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): Dingqing Yang, Junjie Zhu
2. Student ID(s): 38800141, 30921167

1 Calculation Questions

1.1 Convexity

1.1 Convexity

1. $f(w) = \frac{1}{2}(Xw-y)^T V(Xw-y) + \frac{\lambda}{2}\|w\|^2$

$= \frac{1}{2}(w^T X^T V X w - y^T V X w - w^T X^T V y + y^T V y) + \frac{\lambda}{2} w^T I w$

$\therefore \nabla^2 f = X^T V X + \lambda I$

- for $w \in \mathbb{R}^d$ (pf), $w^T X^T V X w = \|V^{\frac{1}{2}} X w\|_2^2 \geq 0$, $w^T \lambda I^T w = \lambda \|w\|_2^2 > 0$

$\therefore w^T \cdot \nabla^2 f \cdot w > 0$ $\therefore \nabla^2 f = X^T V X + \lambda I$ is positive semidefinite $\therefore f$ is convex

2. $f(w) = -y^T X w + 1^T V$, $V_i = \exp(w^T x^i)$

$= -y^T X w + \sum_{i=1}^{n} \exp(w^T x^i)$

$\nabla(-y^T X w) = 0$, Let $g(w) = \sum_{i=1}^{n} \exp(w^T x^i)$

$\therefore \forall j, k \in \{1, 2 \dots d\}$,

$\frac{\partial g}{\partial w_j} = \sum_{i=1}^{n} \exp(w^T x^i) \cdot x_j^i$

$\therefore \frac{\partial g}{\partial w_k \partial w_j} = \sum_{i=1}^{n} \exp(w^T x^i) \cdot x_j^i \cdot x_k^i$. $\therefore \nabla^2 f = \nabla^2 g = X^T \begin{pmatrix} e^{w^T x^1} & 0 & \cdots & 0 \\ 0 & e^{w^T x^2} & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & e^{w^T x^n} \end{pmatrix} X$

- let V be the diagonal matrix of size $n \times n$, with $V_{ii} = e^{w^T x^i}$. Similar to question 1, since each $V_{ii} = e^{w^T x^i} > 0$ $\nabla^2 f = X^T V X$ is positive semi-definite $\therefore \nabla^2 f$ is convex

3. $f(w) = \max_{j \in \{1 \dots d\}} L_j |w_j|$

- for $v, w \in \mathbb{R}^d$, we examine $f((1-t)v + tw)$ for $t \in [0,1]$
- $v = (v_1, \dots v_d)$, $w = (w_1, \dots w_d)$
- $\forall j \in \{1 \dots d\}$,

$L_j |(1-t)v_j + t w_j|$

$\leq L_j [(1-t)|v_j| + t|w_j|]$ (triangle inequality + homogeneity of absolute value)

$= L_j (1-t)|v_j| + t \cdot L_j |w_j|$

$\therefore \max_{j \in \{1 \dots d\}} L_j |(1-t)v_j + t w_j| \leq \max_{j \in \{1 \dots d\}} L_j (1-t)|v_j| + t \cdot L_j |w_j|$

$\leq (1-t) \max_{j \in \{1 \dots d\}} L_j |v_j| + t \cdot \max_{k \in \{1 \dots d\}} L_k |w_k|$

$\therefore f((1-t)v + tw) \leq (1-t) f(v) + t f(w)$ $\forall t \in [0,1]$ $\therefore f$ is convex

4. $f(w) = \|Xw - y\|_p + \lambda \|Aw\|_q$.

- function $\|\cdot\|_p$ ~~are~~ (norm) are convex for $1 \le p \le \infty$. $Xw - y$, Aw are affine maps
- convex function combine with affine map is convex : $\|Xw - y\|_p$, $\|Aw\|_q$ are convex
- multiplication by positive constant ~~is~~ keeps convexity : $\lambda \|Aw\|_q$ is convex
- ~~a~~ sum of convex functions are convex $\therefore f(w) = \|Xw - y\|_p + \lambda \|Aw\|_q$ is convex

5. $f(w) = \sum_{i=1}^{n} \max\{0, |w^T x_i - y_i| - \varepsilon\} + \frac{\lambda}{2}\|w\|_2^2$,

- norm squared is convex, and multiplication of positive constant is convex $\therefore \frac{\lambda}{2}\|w\|_2^2$ is convex
- $w^T x_i - y_i$ is affine, and absolute value is convex. Composition of convex function with ~~linear~~ affine map is convex $\therefore |w^T x_i - y_i|$ is convex
- subtracting a constant (ε) remains convex, and 0 is convex, maximum of 2 functions is convex $\therefore \max\{0, |w^T x_i - y_i| - \varepsilon\}$ is convex
- sum of convex function is convex $\therefore f(w) = \sum_{i=1}^{n} \max\{0, |w^T x_i - y_i| - \varepsilon\} + \frac{\lambda}{2}\|w\|_2^2$ is convex

6. note that $C = \{w \in \mathbb{R}^d \mid Aw \le b\}$ is a convex set. We use the convention that $\infty \cdot 0 = 0$.

- we show that f is convex by cases. Let $v, w \in \mathbb{R}^d$

Case 1 : $v, w \in C$,

$\therefore \forall t \in [0,1]$, $(1-t)v + tw \in C$

$\therefore f((1-t)v + tw) = 0 \le (1-t)\cdot 0 + t \cdot 0 = (1-t) f(v) + t f(w)$

Case 2 : one of v, w is in C^c. Suppose, wlog, $v \in C$, $w \in C^c$.

- $\forall t \in (0,1)$ $f((1-t)v + tw) \le \infty = (1-t) f(v) + t f(w)$ $(\because f(w) = \infty,\ t > 0 \Rightarrow t \cdot f(w) = \infty)$.

Case 3 : $v, w \in C^c$

$\therefore \forall t \in (0,1)$, $f((1-t)v + tw) \le \infty = (1-t) f(v) + t f(w)$ $(f(v) = f(w) = \infty)$

$\therefore f$ is convex.

1.2 Convergence of Gradient Descent

1.2 Convergence of Gradient Descent

1. $\min_{k\in\{1,\dots,t\}} \|\nabla f(\omega^k)\|^2 \le \frac{2L(f(\omega^0)-f^*)}{t^{\frac{2}{3}}} \le \varepsilon$

$\therefore \frac{2L(f(\omega^0)-f^*)}{\varepsilon} \le t^{\frac{2}{3}}$ $\therefore \frac{(2L(f(\omega^0)-f^*))^{\frac{3}{2}}}{\varepsilon^{\frac{3}{2}}} \le t$, $\boxed{t = O\left(\frac{1}{\varepsilon^{\frac{3}{2}}}\right) \text{ iterations}}$

2. Note that by the algorithm, $L^0 \le L^1 \le \dots \le L^k$

- we claim that $L^k \le 2L$.
- suppose, for a contradiction, that $L^k > 2L$. Then $\exists$ a minimum $t \in \{1,\dots,k\}$, $L^t > 2L$
  $\therefore L^{t-1} = \frac{L^t}{2^i}$ for some $i \in \mathbb{N}$, and $L^{t-1} \le 2L$.

There are two cases:

Case 1: $i = 1$. Then $L^{t-1} = \frac{L^t}{2} > L$. But f is L-Lipschitz $\Rightarrow$ L^{t-1}-Lipschitz,
$\forall \omega \in \mathbb{R}^d$, $f\left(\omega - \frac{1}{L^{t-1}}\nabla f(\omega)\right) \le f(\omega) - \frac{1}{2L^{t-1}}\|\nabla f(\omega)\|^2$ (*)
$\therefore$ there is no need to double L^{t-1} for ω^t $\therefore L^t = L^{t-1}$, contradiction.

Case 2: $i \ge 2$.

- if $L \le L^{t-1} \le 2L$, then similar as above, there is no need to double L^{t-1} for ω^t $\therefore L^t = L^{t-1}$, contradiction,
- if $L^{t-1} < L$, then we may need to double L^{t-1} multiply times (to L^t) for the inequality to hold for ω^t. But $2^{i-1}L^{t-1} = \frac{L^t}{2} > L$, and f is $2^{i-1}L^{t-1}$-Lipschitz thus the algorithm will produce an $L^t \le 2^{i-1}L^{t-1}$, but $L^t = 2^i \cdot L^{t-1}$, contradiction.

- $\forall \omega \in \mathbb{R}^d$, $\exists \mu > 0$, $\frac{1}{2}\|\nabla f(\omega)\|^2 \ge \mu(f(\omega) - f^*(\omega^*))$

$\forall k \in \mathbb{N}$: $f(\omega^k) \le f(\omega^{k-1}) - \frac{1}{2L^{k-1}}\|\nabla f(\omega^{k-1})\|^2 \le f(\omega^{k-1}) - \frac{\mu}{L^{k-1}}(f(\omega^{k-1}) - f(\omega^*))$

$\therefore f(\omega^k) \le \left(1 - \frac{\mu}{L^{k-1}}\right) f(\omega^{k-1}) + \frac{\mu}{L^{k-1}} f(\omega^*) \Rightarrow f(\omega^k) - f(\omega^*) \le \left(1 - \frac{\mu}{L^{k-1}}\right)(f(\omega^{k-1}) - f(\omega^*))$

$\therefore f(\omega^k) - f(\omega^*) \le \left[\prod_{j=0}^{k-1}\left(1 - \frac{\mu}{L^j}\right)\right] \cdot (f(\omega^0) - f(\omega^*))$

$\because \forall j \in \{0,1,\dots,k-1\}$, $L^j \le L^k \le 2L$ $\therefore \frac{\mu}{L^j} \ge \frac{\mu}{2L}$, $\therefore 1 - \frac{\mu}{L^j} \le 1 - \frac{\mu}{2L}$.

$\therefore f(\omega^k) - f(\omega^*) \le \left[\prod_{j=0}^{k-1}\left(1 - \frac{\mu}{2L}\right)\right](f(\omega^0) - f(\omega^*))$

$= \left(1 - \frac{\mu}{2L}\right)^k (f(\omega^0) - f(\omega^*))$.

Hilroy

3. $L^k = \frac{1}{2}L^{k-1}$.

- Let $C^k = \{w \in \mathbb{R}^d \mid f(w) - \leq f(w^{k-1})\}$. Then if f restricted to the set C^k is Lipschitz continuous with a smaller L' (as compared to L where $f: \mathbb{R}^d \to \mathbb{R}$), then decreasing L^k will increase the step size.
- This is effective when the ~~stoc~~ gradient descend algorithm is approaching the global minimum of the function, and we want to increase the step size for the algorithm to converge to the ~~min~~global minimum faster.

4. $f(v) = f(w) + \nabla f(w)^T(v-w) + \frac{1}{2}(v-w)^T \nabla^2 f(u)(v-w)$ for some u between v and w

- by strong convexity, $d^T \nabla^2 f(u) d \geq 0 \mu \|d\|^2$

$\therefore f(v) \geq f(w) + \nabla f(w)^T(v-w) + \frac{\mu}{2}\|v-w\|^2$

- let $w = w^*$ $\therefore \nabla f(w^*) = 0$

$\therefore f(v) \geq f(w^*) + \frac{\mu}{2}\|v-w^*\|^2$ (*)

- in class we have shown that $f(w^k) - f(w^*) \overset{\leq}{*} (1-\frac{\mu}{L})^k (f(w^0) - f(w^*))$, $\rho = 1 - \frac{\mu}{L}$
- from (*), we have that $f(w^k) - f(w^*) \geq \frac{\mu}{2}\|w^k - w^*\|^2$

$\therefore \frac{\mu}{2}\|w^k - w^*\|^2 \leq (1-\frac{\mu}{L})^k (f(w^0) - f(w^*))$

$$\|w^k - w^*\| \leq \left(\frac{2}{\mu}\right)^{\frac{1}{2}} (1-\frac{\mu}{L})^{\frac{k}{2}} (f(w^0) - f(w^*))^{\frac{1}{2}}$$

- thus $\|w^k - w^*\|$ has a convergence rate of $O((1-\frac{\mu}{L})^{\frac{k}{2}}) = O(\rho^{\frac{k}{2}})$.

1.3 Beyond Gradient Descent

1.3 Beyond Gradient Descent

1. $\frac{1}{2}\|v - w^{k+1}\|^2 + \alpha_k r(v)$

$= \frac{1}{2}\|v - w^k + \alpha_k \nabla f(w^k)\|^2 + \alpha_k r(v)$

$= \frac{1}{2}(v^T - w^{kT} + \alpha_k \nabla f(w^k)^T)(v - w^k + \alpha_k \nabla f(w^k)) + \alpha_k r(v)$

$= \frac{1}{2}(v^T - w^{kT})(v - w^k) + \alpha_k \nabla f(w^k)^T (v - w^k) + \frac{1}{2}\alpha_k^2 \|\nabla f(w^k)\|^2 + \alpha_k r(v)$

$= \alpha_k \left(\frac{1}{2\alpha_k}\|v - w^k\|^2 + \nabla f(w^k)^T (v - w^k) + r(v)\right) + \frac{1}{2}\alpha_k^2 \|\nabla f(w^k)\|^2$

- since multiplying by a constant (α_k) and adding a constant ($\frac{1}{2}\alpha_k^2 \|\nabla f(w^k)\|^2$ does not depend on v) does not change the minimizer,

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

2. $f(W) = \sum_{i=1}^{n} \sum_{c \neq y^i} [1 - w_{y^i}^T x^i + w_c^T x^i]^+ + \frac{\lambda}{2}\|W\|_F^2$.

Notation: w_e^t: t^{th} weight for class e ($t \in \mathbb{N}$, $1 \leq t \leq d$).

- $1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$, $\delta_{c,e} = \begin{cases} 1 & \text{if } c = e \\ 0 & \text{if } c \neq e \end{cases}$
- $\theta_{i,c}^{e,t} \in [0,1]$ used to describe subgradient when normal gradient does not exist.

$$\therefore \frac{\partial f}{\partial w_e^t} = \sum_{i=1}^{n} \sum_{c \neq y^i} \Big[1_{\{z>0\}}(1 - w_{y^i}^T x^i + w_c^T x^i) \cdot (-x_t^i \cdot \delta_{e,y^i} + x_t^i \delta_{e,c}) + 1_{\{z=0\}}(1 - w_{y^i}^T x^i + w_c^T x^i) \cdot \theta_{i,c}^{e,t} \cdot (-x_t^i \cdot \delta_{e,y^i} + x_t^i \delta_{e,c}) \Big] + \lambda w_e^t.$$

3. $f(v) \le f(u) + \nabla f(u)^T (v-u) + \frac{L_\infty}{2}\|v-u\|_\infty^2$.

$\therefore f(w^{k+1}) \le f(w^k) - \nabla f(w^k)^T \cdot \frac{\|\nabla f(w^k)\|_1}{L_\infty} \cdot \text{sign}(\nabla f(w^k)) + \frac{L_\infty}{2}\left\|\frac{\|\nabla f(w^k)\|_1}{L_\infty} \cdot \text{sign}(\nabla f(w^k))\right\|_\infty^2$

$= f(w^k) - \frac{\|\nabla f(w^k)\|_1}{L_\infty} \nabla f(w^k)^T \cdot \text{sign}(\nabla f(w^k)) + \frac{\|\nabla f(w^k)\|^2}{2L_\infty} \underbrace{\|\text{sign}(\nabla f(w^k))\|_\infty^2}_{\le 1}$.

$\le f(w^k) - \frac{\|\nabla f(w^k)\|_1^2}{L_\infty} \|\nabla f(w^k)\|_1 + \frac{\|\nabla f(w^k)\|_1^2}{2L_\infty} \cdot 1$

$= f(w^k) - \frac{\|\nabla f(w^k)\|_1^2}{2L_\infty}$.

$\therefore \|\nabla f(w^k)\|_1^2 \le 2L_\infty (f(w^k) - f(w^{k+1}))$

$\therefore t \cdot \min_{k \in \{0,\dots,t-1\}} \|\nabla f(w^k)\|_1^2 \le 2L_\infty (f(w^0) - f(w^t))$

If $\|\nabla f(w^k)\|_1^2 \le \varepsilon$ to be guaranteed, then $\frac{2L_\infty (f(w^0) - f(w^t))}{t} \le \varepsilon$.

$\therefore \frac{2L_\infty (f(w^0) - f(w^t))}{\varepsilon} \le t$ $\quad \therefore \|\nabla f(w^k)\|_1^2 \le \varepsilon$ after $O(\frac{1}{\varepsilon})$ 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 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:

```
### Softmax L2 START ###
function softmaxObjL2(w,X,y,k, lambda)
    (n,d) = size(X)

    W = reshape(w,d,k)

    XW = X*W
    Z = sum(exp.(XW),dims=2)

    nll = 0
    G = zeros(d,k)
    for i in 1:n
        nll += -XW[i,y[i]] + log(Z[i])

        pVals = exp.(XW[i,:])./Z[i]
        for c in 1:k
            G[:,c] += X[i,:]*(pVals[c] - (y[i] == c))
        end
    end
    nll += 0.5*lambda*norm(W)^2
    G += lambda*W
    return (nll,reshape(G,d*k,1))
end
```

```
function logRegSoftmaxL2(X,y, lambda)
    (n,d) = size(X)
    k = maximum(y)

    # Each column of 'w' will be a logistic regression classifier
    W = zeros(d,k)

    funObj(w) = softmaxObjL2(w,X,y,k, lambda)

    W[:] = findMin(funObj,W[:],derivativeCheck=true,maxIter=50)

    # Make linear prediction function
    predict(Xhat) = mapslices(argmax,Xhat*W,dims=2)

    return LinearModel(predict,W)
end
### Softmax L2 END ###
```

- validation error is 0.274
- non-zero parameters: 500; original features: 100

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:

```
### Softmax L1 START ###
function logRegSoftmaxL1(X,y, lambda)
    (n,d) = size(X)
    k = maximum(y)

    # Each column of 'w' will be a logistic regression classifier
    W = zeros(d,k)

    funObj(w) = softmaxObj(w,X,y,k)

    W[:] = findMinL1(funObj,W[:], lambda)

    # Make linear prediction function
    predict(Xhat) = mapslices(argmax,Xhat*W,dims=2)

    return LinearModel(predict,W)
end
### Softmax L1 END ###
```

- validation error is 0.08
- non-zero parameters: 35; original features: 19

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:

```
function proxGradGroupL1(funObj,w,lambda, d, k;maxIter=100,epsilon=1e-2)
    # funObj: function that returns (objective,gradient)
    # w: initial guess
    # lambda: value of L1-regularization parmaeter
    # maxIter: maximum number of iterations
    # epsilon: stop if the gradient gets below this

    # Evalluate the intial objective and gradient
    (f,g) = funObj(w)

    # Initial step size and sufficient decrease parameter
    gamma = 1e-4
    alpha = 1
    for i in 1:maxIter

        # Gradient step on smoooth part
        wNew = w - alpha*g
        WNew = reshape(wNew, d, k)
        # Proximal step on each group
        for j = 1 : d
            wg = WNew[j, :]
            WNew[j, :] = wg/norm(wg) .* max.(norm(wg)-lambda*alpha,0)
        end
        # wNew = sign.(wNew).*max.(abs.(wNew) .- lambda*alpha,0)
        wNew = WNew[:]
        (fNew,gNew) = funObj(wNew)
```

```
176         # Decrease the step-size if we increased the function
177         gtd = dot(g,wNew-w)
178         while fNew + lambda*sum(sqrt.(sum(WNew.^2, dims=2))) >
179             f + lambda*sum(sqrt.(sum(reshape(w,d,k).^2, dims=2))) - gamma*alpha*gt
180             @printf("Backtracking\n")
181             alpha /= 2
182
183             # Try out the smaller step-size
184             wNew = w - alpha*g
185             # wNew = sign.(wNew).*max.(abs.(wNew) .- lambda*alpha,0)
186             WNew = reshape(wNew, d, k)
187             for j = 1 : d
188                 wg = WNew[j, :]
189                 WNew[j, :] = wg/norm(wg) .* max.(norm(wg)-lambda*alpha,0)
190             end
191             wNew = WNew[:]
192             (fNew,gNew) = funObj(wNew)
193         end
194
195         # Guess the step-size for the next iteration
196         y = gNew - g
197         alpha *= -dot(y,g)/dot(y,y)
198
199         # Sanity check on the step-size
200         if (!isfinitereal(alpha)) | (alpha < 1e-10) | (alpha > 1e10)
201             alpha = 1
202         end
```

```
        # Accept the new parameters/function/gradient
        w = wNew
        f = fNew
        g = gNew

        # Print out some diagnostics
        optCond = norm(w-sign.(w-g).*max.(abs.(w-g) .- lambda,0),Inf)
        @printf("%6d %15.5e %15.5e %15.5e\n",i,alpha,f+lambda*norm(w,1),optCond)

        # We want to stop if the gradient is really small
        if optCond < epsilon
            @printf("Problem solved up to optimality tolerance\n")
            return w
        end
    end
    @printf("Reached maximum number of iterations\n")
    return w
end
```

```
### Group L1 START ###
function logRegSoftmaxGL1(X,y, lambda)
    (n,d) = size(X)
    k = maximum(y)

    # Each column of 'w' will be a logistic regression classifier
    W = zeros(d,k)

    funObj(w) = softmaxObj(w,X,y,k)

    W[:] = proxGradGroupL1(funObj,W[:], lambda, d, k, maxIter=30)

    # Make linear prediction function
    predict(Xhat) = mapslices(argmax,Xhat*W,dims=2)

    return LinearModel(predict,W)
end
### Group L1 END ###
```

- validation error is 0.054
- non-zero parameters: 115; original features: 23

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 ∇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 $O(d\frac{L_f}{\mu} \log(1/\epsilon))$ iterations to reach an accuracy of ϵ. 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.

Answer: Note the code segment included in this question only contains the part after starting timer (i.e. after line 20 in *example_CD.jl*), and before the progress bound check (i.e. before line 39 in *example_CD.jl*).

Please go to code directory to view the full script if necessary.

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 increasing the step-size to $1/L_c$ (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 hte 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.

   Answer:

```
# 2.2.1 Compute Lc
Lc = maximum(sum(X.^2, dims=1)) + lambda

XTy = X'y
Xw = zeros(n, 1)

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

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

    # Compute partial derivative 'g_j'
    g_j = dot(X[:,j],Xw) -XTy[j]+ lambda*w[j]
    # g_j = g[j];

    # Update variable 2.2.1
    w[j] -= (1/Lc)*g_j
    global Xw -= (1/Lc)*g_j*X[:,j]
```

   - With function evaluation and printing progress diabled, runtime is around 0.15s
   - it takes roughly 200 passes (i.e. 20,000 iterations) to finish

2. While it doesn't improve the worst-case time complexity (without making stronger assumptions), you might expect to improve performance by using a more-clever choice of step-size. Modify the code to compute the optimal step-size (which you can do in closed-form without increasing the runtime), and report the effect of using the optimal step-size on the time and number of iterations.

   Answer:

```
XTy = X'y
Xw = zeros(n, 1)

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

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

    # Compute partial derivative 'g_j'
    g_j = dot(X[:,j],Xw) -XTy[j]+ lambda*w[j]

    # 2.2.2
    Lj = norm(X[:, j])^2 + lambda
    w[j] -= (1/Lj)*g_j
    global Xw -= (1/Lj)*g_j*X[:,j]
```

- With function evaluation and printing progress diabled, runtime is around 0.05s
- it takes roughly 20 passes (i.e. 20,00 iterations) to finish

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.)

Answer: Note the code segment included in this question only contains the part after original initialization (i.e. after line 16 in *example_SG.jl*), and before the progress bound check (i.e. before line 35 in *example_SG.jl*). Please go to code directory to view the full script if necessary.

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:

```
# Start running stochastic gradient
w_old = copy(w);
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/sqrt(k)
    alpha = 0.001

    # Take thes stochastic gradient step
    global w -= alpha*g_i
```

- We choose constant step size 0.001
- The resulting objective for one run is 2.7294e+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 (usually you can use bigger steps when you average).

   Answer:

```
# for running averages
tracked = 4
w_prev = zeros(d, tracked)

# Start running stochastic gradient
w_old = copy(w);
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 = 0.01

    # shift tracked weights
    for i in 1:(tracked-1)
        w_prev[:, i+1] = copy(w_prev[:,i])
    end
    w_prev[:,1] = copy(w)

    # Take thes stochastic gradient step
    global w -= alpha*g_i

    # weight averag ing
    w_total = sum(w_prev, dims=2) + w
    w = w_total / (tracked + 1)
```

- We choose constant step size 0.01
- The resulting objective for one run is around 2.7229e+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:

```
diag = zeros(d, 1) # a vector used to accumulate gradient

# Start running stochastic gradient
w_old = copy(w);
for k in 1:maxPasses*n

    delta = 1e-8

    # 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 = 0.01

    global diag += g_i.^2
    global Diag = Diagonal(sqrt.(diag .+ delta).^(-1))

    # Take thes stochastic gradient step
    global w -= alpha*Diag*g_i
```

- We choose constant step size 0.01
- The resulting objective for one run is 2.7181e+4

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:

```
L = 0.25*maximum(sum(X.^2, dims=1)) + lambda

v = zeros(n, 1) # O(n) to buffer r_i
g = zeros(d, 1)

# Start running stochastic gradient
w_old = copy(w);
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

    global g += g_i - (v[i]*X[i,:] + lambda_i*w)
    v[i] = r_i

    # Choose the step-size
    alpha = 1/L

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

- The resulting objective for one run is 2.7121e+4

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.

   Answer: B, SC

2. There exists a stationary point.

   Answer: SC

3. There exists at most one stationary point.

   Answer: C^+, SC

4. All stationary points are global optima.

   Answer: C, C^+, SC

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.

Answer: It does not violate the no free lunch theorem. The real world is structured, and some models extract patterns in real world data well.

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

   Answer: So the function is convex, and has exactly one global minimu,. The convergence rate of function's value to its minimum is also linear.

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

   Answer: let $f(w) = |w|$, $w = 0$, then the subgradient at that point is any real number between -1 and 1. It will increase the objective for any step size.

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

   Answer: It will be the same as doing regular L1 regularization.

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

   Answer: For inductive SSL, the goal is to predict well on new examples. On the other hand, transductive SSL only care about existing unlabeled examples rather than new samples.

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.

    Answer: DBScan with two centroids in the same cluster. If one centroid is picked extensively at first, then the second centroid will not be associated to points that are quite close to it.

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.)

    Answer: For non-smooth objectives, Deterministic methods are not faster than stochastic method, so use stochastic subgradient is n times faster than subgradient method.

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?

    Answer: It will not work if step size $> 2/L$, so you cannot keep increasing your step size with your batch size

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

    Answer: get rid of memory used to store the computed gradients by occasionally computing exact gradient