CS 229, Public Course
Problem Set #1 Solutions: Supervised Learning

---

1. **Newton's method for computing least squares**

   In this problem, we will prove that if we use Newton's method solve the least squares optimization problem, then we only need one iteration to converge to θ^*.

   (a) Find the Hessian of the cost function $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$.

   Answer: As shown in the class notes

   $$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_j^{(i)}.$$

   So

   $$\begin{aligned}\frac{\partial^2 J(\theta)}{\partial \theta_j \partial \theta_k} &= \sum_{i=1}^{m}\frac{\partial}{\partial \theta_k}(\theta^T x^{(i)} - y^{(i)})x_j^{(i)} \\ &= \sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} = (X^T X)_{jk}\end{aligned}$$

   Therefore, the Hessian of $J(\theta)$ is $H = X^T X$. This can also be derived by simply applying rules from the lecture notes on Linear Algebra.

   (b) Show that the first iteration of Newton's method gives us $\theta^\star = (X^T X)^{-1} X^T \vec{y}$, the solution to our least squares problem.

   Answer: Given any $\theta^{(0)}$, Newton's method finds $\theta^{(1)}$ according to

   $$\begin{aligned}\theta^{(1)} &= \theta^{(0)} - H^{-1}\nabla_\theta J(\theta^{(0)}) \\ &= \theta^{(0)} - (X^T X)^{-1}(X^T X\theta^{(0)} - X^T\vec{y}) \\ &= \theta^{(0)} - \theta^{(0)} + (X^T X)^{-1} X^T \vec{y} \\ &= (X^T X)^{-1} X^T \vec{y}.\end{aligned}$$

   Therefore, no matter what $\theta^{(0)}$ we pick, Newton's method always finds $\theta^\star$ after one iteration.

2. **Locally-weighted logistic regression**

   In this problem you will implement a locally-weighted version of logistic regression, where we weight different training examples differently according to the query point. The locally-weighted logistic regression problem is to maximize

   $$\ell(\theta) = -\frac{\lambda}{2}\theta^T\theta + \sum_{i=1}^{m} w^{(i)}\left[y^{(i)}\log h_\theta(x^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right].$$

The $-\frac{\lambda}{2}\theta^T\theta$ here is what is known as a regularization parameter, which will be discussed in a future lecture, but which we include here because it is needed for Newton's method to perform well on this task. For the entirety of this problem you can use the value $\lambda = 0.0001$.

Using this definition, the gradient of $\ell(\theta)$ is given by

$$\nabla_\theta \ell(\theta) = X^T z - \lambda\theta$$

where $z \in \mathbb{R}^m$ is defined by

$$z_i = w^{(i)}(y^{(i)} - h_\theta(x^{(i)}))$$

and the Hessian is given by

$$H = X^T D X - \lambda I$$

where $D \in \mathbb{R}^{m\times m}$ is a diagonal matrix with

$$D_{ii} = -w^{(i)} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))$$

For the sake of this problem you can just use the above formulas, but you should try to derive these results for yourself as well.

Given a query point x, we choose compute the weights

$$w^{(i)} = \exp\left(-\frac{||x - x^{(i)}||^2}{2\tau^2}\right).$$

Much like the locally weighted linear regression that was discussed in class, this weighting scheme gives more when the "nearby" points when predicting the class of a new example.

(a) Implement the Newton-Raphson algorithm for optimizing $\ell(\theta)$ for a new query point x, and use this to predict the class of x.

The `q2/` directory contains data and code for this problem. You should implement the `y = lwlr(X_train, y_train, x, tau)` function in the `lwlr.m` file. This function takes as input the training set (the `X_train` and `y_train` matrices, in the form described in the class notes), a new query point `x` and the weight bandwitdh `tau`. Given this input the function should 1) compute weights $w^{(i)}$ for each training example, using the formula above, 2) maximize $\ell(\theta)$ using Newton's method, and finally 3) output $y = 1\{h_\theta(x) > 0.5\}$ as the prediction.

We provide two additional functions that might help. The `[X_train, y_train] = load_data;` function will load the matrices from files in the `data/` folder. The function `plot_lwlr(X_train, y_train, tau, resolution)` will plot the resulting classifier (assuming you have properly implemented `lwlr.m`). This function evaluates the locally weighted logistic regression classifier over a large grid of points and plots the resulting prediction as blue (predicting $y = 0$) or red (predicting $y = 1$). Depending on how fast your `lwlr` function is, creating the plot might take some time, so we recommend debugging your code with `resolution = 50;` and later increase it to at least 200 to get a better idea of the decision boundary.

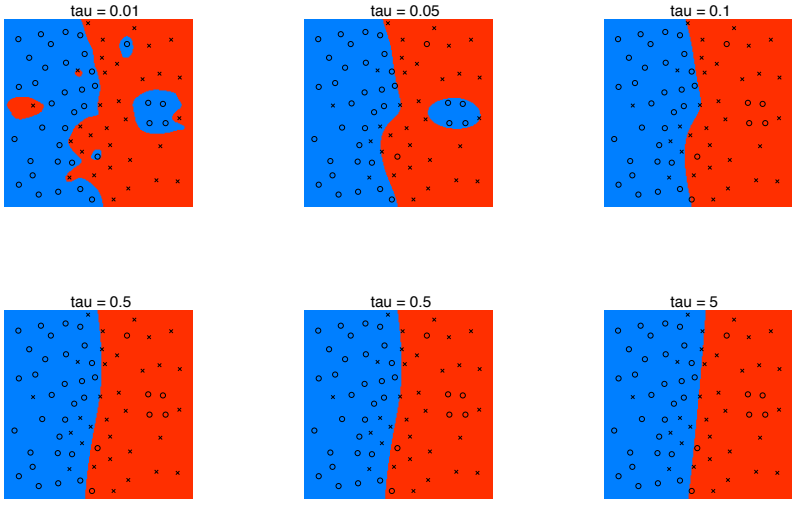
Answer: Our implementation of `lwlr.m`:

```
function y = lwlr(X_train, y_train, x, tau)

m = size(X_train,1);
n = size(X_train,2);
```

```
theta = zeros(n,1);

% compute weights
w = exp(-sum((X_train - repmat(x', m, 1)).^2, 2) / (2*tau));

% perform Newton's method
g = ones(n,1);
while (norm(g) > 1e-6)
  h = 1 ./ (1 + exp(-X_train * theta));
  g = X_train' * (w.*(y_train - h)) - 1e-4*theta;
  H = -X_train' * diag(w.*h.*(1-h)) * X_train - 1e-4*eye(n);
  theta = theta - H \ g;
end

% return predicted y
y = double(x'*theta > 0);
```

(b) Evaluate the system with a variety of different bandwidth parameters τ. In particular, try $\tau = 0.01, 0.050.1, 0.51.0, 5.0$. How does the classification boundary change when varying this parameter? Can you predict what the decision boundary of ordinary (unweighted) logistic regression would look like?

Answer: These are the resulting decision boundaries, for the different values of τ.



For smaller τ, the classifier appears to overfit the data set, obtaining zero training error, but outputting a sporadic looking decision boundary. As τ grows, the resulting decision boundary becomes smoother, eventually converging (in the limit as $\tau \to \infty$ to the unweighted linear regression solution).

3. **Multivariate least squares**

So far in class, we have only considered cases where our target variable y is a scalar value. Suppose that instead of trying to predict a single output, we have a training set with

multiple outputs for each example:

$$\{(x^{(i)}, y^{(i)}),\ i = 1, \ldots, m\},\ x^{(i)} \in \mathbb{R}^n,\ y^{(i)} \in \mathbb{R}^p.$$

Thus for each training example, $y^{(i)}$ is vector-valued, with p entries. We wish to use a linear model to predict the outputs, as in least squares, by specifying the parameter matrix Θ in

$$y = \Theta^T x,$$

where $\Theta \in \mathbb{R}^{n \times p}$.

(a) The cost function for this case is

$$J(\Theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} \left((\Theta^T x^{(i)})_j - y_j^{(i)} \right)^2.$$

Write $J(\Theta)$ in matrix-vector notation (i.e., without using any summations). [Hint: Start with the $m \times n$ design matrix

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix}$$

and the $m \times p$ target matrix

$$Y = \begin{bmatrix} — & (y^{(1)})^T & — \\ — & (y^{(2)})^T & — \\ & \vdots & \\ — & (y^{(m)})^T & — \end{bmatrix}$$

and then work out how to express $J(\Theta)$ in terms of these matrices.]

Answer: The objective function can be expressed as

$$J(\Theta) = \frac{1}{2} \mathrm{tr} \left((X\Theta - Y)^T (X\Theta - Y) \right).$$

To see this, note that

$$\begin{aligned} J(\Theta) &= \frac{1}{2} \mathrm{tr} \left((X\Theta - Y)^T (X\Theta - Y) \right) \\ &= \frac{1}{2} \sum_i \left((X\Theta - Y)^T (X\Theta - Y) \right)_{ii} \\ &= \frac{1}{2} \sum_i \sum_j (X\Theta - Y)_{ij}^2 \\ &= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} \left((\Theta^T x^{(i)})_j - y_j^{(i)} \right)^2 \end{aligned}$$

(b) Find the closed form solution for Θ which minimizes $J(\Theta)$. This is the equivalent to the normal equations for the multivariate case.

Answer: First we take the gradient of $J(\Theta)$ with respect to Θ.

$$\begin{aligned}
\nabla_\Theta J(\Theta) &= \nabla_\Theta \left[\frac{1}{2}\mathrm{tr}\left((X\Theta - Y)^T(X\Theta - Y)\right)\right] \\
&= \nabla_\Theta \left[\frac{1}{2}\mathrm{tr}\left(\Theta^T X^T X\Theta - \Theta^T X^T Y - Y^T X\Theta - Y^T T\right)\right] \\
&= \frac{1}{2}\nabla_\Theta \left[\mathrm{tr}(\Theta^T X^T X\Theta) - \mathrm{tr}(\Theta^T X^T Y) - \mathrm{tr}(Y^T X\Theta) + \mathrm{tr}(Y^T Y)\right] \\
&= \frac{1}{2}\nabla_\Theta \left[\mathrm{tr}(\Theta^T X^T X\Theta) - 2\mathrm{tr}(Y^T X\Theta) + \mathrm{tr}(Y^T Y)\right] \\
&= \frac{1}{2}\left[X^T X\Theta + X^T X\Theta - 2X^T Y\right] \\
&= X^T X\Theta - X^T Y
\end{aligned}$$

Setting this expression to zero we obtain

$$\Theta = (X^T X)^{-1} X^T Y.$$

This looks very similar to the closed form solution in the univariate case, except now Y is a $m \times p$ matrix, so then Θ is also a matrix, of size $n \times p$.

(c) Suppose instead of considering the multivariate vectors $y^{(i)}$ all at once, we instead compute each variable $y_j^{(i)}$ separately for each $j = 1, \ldots, p$. In this case, we have a p individual linear models, of the form

$$y_j^{(i)} = \theta_j^T x^{(i)},\ j = 1, \ldots, p.$$

(So here, each $\theta_j \in \mathbb{R}^n$). How do the parameters from these p independent least squares problems compare to the multivariate solution?

Answer: This time, we construct a set of vectors

$$\vec{y}_j = \begin{bmatrix} y_j^{(1)} \\ y_j^{(2)} \\ \vdots \\ y_j^{(m)} \end{bmatrix},\ j = 1, \ldots, p.$$

Then our j-th linear model can be solved by the least squares solution

$$\theta_j = (X^T X)^{-1} X^T \vec{y}_j.$$

If we line up our θ_j, we see that we have the following equation:

$$\begin{aligned}
[\theta_1\ \theta_2\ \cdots\ \theta_p] &= \left[(X^T X)^{-1} X^T \vec{y}_1\ (X^T X)^{-1} X^T \vec{y}_2\ \cdots\ (X^T X)^{-1} X^T \vec{y}_p\right] \\
&= (X^T X)^{-1} X^T \left[\vec{y}_1\ \vec{y}_2\ \cdots\ \vec{y}_p\right] \\
&= (X^T X)^{-1} X^T Y \\
&= \Theta.
\end{aligned}$$

Thus, our p individual least squares problems give the exact same solution as the multivariate least squares.

4. **Naive Bayes**

In this problem, we look at maximum likelihood parameter estimation using the naive Bayes assumption. Here, the input features $x_j,\ j = 1, \ldots, n$ to our model are discrete, binary-valued variables, so $x_j \in \{0, 1\}$. We call $x = [x_1\ x_2\ \cdots\ x_n]^T$ to be the input vector. For each training example, our output targets are a single binary-value $y \in \{0, 1\}$. Our model is then parameterized by $\phi_{j|y=0} = p(x_j = 1|y = 0)$, $\phi_{j|y=1} = p(x_j = 1|y = 1)$, and $\phi_y = p(y = 1)$. We model the joint distribution of (x, y) according to

$$\begin{aligned}
p(y) &= (\phi_y)^y(1-\phi_y)^{1-y} \\
p(x|y=0) &= \prod_{j=1}^{n} p(x_j|y=0) \\
&= \prod_{j=1}^{n} (\phi_{j|y=0})^{x_j}(1-\phi_{j|y=0})^{1-x_j} \\
p(x|y=1) &= \prod_{j=1}^{n} p(x_j|y=1) \\
&= \prod_{j=1}^{n} (\phi_{j|y=1})^{x_j}(1-\phi_{j|y=1})^{1-x_j}
\end{aligned}$$

(a) Find the joint likelihood function $\ell(\varphi) = \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \varphi)$ in terms of the model parameters given above. Here, φ represents the entire set of parameters $\{\phi_y,\ \phi_{j|y=0},\ \phi_{j|y=1},\ j = 1, \ldots, n\}$.

Answer:

$$\begin{aligned}
\ell(\varphi) &= \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \varphi) \\
&= \log \prod_{i=1}^{m} p(x^{(i)}|y^{(i)}; \varphi) p(y^{(i)}; \varphi) \\
&= \log \prod_{i=1}^{m} \left(\prod_{j=1}^{n} p(x_j^{(i)}|y^{(i)}; \varphi) \right) p(y^{(i)}; \varphi) \\
&= \sum_{i=1}^{m} \left(\log p(y^{(i)}; \varphi) + \sum_{j=1}^{n} \log p(x_j^{(i)}|y^{(i)}; \varphi) \right) \\
&= \sum_{i=1}^{m} \Bigg[y^{(i)} \log \phi_y + (1 - y^{(i)}) \log(1 - \phi_y) \\
&\qquad + \sum_{j=1}^{n} \left(x_j^{(i)} \log \phi_{j|y^{(i)}} + (1 - x_j^{(i)}) \log(1 - \phi_{j|y^{(i)}}) \right) \Bigg]
\end{aligned}$$

(b) Show that the parameters which maximize the likelihood function are the same as

those given in the lecture notes; i.e., that

$$\begin{aligned}
\phi_{j|y=0} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \\
\phi_{j|y=1} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\
\phi_y &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m}.
\end{aligned}$$

Answer: The only terms in $\ell(\varphi)$ which have non-zero gradient with respect to $\phi_{j|y=0}$ are those which include $\phi_{j|y^{(i)}}$. Therefore,

$$\begin{aligned}
\nabla_{\phi_{j|y=0}} \ell(\varphi) &= \nabla_{\phi_{j|y=0}} \sum_{i=1}^m \left(x_j^{(i)} \log \phi_{j|y^{(i)}} + (1 - x_j^{(i)}) \log(1 - \phi_{j|y^{(i)}}) \right) \\
&= \nabla_{\phi_{j|y=0}} \sum_{i=1}^m \Big(x_j^{(i)} \log(\phi_{j|y=0}) 1\{y^{(i)} = 0\} \\
&\qquad\qquad + (1 - x_j^{(i)}) \log(1 - \phi_{j|y=0}) 1\{y^{(i)} = 0\} \Big) \\
&= \sum_{i=1}^m \left(x_j^{(i)} \frac{1}{\phi_{j|y=0}} 1\{y^{(i)} = 0\} - (1 - x_j^{(i)}) \frac{1}{1 - \phi_{j|y=0}} 1\{y^{(i)} = 0\} \right).
\end{aligned}$$

Setting $\nabla_{\phi_{j|y=0}} \ell(\varphi) = 0$ gives

$$\begin{aligned}
0 &= \sum_{i=1}^m \left(x_j^{(i)} \frac{1}{\phi_{j|y=0}} 1\{y^{(i)} = 0\} - (1 - x_j^{(i)}) \frac{1}{1 - \phi_{j|y=0}} 1\{y^{(i)} = 0\} \right) \\
&= \sum_{i=1}^m \left(x_j^{(i)} (1 - \phi_{j|y=0}) 1\{y^{(i)} = 0\} - (1 - x_j^{(i)}) \phi_{j|y=0} 1\{y^{(i)} = 0\} \right) \\
&= \sum_{i=1}^m \left((x_j^{(i)} - \phi_{j|y=0}) 1\{y^{(i)} = 0\} \right) \\
&= \sum_{i=1}^m \left(x_j^{(i)} \cdot 1\{y^{(i)} = 0\} \right) - \phi_{j|y=0} \sum_{i=1}^m 1\{y^{(i)} = 0\} \\
&= \sum_{i=1}^m \left(1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\} \right) - \phi_{j|y=0} \sum_{i=1}^m 1\{y^{(i)} = 0\}.
\end{aligned}$$

We then arrive at our desired result

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}}$$

The solution for $\phi_{j|y=1}$ proceeds in the identical manner.

To solve for ϕ_y,

$$\begin{aligned}\nabla_{\phi_y}\ell(\varphi) &= \nabla_{\phi_y}\sum_{i=1}^{m}\Big(y^{(i)}\log\phi_y + (1-y^{(i)})\log(1-\phi_y)\Big)\\ &= \sum_{i=1}^{m}\left(y^{(i)}\frac{1}{\phi_y} - (1-y^{(i)})\frac{1}{1-\phi_y}\right)\end{aligned}$$

Then setting $\nabla_{\phi_y} = 0$ gives us

$$\begin{aligned}0 &= \sum_{i=1}^{m}\left(y^{(i)}\frac{1}{\phi_y} - (1-y^{(i)})\frac{1}{1-\phi_y}\right)\\ &= \sum_{i=1}^{m}\Big(y^{(i)}(1-\phi_y) - (1-y^{(i)})\phi_y\Big)\\ &= \sum_{i=1}^{m}y^{(i)} - \sum_{i=1}^{m}\phi_y.\end{aligned}$$

Therefore,

$$\phi_y = \frac{\sum_{i=1}^{m}1\{y^{(i)}=1\}}{m}.$$

(c) Consider making a prediction on some new data point x using the most likely class estimate generated by the naive Bayes algorithm. Show that the hypothesis returned by naive Bayes is a linear classifier—i.e., if $p(y=0|x)$ and $p(y=1|x)$ are the class probabilities returned by naive Bayes, show that there exists some $\theta \in \mathbb{R}^{n+1}$ such that

$$p(y=1|x) \geq p(y=0|x) \text{ if and only if } \theta^T\begin{bmatrix}1\\ x\end{bmatrix} \geq 0.$$

(Assume θ_0 is an intercept term.)

Answer:

$$\begin{aligned}&p(y=1|x) \geq p(y=0|x)\\ \iff &\frac{p(y=1|x)}{p(y=0|x)} \geq 1\\ \iff &\frac{\left(\prod_{j=1}^{n}p(x_j|y=1)\right)p(y=1)}{\left(\prod_{j=1}^{n}p(x_j|y=0)\right)p(y=0)} \geq 1\\ \iff &\frac{\left(\prod_{j=1}^{n}(\phi_{j|y=0})^{x_j}(1-\phi_{j|y=0})^{1-x_j}\right)\phi_y}{\left(\prod_{j=1}^{n}(\phi_{j|y=1})^{x_j}(1-\phi_{j|y=1})^{1-x_j}\right)(1-\phi_y)} \geq 1\\ \iff &\sum_{j=1}^{n}\left(x_j\log\left(\frac{\phi_{j|y=1}}{\phi_{j|y=0}}\right) + (1-x_j)\log\left(\frac{1-\phi_{j|y=0}}{1-\phi_j|y=0}\right)\right) + \log\left(\frac{\phi_y}{1-\phi_y}\right) \geq 0\\ \iff &\sum_{j=1}^{n}x_j\log\left(\frac{(\phi_{j|y=1})(1-\phi_{j|y=0})}{(\phi_{j|y=0})(1-\phi_{j|y=1})}\right) + \sum_{j=1}^{n}\log\left(\frac{1-\phi_{j|y=1}}{1-\phi_{j|y=0}}\right) + \log\left(\frac{\phi_y}{1-\phi_y}\right) \geq 0\\ \iff &\theta^T\begin{bmatrix}1\\ x\end{bmatrix} \geq 0,\end{aligned}$$

where

$$\begin{aligned}\theta_0 &= \sum_{j=1}^{n} \log\left(\frac{1-\phi_{j|y=1}}{1-\phi_{j|y=0}}\right) + \log\left(\frac{\phi_y}{1-\phi_y}\right) \\ \theta_j &= \log\left(\frac{(\phi_{j|y=1})(1-\phi_{j|y=0})}{(\phi_{j|y=0})(1-\phi_{j|y=1})}\right),\ j = 1,\ldots,n.\end{aligned}$$

5. **Exponential family and the geometric distribution**

(a) Consider the geometric distribution parameterized by ϕ:

$$p(y;\phi) = (1-\phi)^{y-1}\phi,\ y = 1,2,3,\ldots.$$

Show that the geometric distribution is in the exponential family, and give $b(y)$, η, $T(y)$, and $a(\eta)$.

Answer:

$$\begin{aligned}p(y;\phi) &= (1-\phi)^{y-1}\phi \\ &= \exp\left[\log(1-\phi)^{y-1} + \log\phi\right] \\ &= \exp\left[(y-1)\log(1-\phi) + \log\phi\right] \\ &= \exp\left[y\log(1-\phi) - \log\left(\frac{1-\phi}{\phi}\right)\right]\end{aligned}$$

Then

$$\begin{aligned}b(y) &= 1 \\ \eta &= \log(1-\phi) \\ T(y) &= y \\ a(\eta) &= \log\left(\frac{1-\phi}{\phi}\right) = \log\left(\frac{e^\eta}{1-e^\eta}\right),\end{aligned}$$

where the last line follows becuase $\eta = \log(1-\phi) \Rightarrow e^\eta = 1-\phi \Rightarrow \phi = 1-e^\eta$.

(b) Consider performing regression using a GLM model with a geometric response variable. What is the canonical response function for the family? You may use the fact that the mean of a geometric distribution is given by $1/\phi$.

Answer:

$$g(\eta) = E[y;\phi] = \frac{1}{\phi} = \frac{1}{1-e^\eta}.$$

(c) For a training set $\{(x^{(i)}, y^{(i)});\ i = 1,\ldots,m\}$, let the log-likelihood of an example be $\log p(y^{(i)}|x^{(i)};\theta)$. By taking the derivative of the log-likelihood with respect to θ_j, derive the stochastic gradient ascent rule for learning using a GLM model with goemetric responses y and the canonical response function.

Answer: The log-likelihood of an example $(x^{(i)}, y^{(i)})$ is defined as $\ell(\theta) = \log p(y^{(i)}|x^{(i)};\theta)$. To derive the stochastic gradient ascent rule, use the results from previous parts and the standard GLM assumption that $\eta = \theta^T x$.

$$\begin{aligned}
\ell_i(\theta) &= \log\left[\exp\left(\theta^T x^{(i)} \cdot y^{(i)} - \log\left(\frac{e^{\theta^T x^{(i)}}}{1 - e^{\theta^T x^{(i)}}}\right)\right)\right] \\
&= \log\left[\exp\left(\theta^T x^{(i)} \cdot y^{(i)} - \log\left(\frac{1}{e^{-\theta^T x^{(i)}} - 1}\right)\right)\right] \\
&= \theta^T x^{(i)} \cdot y^{(i)} + \log\left(e^{-\theta^T x^{(i)}} - 1\right) \\
\frac{\partial}{\partial \theta_j}\ell_i(\theta) &= x_j^{(i)} y^{(i)} + \frac{e^{-\theta^T x^{(i)}}}{e^{-\theta^T x^{(i)}} - 1}(-x_j^{(i)}) \\
&= x_j^{(i)} y^{(i)} - \frac{1}{1 - e^{-\theta^T x^{(i)}}} x_j^{(i)} \\
&= \left(y^{(i)} - \frac{1}{1 - e^{\theta^T x^{(i)}}}\right) x_j^{(i)}.
\end{aligned}$$

Thus the stochastic gradient ascent update rule should be

$$\theta_j := \theta_j + \alpha \frac{\partial \ell_i(\theta)}{\partial \theta_j},$$

which is

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - \frac{1}{1 - e^{\theta^T x^{(i)}}}\right) x_j^{(i)}.$$