International Journal of Computer Applications (0975 - 8887)
Volume 172 - No.3, August 2017

BIG Data: Implementation a Scala Approach for Large Scale Classification

Yassine Sabri
STIC Laboratory
Chouaib Doukkali University, B.P: 20
El Jadida MOROCCO

Najib El Kamoun
STIC Laboratory
Chouaib Doukkali University, B.P: 20
El Jadida MOROCCO

ABSTRACT

Many scientic investigations require data-intensive research where big data are collected and analyzed. To get big insights from big data, we need to rst develop our initial hypotheses from the data and then test and validate our hypotheses about the data. We propose FS-S , a flexible and modular *Scala* based implementation of the Fixed Size Least Squares Support Vector Machine (FS-LSSVM) for large data sets. The framework consists of a set of modules for (gradient and gradient free) optimization, model representation, kernel functions and evaluation of FS-LSSVM models. A kernel based *Fixed-Size Least Squares Support Vector Machine* (FS-LSSVM) model is implemented in the proposed framework, while heavily leveraging the parallel computing capabilities of *Apache Spark*. Global optimization routines like *Coupled Simulated Annealing* (CSA) and *Grid Search* are implemented and used to tune the hyper-parameters of the FS-LSSVM model. Finally, we carry out experiments on benchmark data sets like *Magic Gamma*, *Forest Cover*, *Susy* and *higgs* etc. and evaluate the performance of various kernel based FS-LSSVM models, all these combine to reveal an effective and ecient way to perform closed-loop big data analysis with visualization and scalable computing.

General Terms

Big Data, Data Analysing

Keywords

FS-LSSVM, Big Data, Large Scale Models, Non-linear SVMs

1. INTRODUCTION

A recent trend in many scientic investigations is to con-duct data-intensive research by collecting a large amount of high-density high-quality data. These data, such as text, video, audio, images, RFID, and motion tracking, are usually multi-faceted, dynamic, and extremely large in size, and likely to be substantially publically accessible for the purposes of continued and deeper data analysis. Indeed, data-driven discovery has already happened in various research elds, such as earth sciences, medi-cal sciences, biology and physics, to name a few. The 21^{st} century stands out in how mankind learned the value of storing and making predictions/decisions from large volumes of data. A significant aspect of large scale data analysis is distributed computation frameworks like *High Performance Computing*, *Message Passing Interface* etc. Recently large scale commodity hardware clusters have replaced the two former frameworks as the most popular model for parallel data analysis. With this crucial change in hardware came a change in computational models as well. It is at this juncture that distributed *Map Reduce* became the de-facto computational philosophy for large scale data analysis and words such as *Hadoop* [1], [8], [7] and *Apache Spark* [24], [3] have become synonymous with large scale data analysis and machine learning.

Along with innovation in hardware design and distributed computing models, there came a need for good programming libraries and frameworks to work with various Machine Learning models on large data sets. It was demonstrated in [10] that a gigantic language corpus encapsulates almost all aspects of human language and speech. So far the prevalent 'motto' in the Internet industry has been "large data, simple models". Often, this is misunderstood as the Machine Learning translation of *Occam's Razor*. The bias-variance trade-off [22] is a far better mechanism to ensure the model does not become overly complex, and this, rather than restricting the user to simple models, is the real Occam's razor in training a model.

Therefore, in order to extract maximum value from large scale data, it is important to have the flexibility to train and compare different model families before arriving at the one that fits the requirement of the user. Therefore one must be able to train general nonlinear models and tweak them by changing the various components which they employ to learn (i.e., a model may be linear or kernel based, it can be optimized by various methods like *Stochastic Gradient Descent*, *Conjugate Gradient*, etc.). This is not possible in a rigid, monolithic programming framework. Modularity, extensibility and ease of usage are of paramount importance while designing Machine Learning software for large scale data applications.

The current state of the art in distributed Machine Learning in Scala is the *MLLib* module in *Apache Spark* [16]. It has implementations of Linear SVM and Logistic Regression for solving binary classification problems. But a crucial component missing in *MLLib* and all distributed Machine Learning libraries is the ability to learn classification models with nonlinear decision boundaries. fs-s aims to solve the problem of scalable non-linear classification models by implementing the *Fixed-Size Least Squares Support Vector Machine* (FS-LSSVM) algorithm [9, 21] with model tuning capabilities.

In recent literature we find sparse reductions to FS-LSSVM methods [15, 14]. The authors in [15, 14] explored the sparsity vs error trade-off for FS-LSSVM models. Even though they run experiments on large scale datasets like Forest Cover dataset, the scalability of these methods are restricted to available memory on a single machine. Moreover, they don't exploit the possibility of parallelism available in several components of the FS-LSSVM model. Another work [12] converts the Big Data into a Big Network and then uses a network based subset selection technique (*Fast and Unique Representative Subset selection* (FURS) [13]) to obtain a representative subset of the original data. It then builds a FS-LSSVM model using this subset. However, in this paper we showcase that we can parallelize the subset selection technique which maximizes the *Quadratic Rènyi Entropy* for Big datasets and use the generated subset as the set of prototype vectors (PV) essential for building the FS-LSSVM model.

This paper is organized as follows. Section 2 introduces the FS-LSSVM algorithm [9]. Section 3 outlines the various modules that comprise fs-s and their roles. An implementation of the FS-LSSVM model is constructed using the framework and tested on various data sets, with the findings outlined in 4. Finally, we conclude in Section 5.

2. LEAST SQUARES SUPPORT VECTOR MACHINES

2.1 Formulation

Least Squares Support Vector Machines (LSSVM) [19] [20] modifies the SVM formulation to include the *squared error* loss function and equality constraints with respect to the error variables e_i, as shown in (1).

$$\begin{aligned} \min_{w,b,e} \quad & \mathcal{J}(w,e) = \frac{1}{2}w^\intercal w + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2 \\ \text{s.t.} \quad & y_i[w^\intercal \phi(x_i) + b] = 1 - e_i, i = 1, \ldots, N. \end{aligned} \tag{1}$$

Introducing the Lagrangian and applying the KKT conditions gives us the solution of the problem in the dual (2). This solution implies a loss of sparsity as compared to the classical SVM since each point becomes a support vector. However, we gain linearity of the solution (i.e. we do not have to solve the *Quadratic Programming* problem as in the classical SVM). Solving the problem in the dual is not advantageous for large scale analysis as the size of the solution matrix is equal to the size of the original data.

$$\left[\begin{array}{c|c} 0 & y^\intercal \\ \hline y & \Omega + \gamma^{-1} I \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline 1_v \end{array}\right], \tag{2}$$

where $\Omega_{kl} = y_k y_l K(x_k, x_l)$, $\alpha = [\alpha_1; ...; \alpha_N]$ and $K(x_k, x_l) = \phi(x_k)^\intercal \phi(x_l)$.

2.2 FS-LSSVM

In order to make the training of kernel based SVM models for large scale data applications feasible, one needs to make approximations to the computation of the kernel matrices. The Fixed-Size LSSVM (FS-LSSVM) as proposed by De Brabanter, Suykens et. al [9, 21] consists of solving the LSSVM problem in the primal as follows.

$$\min_{w,b} \ \frac{1}{2}w^\intercal w + \frac{\gamma}{2}\sum_{i=1}^{n}\left((y_i - w^\intercal \hat{\phi}(x_i) - b\right)^2. \tag{3}$$

The solution to equation 3 is given by:

$$\begin{pmatrix} \hat{w} \\ \hat{b} \end{pmatrix} = \left(\hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}\right)^{-1} \hat{\Phi}_e^\intercal y, \tag{4}$$

$$\text{where} \quad \hat{\Phi}_e = \begin{pmatrix} \hat{\phi}_1(x_1) & \cdots & \hat{\phi}_m(x_1) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \hat{\phi}_1(x_n) & \cdots & \hat{\phi}_m(x_n) & 1 \end{pmatrix}.$$

In the above formulation, $\hat{\phi}(x_k)$ is an approximation to the true feature map $\phi(x_k)$ which is related to the kernel $K(x_i, x_j) = \phi(x_i)^\intercal \phi(x_j)$ (Mercer's theorem). The approximate feature map $\hat{\phi}(x_k)$ is calculated using the Nyström method as outlined in [9, 15] and [14]. A low rank approximation to the kernel matrix is constructed by iteratively calculating a subset of the original data which maximizes the *Quadratic Rènyi Entropy*. This procedure of extracting $\hat{\phi}(x_k)$ from a data set, given a kernel function, is called *Automatic Feature Extraction* (AFE).

Kernel based models are sensitive to hyper-parameters. In the case of FS-LSSVM we have to tune the model with respect to γ the regularization parameter and the parameters of the kernel chosen. Models are generally compared with their cross-validation performance in which case the objective cost function with respect to the hyper-parameters is in general non-smooth and non-convex. Gradient free methods like Grid Search, Nelder Mead [17] and Coupled Simulated Annealing [23] are suitable to tackle the problem of model selection for FS-LSSVM based kernel models. Algorithm **??** explains the steps involved in tuning the FS-LSSVM model with the bold part representing our contributions in this paper, which have been implemented in a MapReduce setting.

Algorithm 1: Tuning FS-LSSVM

1 **Data**: Data Set, Kernel, Global Optimization routine, grid parameters
2 **Result**: Proposed Tuned FS-LSSVM model
3 Pre-process the data by mean scaling.;
4 **Calculate the prototype set by maximizing the Quadratic Rènyi Entropy in parallel using MapReduce.**;
5 Initialize a grid for the hyper-parameters;
6 **while** *termination of global optimization routine* **do**
7 Initialize the kernel using the hyper-parameters. Do AFE on the kernel matrix constructed from the prototypes, using the Nystrom method;
8 evaluate the cross validation score for the particular hyper-parameter values;
9 **end**

3. FS-LSSVM IMPLEMENTATION

Our Scala-based [18] software, called fs-s, tackles three major issues w.r.t. the implementation of the FS-LSSVM:

—**Tuning Kernel Models**: Since the performance of kernel based models is sensitive with respect to the choice of hyper-parameters, one has to choose a mechanism of model selection or hyper-parameter optimization. In fs-s, we implement the Grid Search and Coupled Simulated Annealing global optimization algorithms for model tuning.

—**Parallel Computation**: Big Data analysis requires the distribution of computational work load, *MapReduce* is the dominant paradigm employed for writing distributed data processing programs. In fs-s we leverage *MapReduce* to distribute the computation in the pre-processing, training and cross-validation tasks.

—**Infrastructure Flexibility**: The big data landscape has many tools which enable the storage and analysis of large streams of data, they consist of technologies such as, but not limited to *Apache Spark*, *Hadoop*, Graph Databases like *Titan* [5], *OrientDB* [4], *Neo4j* [2]. Creating a powerful framework for model training and evaluation requires the decoupling of storage and processing infrastructure from the actual logic that implements the architecture of learning models.

The implementation of the FS-LSSVM in fs-s, outlined in Algorithm **??** is as described in the original article of De Brabanter et al. [9]. Kernel based models all implement the interface *GloballyOptimizable* in the optimization module (see Figure 2). Since the *GlobalOptimizer* and its subclasses (i.e. *GridSearch* and *CoupledSimulated Annealing*) all optimize models which implement the GloballyOptimizable interface, it enables tuning of models with a variety of global optimization algorithms convenient.

Architecture

Figure 1 shows the organization of modules in fs-s. It can be decomposed into five principal modules:

—Model Classes: This is the core set of classes which form the heart of the library, a number of abstract model categories are defined each with its own set of defined behaviours.

—Optimization application programming interface (API): A module which houses the implementation of common optimization methods (i.e. Gradient and Gradient free). Currently fs-s has implementations for Conjugate Gradient, Gradient Descent, Grid Search and Coupled Simulated Annealing [23] (CSA).

—Kernels: fs-s is equipped with a powerful abstract API for representing kernel functions. The module has two abstract classes to outline the behaviors of kernels used in SVM based applications as well as density estimation. The library comes bundled with an implementation for AFE as well as for common SVM kernels i.e. Linear, Radial Basis Function (RBF), Polynomial, Laplace, Exponential. New kernel functions can be easily added to the library by extending the base classes in this module.

—Evaluation Metrics: We have implemented evaluation metrics for Binary Classification and Regression problems. Further more, the implementation of binary classification performance (area under ROC) is carried out using *MapReduce* in a *single pass* fashion through the evaluation data points, which can be seen in algorithm **??**. Calculating the area under the ROC curve in a *single pass* fashion greatly increases the speed of the eventual FS-LSSVM source code.

—Miscellaneous Utilities: This module contains code to carry out auxiliary tasks for model learning and optimization. It contains the implementation of entropy calculation, summary statistics, prototype selection as well as a set of various functions which can be required for implementing new model classes using the library.

The fs-s software is available at [6].

Map Reduce

As discussed above, we use *MapReduce* wherever possible in order to distribute the workload using *Apache Spark*. Due to the primal formulation of the FS-LSSVM, the size of matrix $A = \hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}$, in the linear system in (4) is $(m+1) \times (m+1)$, where m is the number of prototypes selected to construct the kernel matrix in kernel based FS-LSSVM. Procedure **??** outlines the procedure to estimate the parameters $\hat{w},\ \hat{b}$ of the FS-LSSVM model discussed in section 2.2, using the *Conjugate Gradient* algorithm.
Using *MapReduce*, we calculate $A = \hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}$ and $\hat{\Phi}_e^\intercal Y$. We use these results to carry out iterations of the Conjugate Gradient updates until the maximum number of iterations is reached.

Fig. 1: Schematic structure of fs-s

Algorithm 2: Calculate feature matrices from data using MapReduce: $FeatureMat$

Data: $X = [x^i],\ x^i \epsilon\, \mathbf{R}^n,\ \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m,\ Y = [y^i], y^i \epsilon \mathbf{R}$
Result: $\left(\hat{\Phi}_e^\intercal \hat{\Phi}_e\right),\ \hat{\Phi}_e^\intercal Y$
begin
  $MapFn(x, y)$:
    $M \longleftarrow \hat{\phi}(x)\hat{\phi}(x)^T$
    $v \longleftarrow \hat{\phi}(x)\ y$
  $emit(M, v)$
begin
  $RedFn((M, v), (M', v'))$:
  $emit(M + M', v + v')$
begin
  $(F, v) \longleftarrow MapReduce(X, MapFn, RedFn)$
  return (F, v)

Algorithm 3: Conjugate Gradient: CG

Data: $X = [x^i],\ x^i \epsilon\ \mathbf{R}^n, \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m, Y = [y^i], y^i \epsilon \mathbf{R}, \gamma, \epsilon$
Result: $\begin{pmatrix} \hat{w} \\ \hat{b} \end{pmatrix} = \left(\hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}\right)^{-1} \hat{\Phi}_e^\intercal Y$
begin
  $(F, v) \longleftarrow FeatureMat(X, Y, \hat{\phi}, \gamma)$
  $A \longleftarrow F + \frac{1}{\gamma}\mathbf{I}_{m \times m}$
  while *not* $maxiterations$ *and* $\Delta(\hat{w}, \hat{b}) >= \epsilon$ **do**
    $(\hat{w}, \hat{b}) \longleftarrow CGUpdate(\hat{w}, \hat{b}, A, v)$

Algorithm 4: Evaluate performance for fold: $evaluateFold$

Data: $X_f = [x^i],\ x^i \in \mathbf{R}^n, \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m,$
$Y_f = [y^i], y^i \in \mathbf{R}, \hat{w}, \hat{b}.$
Result: score for given fold
begin
  $predictLabel(\hat{w}, \hat{b})(x, y)$:
  $emit(\hat{w} \cdot x + \hat{b}, y)$
begin
  $Vector.fill(length)(IndicatorFn)$:
  $vec \longleftarrow (0, ..., 0)_{length}\ map(IndicatorFn)$
  $return(vec)$
begin
  $MapScore(score, label)$:
  if *label = 1.0* **then**
    $Pos \longleftarrow Pos + 1$
    $tpv \longleftarrow Vector.fill(l)(IndicatorFn(sign(score - thresholds(i)) == 1.0))$
    $fpv \longleftarrow Vector.fill(l)(IndicatorFn(false))$
  else
    $Neg \longleftarrow Neg + 1$
    $tpv \longleftarrow Vector.fill(l)(IndicatorFn(false))$
    $fpv \longleftarrow Vector.fill(l)(IndicatorFn(sign(score - thresholds(i)) == 1.0))$
  $emit(tpv, fpv)$
begin
  $RedScore((u, v), (u', v'))$:
  $emit(u + u', v + v')$
begin
  $thresholds \longleftarrow List(t_1, t_2, \ldots t_l)$
  $Pos \longleftarrow 0$
  $Neg \longleftarrow 0$
  $scoresLabels \longleftarrow (X_f, Y_f)\ map\ predictLabel(\hat{w}, \hat{b})$
  $(tp, fp) \longleftarrow scoresLabels\ map(MapScore)\ reduce(RedScore)$
  $tp \longleftarrow tp/Pos\ fp \longleftarrow fp/Neg$
  $roc \longleftarrow thresholds\ zip(tp\ zip\ fp)$
  return $1 - area(roc)$

Optimization/Hyper-parameter tuning

Figure 2 depicts the class hierarchy structure of the Optimization module of fs-s. The FS-LSSVM model class has an embedded optimization object which it inherits from the *Optimizer* interface.

Algorithm 5: Distributed v-Fold Cross-Validation

Data: $X = [x^i],\ x^i \epsilon\ \mathbf{R}^n, \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m, Y = [y^i], y^i \epsilon \mathbf{R}, \gamma,$ folds
Result: Cross Validation Performance
begin
  $(A, v) \longleftarrow FeatureMat(X, Y, \hat{\phi}, \gamma)$
  $score \longleftarrow 0$
  for $i \longleftarrow 1\ to\ folds$ **do**
    $(X_i, Y_i) \longleftarrow$ fold i
    $(A_i, v_i) \longleftarrow FeatureMat(X_i, Y_i, \hat{\phi}, \gamma)$
    $(\hat{w}, \hat{b}) \longleftarrow CG(A - A_i + \frac{1}{\gamma}\mathbf{I}_{m \times m}, v - v_i)$
    $score \longleftarrow score + evaluateFold(\hat{w}, \hat{b}, X_i, Y_i)$
  return $score/folds$

Implementations of Conjugate Gradient and Gradient Descent are provided in the optimization module. New optimization algorithms can be added by inheriting from the top level *Optimizer* interface or the *RegularizedOptimizer* abstract class in case one is working with parametric models which involve regularization. Another important component of the optimization module is the *GlobalOptimizer* interface which acts as a skeleton for implementing gradient free global optimization algorithms. We have implemented simple Grid Search and CSA, for tuning kernel based models. CSA as proposed by De Souza et al. [23] creates a grid (simplex) of hyper-parameter values and treats each point as a Simulated Annealing (SA) process.

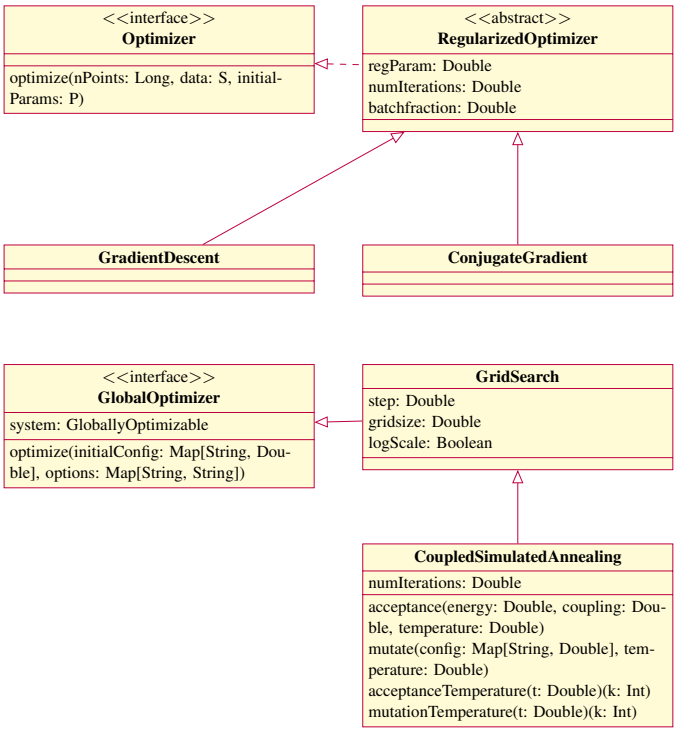

Fig. 2: Class Hierarchy of Optimization API

4. EXPERIMENTS

The experiments are performed on a 40 core 64GB RAM machine at the Department of Electrical Engineering, KU Leuven. The experiment parameters are summarized in table 1. We use the *Magic Gamma* Telescope, and *Adult* data sets available from the UCI Machine Learning Repository [11].

—Magic Gamma: The data is generated by the registration of high speed gamma particles measured by a ground based atmospheric Cherenkov gamma telescope. Each entry consists of 10 numerical attributes and a binary class attribute.

—Adult: This is based on a a census study carried out in 1994, the data consists of 6 numerical attributes and 8 categorical attributes. The target attribute is binary class value, which indicates if the given individual has an annual income more than 50000$.

The performance of binary FS-LSSVM classifiers on the *MAGIC Gamma* Telescope Data Set obtained from the UCI Machine Learning Repository, are summarized in Table 2. FS-LSSVM models trained with polynomial kernels give better classification performance than the RBF and Linear counterparts, on the *MAGIC Gamma* data.
The performance of binary FS-LSSVM classifiers on the *Adult* Data Set, are summarized in Table 3. FS-LSSVM models trained with exponential kernels give better classification performance than the RBF and Linear counterparts, on the *Adult* data. For both the data sets one sees a pattern emerging that tuning kernel models with CSA gives better results than naive *Grid Search* based hyperparameter optimization.

5. CONCLUSION

In this paper *fs-s*, a Scala-based implementation for training and tuning kernel based FS-LSSVM models. As a use case, the kernel based FS-LSSVM model is tested on benchmark data sets. We observed that our implementation enables scalable training, tuning and evaluation of models learning from Big Data, while still providing flexibility to tweak various underlying data processing infrastructure.

Acknowledgment

This work was supported by EU: ERC AdG A-DATADRIVE-B (290923), Research Council KUL: GOA/10/-/09 MaNet , CoE PFV/10/002 (OPTEC), BIL12/11T; PhD/Postdoc grants-Flemish Government; FWO: projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); PhD/Postdoc grants; IWT: projects: SBO POM (100031); PhD/Postdoc grants; iMinds Medical Information Technologies SBO 2014 and 2015-Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012-2017).

6. REFERENCES

[1] Apache hadoop: Lightning-fast cluster computing, 2005 (accessed July 6, 2015).

[2] Neo4j: The worlds leading graph database, 2007 (accessed July 6, 2015).

[3] Apache spark: Lightning-fast cluster computing, 2010 (accessed July 6, 2015).

[4] Orientdb, 2010 (accessed July 6, 2015).

[5] Titan: Distributed graph database, 2014 (accessed July 6, 2015).

[6] Fs-scala: Apache spark implementation of fixed size least squares support vector machines, 2015 (accessed July 12, 2015).

[7] Dhruba Borthakur, Samuel Rash, Rodrigo Schmidt, Amitanand Aiyer, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, and Aravind Menon. Apache hadoop goes realtime at Facebook. *SIGMOD '11 - Proceedings of the 2011 international conference on Management of data*, page 1071, 2011.

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[9] K. De Brabanter, J. De Brabanter, J. A. K. Suykens, and B. De Moor. Optimized fixed-size kernel models for large data sets. *Computational Statistics and Data Analysis*, 54(6):1484–1504, June 2010.

[10] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[11] M. Lichman. UCI machine learning repository, 2013.

[12] R. Mall, V. Jumutc, R. Langone, and J. A. K. Suykens. Representative subsets for big data learning using k-NN graphs. In *Proc. of IEEE BigData*, pages 37–42, 2014.

[13] R. Mall, R. Langone, and J. A. K. Suykens. FURS: Fast and Unique Representative Subset selection retaining large scale community structure. *Social Network Analysis and Mining*, 3(4):1075–1095, 2013.

[14] R. Mall and J. A. K. Suykens. Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data. In *Proc. of 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013)*, pages 161–173, 2013.

[15] R. Mall and J. A. K. Suykens. Very Sparse LSSVM Reductions for Large-Scale Data. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1086–1097, 2015.

[16] Xiangrui Meng, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. MLlib : Machine Learning in Apache Spark. *CoRR*, abs/1505.06807, 2015.

[17] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[18] Martin Odersky and al. An overview of the scala programming language. Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004.

[19] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002.

[20] J. A. K. Suykens and J Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[21] J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, 2002.

Table 1. : Experiment Parameters

Name	Meaning
Kernel	The type of kernel used i.e. RBF, Polynomial, etc
Prototypes	Size of prototype set
Global Opt.	Hyper-parameter optimization algorithm
Grid Size	Number of points (per hyper-parameter) in the grid
Grid Resolution	Distance between two adjacent points on each axis of the grid
Scale	Determines if the grid is on the logarithmic scale or linear
F1 score	avg. F1 score (measure of classification accuracy)
area under ROC	avg. area under the ROC curve

Table 2. : Magic Gamma Test Results

Kernel	**Prototypes**	**Global Opt**	**Grid Size**	**Grid R**	**Scale**	**max F**	**Area ROC**
RBF	50	gs	2	0.525	linear	0.67431(0.01283)	0.61592(0.0536)
RBF	50	gs	3	0.425	linear	0.67555(0.02012)	0.6182(0.0940)
RBF	50	csa	2	0.525	linear	0.69516(0.0463)	0.6726(0.1009)
RBF	50	csa	3	0.425	linear	0.70949(0.0362)	0.68452(0.1215)
Polynomial	50	gs	2	0.525	linear	0.71073(0.0674)	0.60584(0.1897)
Polynomial	50	gs	3	0.425	linear	0.71554(0.0626)	0.64792(0.1548)
Polynomial	50	csa	2	0.525	linear	0.7142(0.0128)	0.7103(0.0028)
Polynomial	50	csa	3	0.425	linear	0.71688(0.0191)	0.69069(0.0708)
Exponential	50	gs	2	0.525	linear	0.66805(0.0036)	0.47326(0.0538)
Exponential	50	gs	3	0.425	linear	0.71769(0.0079)	0.70283(0.03326)
Exponential	50	csa	2	0.525	linear	0.73585(0.0312)	0.74562(0.0304)
Exponential	50	csa	3	0.425	linear	0.72359(0.0249)	0.72973(0.0546)
Linear	50	gs	3	0.425	linear	0.6639(0)	0.4118(0)
Linear	50	csa	3	0.425	linear	0.6639(0)	0.4118(0)

Table 3. : Adult Data Set Test Results

Kernel	**Prototypes**	**Global Opt**	**GS**	**Grid R**	**Scale**	**max F1 score**	**Area ROC**
RBF	50	gs	5	0.35	linear	0.69168(0.0187)	0.65261(0.0559)
RBF	50	csa	5	0.35	linear	0.709(0.0198)	0.71215(0.00149)
RBF	50	gs	4	0.35	linear	0.69291(0.02286)	0.64006(0.0928)
RBF	50	csa	4	0.35	linear	0.7253(0.0229)	0.7205(0.0714)
RBF	50	csa	3	0.35	linear	0.69449(0.0295)	0.68823(0.06209)
RBF	50	gs	3	0.35	linear	0.67212(0.0076)	0.54631(0.06288)
RBF	50	csa	3	0.35	linear	0.70296(0.0521)	0.703(0.0760)
Exponential	50	gs	4	0.35	linear	0.68338(0.0223)	0.66438(0.0465)
Exponential	50	csa	3	0.35	linear	0.73457(0.00621)	0.74955(0.0392)
Exponential	50	csa	4	0.35	linear	0.73579(0.0189)	0.75386(0.0229)
Linear	50	csa	3	0.35	linear	0.69770(0)	0.68936(0)
Linear	50	csa	4	0.35	linear	0.69770(0)	0.68936(0)

[22] Giorgio Valentini, D S I Dipartimento, and Thomas G Dietterich. Bias-Variance Analysis of Support Vector Machines for the Development of SVM-Based Ensemble Methods. *Journal of Machine Learning Research*, 5:725–775, 2004.

[23] Samuel Xavier-De-Souza, J. A. K. Suykens, J. Vandewalle, and Désiré Bolle. Coupled simulated annealing. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(2):320–335, 2010.

[24] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark : Cluster Computing with Working Sets. *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, page 10, 2010.