtained by boosting linear regression

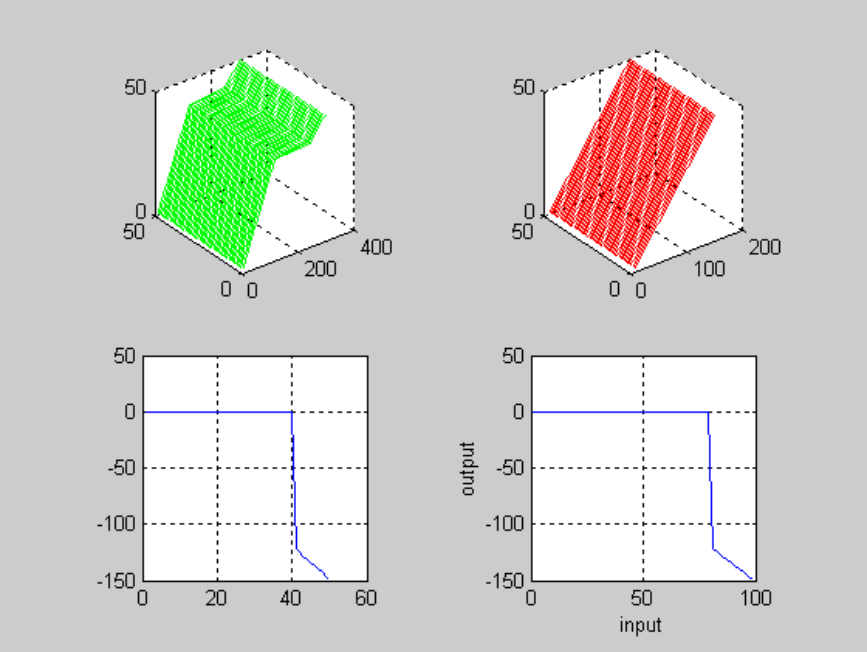

Fig.3. The regression result obtained by boosting linear regression

3.2 Simulation results in vector system

A three dimensional linear system training data is generated by

$$y(x_1, x_2) = \begin{cases} 10 + 0.5x_1 + 1.2x_2 + \omega \\ \quad x_1 \le 40, \quad x_2 \le 80 \\ 10 + 1.5x_1 + 2.2x_2 + \omega \\ \quad x_1 > 40, \quad x_2 > 80 \end{cases} \tag{11}$$

ω is the same noise as above.

The simulation result is shown in Fig. 3, in which the left-up figure generated by the function (11), right-up is the regression result to the first part of the data. And the lower figures are the error indication from axis x_1 and x_1 respectively.

4. Some further consideration in the boosting linear regression

4.1 The phenomena of local focusing

Even if the boosting linear regression is used, sometimes a satisfied result can't be produced. An example can be given by the following function.

$$y(x) = \begin{cases} 7 + 0.2x + \omega & x \le 20 \\ 1 + 0.3x + \omega & 20 < x \le 40 \\ 4 + 0.1x + \omega & x > 40 \end{cases} \tag{12}$$

The simulation result in left figure in Fig.4 shows a very bad final regression line. The reason ca be stated as follows: because of all the initial value of weights in the modified AdaBoost is given the same, so the first regression line is as the same as the result of ordinary regression; further, because of the regression line goes through the points belong to different parts of the segments and the distances between the regression line and these points are small and so the weights of these points will be strengthened by the boosting algorithm, and this will fix the regression line going through these points.

There are two methods to improve this situation. One of them called label analysis method is to monitor this situation and to modify the weights to allow the line go out of this position. Another method is to given different initial value of the weight

.

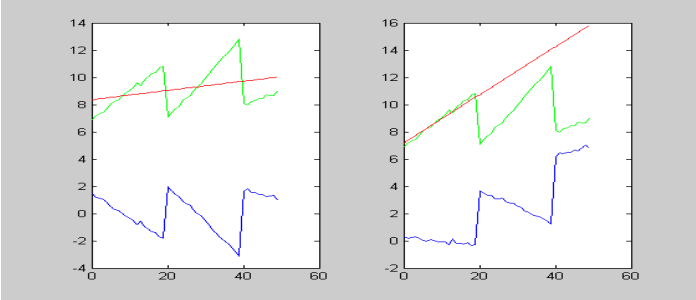

Fig.4. A bad regression line example

4.2 Label analysis method

The phenomenon shown in Fig. 4 can be checked out by analysing the weight distribution of the label related to the training data. The label here means i in (x_i, y_i). The weights related to the data points near to the regression will have big value, while others will have small one. If a big gap is found between the labels of the points with big values, an decision can be made that the regression line goes through different straight parts shown in the left figure in Fig.4. If the above phenomenon is found, the training data with big weight will be segmented and the weight of one of them with the longest length will be remained and the others will be clear to zero. The result by using this methods shown in right of Fig.4 give an positive example of the algorithm.

4.3 Special Initial value method.

For a special training data, if some pre-knowledge can be obtained before the system modelling, different data can be appointed different weight values. In details, if we know the segmentation number and a rough position of each segment part, we can appoint different initial values to different part as desired.

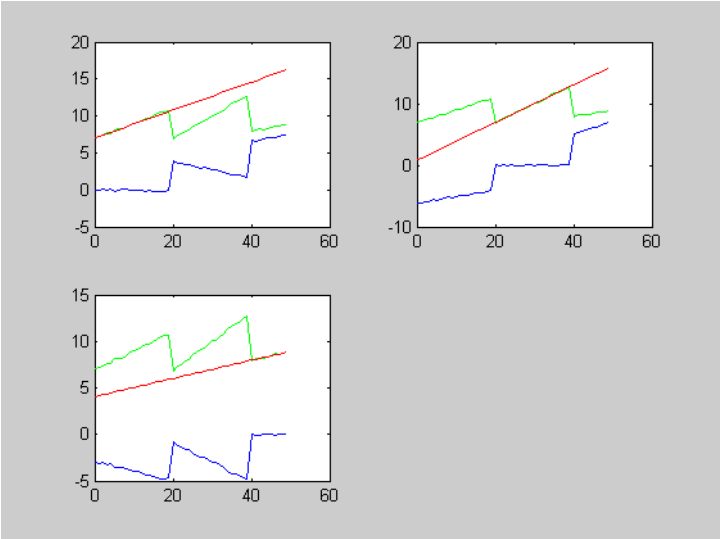

Fig.5. Multiple segmentation simulation example

Here its an example, if 50 data points are given and three line segments is in the training data, then we can set the initial value three different times as follows:
First the following weights are being used.

$$\delta_i = \begin{cases} 1/10, & i \le 10 \\ 0, & otherwise \end{cases}$$

After the first boosting regression finished, the second initial values shown as the following are being used.

$$\delta_i = \begin{cases} 1/10, & 20 < i \le 30 \\ 0, & otherwise \end{cases}$$

And the third of the weights are.

$$\delta_i = \begin{cases} 1/10, & 40 < i \le 50 \\ 0, & otherwise \end{cases}$$

The simulation results are shown in Fig. 5.

4.4 Termination condition of the boosting regression

The linear regression method applied to the weighted data will be used many times until a satisfied result is produced. If the maximum value of the weight, say 0.1 here is produced, and the above gap between the big weights hasn't produced. Then the algorithm will terminate its current segment linear regression. Normally, the maximum weight value will be determine by the number of the training data.

4.5 Numerical calculation of the solution

many times, the solution calculated by using function (7) will not get a satisfied answer due to the limit data length of numerical computation. In this research, the singular value decomposition method is used and is stated below.
For a function given by (13)

$$Ax = b \qquad (13)$$

Assume $A = U\Sigma V^T$, where U, V is orthogonal matrix and $\Sigma = diag(\lambda_i)$ is diagonal matrix , by using singular value decomposition, we have $U\Sigma V^T x = b$, then

$$x = V\Sigma^{-1}U^T b \qquad (14)$$

When $\lambda_i = 0$, we use $1/\lambda_i = 0$ in his reverse from.

5 Conclusion

The algorithm shown in this paper can be implemented by using neural network too. In a typical multi-linear system, there will be some slight non-linear feature between the linear segmentation. By using a linear regression method to model the linear parts, the linear parts can be modelling by a very simple neural network. This will reduce the model complexity very much.

References

1. Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Proceedings of the Second European Conference on Computational Learning Theory, March, 1995.
2. Numerical recipes in C: The art of scientific computing; William H. press, 2ed edition.

Authors:

Xunxian Wang: Ph.D, Research Fellow.
David. Brown: Ph.D Senior Lecture

All are in intelligent system and fault diagnostic group, University of Portsmouth, UK

Contact author: Xunxian Wang. Obtained his Ph.D in Tsinghua University, Beijing, China from Jul. 1999. He had been a post-doc researcher in the above University from Aug.1999 to Aug.2001. From Sep. 2001 he has been a research associate in University of Portsmouth and University of Leeds and currently a research fellow in University of Portsmouth UK. His email address is xunxian.wang@port.ac.uk