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]

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).

   Answer: My proof relies on the following:

   - If the Hessian of the objective is non-negative definite, then the original objective is convex (Where positive definite would imply strong convexity).
   - The all eigen values of a matrix lie within a circle of radius equal to a row of the matrix centered at the diagonal of that matrix (Gerschgorin circle theorem).
   - Commutativity of diagonal matrices.
   - Non-negative definiteness of the transpose of a matrix times the original, which can be shown trivially using the SVD

   Proof:

   - First we can note that for the for the objective function f(w) defined:

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

   We have that the gradient is given as,

   $$\nabla f(w) = X^\top VXw + \lambda w$$

   And finally, the Hessian is given as,

   $$\nabla^2 f(w) = X^\top VX + \lambda I$$

[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.

- By the cummutativity of diagnol matrices, we can write:

$$X^\top V X + \lambda I = V(X^\top X) + \lambda I$$

This implies that we have a diagonal matrix with positive entries scaling the rows of a non-negative definite matrix (as the transpose of a matrix times the original yields a non-negative definite matrix), i.e. $V(X^\top X)$ is Non-negative definite. Finally we can note that $V(X^\top X)$ is Non-negative definite, adding a strictly positive real diagonal matrix (λI), will only shift the matrices centers of the Gerschgorin circles along the positive real axis. And as the radius of these circles is not changing, necessarily all Eigen values must still lie above or on top of the positive real axis. Therefore all eigenvalues of the Hessian are greater then or equal to zero, making the matrix non-negative definite. Finally, this implies that the original function X is convex.

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

   Answer: Similar to the proof above, we can look at the hessian of our objective function, starting with the original objective
$$f(w) = -y^\top Xw + 1^\top v$$

   We get the gradient as,
$$\nabla f(w) = \sum_{i=1}^{n}\sum_{j=1}^{m} -y_i X_{i,j} w_j + \sum_{i=1}^{n} x^i v_i$$

   Which gives a convenient form for the hessian:
$$\nabla^2 f(w) = XX^\top \mathrm{diag}(v)$$

   Similar to the argument above, we know that all elements of $\mathrm{diag}(v)$ are strictly positive, and because $XX^\top$ is hermitian, its eigen values are non negative, thus scaling will again have no affect on the sign of them. Therefore we again have that the hessian is non-negative definite and so necessarily the original objective is convex.

3. Weighted infinity-norm: $f(w) = \max_{j\in\{1,2,\ldots,d\}} L_j|w_j|$.
   Hint: Max and absolute value are not differentiable in general, so you cannot use the Hessian for this question.

   Answer: Assuming that L contains strictly positive values (as it must for the function to define a norm), we can just directly apply the definition of convexity (that is $f(\lambda w_1 + (1-\lambda)w_2) \le \lambda f(w_1) + (1-\lambda) f(w_2)$):

$$\begin{aligned}
f(\lambda w_1 + (1-\lambda)w_2) &= \|L(\lambda w_1 + (1-\lambda)w_2)\|_\infty && (1)\\
&= \|L(\lambda w_1) + L(1-\lambda)w_2\|_\infty && (2)\\
&\le \|L(\lambda w_1)\|_\infty + \|L(1-\lambda)w_2\|_\infty && (3)\\
&= \max_{j\in\{1,2,\ldots,d\}} L_j|\lambda w_j^1| + \max_{j\in\{1,2,\ldots,d\}} L_j|(1-\lambda)w_j^2| && (4)\\
&= \lambda \max_{j\in\{1,2,\ldots,d\}} L_j|w_j^1| + (1-\lambda) \max_{j\in\{1,2,\ldots,d\}} L_j|w_j^2| && (5)\\
&= \lambda f(w_1) + (1-\lambda) f(w_2) && (6)
\end{aligned}$$

   Where we use the fact that because the infinity norm is in fact a norm, it satisfies the triangle inequality.

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

   Answer: We can show that this first objective by just following the definition of convexity,

$$\begin{aligned} f(\lambda w_1 + (1-\lambda)w_2) &= \|X(\lambda w_1 + (1-\lambda)w_2) - y\|_p + \alpha\|A(\lambda w_1 + (1-\lambda)w_2)\|_q \\ &= \|X(\lambda w_1 + (1-\lambda)w_2) - (\lambda y + (1-\lambda)y)\|_p + \alpha\|A(\lambda w_1 + (1-\lambda)w_2)\|_q \\ &= \|\lambda(Xw_1 - y) + (1-\lambda)(Xw_2 - y)\|_p + \alpha\|\lambda Aw_1 + (1-\lambda)w_2)\|_q \\ &\leq \|\lambda(Xw_1 - y)\|_p + \|(1-\lambda)(Xw_2 - y)\|_p + \alpha\|\lambda Aw_1\|_q + \alpha\|(1-\lambda)Aw_2)\|_q \\ &= \lambda\|(Xw_1 - y)\|_p + (1-\lambda)\|(Xw_2 - y)\|_p + \alpha\lambda\|Aw_1\|_q + \alpha(1-\lambda)\|Aw_2)\|_q \\ &= \lambda f(w_1) + (1-\lambda)f(w_2) \end{aligned}$$

   Which give the definition of convexity.

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

   Answer: Again, following the definition of convexity,

$$\begin{aligned} f(\lambda w_1 + (1-\lambda)w_2) &= \sum_{i=1}^N \max\{0, |(\lambda w_1 + (1-\lambda)w_2)^\top x_i - y_i| - \epsilon\} + \frac{\alpha}{2}\|(\lambda w_1 + (1-\lambda)w_2)\|_2^2 \\ &= \sum_{i=1}^N \max\{0, |(\lambda w_1 + (1-\lambda)w_2)^\top x_i - \lambda y_i(1-\lambda)y_i| - \lambda\epsilon + (1-\lambda)\epsilon\} \\ &+ \frac{\alpha}{2}\|(\lambda w_1 + (1-\lambda)w_2)\|_2^2 \\ &= \sum_{i=1}^N \max\{0, |(\lambda(w_1^\top x_i - y_i) + (1-\lambda)(w_2^\top x_i - y_i)| - \lambda\epsilon + (1-\lambda)\epsilon\} \\ &+ \frac{\alpha}{2}\|(\lambda w_1 + (1-\lambda)w_2)\|_2^2 \\ &\leq \sum_{i=1}^N \max\{0, |(\lambda(w_1^\top x_i - y_i)| + |(1-\lambda)(w_2^\top x_i - y_i)| - \lambda\epsilon + (1-\lambda)\epsilon\} \\ &+ \frac{\alpha}{2}\|\lambda w_1\|_2^2 + \frac{\alpha}{2}\|(1-\lambda)w_2\|_2^2 \\ &\leq \sum_{i=1}^N \max\{0, \lambda(|(w_1^\top x_i - y_i)| - \epsilon)\} + \max\{0, (1-\lambda)(|w_2^\top x_i - y_i| - \epsilon)\} \\ &+ \frac{\alpha}{2}\|\lambda w_1\|_2^2 + \frac{\alpha}{2}\|(1-\lambda)w_2\|_2^2 \\ &= \lambda\sum_{i=1}^N \max\{0, (|(w_1^\top x_i - y_i)| - \epsilon)\} + (1-\lambda)\sum_{i=1}^N \max\{0, (|w_2^\top x_i - y_i| - \epsilon)\} \\ &+ \lambda\frac{\alpha}{2}\|w_1\|_2^2 + (1-\lambda)\frac{\alpha}{2}\|w_2\|_2^2 \\ &= \lambda f(w_1) + (1-\lambda)f(w_2) \end{aligned}$$

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

   Answer: We can show this by considering a couple different cases of weights:

- Case 1: w_1, w_2 that satisfy the inequality within the problem above, that is

$$\begin{aligned} Aw_1 - b &\leq 0 \\ \lambda(Aw_1 - b) &\leq 0 \end{aligned}$$

$$\begin{aligned} Aw_2 - b &\leq 0 \\ (1-\lambda)(Aw_2 - b) &\leq 0 \end{aligned}$$

Then we can combine these inequalities, and still have them be satified such that,

$$(1-\lambda)(Aw_2 - b) + \lambda(Aw_1 - b) \leq 0$$

With a little shuffling,

$$\begin{aligned} A(\lambda w_1 + (1-\lambda)w_2) - (1 - \lambda + \lambda)(b) &\leq 0 \\ A(\lambda w_1 + (1-\lambda)w_2) - b &\leq 0 \end{aligned}$$

This means that $f(\lambda w_1 + (1-\lambda)w_2) = 0$, and additionally, $f(w_1)$ and $f(w_2) = 0$, which implies that:

$$f(\lambda w_1 + (1-\lambda)w_2) \leq \lambda f(w_1) + (1-\lambda)f(w_2)$$

- Case 2: Next we consider the case (without loss of generality) of w_1 not satisfying the inequality. In this case note that $f(w_1) = \infty$ so $\lambda f(w_1) + (1-\lambda)f(w_2) = \infty$, and because the maximum value that f can take on for any w is ∞, we necessarily have:

$$f(\lambda w_1 + (1-\lambda)w_2) \leq \lambda f(w_1) + (1-\lambda)f(w_2)$$

- Case 3: Lastly we can consider the case where neither w_1 or w_2 satisfies the inequality, we get that both $f(w_1) = \infty$ and $f(w_2) = \infty$ so necessarily $\lambda f(w_1) + (1-\lambda)f(w_2) = \infty$ and again we recover the definition of convexity for our function,

$$f(\lambda w_1 + (1-\lambda)w_2) \leq \lambda f(w_1) + (1-\lambda)f(w_2)$$

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 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 $\|\nabla f(w^k)\|^2 \leq 2L(f(w^0) - f^*)/t^{4/3}$. How many iterations of this algorithm would we need to find a w^k with $\|\nabla f(w^k)\|^2 \leq \epsilon$?

   Answer: In order to calculate the required number of iterations, all for a given epsilon satisfying $\|\nabla f(w^k)\|^2 \leq \epsilon$, all we need to do is rearrange terms a bit:

$$\|\nabla f(w^k)\|^2 \leq \|\nabla f(w^k)\|^2 \leq 2L(f(w^0) - f^*)/t^{4/3} = \epsilon$$

Now we solve for t,

$$t = (\frac{2L(f(w^0) - f(w^*))}{\epsilon})^{3/4}$$
$$= O\left(\frac{1}{\epsilon^{3/4}}\right)$$

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 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

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

if ∇f is L-Lipschitz continuousn and f is μ-strongly convex.
Hint: if a function is L-Lipschitz continuous that it is also L'-Lipschitz continuous for any $L' \geq L$.

Answer: Starting with a definition for L^k,

$$f\left(w^k - \frac{1}{L^k}\nabla f(w^k)\right) \leq f(w^k) - \frac{1}{2L^k}\|\nabla f(w^k)\|$$
$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L^k}\|\nabla f(w^k)\|$$

We know that L^k must be contained in [L, 2L], there for let $L^k = 2L$. This gives,

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{4L}\|\nabla f(w^k)\|$$

Additionally by mu strong convexity,

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

returning to the inequality from before, we can rearrange,

$$f(w^1) \leq f(w^0) - \frac{1}{4L}\|\nabla f(w^0)\|^2$$
$$f(w^1) - f(w^0) \leq -\frac{1}{4L}\|\nabla f(w^0)\|^2$$
$$4L(f(w^0) - f(w^1)) \geq \|\nabla f(w^0)\|^2$$

Now we can combine this inequality with the one above,

$$2\mu(f(w^0) - f(w^*)) \geq \|\nabla f(w^0)\|^2 \geq 4L(f(w^0) - f(w^*))$$

Now we can ignore the middle term,

$$2\mu(f(w^0) - f(w^*)) \geq 4L(f(w^0) - f(w^*))$$
$$\frac{\mu}{2L}(f(w^0) - f(w^*)) \geq (f(w^0) - f(w^*))$$
$$-\frac{\mu}{2L}(f(w^0) - f(w^*)) \geq (f(w^*) - f(w^0))$$

Next we can add $f(w^0) - f(w*)$ to each side,

$$(1 - \frac{\mu}{2L})(f(w^0) - f(w^*)) \geq (f(w^1) - f(w^*))$$

because we chose 0,1 arbitrarily, this is in fact true for all k, and we recover:

$$(1 - \frac{\mu}{2L})(f(w^k) - f(w^*)) \geq (f(w^{k+1}) - f(w^*))$$

This then allows us to just iteratively bound values for steadily smaller

$$(1 - \frac{\mu}{2L})(f(w^{k-1}) - f(w^*)) \geq (f(w^k) - f(w^*))$$
$$(1 - \frac{\mu}{2L})^2(f(w^{k-2}) - f(w^*)) \geq (f(w^k) - f(w^*))$$
$$(1 - \frac{\mu}{2L})^3(f(w^{k-3}) - f(w^*)) \geq (f(w^k) - f(w^*))$$
$$......$$
$$(1 - \frac{\mu}{2L})^k(f(w^0) - f(w^*)) \geq (f(w^k) - f(w^*))$$

Which gives the desired result.

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.

   Answer: Doing this would allow the L^k to adapt to areas to areas where locally the lipshitz inequality holds for smaller values of L. This is especially true for loss surfaces that go through areas of high variation, and then flatten out. For example, functions that slowly become more linear time. In this case, larger L^k would mean that more of the space could be traversed more quickly and the algorithm would converge faster.

4. 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: Starting out with the decent lemma derived from the Taylor series approximation centered about the optimal w w^*, we have:

$$f(w^k) \geq f(w^*) + \nabla f(w^*)^T(w^k - w^*) + \frac{1}{2}(w^k - w^*)^T\nabla^2 f(w^*)(w^k - w^*)$$
$$f(w^k) - f(w^*) \geq \nabla f(w^*)^T(w^k - w^*) + \frac{1}{2}(w^k - w^*)^T\nabla^2 f(w^*)(w^k - w^*)$$
$$f(w^k) - f(w^*) \geq \frac{1}{2}(w^k - w^*)^T\nabla^2 f(w^*)(w^k - w^*)$$

Next using the fact that our function is μ strongly convex, we can replace the inner product over the hessian with $\mu/2$,

$$
\begin{aligned}
f(w^k) - f(w^*) &\geq \frac{1}{2}(w^k - w^*)^T \nabla^2 f(w^*)(w^k - w^*) \\
f(w^k) - f(w^*) &\geq \frac{\mu}{2}\|w^k - w^*\|^2
\end{aligned}
$$

Lastly, we can replace the left hand side with the assumption from the prompt, and with a little manipulation:

$$
\begin{aligned}
f(w^k) - f(w^*) &\geq \frac{\mu}{2}\|w^k - w^*\|^2 \\
O(\rho^k) &\geq \frac{\mu}{2}\|w^k - w^*\|^2 \\
O(\rho^{k/2}) &\geq \frac{\mu}{2}\|w^k - w^*\|
\end{aligned}
$$

Which gives the desired result.

1.3 Beyond Gradient Descent

1. We can write the proximal-gradient update as

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

Show that this is equivalent to setting

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

Answer: In order to solve this problem, we need to just replace the definitions into the expression above,

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

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

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

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

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

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

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

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

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

Where we remove and add constants, as they do not affect the minimum achieved. Additionally, we divided out the α term and then ignored it as it also has no effect on the argmin.

2. The "sum" version of multi-class SVMs uses an objective of the form

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

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 objetive.

Answer: w

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 ∇f is Lipschitz continuous in the ∞-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_∞. 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.

Answer: Starting from the decent lemma:

$$f(w^{k+1}) \leq f(w^k) + \nabla f(w^k)^T(w^{k+1} - w^k) + \frac{L_\infty}{2}\|w^{k+1} - w^k\|_\infty^2$$

$$f(w^{k+1}) - f(w^k) \leq \nabla f(w^k)^T(w^{k+1} - w^k) + \frac{L_\infty}{2}\|w^{k+1} - w^k\|_\infty^2$$

$$f(w^{k+1}) - f(w^k) \leq -\nabla f(w^k)^\top \frac{\|\nabla f(w^k)\|_1}{L_\infty}\text{sign}(\nabla f(w^k)) + \frac{L_\infty}{2}\|\frac{\|\nabla f(w^k)\|_1}{L_\infty}\text{sign}(\nabla f(w^k))\|_\infty^2$$

$$f(w^{k+1}) - f(w^k) \leq -\frac{\|\nabla f(w^k)\|_1^2}{L_\infty} + \frac{L_\infty}{2}\|\frac{\|\nabla f(w^k)\|_1}{L_\infty}\text{sign}(\nabla f(w^k))\|_\infty^2$$

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

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

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

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

Next we can apply the telescoping series trick to get the inequality into a form closer to the *epsilon* form from earlier:

$$\sum_{k=0}^{t-1} f(k+1) - f(k) \leq \frac{-1}{2L_\infty}\sum_{k=0}^{t-1}\|\nabla f(w^k)\|_1^2$$

$$f(t) - f(0) \leq \frac{-1}{2L_\infty}\sum_{k=0}^{t-1}\|\nabla f(w^k)\|_2^2$$

$$f(t) - f(0) \leq \frac{-t}{2L_\infty}\|\nabla f(w^t)\|_2^2$$

$$f(0) - f(t) \geq \frac{t}{2L_\infty}\|\nabla f(w^t)\|_2^2$$

$$f(0) - f(t^*) \geq \frac{t}{2L_\infty}\|\nabla f(w^t)\|_2^2$$

$$\frac{2L_\infty}{t}(f(0) - f(t^*)) \geq \|\nabla f(w^t)\|_2^2$$

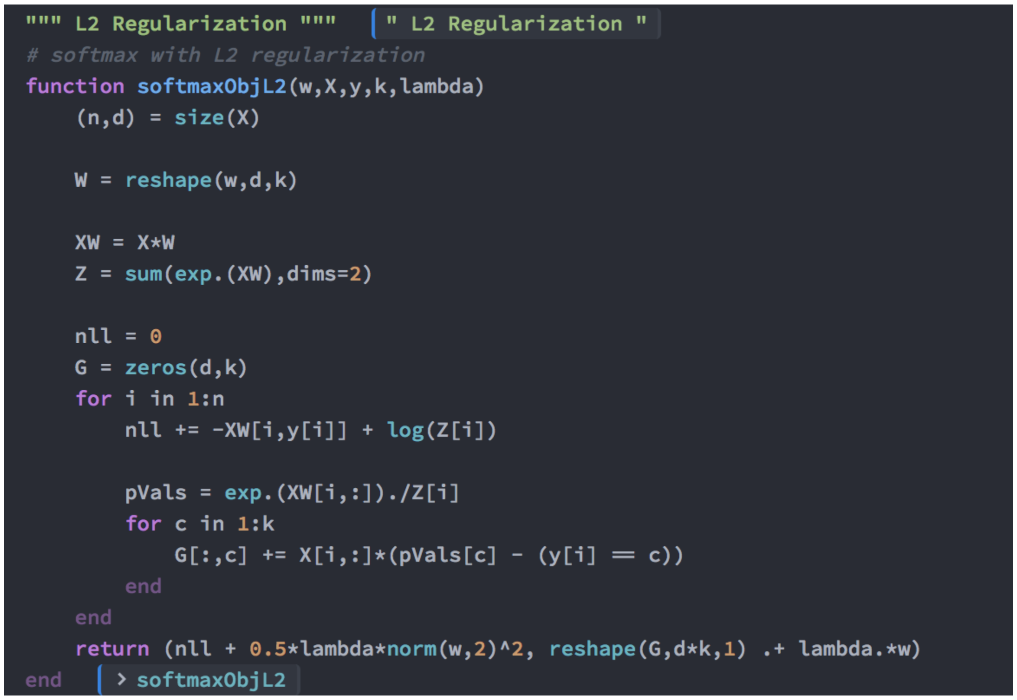
Now we just set this value equal to the required ϵ and solve as before:

$$\frac{2L_\infty}{t}(f(0) - f(t^*)) = \epsilon$$

$$\frac{2L_\infty}{\epsilon}(f(0) - f(t^*)) = t$$

$$O(\frac{1}{\epsilon}) = t$$

Which gives the desired result.



```julia
""" L2 Regularization """    " L2 Regularization "
# softmax with L2 regularization
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
    return (nll + 0.5*lambda*norm(w,2)^2, reshape(G,d*k,1) .+ lambda.*w)
end    > softmaxObjL2
```

Figure 1:

2 Computation Questions

2.1 Proximal-Gradient

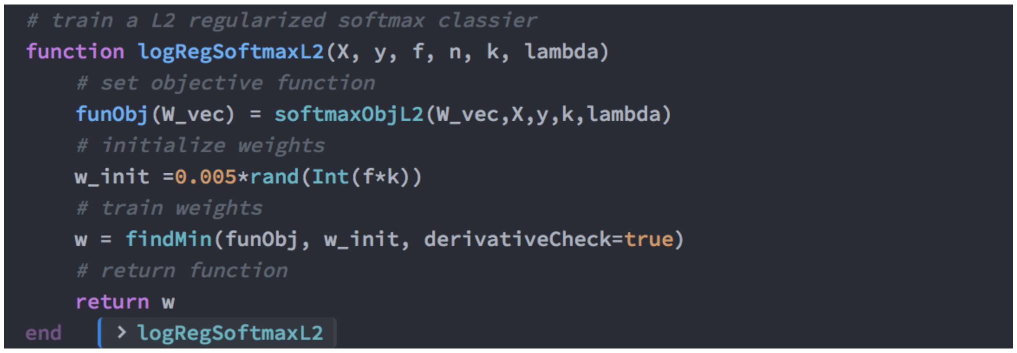
If you run the demo *example_group.jl*, it will load a dataset and fit a multi-class logistic regression (softmax) classifier. This dataset is actually *linearly-separable*, so there exists a set of weights W that can perfectly classify the training data (though it may be difficult to find a W that perfectly 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 number of parameters used were 500, and the number of features was 100. Too see the code used check out Figures 1 and 2.

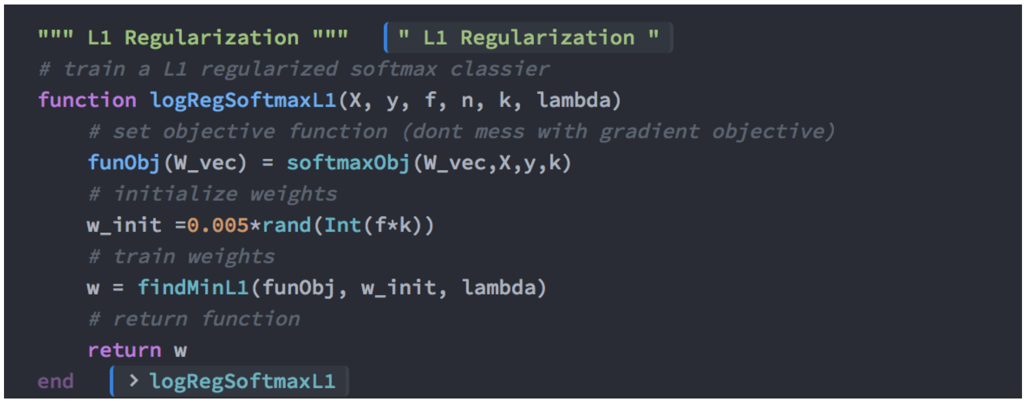
2. While L2-regularization reduces overfitting a bit, it still uses all the variables even though 90% of them are irrelevant. In situations like this, L1-regularization may be more suitable. Write a new function, *logRegSoftmaxL1*, that fits a multi-class logistic regression model with L1-regularization. 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: The number of parameters used were 35, and the number of features was 19. Too see the code used check out Figure 3.



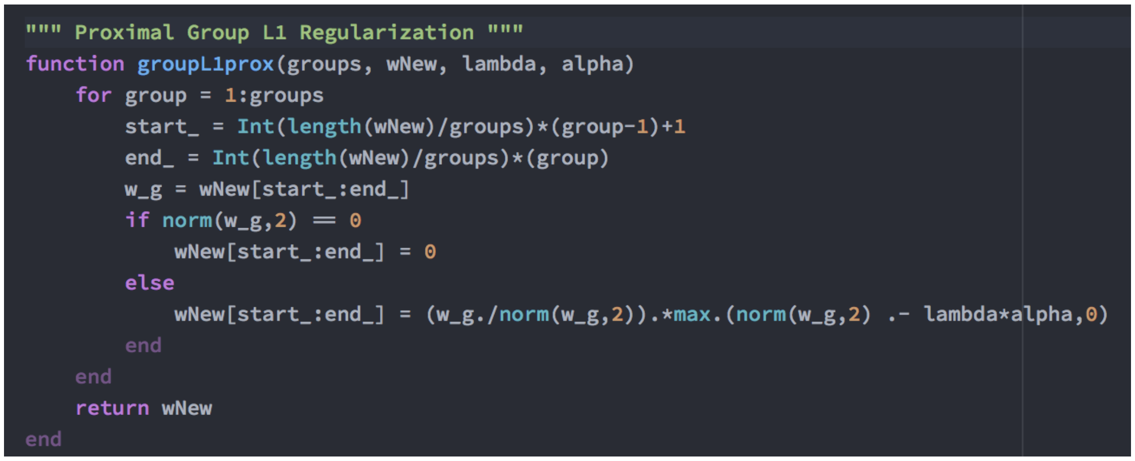
```
# train a L2 regularized softmax classier
function logRegSoftmaxL2(X, y, f, n, k, lambda)
    # set objective function
    funObj(W_vec) = softmaxObjL2(W_vec,X,y,k,lambda)
    # initialize weights
    w_init =0.005*rand(Int(f*k))
    # train weights
    w = findMin(funObj, w_init, derivativeCheck=true)
    # return function
    return w
end   > logRegSoftmaxL2
```

Figure 2:



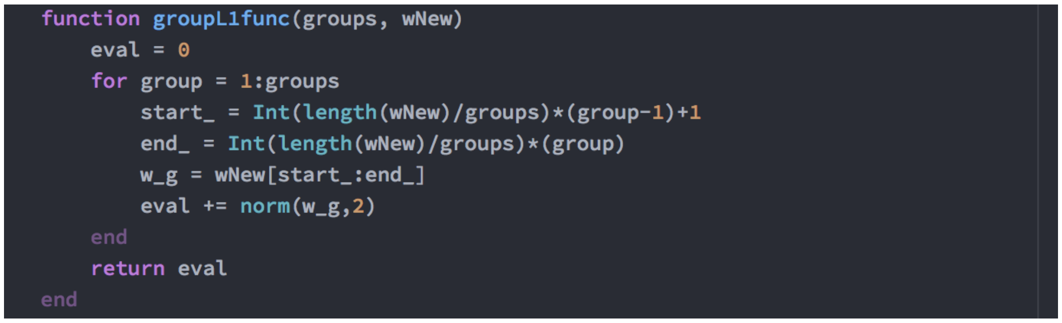
```
""" L1 Regularization """    " L1 Regularization "
# train a L1 regularized softmax classier
function logRegSoftmaxL1(X, y, f, n, k, lambda)
    # set objective function (dont mess with gradient objective)
    funObj(W_vec) = softmaxObj(W_vec,X,y,k)
    # initialize weights
    w_init =0.005*rand(Int(f*k))
    # train weights
    w = findMinL1(funObj, w_init, lambda)
    # return function
    return w
end   > logRegSoftmaxL1
```

Figure 3:



```julia
""" Proximal Group L1 Regularization """
function groupL1prox(groups, wNew, lambda, alpha)
    for group = 1:groups
        start_ = Int(length(wNew)/groups)*(group-1)+1
        end_ = Int(length(wNew)/groups)*(group)
        w_g = wNew[start_:end_]
        if norm(w_g,2) == 0
            wNew[start_:end_] = 0
        else
            wNew[start_:end_] = (w_g./norm(w_g,2)).*max.(norm(w_g,2) .- lambda*alpha,0)
        end
    end
    return wNew
end
```

Figure 4:



```julia
function groupL1func(groups, wNew)
    eval = 0
    for group = 1:groups
        start_ = Int(length(wNew)/groups)*(group-1)+1
        end_ = Int(length(wNew)/groups)*(group)
        w_g = wNew[start_:end_]
        eval += norm(w_g,2)
    end
    return eval
end
```

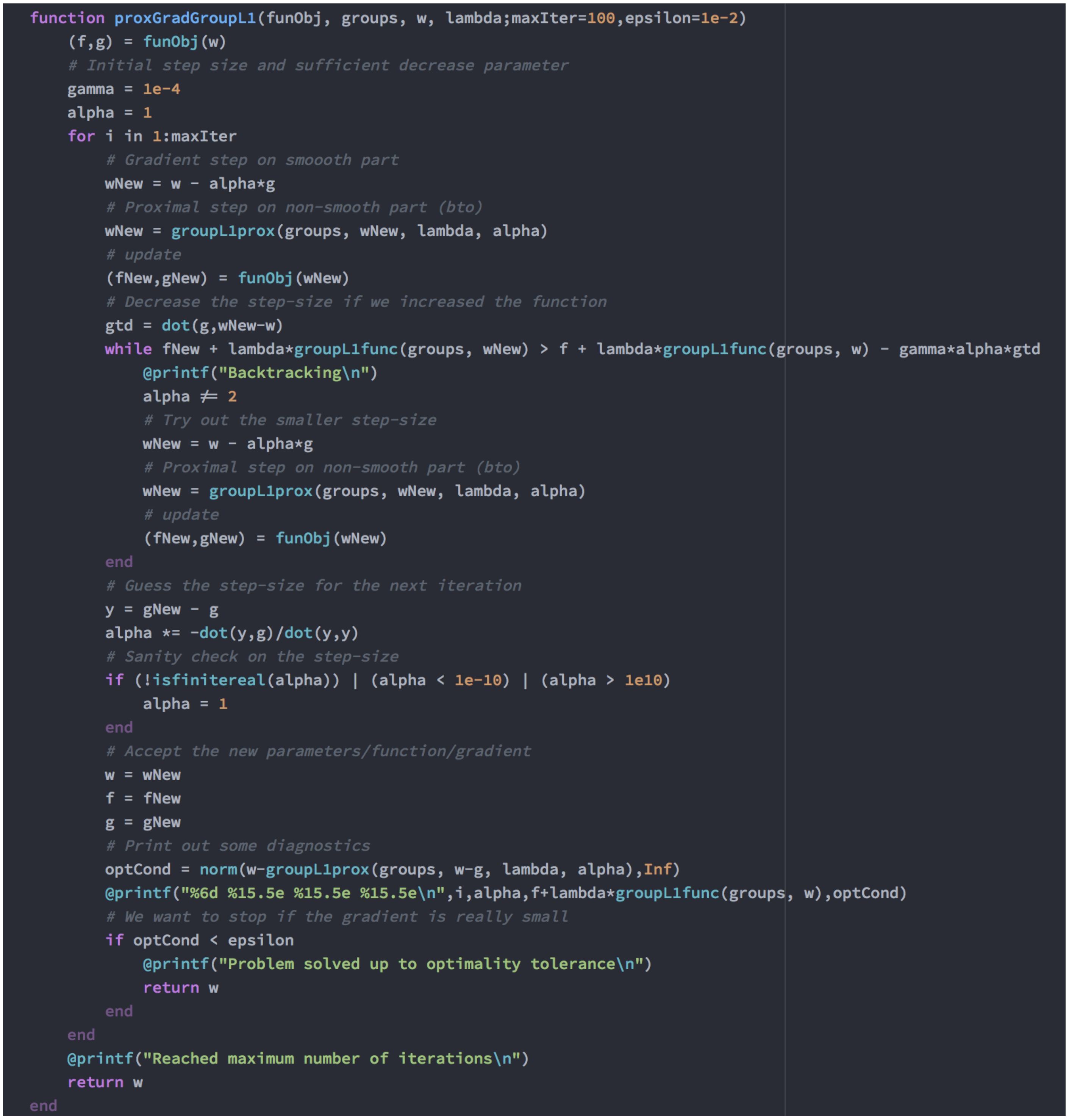
Figure 5:

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.
   Hint: *findMinL1* implements a generic proximal-gradient method for L1-regularization, which you could modify to give a generic proximal-gradient method for group L1-regularization.

   Answer: The number of parameters used were 195, and the number of features was 85. Too see the code used check out Figures 4 5 and 6.

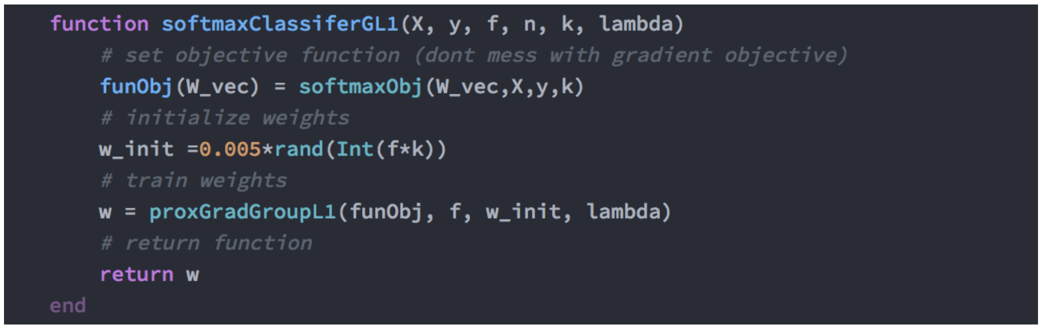
2.2 Coordinate Optimization

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



```julia
function proxGradGroupL1(funObj, groups, w, lambda;maxIter=100,epsilon=1e-2)
    (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
        # Proximal step on non-smooth part (bto)
        wNew = groupL1prox(groups, wNew, lambda, alpha)
        # update
        (fNew,gNew) = funObj(wNew)
        # Decrease the step-size if we increased the function
        gtd = dot(g,wNew-w)
        while fNew + lambda*groupL1func(groups, wNew) > f + lambda*groupL1func(groups, w) - gamma*alpha*gtd
            @printf("Backtracking\n")
            alpha /= 2
            # Try out the smaller step-size
            wNew = w - alpha*g
            # Proximal step on non-smooth part (bto)
            wNew = groupL1prox(groups, wNew, lambda, alpha)
            # update
            (fNew,gNew) = funObj(wNew)
        end
        # Guess the step-size for the next iteration
        y = gNew - g
        alpha *= -dot(y,g)/dot(y,y)
        # Sanity check on the step-size
        if (!isfinitereal(alpha)) | (alpha < 1e-10) | (alpha > 1e10)
            alpha = 1
        end
        # Accept the new parameters/function/gradient
        w = wNew
        f = fNew
        g = gNew
        # Print out some diagnostics
        optCond = norm(w-groupL1prox(groups, w-g, lambda, alpha),Inf)
        @printf("%6d %15.5e %15.5e %15.5e\n",i,alpha,f+lambda*groupL1func(groups, w),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
```

Figure 6:



```
function softmaxClassiferGL1(X, y, f, n, k, lambda)
    # set objective function (dont mess with gradient objective)
    funObj(W_vec) = softmaxObj(W_vec,X,y,k)
    # initialize weights
    w_init =0.005*rand(Int(f*k))
    # train weights
    w = proxGradGroupL1(funObj, f, w_init, lambda)
    # return function
    return w
end
```

Figure 7:

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 $O(nd\frac{L_c}{\mu}\log(1/\epsilon))$, 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: After modifying the code, it took approximately 0.5 seconds and roughly 250 iterations to converge to within tolerances. To check out the code, see Figure 7.

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.

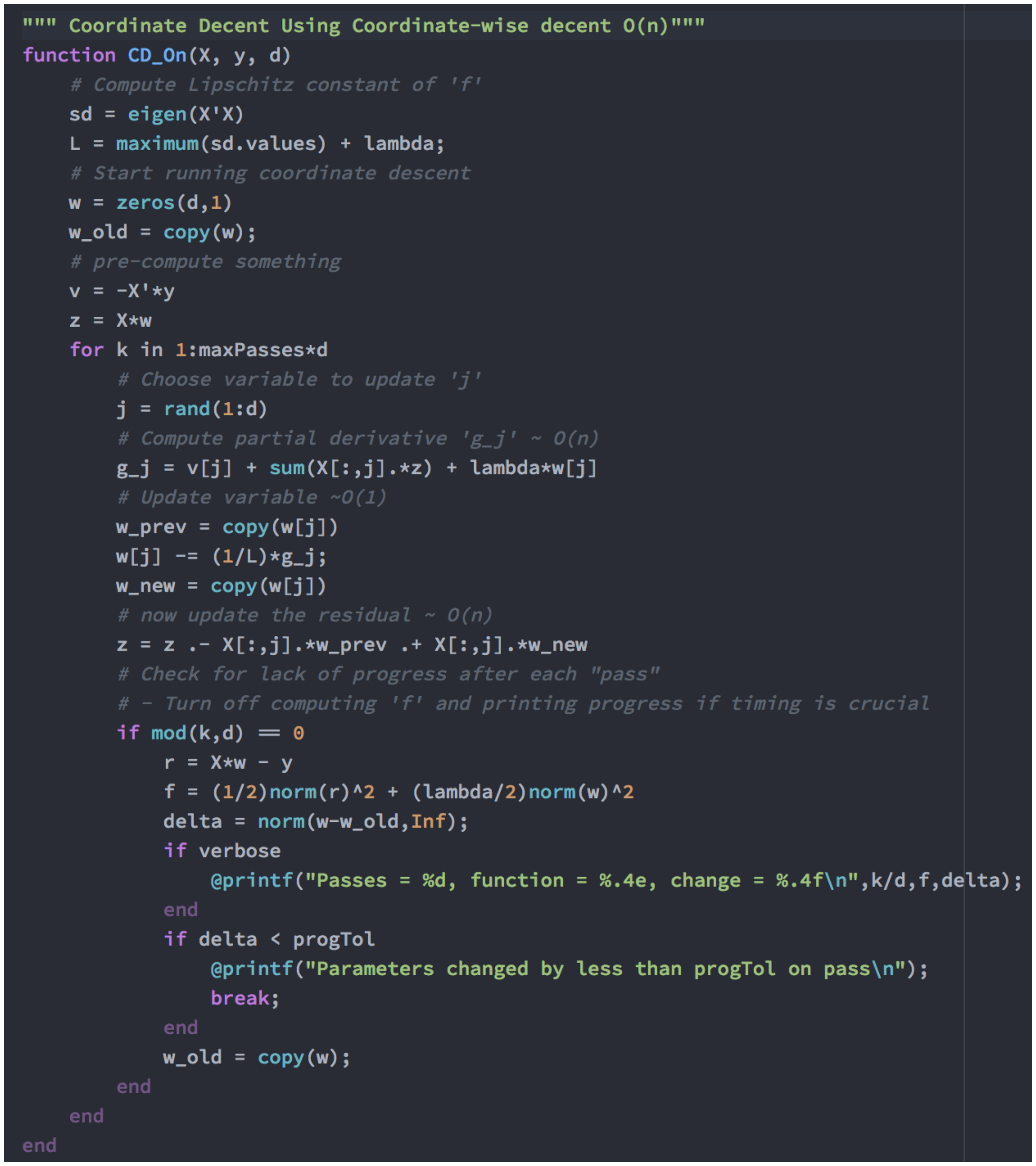
   Answer: After modifying the code, it took approximately 0.25 seconds and roughly 35 iterations to converge to within tolerances. To check out the code, see Figure 11.

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.



```
""" Coordinate Decent Using Coordinate-wise decent O(n)"""
function CD_On(X, y, d)
    # Compute Lipschitz constant of 'f'
    sd = eigen(X'X)
    L = maximum(sd.values) + lambda;
    # Start running coordinate descent
    w = zeros(d,1)
    w_old = copy(w);
    # pre-compute something
    v = -X'*y
    z = X*w
    for k in 1:maxPasses*d
        # Choose variable to update 'j'
        j = rand(1:d)
        # Compute partial derivative 'g_j' ~ O(n)
        g_j = v[j] + sum(X[:,j].*z) + lambda*w[j]
        # Update variable ~O(1)
        w_prev = copy(w[j])
        w[j] -= (1/L)*g_j;
        w_new = copy(w[j])
        # now update the residual ~ O(n)
        z = z .- X[:,j].*w_prev .+ X[:,j].*w_new
        # Check for lack of progress after each "pass"
        # - Turn off computing 'f' and printing progress if timing is crucial
        if mod(k,d) == 0
            r = X*w - y
            f = (1/2)norm(r)^2 + (lambda/2)norm(w)^2
            delta = norm(w-w_old,Inf);
            if verbose
                @printf("Passes = %d, function = %.4e, change = %.4f\n",k/d,f,delta);
            end
            if delta < progTol
                @printf("Parameters changed by less than progTol on pass\n");
                break;
            end
            w_old = copy(w);
        end
    end
end
```

Figure 8:

Answer:

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: 1

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: 1

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: 1

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: bounded below, strongly convex

2. There exists a stationary point.

   Answer: strongly convex

3. There exists at most one stationary point.

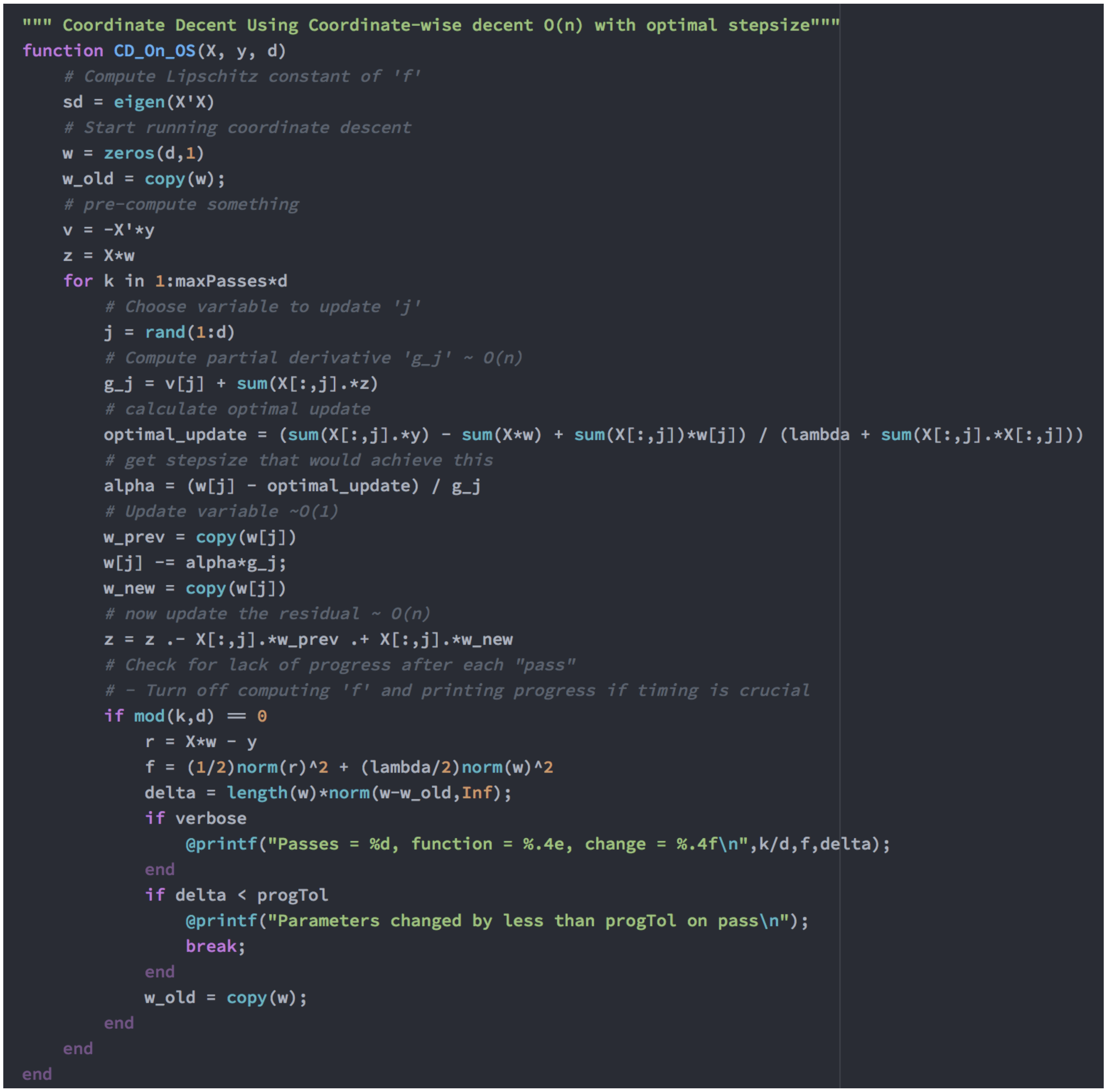
   Answer: strongly convex, strictly convex

4. All stationary points are global optima.

   Answer: convex, strongly convex, strictly convex

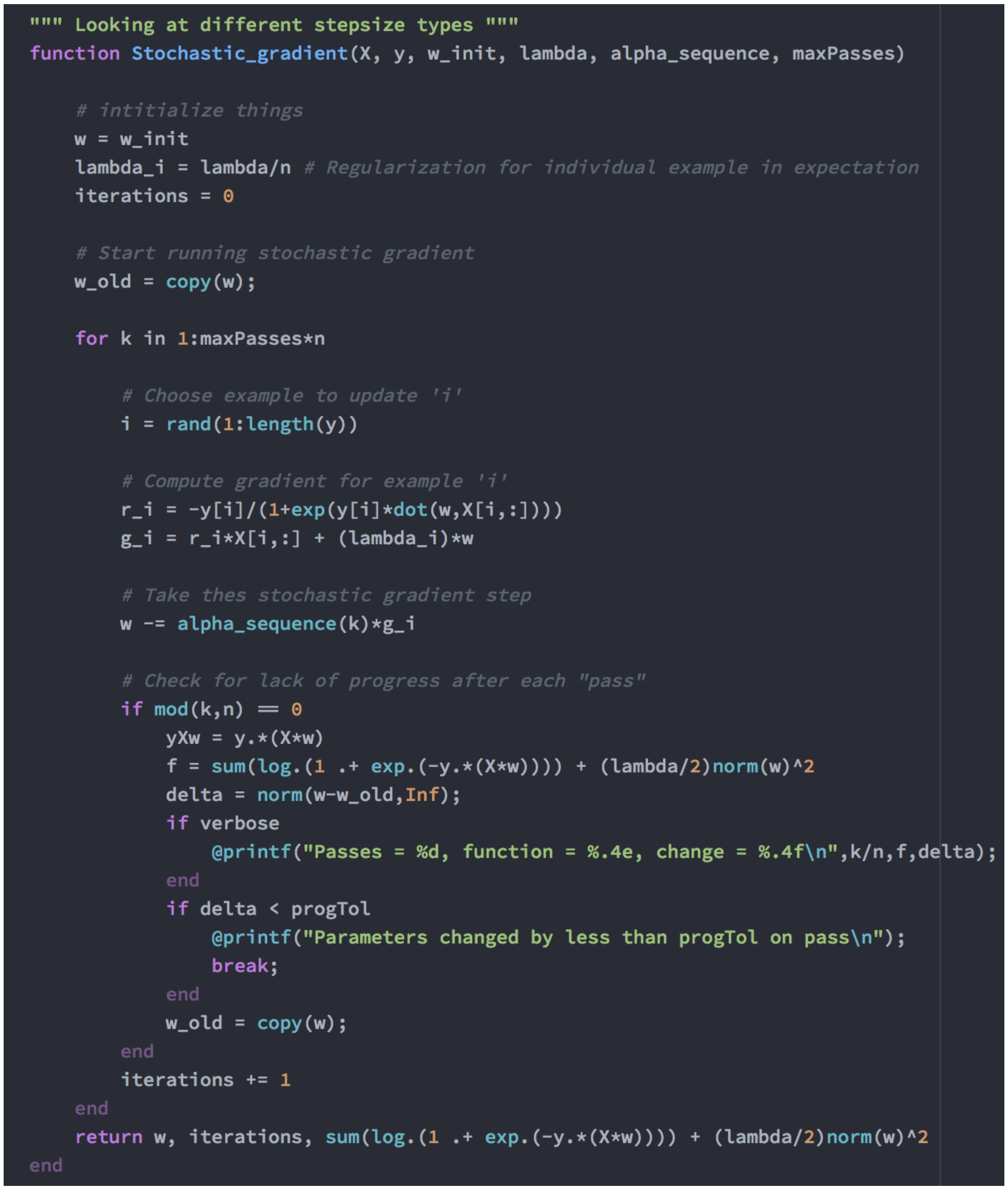
Give a short and concise 1-sentence answer to the below questions.

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.



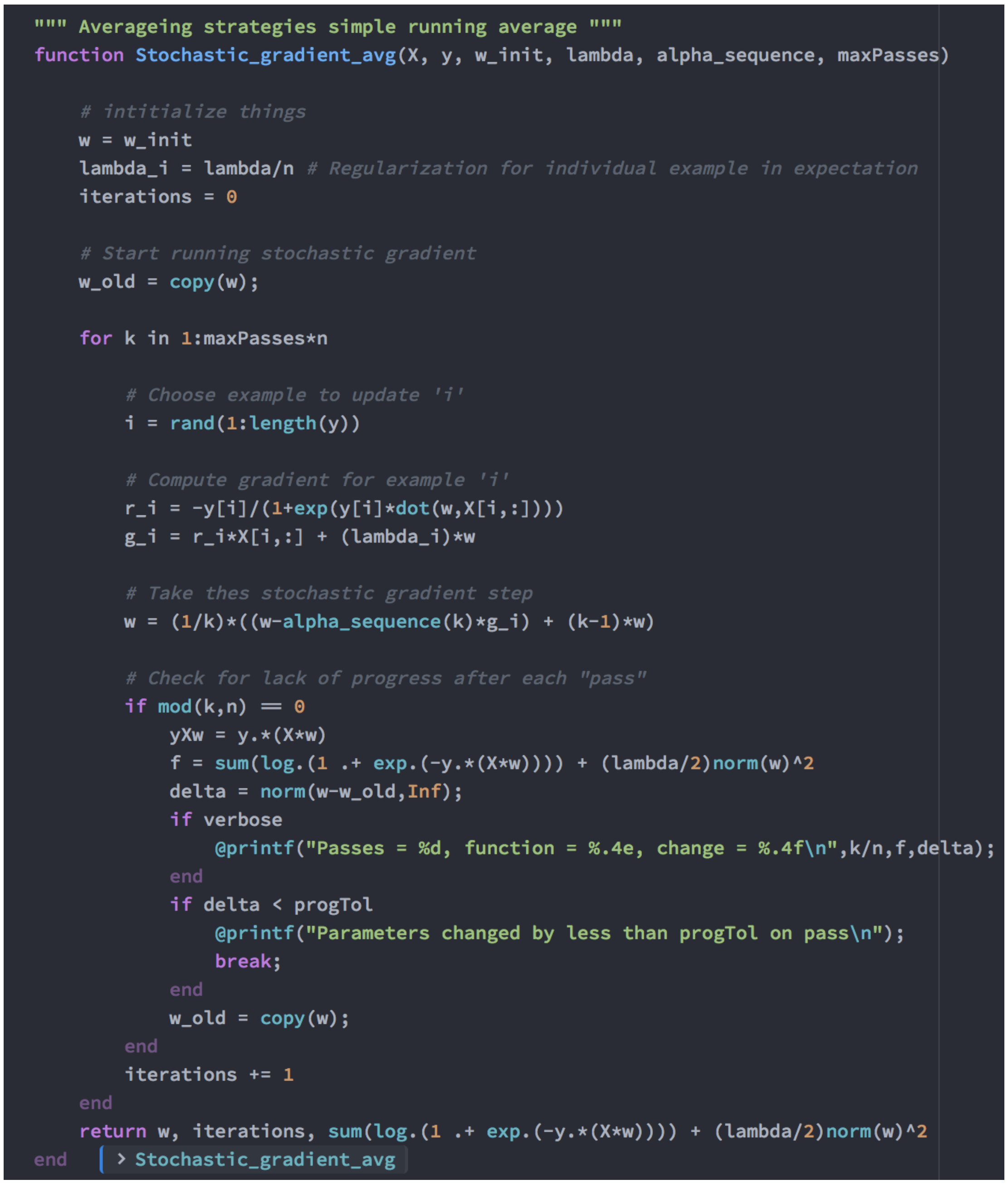
```
""" Coordinate Decent Using Coordinate-wise decent O(n) with optimal stepsize"""
function CD_On_OS(X, y, d)
    # Compute Lipschitz constant of 'f'
    sd = eigen(X'X)
    # Start running coordinate descent
    w = zeros(d,1)
    w_old = copy(w);
    # pre-compute something
    v = -X'*y
    z = X*w
    for k in 1:maxPasses*d
        # Choose variable to update 'j'
        j = rand(1:d)
        # Compute partial derivative 'g_j' ~ O(n)
        g_j = v[j] + sum(X[:,j].*z)
        # calculate optimal update
        optimal_update = (sum(X[:,j].*y) - sum(X*w) + sum(X[:,j])*w[j]) / (lambda + sum(X[:,j].*X[:,j]))
        # get stepsize that would achieve this
        alpha = (w[j] - optimal_update) / g_j
        # Update variable ~O(1)
        w_prev = copy(w[j])
        w[j] -= alpha*g_j;
        w_new = copy(w[j])
        # now update the residual ~ O(n)
        z = z .- X[:,j].*w_prev .+ X[:,j].*w_new
        # Check for lack of progress after each "pass"
        # - Turn off computing 'f' and printing progress if timing is crucial
        if mod(k,d) == 0
            r = X*w - y
            f = (1/2)norm(r)^2 + (lambda/2)norm(w)^2
            delta = length(w)*norm(w-w_old,Inf);
            if verbose
                @printf("Passes = %d, function = %.4e, change = %.4f\n",k/d,f,delta);
            end
            if delta < progTol
                @printf("Parameters changed by less than progTol on pass\n");
                break;
            end
            w_old = copy(w);
        end
    end
end
```

Figure 9:



```julia
""" Looking at different stepsize types """
function Stochastic_gradient(X, y, w_init, lambda, alpha_sequence, maxPasses)

    # intitialize things
    w = w_init
    lambda_i = lambda/n # Regularization for individual example in expectation
    iterations = 0

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

    for k in 1:maxPasses*n

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

        # 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

        # Take thes stochastic gradient step
        w -= alpha_sequence(k)*g_i

        # Check for lack of progress after each "pass"
        if mod(k,n) == 0
            yXw = y.*(X*w)
            f = sum(log.(1 .+ exp.(-y.*(X*w)))) + (lambda/2)norm(w)^2
            delta = norm(w-w_old,Inf);
            if verbose
                @printf("Passes = %d, function = %.4e, change = %.4f\n",k/n,f,delta);
            end
            if delta < progTol
                @printf("Parameters changed by less than progTol on pass\n");
                break;
            end
            w_old = copy(w);
        end
        iterations += 1
    end
    return w, iterations, sum(log.(1 .+ exp.(-y.*(X*w)))) + (lambda/2)norm(w)^2
end
```

Figure 10:



```
""" Averageing strategies simple running average """
function Stochastic_gradient_avg(X, y, w_init, lambda, alpha_sequence, maxPasses)

    # intitialize things
    w = w_init
    lambda_i = lambda/n # Regularization for individual example in expectation
    iterations = 0

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

    for k in 1:maxPasses*n

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

        # 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

        # Take thes stochastic gradient step
        w = (1/k)*((w-alpha_sequence(k)*g_i) + (k-1)*w)

        # Check for lack of progress after each "pass"
        if mod(k,n) == 0
            yXw = y.*(X*w)
            f = sum(log.(1 .+ exp.(-y.*(X*w)))) + (lambda/2)norm(w)^2
            delta = norm(w-w_old,Inf);
            if verbose
                @printf("Passes = %d, function = %.4e, change = %.4f\n",k/n,f,delta);
            end
            if delta < progTol
                @printf("Parameters changed by less than progTol on pass\n");
                break;
            end
            w_old = copy(w);
        end
        iterations += 1
    end
    return w, iterations, sum(log.(1 .+ exp.(-y.*(X*w)))) + (lambda/2)norm(w)^2
end    > Stochastic_gradient_avg
```

Figure 11:

Answer: Because the universe is fundamentally ordered (at least at a local level), models that can capture this order, and its inter-connections in an efficient way become more useful then models that do not. In the case of naive Bayes, we are able to capture some of this local structure, but the independence assumption ignores other parts of the structure; namely: the connections between predictors of our labels. As a result, a model (like XGBoost) that can better express this structure will do better.

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

   Answer: If you know the PL inequality is satisfied, you can prove a linear convergence time. Additionally, we can directly apply the PL inequality to derive a stepsize.

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

   Answer: If it has achieved the minimum of the function, or if the scheme to choose values from the sub differentials is poor.

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

   Answer: Using the L1 norm for each of the groups would just correspond to the original L1 regularization objective, and thus would not incentivise regularizing sets of parameters.

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

   Answer: In transductive learning we dont need to be able to predict on new examples. In inductive semi-supervised learning goal is to predict well on new examples.

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: Coordinate descent is still d times faster in expectation if you randomly pick j_k. Therefore in problems that have large D, it would behoove you to pick uniformly instead of iterating through the data set.

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 problems they are substantially cheaper per iteration to compute, and yield the same convergence rate as non-stochastic sub-gradient methods.

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? In other words, if you find a step-size sequnece $\{\alpha_k\}$ that works for a batch-size of m but now you want to use a batch size of $10m$ (which would divide the variance by 10, why might using $\{10\alpha_k\}$ be problematic?

    Answer: It assumes that the variance in your dataset decreases linearly with the number of examples that you include in your approximation, which might not be true, especially for smaller batch estimates.

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

    Answer: SVRG algorithm gets rid of memory by occasionally computing exact gradient. Additionally, it has convergence properties similar to SAG (for suitable m): its unbiased, theoretically m depends on n μ, and L, and finally in practice m=n works well in practice. This means that you get the same results as SAG, but with less computational cost.