The Journal of Systems and Software 000 (2016) 1–10

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Data stream classification using random feature functions and novel method combinations

Diego Marrón[a,*], Jesse Read[b,c], Albert Bifet[c], Nacho Navarro[a]

[a] Department of Computer Architecture, Universitat Politecnica de Catalunya and with the Department of Computer Science, Barcelona Supercomputing Center, Spain
[b] Aalto University and HIIT, Finland
[c] LTCI, CNRS, Télécom Paris Tech

ARTICLE INFO

Article history:
Received 30 September 2015
Revised 29 March 2016
Accepted 4 June 2016
Available online xxx

Keywords:
Data stream mining
Big data
Classification
GPUs

ABSTRACT

Big Data streams are being generated in a faster, bigger, and more commonplace. In this scenario, Hoeffding Trees are an established method for classification. Several extensions exist, including high-performing ensemble setups such as online and leveraging bagging. Also, k-nearest neighbors is a popular choice, with most extensions dealing with the inherent performance limitations over a potentially-infinite stream.

At the same time, gradient descent methods are becoming increasingly popular, owing in part to the successes of deep learning. Although deep neural networks can learn incrementally, they have so far proved too sensitive to hyper-parameter options and initial conditions to be considered an effective 'off-the-shelf' data-streams solution.

In this work, we look at combinations of Hoeffding-trees, nearest neighbor, and gradient descent methods with a streaming preprocessing approach in the form of a random feature functions filter for additional predictive power.

We further extend the investigation to implementing methods on GPUs, which we test on some large real-world datasets, and show the benefits of using GPUs for data-stream learning due to their high scalability.

Our empirical evaluation yields positive results for the novel approaches that we experiment with, highlighting important issues, and shed light on promising future directions in approaches to data-stream classification.

1. Introduction

There is a trend towards working with big and dynamic data sources. This tendency is clear both in real world applications and the academic literature. Many modern data sources are not only dynamic but often generated at high speed and must be classified in real time. Such contexts can be found in sensor applications (e.g., tracking and activity monitoring), demand prediction (e.g., of electricity), manufacturing processes, robotics, email, news feeds, and social networks. Real-time analysis of data streams is becoming a key area of data mining research as the number of applications in this area grows.

The requirements for a classifier in a data stream are to

- Be able to make a classification at any time
- Deal with a potentially infinite number of examples
- Access each example in the stream just once

These requirements can in fact be met by variety of learning schemes, including even batch learners (e.g., Qu et al., 2009), where batches are constantly gathered over time, and newer models replace older ones as memory fills up. Nevertheless, incremental methods remain strongly preferred in the data streams literature, and particularly the Hoeffding tree (HT) and its variations (Domingos and Hulten, 2000; Bifet et al., 2010b), k-nearest neighbors (kNN) (Shaker and Hüllermeier, 2012). Support for these options is given by large-scale empirical comparisons (Read et al., 2012), where it is also found that methods such as naive Bayes and stochastic gradient descent-based (SGD) are relatively poor performers.

* Corresponding author.
E-mail addresses: dmarron@ac.upc.edu (D. Marrón), jesse.read@telecom-paristech.fr (J. Read), albert.bifet@telecom-paristech.fr (A. Bifet), nacho@ac.upc.edu (N. Navarro).

http://dx.doi.org/10.1016/j.jss.2016.06.009
0164-1212/© 2016 Elsevier Inc. All rights reserved.

Classification in data streams is a major area of research, in which Hoeffding trees have long been a favored method. The main contribution of this paper is to show that random feature function can be leveraged by other algorithms to obtain similar or even improved performance over tree-based methods.

With the recent popularity of Deep Learning (DL) methods we also want to test how a random feature in the form of random projection layer performs on Deep Neural Networks (DNNs).

DL aims for a better data representation at multiple layers of abstraction, and for each layer the network needs to be fine-tuned. In classification, a common algorithm to fine-tune the network is the SGD which tries to minimize the error at the output layer using an objective function, such as Mean Squared Error (MSE). A Gradient vector is used to back-propagate the error to previous layers. This gradient nature of the algorithm makes it suitable to be trained incrementally in batches of size one, similar to how incremental training is done. Unfortunately, DNN are very sensitive to hyper-parameters such as learning rate (η), momentum (μ), number of number neurons per level, or the number of levels. It is then not straight forward to provide an of-the-shelf method for data streams.

Propagation between layers is usually done in the form of matrix-vector or matrix-matrix multiplications, which are computational intensive operation. Often hardware accelerators such as FPGAs or GPUs are used to accelerate the calculations. Despite some efforts, acceleration of HT and *k*NN algorithms for data streams on the GPUs are has some limitations. We talk briefly about this in Section 2.

In recent years, Extreme Learning Machines (Huang, 2015) (ELMs) have emerged as a popular framework in Machine Learning. ELMs are a type of feed-forward neural networks characterized by a random initialization of their hidden layer, combined with a fast training algorithm. Our random feature method is based on this approach.

We made use of the MOA (Massive Online Analysis) framework (Bifet et al., 2010a), a software environment for implementing algorithms and running experiments for online learning from data streams in Java. It implements a large number of modern methods for classification in streams, including HT, *k*NN, and SGD-based methods. We make use of MOA's extensive library of methods to form novel combinations with these methods and further employ an extremely rapid preprocessing technique of projecting the input into a new space via random feature functions (similar to ELMs). We then took the methods purely related to Neural Networks (those which proved most promising under random projections) and implemented them using NVIDIA GPUs and CUDA 7.0; comparing performance to the methods in MOA.

This paper is organized as follows: Section 2 introduces related work on tree based approaches, neural networks, and data streams on GPU. We discuss the use of random features in Sections 3 and 4 for HT/*k*NN methods and neural networks respectively. We first present the evaluation of tree-based methods in Section 5 and later in Section 6 we extend the SGD method in the form of DNNs, using different activation functions. We finally conclude the paper in Section 7.

2. Related work

Hoeffding trees (Domingos and Hulten, 2000) are state-of-the-art in classification for data streams and they predict by choosing the majority class at each leaf. However, these trees may be conservative at first and in many situations naive Bayes method outperforms the standard Hoeffding tree initially, although it is eventually overtaken (Holmes et al., 2005). A proposed hybrid adaptive method by Holmes et al. (2005) is a Hoeffding tree with naive Bayes at the leaves, i.e., returning a naive Bayes prediction at the leaves, if it has been so far more accurate overall than the majority class. Given it's widespread acceptance, this is the default in MOA, and we denote this method in the experimental Section simply as HT. In fact, the naive Bayes classification comes for free, since it can be made with the same statistics that are collected anyway by the tree.

Other established examples include using principal component analysis (reviewed also in Hastie et al., 2001) for this transformation, and also Restricted Boltzmann Machines (RBMs) (Hinton and Salakhutdinov, 2006). RBMs can be seen as a probabilistic binary version of PCA, for finding higher-level feature representations. They have received widespread popularity in recent years due to their use in successful deep learning approaches. In this case, $\mathbf{z} = \phi(\mathbf{x}) = f(\mathbf{W}^\top \mathbf{x})$ for some non-linearity *f*: a sigmoid function is typical, but more recently rectified linear units (ReLUs, Nair and Hinton, 2010) have fallen into favor. The weight matrix $\mathbf{W}$ is learned with gradient-based methods (Hinton, 2000), and the projected output should provide a better feature representation for a neural network or any off-the-shelf method. This approach was applied to data streams already in Read et al. (2015), but concluded that the sensitivity to hyper-parameters and initial conditions prevented good 'out-of-the-box' deployment in data streams.

Approaches such as the so-called extreme learning machines (ELMs) (Huang et al., 2011) avoid tricky parametrizations by simply using random functions (indeed, ELMs are basically linear learners on top of non-linear data transformations). Despite the hidden layer weights being random , it has been proven that ELMs is still capable of universal approximation of any non-constant piecewise continuous function (Huang et al., 2006).

Also an incremental version of ELMs is proposed in bin Huang et al. (2008). It starts with an small network, and new neurons are added at each step until an stopping criterion of size or residual error is reached. The difference with our incremental build is that we use one instance at time simulating they arrive in time, and we incrementally train the network. Also our number of neurons is fixed during the training, in other words, we don't add/remove any neuron during the process.

Nowadays, in 2015, it is difficult when talking about DL and DNNs not to mention GPUs. They are a massive parallel architectures providing an outstanding performance for High Performance Computing and a very good performance/watt ratio, as their architecture suits very fine to their needs of DNNs computations. Many tools include a back-end to offload the computation to the GPU. NVIDIA has its own portal for deep learning on GPUs at https://developer.nvidia.com/deep-learning.

GPUs has not only used to accelerate DL/DNN computations due to its performance, it has been also been used to successfully accelerate HT and ensembles. However, few works are provided in the context of data streams and GPUs.

The only work we are aware of regarding to HT in the context of online real-time data streams mining is Marron et al. (2014), were the authors present a parallel implementation of HT and Random Forests for binary trees and data streams achieving goods speedups, but with limitations on the size and with high memory consumption. More generic HT implementation of Random Forests is presented in Grahn et al. (2011). In Schulz et al. (2015) the authors introduced an open source library, available at github, to predict images labeling using random forests. The library is also tested their on a cell phone with VGA resolution in real-time with good results.

Also, *k*NN has already been successfully ported to GPUs (Garcia et al., 2008). That paper presented one of the first implementations of the "brute force" *k*NN on GPUs, and compared with several CPU-based implementations with speedups up to teo orders of magnitude. *k*NN is also used in business intelligence (Huang et al., 2012) and has also its implementation on the GPU.