Finite Sum Acceleration vs. Adaptive Learning Rates for the Training of Kernel Machines on a Budget

Tobias Glasmachers
Institute for Neural Computation
Ruhr-University Bochum, Germany
tobias.glasmachers@ini.rub.de

Abstract

Training predictive models with stochastic gradient descent is widespread practice in machine learning. Recent advances improve on the basic technique in two ways: adaptive learning rates are widely used for deep learning, while acceleration techniques like stochastic average and variance reduced gradient descent can achieve a linear convergence rate. We investigate the utility of both types of methods as well as combinations thereof for the training of kernel machines on a budget.

1 Introduction

Stochastic gradient descent (SGD) is a standard optimization technique for regularized risk minimization [1, 8]. It is conceptually simple and is easy to implement. It yields an anytime (online) learning algorithm applicable to large-scale problems with millions of instances, where it often delivers non-trivial models even before the first sweep through the data is completed. Therefore, the method is preferred to the full gradient method, also known in machine learning as batch training. For a convex objective function, standard SGD converges rather slowly at a rate of only $\mathcal{O}(1/\sqrt{t})$ (with suitably chosen learning rates, where t is the iteration counter), and at a faster but still slow rate of $\mathcal{O}(1/t)$ if the objective is strongly convex.

Recently, the development of SGD-type approaches has seen rapid progress, in two directions. Driven by the practical needs of deep network training, a variety of online adaptation methods for parameter-wise learning rates has been proposed (see [5, 3] and references therein), culminating in the widely applied ADAM algorithm [3]. Around the same time specialized methods for the optimization of finite sums were developed, e.g., with stochastic average gradients (SAG) [4, 6] or with stochastic variance reduced gradients (SVRG) [2] and related methods. They achieve the same convergence rate as batch gradient descent ($\mathcal{O}(1/t)$ in general and, astonishingly, linear convergence $\mathcal{O}(\rho^t)$ in the strongly convex case), while maintaining the low iteration cost of SGD. Interestingly, ADAM and finite sum methods can be easily combined.

It is by no means clear with (combination) of these techniques is best suited for a given problem. All of them have been tested for neural network training. In contrast, in the present paper we focus on kernel machines. We empirically investigate the practical utility of both types of methods as well as hybrids for the training of kernel machines on a budget.

2 Kernel machines on a budget

Given a labeled collection $\big((x_1, y_1), \dots, (x_n, y_n)\big) \in (X \times Y)^n$, we consider training of a kernel-based decision function $f(x) = \sum_{j=1}^m \alpha_j k(x, \tilde{x}_j)$ via minimization of the regularized empirical risk $\lambda \cdot \Omega(f) + \frac{1}{n} \sum_{i=1}^n L\big(y_i, f(x_i)\big)$ with respect to $\alpha = (\alpha_1, \dots, \alpha_m) \in \mathbb{R}^m$. Here, $k : X \times X \to \mathbb{R}$

is a Mercer kernel, Ω is a convex regularizer (often the squared norm $\Omega(f) = \|f\|_k^2$ induced by k), $\lambda \geq 0$ is a regularization hyperparameter, and the loss $L : Y \times \mathbb{R} \to \mathbb{R}$ is convex in f [7].

In general, the collection $\{\tilde{x}_1, \dots, \tilde{x}_m\}$ of basis points does not need to coincide with the set of training points. Instead, we consider a budgeted setting with $m \ll n$ by clustering or sub-sampling the data, to avoid the so-called curse of kernelization [10], i.e., the unbounded growth of the computational requirements of a single prediction and its gradient with n. However, our considerations apply to any generalized linear model $f(x) = \sum_{j=1}^{m} \alpha_j \phi_j(x)$.

3 Fast stochastic gradient descent: finite sums and adaptive learning rates

Early methods like PEGASOS [9] essentially apply plain SGD to the above training problem. Here we briefly review advanced SGD acceleration methods. Due to space constraints we refer to the literature for details of the algorithms under consideration (see [6, 2, 3]).

For the important special case $\Omega(f) = \frac{1}{2}\|f\|_k^2$ the objective function can be written as a finite sum consisting of $m + n$ terms: m components $f_l = \frac{\lambda}{2} \sum_{j=1}^{m} \alpha_j \alpha_l k(\tilde{x}_j, \tilde{x}_l)$ of the regularizer, and n loss terms $f_i = \frac{1}{n} L\big(y_i, \sum_{j=1}^{m} \alpha_j k(\tilde{x}_j, x_i)\big)$. The gradient of each term requires $\mathcal{O}(m)$ kernel evaluations. For many kernels the evaluation time is linear in the number of non-zero features. If the kernel values are cached, then the number of actual operations is reduced to $\mathcal{O}(m)$. Hence SAG [6] and SVRG [2] are directly applicable. Note that the memory requirement of SAG is as low as $\mathcal{O}(m + n)$ for a (generalized) linear model.

The ADAM algorithm [3] is the state-of-the-art method for online adaptation of individual learning rates for each coefficient α_j. Not only SGD, but also SAG and SVRG (except for SVRG's full gradient steps) compute one update vector per iteration. These updates can be directly plugged into the ADAM procedure, yielding the hybrid methods ADAM/SAG and ADAM/SVRG.

4 Empirical evaluation

We empirically evaluated SAG and SVRG for the training of kernel machines on a budget. We run batch gradient descent (GD) and stochastic gradient descent (SGD) as baselines. All three stochastic methods were run with and without ADAM for the adjustment of parameter-wise learning rates.

We consider three different loss functions: the logistic loss, giving rise to kernel logistic regression, the squared hinge loss yielding a support vector machine model, and the squared loss for kernel ridge regression. We apply the Gaussian kernel $k(x, x') = \exp\big(-\gamma\|x - x'\|^2\big)$ in all experiments. The data sets a9a, cod-rna, cover type, ijcnn1, and skin/non-skin for binary classification, as well as E2006 and year prediction (Million Song Dataset, MSD) for regression were obtained from the libSVM data website.[1] Details on the experimental setup and the tuned parameter values are provided in the supplementary material. The code for reproducing the experiments is available online.[2]

The learning curves of GD, SGD, SAG, and SVRG, with and without ADAM, are found in figures 1 and 2. Overall, SVRG and SAG perform clearly better than SGD, and of course, than plain GD. In all cases they are more successful at minimizing the training objective, and they are more stable. They are also more successful in terms of learning, yielding reduced test errors. Interestingly, the ADAM method is harmful in all cases, including its application to plain SGD, hinting at the interpretation that individual learning rates are not needed, and that noisy learning rate estimates are harmful.

5 Conclusion

We have presented an investigation of two different types of acceleration schemes of SGD to the training of kernel machines. Despite its importance for deep learning, online adaptation of parameter-wise learning rates turned out to be counter-productive for the problem under consideration. This result is somewhat unexpected. In contrast, methods specialized in the minimization of finite sums turned out to be highly effective. They significantly outperform plain SGD. These methods are well suited for the training of kernel machines.

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

[2] `https://www.ini.rub.de/PEOPLE/glasmtbl/code/budgeted_kernel_machine/`

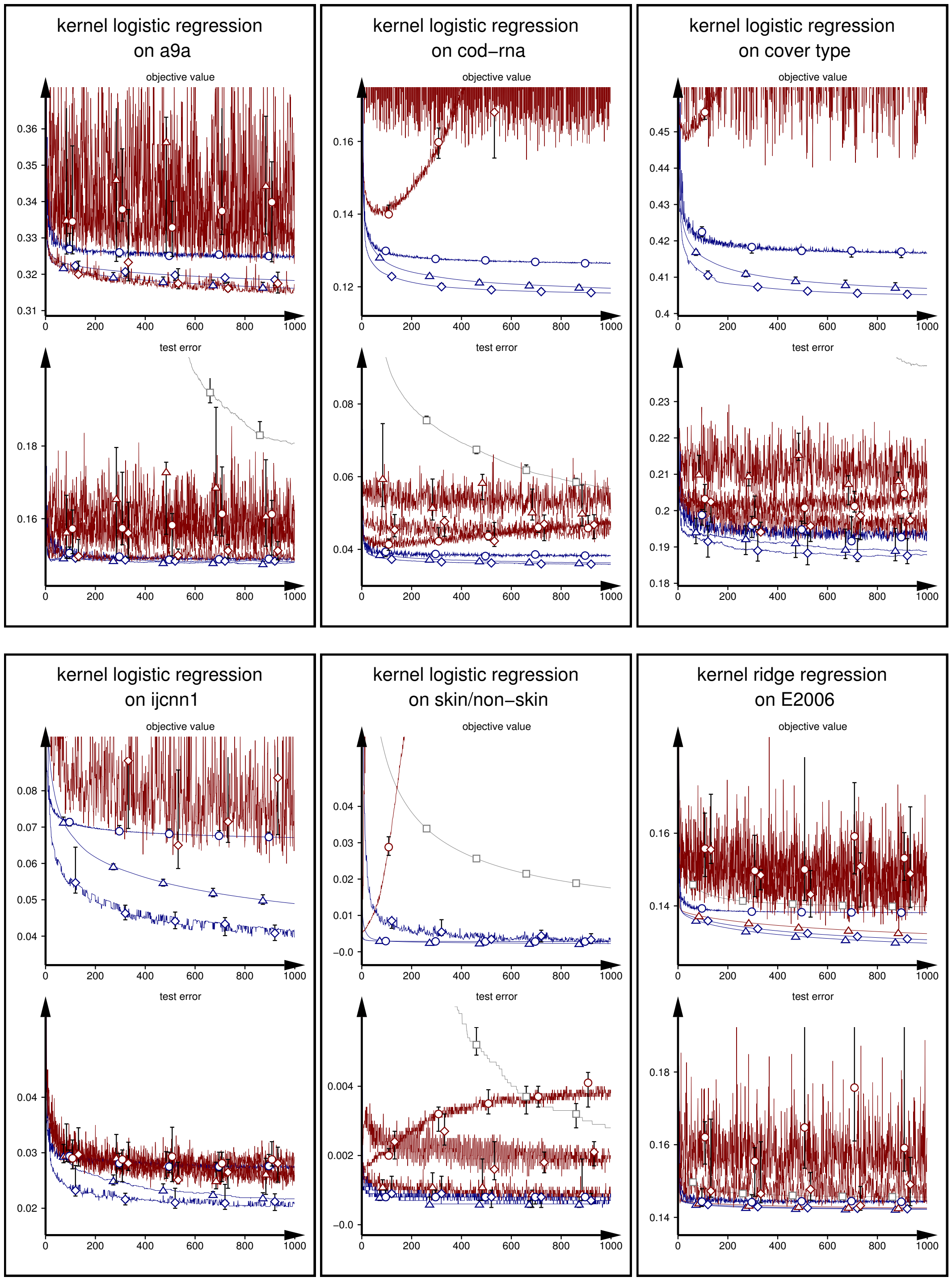

Figure 1: Evolution of median objective value (top part of each sub-figure) and test error (bottom part of each sub-figure) over 1000 epochs. Gray squares: (batch) GD, triangles: SAG, diamonds: SVRG, circle: SGD; blue: without ADAM, red: with ADAM. The 25% and 75% quantiles are indicated as error bars accompanying the symbols.

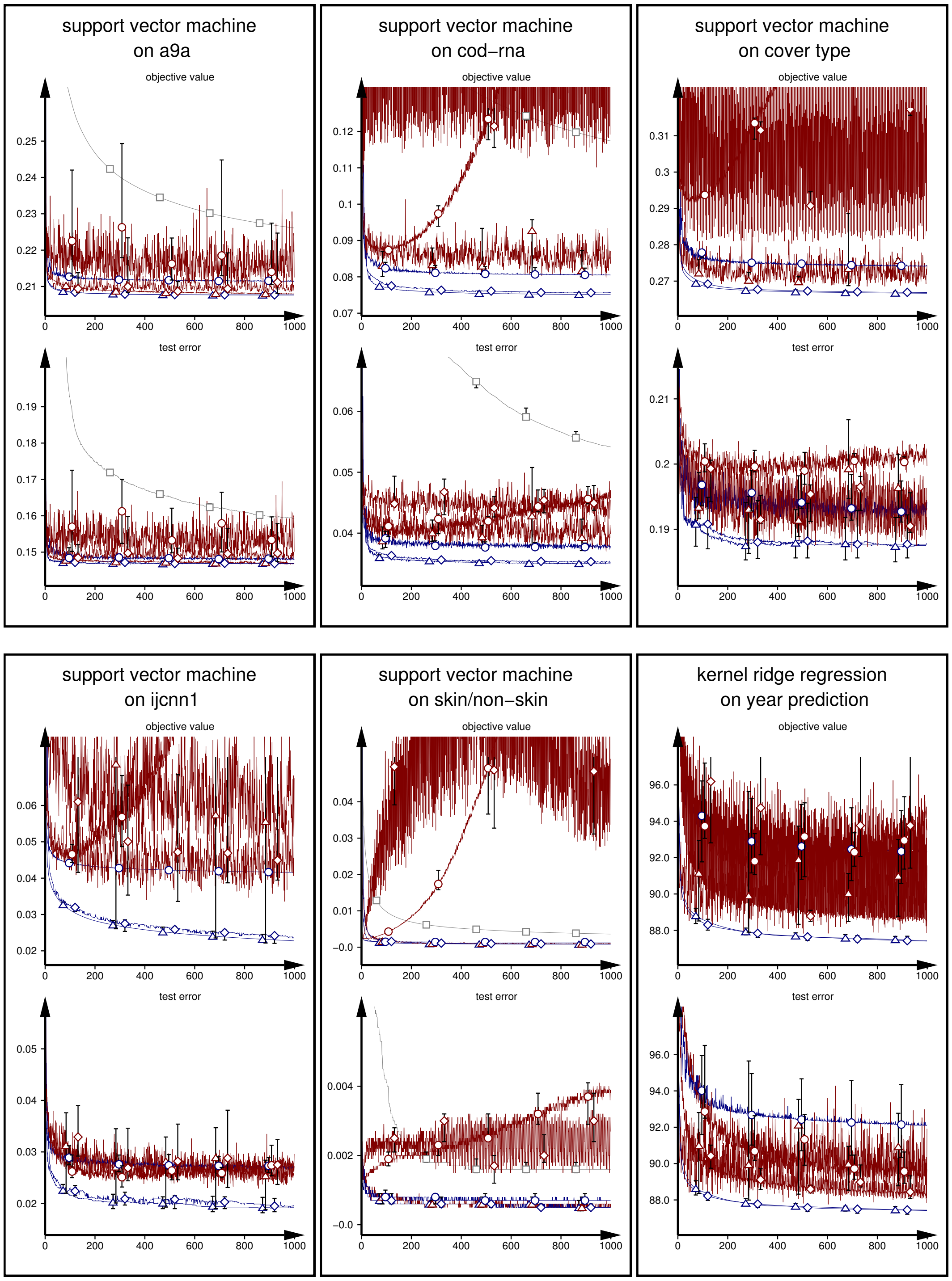

Figure 2: See caption of figure 1 for details.

Acknowledgments The author gratefully acknowledges support from the Deutsche Forschungsgemeinschaft (DFG) under grant number GL 839/3-1.

References

[1] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[2] R. Johnson and T. Zhang. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 315–323, 2013.

[3] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. Technical Report arXiv:1412.6980, arxiv.org, 2014.

[4] N. Le Roux, M. Schmidt, and F. Bach. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2663–2671, 2012.

[5] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In *International Conference on Machine Learning (ICML)*, pages 343–351. JMLR conferece proceedings, 2013.

[6] M. Schmidt, N. Le Roux, and F. Bach. Minimizing Finite Sums with the Stochastic Average Gradient. Technical Report arXiv:1309.2388, arxiv.org, 2013.

[7] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[8] S. Shalev-Shwartz. Online Learning and Online Convex Optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

[9] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical programming*, 127(1):3–30, 2011.

[10] Z. Wang, K. Crammer, and S. Vucetic. Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training. *The Journal of Machine Learning Research (JMLR)*, 13(1):3103–3131, 2012.

[11] Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

Supplementary Material for the Paper Finite Sum Acceleration vs. Adaptive Learning Rates for the Training of Kernel Machines on a Budget

Appendix A: Data Sets

The data sets a9a, cod-rna, cover type (binary), ijcnn1, skin/non-skin, E2006, and year prediction were obtained from the LIBSVM data website:

`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

If a data set did not come with a pre-defined split into training and test set, then 10,000 random points were split off as a test set; the rest was used for training. For all data sets except a9a and E2006 the features were scaled to unit variance to improve prediction performance. In contrast to standardization, the mean was not subtracted, in order to preserve sparsity.

The following table lists elementary properties of each learning problem (given by data set and loss): n is the number of training instances, nnz denotes the average number of non-zero features per data point, memory refers to the size of the kernel matrix between n training points and $m = 1000$ basis points in megabytes (using double precision numbers occupying 8 bytes each, 1 MB = 2^{20} bytes), and the column '# repetitions' lists the number of independent runs per problem.

data set	loss	n	nnz	memory (MB)	# repetitions
a9a	cross entropy	32,561	13.9	248	15
a9a	squared hinge	32,561	13.9	248	15
cod-rna	cross entropy	59,535	8.0	454	15
cod-rna	squared hinge	59,535	8.0	454	15
cover type	cross entropy	571,012	11.9	4,356	5
cover type	squared hinge	571,012	11.9	4,356	5
ijcnn1	cross entropy	39,990	13.0	305	15
ijcnn1	squared hinge	39,990	13.0	305	15
skin/non-skin	cross entropy	235,057	3.0	1,793	15
skin/non-skin	squared hinge	235,057	3.0	1,793	15
E2006	squared loss	16,087	1241.4	123	15
year prediction	squared loss	463,715	90	3,537	15

Appendix B: Parameter Settings

The following parameters were subject to tuning:

- the regularization parameter $\lambda \geq 0$,
- the kernel parameter $\gamma > 0$,
- the learning rate $\eta > 0$ (for SGD the initial learning rate),
- and the budget $m > 0$.

After initial experimentation the budget was set to $m = 1000$ for all data sets. For the data set sizes considered in this work, this value turns out to be close to the "sweet spot" of the trade-off between computational and storage complexity on the one hand, and predictive performance on the other hand.

The parameters λ and γ were tuned over the grid $\lambda \in \{0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and $\gamma \in \{2^{-10}, 2^{-9}, \dots, 2^{5}\}$. The setting $\lambda = 0$ corresponds to no regularization at all. This is feasible since the budget constraint as well as early stopping after a fixed number of epochs both have a regularizing effect [11]; in fact, models with $\lambda = 0$ performed well in many cases.

data set	loss	λ	γ
a9a	cross entropy	10^{-6}	$1/32$
a9a	squared hinge	10^{-4}	$1/16$
cod-rna	cross entropy	10^{-6}	$1/4$
cod-rna	squared hinge	10^{-5}	$1/4$
cover type	cross entropy	0	$1/16$
cover type	squared hinge	0	$1/16$
ijcnn1	cross entropy	0	$1/16$
ijcnn1	squared hinge	0	$1/16$
skin/non-skin	cross entropy	0	2
skin/non-skin	squared hinge	0	2
E2006	squared loss	0	16
year prediction	squared loss	0	$1/1024$

The learning rate η was tuned as follows, with the goal to use an as large as possible value. We started with $\eta = 1$. If a run diverged (the objective function value exceeded the initial value at the end of a sweep over the data) then the experiment was repeated with a halved learning rate. Note that the SAG algorithm contains a mechanism for adapting the learning rate online, hence the effective learning rate is usually lower. Of course, the same holds for all experiments involving the ADAM algorithm. Also note that for SGD the learning rate in iteration t is of the form $\eta^{(t)} = \eta \cdot \frac{n+m}{n+m+t}$, where $n + m$ is the number of terms in the sum (length of one epoch). We obtained the following problem-specific learning rates (median values):

data set	loss	learning rate η			
		GD	SGD	SVRG	SAG
a9a	cross entropy	$1/64$	$1/32$	$1/128$	$1/4$
a9a	squared hinge	$1/64$	$1/128$	$1/256$	$1/8$
cod-rna	cross entropy	1	1	$1/4$	1
cod-rna	squared hinge	$1/4$	$1/8$	$1/16$	1
cover type	cross entropy	1	1	$1/4$	1
cover type	squared hinge	$1/2$	$1/8$	$1/16$	1
ijcnn1	cross entropy	1	1	1	1
ijcnn1	squared hinge	$1/2$	$1/8$	$1/8$	1
skin/non-skin	cross entropy	1	1	1	1
skin/non-skin	squared hinge	1	$1/8$	$1/16$	1
E2006	squared loss	$1/32$	$1/128$	$1/256$	$1/16$
year prediction	squared loss	$1/128$	$1/512$	$1/128$	$1/16$

Appendix C: Selection of Basis Points

We defined basis functions $k(\tilde{x}_i, \cdot)$ by sampling a representative subset of the training points through a diversive hierarchical clustering approach based on a binary space partitioning tree. The root node is formed by the training set. For each cluster we estimate its extent as the maximal distance of 59 random pairs of points, yielding one of the 5% largest distances with probability $> 95\%$.[3] We keep track of the corresponding pair of points. We keep splitting the cluster with maximal extent into two sub-clusters until there are m leaf nodes in the tree. The set of basis functions is obtained by centering Gaussian kernels on the centroids of the leaf nodes. The complexity of this algorithm is $\mathcal{O}(np\log(m))$. In our experiments, it was usually faster than parsing the data files.

[3]The probability is computed as $1 - (1 - 0.05)^{59} \approx 0.9515$.