

# CS 229 Homework

Tyler Neylon

345.2016

These are solutions to the most recent problems posted for Stanford's CS 229 course, as of June 2016. I'm not sure if this course re-uses old problems, but please don't copy the answers if so. This document is also available as a [pdf](#).

## 1 Problem set 1

### 1.1 Logistic regression

#### 1.1.1 Part (a)

The problem is to compute the Hessian matrix  $H$  for the function

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \log(g(y^{(i)} x^{(i)})),$$

where  $g(z)$  is the logistic function, and to show that  $H$  is positive semi-definite; specifically, that  $z^T H z \geq 0$  for any vector  $z$ .

We'll use the fact that  $g'(z) = g(z)(1 - g(z))$ . We'll also note that since all relevant operations are linear, it will suffice to ignore the summation over  $i$  in the definition of  $J$ . I'll use the notation  $\partial_j$  for  $\frac{\partial}{\partial \theta_j}$ , and introduce  $t$  for  $y\theta^T x$ . Then

$$-\partial_j(mJ) = \frac{g(t)(1 - g(t))}{g(t)} x_j y = x_j y (1 - g(t)).$$

Next

$$-\partial_k \partial_j(mJ) = x_j y (-g(t)(1 - g(t))) x_k y,$$

so that

$$\partial_{jk}(mJ) = x_j x_k y^2 \alpha,$$

where  $\alpha = g(t)(1 - g(t)) > 0$ .

Thus we can use repeated-index summation notation to arrive at

$$z^T H z = z_i h_{ij} z_j = (\alpha y^2)(z_i x_i x_j z_j) = (\alpha y^2)(x^T z)^2 \geq 0.$$

This completes this part of the problem.

### 1.1.2 Part (b)

Here is a matlab script to solve this part of the problem:

```
% problem1_1b.m
%
% Run Newton's method on a given cost function for a logistic
% regression setup.
%

printf('Running problem1_1b.m\n');

% Be able to compute J.
function val = J(Z, theta)
    [m, _] = size(Z);
    g      = 1 ./ (1 + exp(Z * theta));
    val    = -sum(log(g)) / m;
end

% Setup.
X      = load('logistic_x.txt');
[m, n] = size(X);
X      = [ones(m, 1) X];
Y      = load('logistic_y.txt');
Z      = diag(Y) * X;

% Initialize the parameters to learn.
old_theta = ones(n + 1, 1);
theta     = zeros(n + 1, 1);
i         = 1; % i = iteration number.

% Perform Newton's method.
while norm(old_theta - theta) > 1e-5
    printf('J = %g\n', J(Z, theta));
```

```

printf('theta:\n');
disp(theta);
printf('Running iteration %d\n', i);

g      = 1 ./ (1 + exp(Z * theta));
f      = (1 - g);
alpha  = f .* g;
A      = diag(alpha);
H      = Z' * A * Z / m;
nabla  = Z' * f / m;
old_theta = theta;
theta  = theta - inv(H) * nabla;

i++;
end

% Show and save output.
printf('Final theta:\n');
disp(theta);
save('theta.mat', 'theta');

```

Because I have copious free time, I also wrote a Python version. Also because I'm learning numpy and would prefer to consistently use a language that I know can produce decent-looking graphs. Here is the Python script:

```

#!/usr/bin/env python

import numpy as np
from numpy import linalg as la

# Define the J function.
def J(Z, theta):
    m, _ = Z.shape
    g     = 1 / (1 + np.exp(Z.dot(theta)))
    return -sum(np.log(g)) / m

# Load data.
X     = np.loadtxt('logistic_x.txt')
m, n  = X.shape
X     = np.insert(X, 0, 1, axis=1) # Prefix an all-1 column.
Y     = np.loadtxt('logistic_y.txt')
Z     = np.diag(Y).dot(X);

# Initialize the learning parameters.
old_theta = np.ones((n + 1,))
theta     = np.zeros((n + 1,))

```

```

i          = 1

# Perform Newton's method.
while np.linalg.norm(old_theta - theta) > 1e-5:

    # Print progress.
    print('J = {}'.format(J(Z, theta)))
    print('theta = {}'.format(theta))
    print('Running iteration {}'.format(i))

    # Update theta.
    g      = 1 / (1 + np.exp(Z.dot(theta)))
    f      = 1 - g
    alpha   = (f * g).flatten()
    H      = (Z.T * alpha).dot(Z) / m
    nabla   = Z.T.dot(f) / m
    old_theta = theta
    theta    = theta - la.inv(H).dot(nabla)

    # Update i = the iteration counter.
    i += 1

# Print and save the final value.
print('Final theta = {}'.format(theta))
np.savetxt('theta.txt', theta)

```

The final value of  $\theta$  that I arrived at is

$$\theta = (2.62051, -0.76037, -1.17195).$$

The first value  $\theta_0$  represents the constant term, so that the final model is given by

$$y = g(2.62 - 0.76x_1 - 1.17x_2).$$

## 1.2 Poisson regression and the exponential family

### 1.2.1 Part (a)

Write the Poisson distribution as an exponential family:

$$p(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta)),$$

where

$$p(y; \lambda) = \frac{e^{-\lambda} \lambda^y}{y!}.$$

This can be done via

$$\begin{aligned} \eta &= \log(\lambda), \\ a(\eta) &= e^\eta = \lambda, \\ b(y) &= 1/y!, \text{ and} \\ T(y) &= y. \end{aligned}$$