Math 4650/MSSC 5650 - Homework 7

Henri Medeiros Dos Reis

Spring 2023

Recall that the *logistic regression cost function* $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$f(\mathbf{x}) = -\sum_{i=1}^{N} \left[b_i \log \sigma(\mathbf{a}_i^\top \mathbf{x}) + (1 - b_i) \log \left(1 - \sigma(\mathbf{a}_i^\top \mathbf{x})\right)\right]. \tag{1}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the logistic function $\sigma(t) = \frac{1}{1+e^{-t}}$, and $\mathbf{a}_1, ..., \mathbf{a}_N \in \mathbb{R}^n$ are the training data vectors with corresponding training labels $b_1, ..., b_N \in \{0, 1\}$.

Problem 1 (10 pts)**.** Let $b \in \{0, 1\}$, and $\mathbf{a} \in \mathbb{R}^n$ be any vector. Define

$$g(\mathbf{x}) = -\left[b \log \sigma(\mathbf{a}^\top \mathbf{x}) + (1 - b) \log \left(1 - \sigma(\mathbf{a}^\top \mathbf{x})\right)\right]$$

i.e., one term appearing in the logistic regression cost function.

(a) Prove that

$$\nabla g(\mathbf{x}) = (\sigma(\mathbf{a}^\top \mathbf{x}) - b)\, \mathbf{a}.$$

(Hint: the identity $\sigma'(t) = \sigma(t)(1 - \sigma(t))$ might come in handy)

(b) Using the result of part (a), deduce that the logistic regression cost function $f(\mathbf{x})$ defined in (1) has the gradient

$$\nabla f(\mathbf{x}) = \sum_{i=1}^{N} (\sigma(\mathbf{a}_i^\top \mathbf{x}) - b_i)\mathbf{a}_i.$$

Solution 1. (a)

$$\begin{aligned} g(x) &= -blog\sigma(a^\top x) - (1 - b)log(1 - \sigma(a^\top x)) \\ &= -blog\sigma(a^\top x) - [log(1 - \sigma(a^\top x)) - blog(1 - \sigma(a^\top x))] \\ &= -blog(\sigma(a^\top x)) - log(1 - \sigma(a^\top x)) + blog(1 - \sigma(a^\top x)) \end{aligned}$$

Then we can take the gradient using the identity $\sigma'(a^\top x) = \sigma(a^\top x)(1-\sigma(a^\top x))a$ by the chain rule

$$\begin{aligned}
\nabla g(x) &= -b\nabla log(\sigma(a^\top x)) - \nabla log(1-\sigma(a^\top x)) + b\nabla log(1-\sigma(a^\top x)) \\
&= -b(\frac{\sigma(a^\top x)(1-\sigma(a^\top x))a}{\sigma(a^\top x)}) - \frac{-\sigma(a^\top x)(1-\sigma(a^\top x))a}{(1-\sigma(a^\top x))} + b\frac{-\sigma(a^\top x)(1-\sigma(a^\top x))a}{(1-\sigma(a^\top x))} \\
&= -b((1-\sigma(a^\top x))a) - (-\sigma(a^\top x)a) + b(-\sigma(a^\top x)a) \\
&= -b(a-\sigma(a^\top x)a) + \sigma(a^\top x)a - b\sigma(a^\top x)a \\
&= -ba + b\sigma(a^\top x)a + \sigma(a^\top x)a - b\sigma(a^\top x)a \\
&= -ba + \sigma(a^\top x)a \\
&= (\sigma(a^\top x) - b)a
\end{aligned}$$

Therefore $\nabla g(x) = (\sigma(a^\top x) - b)\, a$

(b) Since $f(x) = \sum_{i=1}^N g(x)$ then

$$\nabla f(x) = \nabla \sum_{i=1}^N g(x) = \sum_{i=1}^N \nabla g(x) = \sum_{i=1}^N (\sigma(a_i^\top x) - b_i)a_i$$

Problem 2 (5 pts)**.** In logistic regression, if the number of training data points N is less than the ambient dimension n, then it is possible to *overfit* the training data, meaning the trained model will give perfect predictions on on the training set but fail to perform well on new data. This can be mitigated to some extent by including appropriate *regularization*, similar to regularized least squares. One simple form of regularization is to penalize the squared Euclidean norm of the weight vector $\mathbf{x} \in \mathbb{R}^n$ in addition to the typical logistic regression cost function $f(\mathbf{x})$ as defined in (1), i.e., the optimization problem to solve is

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) + \lambda\|\mathbf{x}\|^2 \tag{2}$$

Here $\lambda > 0$ is a regularization parameter that needs to be tuned.

This problem investigates using regularized linear regression for classifying handwritten 3's and 8's using only $N = 50$ training images, where overfitting is a risk. The provided script `logistic_regression_mnist.m` implements unregularized logistic regression ($\lambda = 0$) for this problem. Your task is to modify the script so that it implements regularized logistic regression by making the appropriate modifications to the definitions of the functions `f` and `grad` (these are the only changes to the script you need to make). Then, you should find the parameter $\lambda > 0$ that maximizes the percent correct on the test set (the percent correct is computed for you already, and is automatically output by running the script).

Include the following in your write-up:

- The changes you made to the definitions of `f` and `grad` in the code (i.e., copy and paste your modified code for these lines).

- The value of λ that gave the highest percent correct on the test set.
- Report the percent correct on the training set, and the percent correct on the test set both without regularization ($\lambda = 0$) and with regularization ($\lambda \neq 0$).
- Pictures of the learned weights $\mathbf{x}$ obtained without regularization ($\lambda = 0$) and with regularization ($\lambda \neq 0$). Also, briefly describe in words the visual differences between weights learned with regularization versus without regularization.

Solution 2. The definitions to `f` and `grad`:

```
f = @(x) -sum(b.*log(sigma(A'*x))+(1-b).*log(1-sigma(A'*x)))+
   lambda*norm(x);
grad = @(x) A*(sigma(A'*x)-b)+lambda*2*x;
```

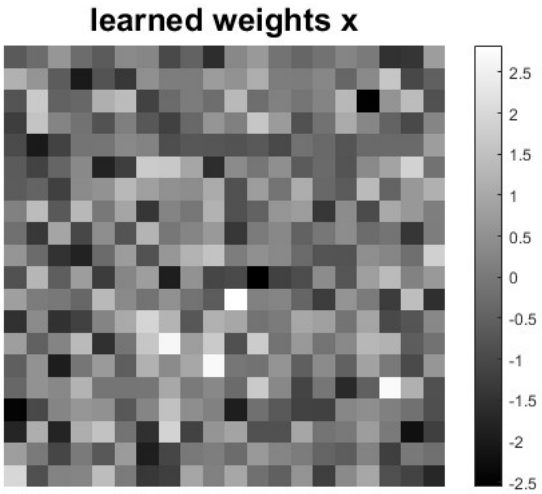
The value of λ that gave the highest percent correct on the test set was $\lambda = 1.5$.
Percent scores:
$\lambda = 0$

```
percent correct on training set = 100.00 %
percent correct on test set = 74.00 %
```

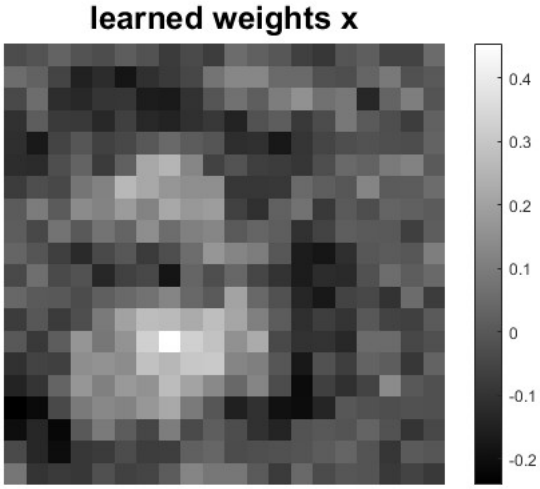
$\lambda \neq 0$

```
percent correct on training set = 100.00 %
percent correct on test set = 91.00 %
```

Learned weights:
$\lambda = 0$



$\lambda \neq 0$



It is possible to see that the learned weights when $\lambda = 0$, the picture does not seem "smooth" the pixels that are close to each other do not seem to be correlated with each other at all. While the picture with the learned weights when $\lambda = 1.5$, the color in the pixels are much closer to each other, it is also possible to see that the white pixels that would represent the hand written number are much more centralized.

Problem 3 (MATLAB, 5 pts)**.** For this problem, you will modify the denoising script `gd_denoise_slow.m` to make it more efficient. In the script, a "naive" version of backtracking line search is implemented, as shown below:

```
tic
cost = f(x);
for k=1:maxiter
    d = -grad(x); %steepest descent direction
    t = s; %initial stepsize
    while (f(x) - f(x+t*d)) < -t*alpha*d'*grad(x)
        t = beta*t; %shrink stepsize
    end
    x = x + t*d;

    cost = [cost,f(x)]; %store f(x_k) for plotting

    if norm(grad(x)) < tol
        fprintf('algorithm converged at iteration k=%d\n',k);
        break;
    end
```

```
end
toc
```

This takes about 19 seconds to run on my laptop. However, this code is very inefficient, and makes several unnecessary calls to `f(x)` and `grad(x)`, both of which take a lot of time to compute. Your goal is to speed up the code by re-writing the main for loop to minimize the number of evaluations `f(x)` and `grad(x)`. After re-writing, your code should compute the *same exact iterates* as before, and should output the same final image, but should be much faster to run. I am able to speed up the code by more than a factor of two, to a runtime of about 7 seconds.

Here are the rules:

- Only the code between `tic` and `toc` can be changed (i.e., the main for-loop). Though, you may add lines of code *before* the for-loop if necessary, but they must come after the `tic` command.
- In particular, the following parameters must remain unchanged: `maxiter = 100`, `tol = 0.05`, `s = 0.5`, `beta = 0.5`, `alpha = 0.25`
- The same `cost` array needs to be computed. (Note: MATLAB gives the warning `The variable 'cost' appears to change size on every loop iteration. Consider preallocating for speed.` You can ignore this; the change it is suggesting won't speed up the loop in this case.)
- The same exit condition needs to be checked at each iteration. In particular, after re-writing the code, the algorithm should exit after the same number of iterations as before.

Include the following in your write-up:

- A printout of your code for the modified main for-loop (between `tic` and `toc`).
- A brief description (2-4 sentences) of any changes you made.
- The time T_0 the main for loop took to run *before* modifying the code, and the time T_1 the main for-loop took to run after *after* modifying the code, and the resulting speed-up factor T_0/T_1.

Solution 3.

```
tic
cost = f(x); %initialize cost array
for k=1:maxiter
    d = -grad(x); %steepest descent direction
    f_x = f(x);
    t = s; %initial stepsize
    while (f_x - f(x+t*d)) < t*alpha*d'*d
```

```
        t = beta*t; %shrink stepsize
    end
    x = x + t*d;

    cost = [cost,f_x]; %store f(x_k) for plotting

    if norm(grad(x)) < tol
        fprintf('algorithm converged at iteration k=%d\n',k);
        break;
    end
end
toc
```

The first change is the creation of the variable `f_k`, that eliminates the process of calculating $f(x)$ two times, and only does it once. The second change is in the while condition, instead of computing the gradient again, just use the variable `d` that already has gradient, where the only difference is the negative sign.

Then the results of that process, $T_0 = 14.929334$, $T_1 = 6.386480$, and $T_0/T_1 = 2.3376$

Problem 4 (5 pts)**.** Consider the problem of image restoration with non-quadratic edge-preserving regularization:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 + \lambda R(\mathbf{x}) \tag{3}$$

where $R(\mathbf{x}) = \sum_{i=1}^{K} \phi([\mathbf{L}\mathbf{x}]_i)$ for some potential function $\phi(t)$, with associated weighting function $\omega(t) = \phi'(t)/t$.

In the lecture notes, it was shown that any minimizer $\mathbf{x}^*$ of the above cost function must satisfy the equation:

$$\mathbf{x}^* = (\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{L}^\top\mathbf{D}_\omega(\mathbf{L}\mathbf{x}^*)\mathbf{L})^{-1}\mathbf{A}^\top\mathbf{b}$$

where for any vector $\mathbf{y} = [y_1, y_2, ..., y_K]^\top \in \mathbb{R}^K$ we define $\mathbf{D}_\omega(\mathbf{y})$ to be the diagonal matrix having entries $\omega(\mathbf{y}) = [\omega(y_1), \omega(y_2), ..., \omega(y_K)]^\top \in \mathbb{R}^K$ along the diagonal.

The above equation suggests the following iterative algorithm for solving the minimization problem in (3): given the current iterate $\mathbf{x}_k$, find the next iterate $\mathbf{x}_{k+1}$ by plugging in $\mathbf{x}_k$ for $\mathbf{x}^*$ in the right-hand side to get

$$\mathbf{x}_{k+1} = (\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{L}^\top\mathbf{D}_\omega(\mathbf{L}\mathbf{x}_k)\mathbf{L})^{-1}\mathbf{A}^\top\mathbf{b}.$$

This approach is known as an "iteratively reweighted least squares" (IRLS) algorithm, since each iteration above can be recast as the solution to a regularized least squares problem.

Your task is to implement the above IRLS algorithm in an image deblurring context by modifying the provided script `deblur_irls.m`.

- In the script, the matricies $\mathbf{A}$, $\mathbf{L}$, and vector $\mathbf{b}$ are defined for you, and so is the function $\omega(\cdot)$ as `omega()`. You only need to edit the code block titled `%% run IRLS alg`.

- Normally, to make a diagonal matrix in MATLAB from a vector `y`, we would use the command `diag(y)`. But due to the large-scale nature of this problem, you will need to use a "sparse" diagonal matrix with the command `spdiag(y)` instead.
- Run 10 iterations of the IRLS algorithm with $\lambda = 0.001$ starting from the initial iterate $\mathbf{x}_0 = \mathbf{0}$.
- (Optional) To ensure your algorithm is working correctly, it can help to monitor the value of the cost function at each iteration. This can be computed with the command `0.5*norm(A*x-b)^2 + lambda*sum(phi(L*x))` (assuming `x` is the current iterate). The cost should decrease with each iteration, with big changes in the first few iterations and smaller changes as the iterations progress.

Include the follwing in your write-up:

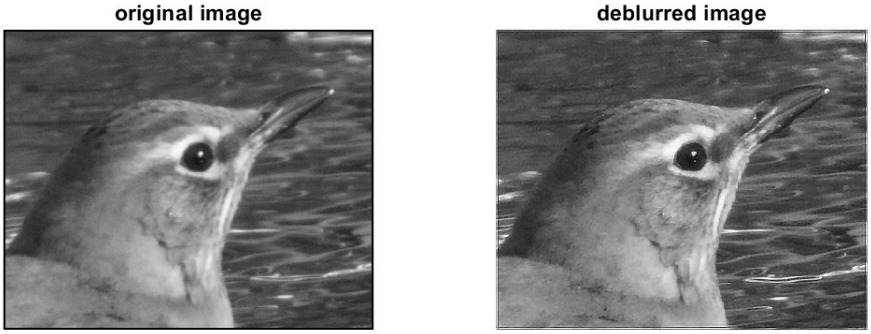
- A printout/screenshot of any code you added to the script.
- A side-by-side comparison of the blurry input image and the deblurred output image (the last code block in the script generates this for you).

Solution 4. Code:

```
%% run IRLS alg
x = zeros(size(A,2),1); %initialize x
lambda = 0.001;
maxiter = 10;
%initialize cost array
cost = 0.5*norm(A*x-b)^2+lambda*sum(phi(L*x));

for k=1:maxiter
    left = (A'*A+lambda*L'*spdiag(omega(L*x))*L);
    x = left\(A'*b);
    cost = [cost,0.5*norm(A*x-b)^2+lambda*sum(phi(L*x))];
    %store f(x_k) for plotting
end
```

Side by side comparison:



Problem 5 (MSSC, 5 pts)**.** Let $f(\mathbf{x})$ be the logistic regression cost function given in (1). Prove that f is *convex*, i.e., $\nabla^2 f(\mathbf{x}) \succeq 0$ for all $\mathbf{x} \in \mathbb{R}^n$. What does this imply about any critical points of f (if they exist)?

Solution 5.

$$\nabla f(x) = \sum_{i=1}^{N} (\sigma(a_i^\top x) - b_i) a_i$$

Let's look at only one term

$$\begin{aligned} \nabla g(x) &= (\sigma(a_i^\top x) - b_i) a_i \\ &= \sigma(a_i^\top x) a_i - b_i a_i \end{aligned}$$

$$\Rightarrow \nabla^2 g(x) = \nabla \sigma(a_i^\top x) a_i - \nabla b_i a_i$$

Where $\sigma(a_i^\top x) a_i = \begin{bmatrix} a_{i1}\sigma(a_i^\top x) \\ a_{i2}\sigma(a_i^\top x) \\ \vdots \\ a_{in}\sigma(a_i^\top x) \end{bmatrix}$ and $b_i a_i$ is a constant vector. Then

$$\nabla^2 g(x) = \begin{bmatrix} \frac{\partial^2 g}{\partial x_1^2} & \frac{\partial^2 g}{\partial x_1 x_2} & \cdots & \cdots \\ \frac{\partial^2 g}{\partial x_2 x_1} & \frac{\partial^2 g}{\partial x_2^2} & \cdots & \cdots \\ \vdots & \vdots & \cdots & \frac{\partial^2 g}{\partial x_n^2} \end{bmatrix}$$

Then the general term $\frac{\partial^2 g}{\partial x_j x_i} = \frac{g}{\partial x_j}[a_i \sigma(a^\top x)] = a_i a_j (\sigma(a^\top x)(1 - \sigma(a^\top x))$, which is the (i,j) entry of $\nabla^2 g(x)$.

$$\Rightarrow \nabla^2 g(x) = \sigma(a^\top x)(1 - \sigma(a^\top x)) \begin{bmatrix} a_1 a_1 & a_1 a_2 & \cdots & a_1 a_n \\ a_1 a_2 & a_2 a_2 & \cdots & a_2 a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_1 a_n & a_2 a_n & \cdots & a_n a_n \end{bmatrix} = \sigma(a^\top x)(1 - \sigma(a^\top x)) a a^\top$$

And we need it to be positive semi-definite. Let's check it by multiplying by any vector $y \in R^n$ on the left and $y^\top$ on the right. And since $\sigma(a^\top x)(1 - \sigma(a^\top x)) a a^\top$ is a scalar.

$$\begin{aligned} y^\top (\sigma(a^\top x)(1 - \sigma(a^\top x)) a a^\top) y &= \sigma(a^\top x)(1 - \sigma(a^\top x)) y^\top a a^\top y \\ &= \sigma(a^\top x)(1 - \sigma(a^\top x)) y^\top a a^\top y \\ &= \sigma(a^\top x)(1 - \sigma(a^\top x))(a^\top y)^\top a^\top y \\ &= \sigma(a^\top x)(1 - \sigma(a^\top x)) \|a^\top y\|^2 \geq 0 \end{aligned}$$

And we know that $\|a^\top y\|^2 \geq 0$, let's check the other term.

$\sigma(a^\top x)(1-\sigma(a^\top x)) = (\frac{1}{1+e^{-a^\top x}})(1-\frac{1}{1+e^{-a^\top x}})$. Since $e^z \geq 0$ for any real value z, then $\frac{1}{1+e^{-a^\top x}} > 0$ and $(1-\frac{1}{1+e^{-a^\top x}})$ is always positive since $\frac{1}{1+e^{-a^\top x}} < 1$.

Therefore $\sigma(a^\top x)(1-\sigma(a^\top x))\|a^\top y\|^2 \geq 0$, $\nabla^2 g(x) \succeq 0$, and $\nabla^2 f(x) = \sum_{i=1}^N \nabla^2 g(x) \succeq 0$, since the sum of positive semi definite matrices is also a positive semi definite matrix. Which means that f is convex, implying that any critical point, if it exists, it is also the global minimum.

Problem 6 (MSSC, 5 pts)**.** Let $f(\mathbf{x})$ be the logistic regression cost function given in (1). Suppose that the number of data points N exceeds the ambient dimension n, and that the matrix of data-points $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_N] \in \mathbb{R}^{n\times N}$ has linearly independent rows (i.e., $\text{rank}(\mathbf{A}) = n$). Prove that in this case f is *strictly convex*, i.e., $\nabla^2 f(\mathbf{x}) \succ 0$ for all $\mathbf{x} \in \mathbb{R}^n$. What implications does this have regarding global minimizers of f (if they exist)?

Solution 6. From last problem we have $\nabla^2 g(x) = \sigma(a^\top x)(1-\sigma(a^\top x))aa^\top$ and $\nabla^2 f(x) = \sum_{i=1}^N \nabla^2 g(x)$.

Let's check for positive semi definiteness by multiplying f by any vector $y \in R^n$ on the left and $y^\top$ on the right.

$$\sum_{i=1}^N y^\top(\sigma(a_i^\top x)(1-\sigma(a_i^\top x))a_i a_i^\top)y > 0$$

$$\sum_{i=1}^N \sigma(a_i^\top x)(1-\sigma(a_i^\top x)\|a_i^\top y\|^2 > 0$$

Which is only true if $\sigma(a_i^\top x)(1-\sigma(a_i^\top x) > 0$ and $\|a_i^\top y\|^2 > 0$. From last problem, we know $\sigma(a_i^\top x)(1-\sigma(a_i^\top x) > 0$, so let's check $\|a_i^\top y\|^2$.

Since A has linearly independent rows, the null space of A is trivial, the zero matrix. That means that $a_i^\top y = 0$ if and only if y is orthogonal to a_i, and for the whole sum to be equal to 0, y would need to be orthogonal to all rows of A, which violates the assumption that the rows of A are linearly independent. Hence, $y^\top(\nabla^2 f(x))y > 0$ for all nonzero y, and $f(x)$ is strictly convex.

The fact that $f(x)$ is strictly convex implies that it has a unique global minimizer (if it exists). Moreover, any local minimizer of $f(x)$ is also a global minimizer, since there are no local minimizers that are not global minimizers in strictly convex functions. Therefore, if we can find a global minimizer of $f(x)$, it is guaranteed to be unique.