Abstract

With the size of genetic datasets growing studying datasets fragmented across multiple institutions is emerging as a powerful paradigm for the future of gene mapping and discovery. In this work, we investigate decentralized optimization algorithms for robust statistical inference on datasets fragmented across multiple data owners. In particular, we consider the logistic and linear regression problems and with extensive simulation show that the inference can be done across multiple data owners in a practical way. Finally, we simulated a realistic GWAS by splitting WTCCC **fill in** disease dataset across 5 data owners and compare the decentralized algorithms with standard meta-analysis algorithms and the combined analysis approach.

Keywords: words, words, words

Introduction

recent datasets are large In recent years, the expansion in the available genomic and phenotypic data, brought about by the next-gen biomolecular and biometric technologies, has lead to breakthroughs in our understanding of human disease. The incorporation of more personal level data, as well as the push for studying lower effect sizes, and rare variants increases the need for a larger cohort in order to achieve the necessary power. This expansion of available data presents new challenges in storage, security and computation.

Shortcomings of current approaches The old approach (Table 1 (a)) of generating, gathering and maintaining these multi-factorial and private datasets in a central location can be expensive, unsafe and time consuming. Sharing and combining summary statistics through meta-analysis techniques can sometimes offer a work-around for these issues. However, these methods have a few limitations: (a) subtle differences in models, assumptions and QC can introduce biases in the results (18), (b) the shared data and summary statistics might be inadequate for some types of inference (e.g. estimating effects from shared two-tailed p-values), and (c) parameter estimates can be unreliable when the effective sample size (N) is small compared to the number of covariates (k) as could happen for variants with low allele frequency. (d) smaller datasets may be ignored as the effort to incorporate them may outweigh their benefits.

Advances in machine learning and algorithms used With the big data revolution and explosion of accessible data in many fields, new parallel or decentralized algorithms become increasingly available (7; 24; 32; 17; 9; 10; 26). In this work we are concerned with exploring inference algorithms that allow the data from the participants to remain fragmented across distant data owners (DOs). These DOs can communicate with the central hub to iteratively solve the problem but they may not transfer the raw data (1 (b)). As opposed to the more common case where the goal of parallel computing is to speed up a computation performed on nodes on the same chip or on a cluster of computers in close proximity. Therefore, we assume that substantial latency might effect the communications within nodes. As such, we use the distinction that while decentralized algorithms can be used in parallel, not every parallel algorithm is appropriate for a decentralized setting.

Summary of what's to come Our main goal for what follows is to demonstrate that simple GWAS studies can be performed under this fragmented data paradigm. Specifically, we limit our discussion to linear and logistic regression and generating principle component values. In what follows, we first review the regression framework used and present the optimization problem that needs to be solved. This is followed by a short discussion of the optimization algorithms and the algorithm for extracting the PCA. In the results section we present extensive simulations for evaluating the trade-offs between algorithms and we perform a GWAS study on WTCCC **fill** dataset split between 5 cohorts.

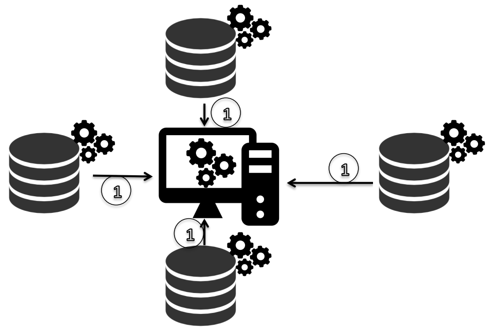
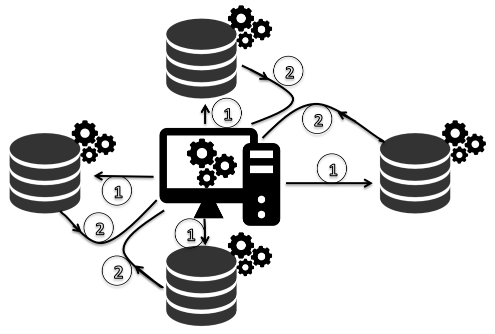
Centralized	Decentralized
	
(a)	(b)

Table 1: (a)The simplest method of data-sharing would involve all DOs sending their data to a single location. That location would then execute the relevant machine learning algorithms. (b) In our model of decentralized scheme, one central hub can communicate with all the DO's without directly requiring the data to be sent to the central hub. The central hub coordinates the execution of the relevant machine learning algorithm and through a potentially iterative process, a single set of parameters is estimated for the machine learning algorithm.

Methods

Generalized Linear Models

In a broad sense, all regression based classifiers are consisted of three parts: representation, evaluation and optimization (12). The **representation** generally describes the relationship between parameters and limits the class of models that the learner can represent. The **evaluation function** (also known as cost function) is a tool for evaluating different parameter settings of the model. Finally, the **optimization technique** is the algorithm used to find the optimum model-parameters with respect to the chosen cost function. Both logistic and linear regression are part of the larger

Type of GLM	Mean function $E[\mathbb{Y}]$	Semantics
Linear regression	$\mu = X\beta$	Outcome is a linear function of the parameters with range $(-\infty, \infty)$
Logistic regression	$\mu = \frac{\exp(X\beta)}{1+\exp(X\beta)}$	Probability of a categorical outcome based on a binomial distribution
Poisson regression	$\mu = \exp X\beta$	Modeling Poisson distributed $\frac{\text{counts}}{\text{time interval}}$
Multinomial regression	$\mu = \frac{\exp(X\beta)}{1+\exp(X\beta)}$	Probability of each outcome for a K-sided loaded die

Table 2: List of most commonly used link functions, the GLM regression they correspond to and their common use case.

generalized linear models (GLMs) family.

Representation: GLMs treat each response variable as a random draw from a distribution in the exponential family where the mean of the distribution can be linked to a linear function of the covariates (28). More concretely, a GLM models the expected value of the outcome variable ($\mathbb{Y}$) as:

$$E(\mathbb{Y}) = \mu = g^{-1}(X\beta) \qquad [1]$$

Where X is a matrix with covariates for each individual as its rows (We will use roman indices's to indicate an individual specific response and covariates.). β is the vector of unknown parameters and g is a link function that is known a priori. In this setting, the link function (g) provides flexibility to model various types of outcomes. Table 2, provides a short list of examples of common link functions and their typical use cases. (see (1) for a comprehensive review)

GLM evaluation function: The unknown parameters (β) are chosen to optimize a particular cost function. In a Bayesian setting, optimizing the posterior probability is the natural cost function. Another approach is to maximize the likelihood function. This approach offers estimators with theoretically attractive properties (33). For GLM's the assumption that each outcome is independent given the covariates leads to log-likelihood that is the sum of per-sample log likelihoods

$$\ell = \log(P(\mathbb{Y} \mid \beta, X) = \sum_{i=1}^{N} \log(P(Y_i \mid \beta, X_i) = \sum_{i=1}^{N} \ell_i \qquad [2]$$

a common modification to this cost function is adding penalization on β (see for a few examples(35; 41; 16)). In particular, after the addition of the penalty term, the cost function remains the sum of individual-level costs. This (penalized) maximum likelihood approach is the approach chosen by many statistical packages (used by glmnet (15) in R and scikit-learn (31) in python).

GLM optimization techniques vary depending on the details of the regression and will constitute our main focus. One common optimization approach is the iteratively reweighted least square (IRLS) algorithm, which iteratively solve a carefully constructed weighted least square problems (see (1) 4.6 or (27) 8.3 for details). While, the default for some common packages (glm2 in R for example (25) for example) uses IRLS, this algorithm is not practical if the data is very high dimensional or, otherwise, cannot be gathered in one location. As a result sub-gradient based methods have been adopted by many packages (such as glmnet in R, (15), Vowpal Wabbit (21) and Python's scikit-learn (31)) to grapple with today's high dimensional data.

Optimization algorithms

For this work we benchmark the performance of 7 optimization algorithms in a decentralized setting. Below, we provide a brief explanation of the algorithms a more detailed explanation of the algorithms and the relevant parameters is available in the Supplementary Materials.

SGD is a simple stochastic gradient algorithm. The step-size is chosen to be constant for the first 500 iterations and will slowly decays after.

ADAGRAD uses the algorithm presented in (13). This algorithm uses previous gradient information to modify the step-size for each covariate. As a result, it is more robust to the choice of step-size while maintaining a per-covariate step-size.

RMSPROP uses the algorithm presented in (36). This algorithm is similar to ADAGRAD but puts a emphasis the recent gradients in order to prevent a heavy reduction in step-size for later iterations.

ADADELTA uses the algorithm presented in (38). This algorithm also uses previous gradients and updates to produce a covariate-specific step-size.

SQN uses the algorithm presented in (8). In this algorithm every few iterations a noisy Hessian is estimated using a relatively large batch size. This Hessian is then used to compute the per-covariate step-size for each stochastic gradient descent step. **nmight need to cite more stuff**

AVG computes the model parameters at each DO and uses a weighted average to compute the overall model parameters. Unlike the other algorithms, this algorithm only has one step and does not have any parameters.

ADMM implements the ADMM algorithm (6). This algorithm generalizes the AVG algorithm by iteratively updating the estimates.

The stochastic gradient for the first 5 algorithms is often computed on a small batch-size. Often the faster computational speed of computing a noisier gradient on a smaller batch-size overcomes the benefits of a less noisy estimate on a big batch-size. In our case, the network latency trumps the gradient computation time, therefore generally we use relatively large batch sizes.

To perform the first 5 algorithms in a decentralized fashion, we follow (9) and compute a mini-batch gradient at each node and simply average the gradient at the central node once results from all DO's have been reported. This provides us with a gradient estimate on a batch-size of $\#_{DO's} \times k$ where k is the per-DO batch-size. Unfortunately, unlike (9), we cannot simply ignore the DO's with a late response time as the data from all DO's might not be identically distributed.

################ ABOVE HAS BEEN UPDATED + TWO NEW FIGURES BE-LOW

parallelization 2) Algorithm **??** assumes the information exchange happens sequentially. In practice, this algorithm can be parallelized. Unfortunately, the parallelization algorithms that assume the data can be assigned to nodes in a balanced way, may not always be applicable (39). The main concern with parallelization is that as one DO updates the parameter the other DO's may be busy computing the gradient using an older version of the parameter. One possibility is to use k DO's, compute $\frac{b}{k}$ gradients in a parallel at each DO and synchronously combine the results into an average gradient from a batch of size b (10; 9).

Centralized vs. distributed

An important factor in choosing the optimization technique is whether the data can be centralized and loaded into the memory. Here, we will consider a model where a central hub coordinates many DOs and solves the problem without copying the data from the DO to the hub (fig**??**) We will assume that each DO has compute power but cannot hold the entire dataset in memory. The general, the algorithm starts with 1) the DO informing the hub of arrival of new sample(s), 2) the hub proceeds to send the DO relevant information and 3) the DO sends information back on how the parameters should be updated.

Stochastic Gradient Descent

Before describing the algorithms for each of the scenarios in table 1, we briefly review the stochastic gradient descent (SGD) algorithm. For independent samples, the MLE-derived cost function for GLM family ($C(\beta; X, y)$) can separated into the sum of cost functions over all the individuals.

$$C(\beta; X, y) = \sum_{i=1}^{N} C_i(\beta) \qquad [3]$$

Where N is the number of individuals and C_i is the cost function calculated over the ith data point. Then the gradient descent update rule is given as

$$\beta_{(t+1)} = \beta_t - \eta_t \sum_{i=1}^{N} \nabla C_i(\beta_t) \qquad [4]$$

Where $\nabla C_i(\beta^t)$ is the gradient of the cost function evaluated at the previous estimate of β, η_t is the step size for the tth iteration of the algorithm. For large N computing the sum can become computationally expensive particularly given the relative slow rate of convergence for this algorithm (linear convergence for a β-smooth and α-convex function **check these conditions and perhaps replace that with GLM family for simplicity (if it is true)**). (Batch) stochastic gradient descent (SGD) attempts to circumvent this problem by updating the estimate based on the gradient from a randomly chosen data points (or batch). SGD trades off faster convergence rate of GD with the smaller cost of each individual updates and for the same set of functions (**match this to GD's claim**), converges in time independent of the total number of samples present (cite Optimization Methods for Large-Scale Machine Learning).

The choice of η is crucial for convergence of SGD. In particular, convergence requires:

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty; \quad \sum_{t=1}^{\infty} \eta_t = \infty \qquad [5]$$

Which intuitively means that the step-size should be chosen to be small but not too small. A learning schedule with $\eta_t = \frac{c_1}{(c_2+t)^\gamma}$ and $0.5 < \gamma \leq 1$ satisfies this condition. However, it must be noted that still the values of $c_1, c_2 > 0$ need to be chosen judiciously. In particular, if the step-sizes are too large, the algorithm may not converge and if the step-sizes are too small, the convergence rate may be too small (cite bottou and the other paper)

The stochastic gradient descent described above is a part of a much larger family of subgradient based stochastic optimizers. One can, largely, avoid the issues related to choosing a good learning rate by utilizing techniques that are robust to the choice of learning rate (cite adagrad and implicit-implicit sgd). These stochastic subgradient algorithms will play a central role in parameter inference for the four scenarios presented in table 1.

shortcomings In this work, we will focus on SGD algorithm for two major reasons. 1) The simplicity of the algorithm and 2) its application to all four settings represented by table 2. However, SGD is not always the appropriate optimization tool. Aside from it's slower rate of convergence compared to second order methods, SGD is 1) reliant on choosing a good step-size and 2) is unable to deal with sparsity inducing, ℓ_1 regularization.

rebuttal to shortcomings Fortunately, SGD algorithm can be modified to remedy both of the aforementioned shortcomings. The reliance on step-size can be reduced by modifying the algorithm to adaptively learn a per-dimension step-size (13; 38; 19). To address the second shortcoming, various algorithms and modifications to the vanilla SGD algorithm can be used to obtain a sparse solution (14; 22; 34).

In what follows, we will explore each setting of table1. For each setting, we offer a short literature review followed by a few relevant algorithms. While we cover all four settings, but we will focus our attention on offline-distributed and online-distributed.

Setting 1: Offline and Centralized

Problem set up and solution is the most common setting for statistical analysis by researchers. We define this setting as the case where all the data is available at the time of computation (therefore offline), and at a single computational node. For a GLM model, the (log)-likelihood can be efficiently evaluating for a parameter vector $\hat{\beta}$. Therefore, for a small enough problem, the solution can be found using Newton-Raphson method. However, since this method requires inverting the Hessian matrix, it scales poorly for large number of covariates (suggest papers for that use). A closely related optimization technique is the Fisher Scoring method which replaces the Hessian in Newton's method with the expected value of Hessian. Using Fisher Scoring method, a model can be solved by iteratively solving weighted linear regression problems (iteratively weighted least squares)

While the rate of convergence is higher for both of the aforementioned methods (cite) stochastic optimization techniques still provide a plausible alternative particularly when the dataset is large. We will further discuss these methods in the next 3 sections.

Setting 2: Offline and Distributed

related lit In this setting, all the required data is gathered but the data is distributed across multiple nodes and cannot be centralized. This setting naturally arises when the communication of data is too costly or privacy is of concern. (17) provides an algorithm for distributed logistic regression but considers the setting where the covariates are distributed across the nodes. (26) provides three novel algorithms for lasso regression. In all three algorithms, the parameters are computed locally and updated based on the parameters computed on "neighboring" datasets. In general, Alternating Direction Method of Multipliers (ADMM) provides a natural framework for consensus optimization of convex functions (6). In this method the number of parameters is first expanded to include a set of private parameters for each node as well as a set of public parameters then the optimization problem is modified to optimize over the local parameters with the constraint that the local parameters must match the global parameters. Using this framework, the solution can be iteratively computed as detailed in (6; 24).

Algorithm Another promising approach to solving the problem is via a simple adaptation of the SGD algorithm presented earlier. Let $D_1 \ldots D_M$ denote the DO's then, Algorithm **??** represents a simple way of performing SGD without gathering the data. In any practical implementation, a few caveats need to be considered.

mini-batch **1)** Due to network latency, it may be beneficial to compute the gradient on a batch size > 1 to reduce number of communications with the DO.

parallelization **2)** Algorithm **??** assumes the information exchange happens sequentially. In practice, this algorithm can be parallelized. Unfortunately, the parallelization algorithms that assume the data can be assigned to nodes in a balanced way, may not always be applicable (39). The main concern with parallelization is that as one DO updates the parameter the other DO's may be busy computing the gradient using an older version of the parameter. One possibility is to use k DO's, compute $\frac{b}{k}$ gradients in a parallel at each DO and synchronously combine the results into an average gradient from a batch of size b (10; 9).

Short comings and Extensions Stochastic gradient descent is readily expendable to online learning

SGD in distributed setting

Algorithm If parallelization is not an issue

Setting 3,4: Online

When data arrives one (or few) at a time, one may choose to rerun the entire regression in an offline setting with the currently available data, however, this approach may be costly and impractical if new data frequently becomes available. Online learning algorithms attempt to compute an efficient update to the previous estimates using the information from the new data point. Bayesian framework provides a natural way of updating the model parameters as new data arrives. In this setting common practices for computing the update include, numerical integration as in (40) or replacing the true posterior distribution with an approximate posterior chosen from a parametric distribution. (30; 37; 29)

Another common approach is to use sub-gradient optimization methods to calculate online updates for the model parameter(13; 22; 20). A simple algorithm, in this setting, would update compute the gradient as a new point arrives, and update the parameter estimates based on this gradient but, in contrast with algorithm **??**, it would not loop over the data to reach convergence.

Results

In this section, we will demonstrate the performance of linear regression and logistic regression optimized using a selected set of algorithms from the list reviewed in the previous section. We will generate synthetic data as explained in the next section to evaluate the performance of each algorithm for varying number of covariates, data sizes, and covariance structures.

DataSets

To benchmark the algorithms presented earlier, we will simulate dataset as follows:

Uncorrelated continuous covariates are drawn from a $Norm(0, 10^{\alpha})$ with α drawn unifromly from the interval $(0, 3)$. The discrete covariates are limited to take dosage values ({0,1,2}). The dosage values are assigned by drawing a frequency parameter for each covariate ($f \sim Unif(0.01, 0.49)$) and assigning a dosage of 0,1,2 with probabilities $f^2, 2f(1-f),$and $(1-f)^2$ for each individual. Let X_i denote the row vector of the covariates for individual i, and β denote the vector of coefficients (drawn from a $Norm(0,1)$ distribution). Using these definitions, the label for individual i is computed as $y_i = X_i\beta + b + \epsilon_i$ for linear regression and $y_i = binom(1, X_i\beta + b + \epsilon_i)$ for logistic regression. In both cases, ϵ_i is an individual-specific Gaussian error term with $Norm(0, 0.1)$ distribution and the bias term (b) is set to zero ($b = 0$). The data is distributed across 5 DO's. The first four DO receive a random number of individuals (distributed according to $Pois(\frac{\text{size}}{\text{\#DO's}})$) and the last DO received the remaining individuals. The simulation is restarted if any DO receives less than 10% of the data.

We study the effects of correlation for continuous variables only. The covariates are drawn from a multivariate normal random variable with a random covariance matrix generated via vine method detained in (23). The remaining values are assigned as before.

Algorithms

In order to benchmark the general approaches outlined in Methods, we implemented benchmarked 7 representative algorithms. Since we anticipate the largest cost to be the communication between the DO's and the central hub, we will evaluate these algorithms based on the number of communications rather than the total compute time. Below is a short list of the implemented algorithms:

SGD is the simple stochastic gradient algorithm introduced earlier. The stepsize is chosen to be constant for the first 500 iterations and will decays as $\eta_t = \frac{\eta}{(t-500+1)^{0.51}}$ for the remaining steps.

ADAGRAD uses the algorithm presented in (13). This algorithm uses previous gradient information to estimate the currect stepsize for each covariate. As a result, it is more robust to stepsize choices and can have a per-covariate stepsize.

RMSPROP uses the algorithm presented in (). This algorithm is similar to ADAGRAD but puts a heavier emphasis on the recent gradients in hopes of recucing the heavy reduction step-size for larger iterations.

ADADELTA uses the algorithm presented in (38). This algorithm also uses previous gradients and updates to produce a covariate-specifid step-size.

SQN uses the algorithm presented in () (byrd). In this algorithm every few iterations a noisy Hessian is estimated using a relatively large batch size. This Hessian is then used to compute the per-covariate step-size for each stochastic gradient descent step.

AVG computes the model parameters at each DO and uses a weighted average to compute the overall model parameter. As a result, this algorithm only has one iteration.

ADMM implements the ADMM algorithm (6). We will use an ℓ_1 normalized logistic regression and LASSO () regression with very small penalties in lieu of logistic regression and oridinary least squares.

More information about the parameters, stepsizes, and implementations can be found in **reference link**.

Performance

In order to benchmark these algorithms, we restricted the number of communications to 1,000 and measured the performance for different number of covariates drawn from purely gaussian, equal mix of guassian and dosage and purely dosage distributions using 5 DO's. For each experiment, the data was gathered to compute the gold-standard solution. Figure 3.4 shows the per-sample logistic cost function difference for these algorithms and the combined logistic regression for 2,3,6,11 covariates (including the intercept value). Figure **??** shows a similar result for least squares regression and per-sample mean-squared error.

We note that ADMM and SQN consistently outperform other algorithms. Simple averaging performs well, when the sample size is large compared to the number of covariates, however, when this is not the case, it can produce very inaccurate estimates and therefore had to be removed from the top row of figure 3.4.

PUT STUFF RELATED TO LEAST SQUARES HERE

Performance as a function of number of DO's

Once again, we see that when the per-DO sample size is large compared to the number of covariates, simply averaging the per-DO estimates is a very accurate estimator, however this estimator fails when the per-DO estimates deteriorate. On the other hand SQN and ADMM are accurate throughout the parameter space explored.

Performance as a function of number of communications

In the previous experiments, we have limited all the algorithms to 1000 iterations.

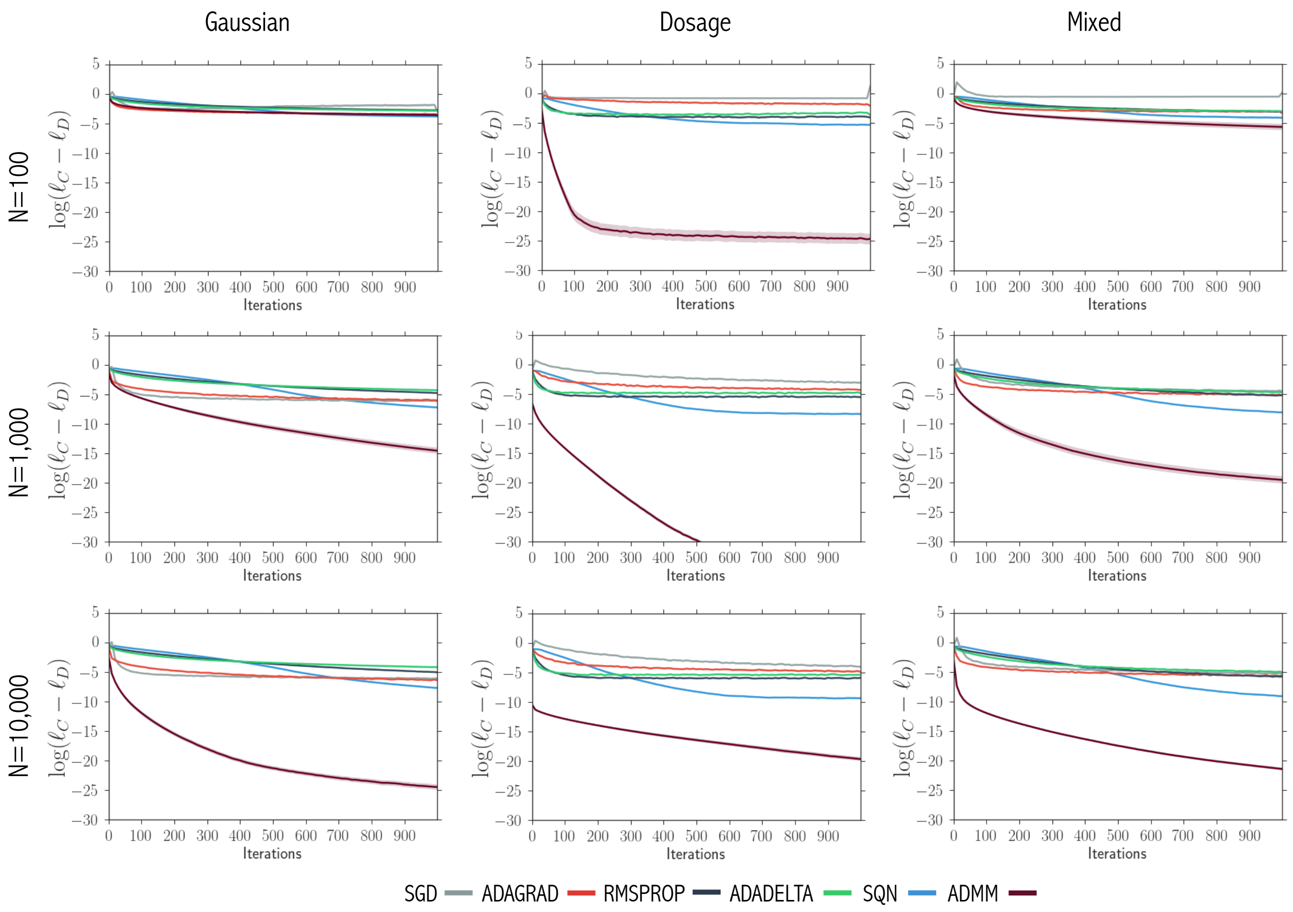


Figure 1: Plots of convergence of each method to the solution from a centralized logistic regression for 9 covariates (+1 intercept). In all cases, ADMM approaches the solution from centralized logistic regression faster than the other methods. Furthermore, at 1,000 iterations, the ADMM solution outperforms all other methods. As expected, ADMM converges much faster when the amount of data in each DO is large.

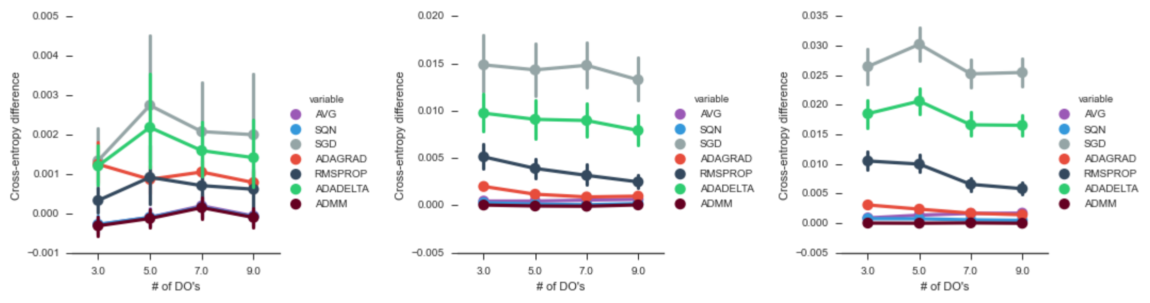


Figure 2: Insert caption w/ batch size and ...

Discussion

Conclusions

Figures and Tables

Figure 3: **This is the 1-line title of the figure.**

This is a longer text for detailed explanation. Should be longer than 1-line.

Table 3: **This is the 1-line title of the table.**

This is a longer text for detailed explanation. Should be longer than 1-line.

References

1 Agresti A, Kateri M. 2011. Categorical data analysis. In: International encyclopedia of statistical science. Springer. p. 206–208.

2 Bottou L. 1998. Online learning and stochastic approximations. On-line learning in neural networks 17:142.

3 Bottou L. 2012. Stochastic gradient descent tricks. In: Neural networks: Tricks of the trade. Springer. p. 421–436.

4 Bottou L, Curtis FE, Nocedal J. 2016. Optimization methods for large-scale machine learning. arXiv preprint arXiv:1606.04838 .

5 Bousquet O, Bottou L. 2008. The tradeoffs of large scale learning. In: Advances in neural information processing systems. p. 161–168.

6 Boyd S. 2011. Alternating direction method of multipliers. In: Talk at NIPS Workshop on Optimization and Machine Learning.

7 Boyd S, Parikh N, Chu E, Peleato B, Eckstein J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning 3:1–122.

8 Byrd RH, Hansen SL, Nocedal J, Singer Y. 2016. A stochastic quasi-newton method for large-scale optimization. SIAM Journal on Optimization 26:1008–1031.

9 Chen J, Monga R, Bengio S, Jozefowicz R. 2016. Revisiting distributed synchronous sgd. arXiv preprint arXiv:1604.00981 .

10 Dekel O, Gilad-Bachrach R, Shamir O, Xiao L. 2012. Optimal distributed online prediction using mini-batches. Journal of Machine Learning Research 13:165–202.

11 Dennis Jr JE, Schnabel RB. 1996. Numerical methods for unconstrained optimization and nonlinear equations. SIAM.

12 Domingos P. 2012. A few useful things to know about machine learning. Communications of the ACM 55:78–87.

13 Duchi J, Hazan E, Singer Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12:2121–2159.

14 Duchi J, Shalev-Shwartz S, Singer Y, Chandra T. 2008. Efficient projections onto the l 1-ball for learning in high dimensions. In: Proceedings of the 25th international conference on Machine learning. ACM. p. 272–279.

15 Friedman J, Hastie T, Tibshirani R. 2010. Regularization paths for generalized linear models via coordinate descent. Journal of statistical software 33:1.

16 Friedman JH. 2012. Fast sparse regression and classification. International Journal of Forecasting 28:722–738.

17 Gopal S, Yang Y. 2013. Distributed training of large-scale logistic models. In: ICML (2). p. 289–297.

18 Greco T, Zangrillo A, Biondi-Zoccai G, Landoni G. 2013. Meta-analysis: pitfalls and hints. Heart, lung and vessels 5:219.

19 Kingma D, Ba J. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

20 Langford J, Li L, Strehl A. 2007. Vowpal wabbit online learning project.

21 Langford J, Li L, Strehl A. 2011. Vowpal wabbit. URL https://github.com/JohnLangford/vowpal wabbit/wiki .

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

23 Lewandowski D, Kurowicka D, Joe H. 2009. Generating random correlation matrices based on vines and extended onion method. Journal of multivariate analysis 100:1989–2001.

24 Lubell-Doughtie P, Sondag J. 2013. Practical distributed classification using the alternating direction method of multipliers algorithm. In: Big Data, 2013 IEEE International Conference on. IEEE. p. 773–776.

25 Marschner IC, et al. 2011. glm2: fitting generalized linear models with convergence problems. The R journal 3:12–15.

26 Mateos G, Bazerque JA, Giannakis GB. 2010. Distributed sparse linear regression. IEEE Transactions on Signal Processing 58:5262–5276.

27 Murphy KP. 2012. Machine learning: a probabilistic perspective. MIT press.

28 Nelder JA, Baker RJ. 1972. Generalized linear models. Encyclopedia of statistical sciences .

29 Opper M. 1996. On-line versus off-line learning from random examples: General results. Physical Review Letters 77:4671.

30 Opper M, Winther O. 1999. A bayesian approach to on-line learning. On-line Learning in Neural Networks, ed. D. Saad :363–378.

31 Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830.

32 Recht B, Re C, Wright S, Niu F. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems. p. 693–701.

33 Scholz F. 1985. Maximum likelihood estimation. Encyclopedia of statistical sciences .

34 Shalev-Shwartz S, Tewari A. 2011. Stochastic methods for l1-regularized loss minimization. Journal of Machine Learning Research 12:1865–1892.

35 Tibshirani R. 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) :267–288.

36 Tieleman T, Hinton G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4:26–31.

37 Winther O, Solla SA. 1998. Optimal bayesian online learning. Theoretical Aspects of Neural Computation (TANC-97), KYM Wong, I. King and D.-Y. Yeung eds., Springer Verlag, Singapore .

38 Zeiler MD. 2012. Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 .

39 Zinkevich M, Weimer M, Li L, Smola AJ. 2010. Parallelized stochastic gradient descent. In: Advances in neural information processing systems. p. 2595–2603.

40 Zoeter O. 2007. Bayesian generalized linear models in a terabyte world. In: Image and Signal Processing and Analysis, 2007. ISPA 2007. 5th International Symposium on. IEEE. p. 435–440.

41 Zou H, Hastie T. 2005. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67:301–320.

==============NEW STUFF============= **make sure the bottou 2016 source is the correct one**

Supplementary Materials

In this section we provide a brief explanation of the algorithms used along with the specific parameters chosen for our experiments. A comprehensive review of these algorithms is beyond the scope of this work.

Stochastic Gradient Descent

Given a function $f(\mathbf{x})$ the Gradient Descent (GD) algorithm iteratively minimizes the function by computing the gradient of function f and updating $\mathbf{x}$ by taking a small step in the direction opposite to the average gradient. Under sufficient regularity assumptions and when the step-size is small enough and the initial point is close enough to a minima, this algorithms converges at a linear rate (11). As discussed in the main text, for independent samples, the (penalized) MLE-derived cost function for GLM family can be separated into the sum of cost functions over all the individuals (for the most usual penalizations). Denote the likelihood by $\ell(\beta; X, \mathbb{Y})$ and the penalty function with $P(\beta)$ and let $C_i(\beta; X, Y) = \ell_i(\beta; X_i, \mathbb{Y}_i) + P(\beta)$ be the overall cost function. Then we can write:

$$\ell(\beta; X, \mathbb{Y}) = \sum_{i=1}^{N} C_i(\beta; X, Y) \qquad [6]$$

And therefore the update becomes:

$$\beta_{t+1} = \beta_t - \alpha_t \nabla_\beta C(\beta; X, Y) = \beta_t - \alpha_t \sum_{i=1}^{N} \nabla_\beta C_i(\beta; X, Y) \qquad [7]$$

Where all functions are evaluated at β_t, $\nabla_\beta f$ indicates the gradient of the function $f(\beta)$ with respect to β and α_t represents the step size at time t.

Unfortunately, for large datasets, this computation may be slow since the gradient includes a sum over the entire dataset. (Mini-batch) stochastic gradient descent overcomes this problem by computing the average gradient and updating the parameter based on a small batch of data, or even a single sample, for each iteration. For a convex function, under relatively mild assumptions, this algorithms almost surely converges to the minimum for step sizes satisfying (2; 27):

$$\begin{gathered} \sum_{0}^{\infty} \alpha_t = \infty \\ \sum_{0}^{\infty} \alpha_t^2 < \infty \end{gathered} \qquad [8]$$

Intuitively, the first requirement insures that the even a poor choice of initial parameters for β will be able to reach the optimal parameters. The second requirement insures that the step size becomes smaller as we reach the optimal parameter so that the estimate does not wobble around the optimal. While the added noise, due to the stochastic approximation of the gradient, slows down the convergence rate of the algorithms, for a large dataset, SGD can outperform GD in the overall amount of computations needed to reach a given precision (5; 4).

In our implementation we use relatively large batch sizes (**TODO**) in order to reduce the total number of updates by reducing the noise in each update. The optimal starting step size was chosen via cross-validation. The step sizes are identical for all covariates and were set to be constant for the first 500 iterations and fall off as: $\alpha_t = \frac{c_1}{(c_2+t-500)^\gamma}$ for $0.5 < \gamma \leq 1$ (general form recommended by (27)) with $C_1 = \alpha_0,\ C_2 = 1$ and $\gamma = 0.51$ for the remaining iterations.

For a comprehensive review of SGD we refer the readers to (27) for a quick introduction and to (4; 3) for a more theoretical approach.

To motivate the next 3 algorithms, we note that equation 9 resembles a Newton's update if the scalar step-size α_t is replaced by the inverse of the Hessian matrix (H^{-1}).

$$\beta_{t+1} = \beta_t - H^{-1}\nabla_\beta C(\beta; X, Y) \qquad [9]$$

For a quadratic function C Newton's method solve the optimization problem in a single step. In more general cases, Newton's method can be viewed as approximating C as a quadratic function and iteratively updating our estimate to represent the minimum of this approximated quadratic function, therefore, Newton's method is known as a second order method. In contrast, simple SGD updates of equation 9 only consider the slope and therefore are known as first order methods. While it has been shown that stochastic method's cannot reach a convergence rate faster than sub-linear, incorporating approximate second-order information in a mini-batch setting can have improve performance. One way to incorporate this information is using a diagonal approximation to the Hessian. Such an approximation in effect scales the step size for each search direction.

ADAGRAD The AdaGrad algorithm (with diagonal matrices) is one popular approach for rescaling the step size using previous gradient values (algorithm 1 in (13)). In doing so, AdaGrad aims to increase the learning rate of infrequent features with the intuition that the learner should pay attention to infrequent features when they do occur (13). This makes AdaGrad particularly suited for dealing with sparse data. The algorithm is presented in (13) however, below, we reproduce the version we implemented for

Algorithm 1: AdaGrad Algorithm (original algorithm in (13))

```
Data: $\alpha > 0, T$
Result: Updated parameter $\beta_{T+1}$
initialization  $G_0, k \leftarrow 0, \beta_k \leftarrow 0$ for $k$ in $1 \dots p$;
for $t = 0$ to $T$ do
    for $i = 0$ to $N$ do
        Receive gradient ($g_t^{(i)} \leftarrow \nabla_\beta C(\beta)$ evaluated at $\beta_t$) from DO $i$;
        Receive number of individuals the gradient was computed on ($n_t^{(i)}$) from DO $i$;
    end
    $n_t \leftarrow \sum_{i=1}^N n_t^{(i)}$;
    $g_t \leftarrow \sum_{i=1}^N \frac{n_t^{(i)}}{n_t} g_t^{(i)}$;
    $G_{t+1,k} \leftarrow G_t + (g_{t,k}^{(i)})^2$ ;
    $\beta_{t+1} \leftarrow \beta_t - (\frac{\alpha}{\sqrt{G_t}})^\top g_t$
end
```

In equation 9, we multiply the gradient by a While equation 8

The choice of η is crucial for convergence of SGD. In particular, convergence requires:

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty; \quad \sum_{t=1}^{\infty} \eta_t = \infty \qquad [10]$$

Which intuitively means that the step-size should be chosen to be small but not too small. A learning schedule with $\eta_t = \frac{c_1}{(c_2+t)^\gamma}$ and $0.5 < \gamma \leq 1$ satisfies this condition. However, it must be noted that still the values of $c_1, c_2 > 0$ need to be chosen judiciously. In particular, if the step-sizes are too large, the algorithm may not converge and if the step-sizes are too small, the convergence rate may be too small (cite bottou and the other paper)

The stochastic gradient descent described above is a part of a much larger family of subgradient based stochastic optimizers. One can, largely, avoid the issues related to choosing a good learning rate by utilizing techniques that are robust to the choice of learning rate (cite adagrad and implicit-implicit sgd). These stochastic subgradient algorithms will play a central role in parameter inference for the four scenarios presented in table 1.