Speeding Up Online Training of L1 Support Vector Machines

Gabriella Melki
Computer Science Department
Virginia Commonwealth University
Richmond, VA, USA
melkiga@vcu.edu

Vojislav Kecman
Computer Science Department
Virginia Commonwealth University
Richmond, VA, USA
vkecman@vcu.edu

***Abstract*—Paper proposes a novel experimental environment for solving a classic nonlinear Soft Margin L1 Support Vector Machine (SVM) problem using a Stochastic Gradient Descent (SGD) algorithm. Our implementation has a unique method of random sampling and alpha calculations. The developed code produces a competitive accuracy as well as very fast training of SVMs (small CPU time). The SGD model's performance is compared to the solutions of the L2 SVM obtained by software for Minimal Norm (MN-SVM) and Non-Negative Iterative Single Data Algorithm (NN-ISDA). The latter two algorithms have shown excellent performances on large datasets; which is why we chose to have our implementation of the SGD algorithm compete with them. All experiments have been done under strict double (nested) cross-validation, and the results are reported in terms of accuracy and CPU times used by the three methods.**

Keywords—support vector machines, classification, large scale learning, stochastic gradient descent, soft-margin support vector machines

I. Introduction

Over the past decade, dataset sizes have grown faster and disproportionately to the speed of processors and memory capacity. With this prominent increase of large-scale data, there is a demand for new machine learning algorithms that are able to process these data to provide insightful information while completing these tasks quickly and feasibly (because of memory constraints).

SVMs represent a set of popular supervised, linear and nonlinear, machine learning algorithms having theoretical foundations based on Vapnik-Chervonenkis theory [1-2]. SVM models have similarities to other machine learning techniques, but research has shown that they usually outperform them in terms of computational efficiency, scalability, robustness against outliers, and most importantly they are well suited for ultra-large data sets [2-4, 6], making them a very useful data mining tool for diverse real-world applications. A traditional approach to training SVMs is the *Sequential Minimal Optimization* (*SMO*) technique [5], which is an efficient approach for solving Quadratic Programming tasks while training the L1 SVM. More recent and efficient approaches include the *Iterative Single Data Algorithm* (*ISDA*) [2-3]. It has been shown that *ISDA,* applied as the L2 SVM solver, is faster than the *SMO* algorithm, and equal in terms of accuracy [7]. Here, we will show that the *SGD* approach is faster than both *MN-SVM* and *NN-ISDA*. *MN-SVM* belongs to the class of geometric approaches for solving an SVM training task and was proposed to remedy the speed of learning issues [6]. However, the newest algebraic solution of the L2 SVMs by *NN-ISDA* seems to be the fastest approach for dealing with large datasets [3-4, 8].

More recently, there has been more investigation into stochastic, also known as online, algorithms to optimize large-scale learning problems. In [9] and [10] it has been shown that stochastic algorithms can be both the fastest, and have the best generalization performances. Bottou [10] and Shalev-Shwartz et al. [11-12] demonstrate that the basic *Stochastic Gradient Descent* (SGD) algorithm is very effective when the data is sparse, taking less than linear [$O(d)$] time and space per iteration to optimize a system with d parameters. It can greatly surpass the performance of more sophisticated batch methods on large data sets. It is also shown in our companion paper [18], that online algorithms' performances surpass that of the *SMO* algorithm, within the MATLAB platform. Other variants include *SGD for minimizing λ-strongly convex functions* [12], which works well for SVMs.

This paper shows the performances of the SGD algorithm as proposed in [12] while varying the parameters of the experiment. Unlike in [12], where the sample to be updated is selected randomly and uniformly within the main training algorithm, we shuffle the data and select the samples cyclically. We have also experimented with three different ways of calculating the final values of dual variables α_i, as well as different configurations for looping through the data during training. The paper is organized as follows. We first present the Soft-Margin L1 SVM problem followed by the Soft-SVM SGD algorithm in [12], with and without using kernels. We then explain our experimental environment; show our results and how they compare with different models. Our experiments were done with nested cross-validation, ensuring reliable model comparisons. Finally, we present the conclusions and discuss our future works.

II. Regularized Loss Minimization Using SGD

The Soft Margin SVM optimization problem associated with regularized hinge loss minimization minimizes the following cost function, which is λ-strongly convex.

$$\min_w \left(\frac{\lambda}{2}\|w\|^2 + L_s(w)\right), \quad (1)$$

978-1-5090-2246-5/16/$31.00 ©2016 IEEE

where w represents the weight vector, and $L_s(w)$ is the hinge loss function. This requires that the norms of the optimal **w** will at most be λ. [12] claims that if a function f is λ-strongly convex for every vector **w**, **u**, and **v** ϵ δf (**w**) then we have $\langle w-u, v\rangle \geq f(w) - f(u) + \frac{\lambda}{2}\|w-u\|^2$ implying that the norms will never be greater than λ. It is then shown that an unbiased estimate of the function's subgradient can be chosen. The update rule for these functions can then be rewritten as

$$w^{(t+1)} = -\frac{1}{\lambda t}\sum_{i=1}^{t} v_i, \quad (2)$$

where t is the number of iterations. This shows, by using SGD, the regularized loss function can directly be minimized. It is done by sampling a point i.i.d from a distribution, then using a subgradient of the loss function at this point as an unbiased estimate of the subgradient of the risk function.

A. Soft-SVM without Kernels

The linear Soft-SVM without kernels is presented as follows,

$$\min_{w,b,\xi}\left(\frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i\right) \quad (3)$$
$$s.t.\, y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, \lambda > 0, \qquad \forall i \in \{1, \ldots, m\}$$

It jointly minimizes the norm of **w** (or maximizes the margin of separability between classes), and the average of ξ_i (corresponding to the violations of the constraints). Here, ξ_i can be rewritten as the averaged hinge loss function

$$L_S^{hinge}(w) = \frac{1}{m}\sum_{i=1}^{m} max\{0, 1 - y_i(\langle w, x_i\rangle + b)\}, \quad (4)$$

where w represents the SVM weight vector, b is the bias term, and x is the input vector belonging to a training set S that contains m samples.

Using the SGD approach for minimizing a regularized loss function (3), one obtains the Algorithm A given below

ALGORITHM A

SGD for Solving Soft-SVM

Goal: Solve Equation (3)
Parameter: T
Initialize: $\boldsymbol{\theta}^{(1)} = 0$
for $t = 1, \ldots, T$
  Let $\boldsymbol{w}^{(t)} = \frac{1}{\lambda m}\boldsymbol{\theta}^{(t)}$
  Choose i uniformly at random from $[m]$
  If $(y_i\langle \boldsymbol{w}(t)), \boldsymbol{x}_i\rangle < 1)$
    Set $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + y_i\boldsymbol{x}_i$
  Else
    Set $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$
Output: $\boldsymbol{w}^* = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{w}^{(t)}$

B. Soft-SVM with Kernels

While the Algorithm A is simple, it only works for linear SVMs. For a general nonlinear SVM (NL-SVM) model, with use of kernels, there is also a simple method for solving the Soft-SVM task by using a stochastic gradient approach. (The presentation below follows [12]). For NL-SVM, one wants to minimize the regularized soft margin loss function

$$\min_{w}\left(\frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m}\max\{0, 1 - y_i\langle w, \psi(x_i)\rangle\}\right) \quad (5)$$

The vector $\mathbf{w}^{(t)}$ that is updated with Algorithm A is always in the linear span of $\{\psi(x_1), \ldots, \psi(x_m)\}$, which means that the corresponding coefficient $\boldsymbol{\alpha}$ can be maintained. In this new SGD procedure, [12] maintains two vectors $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{\alpha}^{(t)}$ such that

$$\theta^{(t)} = \sum_{j=1}^{m}\beta_j^{(t)}\psi(x_j), \quad (6)$$
$$w^{(t)} = \sum_{j=1}^{m}\alpha_j^{(t)}\psi(x_j), \quad (7)$$

They are updated by the following procedure.

ALGORITHM B

SGD for Solving Soft-SVM with Kernels

Goal: Solve Equation (5)
Parameter: T
Initialize: $\boldsymbol{\beta}^{(1)} = 0$
for $t = 1, \ldots, T$
  Let $\boldsymbol{\alpha}^{(t)} = \frac{1}{\lambda m}\boldsymbol{\beta}^{(t)}$
  Choose i uniformly at random from $[m]$
  For all $j \neq i$ set $\beta_j^{(t+1)} = \beta_j^{(t)}$
  If $(y_i\sum_{j=1}^{m}\alpha_j^{(t)}K(x_j, x_i) < 1)$
    Set $\beta_j^{(t+1)} = \beta_j^{(t)} + y_i$
  Else
    Set $\beta_j^{(t+1)} = \beta_j^{(t)}$
Output: $\boldsymbol{w}^* = \sum_{j=1}^{m}\alpha_j^*\psi(x_j)$ where $\boldsymbol{\alpha}^* = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\alpha}^{(t)}$

III. Our Implementation and Contribution

We have implemented Algorithm B within the GSVM framework [15] with our modifications explained below, named *NL*-SGD, and we defined the experiment as mentioned in the introduction. Namely, after shuffling the data, the updates of the dual variables $\boldsymbol{\alpha_i}$ are performed cyclically. This improves the computational complexity of the method by excluding the random sampling from the algorithm's main loop, as well as ensures that each data point gets updated at least once. As for the calculation of the final dual variables, $\boldsymbol{\alpha_i}$, we use three different approaches. The first is averaging over the last 50% of the updates, the second is taking the mean value of the dual variables over the last 25% of updates, and lastly we used only the last value of $\boldsymbol{\alpha}$.

Finally, we conducted some preliminary experiments for clarifying issues associated with stopping the training process. We experimented with using 2 and 5 epochs for small datasets, and 1 and 2 epochs for medium datasets. We also set the initial number of iterations $\boldsymbol{T}$ to be the number of samples in the dataset. When an epoch greater than one is used, we set $\boldsymbol{T}$ to be the number of samples multiplied by the number of epochs.

The results in the below section under the name *NL-SGD*, are obtained using the procedure described in the above paragraphs.

IV. Data, Experiments, and Results

To evaluate our experimental procedure, and the SGD algorithm, as well as compare its performance with the *MN-SVM* and *NN-ISDA* algorithms, we used benchmark datasets from both the UCI Machine Learning Repository [13] and the LIBSVM [14] sites. Table I lists all the datasets used to test our model. For each dataset, the table lists the number of inputs, the number of features (dimensionality), and the number of classes. Note that these datasets are the same as the datasets used in [6] and [7]. In [6] *MN-SVM* was compared to the *BVM* implementation taken from the LibCVM package [16], and in [7] the *NN-ISDA* algorithm was then compared to the results obtained by *MN-SVM* in [6].

TABLE I. Dataset Information

Dataset	# Instances	# Features	# Classes
Small Datasets			
Iris	150	4	3
Glass	214	9	6
Wine	178	13	3
Teach	151	5	3
Sonar	208	60	2
Dermatology	366	33	6
Heart	270	13	2
Prokaryotic	997	20	3
Eukaryotic	2427	20	4
Medium Datasets			
Optdigits	5620	64	10
Usps	9298	256	10
Reuters	11069	8315	2

MN-SVM was implemented in an open source framework called "GSVM – Command Line Tool for Geometric SVM Training" [15] and showed considerable training time speedup in comparison to the L1 and L2 SVM methods from LIBSVM [17], as well as *BVM* from LibCVM. These comparisons are shown in [6]. It has been then shown in [7] that the *NN-ISDA* algorithm is faster than the *MN-SVM*. This is why we compare our implementation of the SGD algorithm with these two approaches. Note that all the three algorithms are implemented within the general framework of GSVM [15].

A. Experimental Environment

The comparison of these models was obtained using a strict nested (a.k.a. double) cross-validation procedure. This experimental environment is computationally expensive, but it ensures that the models' performances are accurately assessed. In the outer loop, the data is separated into equally sized sections; in our experiments we have chosen to use 5 sections. Each part is held out in turn as the test set, and the remaining four parts are used as the training set. In the inner loop, 5-fold cross-validation is used over the training set, where the best hyper-parameters are chosen. The best model obtained by the inner loop is then applied on the outer loop's test set. This procedure ensures that the model's performance is not optimistically biased as what would have occurred using single loop *k*-fold cross-validation.

The preprocessing steps used for our data sets were feature normalization and data shuffling. Our features were rescaled to values between [0, 1], and we shuffled the data to ensure that our method is stochastic when looping through each data point.

The tuning of our model during cross-validation consists of finding the best penalty parameter C, or $\frac{1}{\lambda}$, as well as the best shape parameter γ for the Gaussian RBF kernel. The parameters were selected among 8*x*8 possible combinations, from the below values, which have also been used in [6-7].

$$C \in \{4^n\}, n = -2, -1, \dots, 5, \quad (8.1)$$

$$\gamma \in \{4^n\}, n = -5, -4, \dots, 2, \quad (8.2)$$

To deal with multi-class classification problems, we used the one-vs-one, or pairwise, approach. The pairwise training procedure trains $\frac{c(c-1)}{2}$ binary classifiers, a classifier for each possible pair of classes, where c is the number of classes. During the prediction phase, a voting scheme is used where all $\frac{c(c-1)}{2}$ models predict an unseen data sample and the class that received the highest number of votes is considered to be the sample's true class.

TABLE II. Accuracies Achieved

Dataset	MN-SVM	NN-ISDA	NL-SGD
Small Datasets			
Iris	**96.67**	94.00	96.00
Glass	**69.29**	67.82	69.17
Wine	96.60	96.60	**97.17**
Teach	52.95	52.31	**56.95**
Sonar	87.57	89.48	**89.49**
Dermatology	**98.36**	**98.36**	97.38
Heart	83.33	83.33	**84.81**
Prokaryotic	**88.97**	88.65	87.86
Eukaryotic	**81.21**	79.56	71.36
Average Accuracy	**83.88**	83.34	83.35
Medium Datasets			
Optdigits	**99.31**	99.29	98.88
Usps	**98.21**	98.05	96.93
Reuters	98.05	**98.08**	97.28
Average Accuracy	**98.52**	98.47	97.69

B. Comparison of NL-SGD with. NN-ISDA and MN-SVM

Table II shows the accuracies achieved by our implementation, along with the accuracies achieved by *MN-SVM* [6,15] and *NN-ISDA* [3,7]. The results show that the *NL-SGD* algorithm has similar accuracy to its competitors.

Figures 1 and 2 show the accuracy comparisons of the three algorithms for the small and medium datasets. In all cases, *NL-SGD* performs competitively, or even surpasses the accuracy of *NN-ISDA* and *MN-SVM*.

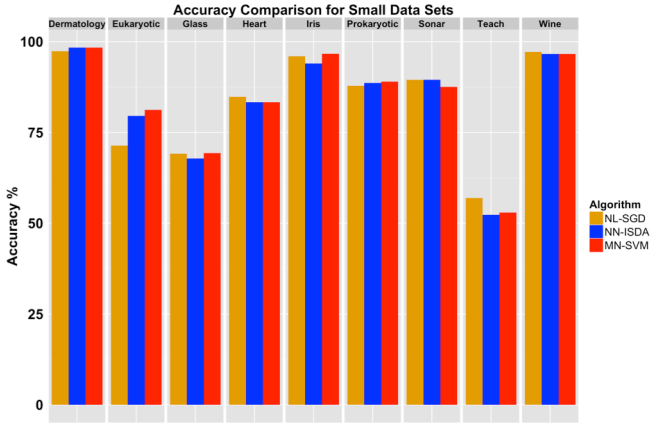

Fig 1. Accuracy on Small Datasets

Table III compares the three algorithms' CPU training and shows that the *NL-SGD* implementation is the choice for learning from datasets. In all cases, except the "Wine" dataset, the *NL-SGD* implementation is fastest. Based on the average CPU time, it is 1.2 times faster than *NN-ISDA* and 8 times faster than *MN-SVM* for training small datasets. For medium datasets, *NL-SGD* is 1.35 times faster than *NN-ISDA* and 1.92 times faster than *MN-SVM*. This, combined with *NL-SGD*'s competitive accuracy, shows that it is definitely suitable for training datasets of any size, especially large datasets.

The main motivation for the experiments presented in this paper is the desire to examine the speed of learning that can be achieved by using the *NL-SGD* algorithm on datasets of increasing size. In that respect, Figs. 3, 4, 5, and Table III show exactly that.

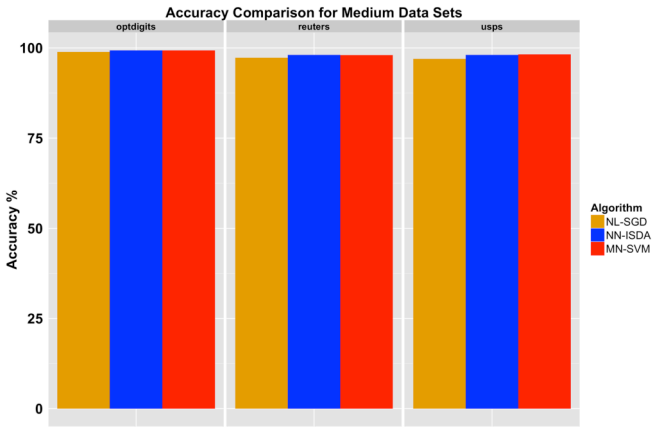

Fig 2. Accuracy on Medium Datasets

TABLE III. CPU TIME NEEDED FOR TRAINING

Dataset	**GSVM**	**NN-ISDA**	**NL-SGD**
Small Datasets			
Iris	3.57	0.27	**0.18**
Glass	11.94	1.01	**0.5**
Wine	4.84	**0.43**	0.47
Teach	8.85	0.44	**0.30**
Sonar	3.03	0.98	**0.74**
Dermatology	11.68	2.47	**1.69**
Heart	6.45	0.91	**0.61**
Prokaryotic	50.86	10.64	**5.61**
Eukaryotic	342.76	49.16	**45.38**
Average CPU time	49.33	7.37	**6.16**
Medium Datasets			
Optdigits	787.85	528.04	**340.70**
Usps	7777.82	5245.55	**4193.36**
Reuters	1657.04	1368.02	**758.05**
Average CPU time	3407.57	2380.54	**1764.04**

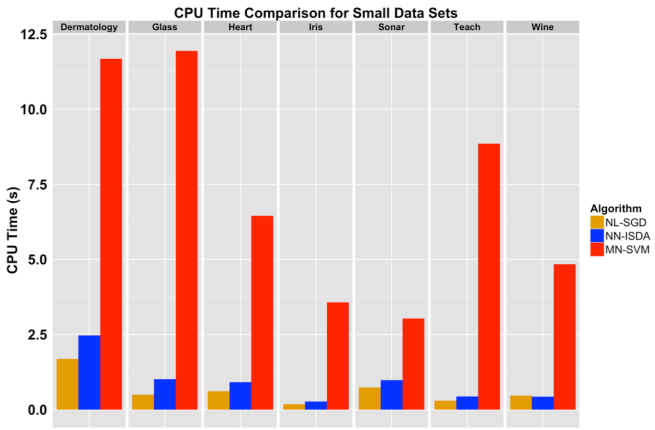

Fig 3. Algorithms' CPU Time for Small Datasets

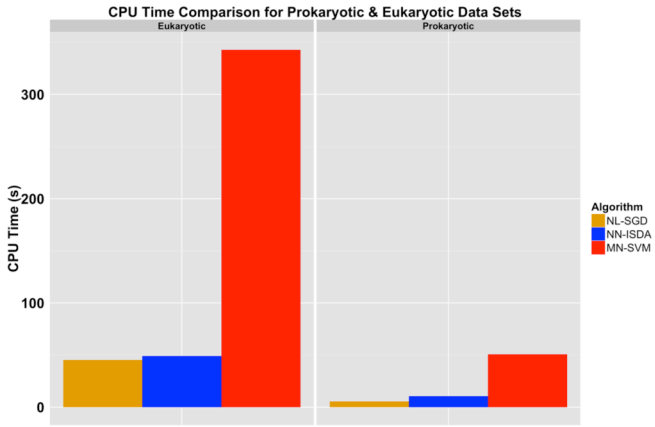

Fig 4. CPU Time for Prokaryotic & Eukaryotic Datasets

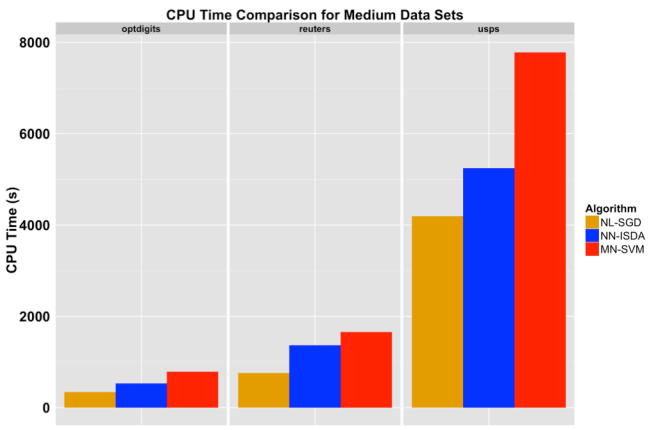

Fig 5. CPU Time for Medium Datasets

From the results above, we can see that our *NL-SGD* implementation is very competitive with respect to the *NN-ISDA* and *MN-SVM* algorithms. Being able to process large amounts of data with these CPU training times provides a sizeable advantage, especially because the accuracy of the model generated is not reduced. The average accuracy of *MN-SVM* is slightly higher, but being able to build a model with significant speedup in the training time compensates this.

C. Results for Different Configurations of NL-SGD

Below we can see how *NL-SGD*'s accuracy is affected by different configurations in terms of the number of epochs chosen, as well as the percentage of alpha changes taken into account. We tested our implementation using three different percentages of the $\boldsymbol{\alpha}$ updates, as explained in section III. These are shown in the following plots on the *x*-axis, where 0% represents taking the last $\boldsymbol{\alpha}$ vector as the final $\boldsymbol{\alpha}$. For the small datasets, we ran our experiment with two and five epochs, which are represented by the color of the bar plots. For the medium datasets, we ran our implementation with one and two epochs. Smaller datasets would require multiple updates to the $\boldsymbol{\alpha}$ vector to ensure reaching optimality, because the algorithm loops cyclically over the data. For larger datasets, this is not the case due to their size. This is shown in Figs. 6 and 7.

Fig. 6 shows that for the smaller sized datasets, such as the "Wine", "Glass", and "Teach", the accuracy achieved with five epochs surpasses that with two epochs. We can also see that for the larger of the small datasets, such as "Eukaryotic" and "Prokaryotic", the accuracy achieved with two epochs is better. This is because there were more updates than needed, and the minimum was overshot. We can also see that taking the average of the last 50% and 75% of alpha updates produces better results than taking the last alpha update.

From Fig. 7 we can see that there are minimal differences in the accuracy's achieved by using one or two epochs. Due to the small difference, we would recommend using one epoch to achieve faster CPU training time. Figures 8, 9, and 10 show the CPU training times achieved by our *NL-SGD* implementation.

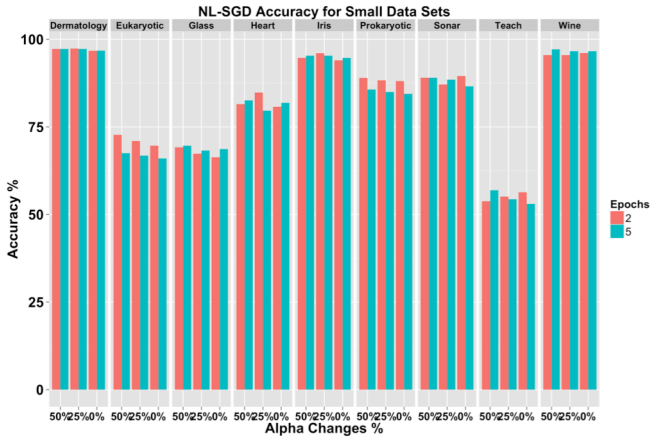

Fig 6. NL-SGD Accuracy on Small Datasets

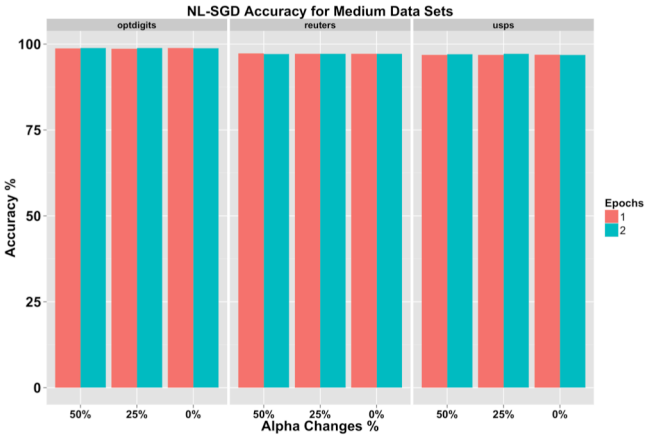

Fig 7. NL-SGD Accuracy on Medium Datasets

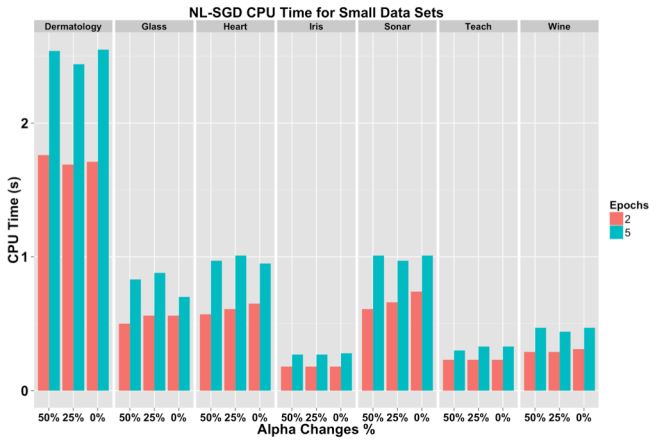

Fig 8. NL-SGD CPU Time on Small Datasets

We can definitely see a trade-off with balancing the number of epochs to achieve higher accuracy, while maintaining fast CPU training times. It is shown that as the number of samples increases, it is preferable to use a lower epoch number. This will ensure achieving a high accuracy while maintaining low CPU training time.

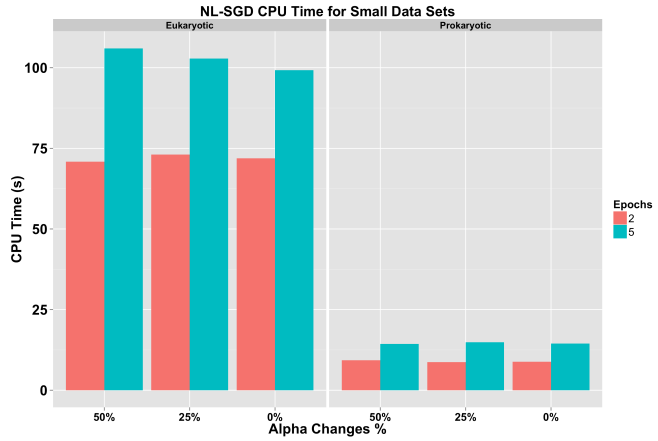

Fig 9. NL-SGD CPU Time on Prokaryotic & Eukaryotic Datasets

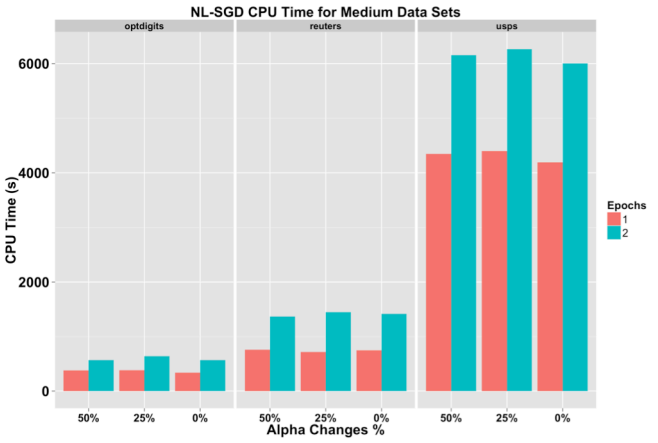

Fig 10. NL-SGD CPU Time on Medium Datasets

V. Conclusions

The potential for fast learning and building accurate models using stochastic gradient algorithms is very promising. Our implementation, named *NL-SGD*, shows CPU time speed up with respect to the fastest (known to us) NL-SVM algorithms implemented within the publically available GSVM code. The results shown in this paper are indicative of the quality of the models built, all the while having the best CPU training time, while keeping similar accuracies on various datasets. The results in [18] also show favorable results for online methods. The fact that online methods do not depend on the SVM penalty parameter *C*, unlike traditional approaches, such as the *SMO* algorithm, gives them a competitive edge. The results presented in this paper, as well as in [18], show that stochastic gradient training for the L1 SVM problem seems to be the strongest contender for classifying large datasets.

References

[1] Schoelkopf, B., Smola, A., Learning with Kernels; Support Vector Machines, Regularization, Optimization, and Beyond, The MIT Press, Cambridge, MA, 2002.

[2] Huang T.-M., Kecman, V., Kopriva, I., Kernel Based Algorithms for Mining Huge Data Sets, Supervised, Semi-supervised, and Unsupervised Learning, Springer-Verlag, Berlin, Heidelberg, 2006.

[3] Kecman V., Zigic Lj., Algorithms for Direct L2 Support Vector Machines, The IEEE International Symposium on INnovations in Intelligent SysTems and Applications, INISTA 2014, 978-1-4799-3020-3/14 ©2014 IEEE, Alberobello, Italy, 2014.

[4] Zigic Lj., Strack R., Kecman V., L2 Support Vector Machines Revisited - Novel Direct Learning Algorithm And Some Geometric Insights, Proceedings of 19th International Conference on Soft Computing, June 26-28, Brno, Czech Republic, pp.334-339, 2013.

[5] Platt J, Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Technical Report MSR-TR-98-14, April 1998.

[6] Strack, R., Geometric Approach to Support Vector Machines Learning for Large Datasets, PhD dissertation, Virginia Commonwealth University, Richmond, VA, 2013.

[7] Zigic Lj., Novel Support Vector Machine Model and Its Algorithms Aimed at Large Datasets, PhD dissertation, VCU, 2016.

[8] Kecman, V., Strack, R., Zigic, Lj., Big Data Mining by L2 SVMs - Geometrical Insights Help, Seminar at Computer Science Department, Virginia Commonwealth University, VCU, Richmond, VA, 2013.

[9] Kivinen J., Smola A. J., Williamson R. C., Online learning with kernel methods. In Advances in Neural Processing Systems, 2001.

[10] Bottou. Stochastic gradient descent on toy problems, http://leon.bottou.org/projects/sgd, 2007.

[11] Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for SVM. In *Proc. 24th Intl. Conf. on Machine Learning (ICML'07)*, pages 807–814. ACM, 2007.

[12] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning From Theory to Algorithms*. New York: Cambridge University Press, pp. 150-166, 2014

[13] 'UCI Machine Learning Repository: Data Sets', https://archive.ics.uci.edu/ml/datasets.html, 2015.

[14] Csie.ntu.edu.tw, 'LIBSVM Data: Classification, Regression, and Multi-label', https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/ datasets, 2015.

[15] Strack R., Kecman V., GitHub, 'strackr/gsvm', https://github.com /strackr/gsvm, 2013.

[16] Kwok J. T. Tsang I. W. and Cheung P.-M. "Core Vector Machines: Fast SVM Training on Very Large Data Sets". In: Journal of Machine Learning Research 6 , pp. 363-392, 2005.

[17] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: ACM Transactions on Intelligent Systems and Technology 2, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1-27:27, 2011

[18] Kecman V., Melki G., Fast Online Algorithms for Support Vector Machines – Models and Experimental Studies, Submitted to IEEE SoutheastConf, Norfolk, VA, 2016.