DISTRIBUTED KERNEL LEARNING USING KERNEL RECURSIVE LEAST SQUARES

Nicholas J. Fraser, Duncan J.M. Moss, Nicolas Epain and Philip H.W. Leong

School of Electrical and Information Engineering
Building J03, The University Of Sydney, 2006, Australia

ABSTRACT

Constructing accurate models that represent the underlying structure of Big Data is a costly process that usually constitutes a compromise between computation time and model accuracy. Methods addressing these issues often employ parallelisation to handle processing. Many of these methods target the Support Vector Machine (SVM) and provide a significant speed up over batch approaches. However, the convergence of these methods often rely on multiple passes through the data. In this paper, we present a parallelised algorithm that constructs a model equivalent to a serial approach, whilst requiring only a single pass of the data. We first employ the Kernel Recursive Least Squares (KRLS) algorithm to construct several models from subsets of the overall data. We then show that these models can be combined using KRLS to create a single compact model. Our parallelised KRLS methodology significantly improves execution time and demonstrates comparable accuracy when compared to the parallel and serial SVM approaches.

Index Terms— Kernel Regression, Data Mining, Kernel Recursive Least Squares, Support Vector Machine

1. INTRODUCTION

The amount of data available continues to increase at an exponential rate [1] and fast implementations of data mining primitives are essential to make sense of them. Parallelism is a common technique used to scale algorithms to Big Data problems. Unfortunately, many standard data mining techniques have dependencies which prevent them from being easily parallelised on a distributed platform. As a result, standard machine learning algorithms such as SVM and kernel based least-squares optimisers are often not considered suitable for large data mining problems.

One method for performing a parallel computation whilst minimising communication overhead is to split the data up into several subsets, each of which is used to create an independent *submodel*. If the learning algorithm is appropriate, the individual submodels are a good representation of their respective subsets, but not necessarily a good representation of the entire data set. The core issue facing this technique then becomes how to combine the submodels into a single model.

In general, works which take this approach, such as [2–7], achieve a high speedup over batch algorithms and have little I/O overhead between nodes. However, this approach often has the following drawbacks: degradation in model accuracy, particularly as the number of processing nodes increases [5–7]; multiple passes of the data are required to guarantee convergence [2–4]; and the resultant model is much larger than one resulting from a batch approach [7]. To our knowledge, none of these methods provide a guarantee of the accuracy of the model for a single pass of the data.

An alternative method is to use a distributed platform to construct a single model. This approach differs from the submodel approach in that there is often a single model that is accessible from all computing nodes. The nodes then collectively optimise the model. A notable example is the paper by Chang et. al. [8]. The authors report a large speedup over LIBSVM [9]. The main drawback of this approach is the communication overhead, which accounts for over 50% of the running time on some datasets for a large number of machines.

In this paper, we take the approach of creating submodels and then combining them. We show that a closed form solution for combining individual Kernel Recursive Least Squares (KRLS) [10] submodels can be achieved. This result can then be simplified to a more compact form by applying the KRLS algorithm an additional time. We show that the final model is close to a serial, single module solution and only requires one-pass through the data. A theoretical analysis of the combined model is provided, along with empirical analysis of its performance and accuracy in relation to standard benchmarks.

The main contributions of this work can be summarised as follows: 1) a technique to combine multiple kernel based regression models on distributed nodes without requiring inter-node communication or multiple passes over the data; 2) a theoretical bound on the approximation error in the representation of the kernel matrix while using this technique; 3) empirical analysis of its performance on benchmark datasets when compared to other batch and distributed techniques.

2. KERNEL RECURSIVE LEAST SQUARES

In this section we briefly survey the Kernel Recursive Least Squares (KRLS) algorithm [10].

2.1. Kernel-based non-linear prediction

Consider a set of N observations in the form of input/output pairs $\{\mathbf{x}_n, y_n\}$, $n \in [1, N]$, where the input entries $\mathbf{x}_n$ are vectors of length M. We refer to this data set as the *training data*. In a typical time series prediction scenario, the vector $\mathbf{x}_n$ consists of the few data samples that directly precede the value y_n. Given a new input entry, $\mathbf{x}$, kernel-based methods predict the corresponding output as:

$$\hat{y} = \sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x})\, \alpha_i \, , \tag{1}$$

where $k(\cdot, \cdot)$ is a positive-definite kernel function.

Kernel-based prediction models thus consist of two types of data: a set of training input vector examples, which we refer to as the *dictionary*, and the corresponding coefficients α. The coefficients are typically calculated such that the prediction is as accurate as possible for the entire training data, in the least-square sense. In other words, the vector of the coefficients $\boldsymbol{\alpha} = [\alpha_1,\, \alpha_2,\, \ldots,\, \alpha_N]^T$ is defined as:

$$\boldsymbol{\alpha} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\| \mathbf{y} - \mathbf{K}\boldsymbol{\gamma} \right\|^2 \, , \tag{2}$$

where $\mathbf{y}$ is the vector of the training output entries, $\mathbf{y} = [y_1,\, y_2,\, \ldots, y_N]^T$, and $\mathbf{K}$ is the matrix with coefficients $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Calculating the kernel for two input entries $(\mathbf{x}_i,\, \mathbf{x}_j)$ is equivalent to calculating the dot product of the vectors $(\phi(\mathbf{x}_i),\, \phi(\mathbf{x}_j))$ where ϕ is a *mapping function* which transforms an input vector to a vector of features in a high-dimensional *feature space*. Thus, Equation (2) is equivalent to solving the following problem:

$$\boldsymbol{\alpha} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\| \mathbf{y} - \boldsymbol{\Phi}^T \boldsymbol{\Phi}\, \boldsymbol{\gamma} \right\|^2 \, , \tag{3}$$

where $\boldsymbol{\Phi}$ is the matrix that concatenates the vectors of features corresponding to every input entry, $\boldsymbol{\Phi} = [\phi(\mathbf{x}_1),\, \phi(\mathbf{x}_2),\, \ldots,\, \phi(\mathbf{x}_N)]$. Note that the function ϕ is implicit, as it is determined by the choice of a kernel, and it is never calculated in practice. This property is often referred to as the "kernel trick" [11].

2.2. Kernel-recursive least-squares

The Kernel-recursive least-squares (KRLS) algorithm [10] is an online algorithm which computes an approximate solution to Eq. (3). The main advantage of KRLS is that the complexity of the obtained prediction model does not depend directly on the size of the dataset but rather on how redundant this dataset is. This is made possible by the fact that the training entries may be linearly dependent in the feature space, *i.e.*, matrix $\boldsymbol{\Phi}$ can be approximated as:

$$\boldsymbol{\Phi} \approx \tilde{\boldsymbol{\Phi}}\, \mathbf{A}^T \, , \tag{4}$$

where $\tilde{\boldsymbol{\Phi}}$ consists of a subset of the columns of $\boldsymbol{\Phi}$ and matrix $\mathbf{A}$ expresses the columns of $\boldsymbol{\Phi}$ as linear combinations of the columns of $\tilde{\boldsymbol{\Phi}}$. In other words, KRLS selects a subset of input entries and computes the prediction model coefficients, $\tilde{\boldsymbol{\alpha}}$, defined by:

$$\tilde{\boldsymbol{\alpha}} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\| \mathbf{y} - \boldsymbol{\Phi}^T \tilde{\boldsymbol{\Phi}}\, \boldsymbol{\gamma} \right\|^2 = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\| \mathbf{y} - \mathbf{A}^T \tilde{\mathbf{K}}\, \boldsymbol{\gamma} \right\|^2 \, , \tag{5}$$

where $\tilde{\mathbf{K}}$ is the matrix of the kernel values for the subset of training entries. The solution to Equation (5) is given by: $\tilde{\boldsymbol{\alpha}} = \tilde{\mathbf{K}}^{-1}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$.

In the KRLS algorithm the approximation defined by Eq. (4) is controlled by a parameter, ν: the larger ν, the smaller the dictionary and the larger the error. The approximation error made on matrix $\mathbf{K}$ can be expressed as: $\mathbf{K} = \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^T + \mathbf{R}$, where $\mathbf{R}$ is a matrix of residual errors. Engel et al. [10] showed that the l_2 norm of $\mathbf{R}$ is bounded by $N\nu$, where the l_2 norm of a matrix is defined as: $\|\mathbf{R}\|_2 = \max_{\mathbf{u}:\|\mathbf{u}\|_2=1} \|\mathbf{R}\mathbf{u}\|_2$.

3. DISTRIBUTED KRLS LEARNING

In the case where KRLS is used on large training data sets, it may be advantageous to divide the learning operation into multiple processes running in parallel. The training data set is divided into subsets, $\mathbf{X} = [\mathbf{X_1}, ..., \mathbf{X_K}]$ and $\mathbf{y} = [\mathbf{y_1}^T, ..., \mathbf{y_K}^T]^T$, and sent to individual computation nodes for processing. Each node creates a model represented by a dictionary $\mathcal{D}_k$, and the corresponding weights, $\boldsymbol{\alpha}_k$. These models must then be combined with each other to form a unique model representing the whole training data set. In this section we show how multiple KRLS models can be combined so that the data contained in the entire training set is optimally approximated.

3.1. Concatenating KRLS models

The simplest way to combine KRLS models is to form a dictionary which concatenates the dictionary entries from every individual model, *i.e.*, $\bar{\mathcal{D}} = [\mathcal{D}_1,\; \ldots,\; \mathcal{D}_K]$. A new set of weights corresponding to dictionary $\bar{\mathcal{D}}$, $\bar{\boldsymbol{\alpha}}$, must then be calculated so that the prediction error is minimal for the entire training data set. In other words, denoting $\bar{\boldsymbol{\Phi}}$ as the dictionary $\bar{\mathcal{D}}$ mapped into the feature space, we have:

$$\bar{\boldsymbol{\alpha}} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\| \mathbf{y} - \boldsymbol{\Phi}^T \bar{\boldsymbol{\Phi}} \boldsymbol{\gamma} \right\| \approx \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\| \mathbf{y} - \bar{\mathbf{A}} \bar{\boldsymbol{\Phi}}^T \bar{\boldsymbol{\Phi}} \boldsymbol{\gamma} \right\| \, , \tag{6}$$

where $\bar{\mathbf{A}}$ is the diagonal-block matrix comprised of the $\mathbf{A}$ matrices for every KRLS model, $\bar{\mathbf{A}} = \operatorname{diag}([\mathbf{A}_1, \ldots, \mathbf{A}_K])$. The closed-form solution for $\bar{\boldsymbol{\alpha}}$ is therefore given by:

$$\bar{\boldsymbol{\alpha}} = \left(\bar{\boldsymbol{\Phi}}^T \bar{\boldsymbol{\Phi}}\right)^{-1} \left(\bar{\mathbf{A}}^T \bar{\mathbf{A}}\right)^{-1} \bar{\mathbf{A}}^T \mathbf{y} \, . \tag{7}$$

We now show that the weights $\bar{\boldsymbol{\alpha}}$ can be calculated without using the entire training data set, $\mathbf{y}$. In other words, $\mathbf{y}$ does not

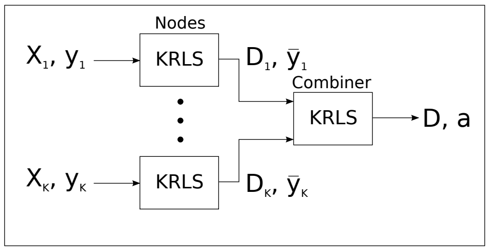

Fig. 1: High level view of the DistKRLS learning method.

have to be stored after the parallel training stage. Using the block-diagonal structure of $\bar{\mathbf{A}}$, Eq. (7) can be rewritten as:

$$\bar{\boldsymbol{\alpha}} = \left(\bar{\boldsymbol{\Phi}}^T\bar{\boldsymbol{\Phi}}\right)^{-1}\bar{\mathbf{y}} \ . \tag{8}$$

where $\bar{\mathbf{y}} = \left[\hat{\mathbf{y}}_1^T, \ldots, \hat{\mathbf{y}}_K^T\right]^T$ and $\hat{\mathbf{y}}_k = \left(\mathbf{A}_k^T\mathbf{A}_k\right)^{-1}\mathbf{A}_k^T\mathbf{y}_k$.

Recalling that the weights obtained for an individual model are given by $\boldsymbol{\alpha}_k = \left(\boldsymbol{\Phi}_k^T\boldsymbol{\Phi}_k\right)^{-1}\left(\mathbf{A}_k^T\mathbf{A}_k\right)^{-1}\mathbf{A}_k^T\mathbf{y}_k$, we have

$$\hat{\mathbf{y}}_k = \left(\boldsymbol{\Phi}_k^T\boldsymbol{\Phi}_k\right)\boldsymbol{\alpha}_k \ , \tag{9}$$

where $\boldsymbol{\Phi}_k$ denotes the dictionary $\mathcal{D}_k$ mapped in the feature space. Therefore, the coefficients of $\bar{\mathbf{y}}$ simply correspond to the data predicted by applying each KRLS model to its dictionary entries.

3.2. Combining KRLS models using the KRLS algorithm

From Eq. (8), we see that $\bar{\boldsymbol{\alpha}}$ is the solution to the following least-square problem:

$$\bar{\boldsymbol{\alpha}} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\|\bar{\mathbf{y}} - \bar{\boldsymbol{\Phi}}^T\bar{\boldsymbol{\Phi}}\boldsymbol{\gamma}\right\|^2 = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \left\|\bar{\mathbf{y}} - \bar{\mathbf{K}}\boldsymbol{\gamma}\right\|^2 \ . \tag{10}$$

Comparing this equation with Eq. (3), we observe that this problem is equivalent to estimating a model for the training set data comprised of: a) the entries of the concatenated dictionary, $\bar{\mathcal{D}}$; and b) the vector $\bar{\mathbf{y}}$. Therefore, the KRLS algorithm can be used to estimate this model. This leads to the distributed KRLS (DistKRLS) learning method summarised in Fig. 1: 1) The original training data set is divided into chunks that are sent to K parallel computing nodes; 2) The KRLS algorithm is used to derive a prediction model for each data chunk; 3) A new training data set is formed of the dictionary $\bar{\mathcal{D}}$ and vector $\bar{\mathbf{y}}$; 4) The KRLS algorithm is used to estimate a model for the new data set.

The advantage of this method is that it may lead to a more compact model than that defined by $\bar{\mathcal{D}}$ if there exists redundancies between entries of this dictionary. The drawback however is that discarding some dictionary entries may increase the prediction error. Let us define the solution after a second application of KRLS as $\breve{\boldsymbol{\alpha}}$, with a corresponding compact kernel matrix, $\breve{\mathbf{K}}$, and expansion matrix, $\breve{\mathbf{A}}$. Matrices $\breve{\mathbf{K}}$ and $\breve{\mathbf{A}}$ provide an approximation of $\bar{\mathbf{K}}$, as follows:

$$\bar{\mathbf{K}} = \breve{\mathbf{A}}\breve{\mathbf{K}}\breve{\mathbf{A}}^T + \breve{\mathbf{R}} \ , \tag{11}$$

where $\breve{\mathbf{R}}$ is the residual error introduced by the second application of the KRLS algorithm. Similarly, $\bar{\mathbf{K}}$ and $\bar{\mathbf{A}}$ provide an approximation of $\mathbf{K}$ and we can write:

$$\mathbf{K} = \bar{\mathbf{A}}\bar{\mathbf{K}}\bar{\mathbf{A}}^T + \bar{\mathbf{R}} = \bar{\mathbf{A}}(\breve{\mathbf{A}}\breve{\mathbf{K}}\breve{\mathbf{A}}^T + \breve{\mathbf{R}})\bar{\mathbf{A}}^T + \bar{\mathbf{R}} \ . \tag{12}$$

Thus, the l_2 norm of the total residual error, $\mathbf{R}_T$, in the representation of $\mathbf{K}$ is:

$$\begin{aligned}\|\mathbf{R}_T\|_2 &= \left\|\bar{\mathbf{R}} + \bar{\mathbf{A}}\breve{\mathbf{R}}\bar{\mathbf{A}}^T\right\|_2 \\ &\leq N\nu + \psi\bar{N}\nu \ ,\end{aligned} \tag{13}$$

where ψ is the maximum singular value of $\bar{\mathbf{A}}^T\bar{\mathbf{A}}$, $\bar{N}$ is the length of $\bar{\mathbf{y}}$ and the second line invokes the bound of the error introduced by the KRLS algorithm [10].

4. RESULTS

In this section, the accuracy and performance of the DistKRLS learning algorithm is shown. Synthetic benchmarks for regression and classification are used for training and test data. For comparison, results for the non-parallel KRLS (henceforth referred to as batch KRLS) algorithm is provided along with SVM and Cascade SVM (CSVM) [2], the latter of which is a technique to implement SVM on a distributed computing platform. CSVM was implemented in a standard binary tree configuration and was only allowed a single pass of the data. Readers should note that if multiple passes were allowed, CSVM would converge to the batch SVM solution but would suffer a performance penalty, which would likely be a linear increase in execution time.

4.1. Accuracy

In order to demonstrate the modelling accuracy of DistKRLS, the algorithm is tested on both a regression and a classification benchmark. DistKRLS and CSVM were tested while varying the number of *splits* in the training data. If $splits = 1$, then this refers to either batch KRLS or SVM. Otherwise, splits refers either to the number of submodels created by DistKRLS, or to the number of submodels created at the first layer of CSVM.

For the regression test, the Mackey-Glass Chaotic Time Series [12] (MG) was used with the chaotic parameter set to 30. The data was configured for a single step prediction problem with a time embedding of 7. In this test, 20 datasets were generated. Each dataset size was 2×10^5, 20% of the data in each set was used as a test set. For all learning algorithms, the Gaussian kernel was used, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$, with $\gamma = 0.5$. The configuration for batch and DistKRLS was $\nu = 10^{-4}$. SVM and CSVM used ϵ-SVR [13], provided by LIBSVM [9], with $\epsilon = 0.01$. The same parameters were used for DistKRLS and CSVM as their batch counterparts. Figure 2 shows the average mean squared error (MSE) and model size for each algorithm across the 20 sets. The error bars denote

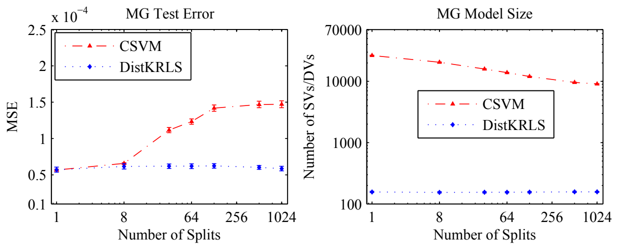

Fig. 2: Mean squared error (MSE) on the test set and model size for each algorithm.

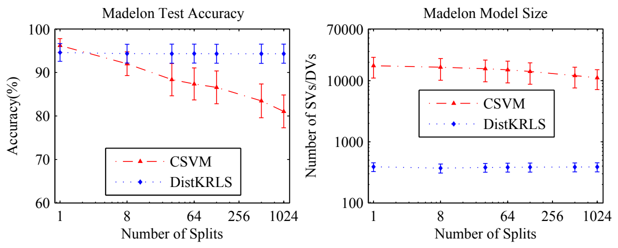

Fig. 3: Classification accuracy on the test set and model size.

one standard deviation above and below the average. Note that "model size" refers to the number of support vectors or dictionary vectors found by the algorithm. Clearly, DistKRLS and batch KRLS perform almost identically, while the accuracy of CSVM degrades slightly with increasing numbers of splits. Specifically, the maximum increase in average MSE between all configurations of DistKRLS and KRLS was 9.09%, while for CSVM it was 160% when compared to SVM. Also, the average model size of KRLS and DistKRLS is more than $150\times$ smaller. This also means the computational cost of subsequent predictions is over $150\times$ smaller.

For the classification test, the Madelon Classification Data [14] (MAD) was used. In this test, 40 datasets were generated. Each set contained 2×10^5 examples and was generated with 4 informative features and 4 redundant features, and 20% of the data in each set was used as the test set. The Madelon input data was normalised prior to training/prediction, and each output label was either -1 or 1. The exponential kernel was used again with $\gamma = 0.5$. KRLS and DistKRLS was used with $\nu = 0.7$. SVM and CSVM used C-SVC [11], also provided by LIBSVM [9], with $C = 50$. Figure 3 shows the classification accuracy and model size between DistKRLS and CSVM over the 40 sets. The error bars denote one standard deviation above and below the average. Using the classification data, the difference between our approach and CSVM is much more pronounced. While SVM works very well, the accuracy of CSVM deteriorates significantly while requiring over $40\times$ more support vectors than the KRLS approaches. Specifically, the greatest difference in average classification accuracy between DistKRLS and KRLS was 0.30%, compared with 15.1% between CSVM and SVM.

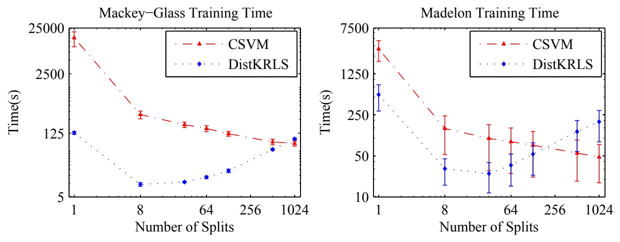

Fig. 4: Training time.

4.2. Performance

In this section, the execution time of the algorithms is considered. The Madelon dataset experiments were run on a Ubuntu Linux system with 2x Intel(R) Xeon(R) E5506 CPUs at 2.13GHz and 48GB of RAM. The Mackey-Glass experiments were run on a 16 node cluster running Centos Linux. MATLAB was used to run all tests, however, LIBSVM was used to implement all SVM algorithms and a C library was created to implement all KRLS algorithms. MATLAB's parallel computing toolbox was used to provide parallelism for the DistKRLS and CSVM.

Figure 4 shows the training time for each of the different algorithms. CSVM achieves a significant performance increase over SVM which continues to steadily improve with an increasing number of splits. DistKRLS performs best for either 8 or 32 splits, but approaches batch KRLS as the number of splits increases. This is due to an increase in computation time for combining the KRLS submodels. Compared to batch KRLS, the best average speedup achieved by DistKRLS is $22.7\times$. In all but 3 configurations, DistKRLS has lower average execution time than CSVM. Based on the results, it is likely that a continued increase in splits would cause CSVM to execute faster than DistKRLS.

5. CONCLUSION

This paper has presented a new method of combining multiple models of kernel based regression algorithms. This technique removes dependencies between models which allows it to be distributed among many machines. In this work, the KRLS algorithm is used for training submodels as it provides a compact, near optimal least squares solution. Speedups of up to $22\times$ are achieved with negligible degradation in accuracy compared to batch KRLS.

6. ACKNOWLEDGEMENT

This research was partially supported by Australian Research Councils Linkage Projects funding scheme (project numbers LP110200413 and LP130101034).

7. REFERENCES

[1] Alexander Szalay and Jim Gray, "2020 computing: Science in an exponential world," *Nature*, vol. 440, no. 7083, pp. 413–414, 2006.

[2] Hans P Graf, Eric Cosatto, Leon Bottou, Igor Dourdanovic, and Vladimir Vapnik, "Parallel support vector machines: The cascade SVM," in *Advances in neural information processing systems*, 2004, pp. 521–528.

[3] Jing Yang, "An improved cascade SVM training algorithm with crossed feedbacks," in *Computer and Computational Sciences, 2006. IMSCCS'06. First International Multi-Symposiums on*. IEEE, 2006, vol. 2, pp. 735–738.

[4] Yumao Lu, Vwani Roychowdhury, and Lieven Vandenberghe, "Distributed parallel support vector machines in strongly connected networks," *Neural Networks, IEEE Transactions on*, vol. 19, no. 7, pp. 1167–1178, 2008.

[5] Godwin Caruana, Maozhen Li, and Man Qi, "A MapReduce based parallel SVM for large scale spam filtering," *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 4, pp. 2659–2662, 2011.

[6] Godwin Caruana, Maozhen Li, and Yang Liu, "An ontology enhanced parallel SVM for scalable spam filter training," *Neurocomputing*, vol. 108, pp. 45–57, 2013.

[7] Bao-Liang Lu, Kai-An Wang, and Yi-Min Wen, "Comparison of parallel and cascade methods for training support vector machines on large-scale problems," in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*. IEEE, 2004, vol. 5, pp. 3056–3061.

[8] Edward Y Chang, "PSVM: Parallelizing support vector machines on distributed computers," in *Foundations of Large-Scale Multimedia Information Management and Retrieval*, pp. 213–230. Springer, 2011.

[9] Chih-Chung Chang and Chih-Jen Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 27, 2011.

[10] Yaakov Engel, Shie Mannor, and Ron Meir, "The kernel recursive least-squares algorithm," *Signal Processing, IEEE Transactions on*, vol. 52, no. 8, pp. 2275–2285, 2004.

[11] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sept. 1995.

[12] Michael C Mackey, Leon Glass, et al., "Oscillation and chaos in physiological control systems," *Science*, vol. 197, no. 4300, pp. 287–289, 1977.

[13] Vladimir Vapnik, *The nature of statistical learning theory*, springer, 2000.

[14] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror, "Result analysis of the NIPS 2003 feature selection challenge," in *Advances in Neural Information Processing Systems*, 2004, pp. 545–552.