CPSC 540 Assignment 4 (due October 15)

Convex Functions and Linear Classifiers

Please put your name and student number on the assignment, staple the assignment together, and you will get 1 bonus point if the assignment is *typed* rather than written.

1 Inequalities involving Norms

To analyze sequences of (possibly random) variables, we will need to start using sequences of inequalities and using various properties of norms. Please read the *Notes on Norms* document on the course webpage, then show the following:

1. "Not the triangle inequality" inequality for Euclidean norm:

$$\|x+y\|_2^2 \leq 2\|x\|_2^2 + 2\|y\|_2^2.$$

   (hint: start from $\|x-y\|_2^2 \geq 0$, reverse inequality and expand, add right side, complete the square)

2. Triangle inequality for Euclidean norm:

$$\|x+y\|_2 \leq \|x\|_2 + \|y\|_2.$$

   (hint: square the left-hand side, use Cauchy-Schwartz, complete the square)

3. Relationships between sizes of p-norms on vectors of length d:

$$\|x\|_\infty \leq \|x\|_1 \leq d\|x\|_\infty \qquad \text{(Hint: } |x_i| \leq \max_j\{|x_j|\}\text{)}$$

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{d}\|x\|_\infty \qquad \text{(Hint: work with the square)}$$

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{d}\|x\|_2 \qquad \text{(Hint: } \|x\|_1 = x^T\text{sign}(x)\text{)}$$

4. All norms are convex.
   (hint: since norms are not differentiable you need to use the most general convexity definition).

2 Showing Functions are Convex

Use one of the definitions of convexity to show that the following functions are convex:

1. $f(x) = \max_i\{x_i\}$ (hint: $\max_i\{a_i+b_i\} \leq \max_i\{a_i\} + \max_i\{b_i\}$)
2. $f(x) = \sum_{i=1}^d x_i \log x_i$ for $x_i > 0$ (negative entropy)
3. $f(x) = \sum_{i=1}^N \log(1+\exp(-b_i x^T a_i))$ (logistic regression: use Hessian structure as with least squares)

Use the results above and from class, along with the operations that preserve convexity, to show that the following functions are convex:

4. $f(x) = \sum_{i=1}^{N} \log(1 + \exp(-b_i x^T a_i)) + \lambda \|x\|_1$ (ℓ_1-regularized logistic regression)
5. $f(x) = \|Ax - b\|_p + \|x\|_p$ (regularized regression with arbitrary p-norms)
6. $f(x) = C \sum_{i=1}^{N} \max\{0, |b_i - x^T a_i| - \epsilon\} + \frac{1}{2}\|x\|_2^2$ (support vector regression)

3 Gradient Methods for Logistic Regression

The function *gradientDemo.m* applies a simple gradient method (*findMin.m*) to optimize the logistic regression objective function,

$$f(w) = \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i)).$$

The function f and its gradient are computed by the function *LogisticLoss.m*. As input, the optimizer requires an initial guess as well as an 'anonymous' function that returns the function value and gradient of the function to optimize, in this case *LogisticLoss*. Modify *LogisticLoss* so that it takes a new parameter λ and computes the objective and gradient for ℓ_2-regularized logistic regression,

$$f(w) = \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2.$$

Hand in the modified *LogisticLoss.m*.

3.1 Smarter Step-Size Choices

To determine the step-size, at each iteration *findMin.m* first tries $\alpha_t = 1$ and then 'backtracks' (decreases the step-size) by halving the step-size until the following 'Armijo' condition is satisfied,

$$f(w^{t+1}) \leq f(w^t) - \gamma \alpha_t \|\nabla f(w^t)\|^2.$$

This textbook strategy guarantees convergence (for $\gamma \leq 1/2$), but in practice often requires too many evaluations of the function. There are much more clever ways to choose the values of α_t to try, and in this question you will implement two such tricks:

1. When we try a value of α_t and it fails, instead of just dividing it by 2 we can use queries to the function that we have available to propose smarter choices. In particular, if have a value α_0 that fails the Armijo condition, we could next test the minimizer of the quadratic function that interpolates $f(w^t)$ and $f(w^t - \alpha_0 \nabla f(w^t))$ and agrees with the directional derivative of f at w^t in the negative-gradient direction. This value is given by

$$\alpha_t = \frac{\alpha_0^2 \|\nabla f(w^t)\|^2}{2[f(w^t - \alpha_0 \nabla f(w^t)) - f(w^t) + \alpha_0 \|\nabla f(w^t)\|^2]}.$$

   Modify the line-search so that instead of trying $\alpha_t = \alpha_0/2$ in half, it uses this choice.

2. We could also consider initializing α_t using quadratic interpolation, but a smarter choice is set it to satisfy the 'quasi-Newton' conditions in a least squares sense (the 'Barzilai-Borwein' step-size). This choice of step-size typically gives much better practical performance. It is given by

$$\alpha_t = -\alpha_{t-1} \frac{(y^t)^T \nabla f(w^{t-1})}{\|y^t\|^2},$$

where we have used

$$y^t = \nabla f(w^t) - \nabla f(w^{t-1}).$$

Modify the optimization code so that instead of re-setting α to one on each iteration, it uses this choice. (Reset the step-size to one if the value of α is not in a reasonable range, such as $\alpha \notin [10^{-10}, 10^{10}]$.)

Hand in the modified *findMin.m* implementing these modifications.

4 Efficient Sparse Stochastic subgradient for SVMs

The function *stochGradDemo.m* applies a stochastic subgradient method to optimize the SVM objective function. However, in this demo the dataset is very sparse (each feature vector x_i only has a few non-zero entries) so the iterations are relatively expensive because the gradient of the regularizer is dense (this happens, for example, with text data where each feature could represent whether a particular English word is present). We can make the iterations only depend on the number of non-zero features in the randomly chosen x_i by representing our parameter vector as $w = \beta v$, where β is a scalar and v is a vector. Modify the demo to use this representation, so that the iteration cost is proportional to the number of non-zero elements in the selected x_i rather and not the total number of variables d. Hand in the modified part of the code.

Hint:

The update can be written

$$w^{t+1} = w^t - \alpha_t(g(w^t) + \lambda w^t),$$

where $g(w^t)$ is a subgradient of the hinge loss at w^t. This subgradient is sparse but the problem comes because adding λw^t is a dense operation. Let's re-write the iteration as

$$w^{t+1} = (1 - \alpha_t\lambda)w^t - \alpha_t g(w^t),$$

and then let's split the update into two parts:

$$w^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)w^t, \quad w^{t+1} = w^{t+\frac{1}{2}} - \alpha_t g(w^t).$$

The second part is fine, but we want to implement the first part without doing a full-vector operation. Using the representation $w = \beta v$, we can implement the above two updates to maintain $w = \beta v$ using

$$\beta^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)\beta^t, \quad v^{t+\frac{1}{2}} = v^t,$$

$$\beta^{t+1} = \beta^{t+\frac{1}{2}}, \quad v^{t+1} = v^{t+\frac{1}{2}} - \frac{\alpha_t}{\beta^{t+1}} g(\beta^t v^t).$$

The second line uses that we want v^{t+1} to satisfy $w^{t+1} = \beta^{t+1}v^{t+1} = \beta^{t+1}v^{t+\frac{1}{2}} - \alpha_t g(\beta^t v^t)$. The update of v depends on the sparsity of $g(x^t)$ but the update of β is constant time, so these updates don't depend on the total number of variables.

A problem with this update is if $(1 - \alpha_t\lambda) = 0$ then you set $\beta^{t+\frac{1}{2}} = 0$. You need to test for this case, and if it occurs an alternate update then maintains $\beta^{t+\frac{1}{2}}v^{t+\frac{1}{2}} = w^{t+\frac{1}{2}}$ is to set $\beta^{t+\frac{1}{2}} = 1$ and $v^{t+\frac{1}{2}} = 0$. The latter is a full vector operation but you only need to do it once.