CPSC 540 Assignment 2 (due January 31 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]

1. Upper bound on second-order Taylor expansion: $f(v) = f(w) + \nabla f(w)^T(v-w) + \frac{L}{2}\|v-w\|^2$.

   Answer:
   $$\nabla^2 f(v) = LI \succeq 0.$$

2. Probit regression: $f(w) = \sum_{i=1}^n -\log p(y_i|w, x_i)$ (where the probability is defined as in the last assignment).

   Answer: From Assignment 1 we have
   $$\nabla^2 f(w) = X^T D X,$$
   for a diagonal matrix D. If we assume that the diagonal elements D_{ii} of D are non-negative, then we have
   $$v^T X^T D X v = v^T X^T D^{\frac{1}{2}} D^{\frac{1}{2}} X v = (D^{\frac{1}{2}} X v)^T (D^{\frac{1}{2}} X v) = \|D^{\frac{1}{2}} X v\| \geq 0.$$
   It remains to show that we have $D_{ii} \geq 0$, where the D_{ii} is given by
   $$D_{ii} = \frac{z \cdot pdf(z)}{cdf(z)} + \frac{pdf(z)^2}{cdf(z)^2}.$$
   where $z = y^i w^T x^i$. By multiplying $D_{ii} \geq 0$ by $cdf(z)^2/pdf(z)$ (which is positive) we can equivalently show that
   $$z \cdot cdf(z) + pdf(z) \geq 0,$$
   which is equivalent to the inequality given in the hint.

[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.

3. Maximum function: $f(w) = \max_{j \in \{1,2,\ldots,d\}} w_j$.

   Answer:

$$\begin{aligned} f(\theta w + (1-\theta)v)) &= \max_j \{\theta w_j + (1-\theta v_j)\} \\ &\leq \max_j \{\theta w_j\} + \max_j \{(1-\theta)v_j\} \\ &= \theta \max_j \{w_j\} + (1-\theta) \max_j \{v_j\} \\ &= \theta f(w) + (1-\theta) f(v). \end{aligned}$$

Hint: Max are is not differentiable in general, so you cannot use the Hessian for the last one. For the second one, you can assume that $PDF(z) \geq -z \cdot \text{CDF}(z)$ (where PDF(z) is the PDF of a standard normal and CDF(z) is the CDF).

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

4. 1-class SVMs: $f(w, w_0) = \sum_{i=1}^{N} [\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2}\|w\|_2^2$, where $\lambda \geq 0$ and w_0 is a variable.

   Answer: This is the sum and max operations (which preserve convexity) applied to a a set of convex functions:

$$\underbrace{\sum_{i=1}^{N} \underbrace{[\max\{\underbrace{0}_{\text{const.}}, \underbrace{w_0 - w^T x_i}_{\text{linear}}\} - \underbrace{w_0}_{\text{linear}}]}_{\text{max}} + \underbrace{\frac{\lambda}{2}\|w\|_2^2}_{\nabla^2 = \lambda I \succeq 0}}_{\text{sum}}$$

5. Mixed-norm regularization: $f(w) = \|w\|_{p,q} = \left(\sum_{g \in \mathcal{G}} \|w_g\|_q^p\right)^{\frac{1}{p}}$ (for $p \geq 1$ and $q \geq 1$).

   Answer: This is a norm, and we showed that norms are convex. The first two properties of norms ($f(w) = 0 \rightarrow w = 0$ and $f(\alpha w) = |\alpha| f(w)$) follow in a straightforward way. To show that the triangle inequality holds let's define a vector v containining the q-norms of the group in w (so that $\|v\|_p = \|w\|_{p,q}$) and v' containing the q-norms of the groups in another arbitrary vector v'. Then by using the triangle inequality twice we have

$$\begin{aligned} \left(\sum_{g \in G} \|w_g + w'_g\|_q^p\right)^{\frac{1}{p}} &\leq \left(\sum_{g \in G} [\|w_g\|_q + \|w'_g\|_q]^p\right)^{\frac{1}{p}} \\ &= \left(\sum_{g \in G} [v_g + v'_g]^p\right)^{\frac{1}{p}} \\ &= \|v + v'\|_p \\ &\leq \|v\|_p + \|v'\|_p \\ &= \|w\|_{p,q} + \|w'\|_{p,q}. \end{aligned}$$

6. Minimum entropy over pairs of variables: $f(w) = \max_{\{i,j \mid i \neq j\}} \{w_i \log w_i + w_j \log w_j\}$ subject to $w_i > 0$ for all i.

Answer: The function $f(w) = w \log w$ has a second-derivative of $f''(w) = 1/w$ which is positive for $w > 0$ so each of the terms $w_i \log w_i$ is convex. From there, we're taking a sum of convex functions and a max over those convex functions.

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 ∇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

$$\|w^k - w^*\| = O(\rho^{k/2}).$$

Answer: From strong-convexity and $\nabla f(w^*) = 0$ we have

$$\begin{aligned} f(w^k) &\geq f(w^*) + \nabla f(w^*)^T(w^k - w^*) + \frac{\mu}{2}\|w^k - w^*\|^2 \\ &= f(w^*) + \frac{\mu}{2}\|w^k - w^*\|^2. \end{aligned}$$

Re-arranging we have

$$\frac{\mu}{2}\|w^k - w^*\|^2 \leq f(w^k) - f(w^*) = O(\rho^k).$$

Taking the square root gives the result.

2. A basic variation on the Armijo line-search is to set the step-size α_k to be $\alpha_k = (0.5)^p$, where p is the smallest constant such that

$$f(w^k - \alpha_k \nabla f(w^k)) \leq f(w^k) - \frac{\alpha_k}{2}\|\nabla f(w^k)\|^2.$$

Show that if ∇f is Lipschitz-continuous and f is bounded below, that setting α_k in this way means gradient descent requires at most $t = O(1/\epsilon)$ iterations to find at least one iteration k with $\|\nabla f(w^k)\|^2 \leq \epsilon$. Hint: you'll first want to figure out a progress bound of the form

$$f(w^{k+1}) \leq f(w^k) - \gamma\|\nabla f(w^k)\|^2,$$

for some constant γ that holds for all iterations. It may also help to recognize that if a function is L-Lipschitz continuous that it is also L'-Lipschitz continuous for any $L' \geq L$.

Answer: Consider first a step-size of $\bar{\alpha} = 1/\bar{L}$ for some $\bar{L} \geq L$. Since an L-Lipschitz function must also be $\bar{L}$-Lipschitz, the argument from class would give a progress bound of

$$\begin{aligned} f(w^{k+1}) &\leq f(w^k) - \frac{1}{2\bar{L}}\|\nabla f(w^k)\|^2 \\ &= f(w^k) - \frac{\bar{\alpha}}{2}\|\nabla f(w^k)\|^2, \end{aligned}$$

and thus if the step-size gets below $1/L$ it will be accepted. Since we're progressively halving α_k, this means the step-size must be at least $\alpha_{\min} = 1/2L$. Using this fact we have

$$\begin{aligned} f(w^{k+1}) &\leq f(w^k) - \frac{\alpha_k}{2}\|\nabla f(w^k)\|^2 \\ &\leq f(w^k) - \frac{\alpha_{\min}}{2}\|\nabla f(w^k)\|^2 \\ &= f(w^k) - \frac{1}{4L}\|\nabla f(w^k)\|^2 \end{aligned}$$

From here we can follow the proof from class, with the 2 replaced by a 4, to get that we need to have

$$t \geq \frac{4L[f(w^0) - f^*]}{\epsilon} = O(1/\epsilon).$$

(But in practice this strategy will be much faster, since we typically won't need a step-size as small $\alpha_{\min}$.)

3. Show that if ∇f is Lipschitz-continuous and f is convex (but not necessarily strongly-convex or PL), that if we run gradient descent for k iterations with $\alpha_k = 1/L$ we have

$$f(w^k) - f(w^*) = O(1/k).$$

Hint: you will have to use one the C^1 definition of convexity in our usual progress bound coming from Lipschitz-continuity of the gradient. You can try to turn this into a telescoping sum in terms of $\|w^k - w^*\|^2$ (which involves "completing the square"). Finally, note that our usual progress bound also gives that $f(w^{k+1}) \leq f(w^k)$ for all k.

Answer: From the C^1 convexity bound we have for any w^k and solution w^* that

$$f(w^*) \geq f(w^k) + \nabla f(w^k)^T(w^* - w^k).$$

Let's use this in our usual progress bound,

$$\begin{aligned} f(w^{k+1}) &\leq f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|^2 \\ &\leq f(w^*) + \nabla f(w^k)^T(w^k - w^*) - \frac{1}{2L}\|\nabla f(w^k)\|^2 \\ &= f(w^*) + \frac{L}{2}\|w^k - w^*\|^2 - \frac{L}{2}\|w^k - w^*\|^2 + \nabla f(w^k)^T(w^k - w^*) - \frac{1}{2L}\|\nabla f(w^k)\|^2 \\ &= f(w^*) + \frac{L}{2}(\|w^k - w^*\|^2 - \|w^k - w^*\|^2 + \frac{2}{L}\nabla f(w^k)^T(w^k - w^*) - \frac{1}{L^2}\|\nabla f(w^k)\|^2) \\ &= f(w^*) + \frac{L}{2}(\|w^k - w^*\|^2 - \|w^k - w^* - \frac{1}{L}\nabla f(w^k)\|^2) \\ &= f(w^*) + \frac{L}{2}(\|w^k - w^*\|^2 - \|w^{k+1} - w^*\|^2) \end{aligned}$$

Move $f(w^*)$ to the other side and sum up over all k to get

$$\sum_{k=1}^{t}[f(w^k) - f(w^*)] \leq \frac{L}{2}\sum_{k=1}^{t}(\|w^{k-1} - w^*\|^2 - \|w^k - w^*\|^2).$$

The right side telescopes and the left side is minimized at the last iteration, so we have

$$t[f(w^t) - f(w^*)] \leq \frac{L}{2}(\|w^0 - w^*\|^2 - \|w^t - w^*\|^2).$$

Removing the negative constant on the right side and dividing by t gives

$$f(w^t) - f(w^*) \leq \frac{L\|w^0 - w^*\|^2}{2t} = O(1/t).$$

1.3 Beyond Gradient Descent

1. We can write the proximal-gradient update as

$$w^{k+\frac{1}{2}} = w^k - \alpha_k \nabla f(w^k)$$
$$w^{k+1} = \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{2}\|v - w^{k+\frac{1}{2}}\|^2 + \alpha_k r(v) \right\}.$$

Show that this is equivalent to setting

$$w^{k+1} \in \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ f(w^k) + \nabla f(w^k)^T(v - w^k) + \frac{1}{2\alpha_k}\|v - w^k\|^2 + r(v) \right\}.$$

Answer:

$$\begin{aligned}
w^{k+1} &\in \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ f(w^k) + \nabla f(w^k)^T(v - w^k) + \frac{1}{2\alpha_k}\|v - w^k\|^2 + r(v) \right\} \\
&\equiv \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \alpha_k f(w^k) + \alpha_k \nabla f(w^k)^T(v - w^k) + \frac{1}{2}\|v - w^k\|^2 + \alpha_k r(v) \right\} \quad \text{(multiply by } \alpha_k\text{)} \\
&\equiv \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{\alpha_k^2}{2}\|\nabla f(w^k)\|^2 + \alpha_k \nabla f(w^k)^T(v - w^k) + \frac{1}{2}\|v - w^k\|^2 + \alpha_k r(v) \right\} \quad (\pm \text{ const.}) \\
&\equiv \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \|(v - w^k) + \alpha_k \nabla f(w^k)\|^2 + \alpha_k r(v) \right\} \quad \text{(complete the square)} \\
&\equiv \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \|v - \underbrace{(w^k - \alpha_k \nabla f(w^k))}_{\text{gradient descent}}\| + \alpha_k r(v) \right\},
\end{aligned}$$

2. In class we showed that if ∇f is coordinate-wise L-Lipschitz and satisfies PL then randomized coordinate optimization with a step-size of $1/L$ satisfies

$$\mathbb{E}[f(w^{k+1}) - f^*] \leq \left(1 - \frac{\mu}{Ld}\right)[f(w^k) - f^*].$$

Consider a C^2 function satisfying PL where coordinate i has its own Lipschitz constant L_j,

$$\nabla^2_{jj} f(w) \leq L_j.$$

Consider using a step-size of $1/L_{j_k}$ and sampling j_k proportional to the L_j, $p(j_k = j) = \frac{L_j}{\sum_{j'} L_{j'}}$. Show that this gives a faster convergence rate. Hint: the previous result corresponds to using $L = \max_j\{L_j\}$.

Answer: Plugging in the iteration into the coordinate-wise descent lemma with L_{j_k} and using $\alpha_k = L_{j_k}$ gives

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L_{j_k}}|\nabla_{j_k} f(w^k)|^2.$$

Taking the expectation we now have

$$\mathbb{E}[f(w^{k+1})] \leq f(w^k) - \frac{1}{2}\sum_j \frac{1}{L_j}\frac{L_j}{\sum_{j'} L_{j'}}|\nabla_j f(w^k)|^2$$
$$= f(w^k) - \frac{1}{2\sum_j L_j}\|\nabla f(w^k)\|^2.$$

Using strong-convexity we get

$$\mathbb{E}[f(w^{k+1}) - f^*] \leq \left(1 - \frac{\mu}{\sum_j L_j}\right)[f(w^k) - f^*].$$

We always have $\sum_j L_j \leq \sum_j \max_{j'} L_{j'} = \sum_j L = Ld$, so if some $L_j < L$ this inequality is strict and the result above is faster.

3. Consider an SVM problem where the x^i are *sparse*. In the bonus slides, we show how to efficiently apply stochastic subgradient methods in this setting by using the representation $w^k = \beta^k v^k$. Now consider a case where we know an L2 ball that contains the optimal solution. In other words, we know a τ such that $\|w^*\| \leq \tau$. If τ is small enough, we can use a *projected* stochastic subgradient method (projecting onto the L2-ball):

$$w^{k+\frac{1}{2}} = w^k - \alpha_k g^k - \alpha_k \lambda w^k$$
$$w^{k+1} = \begin{cases} w^{k+\frac{1}{2}} & \text{if } \|w^{k+\frac{1}{2}}\| \leq \tau \\ \frac{\tau}{\|w^{k+\frac{1}{2}}\|} w^{k+\frac{1}{2}} & \text{if } \|w^{k+\frac{1}{2}}\| > \tau \end{cases}$$

By constraining the w^k to a smaller set, this can sometimes dramatically improve the performance in the early iterations.[2] However, the projection operator is a full-vector operation so this would substantially slow down the runtime of the method. Derive a recursion that implements this algorithm without using full-vector operations. Hint: you may need to track more information than β^k and v^k.

Answer: We can do this by using the $w^k = \beta^k v^k$ representation and also tracking the squared norm of v^k by using $n^k = \|v^k\|^2$. Now let's divide the step up into three parts:

$$w^{k+1/3} = (1 - \lambda\alpha_k)w^k$$
$$w^{k+2/3} = w^{k+1/3} - \alpha_k g^k$$
$$w^{k+1} = \begin{cases} w^{k+2/3} & \text{if } \|w^{k+\frac{2}{3}}\| \leq \tau \\ \frac{\tau}{\|w^{k+\frac{2}{3}}\|} w^{k+\frac{2}{3}} & \text{if } \|w^{k+\frac{2}{3}}\| > \tau \end{cases}$$

The first step can just change β^k

$$\beta^{k+1/3} = (1 - \lambda\alpha_k)\beta^k$$
$$v^{k+1/3} = v^k$$
$$n^{k+1/3} = n^k$$

Using S as the non-zero coordinates of g^k, the second step updates v^k and β^k:

$$\beta^{k+2/3} = \beta^{k+1/3}$$
$$v^{k+2/3} = v^{k+1/3} - \frac{\alpha_k}{\beta^{k+1/3}} g^k \qquad \text{(sparse addition)}$$
$$n^{k+1/3} = n^{k+1/3} - \|v_S^{k+1/3}\|^2 + \|v_S^{k+2/3}\|^2$$

[2] We can obtain a value for τ for any L2-regularized problem where the loss is bounded below, by using that $f(w^*) + \frac{\lambda}{2}\|w^*\|^2 \leq f(0)$ and replacing $f(w^*)$ with its lower bound (which can be 0 for SVMs).

Finally, the projection only needs to update β^k:

$$\beta^{k+1} = \begin{cases} \beta^{k+2/3} & |\beta^{k+2/3}|\sqrt{n^{k+2/3}} \leq \tau \\ \text{sign}(\beta^k)\frac{\tau}{\sqrt{n^{k+2/3}}} & \text{otherwise} \end{cases}$$
$$v^{k+1} = v^{k+2/3}$$
$$n^{k+1} = n^{k+2/3}$$

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 classifiers 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.

   Answer: The minimal change to the existing code is to change the arguments to the softmax and and softmaxObj function to include lambda, and then change the return statement of the objective function to be:

   ```
   return (nll + (lambda)/2*norm(w)^2,reshape(G,d*k,1)+lambda*w)
   ```

   If $\lambda = 10$, then the error decreases from 0.298 down to 0.274. All 500 model parameters are non-zero, and all 100 original features are used.

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.

   Answer: You can call *findMinL1* to solve the L1-regularized version using:

   ```
   W[:] = findMinL1(funObj,W[:],lambda;maxIter=500)
   ```

   If $\lambda = 10$, then the error decreases to 0.08. With this choice, there are only 35 non-zero model parameters and only 19 of the original features are used (the exact values will again vary depending on the implementation and the solution is not unique).

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, *softmaxClassiferGL1*, 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.

Answer: I represented the groups by a vector, where element j of the vector gives the group of variable j. With this representation, calling the proximal-gradient looked like this:

```
groups = repmat(1:d,k)
W[:] = findMinGroupL1(funObj,W[:],lambda,groups;maxIter=500)
```

To make the proximal-gradient method, I started with *findMinL1* and replaced instances of the L1-norm with the groupNorm function below, and replaced the soft-threshold operator with the group soft threshold operator below:

```
function groupNorm(w,groups)
    d = length(w)
    nGroups = maximum(groups)
    v = zeros(nGroups)
    for j in 1:d
        v[groups[j]] += w[j]^2
    end
    return sum(sqrt.(v))
end

function groupSoftThreshold(w,threshold,groups)
    d = length(w)
    nGroups = maximum(groups)
    v = zeros(nGroups)
    for j in 1:d
        v[groups[j]] += w[j]^2
    end
    v = sqrt.(v)
    wNew = zeros(d,1)
    for g in 1:nGroups
        if v[g] != 0
            wNew[groups.==g] = (w[groups.==g]./v[g])*maximum([0 v[g]-threshold])
        end
    end
    return wNew
end
```

If $\lambda = 10$, then the error decreases to 0.0540. The number of non-zero parmaeters increases to 115, but these are only spread across 23 original features (the exact values will again vary depending on the implementation and the solution is not unique).

2.2 Coordinate Optimization

The function *example_CD.jl* loads a dataset and tries to fit an L2-regularized logistic regression model using coordinate optimization. Unfortunately, if we use L_f as the Lipschitz constant of ∇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 a $O(d\frac{L_f}{\mu}\log(1/\epsilon))$ iterations. This non-ideal runtime is also reflected in practice: the algorithm's iterations are relatively slow and even after 500 "passes" through the data it isn't particularly close to the optimal function value.

1. Modify this code so that the runtime of the algorithm is $O(nd\frac{L_c}{\mu}\log(1/\epsilon))$, where L_c is the Lipschitz constant of *all* partial derivatives $\nabla_i f$. You can do this by modifying the iterations so they have a cost $O(n)$ instead of $O(nd)$, and instead of using a step-size of $1/L_f$ they use a step-size of $1/L_c$ (which is given by $\frac{1}{4}\max_j\{\|x_j\|^2\} + \lambda$, where x_j is column j of the matrix X). Hand in your code and report the final function value and total time.

   Answer: The code should look roughly like this (note the different calculation of L, we update Xw

rather than re-compute it, and we only compute one element of the gradient).

```
# Compute Lipschitz constant of each partial derivative
L = .25*maximum(sum(X.^2,1)) + lambda

# Start running coordinate descent
w_old = copy(w);
Xw = zeros(n,1)
for k in 1:maxPasses*d

    # Choose variable to update 'j'
    j = rand(1:d)

    # Compute partial derivative 'g_j'
    sigmoid = 1./(1+exp.(-y.*Xw));
    g_j = -dot(X[:,j],y.*(1-sigmoid)) + lambda*w[j]

    # Update variable
    wjOld = w[j]
    w[j] -= (1/L)*g_j
    Xw = Xw + X[:,j]*(w[j]-wjOld)
```

The performance will vary based on the randomization/hardware, but on my laptop this decreased the time from around 57 seconds down to around 7 seconds. The algorithm still takes 500 iterations and usually finds a slightly lower function value (something like 147.3 or so).

2. To further improve the performance, make a new version of the code which samples the variable to update j_t proportional to the individual Lipschitz constants L_j of the coordinates, and uses a step-size of $1/L_{j_t}$. You can use the function *sampleDiscrete* (in *misc.jl*) to sample from a discrete distribution given the probability mass function. Hand in your code, and report the final function value as well as the number of passes.

   Answer: The code should look roughly like this (note that we now compute a vector of L values, use *sampleDiscrete* to pick j, and use $L[j]$ as the step-size).[3]

```
# Compute Lipschitz constant of each partial derivative
L = .25*sum(X.^2,1) + lambda
sampleProb = L/sum(L)

# Start running coordinate descent
w_old = copy(w);
Xw = zeros(n,1)
for k in 1:maxPasses*d

    # Choose variable to update 'j'
    j = sampleDiscrete(sampleProb)

    # Compute partial derivative 'g_j'
    sigmoid = 1./(1+exp.(-y.*Xw));
    g_j = -dot(X[:,j],y.*(1-sigmoid)) + lambda*w[j]

    # Update variable
    wjOld = w[j]
    w[j] -= (1/L[j])*g_j
    Xw = Xw + X[:,j]*(w[j]-wjOld)
```

This approach gives a lower function value, achieving something like 143.7 and tending to terminate early rather than running for 500 passes.

[3] Technically, using *sampleDiscrete* costs $O(d)$ so we've increased our iteration cost to $O(n+d)$, but for an initial $O(d)$ cost of computing the CDFs needed in sampleDiscrete we could implement the iterations in $O(n+\log(d))$ using a binary search.

3. Report the number of passes the algorithm takes as well as the final function value if you use *uniform sampling* but use a step-size of $1/L_{j_t}$.

   Answer: This strategy reduces the function value to 140.9 and the method terminates after around $150 - 200$ passes or so (which takes about 2 second on my laptop).

4. Suppose that when we use a step-size of $1/L_{j_t}$, we see that uniform sampling outperforms Lipschitz sampling. Why could this be consistent with the bounds we've shown?

   Answer: When we analyzed uniform sampling we used a step-size of $1/L_c$ and when we analyzed Lipschitz sampling we said the we use a step-size of $1/L_{j_t}$. In the above case we're using a step-size of $1/L_{j_t}$ with uniform sampling, which should give a tighter bound than using L_c since it makes more progress per iteration.

2.3 Stochastic Gradient

If you run the demo *example_SG.jl*, it will load a dataset and try to fit an L2-regularized logistic regression model using 10 "passes" of stochastic gradient using the step-size of $\alpha_t = 1/\lambda t$ that is suggested in many theory papers. Note that in other high-level languages (like R/Matlab/Python) this demo would run really slowly so you would need to write the inner loop in a low-level language like C, but in Julia you can directly write the stochastic gradient code and have it run fast.

Unfortunately, despite Julia making this code run fast compared to other high-level languages, the performance of this stochastic gradient method is atrocious. It often goes to areas of the parameter space where the objective function overflows and the final value is usually in the range of something like $6.5 - 7.5 \times 10^4$. This is quite far from the solution of 2.7068×10^4 and is even worse than just choosing $w = 0$ which gives 3.5×10^4. (This is unlike gradient descent and coordinate optimization, which never increase the objective function if your step-size is small enough.)

1. Although $\alpha_t = 1/\lambda t$ gives the best possible convergence rate in the worst case, in practice it's typically horrible (as we're not usually opitmizing the hardest possible λ-strongly convex function). Experiment with different choices of step-size sequence to see if you can get better 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.

   Answer: I found that you could get much better performance by just setting $\alpha_t = 10^{-4}$, and on one run this obtained 2.7117×10^4.

2. Besides tuning the step-size, another strategy that often improves the performance is using a (possibly-weighted) average of the iterations w^t. Explore whether this strategy can improve performance. Report the performance with an averaging strategy, and the objective function value obtained by this strategy for one run. (Note that the best step-size sequence with averaging might be different than without averaging.)

   Answer: By averaging all the iterations I found I could get better performance by setting $\alpha_t = 10^{-3}$, and on one run this obtained 2.7074×10^4.

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 ∇_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 α_t and δ. Hand in your code as well as the best step-size sequence you found and again report the performance for one run.

Answer: I found using $\delta = 1$ and $\alpha_t = 0.1$ worked ok, and on one run got 2.7133×10^4 (e.g., it didn't really outperform averaging or just using a constant step-size). The code could look like this:

```
w_old = copy(w);
D = ones(d,1)
for k in 1:maxPasses*n

    # Choose example to update 'i'
    i = rand(1:n)

    # Compute gradient for example 'i'
    r_i = -y[i]/(1+exp(y[i]*dot(w,X[i,:])))
    g_i = r_i*X[i,:] + (lambda_i)*w

    # Choose the step-size
    alpha = .1

    # Update the diagonal scaling
    D += g_i.^2

    # Take thes stochastic gradient step
    w -= alpha*g_i./sqrt.(D)
```

And taking square roots made it run surprisingly slow in Julia.

4. Impelement 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.

Answer: The code could look like this:

```
V = zeros(n,d)
g = zeros(d,1)
L = .25*maximum(sum(X.^2,2)) + lambda; # Wastes a pass, but could be done online
for k in 1:maxPasses*n

    # Choose example to update 'i'
    i = rand(1:n)

    # Compute gradient for example 'i'
    r_i = -y[i]/(1+exp(y[i]*dot(w,X[i,:])))
    g_i = r_i*X[i,:] + (lambda_i)*w

    # Choose the step-size
    alpha = 1/L

    # Update the direction
    g += g_i - V[i,:]
    V[i,:] = g_i

    # Take thes stochastic gradient step
    w -= (alpha/n)*g
```

I found that SAG tended to find the solution up to five significant digits, 2.7068×10^4, after 10 passes.

3 Very-Short Answer Questions

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

1. What is a situation where we can violate the golden rule on our test set, and still have it be a reasonable approximation of test error?

   Answer: You compute the error on the test set with a small number models (like 10), and have a huge number of examples in your test set (like 100000).

2. Do convex functions necessarily have a minima?

   Answer: No. (A counter-example would be $f(w) = w$, which is convex but has no minimum.)

3. What is the purpose of the descent lemma?

   Answer: It gives us a quadratic upper bound on the function f (based on a given $f(w)$, $\nabla f(w)$, and L value).

4. Why did we show that the gradient norm converges to 0 when we run gradient descent? (As opposed to showing that gradient descent finds the global optimum.)

   Answer: For non-convex functions, gradient descent doesn't guarantee convergence to the global optimum, so all we should expect to be able to show is how quickly it gets to a stationary point.

5. What is the relationship between the projected-gradient method and the proximal-gradient method?

   Answer: Projected-gradient is the special case of proximal-gradient where r is the indicator function for a convex set.

6. Why do we prefer the proximal-gradient method over the subgradient method?

   Answer: The proximal-gradient method is faster (it takes fewer iterations to converge, going from sublinear to linear in the strongly-convex case)

7. Why did we say in 340 that regularizing by the L2-norm doesn't give sparsity, and now we're saying we can use the L2-norm to give sparse solutions?

   Answer: In 340 we squared the L2-norm, which makes it smooth and destroys its sparsity-inducing property.

8. What is an example of a pattern that can be enforced with structured sparsity?

   Answer: If you have sets of variables A and B, you could enforce that we only allow the variables in A to be non-zero if the variables in B are non-zero.

9. Under what condition do we prefer to use coordinate optimization over gradient descent?

   Answer: We can implement the coordinate optimization steps to be d-times faster than the gradient steps.

10. For a machine learning objective that is the sum over n training examples, why should we never use the subgradient method?

    Answer: You can use the stochastic subgradient method, which has the same convergence rate with iterations that are n-times cheaper.

11. What is the advantage of SAG over stochastic gradient methods without a memory?

    Answer: It converges faster (for example, for strongly-convex functions the rate is linear instead of sublinear)