Convex Optimization 10-725/36-725
Homework 1, due September 19

Instructions:

- You must complete Problems 1–3 and **either Problem 4 or Problem 5** (your choice between the two).
- When you submit the homework, upload a single PDF (e.g., produced by LaTeX, or scanned handwritten exercises) for the solution of each problem separately, to blackboard. You should your name at the top of each file, except for the first problem. **Your solution to Problem 1 (mastery set) should appear completely anonymous to a reader.**

1 Mastery set [25 points] (Aaditya)

Be very concise. If written up and scanned, should be less than two sides in total for the whole mastery set. If Latex-ed up, should be less than one side.

A1 [2] Define $\Delta_k := \{\mathbf{x} \in \mathbb{R}^k | x_i \geq 0, \sum_i x_i = 1\}$. Let $M \subset \mathbb{R}^n$ be an arbitrary set, $C \subset \mathbb{R}^n$ be a convex set and $k \in \mathbb{N}$ be an arbitrary natural number. Show that if $x_1, ..., x_k \in C$ and $\theta_1, ..., \theta_k \in \Delta_k$, then their convex combination $y = \sum_i \theta_i x_i$ is also in C. The definition of convexity holds for $k = 2$, you need to prove it for any general $k > 2$.

A2 [3] Then, define $conv_1(M)$ to be the intersection of all convex sets containing M and define $conv_2(M)$ to be the set of all convex combinations of points in M. Use the previous proof to show that $conv_1(M) = conv_2(M)$ by showing that each set contains the other.

B1 [2+2] Is a hyperplane $HP(a, b) = \{x \in \mathbb{R}^d | a^\top x = b\}$ a convex set? Justify. What is the distance between parallel hyperplanes $HP(a, b_1)$ and $HP(a, b_2)$?

B2 [2+2] Is a half space $HS(a,b) = \{x \in \mathbb{R}^d | a^\top x \leq b\}$ a convex set? Justify. Under what conditions on a_1, b_1, a_2, b_2 does one half space $HS(a_1, b_1)$ completely contain another $HS(a_2, b_2)$?

B3 [2] For any two points $u, v \in R^d$, show using the previous parts that the set of points which is closer in euclidean distance to u than to v is a convex set.

C [2+3] Suppose $f : \mathbb{R}_+ \to \mathbb{R}$ is convex. Show that $f(sx)$ is convex for any $s > 0$. Show that its running average $F(x) = \frac{1}{x}\int_0^x f(t)dt$ is convex.

D [3+2] Convert the following LP to standard form of $\min_u c^\top u$ subject to $Au = b, u \geq 0$. $\max_{x,y,z} 3x - y + z$ subject to $-1 \leq x \leq 1$, $-1 \leq y \leq 1$, $x + y + z = 1$. What is the optimal value of this LP and attained at what point?

2 LPs and gradient descent in Stats/ML [25 points] (Sashank)

A [4+4+5] Consider the following basis pursuit problem:

$$\min_{\beta \in \mathbb{R}^p} \|\beta\|_1 \quad \text{s.t.} \quad y = X\beta. \tag{1}$$

We claim that this is equivalent to the linear program

$$\min_{\beta^+, \beta^- \in \mathbb{R}^p} 1^T(\beta^+ + \beta^-) \quad \text{s.t.} \quad y = X(\beta^+ - \beta^-), \quad \beta^+ \geq 0, \quad \beta^- \geq 0. \tag{2}$$

Please answer the following questions concisely.

(a) Argue that the optimal criterion value in (2) is always less than or equal to the optimal criterion value in (1). (Hint: think positive and negative parts.)

(b) Argue that the optimal criterion value in (2) is in fact always equal to that in (1), since given any solution $(\hat{\beta}^+, \hat{\beta}^-)$ in (2), we can always transform it into a solution in (1).

(c) Now consider the following problem:

$$\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{m} \max_{j=1,\ldots n} f_{ij}(\beta) \quad \text{s.t.} \quad y = X\beta. \tag{3}$$

where $f_{ij}(\beta) = a_{ij}^\top \beta + b_{ij}$, $a_{ij} \in \mathbb{R}^p, b_{ij} \in \mathbb{R}$ $\forall i \in \{1, \cdots m\}$ and $j \in \{1, \cdots, n\}$. This can be viewed as a generalization of (1). Write an equivalent LP for the problem (3).

B [6+6] Given pairs (x_i, y_i), for an outcome y_i and feature measurements $x_i \in \mathbb{R}^p$, $i = 1, \ldots n$, suppose that we want to fit a model of the form $y_i \approx \hat{f}(x_i)$. Boosting generally provides a way of fitting a flexible additive model

$$\hat{f}(x) = \sum_{j=1}^{M} \hat{\beta}_j \cdot h_j(x), \tag{4}$$

where $h_j : \mathbb{R}^p \to \mathbb{R}$, $j = 1, \ldots M$ are given functions of the features, called "weak learners". Here M could be very large, or even inifinite, i.e., we may have an enormous collection of possible weak learners.

In particular, gradient boosting fits a model of the form (4) using something like gradient descent. The idea is to greedily build the model (4) by including one weak learner h_j at a time. Here are the details. (See also Friedman (2001), and Chapter 10 of Hastie, Tibshirani, and Friedman (2009).) We first choose a loss function $L(y_i, \hat{y}_i)$; for example, this could be $L(y, \hat{y}_i) = (y_i - \hat{y}_i)^2$, least squares loss, if we were in a regression setting. The total loss between observations $y = (y_1, \ldots y_n)$ and predictions $\hat{y} = (\hat{y}_1, \ldots \hat{y}_n)$ is then

$$F(\hat{y}) = \sum_{i=1}^{n} L(y_i, \hat{y}_i).$$

We start with coefficients $\hat{\beta}_j = 0$ for $j = 1, \ldots M$, and hence predictions $\hat{y}_i = \hat{f}(x_i) = 0$ for $i = 1, \ldots n$. Then we repeat the following steps:

i. Compute the gradient $g \in \mathbb{R}^n$ of $F(\hat{y})$, whose components are given by $g_i = \partial L(y_i, \hat{y}_i)/\partial \hat{y}_i$, evaluated at $\hat{y}_i = \hat{f}(x_i)$, $i = 1, \ldots n$.

ii. Find the weak learner that best matches the negative gradient $-g$, in the least squares sense:

$$h_j(x) = \operatorname*{argmin}_{h_\ell,\, \ell = 1 \ldots M} \left(\min_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \big(-g_i - \alpha \cdot h_\ell(x_i) \big)^2 \right).$$

iii. Step size selection: choose $\hat{\alpha}_j$ to minimize the loss when adding $h_j(x)$ to the model:

$$\hat{\alpha}_j = \operatorname*{argmin}_{\alpha_j \in \mathbb{R}} \sum_{i=1}^{n} L\big(y_i, \hat{y}_i + \alpha_j \cdot h_j(x_i)\big).$$

iv. Update our cofficients $\hat{\beta}_j \leftarrow \hat{\beta}_j + \hat{\alpha}_j$ and hence our fitted model by

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{\alpha}_j \cdot h_j(x).$$

Now the questions:

(a) Suppose that y_i is a continuous measurement, i.e., we are in a regression setting. Derive the gradient boosting updates explicitly for least squares loss $L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$, and for weak learners $h_j(x_i) = x_{ij}$ (i.e., simply the jth component of the p-dimensional feature vector $x_i \in \mathbb{R}^p$), assuming that the features all have unit norm,

$$\sum_{i=1}^{n} x_{ij}^2 = 1, \quad \text{for all } j = 1, \ldots p.$$

What algorithm from lecture does this procedure remind you of?

(Hint: it may save you some algebra to work in matrix notation.)

(b) Suppose that $y_i \in \{-1, 1\}$, i.e., we are in a classification setting. In this setting we actually take the final classifier to be $\text{sign}(\hat{y}_i) = \text{sign}(\hat{f}(x_i))$, since $\hat{f}(x)$ is still real-valued. What are the gradient boosting steps under binomial deviance loss

$$L(y_i, \hat{y}_i) = \log(1 + \exp(-2y_i\hat{y}_i)),$$

and arbitrary weak learners h_j, $j = 1, \ldots M$, written as explicitly as possible? Can $\hat{\alpha}_j$ be determined explicitly here? If not, what is an alternative fitting method for $\hat{\alpha}_j$ in step iii of gradient boosting?

(Hint: again, it may help to work in matrix notation; and for fitting $\hat{\alpha}_j$, think of what is done in gradient descent.)

3 Programming gradient descent [25 points] (Yifei)

There are four functions to be minimized—a quadratic function, a ridge regularized logistic regression, the Himmelblaus function and Rosenbrocks banana function:

$$f_Q(x, y) = 1.125x^2 + 0.5xy + 0.75y^2 + 2x + 2y \tag{5}$$
$$f_{LL}(x, y) = 0.5(x^2 + y^2) + 50 \cdot \log(1 + \exp(-0.5y)) + 50 \cdot \log(1 + \exp(0.2x)) \tag{6}$$
$$f_H(x, y) = 0.1(x^2 + y - 11)^2 + 0.1(x + y^2 - 7)^2 \tag{7}$$
$$f_R(x, y) = 0.002(1 - x)^2 + 0.2(y - x^2)^2 \tag{8}$$

This problem will involve programming. We recommend that you use Matlab or R. Still, you can use another language of your choosing (e.g., Python), but we will only provide support for Matlab and R.

(a) (5 pts) Plot the four functions as in Figure 1(a). The first three functions are on the domain $[-6, 6] \times [-6, 6]$ and the last f_R is on $[-3, 3] \times [-6, 6]$. Which functions are convex? Which are not?

(b) (8 pts) Implement gradient descent with fixed step sizes 0.3 and 0.8 and starting from $(2, 3)^T$, and 1 random point within the range in the previous question. Run 1000 iterations. For every function, every step size, and every initialization, plot the trajectory on top of the contour of the function, like Figure 1(b,c).

(c) (6 pts) Implement gradient descent with back tracking line search from the same starting points. The initial step size is 1, the back track ratio is 0.5, the slope ratio is 0.5, the maximum number of back tracks is 10. Run 1000 iterations and plot one trajectory on top of the contour for every function and every initialization.

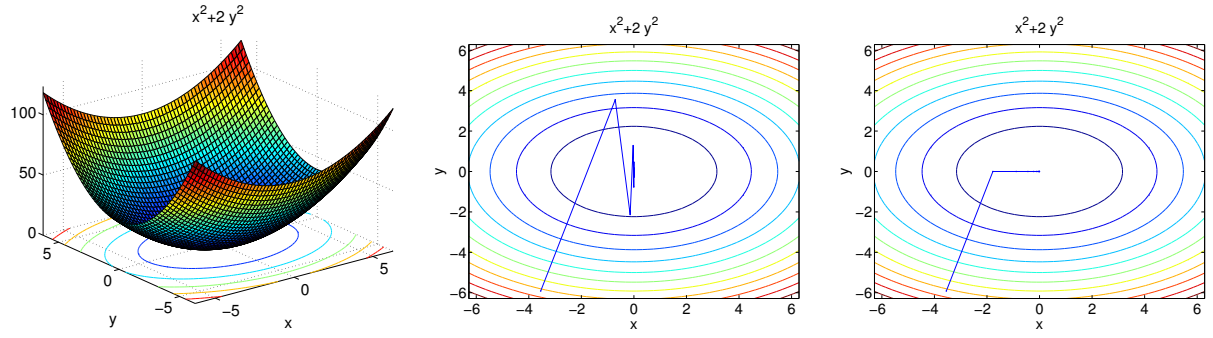
(d) (4 pts) Implement gradient descent from the same starting points with step size $= 1/k$, where k is the current count of iteration. Run 1000 iterations and plot one trajectory on top of the contour for every function and every initialization.

(e) (2 pts) For some problems, why does the large step size not work? [Hint: check the eigenvalues of the Hessian.] What are the two criteria needed for the shrinking step sizes?

Along with your report please submit your code as well in a zip file to blackboard. Your implementation should be commented well enough to be able to understand, and it should also contain a batch file that can reproduce (up to a randomizing effect) all your plots.

Hint: The attached code (demo.m) and the following two missing functions generate plots similar to Figure 1.

```
function [Xs, fcn_vals] = gd_fixed_stepsize( fcn, grad_fcn, ...
  init_X, nb_steps, stepsize_fixed);
function [Xs_bt, fcn_vals_bt, stepsizes_bt] = gd_backtrack( fcn, grad_fcn, ...
  init_X, nb_steps, stepsize0_bt, bt_rate, nb_bt_steps, slope_ratio);
```



(a) function visualization (b) fixed stepsize 0.4 (c) backtracking

Figure 1: Visualization and gradient descent trajectories for $f_{demo}(x, y) = x^2 + 2y^2$.

4 Convergence rate of subgradient method [25 points] (Adona)

Recall that the subgradient method for minimizing a convex function $f : \mathbb{R}^n \to \mathbb{R}$ starts with an initial point $x^{(0)} \in \mathbb{R}^n$, and repeats for $k = 1, 2, 3, \ldots$

$$x^{(k)} = x^{(k-1)} - t_k \cdot g^{(k-1)},$$

where $g^{(k-1)} \in \partial f(x^{(k-1)})$, a subgradient of f evaluated at $x^{(k-1)}$. Since this is not necessarily a descent method, at each iteration k we keep track of the best iterate $x^{(k)}_{\text{best}}$ among $x^{(0)}, x^{(1)}, \ldots x^{(k)}$, defined so that

$$f(x^{(k)}_{\text{best}}) = \min_{i=0,\ldots k} f(x^{(i)}).$$

Here we will derive a basic inequality for bounding $f(x^{(k)}_{\text{best}}) - f(x^\star)$, with $x^\star$ being a minimizer of f, and we will use this bound to analyze the convergence of subgradient method under various choices of step sizes.

In addition to assuming that f is convex, we will also assume that f is Lipschitz continuous with constant $G > 0$, i.e.,

$$|f(x) - f(y)| \leq G\|x - y\|_2.$$

We will denote $R = \|x^{(0)} - x^\star\|_2$.

(a) (4 points) Use $x^{(k)} = x^{(k-1)} - t_k g^{(k-1)}$ and the definition of a subgradient of f at $x^{(k-1)}$ to show that

$$\|x^{(k)} - x^\star\|_2^2 \leq \|x^{(k-1)} - x^\star\|_2^2 - 2t_k\big(f(x^{(k-1)}) - f(x^\star)\big) + t_k^2\|g^{(k-1)}\|_2^2.$$

(b) (5 points) Iterate the inequality from part (a), and use the Lipschitz assumption, and $R = \|x^{(0)} - x^\star\|_2$, to show that

$$\|x^{(k)} - x^\star\|_2^2 \leq R^2 - 2\sum_{i=1}^k t_i\big(f(x^{(i-1)}) - f(x^\star)\big) + G^2\sum_{i=1}^k t_i^2.$$

(Hint: prove that if f is Lipschitz with constant G, then this implies a bound on the norm of its subgradients.)

(c) (4 points) Use $\|x^{(k)} - x^\star\|_2 \geq 0$, and rearrange the result of part (b) to give an upper bound on $2\sum_{i=1}^k t_i(f(x^{(i-1)}) - f(x^\star))$. Then use the definition of $f(x^{(k)}_{\text{best}})$ to conclude that

$$f(x^{(k)}_{\text{best}}) - f(x^\star) \leq \frac{R^2 + G^2\sum_{i=1}^k t_i^2}{2\sum_{i=1}^k t_i}. \tag{9}$$

We'll call this our basic inequality.

(d) (4 points) Consider a constant step size $t_k = t$ for all $k = 1, 2, 3, \ldots$. Plug this into our basic inequality in (9), and take the limit as $k \to \infty$. What do you conclude?

(e) (4 points) Consider a sequence of step sizes satisfying

$$\sum_{i=1}^{\infty} t_i^2 < \infty \quad \text{and} \quad \sum_{i=1}^{\infty} t_i = \infty.$$

For such step sizes, take the limit as $k \to \infty$ in the basic inequality (9). Now what do you conclude?

(f) (4 points) Finally, fixing some number of iterations k, consider the choice of step size $t_i = R/(G\sqrt{k})$ for all $i = 1, \ldots k$. Plugging this into the basic inequality (9), what upper bound do you have on $f(x_{\text{best}}^{(k)}) - f(x^\star)$? As it turns out, this actually minimizes the bound on the right-hand side of (9). Hence, from this, what would be the best we can prove about the convergence rate of the subgradient method?

5 Gradient boosting for digit classification [25 points] (Adona)

In this problem you will implement gradient boosting for a classification task using stumps as weak learners. Recall the description of gradient boosting in Problem 2, part B. In the current classification setting, we have $y_i \in \{-1, 1\}$, but our predictions $\hat{y}_i = \hat{f}(x_i)$ can be actually real-valued (due to the nature of how we fit $\hat{f}$), so we take $\text{sign}(\hat{y}_i) = \text{sign}(\hat{f}(x_i))$ as the class of x_i, and generally, $\text{sign}(\hat{f}(x))$ as our classifier for an input $x \in \mathbb{R}^p$. We will use binomial deviance loss, i.e., the loss between an observation $y_i \in \{-1, 1\}$ and a prediction $\hat{y}_i \in \mathbb{R}$ is given by

$$L(y_i, \hat{y}_i) = \sum_{i=1}^{n} \log\big(1 + \exp(-2y_i\hat{y}_i)\big).$$

As mentioned, our weak learners $h_\ell : \mathbb{R}^p \to \mathbb{R}$, $\ell = 1, \ldots M$ will be stumps: these are just trees of depth 2. We can actually think about indexing the set of all stumps on $\mathbb{R}^p$ by three parameters $\ell = (j, c, s)$, with $j \in \{1, \ldots p\}$ being the variable to be split, $c \in \mathbb{R}$ being the split point, and $s \in \{-1, 1\}$ being the side of the split point that we attribute to class 1. Therefore, a stump $h_{j,c,s}$ gives a prediction

$$h_{j,c,s}(x) \;=\; \begin{cases} 1 & \text{if } s \cdot x_j \geq c \\ -1 & \text{otherwise} \end{cases} \;=\; 2 \cdot 1\{s \cdot x_j \geq c\} - 1.$$

You may think the set of such stumps is infinite (since $c \in \mathbb{R}$), but when fit to a set of features $x_1, \ldots x_n \in \mathbb{R}^p$, it is effectively finite, because for each $j = 1, \ldots p$, we suffer no

loss of generality in letting c range over the midpoints between each pair of adjacent points $x_{1j}, x_{2j}, \ldots x_{nj}$ in sorted order. Your fitted model will have the form

$$\hat{f}(x) = \sum_{\ell=(j,c,s)} \hat{\beta}_j \cdot \big(2 \cdot 1\{s \cdot x_j \geq c\} - 1\big), \tag{10}$$

and remember, you will classify according to $\text{sign}(\hat{f}(x))$.

(a) (15 points) Write a function—in R or Matlab (you can also use Python, or any language of your choosing, but we will only provide support for R and Matlab)—to perform gradient boosting stumps as weak learners, under binomial deviance loss. Your function should be able to use either a fixed step size $\hat{\alpha}_\ell$ at each iteration (step iii of gradient boosting as described in Problem 2, part B), or backtracking line search to adaptively fit $\hat{\alpha}_\ell$ at each iteration. Its termination criterion should just be some maximum number of iterations, to be specified by the user.

You may approach this coding problem in any way you see fit, but we recommend breaking down the task into the following subparts.

- Write a function to output predictions from the model (10), given coefficients $\hat{\beta}_\ell$ corresponding to stumps $\ell = (c, j, s)$.
- Write a function to find the closest stump $\ell = (j, c, s)$, in the least squares sense, to an arbitrary vector $g \in \mathbb{R}^n$; i.e., this is the stump that minimizes

  $$\min_{\alpha \in \mathbb{R}} \sum_{i=1}^n \big(- g_i - \alpha \cdot h_{j,c,s}(x_i)\big)^2,$$

  over all possible choices of j, c, s. (Note: this will be used in step ii of gradient boosting.)
- Write a function to perform backtracking line search given a stump $h_{j,c,s}$ and current predictions $\hat{y} = (\hat{y}_1, \ldots \hat{y}_n)$; i.e., for backtracking parameters γ_1, γ_2, this function begins with some large initial value for the step size α, and while:

  $$\sum_{i=1}^n L\big(y_i, \hat{y}_i + \alpha \cdot h_{j,c,s}(x_i)\big) > \sum_{i=1}^n L(y_i, \hat{y}_i) - \gamma_1 \cdot \alpha \|g\|_2^2,$$

  it repeats $\alpha \leftarrow \gamma_2 \cdot \alpha$. Here g is the gradient, i.e., $g_i = \partial L(y_i, \hat{y}_i)/\partial \hat{y}_i$ for $i = 1, \ldots n$. (Note: this will be used in step iii of gradient boosting.)
- Write the loop for gradient boosting, plugging in the relevant subfunctions, and finally write the master function for gradient boosting that takes in all of the relevant arguments.

(b) (10 points) Download the zip code data from the website for the Elements of Statistical Learning book, `http://www-stat.stanford.edu/~tibs/ElemStatLearn`—go to the "Data" link on the left hand side, and then scroll to the bottom of the page for the zip

code data. Download both the training and test sets; read the info to figure out how to appropriately parse these texts files with R or Matlab. (These are 16×16 grayscale images that have been unraveled in 256 dimensional vectors; you can plot any row of the training or test set matrix to convince yourself that you're looking at handwritten digits.)

For digits "5" and "6" as classes -1 and 1, respectively, run your gradient boosting algorithm from part (a) on the training set. You should keep track of the test error of the fitted classifier at each iteration, over 1000 iterations. In particular, run gradient boosting for several different choices of fixed step sizes, and choices of backtracking parameters, and plot the training and test set error curves as a function of iteration. You should aim for 5-6 plots. What do you see as the difference between small, fixed step sizes and the more aggressive backtracking strategy? Your discussion should cover both the training and test error curves (and it shouldn't be all that long!).