CPSC 540 Assignment 2 (due February 1 at midnight)

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1-3. However, please only hand in one assignment for the group.

1. Name(s):
2. Student ID(s):

1 Calculation Questions

1.1 Convexity
Show that the following functions are convex, by only using one of the definitions of convexity (i.e., without using the “operations that preserve convexity” or using convexity results stated in class):

1. L2-regularized weighted least squares: \( f(w) = \frac{1}{2}(Xw - y)^\top V(Xw - y) + \frac{\lambda}{2}\|w\|^2 \). (\( V \) is a diagonal matrix with positive values on the diagonal).

2. Poisson regression: \( f(w) = -y^\top Xw + 1^\top v \) (where \( v_i = \exp(w^\top x_i) \)).

3. Weighted infinity-norm: \( f(w) = \max_{j \in \{1, 2, \ldots, d\}} L_j |w_j| \) (where each \( L_j \geq 0 \)).

Hint: Max and absolute value are not differentiable in general, so you cannot use the Hessian for this question.

Show that the following functions are convex (you can use results from class and operations that preserve convexity if they help):

4. Regularized regression with arbitrary \( p \)-norm and weighted \( q \)-norm: \( f(w) = \|Xw - y\|_p + \lambda\|Aw\|_q \).

5. Support vector regression: \( f(w) = \sum_{i=1}^{N} \max\{0, |w^\top x_i - y_i| - \epsilon\} + \frac{\lambda}{2}\|w\|^2_2 \).

6. Indicator function for linear constraints: \( f(w) = \begin{cases} 0 & \text{if } Aw \leq b \\ \infty & \text{otherwise} \end{cases} \).

1.2 Convergence of Gradient Descent
For these questions it will be helpful to use the “convexity inequalities” notes posted on the webpage.

1. In class we showed that if \( \nabla f \) is \( L \)-Lipschitz continuous and \( f \) is bounded below then with a step-size of \( 1/L \) gradient descent is guaranteed to have found a \( w^k \) with \( \|\nabla f(w^k)\|^2 \leq \epsilon \) after \( t = O(1/\epsilon) \) iterations. Suppose that a more-clever algorithm exists which, on iteration \( t \), is guaranteed to have found a \( w^k \) satisfying \( \|\nabla f(w^k)\|^2 \leq 2L(f(w^0) - f^*)/t^{3/3} \). How many iterations of this algorithm would we need to find a \( w^k \) with \( \|\nabla f(w^k)\|^2 \leq \epsilon \)?

2. In practice we typically don’t know \( L \). A common strategy in this setting is to start with some small guess \( L^0 \) that we know is smaller than the true \( L \) (usually we take \( L^0 = 1 \)). On each iteration \( k \), we

\(^1\)That \( C^0 \) convex functions are below their chords, that \( C^1 \) convex functions are above their tangents, or that \( C^2 \) convex functions have a positive semidefinite Hessian.
initialize with $L^k = L^{k-1}$ and we check the inequality
\[ f \left( w^k - \frac{1}{L^k} \nabla f(w^k) \right) \leq f(w^k) - \frac{1}{2L^k} \| \nabla f(w^k) \|^2. \]

If this is not satisfied, we double $L^k$ and test it again. This continues until we have an $L^k$ satisfying the inequality, and then we take the step. Show that gradient descent with $\alpha_k = 1/L^k$ defined in this way has a linear convergence rate of
\[ f(w^k) - f(w^*) \leq \left( 1 - \frac{\mu}{2L^k} \right)^k [f(w^0) - f(w^*)], \]
if $\nabla f$ is $L$-Lipschitz continuous and $f$ is $\mu$-strongly convex.

Hint: if a function is $L$-Lipschitz continuous that it is also $L'$-Lipschitz continuous for any $L' \geq L$.

3. Suppose that, in the previous question, we initialized with $L^k = \frac{1}{2}L^{k-1}$. Describe a setting where this could work much better.

4. In class we showed that if $\nabla f$ is $L$-Lipschitz continuous and $f$ is strongly-convex, then with a step-size of $\alpha_k = 1/L$ gradient descent has a convergence rate of
\[ f(w^k) - f(w^*) = O(\rho^k). \]
Show that under these assumptions that a convergence rate of $O(\rho^k)$ in terms of the function values implies that the iterations have a convergence rate of
\[ \|w^k - w^*\| = O(\rho^{k/2}). \]

1.3 Beyond Gradient Descent

1. We can write the proximal-gradient update as
\[ w^{k+1} = w^k - \alpha_k \nabla f(w^k) \]
\[ w^{k+1} = \arg\min_{v \in \mathbb{R}^d} \left\{ \frac{1}{2} \| v - w^{k+\frac{1}{2}} \|^2 + \alpha_k r(v) \right\}. \]

Show that this is equivalent to setting
\[ w^{k+1} = \arg\min_{v \in \mathbb{R}^d} \left\{ f(w^k) + \nabla f(w^k) \top (v - w^k) + \frac{1}{2\alpha_k} \| v - w^k \|^2 + r(v) \right\}. \]

2. The “sum” version of multi-class SVMs uses an objective of the form
\[ f(W) = \sum_{i=1}^n \sum_{c \neq y_i} [1 - w_i \top x_i + w_c \top x_i]^+ + \frac{\lambda}{2} \| W \|_F^2, \]
where $[\gamma]^+$ sets negative values to zero (and you can use $k$ as the number of classes so the inner loop is over $(k-1)$ elements). Derive the sub-differential of this objective.

3. In some situations it might be hard to accurately compute the elements of the gradient, but we might have access to the sign of the gradient (this can also be useful in distributed settings where communicating one bit for each element of the gradient is cheaper than communicating a floating
point number for each gradient element). Consider an \( f \) that is bounded below and where \( \nabla f \) is Lipschitz continuous in the \( \infty \)-norm, meaning that
\[
f(v) \leq f(u) + \nabla f(u)^\top (v - u) + \frac{L_\infty}{2} \|v - u\|_\infty^2,
\]
for all \( v \) and \( w \) and some \( L_\infty \). For this setting, consider a sign-based gradient descent algorithm of the form
\[
w^{k+1} = w^k - \frac{\|\nabla f(w^k)\|_1}{L_\infty} \text{sign}(\nabla f(w^k)),
\]
where we define the sign function element-wise as
\[
\text{sign}(w_j) = \begin{cases} 
+1 & w_j > 0 \\
0 & w_j = 0 \\
-1 & w_j < 0 
\end{cases}
\]
Show that this sign-based gradient descent algorithm finds a \( w^k \) satisfying \( \|\nabla f(w^k)\|^2 \leq \epsilon \) after \( t = O(1/\epsilon) \) iterations.

2 Computation Questions

2.1 Proximal-Gradient

If you run the demo `example_group.jl`, it will load a dataset and fit a multi-class logistic regression (softmax) classifier. This dataset is actually linearly-separable, so there exists a set of weights \( W \) that can perfectly classify the training data (though it may be difficult to find a \( W \) that perfectly classifies the validation data). However, 90\% of the columns of \( X \) are irrelevant. Because of this issue, when you run the demo you find that the training error is 0 while the test error is something like 0.2980.

1. Write a new function, `logRegSoftmaxL2`, that fits a multi-class logistic regression model with L2-regularization (this only involves modifying the objective function). Hand in the modified loss function and report the validation error achieved with \( \lambda = 10 \) (which is the best value among powers to 10). Also report the number of non-zero parameters in the model and the number of original features that the model uses.

2. While L2-regularization reduces overfitting a bit, it still uses all the variables even though 90\% of them are irrelevant. In situations like this, L1-regularization may be more suitable. Write a new function, `logRegSoftmaxL1`, that fits a multi-class logistic regression model with L1-regularization. You can use the function `findMinL1`, which minimizes the sum of a differentiable function and an L1-regularization term. Report the number of non-zero parameters in the model and the number of original features that the model uses.

3. L1-regularization achieves sparsity in the `model parameters`, but in this dataset it’s actually the `original features` that are irrelevant. We can encourage sparsity in the original features by using group L1-regularization. Write a new function, `proxGradGroupL1`, to allow (disjoint) group L1-regularization. Use this within a new function, `softmaxClassifierGL1`, to fit a group L1-regularized multi-class logistic regression model (where rows of \( W \) are grouped together and we use the L2-norm of the groups). Hand in both modified functions (`logRegSoftmaxGL1` and `proxGradGroupL1`) and report the validation error achieved with \( \lambda = 10 \). Also report the number of non-zero parameters in the model and the number of original features that the model uses.


2.2 Coordinate Optimization

The function `example_CD.jl` loads a dataset and tries to fit an L2-regularized least squares model using coordinate descent. Unfortunately, if we use $L_f$ as the Lipschitz constant of $\nabla f$, the runtime of this procedure is $O(d^3 + nd^2 \frac{L_f}{\mu} \log(1/\epsilon))$. This comes from spending $O(d^3)$ computing $L_f$, having an iteration cost of $O(nd)$, and requiring $O(dL_f \mu \log(1/\epsilon))$ iterations to reach an accuracy of $\epsilon$. This non-ideal runtime is also reflected in practice: the algorithm’s iterations are relatively slow and it often takes over 200 “passes” through the data for the parameters to stabilize.

1. Modify this code so that the runtime of the algorithm is $O(ndL_f \mu \log(1/\epsilon))$, where $L_f$ is the Lipschitz constant of all partial derivatives $\nabla_i f$. You can do this by increasing the step-size to $1/L_f$ (the coordinate-wise Lipschitz constant given by $\max_j \|x_j\|^2 + \lambda$ where $x_j$ is column $j$ of the matrix $X$), and modifying the iterations so they have a cost of $O(n)$ instead of $O(nd)$. Hand in your code and report an estimate of the change in time and number of iterations.

2. Besides tuning the step-size, another strategy that often improves the performance is using a (possibly-weighted) average of the iterations $w^k$. Explore whether this strategy can improve performance. Report the step-size sequence that you found gave the best performance, and the objective function value obtained by this strategy for one run.

3. A popular variation on stochastic is AdaGrad, which uses the iteration

$$w^{k+1} = w^k - \alpha_k D_k \nabla f(w^k),$$

where the element in position $(j,j)$ of the diagonal matrix $D_k$ is given by $1/\sqrt{\delta + \sum_{k'=0}^{k}(\nabla_j f_{i_{k'}}(w^{k'}))^2}$. Here, $i_k$ is the example $i$ selected on iteration $k$ and $\nabla_j$ denotes element $j$ of the gradient (and in AdaGrad we typically don’t average the steps). Implement this algorithm and experiment with the
tuning parameters $\alpha_t$ and $\delta$. Hand in your code as well as the best step-size sequence you found and again report the performance for one run.

4. Implement the SAG algorithm with a step-size of $1/L$, where $L$ is the maximum Lipschitz constant across the training examples ($L = \frac{1}{4} \max_i \{\|x^i\|^2\} + \lambda$). Hand in your code and again report the performance for one run.

3 Very-Short Answer Questions

Consider a function that is $C^1$ over $\mathbb{R}^d$ and five possible assumptions: (L) gradient is Lipschitz continuous, (B) function is bounded below, (C) function is convex, (C+) function is strictly-convex, and (SC) function is strongly-convex. Among the choices below, state which of the five assumptions on their own imply the following:

1. There exists an $f^*$ such that $f(w) \geq f^*$ for all $w$.
2. There exists a stationary point.
3. There exists at most one stationary point.
4. All stationary points are global optima.

Give a short and concise 1-sentence answer to the below questions.

5. The no free lunch theorem says that all possible machine learning models have equivalent performance across the set of possible learning problems. However, XGBoost wins a lot of Kaggle competitions while naive Bayes does not. Explain why or why not this empirical observation violates the no free lunch theorem.

6. Why is it useful to know whether a function satisfies the PL inequality when applying gradient descent?

7. Give an example (function, $w$ value, and subgradient) where the subgradient method will increase the objective for any step-size.

8. Why shouldn’t we use the L1-norm of the groups when do group L1-regularization?

9. What is the difference between inductive and transductive semi-supervised learning?

10. We said coordinate optimization makes sense for label propagation when you choose the coordinate to update uniformly at random. Describe a label propagation setting where choosing the coordinates non-uniformly would make the algorithm inefficient.

11. For finite-sum optimization problems, why are stochastic subgradient methods more appealing for non-smooth problems than for smooth problems? (Assuming that you only observe the function through “black box” calls to a function and subgradient oracle.)

12. Despite it’s empirical success in certain settings, what is the flaw in the logic behind with the “linear scaling rule” (“you should double the step-size when you double the batch-size”) in general?

13. What is the key advantage of SVRG over SAG?