2016 23rd International Conference on Pattern Recognition (ICPR)
Cancún Center, Cancún, México, December 4-8, 2016

Distributed Data Augmented Support Vector Machine on Spark

Tu Dinh Nguyen, Vu Nguyen, Trung Le, and Dinh Phung
Center for Pattern Recognition and Data Analytics, Deakin University, Australia
{tu.nguyen, v.nguyen, dinh.phung}@deakin.edu.au, trunglm@hcmup.edu.vn

***Abstract*—Support vector machines (SVMs) are widely-used for classification in machine learning and data mining tasks. However, they traditionally have been applied to small to medium datasets. Recent need to scale up with data size has attracted research attention to develop new methods and implementation for SVM to perform tasks at scale. Distributed SVMs are relatively new and studied recently, but the distributed implementation for SVM with data augmentation has not been developed. This paper introduces a distributed data augmentation implementation for SVM on Apache Spark, a recent advanced and popular platform for distributed computing that has been employed widely in research as well as in industry. We term our implementation *sparkling vector machine* (SkVM) which supports both classification and regression tasks by scanning through the data exactly once. In addition, we further develop a framework to handle the data with new classes arriving under an online classification setting where new data points can have labels that have not previously seen – a problem we term *label-drift classification*. We demonstrate the scalability of our proposed method on large-scale datasets with more than one hundred million data points. The experimental results show that the predictive performances of our method are comparable or better than those of baselines whilst the execution time is much faster at an order of magnitude.**

***Index Terms*—Apache Spark, support vector machine, distributed computing, large-scale classification, big data.**

I. Introduction

Support vector machines (SVMs) [2] are widely-used machine learning methods for classification. Given the rise of big data, these methods have also been recently applied for large-scale linear classification [17]. However, most of the SVM-based implementations are running on a single machine, whilst the ever growing collection of data nowadays is far beyond the capacity that a single machine can handle. Therefore, there is a need for frameworks that support parallel and distributed data processing on multiple machines, both for research communities as well as industry demands.

A new emerging class of such frameworks is Apache Spark [18]. Spark is a cluster computing platform that supports distributed computing, scalability and fault tolerance. Compared with MapReduce [3], a well-known disk-based distributed framework, Spark provides an in-memory processing solution which bypasses the heavy disk I/O bottleneck of reloading the data when performing iterative machine learning methods. In addition, the platform also supports high-level APIs which are more friendly for developers, and especially supplies REPL (read-eval-print-loop) environment for data scientists.

Traditionally, SVMs have been solved via convex optimization techniques (e.g., popularly used SMO method [14]). More recently, SVMs have been reformulated under a probabilistic setting where the original SVM optimization problem can be framed as an MAP estimation [15], [9]. Data augmentation and MCMC technique such as Gibbs sampling have been developed and shown to be effective as an alternative approach to train SVMs. More specifically, Polson *et al*. employed data augmentation formulation in terms of complete data sufficient statistics to improve the mixing rate and computation complexity of SVMs. However, Polson's formulation was developed for binary classification, we further extend this work to support multiclass classification and regression in this paper.

Multiclass classification is the workhorse in the literature of classification, where a set of predefined classes is required either in a batch mode (e.g., [7]) or in an online learning setting (e.g., [1]). These models, however, fail to work in the case where the class labels are not known in advance, a problem we term online *label-drift classification*, in a similar spirit of the concept drift problem known in the literature [5]. Label-drift classification problem naturally occurs in many applications, especially in the context of streaming and online settings where the incoming data may contain samples categorized with new classes that have not been previously seen [6], [13]. Despite its significance, this problem has received little research attention.

To handle data with new classes, one straightforward solution is to restart the whole process to incorporate new unseen data and labels. However, retraining models requires looping over the entire data again and again, leading to an expensive computation that is undesirable for large (growing) datasets. To this end, our proposed solution can naturally handle the new classes without retraining from scratch using an efficient algorithm to update the model on the fly. This significantly reduces the computational complexity, resulting in a very fast and flexible training procedure.

By examining the SVM with data augmentation under a Bayesian setting, we further introduce a distributed model that can parallelize the algorithm to compute its maximum a posteriori (MAP) estimation. Although similar ideas on the exploitation of data augmentation for SVM have been independently proposed in the technical report of [12], our work differs in two points. First we address the label-drift classification problem mentioned previously, a natural, yet challenging problem in big data analytics. Second, there is no

978-1-5090-4846-5/16/$31.00 ©2016 IEEE

distributed implementation for the SVM with data augmentation exists, and our work provides the first implementation of its kind on Apache Spark providing the capacity to conduct supervised learning tasks using SVM for massive datasets at real-world and industry scale. This paper is an extension of our previous work [10] and our implementation will also be released to the research community and public use.

For data practitioners, our model has an additional advantage where it has no hyperparameter to tune, thus there is no need to perform an expensive cross-validation to choose optimal hyperparameters. We conduct experiments on large-scale datasets to demonstrate the capacity of the proposed method. The results show that our method obtains a significant speedup in the training phase, compared with the existing baselines implemented in Spark, and is substantially faster than our single machine implementation counterpart. At the same time, the classification performances are better than or comparable with those of the baselines. We term the resulting model the *sparkling vector machine* (SkVM).

In short, our main contributions are: (i) the extension of a powerful data augmentation technique for SVMs to support multiclass classification, regression and label-drift classification; (ii) the derivation of SkVM, a distributed method that parallelizes the MAP estimation of such augmentation approach; (iii) an implementation of our proposed model in the recent advanced distributed system – Apache Spark; and (iv) a comprehensive evaluation the capability and scalability of SkVM on large-scale datasets with approximately one hundred million data points.

II. Sparkling Vector Machines

A. Latent variable models for SVMs

The SVM aims to find an optimal hyperplane that maximizes the margin between different labeled sets of data samples. More formally, let $\mathcal{D} = \{(\mathbf{x}_n, \mathrm{y}_n)\}_{n=1}^{\mathrm{N}}$ denote the dataset wherein $\mathbf{x}_n \in \mathbb{R}^{\mathrm{D}}$ is the D-dimensional vector of the data sample and $\mathrm{y}_n \in \{-1, 1\}$ is the data label. The learning of SVM with ℓ_α-norm regularization is to minimize the objective function as follows:

$$L(\mathbf{w}; C, \alpha, \boldsymbol{\sigma}) = \sum_{n=1}^{\mathrm{N}} \max\left\{1 - \mathrm{y}_n \mathbf{w}^\top \mathbf{x}_n, 0\right\} + \frac{1}{C^\alpha} \sum_{d=1}^{\mathrm{D}} \left|\frac{\mathrm{w}_d}{\sigma_d}\right|^\alpha \quad (1)$$

where $\mathbf{w}$ is the vector of coefficient parameters, σ_d is the standard deviation of the d-th feature of $\mathbf{x}$, $C > 0$ is the penalty hyperparameter that can be tuned for the best performance using cross-validation. The second term is a regularization penalty corresponding to a prior distribution $p(\mathrm{w}_d \mid C, \alpha, \sigma_d)$.

A pseudo-likelihood of the label y has been introduced to represent SVMs as latent variable models, so that Bayesian inference techniques can be employed to perform parameter estimation [15]. The pseudo-likelihood is: $p(\mathrm{y} \mid \mathbf{x}, \mathbf{w}) = \exp\left\{-2\max\left(1 - \mathrm{y}\mathbf{w}^\top \mathbf{x}, 0\right)\right\}$. Minimizing the loss function in Eq. (1) now turns into estimating the maximum a posterior (MAP) of the following pseudo-posterior distribution:

$$\begin{aligned}\hat{\mathbf{w}}_{\text{MAP}} &\propto \underset{\mathbf{w}}{\text{argmax}} \exp\left[-L(\mathbf{w}; C, \alpha, \boldsymbol{\sigma})\right] \\ &\propto \underset{\mathbf{w}}{\text{argmax}} \mathcal{Z}_\alpha(C, \boldsymbol{\sigma})\, p(\mathrm{y} \mid \mathbf{x}, \mathbf{w})\, p(\mathbf{w} \mid C, \alpha, \boldsymbol{\sigma})\end{aligned}$$

in which $\mathcal{Z}_\alpha(C, \boldsymbol{\sigma})$ is a pseudo-posterior normalization constant. For the purpose of model simplicity, we consider a special case with Gaussian prior ($\alpha = 2$) and a fixed penalty constant $C = 1$.

Data augmentation approach further introduces an auxiliary variable $\lambda > 0$ for each observation label y [15], [11], in such a way that $p(\mathrm{y} \mid \mathbf{x}, \mathbf{w})$ becomes the marginal of the joint distribution $p(\mathrm{y}, \lambda \mid \mathbf{x}, \mathbf{w})$. More importantly, the auxiliary variable λ can be efficiently sampled from an *Inverse Gaussian* (IG) distribution:

$$p\left(\lambda^{-1} \mid \mathbf{x}, \mathrm{y}, \mathbf{w}\right) \sim \text{IG}\left(\left|1 - \mathrm{y}\mathbf{w}\mathbf{x}^\top\right|^{-1}, 1\right)$$

Assuming a Gaussian prior for $\mathbf{w}$: $p(\mathbf{w}) \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, the data pseudo-likelihood and the posterior conditional distribution of $\mathbf{w}$ can be shown to have the following forms [15]:

$$p(\mathrm{y} \mid \mathbf{x}, \mathbf{w}, \lambda) = \int_0^\infty \frac{1}{\sqrt{2\pi\lambda}} \exp\left\{-\frac{\left[\lambda + \left(1 - \mathrm{y}\mathbf{w}^\top \mathbf{x}\right)\right]^2}{2\lambda}\right\} \mathrm{d}\lambda \quad (2)$$

$$p(\mathbf{w} \mid \mathbf{x}, \mathrm{y}, \lambda) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (3)$$

where $\boldsymbol{\Sigma}^{-1} = \mathbf{X}^\top \text{diag}\left(\boldsymbol{\lambda}^{-1}\right)\mathbf{X} + \boldsymbol{\Sigma}_0^{-1}$, $\boldsymbol{\mu} = \boldsymbol{\Sigma}\left[\mathbf{X}^\top\left(\mathbf{1} + \boldsymbol{\lambda}^{-1}\right) + \boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0\right]$ with $\mathbf{1}$ denotes a vector of 1's. Here the data matrix is denoted by $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{\mathrm{N}}]^\top$.

Viewing SVMs under this latent variable perspective enables us to employ either expectation maximization using point estimation, or Markov chain Monte Carlo (MCMC) algorithms using Bayesian inference to learn parameters [15]. It is also proven that such algorithms are more robust and provide more accurate parameter estimation than what learned by the standard solvers of SVMs [15]. In addition, the latent SVM offers a nice closed-form of MAP estimation that can be efficiently parallelized in distributed manner (see Section II-E).

B. Multiclass latent SVMs

Here we extend the latent SVM schemes presented in Section II-A for multiclass classification. Suppose that there are K classes, the label y_n now takes values on the set $\{1, 2, .., \mathrm{K}\}$. We consider a set of parameters $\{\mathbf{w}_1, \mathbf{w}_2, ...\mathbf{w}_{\mathrm{K}}\}$ wherein the parameter for the k-th class is $\mathbf{w}_k$. These parameters can be initially set to $\mathbf{1}$. Then the auxiliary variable λ_n for the n-th data point is independently sampled as follows:

$$\lambda_n^{-1} \sim \text{IG}\left(\left|1 - \mathbf{w}_{\mathrm{y}_n}\mathbf{x}_n^\top\right|^{-1}, 1\right) \quad (4)$$

Consider the case where the prior distribution for $\mathbf{w}_k$ is a normal distribution with zero mean ($\boldsymbol{\mu}_0 = 0$) and unit variance ($\boldsymbol{\Sigma}_0 = \mathbf{I}$): $p(\mathbf{w}_k \mid \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \sim \mathcal{N}(0, \mathbf{I})$. We can then derive the MAP for the posterior distribution for each $\mathbf{w}_k$ as: $\mathbf{w}_k = \boldsymbol{\mu}_k = \boldsymbol{\Sigma}_k \mathbf{X}^\top\left[\mathbf{I} + \text{diag}\left(\boldsymbol{\lambda}^{-1}\right)\right]\mathbf{Z} + \boldsymbol{\mu}_0\boldsymbol{\Sigma}_0$ where $\mathbf{Z} \in \{-1, 1\}^{\mathrm{N}\times\mathrm{K}}$ denotes the indicator matrix for the labels: $\mathrm{z}_{nk} = 1$ if $\mathrm{y}_n = k$, otherwise $\mathrm{z}_{nk} = -1$. Let:

$$\mathbf{P} \in \mathbb{R}^{\mathrm{D}\times\mathrm{D}} = \mathbf{X}^\top \text{diag}\left(\boldsymbol{\lambda}^{-1}\right)\mathbf{X} + \mathbf{I} \quad (5)$$

$$\mathbf{Q} \in \mathbb{R}^{\mathrm{D}\times\mathrm{K}} = \mathbf{X}^\top\left[\mathbf{I} + \text{diag}\left(\boldsymbol{\lambda}^{-1}\right)\right]\mathbf{Z} \quad (6)$$

From Eqs. (3), (5) and (6), we have $\mathbf{P}\mathbf{w}_k = \mathbf{Q}_{\bullet k}$ where $\mathbf{Q}_{\bullet k}$ is the k-th column vector of matrix $\mathbf{Q}$. Therefore, we

obtain $\mathbf{w}_k = \mathbf{P}\backslash\mathbf{Q}_{\cdot k}$ wherein the backslash ($\backslash$) indicates the solving the system of linear equations. We prefer solving the linear system of equations to computing the inversion of the matrix $\mathbf{P}$ for computational efficiency. One can use an iterative algorithm to alternatively sample the latent variable λ_n and compute $\mathbf{w}$. However, in practice we empirically find that a single pass gains sufficiently good performance, thus we only sample λ_n once. Once the parameters have been learned, the label $\hat{y}_{new}$ for a newly seen data instance $\mathbf{x}_{new}$ can be predicted as: $\hat{\mathrm{y}}_{new} = \underset{k}{\operatorname{argmax}} \left[\mathbf{w}_k^\top \mathbf{x}_{new}\right]$.

C. Label-drift latent SVMs

When the latent SVM scans through the data in one pass, the sufficient statistic matrices $\mathbf{P}, \mathbf{Q}$ to estimate the parameters $\mathbf{w}$ can be updated incrementally as the new data come (see Eqs. (4,5,6)). This allows the model to be learned in an online setting where the data arrive in the form of mini-batches. When a mini-batch contains only one data point, it reverts to the standard online learning mode.

Furthermore, the parameters of each class in multiclass latent SVMs are updated independently (cf. Section II-B). This naturally suggests a new learning capability where our proposed method can adapt, without retraining from scratch, to learn new data with new class labels which have not been seen before during the training phase – a problem we term *label-drift classification*.

Figure 1 shows an illustration of the data setup for this task. In this setting, the data is divided into $t+1$ mini-batches: $\{B_i = (\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{t+1}$. The labels of the first i batches belong to the set $\mathcal{K}_i$, thus $\mathcal{K}_i \subseteq \mathcal{K}_j$, $i \leq j$. We have already trained the model using the first t batches and the goal now is to update its parameters using data batch $t+1$.

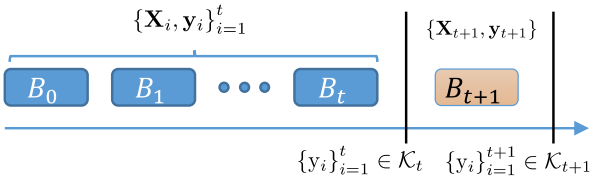

Fig. 1. An illustration of label-drift classification problem.

Denote $\mathcal{K}_t = \{1, 2, ..., \mathrm{K}\}$ and $\mathcal{K}_{t+1} = \{1, 2, ..., \mathrm{K}, \mathrm{K}+1\}$, we adjust the parameters $\{\mathbf{w}_1, \mathbf{w}_2, ...\mathbf{w}_\mathrm{K}\}$ and sufficient statistic $\{\mathbf{P}, \mathbf{Q}\}$ of the latent SVM, and at the same time add new parameters $\mathbf{w}_{\mathrm{K}+1}$ and $\mathbf{Q}_{\cdot[\mathrm{K}+1]}$ for the new class label $\mathrm{K}+1$. To do so, we introduce a new column $\mathbf{Q}_{\cdot 0}$ into the matrix $\mathbf{Q}$ to keep information of previous data points as they will be discarded later on. The additional column $\mathbf{Q}_{\cdot 0}$ is used to penalize the sufficient statistic $\mathbf{Q}_{\cdot[\mathrm{K}+1]}$ of the new class as all previous data points do not belong to the new class. The parameter update of label-drift latent SVMs for a mini-batch is presented in Alg. 1.

D. Latent support vector regressions

We propose in this section the use of the latent SVM presented in Section II-A for regression task. In a standard support vector regression (SVR), the data label is represented by a real-valued response variable $\mathrm{y}_n \in \mathbb{R}$. The objective function in Eq. (1) now becomes:

$$L\left(\mathbf{w}; C, \alpha, \boldsymbol{\sigma}\right) = \sum_{n=1}^{\mathrm{N}} \max\left(\left|\mathrm{y}_n - \mathbf{w}^\top \mathbf{x}_n\right|, 0\right) + \frac{1}{C^\alpha} \sum_{d=1}^{\mathrm{D}} \left|\frac{\mathrm{w}_d}{\sigma_d}\right|^\alpha$$

wherein we have utilized ℓ_1-loss function. One can use the ϵ-insensitive loss [16], but this introduces a hyperparameter ϵ (i.e., the precision parameter), and thus we do not include this loss to keep our models hyperparemeter-free.

To develop a data augmentation approach for SVR, we introduce two auxiliary variables $\lambda, \eta > 0$ (rather than one as in latent SVMs for classification task) for each observation label y. The inverses λ^{-1}, η^{-1} of such auxiliary variables also follow *Inverse Gaussian* (IG) distributions:

$$p\left(\lambda^{-1} \mid \mathbf{x}, \mathrm{y}, \mathbf{w}\right) \sim \mathrm{IG}\left(\left|\mathrm{y} - \mathbf{w}\mathbf{x}^\top\right|^{-1}, 1\right)$$
$$p\left(\eta^{-1} \mid \mathbf{x}, \mathrm{y}, \mathbf{w}\right) \sim \mathrm{IG}\left(\left|\mathrm{y} - \mathbf{w}\mathbf{x}^\top\right|^{-1}, 1\right)$$

The data pseudo-likelihood in Eq. (2) now turns into the following double scale mixture form:

$$p\left(\mathrm{y} \mid \mathbf{x}, \mathbf{w}, \lambda\right) = \int_0^\infty \frac{1}{\sqrt{2\pi\lambda}} \exp\left\{-\frac{\left[\lambda + \left(\mathrm{y} - \mathbf{w}^\top \mathbf{x}\right)\right]^2}{2\lambda}\right\} \mathrm{d}\lambda$$
$$\times \int_0^\infty \frac{1}{\sqrt{2\pi\eta}} \exp\left\{-\frac{\left[\eta - \left(\mathrm{y} - \mathbf{w}^\top \mathbf{x}\right)\right]^2}{2\eta}\right\} \mathrm{d}\eta$$

The posterior conditional distribution of $\mathbf{w}$: $p\left(\mathbf{w} \mid \mathbf{x}, \mathrm{y}, \lambda, \eta\right) = \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$ has the following mean and covariance:

$$\boldsymbol{\Sigma}^{-1} = \mathbf{X}^\top \left[\mathrm{diag}\left(\boldsymbol{\lambda}^{-1}\right) + \mathrm{diag}\left(\boldsymbol{\eta}^{-1}\right)\right] \mathbf{X} + \mathbf{I}$$
$$\boldsymbol{\mu} = \boldsymbol{\Sigma}\mathbf{X}^\top \left[\mathrm{diag}\left(\boldsymbol{\lambda}^{-1}\right) + \mathrm{diag}\left(\boldsymbol{\eta}^{-1}\right)\right] \mathbf{y}$$

The parameters of SVRs are then computed similarly to those of the multiclass latent SVMs.

Algorithm 1 Parameter update of label-drift latent SVMs for a mini-batch containing M data points.

Input: $\mathbf{y} = \{1, .., \mathrm{K}+1\}^{\mathrm{M}\times 1}$, $\{\mathrm{y}_m \mid \mathrm{y}_m = \mathrm{K}+1\}_{m=1}^{\mathrm{M}} \neq \emptyset$, $\mathbf{X} \in \mathbb{R}^{\mathrm{M}\times\mathrm{D}}$, $\mathbf{P} \in \mathbb{R}^{\mathrm{D}\times\mathrm{D}}$ and $\mathbf{Q} \in \mathbb{R}^{\mathrm{D}\times(\mathrm{K}+1)}$

1: Initialize $\mathbf{w}_k = \mathbf{1}, \forall k = 1, 2, ..., \mathrm{K}+1$
2: $\boldsymbol{\lambda} = [\lambda_m]_{m=1}^{\mathrm{M}} : \lambda_m^{-1} \sim \mathrm{IG}\left(\left|1 - \mathbf{w}_{\mathrm{y}_m}\mathbf{x}_m^\top\right|^{-1}, 1\right)$
3: $\hat{\mathbf{P}} = \mathbf{P} + \mathbf{X}^\top \mathrm{diag}\left(\boldsymbol{\lambda}^{-1}\right) \mathbf{X}$
4: $\mathbf{Z} = [\mathrm{z}_{mk}]_{\mathrm{M}\times(\mathrm{K}+1)} : \mathrm{z}_{mk} = 1 \mid \mathrm{y}_m = k, \mathrm{z}_{mk} = -1 \mid \mathrm{y}_m \neq k$
5: $\hat{\mathbf{Q}}_{\cdot[1:\mathrm{K}+1]} = \mathbf{Q}_{\cdot[1:\mathrm{K}+1]} + \mathbf{X}^\top \left[\mathbf{I} + \mathrm{diag}\left(\boldsymbol{\lambda}^{-1}\right)\right] \mathbf{Z}$
6: $\hat{\mathbf{Q}}_{\cdot[\mathrm{K}+1]} = \hat{\mathbf{Q}}_{\cdot[\mathrm{K}+1]} - \mathbf{Q}_{\cdot 0}$
7: $\hat{\mathbf{Q}}_{\cdot 0} = \mathbf{Q}_{\cdot 0} + \sum_{k=1}^{\mathrm{K}} \mathbf{X}^\top \left[\mathbf{I} + \mathrm{diag}\left(\boldsymbol{\lambda}^{-1}\right)\right] \mathbb{I}\left[\mathbf{y} = k\right]$
8: $\mathbf{w}_k = \mathbf{P}\backslash\mathbf{Q}_{\cdot k}, \forall k = 1, 2, ..., \mathrm{K}+1$

Output: $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{\mathrm{K}+1}\}$, $\hat{\mathbf{P}}$, and $\hat{\mathbf{Q}} \in \mathbb{R}^{\mathrm{D}\times(\mathrm{K}+2)}$

E. Latent SVMs on Spark

In tackling large-scale data classification problems, the number of data points is often significantly greater than the number of features, i.e., $\mathrm{N} \gg \mathrm{D}$. Our computational bottlenecks reside in computing $\mathbf{P}$ and $\mathbf{Q}$ in Eqs. (5) and (6) with the computational complexity of $\mathcal{O}\left(\mathrm{N} \times \mathrm{D}^2\right)$ and $\mathcal{O}\left(\mathrm{N} \times \mathrm{D} \times \mathrm{K}\right)$, respectively. These operations, however, can be parallelized in a distributed system in a similar way to parallelize matrix multiplication problem. Thus we can efficiently address these issues using a distributed computing framework, specifically on Apache Spark.

1) Distributed algorithm: Without loss of generality, we present here the distributed version developed for the latent SVMs for multiclass classification; the regression case follows in a similar way. We first partition the data matrix $\mathbf{X}$ and the label $\mathbf{y}$ into disjoint M parts: $\mathbf{X} = \left[\mathbf{X}^{\langle 1 \rangle}, \mathbf{X}^{\langle 2 \rangle}, ..., \mathbf{X}^{\langle \mathrm{M} \rangle}\right]^\top$ and $\mathbf{y} = \left[\mathbf{y}^{\langle 1 \rangle}, \mathbf{y}^{\langle 2 \rangle}, ..., \mathbf{y}^{\langle \mathrm{M} \rangle}\right]^\top$. These partitions are then stored distributedly in multiple machines. For the m-th part, the auxiliary variables $\boldsymbol{\lambda}^{\langle m \rangle}$ are sampled using Eq. (4) with $\mathbf{x}_n \in \mathbf{X}^{\langle m \rangle}$ and $\mathrm{y}_n \in \mathbf{y}^{\langle m \rangle}$. These auxiliary variables reside in the corresponding data partitions. The functions to compute matrices $\mathbf{P} \in \mathbb{R}^{\mathrm{D} \times \mathrm{D}}$ and $\mathbf{Q} \in \mathbb{R}^{\mathrm{K} \times \mathrm{D}}$ can be reformulated using M parts as: $\mathbf{P} = \sum_{m=1}^{\mathrm{M}} \mathbf{P}^{\langle m \rangle} + \mathbf{I},\ \mathbf{Q} = \sum_{m=1}^{\mathrm{M}} \mathbf{Q}^{\langle m \rangle}$ wherein the m-th parts are computed as:

$$\mathbf{P}^{\langle m \rangle} = {\mathbf{X}^{\langle m \rangle}}^\top \mathrm{diag}\left(\boldsymbol{\lambda}^{\langle m \rangle^{-1}}\right) \mathbf{X}^{\langle m \rangle}$$
$$\mathbf{Q}^{\langle m \rangle} = {\mathbf{X}^{\langle m \rangle}}^\top \left[\mathbf{I} + \mathrm{diag}\left(\boldsymbol{\lambda}^{\langle m \rangle^{-1}}\right)\right] \mathbf{Z}^{\langle m \rangle}$$

It is clear that we only need the data part $\mathbf{X}^{\langle m \rangle}$ and auxiliary variable part $\boldsymbol{\lambda}^{\langle m \rangle}$ to compute $\mathbf{P}^{\langle m \rangle}$ and $\mathbf{Q}^{\langle m \rangle}$. Therefore, in the Spark system, the computation can be done in parallel with each partition being processed by one worker node. The sampling and computations of $\boldsymbol{\lambda}^{\langle m \rangle}$, $\mathbf{P}^{\langle m \rangle}$, $\mathbf{Q}^{\langle m \rangle}$ are *map* functions operating on the m-th partition. After the *map* functions are computed for all M parts, *reduce* steps are needed to summing over all results to obtain the $\mathbf{P}$ and $\mathbf{Q}$. This algorithm requires two phases of communications between driver and worker nodes. In the first phase, the driver machine must ship the parameters $\mathbf{w}$ to all worker machines which perform *map* functions. The results of these functions are then reduced and sent back to the driver in the second phase.

Once the matrices $\mathbf{P}$ and $\mathbf{Q}$ are fully specified, the driver will compute the parameters $\mathbf{w}$ by solving the system of linear equations. It is noteworthy to point out that this operator is conducted on the driver node, thus not distributed. However, it performs on the feature dimension with the computational complexity $\mathcal{O}\left(\mathrm{D}^{2.376}\right)$ [4]. Thus this step will not become a bottleneck in our method. The pseudo-code is described in Alg. 2. We term our proposed method the *sparkling vector machines* (SkVM).

2) Implementation: We use Python API[1] of Spark version 1.4.1 to implement our proposed method. For reproducibility of our experiments and open science, we make our source code available at https://github.com/tund/skvm. In the near future, we also intend to release an implementation in Scala – the native programming language in which Spark is written, and its Python wrapper.

Algorithm 2 Distributed learning algorithm of SkVM.

Input: $\mathbf{y} \in \{1, 2, ..., \mathrm{K}\}^{\mathrm{N} \times 1}, \mathbf{X} \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$

1: Initialize $\mathbf{w}_k = \mathbf{1},\ \forall k = 1, 2, ..., \mathrm{K}$
2: The driver ships $\mathbf{w}_{1:\mathrm{K}}$ to every worker.
3: The workers perform the following steps for the m-th partition:
4: $\lambda_n^{-1} \sim \mathrm{IG}\left(\left|1 - \mathbf{w}_{\mathrm{y}_n} \mathbf{x}_n^\top\right|^{-1}, 1\right), \forall n \in m$-th partition,
5: $\mathbf{P}^{\langle m \rangle} = {\mathbf{X}^{\langle m \rangle}}^\top \mathrm{diag}\left(\boldsymbol{\lambda}^{\langle m \rangle^{-1}}\right) \mathbf{X}^{\langle m \rangle}$
6: $\mathbf{Z}^{\langle m \rangle} = \left[\mathrm{z}_{nk}^{\langle m \rangle}\right]_{\mathrm{N} \times \mathrm{K}} : \mathrm{z}_{nk}^{\langle m \rangle} = 1 \mid \mathrm{y}_n = k, \mathrm{z}_{nk}^{\langle m \rangle} = -1 \mid \mathrm{y}_n \neq k$
7: $\mathbf{Q}^{\langle m \rangle} = {\mathbf{X}^{\langle m \rangle}}^\top \left[\mathbf{I} + \mathrm{diag}\left(\boldsymbol{\lambda}^{\langle m \rangle^{-1}}\right)\right] \mathbf{Z}^{\langle m \rangle}$
8: The workers reduce and send $\mathbf{P}^{\langle m \rangle}$ and $\mathbf{Q}^{\langle m \rangle}$ back to the driver to obtain $\mathbf{P}$ and $\mathbf{Q}$
9: The driver computes: $\mathbf{w}_k = \mathbf{P} \backslash \mathbf{Q}_k : \forall k = 1, ..., \mathrm{K}$

Output: $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_\mathrm{K}\}$

III. Evaluations

In this section, we quantitatively evaluate the performance of our proposed method on three tasks: standard classification, label-drift classification and regression. Our main goal is to demonstrate the scalability of SkVM in training large-scale datasets, and its capability in handling the arrival of data points with new labels, or the label-drift classification problem described earlier. We directly compare the performance of our proposed model with several existing methods implemented in machine learning library (MLlib) of Apache Spark [8].

A. Data statistics

We use four large-scale public datasets (i.e., Epsilon, Susy, MNIST8M and Airlines dataset) in which the number of training samples is much larger than the feature dimension (millions against hundreds). The first three datasets were obtained from UCI repository[2] and LIBSVM collection[3]. These datasets are already available in the form of instance-feature matrices, and divided into training and testing subsets.

The Airlines dataset is provided by the American Statistical Association[4]. The dataset contains information of all commercial flights in the US from October 1987 to April 2008. The aim is to predict whether a flight will be delayed or not. A flight is considered *delayed* if its delay time is above 15 minutes, and *non-delayed* otherwise. Our preprocessing consists of two steps. First, we extract 11 fields (year; month; days of week and month; scheduled departure and arrival hours; unique carrier code; origin and destination airport

[1] http://spark.apache.org/docs/1.4.1/api/python/

[2] https://archive.ics.uci.edu/ml/datasets

[3] https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

[4] http://stat-computing.org/dataexpo/2009/.

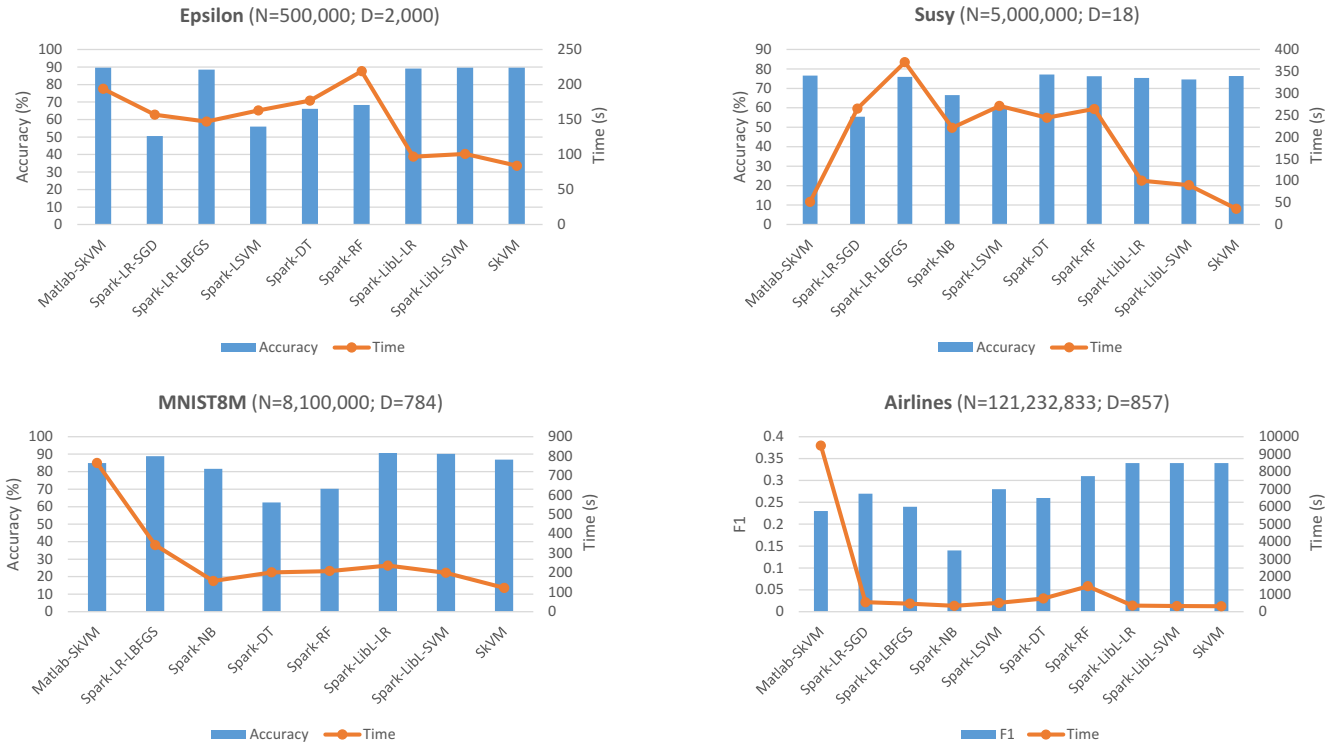

Fig. 2. Comparison of classification performances with 7 baselines on four large-scale datasets.

codes; reason for cancellation; diverted or not) as categorical features which are encoded into one-hot representations, and distance as a real-valued feature. The departure delay time is used to label the delayed flight. We then split the data into 90% for training and 10% for testing. This results in $109,106,460$ training and $12,126,373$ testing data points with 857 features.

B. Classification

In the first experiment, we examine the performances of the proposed SkVMs in binary and multiclass classification tasks. The classification performance is evaluated using the accuracy for the Epsilon, Susy, MNIST8M datasets as their label quantities are almost equal, whilst using F1 for the Airlines data as it is unbalanced with only 10% delayed labels. For further comparison, we implement a version of our proposed model in Matlab (denoted by Matlab-SkVM) and use 8 baselines implemented in Spark: logistic regression using stochastic gradient descent (Spark-LR-SGD) and limited-memory BFGS (Spark-LR-LBFGS); Naive Bayes (Spark-NB); linear SVM (Spark-LSVM); decision tree (Spark-DT); random forest (Spark-RF) [8], Spark-LIBLINEAR with logistic regression (Spark-LLin-LR) and with L2-loss SVM (Spark-LLin-SVM) [7]. All Spark-based methods run on a Hadoop-Spark cluster with 8 worker nodes, each node equipped with 32 vcores CPU, 128GB RAM. The Matlab-SkVM runs in parallel on multiple vcores of a single machine and alternatively loads and unloads smaller chunks of data, which is more efficient than loading the entire data.

Fig. 2 presents the classification performance on the testing set and execution time on both training and testing sets. Note that the Python API of Spark-LR-SGD and Spark-LSVM have not supported multiclass classification, and the Spark-NB has not been implemented to model Gaussian distribution (real-valued data). Hence their results for some datasets are not available. Overall, the training time of SkVM is consistently superior to the Spark-implementation baselines and our Matlab implementation while keeping comparable predictive performances. Our method can achieve ten times speedup in training the largest dataset (Airlines). Particularly, compared to the Spark-LIBLINEAR – the most competitive baseline, the SkVM achieves comparable classification results and from 10% (Airlines) to 250% (Susy) faster speed. For Susy dataset, the Matlab-SkVM performs surprisingly fast as this dataset is small in terms of size on disk (less than 1GB), thus the model can load data significantly faster.

C. Label-drift classification

Next we demonstrate the new classification capacity of our proposed method in handling the label-drift classification problem using the MNIST8M dataset. The data contains images of 10 classes of handwritten digits ranging from 0 to 9. Let $\{i_0, i_1, ..., i_9\}$ be an array of class indices in an arbitrary order. The dataset is then partitioned into ten blocks. The first block contains one-tenth the number of instances of class i_0, denoted by $\left\{\frac{1}{10}i_0\right\}$. The second block contains another one-tenth and one-ninth of class i_1, denoted by $\left\{\frac{1}{10}i_0; \frac{1}{9}i_1\right\}$, and

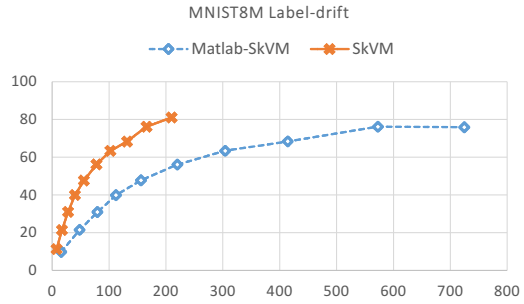

Fig. 3. The label-drift classification performance on MNIST8M dataset.

similarly for the remaining 8 blocks. Thus the last one consists of $\left\{\frac{1}{10}i_0; \frac{1}{9}i_1; ...; \frac{1}{2}i_8; i_9\right\}$. After training the model on each block, we record its classification performance on the original testing part of MNIST dataset which contains 10,000 images.

Fig. 3 shows the relative performance with respect to classification accuracy and computational time cost of SkVM and Matlab-SkVM. It is clear that our Spark implementation consistently performs much faster than the Matlab implementation. There is a minor difference between accuracies of SkVM and Matlab-SkVM with the higher belonging the SkVM due to the random sampling process. After seeing the entire data, the final classification result is slightly worse than that of learning in the standard multiclass classification setting (see Fig. 2). This is because the numbers of class instances are not equal in each stage when the model receiving the new data block. Thus this task can be referred to as the unbalanced classification that is more difficult.

D. Regression

Our last experiment is to validate the capability of our proposed method for regression problem with ℓ_1-loss described in Section II-D. We use the Airlines dataset (predict flight delay minutes) and compare with five baseline implementations in Spark MLLib [8]: Spark-DT, Spark-RF, linear regression using SGD (Spark-LinR-SGD), lasso using SGD (Spark-Lasso-SGD) and ridge regression using SGD (Spark-RidR-SGD). We do not include the Spark-LIBLINEAR in this experiment as it does not support regression. Table I reports the results measured using mean absolute error (MAE). Our proposed models achieve the most optimal regression results which are 18% better than that of the best baseline – Spark-Lasso-SGD. For wall-clock time, the SkVM outperforms the fastest Spark-implementation baseline by twice, and significantly improves the speed of Matlab-SkVM by 30 times.

TABLE I
COMPARISON OF REGRESSION PERFORMANCES.

Dataset	Airlines	
Method	MAE	Time
Matlab-SkVM	**10.89**	3,882.38
Spark-LinR-SGD	13.86	*272.73*
Spark-Lasso-SGD	13.52	325.17
Spark-RidR-SGD	13.85	283.79
Spark-DT	13.78	363.64
Spark-RF	13.82	661.36
SkVM	*11.07*	**142.73**

IV. CONCLUSION

We have introduced a distributed version termed *sparkling vector machine* (SkVM) which is implemented in Apache Spark. The SkVM is based on the data augmentation technique for SVMs. Our main contributions are the extensions to handle multiclass classification, regression, label-drift classification, and the distributed algorithm. Our experiments on large-scale datasets with hundreds of million data points confirm the scalability of our proposed method as well as its predictive performance where the performance of our proposed SkVM are comparable or better than state-of-the-art baselines (including those already implemented in Spark) whilst the execution time is much faster at an order of magnitude.

REFERENCES

[1] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 2005.
[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[3] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[4] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.
[5] D. Hand, "Classifier technology and the illusion of progress," *Statistical Science*, vol. 21, no. 1, pp. 1–14, 2006.
[6] X.-S. Hua and G.-J. Qi, "Online multi-label active learning for large-scale multimedia annotation," Tech. Rep., 2008.
[7] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin, "Large-scale logistic regression and linear support vector machines using spark," in *Proceedings of IEEE International Conference on Big Data*, 2015, pp. 519–528.
[8] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *arXiv preprint arXiv:1505.06807*, 2015.
[9] K. Nguyen, T. Le, V. Nguyen, T. D. Nguyen, and D. Phung, "Multiple kernel learning with data augmentation," in *Proceedings of the 8th Asian Conference on Machine Learning (ACML)*, 2016.
[10] T. D. Nguyen, V. Nguyen, T. Le, and D. Phung, "Sparkling vector machines," in *Proceedings of the Workshop on Machine Learning Systems at Neural Information Processing Systems (NIPS)*, Palais des Congres de Montreal, Montreal, Canada, December 2015.
[11] V. Nguyen, T. D. Nguyen, T. Le, D. Phung, and S. Venkatesh, "One-pass logistic regression for label-drift and large-scale classification on distributed systems," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, Barcelona, Spain, 12–15 December 2016.
[12] H. Perkins, M. Xu, J. Zhu, and B. Zhang, "Fast parallel svms using data augmentation," *Technical Report, Department of Computer Science and Technology, Tsinghua University*, 2013.
[13] A. T. Pham, R. Raich, X. Z. Fern, J. P. Arriaga, and D. UNICAN, "Multi-instance multi-label learning in the presence of novel class instances," in *Proceedings of The 32nd ICML*, 2015, pp. 2427–2435.
[14] J. C. Platt, "12 fast training of support vector machines using sequential minimal optimization," *Advances in kernel methods*, pp. 185–208, 1999.
[15] N. G. Polson, S. L. Scott *et al.*, "Data augmentation for support vector machines," *Bayesian Analysis*, vol. 6, no. 1, pp. 1–23, 2011.
[16] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.
[17] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "Recent advances of large-scale linear classification," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.
[18] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.