Effective Parallelisation for Machine Learning

Michael Kamp
University of Bonn
and Fraunhofer IAIS
kamp@cs.uni-bonn.de

Mario Boley
Max Planck Institute for Informatics
and Saarland University
mboley@mpi-inf.mpg.de

Olana Missura
Google Inc.
olanam@google.com

Thomas Gärtner
University of Nottingham
thomas.gaertner@nottingham.ac.uk

Abstract

We present a novel parallelisation scheme that simplifies the adaptation of learning algorithms to growing amounts of data as well as growing needs for accurate and confident predictions in critical applications. In contrast to other parallelisation techniques, it can be applied to a broad class of learning algorithms without further mathematical derivations and without writing dedicated code, while at the same time maintaining theoretical performance guarantees. Moreover, our parallelisation scheme is able to reduce the runtime of many learning algorithms to polylogarithmic time on quasi-polynomially many processing units. This is a significant step towards a general answer to an open question [21] on efficient parallelisation of machine learning algorithms in the sense of Nick's Class ($\mathcal{NC}$). The cost of this parallelisation is in the form of a larger sample complexity. Our empirical study confirms the potential of our parallelisation scheme with fixed numbers of processors and instances in realistic application scenarios.

1 Introduction

This paper contributes a novel and provably effective parallelisation scheme for a broad class of learning algorithms. The significance of this result is to allow the confident application of machine learning algorithms with growing amounts of data. In critical application scenarios, i.e., when errors have almost prohibitively high cost, this confidence is essential [27, 36]. To this end, we consider the parallelisation of an algorithm to be effective if it achieves the same confidence and error bounds as the sequential execution of that algorithm in much shorter time. Indeed, our parallelisation scheme can reduce the runtime of learning algorithms from polynomial to polylogarithmic. For that, it consumes more data and is executed on a quasi-polynomial number of processing units.

To formally describe and analyse our parallelisation scheme, we consider the regularised risk minimisation setting. For a fixed but unknown joint probability distribution $\mathcal{D}$ over an *input space* $\mathcal{X}$ and an *output space* $\mathcal{Y}$, a dataset $D \subseteq \mathcal{X} \times \mathcal{Y}$ of size $N \in \mathbb{N}$ drawn iid from $\mathcal{D}$, a convex *hypothesis space* $\mathcal{F}$ of functions $f\colon \mathcal{X} \to \mathcal{Y}$, a loss function $\ell\colon \mathcal{F} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that is convex in $\mathcal{F}$, and a convex regularisation term $\Omega\colon \mathcal{F} \to \mathbb{R}$, *regularised risk minimisation algorithms* solve

$$\mathcal{L}(D) = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{i=1}^{N} \ell\left(f, X_i, Y_i\right) + \Omega(f) \ . \tag{1}$$

The aim of this approach is to obtain a hypothesis $f \in \mathcal{F}$ with small *regret*

$$\mathcal{Q}\left(f\right) = \mathbb{E}\left[\ell\left(f, X, Y\right)\right] - \operatorname*{argmin}_{f' \in \mathcal{F}} \mathbb{E}\left[\ell\left(f', X, Y\right)\right] \ . \tag{2}$$

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Regularised risk minimisation algorithms are typically designed to be *consistent* and *efficient*. They are consistent if there is a function $N_0 \colon \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ such that for all $\varepsilon > 0$, $\Delta \in (0, 1]$, $N \in \mathbb{N}$ with $N \geq N_0(\varepsilon, \Delta)$, and training data $D \sim \mathcal{D}^N$, the probability of generating an ε-bad hypothesis is smaller than Δ, i.e.,

$$P\left(\mathcal{Q}\left(\mathcal{L}(D)\right) > \varepsilon\right) \leq \Delta \ . \tag{3}$$

They are efficient if the *sample complexity* $N_0(\varepsilon, \Delta)$ is polynomial in $1/\varepsilon, \log 1/\Delta$ and the *runtime complexity* $T_{\mathcal{L}}$ is polynomial in the sample complexity. This paper considers the parallelisation of such consistent and efficient learning algorithms, e.g., support vector machines, regularised least squares regression, and logistic regression. We additionally assume that data is abundant and that $\mathcal{F}$ can be parametrised in a fixed, finite dimensional Euclidean space $\mathbb{R}^d$ such that the convexity of the regularised risk minimisation problem (1) is preserved. In other cases, (non-linear) low-dimensional embeddings [2, 28] can preprocess the data to facilitate parallel learning with our scheme. With slight abuse of notation, we identify the hypothesis space with its parametrisation.

The main theoretical contribution of this paper is to show that algorithms satisfying the above conditions can be parallelised *effectively*. We consider a parallelisation to be effective if the (ε, Δ)-guarantees (Equation 3) are achieved in time polylogarithmic in $N_0(\varepsilon, \Delta)$. The cost for achieving this reduction in runtime comes in the form of an increased data size and through the number of processing units used. For the parallelisation scheme presented in this paper, we are able to bound this cost by a quasi-polynomial in $1/\varepsilon$ and $\log 1/\Delta$. The main practical contribution of this paper is an effective parallelisation scheme that treats the underlying learning algorithm as a *black-box*, i.e., it can be parallelised without further mathematical derivations and without writing dedicated code.

Similar to averaging-based parallelisations [32, 44, 45], we apply the underlying learning algorithm in parallel to random subsets of the data. Each resulting hypothesis is assigned to a leaf of an aggregation tree which is then traversed bottom-up. Each inner node computes a new hypothesis that is a *Radon point* [30] of its children's hypotheses. In contrast to aggregation by averaging, the Radon point increases the confidence in the aggregate doubly-exponentially with the height of the aggregation tree. We describe our parallelisation scheme, called the *Radon machine*, in detail in Section 2. Comparing the Radon machine to a sequential application of the underlying learning algorithm which achieves the same confidence, we are able to show a strong reduction in runtime from polynomial to polylogarithmic in Section 3.

The empirical evaluation of the Radon machine in Section 4 confirms its potential in practical settings. Given the same data as the sequential application of the base learning algorithm, the Radon machine achieves a substantial reduction of computation time in realistic application scenarios. In particular, using 150 processors, the Radon machine is between 80 and around 700-times faster than the base learner. Notice that superlinear speed-ups are possible for base learning algorithms with superlinear runtime. Compared with parallel learning algorithms from the Spark machine learning library, it achieves hypotheses of similar quality, while requiring only $15 - 85\%$ of their runtime.

Parallel computing [18] and its limitations [14] have been studied for a long time in theoretical computer science [7]. Parallelising polynomial time algorithms ranges from being 'embarrassingly' [26] easy to being believed to be impossible: For the class of decision problems that are the hardest in P, i.e., for P-complete problems, it is believed that there is no efficient parallel algorithm in the sense of Nick's Class (NC [9]): efficient parallel algorithms in this sense are those that can be executed

Algorithm 1 Radon Machine

Input: learning algorithm $\mathcal{L}$, dataset $D \subseteq \mathcal{X} \times \mathcal{Y}$, Radon number $r \in \mathbb{N}$, and parameter $h \in \mathbb{N}$
Output: hypothesis $f \in \mathcal{F}$

1: **divide** D into r^h iid subsets D_i of roughly equal size
2: **run** $\mathcal{L}$ in parallel to obtain $f_i = \mathcal{L}(D_i)$
3: $S \leftarrow \{f_1, \ldots, f_{r^h}\}$
4: **for** $i = h - 1, \ldots, 1$ **do**
5: **partition** S into iid subsets $S_1, \ldots, S_{r^i}$ of size r each
6: **calculate** $\mathfrak{r}(S_1), \ldots, \mathfrak{r}(S_{r^i})$ in parallel
7: $S \leftarrow \{\mathfrak{r}(S_1), \ldots, \mathfrak{r}(S_{r^i})\}$
8: **end for**
9: **return** $\mathfrak{r}(S)$

in *polylogarithmic time* on a *polynomial number of processing units*. Our paper thus contributes to understanding the extent to which efficient parallelisation of polynomial time learning algorithms is possible. This connection and other approaches to parallel learning are discussed in Section 5.

2 From Radon Points to Radon Machines

The Radon machine, as described in Algorithm 1, first executes the base learning algorithm on random subsets of the data to quickly achieve weak hypotheses and then iteratively aggregates them to stronger ones. Both the generation of weak hypotheses and the aggregation can be executed in parallel. To aggregate hypotheses, we follow along the lines of the iterated Radon point algorithm which was originally devised to approximate the centre point of a finite set of points [8]. The Radon point [30] of a set of points is defined as follows:

Definition 1. *A* Radon partition *of a set $S \subset \mathcal{F}$ is a pair $A, B \subset S$ such that $A \cap B = \emptyset$ but $\langle A \rangle \cap \langle B \rangle \neq \emptyset$, where $\langle \cdot \rangle$ denotes the convex hull. The* Radon number *of a space $\mathcal{F}$ is the smallest $r \in \mathbb{N}$ such that for all $S \subset \mathcal{F}$ with $|S| \geq r$ there is a Radon partition, or ∞ if no Radon partition exists. A* Radon point *of a set S with Radon partition $A, B \subset S$ is any $\mathfrak{r} \in \langle A \rangle \cap \langle B \rangle$.*

We now present the Radon machinein Algorithm 1, which is able to effectively parallelise consistent and efficient learning algorithms. Input to this parallelisation scheme is a learning algorithm $\mathcal{L}$ on a hypothesis space $\mathcal{F}$, a dataset $D \subseteq \mathcal{X} \times \mathcal{Y}$, the Radon number $r \in \mathbb{N}$ of the hypothesis space $\mathcal{F}$, and a parameter $h \in \mathbb{N}$. It divides the dataset into r^h subsets $D_1, \ldots, D_{r^h}$ (line 1) and runs the algorithm $\mathcal{L}$ on each subset in parallel (line 2). Then, the set of hypotheses (line 3) is iteratively aggregated to form better sets of hypotheses (line 4-8). For that the set is partitioned into subsets of size r (line 5) and the Radon point of each subset is calculated in parallel (line 6). The final step of each iteration is to replace the set of hypotheses by the set of Radon points (line 7).

The scheme requires a hypothesis space with a valid notion of convexity and finite Radon number. While other notions of convexity are possible [16, 33], in this paper we restrict our consideration to Euclidean spaces with the usual notion of convexity. Radon's theorem [30] states that the Euclidean space $\mathbb{R}^d$ has Radon number $r = d + 2$. Radon points can then be obtained by solving a system of linear equations of size $r \times r$ (to be fully self-contained we state the system of linear equations explicitly in Appendix C.1). The next proposition gives a guarantee on the quality of Radon points:

Proposition 2. *Given a probability measure P over a hypothesis space $\mathcal{F}$ with finite Radon number r, let F denote a random variable with distribution P. Furthermore, let $\mathfrak{r}$ be the random variable obtained by computing the Radon point of r random points drawn according to P^r. Then it holds for the expected regret $\mathcal{Q}$ and all $\varepsilon \in \mathbb{R}$ that*

$$P\left(\mathcal{Q}\left(\mathfrak{r}\right) > \varepsilon\right) \leq \left(rP\left(\mathcal{Q}\left(F\right) > \varepsilon\right)\right)^2 \quad .$$

A direct consequence of this proposition is a bound on the probability that the output of the Radon machine with parameter h is bad:

Theorem 3. *Given a probability measure P over a hypothesis space $\mathcal{F}$ with finite Radon number r, let F denote a random variable with distribution P. Furthermore, let $\mathfrak{r}_h$ be the random variable representing the Radon point obtained after h iterations with base hypotheses drawn according to P. Then for any convex function $\mathcal{Q} : \mathcal{F} \to \mathbb{R}$ and all $\varepsilon \in \mathbb{R}$ it holds that*

$$P\left(\mathcal{Q}(\mathfrak{r}_h) > \varepsilon\right) \leq \left(rP\left(\mathcal{Q}(F) > \varepsilon\right)\right)^{2^h} \quad .$$

The proofs of Proposition 2 and Theorem 3 are provided in Section 7. Note that this proof also shows the robustness of the Radon point compared to the average: if only one of r points is ε-bad, the Radon point is still ε-good, while the average may or may not be; indeed, in a linear space with any set of ε-good hypotheses and any $\varepsilon' \geq \varepsilon$, we can always find a single ε'-bad hypothesis such that the average of all these hypotheses is ε'-bad. For the Radon machine with parameter h, Theorem 3 shows that the probability of obtaining an ε-bad hypothesis is doubly exponentially reduced: with a bound δ on this probability for the base learning algorithm, the bound Δ for the Radon machine is

$$\Delta = (r\delta)^{2^h} \quad . \tag{4}$$

In the next section we will use this relation between Δ and δ to compare the Radon machine to a sequential application of the base learning algorithm which both achieve the same (ε, Δ)-guarantee.

3 Sample and Runtime Complexity

In this section we first derive the sample and runtime complexity of the Radon machine $\mathcal{R}$ from the sample and runtime complexity of the base learning algorithm $\mathcal{L}$. We then relate the runtime complexity of the Radon machine to a sequential application of the base learning algorithm when both achieve the the same (ε, Δ)-guarantee. For that, we assume that the base learning algorithms is consistent and efficient with a sample complexity of the form $N_0^{\mathcal{L}}(\varepsilon, \delta) = (\alpha_\varepsilon + \beta_\varepsilon \operatorname{ld} 1/\delta)^k$, for some[1] $\alpha_\varepsilon, \beta_\varepsilon \in \mathbb{R}$, and $k \in \mathbb{N}$. We assume for the base learning algorithm that $\delta \leq 1/2r$.

The Radon machine creates r^h base hypotheses and, with Δ as in Equation 4, has sample complexity

$$N_0^{\mathcal{R}}(\varepsilon, \Delta) \;=\; r^h N_0^{\mathcal{L}}(\varepsilon, \delta) \;=\; r^h \cdot \left(\alpha_\varepsilon \;+\; \beta_\varepsilon \operatorname{ld} \frac{1}{\delta}\right)^k \quad . \tag{5}$$

Theorem 3 then shows that the Radon machine with base learning algorithm $\mathcal{L}$ is consistent: with $N \geq N_0^{\mathcal{R}}(\varepsilon, \Delta)$ samples it achieves an (ε, Δ)-guarantee. To achieve the same guarantee, the application of $\mathcal{L}$ itself, sequentially, would require $M \geq N_0^{\mathcal{L}}(\varepsilon, \Delta)$ samples, where

$$N_0^{\mathcal{L}}(\varepsilon, \Delta) \;=\; N_0^{\mathcal{L}}\left(\varepsilon, (r\delta)^{2^h}\right) \;=\; \left(\alpha_\varepsilon \;+\; 2^h \cdot \beta_\varepsilon \operatorname{ld} \frac{1}{r\delta}\right)^k \quad . \tag{6}$$

For base learning algorithms $\mathcal{L}$ with runtime $T_{\mathcal{L}}(n)$ polynomial in the data size $n \in \mathbb{N}$, i.e., $T_{\mathcal{L}}(n) \in \mathcal{O}\left(n^\kappa\right)$ with $\kappa \in \mathbb{N}$, we now determine the runtime $T_{\mathcal{R},h}(N)$ of the Radon machine with h iterations and $c = r^h$ processing units on $N \in \mathbb{N}$ samples. In this case all base learning algorithms can be executed in parallel. In practical applications fewer physical processors can be used to simulate r^h processing units—we discuss this case in Section 5.

The runtime of the Radon machine can be decomposed into the runtime of the base learning algorithm and the runtime for the aggregation. The base learning algorithm requires $n \geq N_0^{\mathcal{R}}(\varepsilon, \Delta)/r^h$ samples and can be executed on r^h processors in parallel in time $T_{\mathcal{L}}(n)$. The Radon point in each of the h iterations can then be calculated in parallel in time r^3 (see Appendix C.1). Thus, the runtime of the Radon machine with $N = r^h n$ samples is

$$T_{\mathcal{R},h}(N) = T_{\mathcal{L}}\left(n\right) + hr^3 \quad . \tag{7}$$

In contrast, the runtime of the base learning algorithm for achieving the same guarantee is $T_{\mathcal{L}}(M)$ with $M \geq N_0^{\mathcal{L}}(\varepsilon, \Delta)$. Ignoring logarithmic and constant terms, $N_0^{\mathcal{L}}(\varepsilon, \Delta)$ behaves as $2^h N_0^{\mathcal{L}}(\varepsilon, \delta)$. To obtain polylogarithmic runtime of $\mathcal{R}$ compared to $T_{\mathcal{L}}(M)$, we choose the parameter $h \approx \operatorname{ld} M - \operatorname{ld}\operatorname{ld} M$ such that $n \approx M/2^h = \operatorname{ld} M$. Thus, the runtime of the Radon machine is in $\mathcal{O}\left(\operatorname{ld}^\kappa M + r^3 \operatorname{ld} M\right)$. This result is formally summarised in Theorem 4.

Theorem 4. *The Radon machine with a consistent and efficient regularised risk minimisation algorithm on a hypothesis space with finite Radon number has polylogarithmic runtime on quasi-polynomially many processing units if the Radon number is upper bounded by a function polylogarithmic in the sample complexity of the efficient regularised risk minimisation algorithm.*

The theorem is proven in Appendix A.1 and relates to Nick's Class [1]: A decision problem can be solved efficiently in parallel in the sense of Nick's Class, if it can be decided by an algorithm in polylogarithmic time on polynomially many processors (assuming, e.g., PRAM model). For the class of decision problems that are the hardest in P, i.e., for P-complete problems, it is believed that there is no efficient parallel algorithm for solving them in this sense. Theorem 4 provides a step towards finding efficient parallelisations of regularised risk minimisers and towards answering the open question: is consistent regularised risk minimisation possible in polylogarithmic time on polynomially many processors. A similar question, for the case of learning half spaces, has been coined a fundamental open problem by Long and Servedio [21] who gave an algorithms which runs on polynomially many processors in time that depends polylogarithmically on the sample size but is inversely proportional to a parameter of the learning problem. While Nick's Class as a notion of efficiency has been criticised, e.g., by Kruskal et al. [17], it is the only notion of efficiency that forms a proper complexity class, in the sense of Blum [4]. Additionally, Kruskal et al. [17] suggested to also consider the inefficiency of simulating the parallel algorithm on a single processing unit. We consider this in Appendix A.2, where we also discuss the speed-up [17] using c processing units.

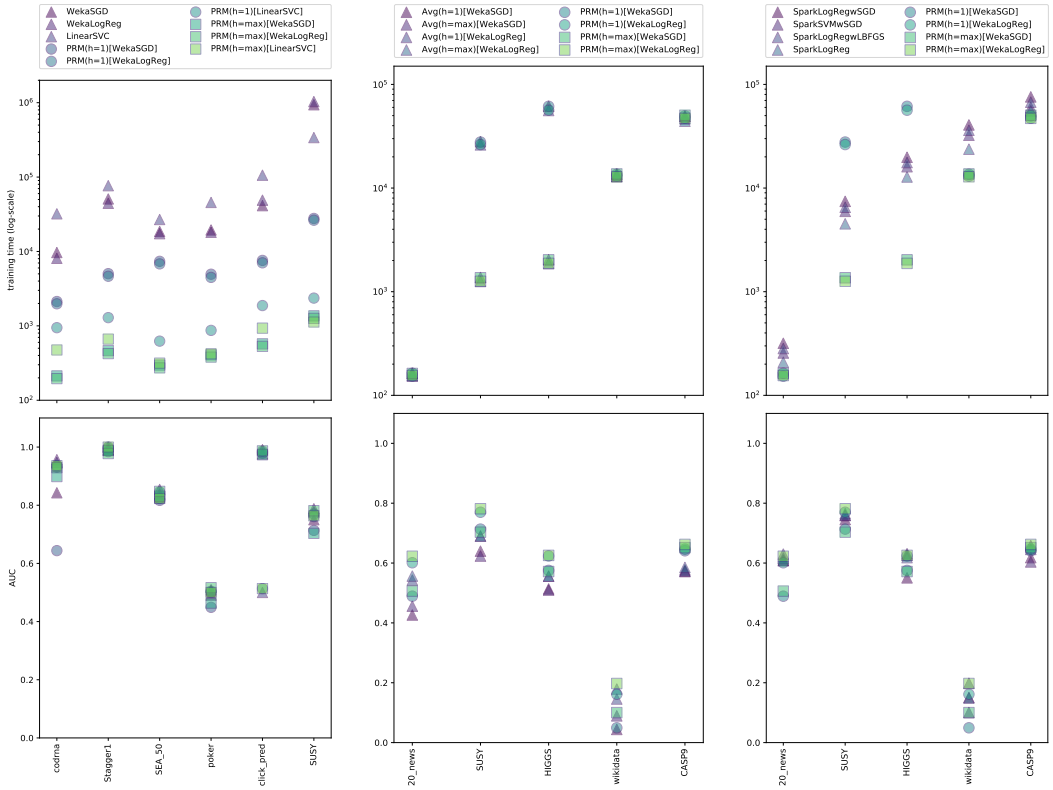
[1] We derive $\alpha_\varepsilon, \beta_\varepsilon$ for hypothesis spaces with finite VC [40] and Rademacher [3] complexity in App. C.2.

4 Empirical Evaluation

This empirical study compares the Radon machine to state-of-the-art parallel machine learning algorithms from the Spark machine learning library [25], as well as the natural baseline of averaging hypotheses instead of calculating their Radon point (denoted averaging-at-the-end). In this study, we use base learning algorithms from WEKA [43] and scikit-learn [29]. We compare the Radon machine to the base learning algorithms on moderately sized datasets, due to scalability limitations of the base learners, and reserve larger datasets for the comparison with parallel learners. The experiments are executed on a Spark cluster (5 worker nodes, 25 processors per node)[2]. In this study, we apply the Radon machine with parameter $h = 1$ and the maximal parameter h (denoted $h = max$) such that each instance of the base learning algorithm is executed on a subset of size at least 100. Averaging-at-the-end uses the same parameter h and executes the base learning algorithm on r^h of subsets, i.e., the same number as the Radon machine with that parameter.

What is the speed-up of our scheme in practice? In Figure 1(a), we compare the Radon machine to its base learners on moderately sized datasets (details on the datasets are provided in Appendix B). There, the Radon machine is is between 80 and around 700-times faster than the base learner, using 150 processors. The speed-up is detailed in Figure 2. On the SUSY dataset

[2]The source code implementation in Spark can be found in the bitbucket repository https://bitbucket.org/Michael_Kamp/radonmachine.



(a) (b) (c)

Figure 1: (a) Runtime (log-scale) and AUC of base learners and their parallelisation using the Radon machine (PRM) for 6 datasets with $N \in [488\,565, 5\,000\,000]$, $d \in [3, 18]$. Each point represents the average runtime (upper part) and AUC (lower part) over 10 folds of a learner—or its parallelisation—on one datasets. (b) Runtime and AUC of the Radon machine compared to the averaging-at-the-end baseline (Avg) on 5 datasets with $N \in [5\,000\,000, 32\,000\,000]$, $d \in [18, 2\,331]$. (c) Runtime and AUC of several Spark machine learning library algorithms and the Radon machine using base learners that are comparable to the Spark algorithms on the same datasets as in Figure 1(b).

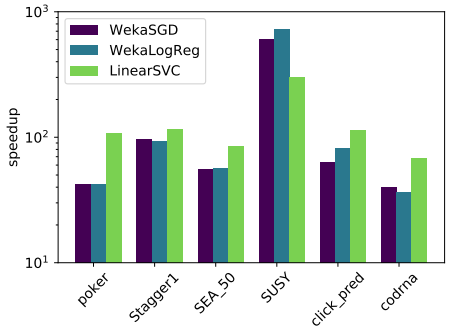


Figure 2: Speed-up (log-scale) of the Radon machine over its base learners per dataset from the same experiment as in Figure 1(a).

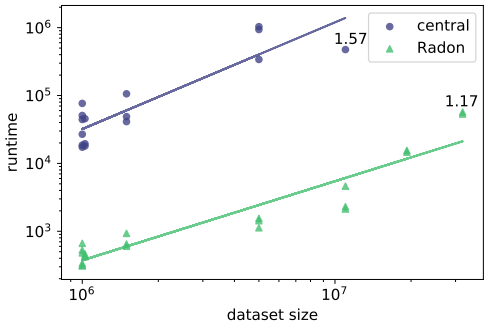


Figure 3: Dependence of the runtime on the dataset size for of the Radon machine compared to its base learners.

(with 5 000 000 instances and 18 features), the Radon machine on 150 processors with $h = 3$ is 721 times faster than its base learning algorithms. At the same time, their practical performances, measured by the area under the ROC curve (AUC) on an independent test dataset, are comparable.

How does the scheme compare to averaging-at-the-end? In Figure 1(b) we compare the runtime and AUC of the parallelisation scheme against the *Avg* baseline. Since averaging is less computationally expensive than calculating the Radon point, the runtimes of the Avg baselines are slightly lower than the ones of the Radon machine. However, compared to the computational complexity of executing the base learner, this advantage becomes negligible. In terms of AUC, the Radon machine outperforms the averaging baseline on all datasets by at least 10%.

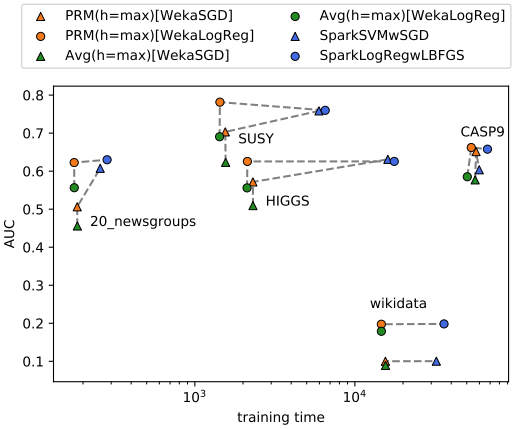


Figure 4: Representation of the results in Figure 1(b) and 1(c) in terms of the trade-off between runtime and AUC for the Radon machine (PRM) and averaging-at-the-end (Avg), both with parameter $h = max$, and parallel machine learning algorithms in Spark.

How does our scheme compare to state-of-the-art Spark machine learning algorithms? We compare the Radon machine to various Spark machine learning algorithms on 5 large datasets. The results in Figure 1(c) indicate that the proposed parallelisation scheme with $h = max$ has a significantly smaller runtime than the Spark algorithms on all datasets. On the SUSY and HIGGS dataset, the Radon machine is one order of magnitude faster than the Spark implementations—here the comparatively small number of features allows for a high level of parallelism. On the CASP9 dataset, the Radon machine is 15% faster than the fastest Spark algorithm. The performance in terms of AUC of the Radon machine is similar to the Spark algorithms. In particular, when using WekaLogReg with $h = max$, the Radon machine outperforms the Spark algorithms in terms of AUC and runtime on the datasets SUSY, wikidata, and CASP9. Details are given in the Appendix B. A summarizing comparison of the parallel approaches in terms of their trade-off between runtime and predictive performance is depicted in Figure 4. Here, results are shown for the Radon machine and averaging-at-the-end with parameter $h = max$ and for the two Spark algorithms most similar to the base learning algorithms. Note that it is unclear, what caused the consistently weak performance of all algorithms on wikidata. Nonetheless, the results show that on all datasets the Radon machine has comparable predictive performance to the Spark algorithms and substantially higher predictive performance than averaging-at-the-end. At the same time, the Radon machine has a runtime comparable to averaging-at-the-end on all datasets, both are substantially faster than the Spark algorithms.

How does the runtime depend on the dataset size in a real-world system? In Figure 3 we compare the runtimes of all base learning algorithms per dataset size to the Radon machines. Results indicate that, while the runtimes of the base learning algorithms depends on the dataset size with an average exponent of 1.57, the runtime of the Radon machine depends on the dataset size with an exponent of only 1.17. This is plausible because with enough processors the generation of weak hypotheses can be done completely in parallel. Moreover, the time for aggregating the hypotheses does not depend on the number of instances in the dataset, but only on the number of iterations and the dimension of the hypothesis space.

How generally applicable is the scheme? As an indication of the general applicability in practice, we apply the scheme to an Scikit-learn implementation of regularised least squares regression [29]. On the dataset *YearPredictionMSD*, regularised least squares regression achieves an RMSE of 12.57, whereas the Radon machine achieved an RMSE of 13.64. At the same time, the Radon machine is 197-times faster. We also compare the Radon machine on a multi-class prediction problem using conditional maximum entropy models. We use the implementation described in Mcdonald et al. [23], who also propose to use averaging-at-the-end for distributed training. We compare the Radon machine to averaging-at-the-end with conditional maximum entropy models on two large multi-class datasets (*drift* and *spoken-arabic-digit*). On average, our scheme performs 4% better with only 0.2% longer runtime. The minimal difference in runtime can be explained—similar to the results in Figure 1(b)—by the smaller complexity of calculating the average instead of the Radon point.

5 Discussion and Related Work

In this paper we provided a step towards answering an open problem: *Is parallel machine learning possible in polylogarithmic time using a polynomial number of processors only?* This question has been posed for half-spaces by Long and Servedio [21] and called "a fundamental open problem about the abilities and limitations of efficient parallel learning algorithms". It relates machine learning to Nick's Class of parallelisable decision problems and its variants [14]. Early theoretical treatments of parallel learning with respect to NC considered *probably approximately correct* (PAC) [5, 38] concept learning. Vitter and Lin [41] introduced the notion of *NC-learnable* for concept classes for which there is an algorithm that outputs a probably approximately correct hypothesis in polylogarithmic time using a polynomial number of processors. In this setting, they proved positive and negative learnability results for a number of concept classes that were previously known to be PAC-learnable in polynomial time. More recently, the special case of learning half spaces in parallel was considered by Long and Servedio [21] who gave an algorithm for this case that runs on polynomially many processors in time that depends polylogarithmically on the size of the instances but is inversely proportional to a parameter of the learning problem. Our paper complements these theoretical treatments of parallel machine learning and provides a provably effective parallelisation scheme for a broad class of regularised risk minimisation algorithms.

Some parallelisation schemes also train learning algorithms on small chunks of data and average the found hypotheses. While this approach has advantages [13, 32], current error bounds do not allow a derivation of polylogarithmic runtime [20, 35, 44] and it has been doubted to have any benefit over learning on a single chunk [34]. Another popular class of parallel learning algorithms is based on stochastic gradient descent, targeting expected risk minimisation directly [34, and references therein]. The best, so far known algorithm in this class [34] is the distributed mini-batch algorithm [10]. This algorithm still runs for a number of rounds inversely proportional to the desired optimisation error, hence not in polylogarithmic time. A more traditional approach is to minimise the *empirical risk*, i.e., an empirical sample-based approximation of the expected risk, using any, deterministic or randomised, optimisation algorithm. This approach relies on generalisation guarantees relating the expected and empirical risk minimisation as well as a guarantee on the optimisation error introduced by the optimisation algorithm. The approach is readily parallelisable by employing available parallel optimisation algorithms [e.g., 6]. It is worth noting that these algorithms solve a harder than necessary optimisation problem and often come with prohibitively high communication cost in distributed settings [34]. Recent results improve over these [22] but cannot achieve polylogarithmic time as the number of iterations depends linearly on the number of processors.

In the following, we want to discuss properties and limitations of the proposed parallelisation scheme. To that end, we address potential questions about the Radon machine.

In the experiments we considered datasets where the number of dimensions is much smaller than the number of instances. **What about high-dimensional models?** The basic version of the parallelisation scheme presented in this paper cannot directly be applied to cases in which the size of the dataset is not at least a multiple of the Radon number of the hypothesis space. For various types of data such as text, this might cause concerns. However, random projections [15] or low-rank approximations [2, 28] can alleviate this problem and are already frequently employed in machine learning. An alternative might be to combine our parallelisation scheme with block coordinate descent [37]. In this case, the scheme can be applied iteratively to subsets of the features.

In the experiments we considered only linear models. **What about non-linear models?** Learning non-linear models causes similar problems to learning high-dimensional ones. In non-parametric methods like kernel methods, for instance, the dimensionality of the optimisation problem is equal to the number of instances, thus prohibiting the application of our parallelisation scheme. However, similar low-rank approximation techniques as described above have been applied with non-linear kernels [11]. Alternatively, novel methods for speeding up the learning process for non-linear models rely on explicitly constructing an embedding in which a linear model can be learned [31]. Using explicitly constructed feature spaces, Radon machines can directly be applied to non-linear models.

We have theoretically analysed our parallelisation scheme for the case that there are enough processors available to find each weak hypothesis on a separate processor. **What if there are less than r^h processors?** The parallelisation scheme can quite naturally be de-parallelised and partially executed in sequence. For the runtime this implies an additional factor of $\max\{1, r^h/c\}$. Thus, the Radon machine can be applied with any number of processors.

The scheme improves the confidence Δ doubly exponentially in its parameter h but for that it requires the weak hypotheses to already achieve a base confidence of $1 - \delta > 1 - 1/2r$. **Is the scheme only applicable in high-confidence domains?** Many application scenarios require high-confidence error bounds, e.g., in the medical domain [27] or in intrusion detection [36]. Apart from these theoretical considerations, in practice our scheme performs comparably to its base learner.

Besides runtime, communication plays an essential role in parallel learning. **What is the communication complexity of the scheme?** As for all aggregation at the end strategies, the overall amount of communication is low compared to periodically communicating schemes. For the parallel aggregation of hypotheses, the scheme requires $\mathcal{O}(r^{h+1})$ messages each containing a single hypothesis of size $\mathcal{O}(d)$. Furthermore, only a fraction of the data has to be transferred to each processor. Our scheme is ideally suited for inherently distributed data.

6 Conclusion and Future Work

We have proposed a parallelisation scheme that is effective, i.e., it speeds up computation through parallelisation while achieving the same hypothesis quality as the base learner. It is a black-box parallelisation in the sense that it is applicable to a wide range of machine learning algorithms and is oblivious to the implementation of these algorithms. Our empirical evaluation shows that in practice substantial speed-ups are achieved by the Radon machine.

Since in a lot of applications data is no longer available as a batch but in the form of data streams, as future work it would be interesting to investigate how the scheme can be applied to distributed data streams. A promising approach is to aggregate hypotheses periodically using the Radon machine, similar to the federated learning approach proposed by McMahan et al. [24]. Another direction for future work is to apply the scheme to general randomized convex optimization algorithms with unobservable target function.

7 Proof of Proposition 2 and Theorem 3

In order to prove Proposition 2 and consecutively Theorem 3, we require the following properties of Radon points and convex functions. We proof these properties for the more general case of quasi-convex functions. Since every convex function is also quasi-convex, the results hold for convex functions as well. A quasi-convex function is defined as follows.

Definition 5. *A function* $\mathcal{Q} : \mathcal{F} \to \mathbb{R}$ *is called* quasi-convex *if all its sublevel sets are convex, i.e.,*

$$\forall \theta \in \mathbb{R} : \{f \in \mathcal{F} | \mathcal{Q}(f) < \theta\} \text{ is convex.}$$

First we give a different characterisation of quasi-convex functions.

Proposition 6. *A function* $\mathcal{Q} : \mathcal{F} \to \mathbb{R}$ *is quasi-convex* $\Leftrightarrow \forall S \subseteq \mathcal{F}, \forall s' \in \langle S \rangle, \exists s \in S : \mathcal{Q}(s) \geq \mathcal{Q}(s')$.

Proof.

- ($\Rightarrow$) Suppose this direction does not hold. Then there is a convex function $\mathcal{Q}$, a set $S \subseteq \mathcal{F}$, and an $s' \in \langle S \rangle$ such that for all $s \in S$ it holds that $\mathcal{Q}(s) < \mathcal{Q}(s')$ (therefore $s' \notin S$). Let $C = \{c \in \mathcal{F} \mid \mathcal{Q}(c) < \mathcal{Q}(s')\}$. As $S \subseteq C$ we also have that $\langle S \rangle \subseteq \langle C \rangle$ which contradicts $\langle S \rangle \ni s' \notin C$.
- ($\Leftarrow$) Suppose this direction does not hold. Then there exists an ε such that $S = \{s \in \mathcal{F} \mid \mathcal{Q}(s) < \varepsilon\}$ is not convex and there is an $s' \in \langle S \rangle \setminus S$. By assumption $\exists s \in S : \mathcal{Q}(s) \geq \mathcal{Q}(s')$. Hence $\mathcal{Q}(s') < \varepsilon$ and we have a contradiction since this would imply $s' \in S$.

□

The next proposition concerns the value of any convex function at a Radon point.

Proposition 7. *For every set* S *with Radon point* $\mathfrak{r}$ *and every quasi-convex function* $\mathcal{Q}$ *it holds that* $|\{s \in S \mid \mathcal{Q}(s) \geq \mathcal{Q}(\mathfrak{r})\}| \geq 2$.

Proof. We show a slightly stronger result: Take any family of pairwise disjoint sets A_i with $\bigcap_i \langle A_i \rangle \neq \emptyset$ and $\mathfrak{r} \in \bigcap_i \langle A_i \rangle$. From proposition 6 follows directly the existence of an $a_i \in A_i$ such that $\mathcal{Q}(a_i) \geq \mathcal{Q}(\mathfrak{r})$. The desired result follows then from $a_i \neq a_j \Leftarrow i \neq j$. □

Using this property, we can proof Proposition 2 and Theorem 3.

Proof of Proposition 2 and Theorem 3. By proposition 7, for any Radon point $\mathfrak{r}$ of a set S there must be two points $a, b \in S$ with $\mathcal{Q}(a), \mathcal{Q}(b) \geq \mathcal{Q}(\mathfrak{r})$. Henceforth, the probability of $\mathcal{Q}(\mathfrak{r}) > \varepsilon$ is smaller or equal than the probability of the pair a, b having $\mathcal{Q}(a), \mathcal{Q}(b) > \varepsilon$. Proposition 2 follows by an application of the union bound on all pairs from S. Repeated application of the proposition proves Theorem 3. □

Acknowledgements

Part of this work was conducted while Mario Boley, Olana Missura, and Thomas Gärtner were at the University of Bonn and partially funded by the German Science Foundation (DFG, under ref. GA 1615/1-1 and GA 1615/2-1). The authors would like to thank Dino Oglic, Graham Hutton, Roderick MacKenzie, and Stefan Wrobel for valuable discussions and comments.

A Theory

A.1 Proof of Theorem 4

In the following, Theorem 4 is proven which states that the parallelisation of a polynomial-time regularised risk minimisation algorithm using the Radon machine has polylogarithmic runtime in the sample complexity on quasi-polynomially $\left(\exp\left((\log M)^{\mathcal{O}(1)}\right)\right)$ many processors.

Proof. We assume the base learning algorithm $\mathcal{L}$ to be a consistent and efficient regularised risk minimisation algorithm on a hypothesis space with finite Radon number. Let $r \in \mathbb{N}$ be the Radon number of the hypothesis space and

$$N_0^{\mathcal{L}}(\varepsilon, \delta) = \left(\alpha_\varepsilon + \beta_\varepsilon \operatorname{ld} \frac{1}{\delta}\right)^k$$

be its sample complexity with $\alpha_\varepsilon, \beta_\varepsilon \geq 0$. In the following, we want to compare the runtime of the Radon machine for achieving an (ε, Δ)-guarantee to the runtime of the sequential application of the base learning algorithm for achieving the same (ε, Δ)-guarantee.

To achieve an (ε, Δ)-guarantee, the Radon machine with parameter $h \in \mathbb{N}$ requires $N = nr^h$ examples (i.e., with r^h processing units), where n denotes the size of the data subset available to each parallel instance of the base learning algorithm. Since $\Delta = (r\delta)^{2^h}$, each base learning algorithm needs to achieve an (ε, δ)-guarantee and thus requires

$$n = \left\lceil N_0^{\mathcal{L}}(\varepsilon, \delta) \right\rceil \leq \left(\alpha_\varepsilon + \beta_\varepsilon \operatorname{ld} \frac{1}{\delta}\right)^k + 1 \tag{8}$$

examples (here, ld denotes the logarithm with basis 2). The sequential application of the base learning algorithm requires at least (cf. Equation 6 in Section 3)

$$M = \left\lceil N_0^{\mathcal{L}}(\varepsilon, \Delta) \right\rceil = \left\lceil \left(\alpha_\varepsilon \ + \ 2^h \cdot \beta_\varepsilon \operatorname{ld} \frac{1}{r\delta}\right)^k \right\rceil \ . \tag{9}$$

Solving Equation 8 for δ yields

$$\delta \leq 2^{-\frac{(n-1)^{\frac{1}{k}} - \alpha_\varepsilon}{\beta_\varepsilon}} \ .$$

By inserting this into Equation 9 we obtain

$$M \geq \left\lceil \left(\alpha_\varepsilon - 2^h \alpha_\varepsilon + 2^h \left((n-1)^{\frac{1}{k}} - \beta_\varepsilon \operatorname{ld} r\right)\right)^k \right\rceil \in \mathcal{O}\left(2^h \left(n - \operatorname{ld} r\right)\right) \ . \tag{10}$$

Note that we additionally assume that $M \geq 2^{k\beta_\varepsilon(\alpha_\varepsilon+1)}$.

In the following, we show that for the choice of

$$h = \left\lceil \frac{1}{k} \left(\operatorname{ld} M - \operatorname{ld} \operatorname{ld} M\right) \right\rceil \ , \tag{11}$$

the runtime of the Radon machine is polylogarithmic in M, i.e., polylogarithmic in the number of examples the sequential application of the base learner requires to achieve an (ε, Δ)-guarantee. For that, the Radon machine requires quasi-polynomially many processors in M. Note that the Radon machine processes $N \geq M$ many samples to achieve that (ε, Δ)-guarantee, which is more than the sequential application of the base learner requires with $N \in \mathcal{O}\left(r^h / 2^{hk} M\right)$.

Thus, we need to express the runtime of the Radon machine, which is

$$T_{\mathcal{R},h}(N) = T_{\mathcal{L}}\left(\frac{N}{r^h}\right) + r^3 \log_r r^h = T_{\mathcal{L}}(n) + r^3 \log_r r^h \ ,$$

in terms of M instead of N. First, we express n in terms of M, by solving Equation 10 for n which yields

$$n \leq \left(\left(\alpha_\varepsilon \left(1 - \frac{1}{2^h}\right) + \beta_\varepsilon \operatorname{ld} r + \frac{1}{2^h} M^{\frac{1}{k}}\right)^k + 1\right) \in \mathcal{O}\left(\log_2^k r + \frac{1}{2^{hk}} M\right) \ . \tag{12}$$

Since $\mathcal{L}$ is efficient, $T_{\mathcal{L}}(n) \in \mathcal{O}(n^\kappa)$ and thus the runtime of the Radon machine in terms of M, denoted $T_{\mathcal{R}}^M$, is

$$T_{\mathcal{R}}^M = T_{\mathcal{L}}(n) + r^3 \log_r r^h \in \mathcal{O}\left(\left(\log_2^k r + \frac{1}{2^{hk}} M\right)^\kappa + r^3 \operatorname{ld} r^h\right) .$$

Inserting h as in Equation 11 yields

$$\begin{aligned}\left(\log_2^k r + \frac{1}{2^{hk}} M\right)^\kappa + r^3 \operatorname{ld} \frac{M}{\operatorname{ld} M} &= \left(\log_2^k r + \frac{M}{2^{k \frac{1}{k} \operatorname{ld} \frac{M}{\operatorname{ld} M}}}\right)^\kappa + r^3 \operatorname{ld} \frac{M}{\operatorname{ld} M} \\ &= \left(\log_2^k r + \frac{M}{\frac{M}{\operatorname{ld} M}}\right)^\kappa + r^3 \operatorname{ld} \frac{M}{\operatorname{ld} M} \\ &= \left(\log_2^k r + \operatorname{ld} M\right)^\kappa + r^3 \operatorname{ld} \frac{M}{\operatorname{ld} M} .\end{aligned}$$

This shows that

$$T_{\mathcal{R}}^M \in \mathcal{O}\left(\operatorname{ld}^\kappa M + \operatorname{ld}^{k\kappa} r + r^3 \operatorname{ld} M\right) .$$

Thus, the runtime of the Radon machine to achieve an (ε, Δ)-guarantee in terms of M (i.e., the number of samples required by the sequential application of the base learning algorithm) is in $\mathcal{O}\left(\operatorname{ld}^\kappa M + \operatorname{ld}^{k\kappa} r + r^3 \operatorname{ld} M\right)$ and thus polylogarithmic in M.

We now determine the number of processing units $c = r^h$ in terms of M. For that, observe that h as in Equation 11 can be expressed as

$$h = \left\lceil \frac{1}{k} (\operatorname{ld} M - \operatorname{ld} \operatorname{ld} M) \right\rceil = \left\lceil \frac{1}{k} \left(\operatorname{ld} \frac{M}{\operatorname{ld} M} \right) \right\rceil = \left\lceil \frac{\operatorname{ld} r}{k} \log_r \frac{M}{\operatorname{ld} M} \right\rceil ,$$

and thus the number of processing units is

$$c = r^h \in \mathcal{O}\left(2^{\operatorname{ld} M \operatorname{ld} r}\right) .$$

Note that r cannot be treated as a constant here, because it depends on the dimension of the model space, which in turn usually depends on the dimension of the input space. Thus, r may depend on the input size as well.

It remains to be shown that the Radon machine is applicable, i.e., that the sample size with respect to M is large enough so that each base learner achieves a minimum confidence of $\delta \leq 1/2r$. Using Equation 9 this implies that

$$M \geq \left(\alpha_\varepsilon + 2^h \cdot \beta_\varepsilon \operatorname{ld} \frac{1}{r^{\frac{1}{2r}}}\right)^k = \left(\alpha_\varepsilon + 2^h \beta_\varepsilon\right)^k = \left(\alpha_\varepsilon + \left(\frac{M}{\operatorname{ld} M}\right)^{\frac{1}{k}} \beta_\varepsilon\right)^k . \tag{13}$$

Since we assume $M \geq 2^{k\beta_\varepsilon(\alpha_\varepsilon+1)} \geq 2^{k\beta_\varepsilon}$ we have

$$\left(\frac{M}{\operatorname{ld} M}\right)^{\frac{1}{k}} \beta_\varepsilon \geq 1 ,$$

and thus we can upper bound the right hand side of Equation 13 by

$$\begin{aligned}\left(\alpha_\varepsilon + \left(\frac{M}{\operatorname{ld} M}\right)^{\frac{1}{k}} \beta_\varepsilon\right)^k &= \left(\left(\frac{M}{\operatorname{ld} M}\right)^{\frac{1}{k}} \beta_\varepsilon \left(\frac{\alpha_\varepsilon}{\left(\frac{M}{\operatorname{ld} M}\right)^{\frac{1}{k}} \beta_\varepsilon} + 1\right)\right)^k \\ &\leq \frac{M}{\operatorname{ld} M} \beta_\varepsilon^k \left(\alpha_\varepsilon + 1\right)^k \overset{!}{\leq} M .\end{aligned}$$

Thus, we require $\operatorname{ld} M \geq \beta_\varepsilon^k \left(\alpha_\varepsilon + 1\right)^k$ which holds since we assume $M \geq 2^{k\beta_\varepsilon(\alpha_\varepsilon+1)}$.

□

A.2 Analysis of the Speedup of the Radon machine

In this section, we analyse the speed-up of the Radon machine over the sequential execution of the base learning algorithm when both achieve the same (ε, Δ)-guarantee. For that, recall that the sample complexity of the base algorithm for a given $\varepsilon > 0$, $0 < \Delta < 1$ is

$$N_0^{\mathcal{L}}(\varepsilon, \Delta) = \left(\alpha_\varepsilon + \beta_\varepsilon \log_2 \frac{1}{\Delta}\right)^k \quad .$$

Long and Servedio [21] point out that, since the behaviour of parallelisations with respect to ε has been sufficiently established by Freund [12], it suffices to analyse the sample complexity with respect to Δ. Thus, we ignore the constants α_ε and β_ε and assume the sample complexity can be expressed as

$$N_0^{\mathcal{L}}(\Delta) = \left(\gamma \log_2 \frac{1}{\Delta}\right)^k \quad ,$$

for $\gamma \geq (\alpha_\varepsilon + 1)\beta_\varepsilon$. Similar to Kruskal et al. [17], we assume the base algorithm to have a runtime polynomial in N, i.e., $T_{\mathcal{L}} \in \Theta\left(n^\kappa\right)$. For simplicity, we furthermore assume the runtime can be expressed as $T_{\mathcal{L}}(n) = (\rho n)^\kappa$ for some constant $\rho \in \mathbb{R}_+$. Inserting the sample size yields

$$T_{\mathcal{L}} = \rho^\kappa \left(\gamma \log_2 \frac{1}{\Delta}\right)^{k\kappa} \tag{14}$$

The Radon machine runs $\mathcal{L}$ in parallel on c processors to obtain r^h weak hypotheses with (ε, δ)-guarantee. It then combines the obtained solutions h times—level-wise in parallel—calculating the Radon point (which takes time r^3). It follows that its runtime can be expressed as

$$T_{\mathcal{R}} = \frac{r^h}{c}\left(\rho^\kappa \left(\gamma \log_2 \frac{1}{\delta}\right)^{k\kappa}\right) + r^3 \sum_{i=1}^{h} \left\lceil \frac{r^i}{c} \right\rceil \quad . \tag{15}$$

We now provide a lower bound on the speed-up for arbitrary $h \in \mathbb{N}$, $c \in [2, r^h]$.

Proposition 8. *Given a polynomial time consistent regularised risk minimisation algorithm $\mathcal{L}$ with runtime as in Equation 14, the Radon machine run with parameter $h \in \mathbb{N}$ on $c \in [2, r^h]$ processors, the ratio of the runtime of the base learner over on the runtime of the Radon machine, denoted the speed-up [17]*

$$\sigma(h, c) = \frac{T_{\mathcal{L}}}{T_{\mathcal{R}}} \quad ,$$

is larger than

$$\sigma(h, c) \geq \left(2^h\right)^{k\kappa} \frac{c}{h r^h} \left((1 + \log_2 r)^{k\kappa} + \frac{r^3}{\rho^\kappa \gamma^{k\kappa}}\right)^{-1} \quad .$$

In order to proof Proposition 8, first some technical results need to be established.

Lemma 9. *For $0 < \delta < 1/2r$, $h, r \in \mathbb{N}$ and $\Delta = (r\delta)^{2^h}$ it holds that*

$$\frac{\log_2 \frac{1}{\delta}}{\log_2 \frac{1}{\Delta}} = \frac{1}{2^h} + \frac{\log_2 r}{\log_2 \frac{1}{\Delta}} \quad .$$

Proof. From $\Delta = (r\delta)^{2^h}$ it follows that $\delta = 1/r \Delta^{1/2^h}$. Using this yields

$$\frac{\log_2 \frac{1}{\delta}}{\log_2 \frac{1}{\Delta}} = \frac{\log_2 \frac{1}{\frac{1}{r}\Delta^{\frac{1}{2^h}}}}{\log_2 \frac{1}{\Delta}} = \frac{\log_2 r + \log_2\left(\left(\frac{1}{\Delta}\right)^{\frac{1}{2^h}}\right)}{\log_2 \frac{1}{\Delta}} = \frac{\log_2 r + \frac{1}{2^h}\log_2 \frac{1}{\Delta}}{\log_2 \frac{1}{\Delta}} = \frac{\log_2 r}{\log_2 \frac{1}{\Delta}} + \frac{1}{2^h} \quad .$$

□

Lemma 10. *For $0 < \delta < 1/2r$, $h, r \in \mathbb{N}$ and $\Delta = (r\delta)^{2^h}$ it holds that*

$$\frac{1}{\log_2 \frac{1}{\Delta}} \leq \frac{1}{2^h} \quad .$$

Proof. From $\Delta = (r\delta)^{2^h}$ and $\delta < 1/2r$ it follows that $\Delta \leq (1/2)^{2^h}$. Using this yields

$$\frac{1}{\log_2 \frac{1}{\Delta}} \leq \frac{1}{\log_2 \frac{1}{\left(\frac{1}{2}\right)^{2^h}}} = \frac{1}{2^h} \ .$$

□

With these two lemmata, we can proof Proposition 8:

Proof of Proposition 8. Inserting Equation 14 and Equation 15 into the definition of speed-up yields

$$\begin{aligned}
\sigma(h,c) =& \frac{\left(\rho\left(\gamma \log_2 \frac{1}{\Delta}\right)^k\right)^\kappa}{\frac{r^h}{c}\left(\rho\left(\gamma \log_2 \frac{1}{\delta}\right)^k\right)^\kappa + r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil} \\
=& \frac{\rho^\kappa \gamma^{k\kappa} \left(\log_2 \frac{1}{\Delta}\right)^{k\kappa}}{\frac{r^h}{c}\rho^\kappa \gamma^{k\kappa} \left(\log_2 \frac{1}{\delta}\right)^{k\kappa} + r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil} \\
=& \left(\frac{\frac{r^h}{c}\rho^\kappa \gamma^{k\kappa} \left(\log_2 \frac{1}{\delta}\right)^{k\kappa} + r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil}{\rho^\kappa \gamma^{k\kappa} \left(\log_2 \frac{1}{\Delta}\right)^{k\kappa}} \right)^{-1} \\
=& \left(\frac{r^h}{c} \left(\frac{\log_2 \frac{1}{\delta}}{\log_2 \frac{1}{\Delta}} \right)^{k\kappa} + \frac{r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil}{\rho^\kappa \gamma^{k\kappa} \left(\log_2 \frac{1}{\Delta}\right)^{k\kappa}} \right)^{-1} \\
\underbrace{=}_{Lm.\ 9}& \left(\frac{r^h}{c} \left(\frac{1}{2^h} + \frac{\log_2 r}{\log_2 \frac{1}{\Delta}} \right)^{k\kappa} + \frac{r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil}{\rho^\kappa \gamma^{k\kappa} \left(\log_2 \frac{1}{\Delta}\right)^{k\kappa}} \right)^{-1} \\
\underbrace{\geq}_{Lm.\ 10}& \left(\frac{r^h}{c} \left(\frac{1}{2^h} \right)^{k\kappa} (1+\log_2 r)^{k\kappa} + \frac{r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil}{\rho^\kappa \gamma^{k\kappa} \left(2^h\right)^{k\kappa}} \right)^{-1} \\
=& \left(\frac{r^h}{c} \left(\frac{1}{2^h} \right)^{k\kappa} (1+\log_2 r)^{k\kappa} + \left(\frac{1}{2^h} \right)^{k\kappa} \frac{r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil}{\rho^\kappa \gamma^{k\kappa}} \right)^{-1} \\
=& \left(2^h\right)^{k\kappa} \left(\frac{r^h}{c} (1+\log_2 r)^{k\kappa} + \frac{r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil}{\rho^\kappa \gamma^{k\kappa}} \right)^{-1}
\end{aligned}$$

The iterated computation of the Radon point parallelised on c processors takes time $r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil$. This can be upper bounded by

$$r^3 \sum_{i=1}^h \left\lceil \frac{r^i}{c} \right\rceil \leq r^3 \sum_{i=1}^h \frac{r^h}{c} = r^3 h \frac{r^h}{c} \ .$$

Inserting this in the inequality above yields

$$\begin{aligned}
\sigma(h,c) &\geq \left(2^h\right)^{k\kappa} \left(\frac{r^h}{c} (1+\log_2 r)^{k\kappa} + \frac{r^3 h \frac{r^h}{c}}{\rho^\kappa \gamma^{k\kappa}} \right)^{-1} \\
&= \left(2^h\right)^{k\kappa} \frac{c}{r^h} \left((1+\log_2 r)^{k\kappa} + \frac{r^3 h}{\rho^\kappa \gamma^{k\kappa}} \right)^{-1} \\
&\geq \left(2^h\right)^{k\kappa} \frac{c}{r^h} \left((1+\log_2 r)^{k\kappa} h + \frac{r^3 h}{\rho^\kappa \gamma^{k\kappa}} \right)^{-1} \\
&= \left(2^h\right)^{k\kappa} \frac{c}{r^h} \frac{1}{h} \left((1+\log_2 r)^{k\kappa} + \frac{r^3}{\rho^\kappa \gamma^{k\kappa}} \right)^{-1}
\end{aligned}$$

□

Note that this analysis holds for polynomial-time algorithms. For algorithms with a higher runtime complexity, the speed-up is even larger[3].

In the following, we analyse the inefficiency [17] of the Radon machine, i.e., the ratio between the total number of operations executed by all processors, and the work of the sequential algorithm.

Proposition 11. *The Radon machine with a consistent and efficient regularised risk minimisation algorithm on a hypothesis space with finite Radon number has quasi-polynomial inefficiency.*

Proof. Let $\mathcal{L}$ be a consistent and efficient regularised risk minimisation algorithm on a hypothesis space with finite Radon number $r \in \mathbb{N}$. Since $\mathcal{L}$ is efficient, its runtime $T_{\mathcal{L}}(M)$ is in $\mathcal{O}(M^\kappa)$. From the proof of Theorem 4 follows that, when choosing $h = \left\lceil \frac{1}{k} \left(\operatorname{ld} M - \operatorname{ld} \operatorname{ld} M \right) \right\rceil$ the Radon machine has a runtime in $\mathcal{O}\left(\operatorname{ld}^\kappa M + \operatorname{ld}^{k\kappa} r + r^3 \operatorname{ld} M \right)$ using $\mathcal{O}\left(M^{\operatorname{ld} r} \right)$ processing units. The inefficiency of the Radon machine then is in

$$\mathcal{O}\left(\frac{M^{\operatorname{ld} r} \left(\operatorname{ld}^\kappa M + \operatorname{ld}^{k\kappa} r + r^3 \operatorname{ld} M \right)}{M^\kappa} \right) \in \mathcal{O}\left(M^{\operatorname{ld} r - \kappa} \left(\operatorname{ld}^\kappa M + \operatorname{ld}^{k\kappa} r + r^3 \operatorname{ld} M \right) \right) \quad .$$

Thus, the inefficiency of the Radon machine is quasi-polynomially bounded or, for short, it has quasi-polynomial inefficiency. □

B Experiments

This section provides additional details on the experiments conducted. All experiments are performed on a Spark cluster with a master node, 5 worker nodes, 25 processors and 64GB of RAM per node. The Radon machine is applied with parameter $h = 1$ and with the maximal h for a given dataset. Recall, that the number of iterations h is limited by the dataset size (i.e., number of instances) and the Radon number of the hypothesis space, since the dataset is partitioned into r^h parts of size n. Thus, given a data set of size N, the maximal h is given by

$$h_{\text{max}} = \left\lfloor \log_r \frac{N}{n_{\text{min}}} \right\rfloor \quad ,$$

where n_{min} denotes the minimum size of the local subset of data that each instance of the base learner is executed on. The experiments have been carried out with $n_{\text{min}} = 100$. As discussed in Section 5, if r^h is larger than the actual number of processing units, some instances of the base learner are executed sequentially.

As base learning algorithms we use the WEKA [43] implementation of Stochastic Gradient Descent (*WekaSGD*), and Logistic Regression (*WekaLogReg*), as well as a the Scikit-learn [29] implementation of the linear support vector machine (*LinearSVM*) with pyspark. The paralellisation of a base learner using the Radon machine is denoted *PRM(h=?)[<base learner>]*.

We compare the Radon machine to the natural baseline of aggregating hypotheses by calculating their average, denoted averaging-at-the-end (*Avg(h=?)[<base learner>]*). Given a parameter $h \in \mathbb{N}$, averaging-at-the-end executes the base learning algorithm on r^h subsets of the data, i.e., on the same number of subsets as the Radon machine. Accordingly, the runtime for obtaining the set of hypotheses is similar, but the time for aggregating the models is shorter, since averaging is less computationally expensive than calculating the Radon point.

We compare the Radon machine to parallel machine learning algorithms from the Spark machine learning library, as well. That is, we compare it to SparkMLLibLogisticRegressionWithLBFGS (*SparkLogRegwLBFGS*), SparkMLLibLogisticRegressionWithSGD (*SparkLogRegwSGD*), SparkMLLibSVMWithSGD (*SparkSVMwSGD*), and SparkMLLogisticRegression (*SparkLogReg*).

The properties of the datasets used in the empirical evaluation are presented in Table 1. Datasets have been acquired from OpenML [39], the UCI machine learning repository [19], and Big Data

[3] For the special case of exponential time algorithms, the speed-up is superlinear with the number of cores.

Name	Instances	Dimensions	Output
click_prediction	1 496 391	11	$\mathcal{Y} = \{-1, 1\}$
poker	1 025 010	10	$\mathcal{Y} = \{-1, 1\}$
SUSY	5 000 000	18	$\mathcal{Y} = \{-1, 1\}$
Stagger1	1 000 000	9	$\mathcal{Y} = \{-1, 1\}$
HIGGS	11 000 000	28	$\mathcal{Y} = \{-1, 1\}$
SEA_50	1 000 000	3	$\mathcal{Y} = \{-1, 1\}$
codrna	488 565	8	$\mathcal{Y} = \{-1, 1\}$
CASP9	31 993 555	631	$\mathcal{Y} = \{-1, 1\}$
wikidata	19 254 100	2331	$\mathcal{Y} = \{-1, 1\}$
20_newsgroups	399 940	1002	$\mathcal{Y} = \{-1, 1\}$
YearPredictionMSD	515 345	90	$\mathcal{Y} \subseteq \mathbb{R}$
drift	13 991	90	$\mathcal{Y} = \{1, \ldots, 89\}$
spoken-arabic-digit	263 256	15	$\mathcal{Y} = \{1, \ldots, 10\}$

Table 1: Description of the datasets used in our experiments.

competition of the ECDBL'14 workshop[4]. Experiments on moderately sized datasets—on which we compared the Radon machine to the base learners executed on the entire dataset have been conducted on the datasets click_prediction, poker, SUSY, Stagger1, SEA_50, and codrna. The comparison of Radon machine and Spark ML learners has been executed on the datasets CASP9, HIGGS, wikidata, 20_newsgroups, and SUSY. The regression experiment was conducted using the YearPredictionMSD dataset, multiclass-prediction experiments using the drift, and spoken-arabic-digit datasets.

In the following, we provide more details on the experiments presented in Figures 1(a), 1(b), and 1(c) in Section 4. In particular, we analyse the trade-off between training time and AUC per dataset.

Figure 5 shows the trade-off between training time and AUC for base learning algorithms and their parallelisation using the Radon machine. It confirms that the training time for the Radon machine is orders of magnitude smaller than the base learners on all datasets. Moreover, the training time is substantially smaller for the Radon machine with maximal height ($h = max$), compared to a height of 1. In terms of AUC, the performance of the parallelisation is comparable to the base learner for WekaLogReg and LinearSVC on all datasets. For the base learner WekaSGD, its parallelisation with the Radon machine only has substantially lower AUC on the dataset codrna with parameter $h = 1$.

Figure 6 presents the trade-off for the Radon machinecompared to the AvergingAtTheEnd-baseline. It confirms that the Radon machine has substantially higher AUC than a parallelisation of the same base learning algorithm using the averaging baseline. The schemes only differ in the aggregation operation at the end, so that the difference in training time follows from the faster computation of the average, compared to the iterated Radon point computation.

In Figure 7, the trade-off between training time and AUC of the Radon machine compared to the Spark learners is plotted. While it confirms that the Radon machine is always favourable in terms of training time, in terms of AUC the results are mixed. For the base learner WekaLogReg, its parallelisation is always among the best in terms of AUC. The parallelisation of WekaSGD, however, has worse performance than the Spark learners on 2 out of 5 datasets. It also confirms that for the datasets SUSY and HIGGS, the runtime of the Radon machine with $h = 1$ is substantially larger than for $h = max$. Thus, for the best performance in terms of runtime and AUC, the height should be maximal.

[4]Big Data Competition 2014: http://cruncher.ncl.ac.uk/bdcomp/

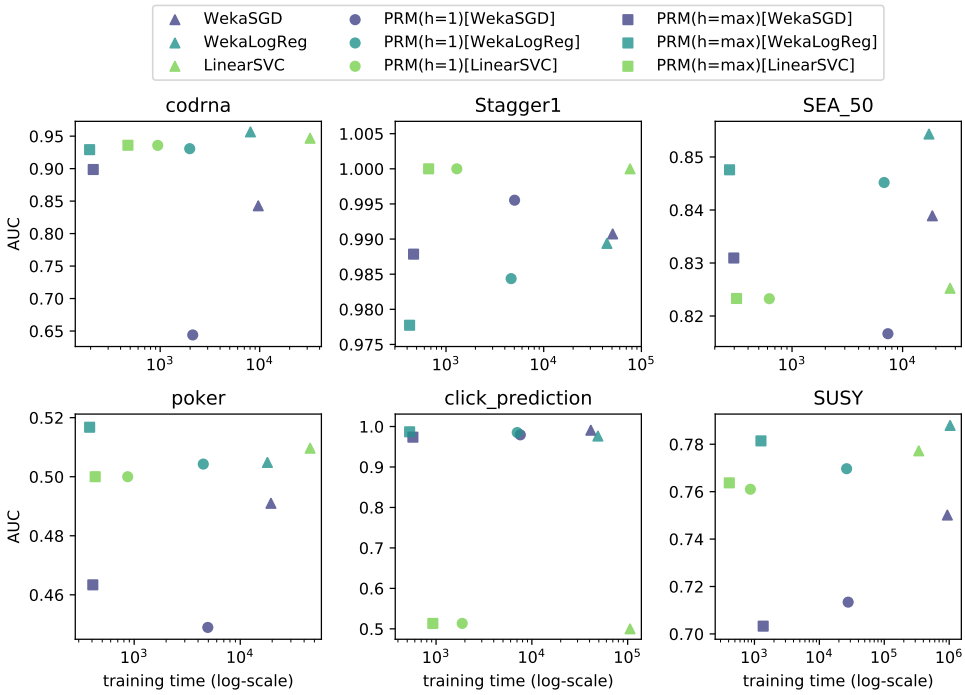


Figure 5: AUC vs. training time for base learning algorithms and their parallelisation with the Radon machine per dataset from the same experiment as in Figure 1(a).

In the following, we want to investigate these more closely. For that, we provide the training times and AUCs in detail in Table 2. As mentioned above, the Radon machine using WekaLogReg as base learner has better runtime than all Spark algorithms. At the same time, this version of the Radon machine outperforms the Spark algorithms in terms of AUC on all datasets but 20_newsgroups—there it is 2.2% worse than the best Spark algorithm. In particular, on the largest dataset in the experiments—the CASP9 dataset with 32 million instances and 631 features—the Radon machine is 15% faster and 2.6% better in terms of AUC than the best Spark algorithm.

Note that for HIGGS and SUSY, the Radon machine with $h = 1$ is an order of magnitude slower than with $h = max$ as well as the Spark algorithms. This follows from the low degree of parallelisation, since for $h = 1$ only 20 (for SUSY), respectively 30 (for HIGGS) hypotheses have to be generated. Thus, only 20, or 30 of the 150 available processors are used in parallel. At the same time, the amount of data each processor has to process is orders of magnitude larger than for $h = max$.

Dataset								
	Runtime							
	SparkLogReg wSGD	**SparkSVM wSGD**	**PRM(h=1) [WekaSGD]**	**PRM(h=max) [WekaSGD]**	**SparkLogReg wLBFGS**	**SparkLogReg**	**PRM(h=1) [WekaLogReg]**	**PRM(h=max) [WekaLogReg]**
20_newsgroups	317.7	256.2	163.4	162.5	282.9	208.5	**152.8**	155.4
SUSY	7 439.5	5 961.8	27 781.6	1 363.7	6 526.3	4 516.8	26 299.6	**1259.7**
HIGGS	19 815.1	16 071.9	61 429.5	2 029.7	17 617.4	12 783.6	56 394.2	**1876.2**
wikidata	40 645.8	32 288.5	13 575.7	13 677.3	36 060.1	23 702.0	13 039.5	**12845.5**
CASP9	75 782.4	59 864.7	49 711.5	50 430.6	67 367.3	55 523.5	47 085.1	**47070.1**
	AUC							
20_newsgroups	0.6098	0.6075	0.4893	0.5063	**0.63**	0.6165	0.601	0.6226
SUSY	0.7454	0.7585	0.7134	0.7033	0.76	0.7652	0.7697	**0.7814**
HIGGS	0.5506	**0.631**	0.5753	0.5717	0.6257	0.6181	0.6237	0.6256
wikidata	0.1505	0.1004	0.0494	0.1002	0.1983	0.1489	0.1615	**0.1974**
CASP9	0.6181	0.6037	0.641	0.6514	0.6579	0.6454	0.6464	**0.6622**

Table 2: Runtime and AUC of Spark machine learning library algorithms and the Radon machine using WekaSGD and WekaLogReg as base learners. The results, reported for each dataset, are the average over all folds in a 10-fold cross-validation. These results correspond to the ones presented in Figure 1(c) in Section 4.

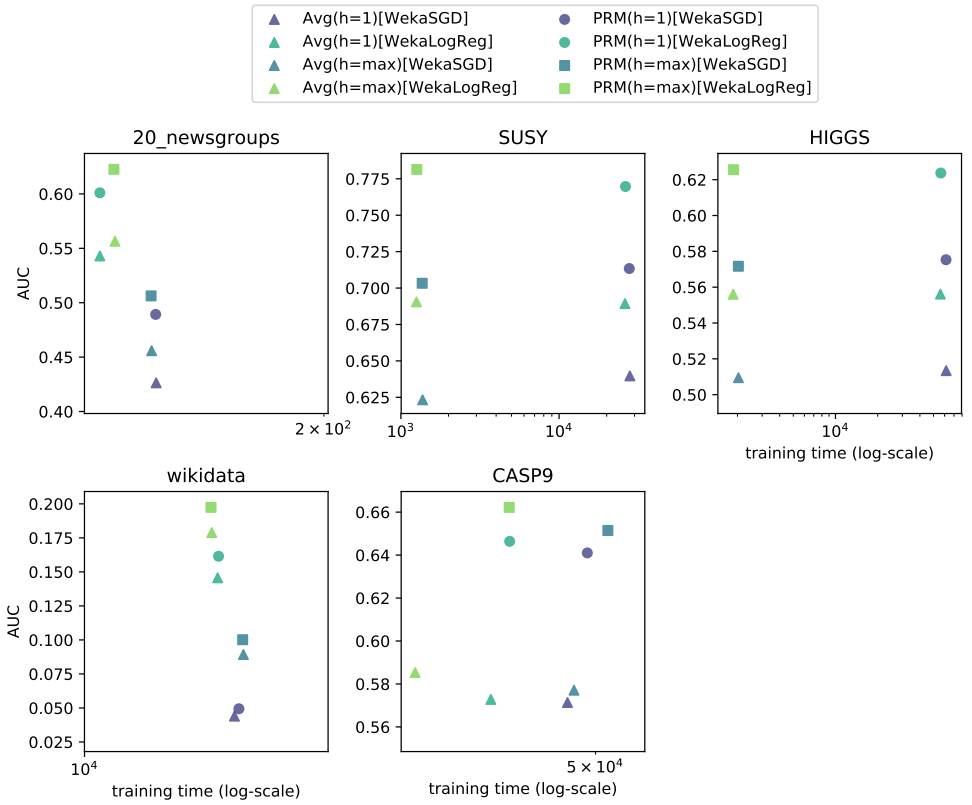


Figure 6: AUC vs. training time for the parallelisation of base learning algorithms using the averaging baseline and the Radon machine per dataset from the same experiment as in Figure 1(b).

For the above experiments we assume that the data is already distributed over the nodes in the cluster so that it can directly be processed by the Radon machine. When loading data in Spark, this data is distributed over the worker nodes in subsetsbut not necessarily in r^h subsets. In Spark, distributed data is organised in partitions, where each partition corresponds to the subset of data available to one instance of the base learning algorithm. In order to apply the Radon machine to a dataset within the Spark framework, the data needs to re-distributed and partitioned into r^h partitions which is achieved by a method called repartition. In the experiments in Section 4, we assume that the data is already partitioned to make a fair comparison to the Spark learning algorithms which do not require repartitioning. Figure 8(a) illustrates the time required for repartitioning a dataset in contrast to the runtime of the Radon machine. Unfortunately, repartitioning in Spark always includes a complete shuffling of the data, requiring communication to redistribute the dataset. This is rather inefficient in our context. Nonetheless, the time required for repartitioning is small compared to the overall runtime—in the worst case it takes 14% of the runtime of the Radon machine. Still, taking into account the time for repartitioning the data shrinks the runtime advantage of the proposed scheme over the Spark algorithms. Figure 8(b) shows the runtimes of the Spark algorithms compared to the Radon machine—similar to Figure 1(c) in Section 4—but with the time required for repartitioning the data added to the runtime of the *Radon machines*. The Radon machine with $h = max$ remains superior to the Spark algorithms in terms of runtime.

C Practical Aspects

C.1 Radon Point Construction

In the following, a simple construction is given for a system of linear equations with which a Radon point of a set can be determined. In his main theorem, Radon [30] gives the following construction of a Radon point for a set $S = \{s_1, ..., s_r\} \subseteq \mathbb{R}^d$. Find a non-zero solution $\lambda \in \mathbb{R}^{|S|}$ for the

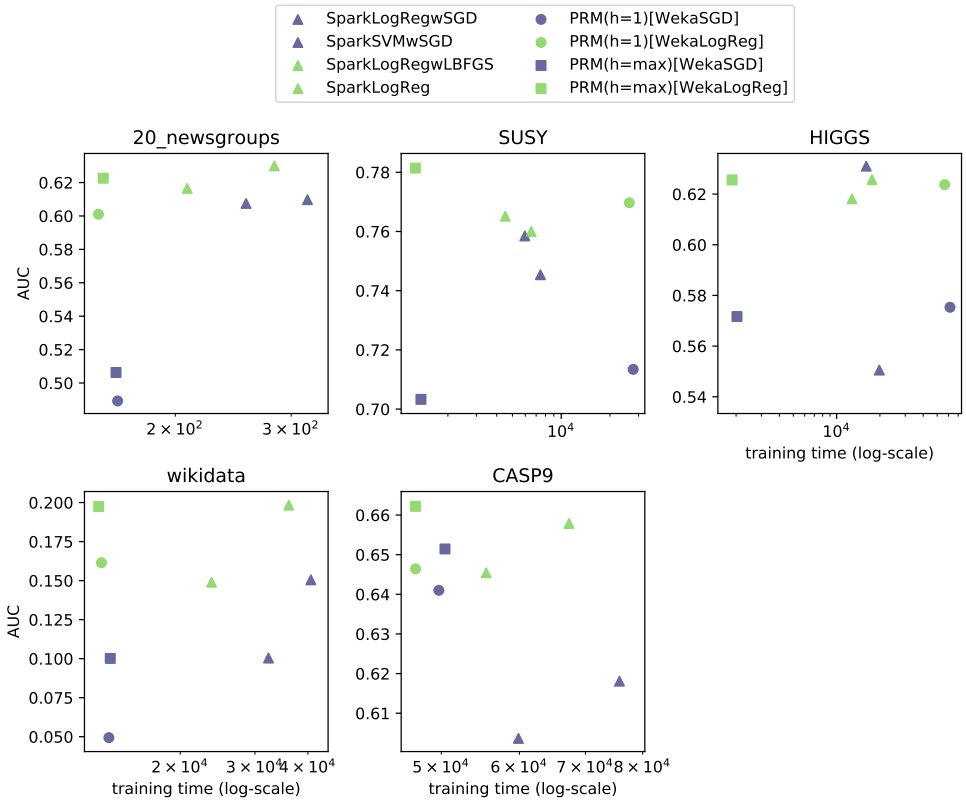
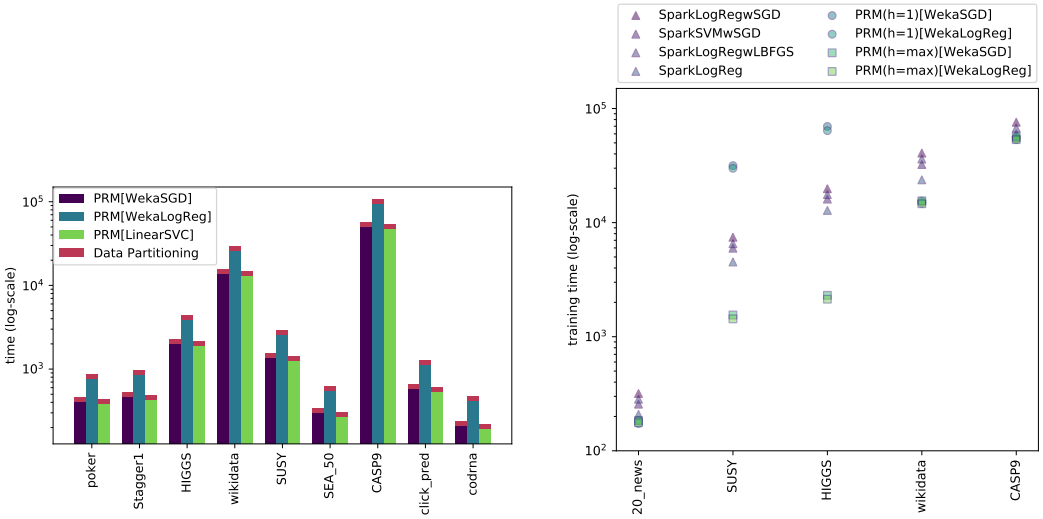


Figure 7: AUC vs. training time for Spark learners and parallelisations of comparable base learners with the Radon machine per dataset from the same experiment as in Figure 1(c).



(a) (b)

Figure 8: (a) Runtime of the Radon machine together with the time required for repartitioning the data to fit the parallelisation scheme. (b) Runtime and AUC of several Spark machine learning library algorithms and the Radon machine including the time required for repartitioning the data before training.

following linear equations.

$$\sum_{i=1}^{r} \lambda_i s_i = (0, \ldots, 0) \ , \ \sum_{i=1}^{r} \lambda_i = 0$$

Such a solution exists, since $|S| > d+1$ implies that S is linearly dependent. Then, let I, J be index sets such that for all $i \in I : \lambda_i \geq 0$ and for all $j \in J : \lambda_j < 0$. Then a Radon point is defined by

$$\mathfrak{r}(\lambda) = \sum_{i \in I} \frac{\lambda_i}{\Lambda} s_i = \sum_{j \in J} \frac{\lambda_j}{\Lambda} s_j \ ,$$

where $\Lambda = \sum_{i \in I} \lambda_i = -\sum_{j \in J} \lambda_j$. Any solution to this linear system of equations is a Radon point. The equation system can be solved in time r^3. By setting the first element of λ to one, we obtain a unique solution of the system of linear equations. Using this solution λ, we define the Radon point of a set S as $\mathfrak{r}(S) = \mathfrak{r}(\lambda)$ in order to resolve ambiguity.

C.2 Consistency Results for Empirical Risk Minimisation

In this section we provide some technical results on the consistency of empirical risk minimisation algorithms.

Lemma 12. *For consistent empirical risk minimisers with a hypothesis class of finite Vapnik-Chervonenkis (VC) dimension the sample size required to achieve an (ε, Δ)-guarantee is given by $N(\Delta) = (\alpha_\varepsilon + \beta_\varepsilon \log_2 {}^1\!/\!_\Delta)^k$ with $\alpha_\varepsilon = 4 \ln 2 {}^1\!/\!_{\varepsilon^2}, \beta_\varepsilon = {}^4\!/\!_{\varepsilon^2 \log_2 e}$ and $k = 2$.*

Proof. For consistent empirical risk minimisers with finite VC-dimension, the confidence $1 - \Delta$ for a given N and ε is $\Delta = 2\mathcal{N}(\mathcal{F}, N) \exp(-N\varepsilon^2/4)$ [42], where the shattering coefficient $\mathcal{N}(\mathcal{F}, N)$ is a polynomial in N for finite VC-dimension. Solving for N yields that the algorithm run with

$$N \geq \frac{1}{\varepsilon^2} \left(\ln 2 + 4 \frac{1}{\log_2(e)} \log_2 \frac{1}{\delta} \right)$$

achieves a confidence larger or equal to the desired $1 - \Delta$. □

Lemma 13. *For consistent empirical risk minimisers with a hypothesis class of finite Rademacher complexity the sample size required to achieve an (ε, Δ)-guarantee is given by $N(\Delta) = (\alpha_\varepsilon + \beta_\varepsilon \log_2 {}^1\!/\!_\Delta)^k$ with $\alpha_\varepsilon = 0, \beta_\varepsilon = {}^1\!/\!_{2(\varepsilon + 2\rho)^2}$ and $k = 1$, where ρ denotes the Rademacher complexity.*

Proof. For consistent empirical risk minimisers with a hypothesis class of finite Rademacher complexity ρ, a given Δ and N the error bound is given by $\varepsilon = 2\rho + \sqrt{{}^{\log_2 1/\delta}\!/\!_{2N}}$ [42]. Solving for N yields the above result. □

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009.

[2] Maria Florina Balcan, Yingyu Liang, Le Song, David Woodruff, and Bo Xie. Communication efficient distributed kernel principal component analysis. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 725–734, 2016.

[3] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2003.

[4] Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM (JACM)*, 14(2):322–336, 1967.

[5] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[6] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[7] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science*, pages 98–108, 1976.

[8] Kenneth L. Clarkson, David Eppstein, Gary L. Miller, Carl Sturtivant, and Shang-Hua Teng. Approximating center points with iterative Radon points. *International Journal of Computational Geometry & Applications*, 6(3):357–377, 1996.

[9] Stephen A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 338–345, 1979.

[10] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1):165–202, 2012.

[11] Shai Fine and Katya Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.

[12] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[13] Yoav Freund, Yishay Mansour, and Robert E. Schapire. Why averaging classifiers can protect against overfitting. In *Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics*, 2001.

[14] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, Inc., 1995.

[15] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[16] David Kay and Eugene W. Womble. Axiomatic convexity theory and relationships between the Carathéodory, Helly, and Radon numbers. *Pacific Journal of Mathematics*, 38(2):471–485, 1971.

[17] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71(1):95–132, 1990.

[18] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin-Cummings Publishing Co., Inc., 1994.

[19] Moshe Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[20] Shao-Bo Lin, Xin Guo, and Ding-Xuan Zhou. Distributed learning with regularized least squares. *Journal of Machine Learning Research*, 18(92):1–31, 2017. URL http://jmlr.org/papers/v18/15-586.html.

[21] Philip M. Long and Rocco A. Servedio. Algorithms and hardness results for parallel large margin learning. *Journal of Machine Learning Research*, 14:3105–3128, 2013.

[22] Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I. Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, 32(4):813–848, 2017.

[23] Ryan Mcdonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S. Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, pages 1231–1239, 2009.

[24] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[25] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB 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. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[26] Cleve Moler. Matrix computation on distributed memory multiprocessors. *Hypercube Multiprocessors*, 86(181-195):31, 1986.

[27] Ilia Nouretdinov, Sergi G. Costafreda, Alexander Gammerman, Alexey Chervonenkis, Vladimir Vovk, Vladimir Vapnik, and Cynthia H.Y. Fu. Machine learning classification with confidence: application of transductive conformal predictors to MRI-based diagnostic and prognostic markers in depression. *Neuroimage*, 56(2):809–813, 2011.

[28] Dino Oglic and Thomas Gärtner. Nyström method with kernel k-means++ samples as landmarks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2652–2660, 06–11 Aug 2017.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, RRon Weiss, Vincent Dubourg, Jake Vanderplas, AAlexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] Johann Radon. Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten. *Mathematische Annalen*, 83(1):113–115, 1921.

[31] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, 2007.

[32] Jonathan D. Rosenblatt and Boaz Nadler. On the optimality of averaging in distributed statistical learning. *Information and Inference*, 5(4):379–404, 2016.

[33] Alexander M. Rubinov. *Abstract convexity and global optimization*, volume 44. Springer Science & Business Media, 2013.

[34] Ohad Shamir and Nathan Srebro. Distributed stochastic optimization and learning. In *Proceedings of the 52nd Annual Allerton Conference on Communication, Control, and Computing*, pages 850–857, 2014.

[35] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014.

[36] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Symposium on Security and Privacy*, pages 305–316, 2010.

[37] Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. *Optimization for machine learning*. MIT Press, 2012.

[38] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[39] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[40] Vladimir N. Vapnik and Alexey Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2): 264–280, 1971.

[41] Jeffrey S. Vitter and Jyh-Han Lin. Learning in parallel. *Information and Computation*, 96(2): 179–202, 1992.

[42] Ulrike Von Luxburg and Bernhard Schölkopf. Statistical learning theory: models, concepts, and results. In *Inductive Logic*, volume 10 of *Handbook of the History of Logic*, pages 651–706. Elsevier, 2011.

[43] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining: Practical machine learning tools and techniques*. Elsevier, 2017.

[44] Yuchen Zhang, John C. Duchi, and Martin J. Wainwright. Communication-efficient algorithms for statistical optimization. *Journal of Machine Learning Research*, 14(1):3321–3363, 2013.

[45] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2595–2603, 2010.