Sparkling Vector Machines

Tu Dinh Nguyen†, Vu Nguyen†, Trung Le‡, Dinh Phung†
tu.nguyen@deakin.edu.au, v.nguyen@deakin.edu.au
trunglm@hcmup.edu.vn, dinh.phung@deakin.edu.au

†Center for Pattern Recognition and Data Analytics
School of Information Technology, Deakin University, Geelong, Australia
‡Faculty of Information Technology,
HCMc University of Pedagogy, Ho Chi Minh city, Vietnam

Abstract

Support vector machines (SVMs) are widely-used for classification task in literature. A data augmentation algorithm is proposed to improve the learning of the machinery. Distributed SVMs are well-studied, but the distributed implementation for SVM with data augmentation has not been explored. This paper introduces a distributed version called *sparkling vector machine* which is implemented in Apache Spark, a recent advanced platform for distributed computing. We demonstrate the scalability of our proposed method on large-scale datasets with hundreds of million data points. The experimental results show that the predictive performances of our method are better than or comparable with those of baselines whilst the execution time is orders of magnitude lower.

1 Introduction

Support vector machines (SVMs) [1] are widely-used machine learning methods for classification. The methods can also apply on big data for large-scale linear classification [2]. A data augmentation algorithm is integrated into the machinery to increase the mixing rate and improve the computation complexity using the formulation in terms of complete data sufficient statistics [3]. More importantly, this augmentation scheme allows us to examine the SVM under Bayesian settings. Most of the SVM-based implementations are running on a single machine. However, the ever growing collection of data nowadays is far beyond the capacity of a single machine. Therefore, there is a need for frameworks that support parallel and distributed data processing on multiple machines.

A new emerging class of such frameworks is Apache Spark [4]. Spark is a cluster computing platform that supports distributed computing, scalability and fault tolerance. Compared with MapReduce [5], 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 (real-evaluate-print-loop) environment for data scientists.

In this paper, we extend the data augmentation framework of SVMs to support multiclass classification, and introduce a distributed model that parallelizes the maximum a posterior (MAP) estimation of such framework. 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 training phrase, compared with the existing baselines implemented in Spark, and is substantially faster than its Matlab implementation. 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 multiclass setting; (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 significant evaluation the capability and scalability of SkVM on large-scale datasets with hundreds of million data points.

2 Sparkling Vector Machines

In this section, we first provide a brief review on how to integrate the data augmentation technique into SVMs. We then describe our extension to perform multiclass classification task. Furthermore, we present our parallelization procedure for such approach to construct our distributed framework.

2.1 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 $\mathcal{D} = \{(\mathbf{x}_n, \mathrm{y}_n)\}$ 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:

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

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.

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 [3]. The pseudo-likelihood is given as below:

$$p(\mathrm{y} \mid \mathbf{x}, \mathbf{w}) = \exp\left\{-2\max\left(1 - \mathrm{y}\mathbf{w}^\top \mathbf{x}, 0\right)\right\}$$

Minimizing the loss function in Eq. (1) now turns into estimating the maximum a posterior (MAP) of the following pseudo-posterior distribution:

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

in which $\mathcal{Z}_\alpha(C)$ is a pseudo-posterior normalization constant.

To integrate the data augmentation scheme into this model, an auxiliary variable $\lambda > 0$ is introduced for each observation label y [3], in a way that $p(\mathrm{y} \mid \mathbf{x}, \mathbf{w})$ becomes the marginal form of a joint distribution $p(\mathrm{y}, \lambda \mid \mathbf{x}, \mathbf{w})$. More specifically, the inverse λ^{-1} of such auxiliary variable is sampled from an *Inverse Gaussian* (IG) distribution:

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

Assuming that the prior distribution of parameter $\mathbf{w}$ is: $p(\mathbf{w}) \sim \mathcal{N}(\mu_0, \Sigma_0)$. The data pseudo-likelihood and the posterior conditional distribution of $\mathbf{w}$ reads:

$$\begin{aligned}p(\mathrm{y} \mid \mathbf{x}, \mathbf{w}, \lambda) &= \int_0^\infty \frac{1}{\sqrt{2\pi\lambda}} \exp\left\{-\frac{\left[\lambda + \left(1 - \mathbf{w}^\top \mathbf{x}\right)\right]}{2\lambda}\right\} \mathrm{d}\lambda \\ p(\mathbf{w} \mid \mathbf{x}, \mathrm{y}, \lambda) &= \mathcal{N}(\mu, \Sigma)\end{aligned} \tag{2}$$

where $\Sigma^{-1} = \mathbf{X}^\top \Lambda^{-1} \mathbf{X} + C\Sigma_0^{-1}$, $\mu = \Sigma \mathbf{X}^\top \left(\mathbf{1} + \lambda^{-1}\right)$ with $\Lambda = \mathrm{diag}(\lambda)$ and $\mathbf{1}$ denotes a vector of 1's. Here the data matrix is denoted by $\mathbf{X} = \left[\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{\mathrm{N}}\right]^\top$.

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 [3]. 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 [3]. In what follows, we introduce an upgrade of the presented framework for multiclass classification tasks, followed by a distributed version of the latent variable SVMs.

2.2 Multiclass latent SVMs

Here we extend the latent SVM schemes presented in Section 2.1 for multiclass classification. Suppose that there are K classes, the label is now $\mathrm{y}_n \in \{1, 2, .., \mathrm{K}\} \forall n = 1...\mathrm{N}$. 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:

$$\lambda_n^{-1} \sim \mathrm{IG}\left(\left|1 - \mathbf{w}_{\mathrm{y}_n}\mathbf{x}_n^\top\right|^{-1}, 1\right) \tag{3}$$

For the posterior of parameters $\mathbf{w}_k$, we can use MAP estimation: $\mathbf{w}_k = \mu_k = \Sigma_k \mathbf{X}^\top \mathbf{Z} + \mu_0 \Sigma_0$ where $\mathbf{Z} \in \{-1, 1\}^{\mathrm{N}\times\mathrm{K}}$ denotes the indicator matrix for the labels: $\mathrm{z}_{nk} = 1 \mid \forall n, y_i = k$, and $\mathrm{z}_{nk} = -1 \mid \forall n, y_n \neq k$. Assume that the prior distribution is $p\left(\mathbf{w} \mid \mu_0, \Sigma_0\right) \sim \mathcal{N}\left(0, \mathbf{I}\right)$. Let:

$$\mathbf{P} = \mathbf{X}^\top \mathrm{diag}\left(\boldsymbol{\lambda}\right)\mathbf{X} + \mathbf{I} \tag{4}$$

$$\mathbf{Q} = \mathbf{X}^\top \mathbf{Z} \tag{5}$$

From Eqs. (2,4,5), we have $\mathbf{P}\mathbf{w}_k = \mathbf{Q}_k$. Thus $\mathbf{w}_k = \mathbf{P}\backslash\mathbf{Q}_k$ in which 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 interative algorithm to alternatively sample the latent variable λ_n and compute $\mathbf{w}$. However, in practice we find that a single pass gains sufficiently good performance, thus we only perform sampling once. Once the parameters have been learned, the label $\hat{y}_{new}$ of new data instance $\mathbf{x}_{new}$ can be predicted as: $\hat{\mathrm{y}}_{new} = \underset{k}{\mathrm{argmax}}\left[\mathbf{w}_k^\top \mathbf{x}_{new}\right]$.

2.3 Distributed algorithm

Assuming that we are solving large-scale data classification problem, the number of data points is significantly greater than the number of features, i.e., $\mathrm{N} \gg \mathrm{D}$. Our computational bottlenecks are computing $\mathbf{P}$ and $\mathbf{Q}$ in Eqs. (4,5) with the computational complexity $\mathcal{O}\left(\mathrm{N} \times \mathrm{D}^2\right)$ and $\mathcal{O}\left(\mathrm{N} \times \mathrm{D} \times \mathrm{K}\right)$, respectively. These operations, however, can be parallelized in a distributed system as parallelizing the matrix multiplication. Thus we can bypass these issues using a distributed computing framework such as Spark.

More specifically, we first partition the data matrix $\mathbf{X}$ and the label $\mathbf{y}$ into disjoint M parts: $\mathbf{X} = \left[\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_\mathrm{M}\right]^\top$ and $\mathbf{y} = \left[\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_\mathrm{M}\right]^\top$. These partitions are then stored distributedly in multiple machines. For the m-th part, the auxiliary variables $\boldsymbol{\lambda}_m$ are sampled using Eq. (3) with $\mathbf{x}_n \in \mathbf{X}_m$ and $\mathrm{y}_n \in \mathbf{y}_m$. The part of these auxiliary variables resides 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: $\mathbf{P} = \sum_{m=1}^{\mathrm{M}} \mathbf{P}_m + \mathbf{I},\ \mathbf{Q} = \sum_{m=1}^{\mathrm{M}} \mathbf{Q}_m$ in which the m-th parts are computed as:

$$\mathbf{P}_m = \mathbf{X}_m^\top \mathrm{diag}\left(\boldsymbol{\lambda}_m\right)\mathbf{X}_m \qquad\qquad \mathbf{Q}_m = \mathbf{X}_m^\top \mathbf{Z}_m$$

It is clear that we only need the data part $\mathbf{X}_m$ and auxiliary variable part $\boldsymbol{\lambda}_m$ to compute $\mathbf{P}_m$ and $\mathbf{Q}_m$. 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 $\boldsymbol{\lambda}_m$, $\mathbf{P}_m$, $\mathbf{Q}_m$ 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 phrases of communications between driver and worker nodes. In the first phrase, 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 phrase.

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 that this operator is conducted on the driver node, thus not distributed. However, it performs on the feature dimension with the computational complexity $\mathcal{O}\left(\mathrm{D}^{2.376}\right)$ [6]. Thus this step will not become a bottleneck of our method. The pseudo-code is described in Alg. 1. We term our proposed method the *sparkling vector machines* (SkVM).

Algorithm 1 Learning algorithm of sparkling vector machine.

Input: $\mathbf{y} \in \mathbb{R}^{\mathrm{N}\times 1}$, $\mathbf{X} \in \mathbb{R}^{\mathrm{N}\times \mathrm{D}}$,$\mathbf{w}_k = \mathbf{1} : \forall k = 1, ..., \mathrm{K}$

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

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

3 Evaluations

In this section, we quantitatively evaluate our proposed method on both binary and multiclass classification tasks. Our main goal is to demonstrate the scalability of SkVM in fast training large-scale data. We use four big public datasets (i.e., Epsilon [7], Susy [8], MNIST8M [9] and Airlines dataset) in which the number training samples is much larger than the feature dimension (millions against hundreds). The first three datasets are obtained from UCI repository[1] and LIBSVM collection[2]. These datasets are readily published in the form of instance-feature matrices, and divided into training and testing subsets.

The airlines dataset is provided by American Statistical Association[3]. 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 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.

We use Python API[4] of Spark to implement our proposed method. 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 comparison, we implement a version of our proposed model in Matlab (denoted by Matlab-SkVM) and recruit 6 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); and random forest (Spark-RF) [10]. All Spark-based methods run on a Hadoop-Spark cluster with 8 worker nodes, each node equipped with 32 vcores CPU. The Matlab-SkVM runs on a single machine whose memory cannot fit the entire data, thus it alternatively loads and unloads smaller chunks of data.

Table 1 reports the classification performance on the testing set and execution time on the training set. 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 implementa-

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

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

[3] The data can be downloaded from http://stat-computing.org/dataexpo/2009/.

[4] http://spark.apache.org/docs/latest/api/python/

tion while keeping comparable predictive performances. Our method can achieve ten times speedup in training gigantic large amount of data.

Table 1: Comparison of classification performances with 7 baselines on four large-scale datasets. The accuracy is reported in percent (%), the execution time is reported in seconds. The higher accuracy and F1 are the better whilst the lower running time is the better. The best performance is in bold and the second-best is in italic.

Dataset	Epsilon		Susy		MNIST8M		Airlines	
Method	Accuracy	Time	Accuracy	Time	Accuracy	Time	F1	Time
Matlab-SkVM	**89.60**	194	*76.58*	*52*	84.88	765	0.23	9492
Spark-LR-SGD	50.55	157	55.37	265	–	–	0.27	544
Spark-LR-LBFGS	88.52	*147*	75.88	371	**88.82**	343	0.24	461
Spark-NB	–	–	66.57	221	81.60	*158*	0.14	*332*
Spark-LSVM	55.99	163	60.24	271	–	–	0.28	509
Spark-DT	66.10	177	**77.11**	244	62.40	202	0.26	755
Spark-RF	68.35	219	76.24	264	70.17	209	*0.31*	1455
SkVM	*89.58*	**84**	76.39	**36**	*86.82*	**122**	**0.34**	**314**

References

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 1

[2] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012. 1

[3] Nicholas G Polson, Steven L Scott, et al. Data augmentation for support vector machines. *Bayesian Analysis*, 6(1):1–23, 2011. 1, 2.1, 2.1

[4] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010. 1

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 1

[6] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012. 2.3

[7] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. An improved glmnet for l1-regularized logistic regression. *The Journal of Machine Learning Research*, 13(1):1999–2030, 2012. 3

[8] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5, 2014. 3

[9] Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, pages 301–320, 2007. 3

[10] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*, 2015. 3