Assignment 2

CS6230: Optimization Methods in Machine Learning

Due: 24 Sep 2016, 13:00 hours
Max Marks: 100

INSTRUCTIONS

- Please submit your solutions to theory questions on an A4 sheet and hand it over to the TA (Adepu Ravi Sankar) in Room No 611, 6th floor, Academic Block A by the due date (and time). Solutions to programming questions should be uploaded as a single ZIP file named '⟨YourRollNo⟩_assign2.zip' through the Google Classroom submission link.
- Your solution to coding problems should include plots and whatever explanation necessary to answer the questions asked.
- For late submissions, 10% is deducted for each day (including weekend) late after an assignment is due. Note that each student begins the course with 5 grace days for late submission of assignments. Late submissions will automatically use your grace days balance, if you have any left. You can see your balance on the CS6230 Marks and Grace Days document under the course Google drive.
- Please read the department plagiarism policy. Do not engage in any form of cheating - strict penalties will be imposed for both givers and takers. Please talk to instructor or TA if you have concerns.

1 THEORY (70 POINTS)

[1] **[3+3+3+3=12 points]**

a. Define: (i) Q-superlinearly convergent; (ii) Q-quadratically convergent; and (iii) R-quadratically convergent.

b. Show that the sequence $x_k = \frac{1}{k}$ is not Q-linearly convergent, though it does converge to zero.

c. Does the sequence $x_k = \frac{1}{k!}$ converge Q-superlinearly? Q-quadratically? Justify your answer.

d. Consider the sequence $\{x_k\}$ defined by:

$$x_k = \begin{cases} \left(\frac{1}{4}^{2^k}\right) & k \text{ even,} \\ (x_{k-1})/k & k \text{ odd.} \end{cases}$$

Is this sequence Q-superlinearly convergent? Q-quadratically convergent? R-quadratically convergent? Justify your answer.

[2] **[8 points]** Consider $f(x) = \frac{1}{2}x^TQx - b^Tx$. If steepest descent method with exact line search $\alpha_k = \frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T Q \nabla f_k}$ is applied to this strongly convex quadratic function, show that the error norm satisfies:

$$||x_{k+1} - x^*||_Q^2 = \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 ||x_k - x^*||_Q^2$$

where $0 < \lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ are the eigen values of Q.

[3] **[3+3+3 = 9 points]** Instead of measuring the error on iteration t, we may be interested in the number of iterations required before $(f(x^t) - f(x^*)) < \epsilon$ for some small ϵ. This allows direct comparison of the runtimes of different iterative algorithms to each other and to non-iterative methods. Show the following:

a. If $(f(x^t) - f(x^*)) = O(1/t)$, then we require $t = \Omega(1/\epsilon)$ iterations.

b. If $(f(x^t) - f(x^*)) = O(1/t^2)$, then we require $t = \Omega(1/\sqrt{\epsilon})$ iterations.

c. If $(f(x^t) - f(x^*)) = O(\rho^t)$, then we require $t = \Omega(\log(1/\epsilon))$ iterations.

[4] **[5+20+16 = 41 points]** In this problem, we will work with a fixed, closed, non-empty, convex set $S \subset \mathbb{R}^N$, and assume that its diameter is a constant $D = \max_{x,y\in S} \|x - y\|_2$. Assume that F is the set of all convex functions with domain S such that every subgradient of every function is bounded by G everywhere in its domain, i.e. $\forall f \in F$, $\forall x \in S$, $\forall g_x \in \partial f(x)$, $\|g_x\|_2 \le G$.

a. **A Game against an Adversary:** We are going to play a game against an adversary that will last T rounds (T is fixed). In each round, t ($1 \le t \le T$), you have to choose a point $x_t \in S$. Simultaneously and independently, nature chooses a function $f_t \in F$. After you choose x_t, nature's choice f_t is revealed completely to you, and you incur a penalty/loss in this round of $f_t(x_t)$.

   Your aim when playing this game is to choose points x_t in order to minimize your total loss/penalty $\sum_{t=1}^T f_t(x_t)$. Clearly, this is hard because you are forced to choose x_t before finding out f_t. To make the problem easier, we are going to instead minimize *regret*. At the end of the game, you have seen $f_1, ..., f_T$, and you can calculate the best point x^* that you could have chosen, when you look back in hindsight. This x^* can be defined as the one point that minimizes your total penalty, ie $x^* = \arg\min_{x\in S} \sum_{t=1}^T f_t(x)$.

   The difference between your total loss and the loss incurred by the best fixed point is called regret, ie $R_T = \sum_{t=1}^T f_t(x_t) - \min_{x\in S} \sum_{t=1}^T f_t(x)$. Your first task is to show that solving for x^* is a convex optimization problem. **[5 points]**

b. **Projected Subgradient Descent:** Consider the following algorithm. Start at an arbitrary point $x_1 \in S$. After round t, on receiving f_t, calculate any subgradient $g_t \in \partial f_t(x_t)$. Take a step in the direction of negative subgradient g_t with constant step-size η. Since you might be outside the set, project back onto the set S and choose that as your next point x_{t+1}. So $x_{t+1} = \prod_S(x_t - \eta g_t)$ where $\prod_S(x) = \arg\min_{y\in S} \|x - y\|_2$ is the projection of x onto S (nearest point in S to x). We are going to show that the regret grows sublinearly - specifically, if we play the game for T rounds, then our regret is only going to grow like $R_T = O(\sqrt{T})$. The proof will keep track of $\|x_t - x^*\|^2$ where distances will always be in L2-norm. We will use the shorthand $x'_{t+1} = x_t - \eta g_t$ to denote the unprojected point, so now, $x_{t+1} = \prod_S(x'_{t+1})$.

   (a) Show that finding $\prod_S(x)$ is a convex optimization problem. **[3 points]**

   (b) Argue that whatever x^* might be, the distance between x_{t+1} and x^* can only be smaller than the distance between x'_{t+1} and x^*. **[3 points]**

   (c) Use the above fact and the algorithm's update rule to derive a simple recursive inequality between $\|x_{t+1} - x^*\|^2$ and $\|x_t - x^*\|^2$. Rearrange the terms to get an upper bound on $g_t^\top(x_t - x^*)$. **[3 points]**

   (d) Use the definition of convexity to upper bound $f_t(x_t) - f_t(x^*)$ which is the "regret due to round t", in terms of η, G and $[\|x_t - x^*\|^2 - \|x_{t+1} - x^*\|^2]$. **[4 points]**

(e) Sum the above upper bound over all time steps $t = 1, \dots, T$ to derive an upper bound on the regret in terms of η, G, D, T. **[3 points]**

(f) Show how to set step size η so that the total regret is at most $O(GD\sqrt{T})$. **[4 points]**

c. **Stochastic First Order Oracle Model:** In convex optimization theory, we often come across the black-box oracle model of computation. In this model, there is an oracle which has access to a hidden unknown function f, and we get to query that oracle at points x_t in the domain. A zeroth order oracle, when queried at x_t, will give you only the function value $f(x_t)$. A first order oracle gives you both $f(x_t)$ and any subgradient $g_t \in \partial f(x_t)$ (similarly, the second order oracle also provides a Hessian).

In the algorithm discussed so far, note that we didn't really need to know the function f_t to calculate our next move (we only need it to calculate our regret). In fact, given only any one subgradient $g_t \in \partial f_t(x_t)$ we could have run the same algorithm, and we would've been sure that whatever the functions were, our average regret would grow like $O(1/\sqrt{T})$. So, we say that this algorithm achieves a regret rate of $1/\sqrt{T}$ in the first order oracle model of computation. In online learning, this is also called the *full information model*, where you get the entire function f_t (and hence its gradient) rather than the bandit setting where you only get to know $f_t(x_t)$ but not the function f_t.

We will now show a reduction from a regret algorithm to traditional convex optimization, i.e., any algorithm that achieves a regret bound $\frac{\sum_{t=1}^T f_t(x_t)}{T} - f_t(x^*) = O(1/\sqrt{T})$ can be used as an algorithm to achieve a convex optimization error of $O(1/\sqrt{T})$. For this, assume that we run the same algorithm from the previous section, but with the additional knowledge that since this is a convex optimization problem (and not an online game with an adversary), we have $f_t = f$ for all time steps t.

(a) *From Regret To Error:* Let $x_1, \dots, x_T$ be the T points chosen by our algorithm. Define $\bar{x}_T = \frac{x_1 + \dots + x_T}{T}$. Explain why the following equations hold: **[6 points]**

$$f(\bar{x}_T) \le \frac{f(x_1) + \dots + f(x_T)}{T} \tag{1.1}$$

$$f(\bar{x}_T) - f(x^*) = O(1/\sqrt{T}) \tag{1.2}$$

(b) *Stochastic Model:* Stochastic convex optimization deals with the setting where a noiseless subgradient or function value may not be available (or is too costly to compute exactly). A stochastic first order oracle captures this setting by not answering a query at x_t exactly with $(f(x_t), g_t)$, but instead with $(\hat{f}(x_t), \hat{g}_t)$ satisfying $\mathbb{E}[\hat{f}(x_t)] = f(x_t)$, and $\mathbb{E}[\hat{g}_t] = g_t \in \partial f(x_t)$ with $\|\hat{g}_t\| \le G$. Here, the expectation is with respect to any randomness of the oracle for this particular query (for example, when you query at x_t, it may add Gaussian noise to the true values before returning them to you; alternatively, if f is a sum of a loss function over a very large number of training examples, it may compute an approximate subgradient at a point by only evaluating it over a random subset of examples). We will show that the exact same algorithm works with noisy subgradients by deriving a bound that looks like $\mathbb{E}(f(\bar{x}_T) - f(x^*)) = O(1/\sqrt{T})$, where the expectation is over all the randomness of the oracle during all rounds. **[10 points]**

HINT: Let $\mathbb{E}_{t-1}$ denote taking expectation of the randomness of round t *conditioned* on all randomness till round $t-1$. Justify why the following equations hold:

$$\mathbb{E}_{t-1}[\hat{g}_t] = g_t$$

$$f(x_t) - f(x^*) \le \mathbb{E}_{t-1}[\hat{g}_t]^\top (x_t - x^*)$$

$$\mathbb{E}[f(x_t)] - f(x^*) \le \mathbb{E}[\hat{g}_t]^\top (x_t - x^*)$$

The rest of the proof should carry through just like the noiseless regret case, finally proving that projected subgradient descent achieves a $O(1/\sqrt{T})$ rate for any general convex function over any general convex set in the black-box stochastic first order oracle model of optimization.

2 Programming (30 points)

Proximal gradient for sparse logistic regression: In this problem, we will employ proximal gradient methods to build a classifier for recognizing handwritten digits from images. Instead of using the pixel values of the digit images directly, we have provided features (Fourier-based) that will allow us to learn a nonlinear decision function with a linear classifier. We formulate the problem as multi-class classification with output label $y \in \{0, ..., 9\}$ and input features $x \in \mathbb{R}^n$. We model $y|x$ with the probabilistic model:

$$p(y = j-1|x) = \frac{\exp(x^T \beta_j)}{\sum_{k=1}^{10} \exp(x^T \beta_j)}$$

corresponding to a multinomial distribution and feature weights $\beta_j \in \mathbb{R}^n$ for digit $j-1$.
We will learn the weights $\beta_1, ..., \beta_{10}$ from training data by minimizing the negative log-likelihood over m training examples

$$\begin{aligned} g(\beta_1, ..., \beta_{10}) &= \sum_{i=1}^{m} -\log p(y_i | x_i; \beta_1, ..., \beta_{10}) \\ &= \sum_{i=1}^{m} \left(\log \sum_{k=1}^{10} \exp(x_i^T \beta_k) - x_i^T \beta_{y_i} \right) \end{aligned}$$

with the addition of l_1-regularization. The final optimization problem is

$$\min_{\beta_1, ..., \beta_{10}} f(\beta_1, ..., \beta_{10}) = \min_{\beta_1, ..., \beta_{10}} g(\beta_1, ..., \beta_{10}) + \lambda \sum_{k=1}^{10} ||\beta_k||_1$$

The data for this problem is from the classic machine learning dataset MNIST. This dataset has 60K training examples but in order to reduce computation time, we will use only 1K in this problem. The training and test datasets with the features we will use are in `mnist.mat` and the original images are in `mnist_original.mat` (Click on the file names to download - the files are in MATLAB format; please convert them to other formats if required).

When implementing numerical methods, it is often convenient (and more efficient) to arrange parameters in vectors and matrices rather than dealing with sums explicitly. Let

$$B = \begin{bmatrix} | & & | \\ \beta_1 & \dots & \beta_{10} \\ | & & | \end{bmatrix}, X = \begin{bmatrix} — & x_1^T & — \\ & \vdots & \\ — & x_m^T & — \end{bmatrix}$$

and let $Y \in \mathbb{R}^{m \times 10}$ be the "one-hot" encoding of the training labels: $y_{ik} = 1$ if example i has label $k-1$ and $y_{ik} = 0$ otherwise. The gradient of the smooth component of the objective is

$$\nabla_B g(B) = X^T (Z \exp(XB) - Y)$$

where exp(.) is applied elementwise and $Z \in \mathbb{R}^{mxm}$ is the diagonal matrix with

$$Z_{ii} = \frac{1}{\sum_{k=1}^{10} \exp(x_i^T \beta_k)}$$

a. Implement the proximal gradient method with and without acceleration. Fit the model parameters on the training data (X and y) using regularization parameter $\lambda = 1$. Run both algorithms for 5000 iterations with fixed step size $t = 10^{-4}$. Plot the convergence of both methods in terms of $f - f^*$ (provided as `f_star` in `mnist.mat`) scaled logarithmically on the y-axis and number of iterations on the x-axis.**[10 points]**

b. Next, we will evaluate how error rate decreases as a function of the regularization parameter λ. For this part, we will run the accelerated proximal gradient method from the previous question for 1000 iterations. Solve the problem with a sequence of logarithmically-spaced values of λ from 10^1 to 10^{-2} and plot the error rate measured on the test data (`Xtest` and `ytest`) as a function of λ. What value of λ achieves the lowest error rate on the test data? How many non-zeros does the solution B^* have for this value of λ? **[10 points]**

c. Using the best value of λ found in part (b), build a model and use it to make predictions on the test set. Using these predictions, order the examples in the test set by the maximum probability for a single class, i.e. by $\max_{k=1,..,10} p(y = k-1|x)$. Identify 5 correctly classified examples and 5 incorrectly classified examples for which the maximum single class probability is largest. Print these 10 images (the pixel data is in `mnist_original.mat`) and include them in your report. **[10 points]**