Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Merging weighted SVMs for parallel incremental learning

Lei Zhu [a], Kazushi Ikeda [b], Shaoning Pang [a,*], Tao Ban [c], Abdolhossein Sarrafzadeh [a]

[a] Unitec Institute of Technology, New Zealand
[b] NARA Institute of Science and Technology, Japan
[c] National Institute of Information and Communications Technology, Japan

ARTICLE INFO

Article history:
Received 4 September 2017
Received in revised form 21 November 2017
Accepted 15 January 2018
Available online 2 February 2018

Keywords:
Incremental learning
Parallel learning
Parallel incremental learning
Knowledge merging
Extreme support vector machine (ESVM)
Weighted ESVM (wESVM)

ABSTRACT

Parallel incremental learning is an effective approach for rapidly processing large scale data streams, where parallel and incremental learning are often treated as two separate problems and solved one after another. Incremental learning can be implemented by merging knowledge from incoming data and parallel learning can be performed by merging knowledge from simultaneous learners. We propose to simultaneously solve the two learning problems with a single process of knowledge merging, and we propose parallel incremental wESVM (weighted Extreme Support Vector Machine) to do so. Here, wESVM is reformulated such that knowledge from subsets of training data can be merged via simple matrix addition. As such, the proposed algorithm is able to conduct parallel incremental learning by merging knowledge over data slices arriving at each incremental stage. Both theoretical and experimental studies show the equivalence of the proposed algorithm to batch wESVM in terms of learning effectiveness. In particular, the algorithm demonstrates desired scalability and clear speed advantages to batch retraining.

1. Introduction

Incremental and parallel are two appeal capabilities for machine learning algorithms to accommodate data from real world applications. Incremental learning (Joshi & Kulkarni, 2012) addresses streaming data by constructing a learning model that is updated continuously in response to newly arrived samples. Hou and Zhou (2016) introduce interesting variants of incremental learning, in which features, instead of data, are added and removed during the learning procedure. To solve the computational problems posed by large data scale, parallel learning (Upadhyaya, 2013) distributes the computational efforts among multiple nodes within a cloud or cluster to speed up the calculation. In the work of Zhao, Liang, and Yang (2012), SVM training is parallelized by an ensemble technique in which multiple SVMs are trained in parallel, given different subsets of training data, and the classification is made by ensembling these SVMs by majority vote. Caruana, Li, and Qi (2011) parallelize the classic Sequential Minimal Optimization (SMO) algorithm, in which each computational nodes only works on a subset of the complete optimization. With the rise of BigData, data become both large scale and streaming. Interest on developing algorithms that are both parallel and incremental is increasing, and this interest leads to the parallel incremental algorithm proposed in this paper.

To developing a parallel incremental algorithm, three straightforward route maps are considered: (a) parallelize an existing incremental algorithm, for example Tsianos and Rabbat (2016) recently proposed a gossip-based approach that parallelizes online prediction and stochastic optimization; (b) renovate an existing parallel algorithm to be incremental, but few research study can be found in this category; and (c) derive a parallel incremental algorithm from scratch. In this category, existing works normally treat incremental and parallel learning as two separate problems. One either first parallelizes learning then designs the incremental rule (e.g., Böse, Andrzejak, and Högqvist (2010), Yue, Fang, Wang, Li, and Liu (2015), Zhao et al. (2016)), or the other way around (e.g., Chen and Huo (2016), Doan, Do, and Poulet (2013), Xu and Yun (2015), Yoo and Boulware (2014)).

Alternatively, we achieve parallel and incremental learning by proposing a novel knowledge merging approach, and apply the approach to weighted Extreme Support Vector Machine (wESVM) for parallel incremental learning. This approach enables merging of wESVMs on subsets of data into one model whose learning result equals that from the whole dataset. In doing this, we derive a new formula of wESVM in which knowledge is represented as a set of class-wised matrices, and we prove that the merging of wESVMs can be performed through simple matrix addition. Through knowledge merging, parallel learning can be implemented by letting multiple nodes learn simultaneously from data slices, then combine the knowledge obtained. Additionally, incremental learning can be carried out by adding up knowledge acquired at different

* Corresponding author.
E-mail address: ppang@unitec.ac.nz (S. Pang).

https://doi.org/10.1016/j.neunet.2018.01.001
0893-6080/© 2018 Elsevier Ltd. All rights reserved.

incremental stages. Thus, wESVM is transformed without information lost for parallel incremental learning.

The rest of this paper is organized as follows. Section 2 introduces preliminary work including MapReduce and the batch algorithms that our work is based on. In Section 3, we implement the idea of knowledge merging and develop the proposed parallel incremental wESVM. The proposed algorithm is evaluated in Section 4, and we conclude our work in Section 5.

2. Preliminary

2.1. LPSVM

Given a training set $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, having instance matrix $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n\end{bmatrix}' \in \mathbb{R}^{n\times d}$ and label vector $\boldsymbol{Y} = \begin{bmatrix}y_1 & y_2 & \cdots & y_n\end{bmatrix}' \in \{+1, -1\}^{n\times 1}$. A classic Support Vector Machine (SVM) (Vapnik, 1995) solves the following optimization

$$\begin{aligned} \min \quad & C\sum_{i=1}^{n}\xi_i + \frac{\|\boldsymbol{w}\|^2}{2} \\ \text{s.t.} \quad & y_i(\boldsymbol{x}_i'\boldsymbol{w} + b) + \xi_i \geq 1 \\ & \xi_i \geq 0 \quad \forall i \in \{1, \dots, n\}, \end{aligned} \tag{1}$$

in order to learn a separation plane

$$\boldsymbol{x}'\boldsymbol{w} + b = 0, \tag{2}$$

which is located in the midway point of two bounding planes

$$\begin{aligned} \boldsymbol{x}'\boldsymbol{w} + b = +1 \\ \boldsymbol{x}'\boldsymbol{w} + b = -1. \end{aligned} \tag{3}$$

Training samples from two classes are bounded by (3) with some non-negative slacks ξ_i

$$\begin{aligned} \boldsymbol{x}_i'\boldsymbol{w} + b + \xi_i \geq +1 \quad \text{for} \quad y_i = +1 \\ \boldsymbol{x}_i'\boldsymbol{w} + b - \xi_i \leq -1 \quad \text{for} \quad y_i = -1. \end{aligned} \tag{4}$$

$\frac{2}{\|\boldsymbol{w}\|}$ is the distance between bounding planes in (3), also known as margin in literature (Yamasaki & Ikeda, 2005). In optimization (1), the margin is maximized by minimizing $\frac{\|\boldsymbol{w}\|^2}{2}$, the total slack is minimized by minimizing $\sum_{i=1}^{n}\xi_i$, and the importance of margin maximization and total slacks minimization is balanced by parameter C (Cauwenberghs & Poggio, 2000; Karasuyama & Takeuchi, 2010).

Unlike classic SVM (Vapnik, 1995), Linear Proximal SVM (LPSVM) simplifies the above binary classification as an regularized least square problem, thus the training of LPSVM becomes more efficient (Fung & Mangasarian, 2001) than the classic SVM. Specifically, LPSVM solves the following optimization

$$\begin{aligned} \min \quad & \frac{1}{2}(\|\boldsymbol{w}\|^2 + b^2) + \frac{C}{2}\|\boldsymbol{\xi}\|^2 \\ \text{s.t.} \quad & \boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{e}b) + \boldsymbol{\xi} = \boldsymbol{e}, \end{aligned} \tag{5}$$

where $\boldsymbol{\xi}$ is a $n \times 1$ slack vector, $n \times n$ diagonal matrix $\boldsymbol{D} = diag(\boldsymbol{Y})$ represents class labels and $\boldsymbol{e}$ is a $n \times 1$ vector of ones. Through solving (5), LPSVM obtains a separating plane

$$\boldsymbol{x}'\boldsymbol{w} - b = 0 \tag{6}$$

which lies in the middle of two proximal planes

$$\begin{aligned} \boldsymbol{x}'\boldsymbol{w} - b = +1 \\ \boldsymbol{x}'\boldsymbol{w} - b = -1. \end{aligned} \tag{7}$$

As compared to the classic SVM optimization (1) which has inequality constraint, LPSVM applies an equality constraint in (5). As a result, the planes (7) no longer bound training samples, but become the proximal planes with data points of each class clustered around. In LPSVM optimization (5), the margin between proximal planes is maximized by minimizing term $(\|\boldsymbol{w}\|^2 + b^2)$, the total slack is minimized by minimizing $\|\boldsymbol{\xi}\|^2$ and the importance of these two objectives are balanced by parameter C.

The solution of LPSVM can be given explicitly as

$$\begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix} = \left[\frac{\boldsymbol{I}}{C} + \begin{bmatrix}\boldsymbol{X}' \\ -\boldsymbol{e}'\end{bmatrix}\begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}\right]^{-1}\begin{bmatrix}\boldsymbol{X}'\boldsymbol{D}\boldsymbol{e} \\ -\boldsymbol{e}'\boldsymbol{D}\boldsymbol{e}\end{bmatrix}. \tag{8}$$

For the derivation of (8), please refer to the work of Fung and Mangasarian (2001). Let $\boldsymbol{F} = \begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}$, we have (8) simplified as

$$\begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix} = (\frac{\boldsymbol{I}}{C} + \boldsymbol{F}'\boldsymbol{F})^{-1}\boldsymbol{F}'\boldsymbol{D}\boldsymbol{e}, \tag{9}$$

where $\boldsymbol{I}$ is a $(d+1) \times (d+1)$ identity matrix.

The steps of batch LPSVM training is summarized in Algorithm 1.

Algorithm 1 LPSVM

```
Input: $\boldsymbol{X} \in \mathbb{R}^{n\times d}, \boldsymbol{Y} \in \{+1,-1\}^{n\times 1}, C \in \mathbb{R}^+$.
Output: $\begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix}$.
1: Generate $\boldsymbol{e} \in \mathbb{R}^{n\times 1}$;
2: Generate $\boldsymbol{F}$, as $\boldsymbol{F} = \begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}$;
3: Transform $\boldsymbol{Y}$ into $\boldsymbol{D}$, as $\boldsymbol{D} = diag(\boldsymbol{Y})$;
4: Generate $\boldsymbol{I} \in \mathbb{R}^{(d+1)\times(d+1)}$;
5: Compute $\begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix}$ as (9).
```

Once $\begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix}$ is computed, the classification decision is made by

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{x}'\boldsymbol{w} - b \\ &= \begin{bmatrix}\boldsymbol{x}' & -1\end{bmatrix}\begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix} \begin{cases} > 0 & \text{then} \quad y = +1 \\ < 0 & \text{then} \quad y = -1. \end{cases} \end{aligned} \tag{10}$$

2.2. ESVM

Extreme SVM (ESVM) (Liu, He, & Shi, 2008) is essentially an LPSVM in Extreme Learning Machine (ELM) (Huang, Zhu, & Siew, 2004, 2006) feature space. For nonlinear classification, ESVM introduces a nonlinear mapping function $\Phi(\boldsymbol{x})$, through which d dimensional input samples are mapped explicitly into a $\tilde{d}$ dimensional feature space, so that an LPSVM linear separation can be conducted in feature space to achieve nonlinear classification of input space.

The mapping $\Phi(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ is performed as

$$\begin{aligned} \Phi(\boldsymbol{x}) &= G(\boldsymbol{W}\boldsymbol{x}^1) \\ &= (g(\sum_{i=1}^{d}\boldsymbol{W}_{1i}\boldsymbol{x}_i + \boldsymbol{W}_{1(d+1)}), \dots, \\ &\quad g(\sum_{i=1}^{d}\boldsymbol{W}_{\tilde{d}i}\boldsymbol{x}_i + \boldsymbol{W}_{\tilde{d}(d+1)}))'. \end{aligned} \tag{11}$$

where $\boldsymbol{x} \in \mathbb{R}^{d\times 1}$ is the sample of input space, $\boldsymbol{x}^1 = [\boldsymbol{x}', 1]'$, $\boldsymbol{W} \in \mathbb{R}^{\tilde{d}\times(d+1)}$ is a weighting matrix whose elements are randomly generated and $\Phi(\boldsymbol{x})$ is the projection of $\boldsymbol{x}$ in $\mathbb{R}^{\tilde{d}}$. Here, $G(\cdot)$ stands for a mapping function that projects each element z_{ij} of input matrix $\boldsymbol{Z}$ into corresponding $g(z_{ij})$ of output matrix $G(\boldsymbol{Z})$, where $g(\cdot)$ is an activation function specified by user such as sigmoidal function. From the view point of ELM, $\Phi(\boldsymbol{x})$ can be seen as the output of $\boldsymbol{x}$