The University of Texas at Austin
Department of Electrical and Computer Engineering

EE381K: Large Scale Optimization — Fall 2015

Problem Set Six

Constantine Caramanis Due: Thursday, October 29, 2015.

Reading Assignments

1. Reading: Boyd & Vandenberghe: Chapters 4 & 5.

Computational Problems

Background

Logistic Regression

Logistic regression is a simple statistical classification method which models the conditional distribution of the class variable y being equal to class c given an input $x \in \mathbb{R}^n$. We will examine two classification tasks, one classifying newsgroup posts, and the other classifying digits. In these tasks the input x is some description of the sample (e.g. word counts in the news case) and y is the category the sample belongs to (e.g. sports, politics). The Logistic Regression model assumes the class distribution conditioned on x is log-linear:

$$p(y = c|x, b_{1:C}) = \frac{e^{-b_c^\top x}}{\sum_{j=1}^{C} e^{-b_j^\top x}},$$

where C is the total number of classes, and the denominator sums over all classes to ensure that $p(y|x)$ is a proper probability distribution. Each class $c \in 1, 2, \ldots, C$ has a parameter b_c, and $\mathbf{b} \in \mathbb{R}^{nC}$ is the vector of concatenated parameters $\mathbf{b} = [b_1^\top, b_2^\top, \ldots, b_C^\top]^\top$. Let $X \in \mathbb{R}^{N \times n}$ be the data matrix where each sample $x_i^\top$ is a row and N is the number of samples. The maximum likelihood approach seeks to find the parameter $\mathbf{b}$ which maximizes the likelihood of the classes given the input data and the model:

$$\max_{b_{1:C}} p(y_{1:N}|x_{1:N}, b_{1:C}) = \prod_{i=1}^{N} p(y_i|x_i, b_{1:C}) = \prod_{i=1}^{N} \frac{e^{-b_{y_i}^\top x_i}}{\sum_{j=1}^{C} e^{-b_j^\top x_i}}.$$

For the purposes of optimization, we can equivalently minimize the negative log likelihood:

$$\min_{\mathbf{b}} \ell(\mathbf{b}) = -\log p(\mathbf{y}|X, \mathbf{b}) = \sum_{i=1}^{N} \left(b_{y_i}^\top x_i + \log \sum_{j=1}^{C} e^{-b_j^\top x_i} \right).$$

After optimization, the model can be used to classify a new input by choosing the class that the model predicts as having the highest likelihood; note that we don't have to compute the normalizing quantity $\sum_{j=1}^{C} e^{-b_j^\top x}$ as it is constant across all classes:

$$y = \arg\max_j p(y = j|x, \mathbf{b}) = \arg\min_j b_j^\top x$$

In this problem, you will optimize the logistic regression model for the two classification tasks mentioned above which vary in dimension and number of classes. The digits dataset has $C = 10$ classes, while the newsgroup dataset has $C = 20$. We focus on two 1st order methods which fill the gap between gradient descent and full Newton's method by accumulating 2nd order information over iterations (without the need for direct access to the Hessian, as in Newton's). Specifically, you will implement and compare Nonlinear Conjugate Gradient (CG) and a limited-memory variant of BFGS, L-BFGS. For more on both of these methods, see Nocedal and Wright.

Nonlinear Conjugate Gradient

Recall that the linear conjugate gradient method iteratively solves the linear equation $Ax = b$, where $A \in S^n_+$. This method can be used to approximate the Newton step by solving $\nabla^2 f(x)\Delta_{\text{nt}} = \nabla f(x)$ and stopping after k iterations, $k << n$. However, this approach, known as Newton-CG or Truncated Newton, still requires computing and storing the Hessian. In this section, we will explore a generalization of CG to nonlinear problems which additionally removes the requirement of having oracle access to the Hessian.

To motivate the nonlinear case, lets re-examine the linear CG algorithm. As A is positive semi-definite, it can be decomposed as $A = M^\top M$ for some matrix M. Thus linear conjugate gradient can be thought of as minimizing the least squares problem $\min_x f(x) = ||Mx - d||$ where $b = M^\top d$. The linear equation $Ax = b$ then comes from minimizing the quadratic $f(x)$ by setting $\nabla f(x) = 0$.

The nonlinear algorithm extends these ideas to general smooth, convex $f(x)$. In the resulting algorithm, the residual $r_k = b - Ax_k$ is replaced by the negative gradient $r_k = -\nabla f(x)$, and rather than having an analytic form for the step-size α_k, it is determined by line search. To form conjugate search directions sequentially, the algorithm (as in the linear case), combines the previous search direction p_{k-1} weighted by β_k with the current negative gradient $-\nabla f(x_k)$ to form the next search direction p_k. For Nonlinear CG, β_k can be computed using several formulas, which are equivalent when $f(x)$ is a quadratic. One of the most popular is the Polak-Rebiere formula:

$$\beta_k^{PR} = \frac{r_k^\top (r_k - r_{k-1})}{r_{k-1}^\top r_{k-1}}.$$

As the function being optimized may not actually be quadratic, the search directions can eventually lose conjugacy and should be reset to the gradient periodically (by e.g. setting $\beta_k = 0$ for one iteration). A popular way to do this automatically is to use $\beta_k = \max(\beta_k^{PR}, 0)$. Putting all this together, the Nonlinear Conjugate Gradient algorithm is given in Algorithm 1.

Input: Tolerance ϵ, initial point $\mathbf{x}_0 \in \mathbb{R}^n$, oracle access to th+ function to be optimized $f(x)$ and its gradient $\nabla f(x)$
Initialize: $\mathbf{r}_0 = -\nabla f(x_0)$, $\mathbf{p}_0 = \mathbf{r}_0$, $k = 0$
while $||p_k||^2 > \epsilon$ **do**
  $\alpha_k = \arg\min_\alpha f(x_k + \alpha p_k)$; `// Approximate using BTLS`
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = -\nabla f(x_{k+1})$
  $\beta_{k+1} = \max(\frac{r_{k+1}^\top (r_{k+1} - r_k)}{r_k^\top r_k}, 0)$
  $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$
  $k = k + 1$
end
return x_k

Algorithm 1: Nonlinear Conjugate Gradient

L-BFGS

BFGS similarly aggregates curvature information over the iterates using only gradient evaluations, but is motivated by Newton's method and the secant equation rather than conjugacy. Recall that the BFGS update of the approximate inverse Hessian H_{k+1} can be written as

$$H_{k+1} = (I - \rho_k \mathbf{s}_k \mathbf{v}_k^\top) H_k (I - \rho_k \mathbf{v}_k \mathbf{s}_k^\top) + \rho_k \mathbf{s}_k \mathbf{s}_k^\top,$$
$$\rho_k = \frac{1}{\mathbf{v}_k^\top \mathbf{s}_k}.$$

Here $\mathbf{s}_i = x_{i+1} - x_i$ and $\mathbf{v}_i = \nabla f(x_{i+1}) - \nabla f(x_i)$. Though this iterative process is more efficient than the $O(n^3)$ inversion of the true Hessian required for Newton's method, it still requires $O(n^2)$ computation for the matrix-vector products, as well as $O(n^2)$ storage. When n is very large, even n^2 may be computationally prohibitive.

An interesting observation is that H_k can be computed using the full history $\mathbf{s}_{1:k-1}$ and $\mathbf{v}_{1:k-1}$ using the two-loop recursion given in algorithm 2 with $m = k - 1$. This method of computing the full BFGS update for H_k requires memory and computation linear in k and is only more efficient when $k < n$, but we would like to be able to run the algorithm for an large number of iterations (i.e. large k). The idea behind L-BFGS is that computing H_k using the two-loop recursion with $m < n$ of the most recent updates often provides a sufficient approximation of H_k while reducing the memory and computational costs to $O(mn)$. To keep the storage linear, the initial approximate inverse Hessian H_k^0 is typically chosen to be diagonal. A popular choice simply scales the identity:

$$H_k^0 = \gamma_k I,$$
$$\gamma_k = \frac{\mathbf{s}_{k-1}^\top \mathbf{v}_{k-1}}{\mathbf{v}_{k-1}^\top \mathbf{v}_{k-1}}.$$

γ_k is the scaling factor that attempts to estimate the size of the true Hessian matrix along the most recent search direction. This helps to ensure that the search direction p_k is well scaled, and as a result the step length 1 is accepted in most line search iterations.

Input: Current gradient $\nabla f(x_k)$, initial approximate inverse Hessian H_k^0, update history $\mathbf{v}_{k-m:k-1}$ and $\mathbf{s}_{k-m:k-1}$.
Output: Search direction p_k which satisfies $p_k = H_k \nabla f(x_k)$
Initialization: $q = \nabla f(x_k)$
for $i = k-1, k-2, \ldots, k-m$ **do**
  $\alpha_i = \rho_i \mathbf{s}_i^\top q$
  $q = q - \alpha_i \mathbf{v}_i$
end
$p_k = H_k^0 q$
for $i = k-m, k-m+1, \ldots, k-1$ **do**
  $\beta = \rho_i \mathbf{v}_i^\top p_k$
  $p_k = p_k + \mathbf{s}_i(\alpha_i - \beta)$
end
return p_k

Algorithm 2: Two-loop recursion for computing L-BFGS search direction, equivalent to full BFGS when m=k-1

The full L-BFGS algorthm is the same descent procedure as the one given above in algorithm 1, but computing the line search direction p_k is replaced by the two-loop recursion in algorithm 2.

Assignment

The inputs X and labels $\mathbf{y}$ are separated into *test* and *train*. Perform optimization using the training data, and record performance using the test data.

1. **L-BFGS v. Nonlinear CG**

   For the datasets given in `logistic-digits.mat` and `logistic-news.mat`, compare the iteration and computational complexity of Nonlinear CG and L-BFGS. For both methods, plot $\ell(\mathbf{b})$ and the misclassification rate (1 - #correctly classified/#samples) against the number of iterations (putting both algorithms' performance on the same plot for comparison). Make a second pair of plots with $\ell(\mathbf{b})$ and misclassification rate against computation time (or, even better, the number of arithmetic operations).

   Discuss any difference in performance you observe between the two algorithms. Is the story different for iterations vs. computation time? What about for small vs. larger datasets? Does either method require more function evaluations during line search? How does their behavior compare to gradient descent?

2. **BFGS v. L-BFGS**

   Using the data in `logistic-digits.mat` compare the iteration and computational complexity of BFGS with L-BFGS for varying values of m, the length of the stored history. Using several values of m, plot the test loss $\ell(\mathbf{b})$ against the number of iterations and (separately) against the computation time for both BFGS and L-BFGS. Discuss the difference in performance in terms of iteration complexity and computation time. How does m affect this relationship?

3. **Full Comparison (optional)**

   Compare the performance of full Newton's method, truncated Newton's (using a small number of linear CG iterations), Nonlinear CG, BFGS, and L-BFGS on the digits dataset. Note that the first two methods listed require explicit computation of the Hessian. Measure performance by the test loss $\ell(\mathbf{b})$ and plot this against the number of iterations and computation time (as above). Discuss the iteration, computational, and storage complexity of the various methods. Is there a pronounced difference between the methods that require access to the Hessian and those that only use gradient information?

Written Problems

1. **Compressed Sensing** Consider the following optimization problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \|x\|_1 \\ s.t. \quad & Ax = b \end{aligned}$$

   Write this as a linear program. Find its dual.

2. Problem 5.7 in the textbook, Boyd and Vandenberghe.

3. **Exponential Families** In this problem we investigate the natural motivation for an important class of distributions: exponential families. Let X be a discrete[1] random variable, with

[1] The same property holds for all random variables, but we will keep it discrete here for simplicity.

possible values $x \in \mathcal{X}$. Given a set of functions $\{\phi_k(x)\}$, the corresponding exponential family is all probability mass functions of the form

$$p(x) \;=\; \frac{1}{Z(\theta)} \exp\left(\sum_k \theta_k \phi_k(x)\right) \tag{1}$$

where all $\theta_k \in \mathbb{R}$ and $Z(\theta)$ is a normalizing constant. Examples include bernoulli, exponential, gaussian, poisson etc.

(a) Consider the entropy function $H(p) := -\sum_x p(x) \log p(x)$. As is well known, the higher the entropy of a random variable, the "more random" it is. Show that $H(\cdot)$ is a concave function of p.

(b) Consider the following optimization problem, which maximizes entropy subject to moment constraints on certain functions:

$$\begin{aligned} \max_p \quad & H(p) \\ s.t. \quad & E_p[\phi_k(X)] = a_k \quad \text{for all } k \end{aligned}$$

where $E_p[\cdot]$ is the expectation when X has pmf p. Why is this a convex program ?

(c) Given a set of functions $\{\phi_k(x)\}$, show that the optimum of the convex program above is a pmf in the *corresponding* exponential family (1).

4. **Fast-Mixing Markov Chains** A doubly-stochastic matrix P is a symmetric matrix with non-negative entries such that every row and every column sums up to 1. Its leading eigenvalue is always 1, corresponding to the eigenvector $\mathbf{1}$. Consider the absolute values of all the other eigenvalues of P, say $\lambda_2(P) \geq \ldots \geq \lambda_n(P)$, and let $\mu(P) := \max_{i \neq 1} |\lambda_i(P)|$ denote the largest such absolute value[2].

   (a) Show that $\mu(P)$ is a convex function of P. *(Hint: $\mu(P) = \max\{\lambda_2(P), -\lambda_n(P)\}$).*

   (b) Write $\mu(P)$ as the spectral norm of P minus another matrix. (Recall: for symmetric matrices, spectral norm is the largest absolute value of an eignevalue.) *(Hint: all eigenvectors are orthogonal.)*

5. **Duality in graph theory**. A graph is a set of nodes, $v_i \in V$, and a set of edges $e_{ij} \in E$, joining them. Given a graph with non-negative edge weights $w_{ij} \geq 0$, the *max-weight matching* problem is: find the heaviest set of disjoint edges (i.e., no two edges in the set share a node). The *min-weight vertex cover* problem is: put non-negative weights u_i on each vertex, so that *(a)* for every edge we have $w_{ij} \leq u_i + u_j$, and *(b)* the total node weights $\sum_i u_i$ is minimized. Show that the LP relaxations of these two problems are the duals of each other. *Note: there may be more than one way to relax the original binary optimization problem.*

6. **Robust Optimization**. Recall the Robust Optimization framework we introduced in class. We have a linear program,

$$\begin{aligned} \min: \quad & c^\top x \\ \text{s.t.}: \quad & a_i^\top x \leq b, \quad \forall a_i \in \mathcal{U}_i, \quad i = 1, \ldots, m, \end{aligned}$$

[2] $1 - \mu(P)$ governs the time it takes for a Markov chain, with probability transition matrix P, to converge to the unique stationary distribution $\frac{1}{n}\mathbf{1}$.

where $\mathcal{U}_i$ represents the uncertainty set. In class we considered polyhedral and ellipsoidal uncertainty sets. Now consider the following cardinality-constrained robust problem:[3] Each constraint, $a_i^\top x \leq b_i$, has some integer r_i of its entries that may deviate from some nominal value, while the remaining $(n - r_i)$ entries are known exactly. Thus we have:

$$\mathcal{U}_i = \{a = a_i^0 + \hat{a}_i \,:\, |\hat{a}_{ij}| \leq \Delta_{ij},\ |\text{supp}(\hat{a}_i)| \leq r_i\}.$$

That is, $\hat{a}_i$ is non-zero on at most r_i entries. Here, $\hat{a}_{ij}$ is the j^{th} entry of the vector $\hat{a}_i$.

Show that the robust linear program can be rewritten as a linear program. Note that you have a non-convex problem to deal with, because of the cardinality constraint. The final outcome, however, is just a linear program.

[3] As motivation, consider the following: if you are modeling measurements, it may make sense to assume that all entries may be off by some amount. But if you are modeling, say, faulty components, where something either fails or does not, it may make more sense to consider the case where at most some finite number of components fail, and the others operate perfectly.