Comparison of Fast Learning Large Scale Multi-Class Classification

A. Jovanovich and A. Lazar

Department of Computer Science and Information Systems
Youngstown State University
Youngstown, Oh 44555
{ajovanovich, alazar}@gmail.com

Abstract

Recent progress in the development of techniques to optimize large-scale classification problems has extended the use of multi-class classification. Specifically the use of multi-class classification algorithms when the dataset is to large to fit into limited memory available of most computers. The most prominent algorithms used today solve the multi-class classification problem through an optimization approach based on coordinate decent. Two of the most recognized algorithms, Vowpal Wabbit and LIBLINEAR LibSVM have emerged as the most consistent options when solving for a multi-class problems. This paper proposes an analysis of these methods and tests the efficiency and performance of each algorithm. The results are recorded and comparisons are made. After analyzing the results, the conclusion made is that the Vowpal Wabbit algorithm is best suited for solving large-scale multi-class classification problems when computer memory is constrained.

Introduction

Big Data has affected the way statistical analysis is being conducted. Many real-world datasets contain hundreds or thousands of variables of interest which can contain hundreds of thousand or millions of records. Time spent on reading/writing between memory and disk becomes the bottleneck, rendering most algorithms inefficient (Yu et al. 2012). Even with the growing memory sizes of computers, a large data set can still be problematic. As a consequence, the complexity of analysis increasingly becomes unmanageable by using traditional machine learning algorithms. To extract useful knowledge from dense data makes the task of analysis time consuming. Coupled with the fact that most algorithms use a iteration process that cycles through the dataset multiple times, the process is seemingly impossible to finish.

In the past, classification models have been shown to handle large amounts of data well, and several optimization techniques have been applied to efficiently train data intensive models (Aly 2005). However the performance of the algorithms begin to decline when the data cannot be processed into memory (Yu et al. 2012). In these cases, training techniques that deal well with memory limitations become critical.

Recent progress has been made in the development of techniques to optimize over memory constrained systems, including binary classification. It is a well studied special case of the classification problem that provides the foundation for multi-class problem solvers. Statistical properties of binary classifiers, such as consistency, have been investigated in a variety of settings. Most machine-learning algorithms are originally developed for binary decision problems, and extended to handle multi-class problems (Argyriou, Herbster, and Pontil 2005).

The goal of this paper is to study the existing multi-class classification methods and provide a noteworthy comparison. In order to provide a thorough and comprehensive comparison, classification tests were run multiple times over contrasting datasets. The paper is organized in the following manner. Section 2 contains background information. Section 3 presents related work. Our approach is in section 4, followed by experiments in section 5. Conclusions follow and finally work cited.

Background

The principles of Multi-class classification are founded upon the same principles of binary classification. Each case involves assigning a class label for each instance present. Given a training data set of the form (x_i , y_i), where $x_i \in R_n$ is the i^{th} example and $y_i \in \{1, \ldots , K\}$ is the i^{th} class label, we aim at finding a learning model H

such that $H(x_i) = y_i$ for new unseen examples (Menon 2009). The problem is simply formulated in the two class case, where the labels y_i are just 0 or 1 for the two classes involved.

Several algorithms have been proposed to solve the binary problem. Some of these can be easily extended to the multi-class case, and some that involve high levels of computation. In contrast to traditional data classification where each instance is assigned to only one label, instances used in multi-class classification may be simultaneously relevant to several labels. For example: in text categorization, one document can belong to multiple topics.

Existing approaches to handle Large Scale multi-class classification can be categorized into two types. The first approach solves problems through a block minimization scheme. The second approach considers online learning algorithms (Dekel 2008). Both algorithms attempt to solve the classification problem of assigning labels from Y to instances X (Tewari and Bartlett 2007).

Block Minimization For Support Vector Machines

Optimization through block minimization has been used in the past to efficiently deal with data to large too fit into memory. One of the most prominent learning algorithms associated with block minimization is Support Vector Machine (SVM) algorithm. Proposed by Cortes and Vapnik in 1995, the algorithm has since grown into one of the most widely used learning algorithm in the world (Bottou and Lin 2007). The implementation and broad uses of SVMs have been well documented in the years since past.

The framework for an SVM is modified to solve for classification problems in the event limited memory is available. The process entails using an optimization technique based on block minimization. Where the term "block" refers to partitions of the dataset that can be read through the memory available. The size and content of each block varies from approach to approach. Pérez-Cruz, Figueiras, and Artes (2004) propose the use of "double chunking." Where data is partitioned into both "large chunks" and "small chunks." Another approach described by Chang and Roth (2011) uses selective sampling for block minimization. By selecting only significant instances, the goal is to minimize the size of data blocks and speed up the iteration process. Yu et al. (2012) suggest a framework for block minimization that is also used for testing in this paper. In this approach the amount of memory available for processing correlates to the size of blocks. The framework consists of 3 steps that split the data and read the blocks into memory, set initial values before solving for classification through an iterative process. The algorithm can be summarized as following:

Algorithm 1 A block minimization framework for Dual SVM

1. Split $\{1, \ldots, l\}$ to $B1, \ldots, B_m$ and store data into m partitions accordingly.
2. Set initial w
3. For j = 1, . . . , m (inner iterations)
   For K = 1, 2, . . . (outer iterations)
   3.1. Read xr , $\forall r \in B_j$ from disk
   3.2 Approximately solve the sub-problem (1)
   3.3 Update w by (2)

Here, $\{B_1, \ldots, B_m\}$ are sequential partitions of all data indices $\{1, \ldots, l\}$. The size of the blocks are determined by the known memory constraints. Each instance is read and randomly assigned a block. Algorithm 2 explains the process for data splitting. Optimization over a single block is identified as the inner iteration, whereas the m steps of going over all blocks is deemed an outer iteration (Yu et al. 2012).

Algorithm 2 Framework for block splitting

1. Decide m and create m empty files
2. For i = 1, ….
   2.1 Convert x_i to a binary format $\bar{x}_i$.
   2.2 Randomly choose number j {1,...,m}.
   2.3 append $\bar{x}_i$ into the end of the j^{th} file.

In order to optimize through block minimization only the dual form of SVM must be used. By examining the dual form of the optimization problem we are able to write the entire algorithm in terms of only inner products between input feature vectors. Updates to the weight vector **w**, which corresponds to the entire data set treating instances uniformly prevent the primal for of SVM to be used (Shalev-Shwartz et al. 2007). Algorithm 1 is able to efficiently learn in very high dimensional spaces.

The sub-problem is solved in the inner iteration step, and the solution is then used to update w. The solution uses only instances that belong to block B_j. The solution to the sub-problem is presented below in (1).

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{1}$$

The iteration round is then complete after **w** is updated. To update w, if d_{B_j} is an optimal solution for the sub-problem:

$$\boldsymbol{w} \leftarrow w + \sum_{r \in Bj} d_r y_r x_r \qquad (2)$$

The iteration process continues until optimization is reached, converging when one of two conditions are met (Yu et al. 2012). The first condition states that optimization is complete when the sub-problem for each block is solved and the solutions converge. The second condition is a stopping criteria. Usually a finite number of iterations is chosen, or an accuracy threshold is obtained.

LIBLINEAR addresses both conditions while solving for the sub-problem. The software contains a library with tools used for SVM classification when data cannot fit into memory (Yu et al. 2012). LIBLINEAR sequentially selects one variable for update and fixes others inside the block. The framework not shown in this paper is explained by Yu et al. (2012). LIBLINEAR uses a SVM coordinate descent method and solver to update instances in block B_j before solving for (1).

Online Learning with Stochastic Gradient Descent

Online learning algorithms are designed to effectively classify data by building a weight model derived from sequentially received training examples. Compared to block minimization which solves for the sub-problem of each block, online learning updates instances through the use of a cache file. Each iteration round updates the cache file where the weight model is stored. The algorithm classifies each instance, and uses the new "instance-label pair" to update and improve the stored model (Tewari and Bartlett 2007). This method is expected to accurately predict the labels of instances that are not part of the training set.

Several strategies were proposed to optimize online learning algorithms. Most of which aim to extend the original purpose of binary classification to multi-class learning. In the study conducted by Argyriou et al (2005), an extension to a graph-based approach to online learning was discussed. Shalev-Shwartz et al (2007) exploited the dual formation of optimization to create a more efficient online learning algorithm.

John Langford and his colleagues at Yahoo! Research developed Vowpal Wabbit, a fast online-learning algorithm that uses stochastic gradient descent. Vowpal Wabbit can handle very large datasets without ever needing to load the entire dataset into memory. The algorithm also requires less computational power and far fewer resources by learning through online gradient descent (Langford, Li, and Zhang, 2009). Algorithm 3 below presents the framework.

Algorithm 3 Online-learning framework for Vowpal Wabbit

Inputs:
- threshold $\theta \geq 0$
- gravity sequence $g_i \geq 0$
- learning rate $\eta \in (0, 1)$

initialize weights $w^j \leftarrow 0$ (j = 1, . . . , d)
1. Acquire an unlabeled example $x = [x^1, x^2, \ldots, x^d]$
2. Compute prediction: $y = \sum_j w^j x^j$
3. Acquire the label y from dataset
   for all weights w^j (j = 1, . . . , d)
   (a) if $w^j > 0$ and $w^j \leq \theta$ then $w^j \leftarrow \max\{w^j - g_i \eta, 0\}$
   (b) else if $w^j < 0$ and $w^j \geq -\theta$ then $w^j \leftarrow \min\{w^j + g_i \eta, 0\}$
4. Update weights for all features j: $w^j \leftarrow w^j + 2\eta(y - y)x^j$

Here, the superscripted symbol w^j denotes the j^{th} component of vector w in order to differentiate from w_i, which references the i^{th} weight vector (Langford and Zhang 2009).

Updates to the assigned weights are stored in a cache file that is small enough to be read into the limited memory available. Vowpal Wabbit stores only the cache file, allowing for faster implementation, and increased performance through the optimization process.

Related Work

In this section we discuss related work pertaining to comparisons of Large-scale Classification problems. The topic has compiled a comprehensive library primarily for binary classification. As such there seems to be a gap in knowledge pertaining to multi-class classification. We derive the techniques used for our comparison from strategies implemented in related work.

Yu et al. first introduced a comparison for large linear classification in his paper published in 2012. His comparison of SVM solvers and online-learning algorithms only extended to binary classification. The study defined one assumption that is significant in our comparison.

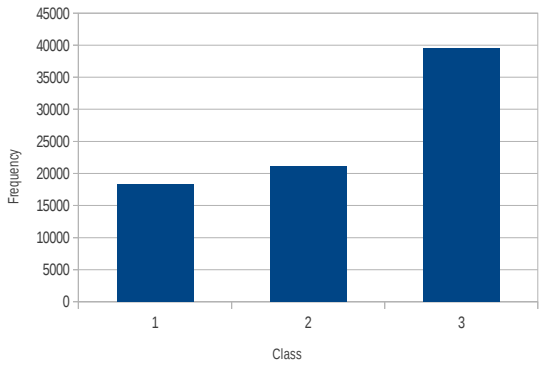

Figure 1 SensIT Vehicle Class Training Frequency

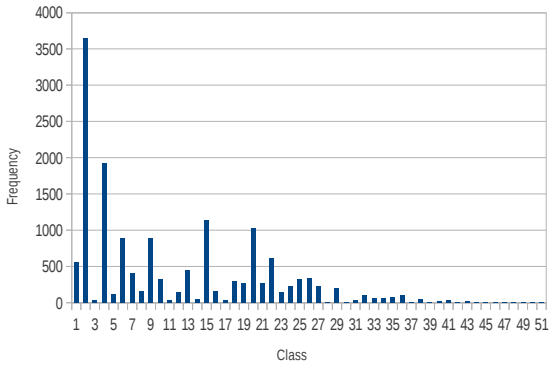

Figure 2 RCV1 Class Training Frequency

Assuming that the amount of available memory is limited, entire datasets cannot be stored in memory, but can be stored in the disk of one computer. The size of the datasets used in this paper are large enough to satisfying this constraint, and must be accessed through the hard drive where they are stored.

Multi-class classification was addressed in the paper by Chang and Roth in 2011. In the paper comparisons were drawn from both batch learning as well as online-learning algorithms. Unfortunately Vowpal Wabbit at that time was limited to binary classification and no comparisons were drawn. Instead the focus was primarily on block minimizing algorithms. Of which LibSVM was deemed superior to several other algorithms. It is our intent to extend this comparison of LibSVM to Vowpal Wabbit.

Approach

The aim of this paper is to test the performance of Vowpal Wabbit (Langford, Li, and Zhang, 2009) against the LIBLINEAR block learning extension of LibSVM (Yu et al 2012). Due to the recent advances in the Vowpal Wabbit library this is one of the first tests using the multi-class solver.

Testing was performed using one-against-all multi-class solver options implemented in both algorithms. For a K-class problem the one-against-all method constructs K models, where each model separates a class from the rest. The i^{th} model is trained with all of the binary instances pertaining to the ith class. The final output of the one-against-all method is the class that corresponds to the SVM with the highest output value (Liu, Wang, and Zeng 2007).

Data

Comparisons were drawn from test conducted over multi-class datasets. The SensIT Vehicle dataset was used, in addition to a larger 53 class (RCV1) dataset. Using the smaller dataset to establish a baseline reading and evaluation. The classification problem was then extended the simple classification problem to one more complex in that of the RCV1 dataset. The tests conducted here would reinforce the results gathered from the SensIT dataset.

The SensIT Vehicle dataset features 3-class labels. The instances were extracted from sensor data collected during a real world experiment carried out at Twenty nine Palms, CA in November of 2001 (Duarte and Hen 2004). The sensors were used to obtain both acoustic and seismic activity from vehicles in the vicinity. In total 50 features were extracted. Each vehicle was driven around a road while sensors collected information as they passed. Classes included in the training set included: AAV3 (class 1), DW3 (class 2), and a third class for noise (class 3). In total there are 78,823 training samples and 19,705 testing samples. In Figure 1 the frequency of each class used for training is presented. The distribution of samples shows an unbalanced dataset where more instances of noise classification are recorded than vehicle. The total sample size of the 2 vehicles combines is close to the total sample size of the noise. The data being skewed as such can increase the error rate and degrade performance in certain cases. Later in the paper we discuss the impact of this on classification accuracy.

The RCV1 dataset was used in part due to its 53-class problem. The Reuters Corpus Volume I (Reuters RCV1) is one of the most widely used test collection for text categorization research (Lewis et al. 2004). It contains 534,135 newswire documents, which are split into 15,564 training documents and 518,571 test documents. The

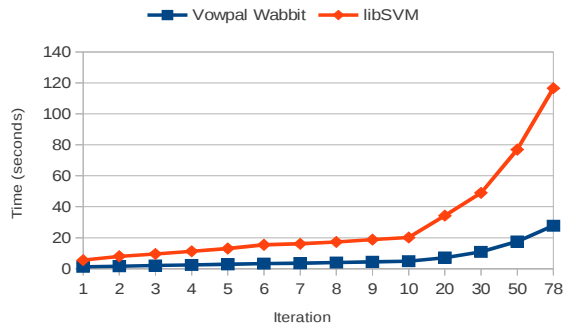

Figure 3 Iteration time over SensIT Data

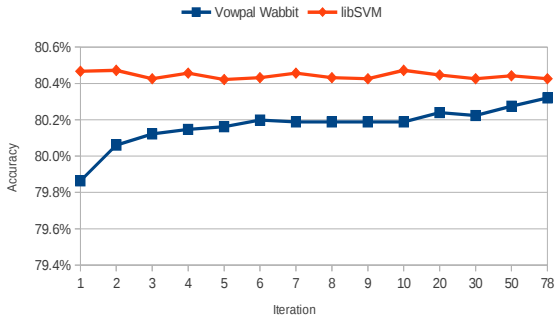

Figure 4 Percent Accuracy Per Iteration over SensIT Data

dataset contains 47,236 features. The class frequency of all 53 labels is presented in figure 2. Again the frequency of the samples collected in the training dataset represent an unbalanced sample size where the first 25 class frequencies are significantly larger than the last 28.

Experiment

In order to successfully draw comparisons testing criteria was first established. The criteria is based of overall performance of the ability of the algorithms to learn a classification model and successfully label the instance into the the right class. Stopping criteria was also established before tests were conducted. The iteration threshold was set to 78 rounds. Figures 3 – 6 present the individual performance measures of the algorithms from iteration round 1 to the stopping threshold reached in round 78. The algorithms were compiled in C++ and run on the Linux operating system. All experiments were run on a laptop with 6GB of memory. To ensure accurate results each test was repeated 3 times and the averages were used to rate performance.

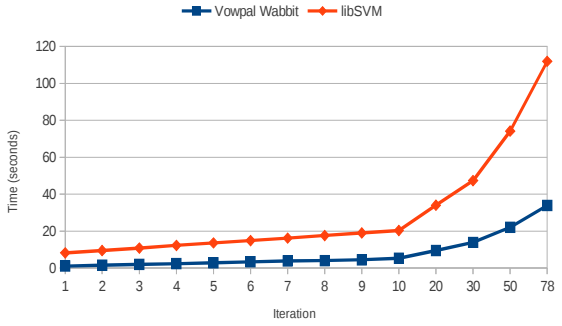

Figure 5 Iteration time over RCV1 Data

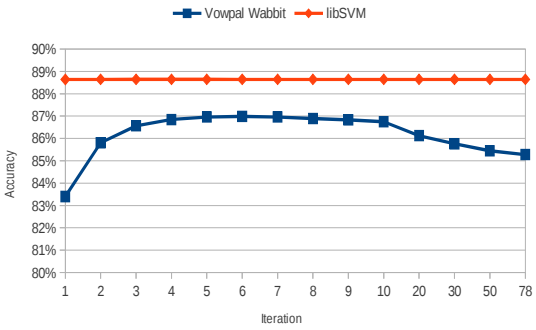

Figure 6 Percent Accuracy Per Iteration over RCV1 Data

Before the experiment can be run using the LIBLINEAR extension LibSVM, the training dataset must be partitioned into smaller blocks. The optimal number of partitions was discovered to be 8 for this case. Vowpal Wabbit on the other hand uses a cache file which fulfills the memory constraint. The dataset does not need to be split or compressed, Vowpal Wabbit can access each instance without reading the entire datasets. The algorithms were first trained and tested using the SensIT dataset. The dataset represents a simple scenario where binary classification is extended into a 3-class problem. The results from both algorithms are shown in figures 3 and 4.

On average the runtime per iteration for Vowpal Wabbit was 0.36 seconds, while the average for libSVM was 1.48 seconds. Vowpal Wabbit performed at a pace 4 times as fast as LibSVM. Both algorithms scored over 80% accuracy classifying testing instances. By the end of the 78 iteration the algorithms were close to converging around the 80.3 percentile. The biggest differences recorded can be seen between the first 5 iteration rounds. This is were the optimization approaches can be

SensIT Dataset					
Vowpal Wabbit			libSVM		
Iteration	Time	Acc	Iteration	Time	Acc
1	1.28	79.86%	1	5.52	80.47%
2	1.64	80.06%	2	8.00	80.47%
3	2.05	80.12%	3	9.53	80.43%
4	2.46	80.15%	4	11.25	80.46%
5	2.87	80.16%	5	13.03	80.42%
6	3.29	80.20%	6	15.43	80.43%
7	3.56	80.19%	7	16.09	80.46%
8	3.95	80.19%	8	17.27	80.43%
9	4.40	80.19%	9	18.76	80.43%
10	4.81	80.19%	10	20.14	80.47%
20	7.13	80.24%	20	34.28	80.45%
30	10.83	80.22%	30	48.93	80.43%
50	17.52	80.27%	50	76.93	80.44%
78	27.81	80.32%	78	116.56	80.43%

Figure 7 SensIT Data Table of Statistics

RCV1 Dataset					
Vowpal Wabbit			libSVM		
Iteration	Time	Acc	Iteration	Time	Acc
1	1.06	83.40%	1	8.20	88.64%
2	1.53	85.81%	2	9.46	88.64%
3	1.94	86.57%	3	10.80	88.64%
4	2.36	86.85%	4	12.25	88.64%
5	2.85	86.96%	5	13.61	88.64%
6	3.37	86.99%	6	14.87	88.64%
7	3.84	86.96%	7	16.20	88.64%
8	4.02	86.90%	8	17.63	88.64%
9	4.48	86.83%	9	19.01	88.64%
10	5.26	86.75%	10	20.33	88.64%
20	9.46	86.12%	20	33.99	80.45%
30	13.91	85.76%	30	47.30	80.43%
50	22.06	85.45%	50	74.17	80.44%
78	33.85	85.28%	78	111.93	80.43%

Figure 8 RCV1 Data Table of Statistics

distinguished. As the Vowpal Wabbit algorithm passes through the dataset and the weights are updated from the initial label, the accuracy of classification begins to rise. While the consistent accuracy rate for LibSVM can be attributed to the block minimization framework. By solving the subproblem within each equally distributed block first, the updated weights do not yield higher accuracy. Instead the rate is consistent throughout the optimization process.

The second experiment was conducting over the RCV1 dataset. Classification using the one-against-all option is increasingly complicated with each additional class. The increased sample, class, and feature size in the RCV1 dataset taxes the memory load in such a constrained environment. Performance from both algorithms stayed consistent with the baseline ratings established with the tests run over the SensIT dataset. Average runtime per iteration for Vowpal Wabbit increases slightly to 0.43 seconds, while LibSVM decreased slightly to 1.43 seconds. The runtime results of RCV1 is presented in figure 5 are similar to those of SensIT testing in figure 3. As with the sensIT data set, Vowpal Wabbit is consistently quicker than libSVM.

Classification accuracy rose 8 percent when comparing the results of figure 6 to figure 4. The larger dataset has had a positive impact on the accuracy rate of classification for the two algorithms. Again LibSVM scores a higher classification accuracy while achieving more consistent results. The accuracy of Vowpal Wabbit begins to rise until the 20th iterations. From there the accuracy begins to drop until the 78th iteration. Unlike the SensIT tests, the algorithms never appear to converge.

Overall the LIBLINEAR extension LibSVM classification algorithm achieved the highest classification rates. At no point in the iteration process did the Vowpal Wabbit algorithm correctly classify more test instances. The highest classification accuracy rate occurred during the first iteration step. LibSVM correctly classified 80.47% of the SensIT testing dataset, whereas Vowpal Wabbit managed to obtain a lower rate of 80.32% after the final iteration. Figure 7 presents a table with the results gathered testing over the SensIT test dataset. Once more LibSVM attained the highest classification rate over the RCV1 testing dataset. The results from both algorithms are presented in figure 8. the highest accuracy mark was obtained during the first iteration round when LibSVM correctly classifies 88.64% of testing instances. This rate was maintained throughout the iteration process. Vowpal Wabbit reached the $_t$ highest mark with an accuracy of 86.99% after the 6^{th} iteration round.

The higher accuracy ratings from LibSVM do come at a cost however. The runtime for each iteration was significantly slower than that of Vowpal Wabbit. The total time needed to reach the 78^{th} iteration round presented in figure 7 was only 27.81 seconds. Compared to 116.56 seconds for LibSVM. Figure 8 presented similar results. Vowpal Wabbit was recorded at 33.85 seconds where LibSVM reached the stopping threshold in 111.93 seconds. In both cases Vowpal Wabbit was 4 times as fast.

Surprisingly both algorithms achieved higher levels of performance solving for the larger RCV1 classification problem. Iteration runtime of both algorithms experienced a small increase in time when compared in figures 3 and 5. The classification accuracy improved in figure 6 when compared to the accuracy ratings in figure 3. The lower rates of classification accuracy can can be contributed to the skewed training dataset. The frequency distribution in presented in figures 1 and 2 are unbalanced. As such the classification accuracy over the smaller SensIT data set was lower than the larger RCV1 dataset. The increase in instances correlated with the accuracy of overall classification. The more instances the algorithm had to train over, the more classification accuracy improved.

Conclusion

Using the results recorded from figures 3 -8 the following question about Fast Learning large-scale multi-class classification can be answered:

Question: Which algorithm is most efficient when their are constraints to the memory? When compared, Vowpal Wabbit is the most efficient multi-class classification algorithm. The results from the SensIT test case suggested that Vowpal Wabbit was the quicker algorithm while maintaining a slightly lower accuracy percentile than LIBSVM. Moving from the SensIT dataset to the larger RCV1, the results remained consistent. We have concluded that Vowpal Wabbit had a slight advantage in overall efficiency when there is a constraint placed of computer memory size.

The performance of both algorithms was relatively close however. Due to the small size of the experiment, further testing is needed for a thorough comparison. Testing over datasets that are more expansive, both in sample and feature size could be used for a more significant experiment. However, we feel that the size of the datasets used in this paper are adequate to provide conclusive results for the comparisons made.

References

Aly, M. 2005. Survey on multiclass classification methods. *Neural networks*, 1-9.

Argyriou, A., Herbster, M., and Pontil, M. 2005. Combining graph Laplacians for semi--supervised learning.

Bottou, L., and Lin, C. J. 2007. Support vector machine solvers. *Large scale kernel machines:* 301-320.

Chang, K. W., and Roth, D. 2011. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining:* 699-707.

Chang, C. and Lin, C. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3): 27:1--27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine learning*, *20*(3): 273-297.

Dekel, O. 2008. From online to batch learning with cutoff-averaging. NIPS.

Duarte, M. F., and Hen Hu, Y. 2004. Vehicle classification in distributed sensor networks. Journal of Parallel and Distributed Computing *64*(7): 826-838.

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research 9:* 1871-1874.

Langford, J., Li, L., and Zhang, T. 2009. Sparse online learning via truncated gradient. *The Journal of Machine Learning Research 10:* 777-801.

Lewis, D., Yang, Y., Rose, T., and Li, F. 2004. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research* 5: 361-397.

Liu, Y., Wang, R., and Zeng, Y. S. 2007. An Improvement of One-Against-One Method for Multi-class Support Vector Machine. In *Machine Learning and Cybernetics, 2007 International Conference* 5: 2915-2920.

Menon, A. K. 2009. Large-scale support vector machines: algorithms and theory. *Research Exam, University of California, San Diego*.

Pechyony, D., Shen, L., and Jones, R. Solving Large Scale Linear SVM with Distributed Block Minimization.

Pérez-Cruz, F., Figueiras, A., and Artes, A. 2004. Double chunking for solving SVMs for very large datasets.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. 2007. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, 807-814.: ACM.

Tewari, A., and Bartlett, P. L. 2007. On the consistency of multiclass classification methods. *Journal of Machine Learning Research 8:* 1007-1025.

Yu, H. F., Hsieh, C. J., Chang, K. W., and Lin, C. J. 2012. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data TKDD 5*(4): 23.