# Budgeted Mini-Batch Parallel Gradient Descent for Support Vector Machines on Spark

Hang Tao
School of Computer Science,
Beijing University of Posts and Telecommunications,
Beijing 100876, China
bupt622@126.com

Bin Wu, Xiuqin Lin
Key Lab of Intelligent Telecommunication Software and Multimedia, Beijing 100876, China
wubin@bupt.edu.cn
xqlin@bupt.edu.cn

***Abstract*—Mini-batch gradient descent (MBGD) is an attractive choice for support vector machines (SVM), because processing part of examples at a time is advantageous when disposing large data. Similar to other SVM learning algorithms, MBGD is vulnerable to the curse of kernelization when equipped with kernel functions, which results in unbounded linear growth in model size and update time with data size. This paper presents a budgeted mini-batch parallel gradient descent algorithm (BMBPGD) for large-scale kernel SVM training which can run efficiently on Apache Spark. Spark is a fast and general engine for large-scale data processing which is originally intended to deal with iterative algorithms. BMBPGD algorithm has constant space and time complexity per update. It uses removal budget maintenance method to keep the number of support vectors (SVs). The experiment results show that BMBPGD achieves higher accuracy than SVMWithSGD algorithm in MLlib on Spark environment, and it takes much shorter time than LibSVM.**

***Keywords—mini-batch gradient descent; support vector machines; Spark; kernel method; large-scale learning; budget maintenance; stochastic gradient descent.***

## I. Introduction

Support vector machine (SVM) [2] [3] algorithm is a classical data mining algorithm, which is widely used in statistical classification and regression analysis. SVMs have advantages in high dimensional eigenvectors and small sample classification situations. However, with the constant development of Internet and the rapid increase of its users, the size of data rises significantly, and how to classify and predict these large-scale data rapidly and effectively becomes a serious problem.

Stochastic gradient descent (SGD) [1] [6] is a gradient descent optimization method to minimize an objective function, suitable for large-scale problems and supervised machine learning optimization problems such as SVM. SGD only uses one example in each iteration step, but MBGD (mini-batch gradient descent) uses part of examples in each iteration. For SGD and batch gradient descent (BGD), MBGD is a compromise algorithm. It reduces the amount of calculation for BGD, and the accuracy is higher than SGD.

For about 18 years of development, there are a lot of SVM's algorithms, for example, SMO (Platt, 1998), SVMlight (Joachim, 1998) [25], LS-SVM (Suykens, 1998) [18], SimpleSVM (S.V.N. Vishwanathan, 2003) [24], CascadeSVM (Hans Peter Graf et al., 2004) [15], CVM (Tsang et al., 2005) [13] and BVM (Ivor W.Tsang et al., 2007) [14]. Some of them are for kernel SVM training [12] of large data. However, these algorithms are not limited to the size of the model. Therefore, this limits their practical application to large-scale data. There are also large number of gradient-based methods for SVMs (Mangasarian, 2002 [16]; Zhang, 2004 [20]; Keerthi and DeCoste, 2005 [17]; Lin, 2007 [19]; Shalew- Shwartz, 2007 [21]; Bottou, 2007 [23]). Many methods here are only for linear situations of SVM, the area of large-scale kernel SVM training remains less explored.

Spark [8] is a fast and general environment for large scale data processing especially for iterative algorithms, developed by the University of California, Berkeley AMP Lab (Algorithms, Machines, and People Lab). For iterative situations, Spark can run programs much faster than Hadoop [26] MapReduce because of resilient distributed dataset (RDD) [22], which will be introduced in section II. SVMWithSGD is an algorithm in MLlib [9] that is Apache Spark's (Spark 0.9.1, April 09, 2014) scalable machine learning library.

According to these above reasons, this paper proposes a new method budgeted mini-batch parallel gradient descent (BMBPGD) on Spark to deal with large-scale datasets for kernel [4] SVM training quickly and efficiently. MBGD is vulnerable to the curse of kernelization with kernel functions, so that it is not suitable for big data. We use removal budget maintenance [5] [7] method to keep the number of support vectors (SVs), therefore BMBPGD algorithm has constant space and time complexity per update. Then, we compare our method with SVMWithSGD and LibSVM [10]. LibSVM is a classic and effective software for SVM.

This paper is organized as follows. We first introduce SVM, SGD, MBGD and Spark in section II. Section III proposes budget maintenance through removal and BMBPGD algorithm considering the non-linear situation for SVM on Spark. Section IV presents our experiment environment, as well as accuracy and time results of our algorithm compared with SVMWithSGD and LibSVM. Finally, section V summarizes our work.

## II. Preliminaries

In this section, we will introduce support vector machines, stochastic gradient descent, mini-batch gradient descent and Spark. SGD and MBGD are usually used to optimize SVM and reduce the time complexity. Spark is an efficient distributed computing system for large-scale data sets.

## *A. Support Vector Machines*

Support vector machine was put forward by Cortes and Vapnik in 1995. Generally speaking, SVM constructs a hyper-plane to make the largest distance to the nearest training data point of any class in a high- or infinite-dimensional space [11]. Standard SVM training algorithm can be attributed to solve QP problems.

The decisions function of SVM:

$$g\ (\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i\, y_i k(\boldsymbol{x}, \boldsymbol{x_i}) + b\right) \qquad (1)$$

Where the data is $\mathrm{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2) \dots (\boldsymbol{x}_n, y_n\}$, $x_i \epsilon R^n$ is a $n$-dimensional input vector, $y_i = \{\,0, 1\,\}$ is the label of class. $\alpha = \{\alpha_1, \alpha_i, \dots, \alpha_n\}, 0 \le \alpha_i \le C$ is the vector of Lagrange multipliers, and C is the parameter. $b$ is the bias or called offset. $k(\boldsymbol{x}, \boldsymbol{x_i})$ is the kernel function for non-linear data that will be described in section III. For linear datasets, $k(\boldsymbol{x}, \boldsymbol{x_i})$ is changed to $(\boldsymbol{x}, \boldsymbol{x_i})$.

Therefore, this problem can be transformed to solving the following optimization problem:

$$\min_{\alpha}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) - \sum_{i=1}^{n} \alpha_i \qquad (2)$$

$$s.t.\quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

## *B. Stochastic Gradient Descent*

Assuming that the data obey the probability distribution $\mathrm{P}(\boldsymbol{x}, y)$. Loss function $l(\hat{y}, y)$ describes the difference of the predicted label $\hat{y}$ and the actual label $y$. Predictive value is the result of $f_w(\boldsymbol{x})$ which is selected from function family $F$ with the parameter w. We try to find a function $f \in F$ to minimize the expected risk in the training data. Equation (3) is the average of loss function.

$$E(f) = \int l(f(\boldsymbol{x}), y)\mathrm{dP}\,(\boldsymbol{x}, y) \qquad (3)$$

Since we do not know the real distribution of the data, empirical risk is generally used to replace expected risk. Equation (4) is the empirical risk

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y_i) \qquad (4)$$

The empirical risk is minimized through the usage of the true gradient of the $\boldsymbol{w}$ vector which is estimated as the sum of the gradients arising from each individual training sample in standard gradient descent (GD).

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \eta\frac{1}{n}\sum_{i=1}^{n} \nabla_w l(f_t(\boldsymbol{x}_i), y_i) \qquad (5)$$

Note that $\eta$ is known as step size or update/gain factor to update the $\boldsymbol{w_t}$ at the step $t$. Standard GD needs the whole training data in order to calculate the gradient and update the optimization parameters. For large datasets, it is difficult to reach the global optimum for multiple iterations. In the field of machine learning, the problem is very common, like Perceptron, SVM and so on. Stochastic gradient descent (SGD) does not directly calculate the exact value of gradient. But, we use unbiased estimate of gradient to replace it. And SGD considers one sample during each iteration step.

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \frac{\eta}{t}\nabla_w l(f_t(\boldsymbol{x}_t), y_t) \qquad (6)$$

SGD is based on a single sample to update the weights, and get the approximate gradient descent search (take a random sample). SGD has faster convergence than GD. SGD just uses one sample at a time, so, updates of the optimization parameter $\boldsymbol{w_t}$ are noisy.

Here, we assume the bias $b = 0$. Training a linear SVM classifier $f(\boldsymbol{x}) = \boldsymbol{w^T x}$. Because $y_i = \{\,0, 1\,\}$, the hinge loss [27] function is defined as follows.

$$l(\boldsymbol{w};\ (\boldsymbol{x_t}, y_t)) = max(0, 1 - (2y_t - 1)\boldsymbol{w^T x}) \quad (7)$$

We need to solve the following optimization problem.

$$\frac{\lambda}{2}||\boldsymbol{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} l\big(\boldsymbol{w};\ (\boldsymbol{x}_t, y_t)\big) \qquad (8)$$

Note that $\lambda$ is a regularization parameter used to control model complexity, which should be greater than zero or equal to zero. The regularization term $||\boldsymbol{w}||$ is represented to the size of the margin in feature space. Put formula (8) into equation (6).

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \eta_t(\lambda \boldsymbol{w}_t + \nabla_w l(\boldsymbol{w};\ (x_t, y_t))) \qquad (9)$$

$$\boldsymbol{w_{t+1}} = (1 - \lambda\eta_t)\boldsymbol{w}_t + \beta_t \boldsymbol{x_t} \qquad (10)$$

Where, for the Norma algorithm (Kivinen et al., 2004), $\eta_t = \eta/\sqrt{t}$ .

$$\beta_t = \left\langle \begin{array}{c} (2y_t - 1)\eta_t, if\,(2y_t - 1)\boldsymbol{w}_t^T \boldsymbol{x} < 1 \\ 0, otherwise \end{array} \right. \qquad (11)$$

## *C. Mini-Batch Gradient Descent*

MBGD (mini-batch gradient descent) applies part of examples in each iteration step, but SGD only uses one example in every iteration step, BGD (batch gradient descent) uses the whole examples in each iteration step.

BGD is a method to cumulate the update of parameters and batch, but it is not suitable for large-scale data. However, SGD trains one example, and updates parameters one by one. MBGD updates the selected portion of the samples at the time of solving the direction, which reduces the amount of calculation and ensures the accuracy.

## *D. Spark*

Spark is a fast and general engine for large-scale data processing which is originally intended to deal with iterative algorithms, such as machine learning and graph processing algorithms, and interactive data mining algorithms. In these scenarios, Spark can run programs much faster than Hadoop MapReduce.

To use Spark, developers code a driver program that links to a cluster to run workers, as shown in Figure 1. The driver predefines one or more RDDs and calls actions on them.

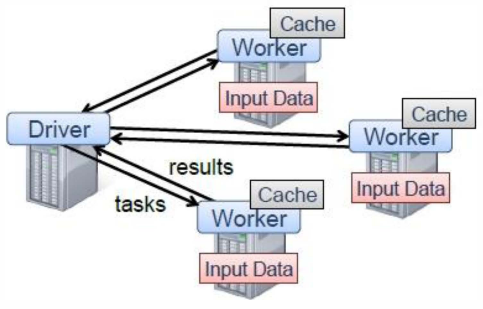


Figure 1. Spark runtime. Developers' driver program calls multiple workers, which read data from a distributed file system and can cache computed RDD partitions in memory.

Spark proposes a new storage mechanism named resilient distributed dataset, which allows the application to save the working sets in memory and cached for efficient reuse. RDD overcomes the problem that Hadoop needs to repeatedly read and write data from Hadoop distributed file system (HDFS) and takes plenty of time.

There are two methods of data for RDD: transformations (map, sample, filter, flatMap, join, union, groupByKey and so on) and actions (reduce, collect, count, save and so on). In our algorithm, we mainly use map, sample and reduce. Calling map on any RDD returns an already mapped RDD object that has same partitions and locations as its parent, but uses the function passed to map to the parent's records in its iterative method. Sample is similar to map, except that RDD stores a random number generator seed for each partition to deterministically sample parent records. Reduce belongs to the actions of data.

SVMWithSGD is an algorithm for linear SVM training in MLlib. This algorithm uses stochastic parallel gradient descent to realize linear SVM training on Spark, which is similar to MBPGD in section III (algorithm 1).

## III. Budgeted Mini-Batch Parallel Gradient Descent Algorithms

In this section, we will consider the linear and non-linear situations for SVM. On the basis of mini-batch parallel gradient descent for linear data, we propose a new method budgeted mini-batch parallel gradient descent (BMBPGD) algorithm for non-linear SVM.

### A. Mini-Batch Parallel Gradient Descent

For large datasets, we consider the parallel method of MBGD. The mini-batch parallel gradient descent algorithm is as follows. Algorithm 1 is the pseudo code of MBPGD. $\mu$ is the min-batch size used for choose the appropriate size of samples for each iteration. Each processor of Spark carries out mini-batch gradient descent on the set of data with a fixed step size $\eta$ for N iterations as described in Algorithm 1.

Algorithm 1 Mini-Batch parallel gradient descent (MBPGD)

```
Input: training data D, step size $\eta$, min-batch size $\mu$, number of iterations N, regularization parameter $\lambda$.
For all $i \in \{1,2,\dots,N\}$ parallel do
    Shuffle data on data
    Get the data $D_i * \mu$
    $\boldsymbol{w}_{t+1} = (1-\lambda\eta_t)\boldsymbol{w}_t + \beta_t \mathbf{x}_t$
end for
```

### B. Kernelization

Many algorithms are only for linear situation, but non-linear situations are more common. With the usage of Mercer kernel, MBGD for SVM can be used to solve the non-linear datasets. The training data are mapped from the original space into a high dimensional feature space, the mapping function is represented as $\phi$. So the nonlinear case is transformed into linear problem, $\boldsymbol{x}$ is replaced by $\phi(\boldsymbol{x})$. For $\boldsymbol{x},\boldsymbol{z} \in R^n$, Equation (12) is the kernel function.

$$k(\boldsymbol{x},\boldsymbol{z}) = \phi(\boldsymbol{x})\phi(\boldsymbol{z}) \quad (12)$$

$$w_t = \sum_{j=1}^{t} \alpha_j \phi(\boldsymbol{x}_j) \quad (13)$$

$$\alpha_j = \beta_j \prod_{m=j+1}^{t} (1-\eta_m\lambda) \quad (14)$$

$$f_t(\boldsymbol{x}) = \boldsymbol{w}_t\phi(\boldsymbol{x}) = \sum_{j \in I_t} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}) \quad (15)$$

The non-zero values $\alpha$ are also called support vectors. If the value of $\alpha$ is zero at time $t$, the hinge loss of the data is also zero. In the formula (15), $I_t$ is the set of indicators of all support vectors in $\boldsymbol{w}_t$. A weight vector $\boldsymbol{w}$ or an SV set $\{(\alpha_i, \boldsymbol{x}_i),\ i \in I_t\}$ can be used to describe the SVM classifier.

### C. Budgeted Mini-Batch Parallel Gradient Descent

Similar to other SVM learning algorithms, MBGD is vulnerable to the curse of kernelization when equipped with kernel functions, which results in unbounded linear growth in model size and update time with data size. This makes SGD cannot be applied to large data sets. This paper proposes a BMBPGD for large-scale kernel SVM training, and the method has constant space and constant time complexity per update. Now we introduce our algorithm where the data are distributed over distinct processors on Spark.

The algorithm reduces the size of $I_{t+1}$ by one, so that $w_{t+1}$ is only spanned by B SVs. This maybe leads to the degradation of the SVM classifier. Therefore, we define a budget B, if the number of SVs is larger than $B$, BMBPGD implements a budgeted maintenance step to make sure the number of SVs.

Algorithm 2 Budgeted mini-batch parallel gradient descent (BMBPGD)

```
Input: training data D, step size $\eta$, min-batch size $\mu$, number of iterations N, regularization parameter $\lambda$, budget $B$, kernel k.
For all $i \in \{1,2,\dots,N\}$ parallel do
    Shuffle data on Spark using map and sample.
    Get the data $D_i * \mu$
    $w_{t+1} = (1 - \lambda\eta_t)w_t + \beta_t\phi(x)$
    $a = a + 1$ //SV number
    if $a > B$ then
        $w_{t+1} = w_{t+1} - \Delta_t$
        $a = a - 1$
    end if
end for
```

We propose a generic BMBPGD algorithm for SVM in algorithm 2. That is the pseudo code of BMBPGD. Here, $\Delta_t$ is the weight degradation caused by budget maintenance at $\eta$-th round, which is considered as the difference between model weights before and after budget maintenance. $\Delta_t$ can effectively improve the classification results of SVMs. We get the minimum $\Delta_t$ through greedy budget maintenance method based on removal of SVs. In section D, we will specifically explain the principle of budget maintenance through removal.

### *D. Budgeted Maintenance through Removal*

Define the gradient error $E_t = \Delta_t/\eta_t$, and hypothesis that $||E_t|| \le 1$. Define the average gradient error in equation (16).

$$\bar{E} = \frac{1}{n}\sum_{t=1}^{n}||E_t|| \qquad (16)$$

Budget maintenance for BMBPGD plays a pivotal role. So we should try best to minimize the averaged gradient error $\bar{E}$. To achieve that gain, a greedy procedure is proposed to minimize the gradient error $||E_t||$ at each iteration step, which is equivalent to minimize the weight degradation $||\Delta_t||$. We can get the minimum budget maintenance with the method of removing of SVs.

The method of removal budget maintenance assumes that budget maintenance removes the $p$-th SV for binary class SVM, so that

$$\Delta_t = \alpha_p\phi(\boldsymbol{x_p}) \qquad (17)$$

Note that $\alpha_p = [\alpha_p{}^{(1)}, \alpha_p{}^{(2)}]$ has binary categories of specific factors of $j$-th SV. To minimize the weight degradation $||\Delta_t||$ for the removal method, we just need to delete SV with the smallest norm,

$$p = \arg\min_{j \in I_{t+1}} \|\alpha_j\|^2 k(\boldsymbol{x_j}, \boldsymbol{x_j}) \qquad (18)$$

Gaussian Radial Basis Function (RBF) is frequently used as the kernel function, and it is also the translation invariant kernel $k(\boldsymbol{x},\boldsymbol{x}') = \tilde{k}(\boldsymbol{x}-\boldsymbol{x}')$. Assuming there is no loss of generality, and $k(\boldsymbol{x},\boldsymbol{x}') = 1$. Therefore, to minimize the weight degradation $||\Delta_t||$ is equivalent to minimize the $\|\alpha_p\|$. That is, we should remove SVs with the smallest $\|\alpha_p\|$.

## IV. Experiments

In this section, we evaluate BMBPGD algorithm. The experiments mainly focus on the comparison between BMBPGD and SVMWithSVM on Spark.

In this paper, we use Apache Spark to do the experiment. All experiments are implemented on the computers, each one with an Intel Xeon 2.00GHz CPU and 8Gbytes memory. BMBPGD and SVMWithSGD are executed on five computers, including 1 master and 4 workers. SVMWithSGD algorithm is from MLlib on Apache Spark. It is stochastic parallel gradient descent method for SVMs on Spark.

We use three datasets: a9a, w8a, covtype for all of the experiments. Table I shows the size parameters for three datasets, and the source of datasets. The testing data is the same amount of the training data. Table II displays the parameters of BMBPGD algorithm which are used in the whole experiments. In this algorithm, we use L1 regularization.

TABLE I. Sizes parameters for three datasets used in experiments

| Dataset | a9a | w8a | Covtype |
|---|---|---|---|
| Data source | UCI / Adult | JP98a | UCI / Covertype |
| Number of data | 32,561 | 49,749 | 581,012 (extend to 500× $10^4$) |
| Number of feature | 123 | 300 | 54 |

TABLE II. Parameters of BMBPGD

| | |
|---|---|
| **step size** | $\eta = 1$ |
| **min-batch size** | $\mu = 1$ |
| **number of iterations** | N = 200 |
| **regularization parameter** | L1regularization, $\lambda = 0.01$ |
| **budget** | $B = 100$ |

TABLE III. The accuracy and time of different algorithms, and iterative number N = 20

| | | |
|---|---|---|
| | **a9a** | |
| | **accuracy** | **time (s)** |
| **BMBPGD** | 83.57% | 32 |
| **SVMWithSGD** | 82.83% | 30 |
| **LibSVM** | 84.29% | 130 |
| | **w8a** | |
| **BMBPGD** | 97.63% | 50 |
| **SVMWithSGD** | 97.03% | 49 |
| **LibSVM** | 97.58% | 46 |
| | **covtype** | |
| **BMBPGD** | 63.58% | 99 |
| **SVMWithSGD** | 61.18% | 96 |
| **LibSVM** | 77.12% | 54320 |

We measure performance mainly in two parameters, accuracy and the running time. First, we compare BMBPGD (our parallel algorithm), SVMWithSGD (the algorithm in MLlib) and LibSVM.

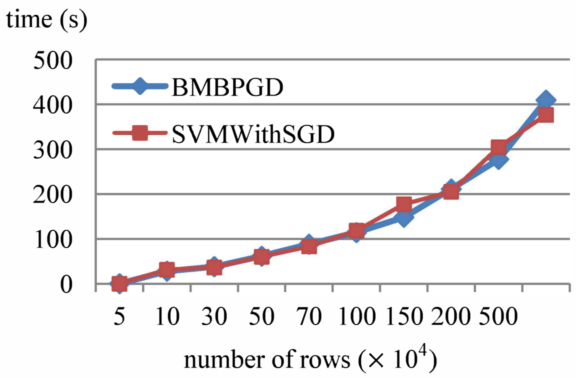
Table III and Table IV shows the accuracy and time of different algorithms, when iterative time are $N = 20$, $N = 200$, respectively. In Table III, if the iterative number is 20, the accuracy of BMBPGD is always higher than SVMWithSGD, and its time is a little bit longer than SVMWithSGD because budget maintenance step takes some time. These two algorithms are parallelized running on Spark. Therefore the running time of BMBPGD and SVMWithSGD is much shorter than LibSVM with small iterative number, but sometimes the accuracy of classification is likely to have a decrease to LibSVM.

In Table IV, when the number of iteration is 200, the accuracy is much higher than the situation $N = 20$, as well as accuracy of BMBPGD is higher than SVMWithSGD at the same iterative number. For the data of covtype we can see that the accuracy of our algorithm is about 10% higher than SVMWithSGD. For a9a and w8a, the accuracy of BMBPGD is higher than LibSVM.

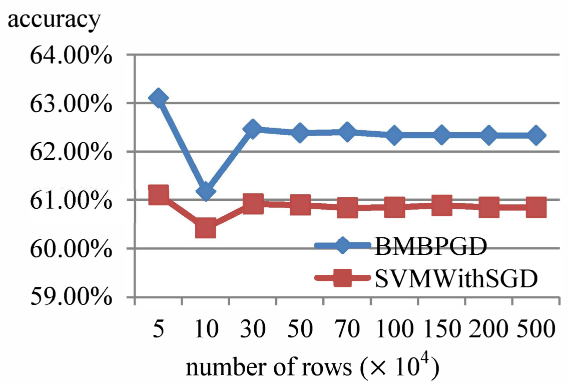
TABLE IV. THE ACCURACY AND TIME OF DIFFERENT ALGORITHMS, AND ITERATIVE NUMBER $N = 200$

| | **a9a** | |
|---|---|---|
| | **accuracy** | **time (s)** |
| **BMBPGD** | 84.43% | 146 |
| **SVMWithSGD** | 83.81% | 140 |
| **LibSVM** | 84.29% | 130 |
| | **w8a** | |
| **BMBPGD** | 97.82% | 322 |
| **SVMWithSGD** | 97.03% | 318 |
| **LibSVM** | 97.58% | 46 |
| | **covtype** | |
| **BMBPGD** | 72.13% | 696 |
| **SVMWithSGD** | 62.86% | 705 |
| **LibSVM** | 77.12% | 54320 |

Figure 2 (a) and (b) show the time and accuracy for different number of lines on $N = 20$ (covtype data), respectively. The largest data (the number of rows $500 \times 10^4$ and number of feature 54) here is nearly 1 GB (gigabyte). For constant iterative number, with the increase of the number of data, the running time of SMBPGD and SVMWithSGD grows nearly linearly. For different sizes of the same data, the accuracy of BMBPGD is almost higher than SVMWithSGD. When the data size is big enough, the accuracy remains same.



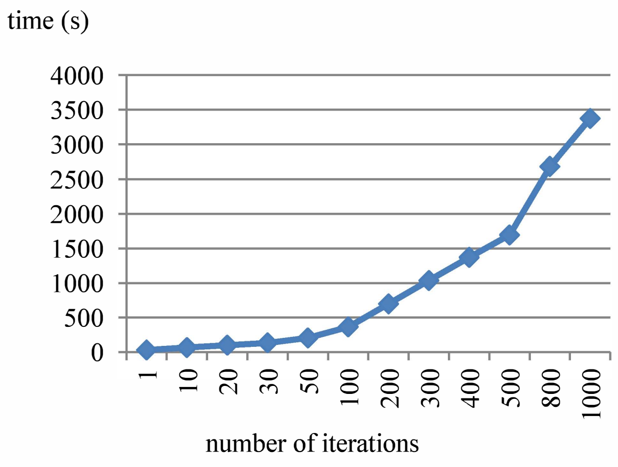
(a)



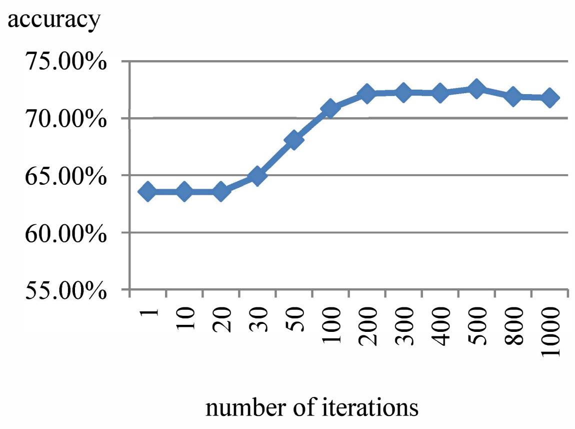
(b)

Figure 2. (a) time (b) accuracy for different number of rows on $N = 20$ (covtype data)

The time and accuracy for different number of iterations (covtype data) are respectively illustrated in Figure 3 (a) and (b). With the increase of the number of iterations, the time of SMBPGD rises. When the number of iterations is less than 200, the accuracy of BMBPGD continuously grows when the increase of the number of iterations. When the iterative number is 500, the accuracy is highest 72.57%. From Figure2, $N = 200$ is a compromise choice, both for the relatively high accuracy and less time of the BMBPGD algorithm.



(a)



(b)

Figure 3. (a)time (b) accuracy of BMBPGD for different number of iterations (covtype data)

According to these experiments, we think that our algorithm is useful, fast and efficient for large-scale kernel SVM training on Spark. The accuracy of BMBPGD is higher than SVMWithSGD, when selected with the appropriate parameters and number of iterations. Our algorithm uses removal budget maintenance method to keep the number of support vectors (SVs), which solves the problem of curse of kernelization. Therefore, BMBPGD algorithm has constant space and time complexity per update.

## V. CONCLUSIONS

In this paper, we put forward a framework for large-scale kernel SVM by using BMBPGD algorithm on Spark. We got theoretical bounds on the performance. It indicated that the accuracy of our algorithm is closely related to the model degradation because of the budget maintenance. So that, we proposed budget maintenance strategies based on removal of SVs. Through the experiment on Spark, we evaluated the MBMPGD algorithm from accuracy and time on different number of iterations and data size, and compared it with SVMWithSGD, as well as the classic serial algorithm LibSVM. The results of our experiments show that our algorithm has higher accuracy than SVMWithSGD on Spark distributed environment, and it achieves impressive speedup relative to LibSVM for very large-scale datasets. To some extent, the larger number of iterations, the relatively higher accuracy of BMBPGD, but it is at the cost of spending more time. Therefore, we chose $N = 200$ as a compromise choice that ensures the accuracy of BMBPGD, and it takes relatively short time. In the future, we will continue to focus on distributed computing systems (Hadoop and Spark), as well as optimize our algorithm, expand the size of training data (hundreds of GB and TB).

## ACKNOWLEDGEMENT

This work is supported by the National Key Basic Research and Department (973) of China (No.2013CB329603), the National Science Foundation of China (Nos.61375058, and 71231002), the Ministry of Education of China Mobile Research Fund (MCM20130351) and the Special Co-construction Project of Beijing Municipal Commission of Education.

## REFERENCES

[1] Rob G.J. Wijnhoven, Peter H.N. de With, “Fast Training of Object Detection using Stochastic Gradient Descent,” International Conference on Pattern Recognition, 2010, DOI 10.1109/ICPR.2010.112.

[2] Aditya Krishna Menon, “Large-Scale Support Vector Machines: Algorithms and Theory,” Research Exam, Universi-ty of Califo rnia, San Diego. 2009.

[3] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, “A Dual Coordinate Descent Method for Large-scale Linear SVM,” ICML, 2008.

[4] Te-Ming Huang, Vojislav Kecman,Ivica Kopriva, “Kernel Based Algorithms for Mining Huge Data Set,” Springer Verlag, 2006.

[5] Zhuang Wang, Koby Crammer, Slobodan Vucetic, “Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training,” Journal of Machine Learning Research 13, 2012, s3103-3131.

[6] Martin A. Zinkevich, Markus Weimer, Alex Smola, Lihong Li, “Parallelized Stochastic Gradient Descent,” Advances in Neural Information Processing Systems 23 (NIPS-10), 2011, 2595–2603.

[7] N. Cesa-Bianchi, C. Gentile, “Tracking the best hyperplane with a simple budget perceptron,” In Annual Conference on Learning Theory, 2006.

[8] Apache spark - lightning-fast cluster computing. URL https://spark.apache.org.

[9] Machine Learning library (MLlib) in Spark. URL http://spark.apache.org/mllib/

[10] LIBSVM - A Library for Support Vector Machines. ULR http:// www.csie.ntu.edu.tw/~cjlin/libsvm/

[11] Jian-Pei Zhang, Zhong-Wei Li, Jing Yang, “Parallel SVM Training Algorithm on Large-scale Classification Problems,” Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 2005.

[12] Christianini, J. Shawe-Layer, “An Introduction to Support Vector Machines and Other Kernel-based Learning Methods,” Cambridge University Press. 2000. Page93-122.

[13] Ivor W.Tsang, James T.Kwok, and Pak-Ming Cheung, “Core Vector Machines: Fast Training on Very Large Data Sets,” Journal of Machine Learning Research, 6(4), 2005:363-292.

[14] Ivor W.Tsang, Andras Kocsor, James T.Kwok, “Simpler Core Vector Machines with Enclosing Balls,” Proceeding of the Twenty-Fourth International Conference on Machine Learning(ICML), Corvallis, Oregon, USA, June 2007:911-918.

[15] Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, “Vladimir Vapnik. Parallel Support Vector Machines: The Cascade SVM,” Processing of NIPS.2004.196-212.

[16] Olvi L. Mangasarian, “A finite Newton method for classification,” Optimization Methods and Software, 17(5), 2002: 913-929.

[17] S.Sathiya Keerithi, Dennis DeCoste, “A modified finite Newton Method for Fast Solution of Large Scale Linear SVMs,” Journal of Machine Learning Research, 6, 2005:341-361.

[18] Suykens J., Vandewalle J., “Least Square Support Vector Machine Classifiers,” Neural Processing Letters, 1999, 9(3): 29-300P

[19] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi, “Trust Region Newton Method for Large-scale LogisticRegressio,” Journal of Machine Learning Research, 9, 2007: 627-650.

[20] Tong Zhang, “Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms,” In Proceedings of the 21th International Conference on Machine Learning (ICML), 2004.

[21] Shai Shalev-Shwartz, Yoram Singer, and SNathan Srebro, “Pegasos: Primal Estimated sub-Gradient Solver for SVM,” In Proceedings of the 24th International Conference on Machine Learning (ICML), 2007.

[22] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, Ion Stoica, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,” Technical Report No. UCB/EECS-2011-82. 2011.

[23] Leon Bottou. Stochastic Gradient Descent Examples. URL http://leon.bottou.org/projects/sgd, 2007.

[24] S.V.N. Vishwanathan, Alexander J. Smola, M. Narasimha Murty, “SimpleSVM,” Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

[25] Thorsten Joachims. SVM-Light Support Vector Machine. URL http://www.cs.cornell.edu/People/tj/svm_light/

[26] Hadoop. URL http://hadoop.apache.org/.

[27] Robert C. Moore, John DeNero, “L1 and L2 Regularizati-on for Multiclass Hinge Loss Models,” Symposium on Machine Learni ng in Speech and Natural Language Process-ing, 2011.