problem set no. 6 (Final Project) — due tuesday 12/10 at 11:59 pm.

problem background. Efficient methods for model fitting, both in a statistical and computational sense, are an imperative for "Big Data". For example, MLE estimation with large-scale models (e.g. thousands to millions of parameters) is (generally) impossible to achieve without the use of *online learning* methods. A very popular method for online learning is Stochastic Gradient Descent (SGD).

This method originates from Sakrison's work [4] that modified the Robbins-Monro procedure [3] for *recursive estimation* of the MLE. For this project we will assume that Y is a univariate random variable and that follows a known distribution model with *unknown* parameters $\boldsymbol{\theta^*} \in \mathbb{R}^k$ and log-likelihood $\ell(\cdots)$, where k is potentially very large. There also exists a dataset $\mathbf{y} = (y_1, y_2, \cdots)$, with covariates $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots\}$, $\mathbf{x}_i \in \mathbb{R}^k$, that is potentially very large. However, we can have *online access* to this dataset i.e., access y_t one-by-one for $t = 1, 2, \cdots$. At every step t we have an estimate $\boldsymbol{\theta}_t$ of $\boldsymbol{\theta^*}$ and upon observing y_t, we update the estimate to $\boldsymbol{\theta}_{t+1}$ and so on. Sakrison [4] considers the following procedure:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + a_t \nabla \ell(\boldsymbol{\theta}_t; y_t, \mathbf{x}_t) \tag{1}$$

It can be shown that Equation (1) estimates the MLE under some conditions and it was re-discovered (and re-branded) recently as Stochastic Gradient Descent (SGD) in machine learning (ML) [6, 1]. A few important remarks about Equation (1):

- A Newton-Raphson approach would need to calculate $\mathcal{H}_\ell(\boldsymbol{\theta}_t)^{-1} \nabla \ell(\boldsymbol{\theta}_t; \mathbf{y}, \mathbf{X})$, i.e. use inverses of the Hessian and evaluate the derivative at *all datapoints*. This is a *batch training* method and it is generally un-suited for large-scale learning problems. In constrast, SGD is not making expensive matrix inversions (a_t is one-dimensional) and evaluates the log-likelihood at the *current* data point y_t. It is thus an *online learning* method.
- The parameter $a_t \in \mathbb{R}, a_t > 0$ is called a *learning rate* and governs the convergence of the algorithm. According to classical theory (e.g. see [3]) it should be $\sum a_t = \infty$ and $\sum a_t^2 < \infty$. The first condition ensures that we will be able to traverse the parameter space and the second ensures that the estimates will converge and will not oscillate around the limit. One typical way to define the rate is $a_t \propto 1/t$.

There are numerous modifications to the SGD algorithm. One of the most interesting ones is using *implicit updates* that was first applied in the normal linear models [2]. The equation of updates becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + a_t \nabla \ell(\boldsymbol{\theta}_{t+1}; y_t, \mathbf{x}_t) \tag{2}$$

The key difference is that the update is being calculated in the *future estimate* $\boldsymbol{\theta}_{t+1}$. The estimate appears in both sides of the equation and so the update is called implicit. This is also an online learning method, however performing the update is more costly computationally (sometimes it is impossible). Implicit updates have been shown to be more robust, especially to signal (covariate) noise.

overview. You will implement and run SGD and Implicit algorithms to fit large-scale statistical models. You will split in two teams, namely `R-Dev` and `Cpp-Dev`. Team `R-Dev` will implement SGD and Implicit methods in R and will explore their asympotic bias and variance properties. Team `Cpp-Dev` will develop the Implicit method in SVM models and will compare it to existing SGD methods written in C++. Both teams will work on large-scale datasets that are commonly used in machine learning. The workload will be distributed equally in the two teams.

We will consider Generalized linear models (GLM):

$$\begin{aligned} \mathbf{x}_t &\sim G \\ \eta_t &= \mathbf{x}_t^\intercal \boldsymbol{\theta}^* \\ y_t|\eta_t &\sim \exp\left\{\tfrac{\eta_t y_t - b(\eta_t)}{\phi}\right\} \cdot c(y_t, \phi) \end{aligned} \tag{3}$$

IN a GLM, the expected value $E\left(y_t|\eta_t\right)$ is set equal to $E\left(y_t|\eta_t\right) = h(\eta_t)$ for some function $h(\cdot)$ that depends on the distributional assumption for Y. The function $h(\cdot)$ is called the *transfer function* and $g(\cdot) = h^{-1}(\cdot)$ is the *link function*, since it links the expected value to predictors, i.e. $g(E\left(y_t|\eta_t\right)) = \eta_t = \mathbf{x}_t^\intercal \boldsymbol{\theta}^*$. In this case, since η_t is linear with y_t in the density function, the link is called *canonical*. Also, $\phi > 0$ is the *dispersion parameter* and controls the variance of Y. Last, the distribution G of $\mathbf{x}_t$ is considered known.

question 1.1. (5 point) `-Both.Teams-`
Explain intuitively why Sakrison's method computes the MLE and what assumptions might be needed. Explain intuitively why the implicit method should also compute the MLE.

question 1.2. (15 point) `-Both.Teams-`
Show that the following hold for the GLM of Eqs. (3).

(a) $E\left(y_t|\mathbf{x}_t\right) = h(\mathbf{x}_t^\intercal \boldsymbol{\theta}^*) = b'(\eta_t)$.

(b) $Var\left(y_t|\eta_t\right) = \phi \cdot h'(\eta_t)$.

(c) $\nabla \ell(\boldsymbol{\theta}; y_t, \mathbf{x}_t) = \frac{1}{\phi}\left(y_t - h(\mathbf{x}_t^\intercal \boldsymbol{\theta})\right)\mathbf{x}_t$, derivative w.r.t. $\boldsymbol{\theta}$.

(d) $\mathbf{J}(\boldsymbol{\theta}) = -E\left(\nabla\nabla \ell(\boldsymbol{\theta}; y_t, \mathbf{x}_t)\right) = \frac{1}{\phi} E\left(h'(\mathbf{x}_t^\intercal \boldsymbol{\theta})\mathbf{x}_t \mathbf{x}_t'\right)$

R-Dev-only

question 1.3. (80 point) Work on the paper by Xu [5] and replicate the experiments in Sections 6.1 and 6.2 that includes R implementations of SGD, AVSG and Implicit.

(a) In Section 6.1, work on the toy model set in [5]. Derive the SGD, ASGD (averaged-SGD) and Implicit updates for the model. The averaged-SGD (ASGD) is a variant that takes averages of recent estimates. It is defined in Algorithm 1 of [5] and it is straightforward to implement. **Replicate** Figure 1 and add the Implicit method as well. You don't need to have the same learning rate for Implicit and SGD. Experiment with the learning rate of the Implicit method to make it perform best, and report your findings quantitatively and graphically.

(b) In Section 6.2, work on the toy normal linear model $y_t|\eta_t \sim \mathcal{N}(\eta_t, \sigma^2)$, $\eta_t = \mathbf{x}_t\boldsymbol{\theta}^*$, and $\boldsymbol{\theta}^* \in \mathbb{R}^k$. Xu [5] assumes $k = 100, \sigma^2 = 1, \boldsymbol{\theta}^* = \mathbf{1}$ and $\mathbf{x}_t \sim \mathcal{N}_{100}(0, A)$, where A has a specific eigenvalue form (see paper). Again, derive the SGD, ASGD and implicit updates. **Replicate** Figure 2 including the Implicit method as well. The implicit method needs not to have the same learning rate *schedule* as SGD. Experiment with the learning rate of the Implicit method to make it perform best, and report your findings quantitatively and graphically.

(c) Note that Xu [5] is using γ_t to denote the learning rate, i.e. $a_t = \gamma_t \propto t^{-c}, c < 1$. In this part, we will assume a rate $a_t = \gamma_t = \alpha/t$ and find the optimal α for all Algorithms and explore the differences. Work on the (a) and (b) parts assuming the aforementioned rate and find the optimal α. **Replicate** Figures 1,2 and compare performances. Do you see significant differences? Could you explain the differences?

(d) Now, fix the learning rate schedules of the Algorithms to the optimal ones you found in Part (c). Work on the model of (b) and explore the bias and variance of the estimates of SGD, ASGD and Implicit. Using Odyssey, obtain m samples of $\boldsymbol{\theta}_t$, for various t, and compute $||\boldsymbol{\theta}_t - \boldsymbol{\theta}^*||$ and the empirical variance $\boldsymbol{\Sigma}_t = \widehat{Var(\boldsymbol{\theta}_t)}$, for SGD, ASGD and the Implicit method. We wish to investigate the claim that, with a learning rate $\alpha_t = \alpha/t$, the following holds:

$$t \cdot \boldsymbol{\Sigma}_t \to \alpha^2\phi^2(2\alpha\phi\mathbf{J}(\boldsymbol{\theta}^*) - \mathbf{I})^{-1}\mathbf{J}(\boldsymbol{\theta}^*) \doteq \boldsymbol{\Sigma} \tag{4}$$

Derive the matrix $\boldsymbol{\Sigma}$ for the normal model and for each method individually (each one might have a different α). Be careful to include the proper learning rate α for each method. Then test the following hypothesis using your Odyssey design:

$$H_0 : \boldsymbol{\theta}_t \sim \mathcal{N}_{100}(\boldsymbol{\theta}^*, \boldsymbol{\Sigma}/t) \tag{5}$$

Test the hypothesis at time-points, $t = 10^3 \times \{1, 5, 10, 20, 50, 100\}$ and sample sizes $m \in 10^3 \times \{1, 10, 50, 100\}$ (one hypothesis test for a pair t, m). Your testing approach needs to be both formal (e.g. based on rigorous statistical hypothesis testing such as the likelihood test) and informal (e.g. plots of residuals, histograms etc). If you think your experiments don't give you enough power, try testing the following nulls instead:

$$H_0^{mean} : E\left(\boldsymbol{\theta}_t\right) \sim \boldsymbol{\theta}^* \\ H_0^{var} : Var(\boldsymbol{\theta}_t) = \boldsymbol{\Sigma}/t \tag{6}$$

`Cpp-Dev`-only

question 1.4. (80 point) We will work on `SVM` which is a popular model for classification. We will follow a formulation and approach that is typical in ML. Assume data $(\mathbf{x}_t, y_t)$, such that $\mathbf{x}_t \in \mathbb{R}^k$ and $y_t \in \{-1, 1\}$ are the labels. Our parameters are $\boldsymbol{\theta}^* \in \mathbb{R}^k$ and the classification model is:

$$y_t = +1 \text{ if } \mathbf{x}_t^\intercal \boldsymbol{\theta}^* > 0 \\ y_t = -1 \text{ if } \mathbf{x}_t^\intercal \boldsymbol{\theta}^* < 0 \tag{7}$$

In more compact notation $y_t = \text{sign}(\mathbf{x}_t^\intercal \boldsymbol{\theta}^*)$[1]. For a given example $\mathbf{x}_t$ a model gives a prediction $\hat{y}_t = \text{sign}(\mathbf{x}_t^\intercal \boldsymbol{\theta}_t)$. Given a ground truth y_t the associated *loss* from the prediction is $L(y_t, \hat{y}_t) > 0$. Typical choices for the loss function are shown in the following table:

loss function $L(y, \hat{y})$	formula
log-loss	$\log\left(1 + e^{-y\hat{y}}\right)$
Hinge loss	$[1 - y\hat{y}]_+$
Squared-loss	$(y - \hat{y})^2$

Table 1: Loss functions

The optimization problem of the `SVM` is then:

$$\min_{\boldsymbol{\theta}} \left(\sum_t L(y_t, \hat{y}_t) + \frac{\lambda}{2} ||\boldsymbol{\theta}||^2 \right) \tag{8}$$

(a) Derive the SGD and Implicit updates for the optimization problem in (8) and for all three loss functions in Table 2. Bottou [1] Table 1 is a helpful start. Note that,

[1] Note that, usually the first element of the feature vector $\Phi(\mathbf{x}_t)$ is set to 1 so that there is an intercept term θ_0^* that does not depend on the examples (in ML this is called, rather confusingly, a *bias term*).

usually we perform some transformation of the covariate data $\mathbf{x}_t$ into *features* through a function $\Phi(\mathbf{x}_t)$ and then use this in the linear predictor i.e. $y_t = \text{sign}(\Phi(\mathbf{x}_t)^\intercal \boldsymbol{\theta}^*)$ etc. However, the analysis remains the same (e.g. could define $\mathbf{x}_t \doteq \Phi(\mathbf{x}_t)$).

(b) Go to `http://leon.bottou.org/projects/sgd`, download and compile the provided source code (Section "`Download and Compilation`). Download the data files of `RCV1-V2` and the `Pascal` large-scale challenge. Compile and make sure everything is OK.

(c) Look into the `sgd/svm` folder and understand the implementation code of SGD and ASGD. **Implement** the Implicit updates for Equation (8) in C++ in file named `svmimplicit.cpp`.

(d) **Replicate** the two Tables in Section `RCV1 Benchmark` of the Bottou's webpage. You will not need to use `SVMLight` or `SVMperf` but you will need to download, install and run `LibLinear`[2]. In addition, you will use your implicit method to fit the same data and report the results in the same table. Note that you will only work on log-loss and hinge loss for this part. Your final tables will look like the following:

algorithm (loss ?)	training time	test error
`LibLinear`	?	?
SGD	?	?
ASGD	?	?
Implicit	?	?

Table 2: Results table for a specific loss function

(e) Similarly work on the "Pascal large-scale challenge" data (specifically the `alpha` dataset) and replicate the Table in Section `Alpha Benchmark` including your implicit implementation. Your table will have the same format as before.

* * *

what to submit. Each team will submit 1 zip file to ptoulis@fas.harvard.edu. Do not include datasets. The zip file should be named `team_final.zip`, where `team=rdev` for the `R-Dev` team and `team=cppdev` for the `Cpp-Dev` team. The zip will contain `R` + `slurm` + C++ code run the simulations, and the required plots and tables.

[2] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`

The `R-Dev` team will submit 4 Figures for parts (a)-(d) and any figures/plots for part (e). The code scripts should be named `sgd.R, asgd.R, implicit.R` for the implementation of the fitting algorithms. The experiments code should be in a separate file `experiments.R` and should implement the experiments of Sections 6.1 and 6.2 in Xu [5]. Part (d) should be on a separate script under the name `sa-hypothesis.R`.

The `Cpp-Dev` team will submit 4 Tables for parts (d) and (e) and 4 respective test error figures. The code implementing the Implicit updates for Part (c) should be named `svmimplicit.cpp`.

Happy Coding!

References

[1] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[2] Jin-Ichi Nagumo and Atsuhiko Noda. A learning method for system identification. *Automatic Control, IEEE Transactions on*, 12(3):282–287, 1967.

[3] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[4] David J Sakrison. Efficient recursive estimation; application to estimating the parameters of a covariance function. *International Journal of Engineering Science*, 3(4):461–483, 1965.

[5] Wei Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, 2011.

[6] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.