EL-GY 9123: Introduction to Machine Learning
Midterm Solutions, Fall 2017

1. (a) A linear model would be $f(u) = \beta_0 + \beta_1 u$. Since we want $f(0) = 0$, we can remove the constant term $\beta_0 = 0$. This leaves the model $f(u) = \beta_1 u$ with one parameter β_1.

   (b) Again, since we want $f(u, \lambda) = 0$ when $u = 0$ for all λ, we should take $A = 0$. Also, since $f(u, \lambda)$ is continuous, we need $B(\lambda)$ to be continuous. Since $B(\lambda)$ is constant for $\lambda \geq 800$, we must have $B(\lambda) = B(800)$ for $\lambda \geq 800$. Therefore,

$$f(u, \lambda) = B(\lambda)u = \begin{cases} u(B_1 + B_2\lambda) & \text{if } \lambda \leq 800, \\ u(B_1 + B_2(800)) & \text{if } \lambda > 800 \end{cases}$$

   We can write this as a linear model,

$$f(u, \lambda) = B_1\phi_1(u, \lambda) + B_2\phi_2(u, \lambda), \tag{1}$$

   with the basis functions,

$$\phi_1(u, \lambda) = u, \quad \phi_2(u, \lambda) = \begin{cases} u\lambda & \text{if } \lambda \leq 800, \\ u(800) & \text{if } \lambda > 800. \end{cases}$$

   The parameters are $\beta = (B_1, B_2)$. In (1), you can also write the second basis function as

$$\phi_2(u, \lambda) = u \min\{\lambda, 800\}.$$

   There are other ways to describe the linear model, but all descriptions should have two basis functions with two parameters.

   (c) Using the parametrization in (1), the matrix $\mathbf{A}$ should be

$$\mathbf{A} = \begin{bmatrix} u_1 & u_1 \min\{\lambda_1, 800\} \\ u_2 & u_2 \min\{\lambda_2, 800\} \\ u_3 & u_3 \min\{\lambda_3, 800\} \\ \vdots & \vdots \end{bmatrix} = \begin{bmatrix} 40 & 40(600) \\ 80 & 80(700) \\ 140 & 140(800) \\ \vdots & \vdots \end{bmatrix},$$

   (d) One simple code is:

```
def evaluate(u,lam,ytrue,beta):
    yhat = beta[0]*u + beta[1]*u*np.minimum(lam,800)
    mse = np.mean((yhat-ytrue)**2)
    return mse
```

Note that in `numpy`, you want to use `np.minimum` which takes the minimum between two arrays. Not, `np.min` which takes the minimum within an array. But, if you didn't know this, no marks will be deducted.

2. (a) Since the true function is

$$f_0(x) = -De^{-Gx} + D,$$

it falls within Model 2 with $(a, b, c) = (-D, G, D)$. It does not fall in Model 1 since it requires $c = D \neq 0$.

(b) Since $y_i = a_0 e^{-b_0 x_i}$ and $x_i = i\Delta$,

$$\frac{y_{i+1}}{y_i} - \frac{a_0 e^{-b_0(i+1)\Delta}}{a_0 e^{-b_0 i\Delta}} = e^{-b_0\Delta}.$$

Hence,

$$\ln\left[\frac{1}{N-1}\sum_{i=1}^{N-1}\frac{y_{i+1}}{y_i}\right] = \ln\left[\frac{N-1}{N-1}e^{-b_0\Delta}\right] = \ln\left[e^{-b_0\Delta}\right] = -b_0\Delta.$$

Therefore, if we take $C = -1/\Delta$, we obtain

$$\hat{b} = -\frac{1}{\Delta}\ln\left[\frac{1}{N-1}\sum_{i=1}^{N-1}\frac{y_{i+1}}{y_i}\right] = -\frac{-b_0\Delta}{\Delta} = b_0,$$

so the estimate is unbiased.

(c) In this case, the bias is

$$\hat{c} - c_0 = y_{i_0} - c_0 = a_0 e^{-i_0 b_0\Delta} + c_0 - c_0 == a_0 e^{-i_0 b_0\Delta}.$$

Since $b_0 > 0$, the bias is decreasing with i_0. So, we should take i_0 to be the largest possible value $i_0 = n$.

(d) One possible solution is as follows:

```
# Split into training and test
ntr = 50
xtr = x[:ntr]
ytr = y[:ntr]
xts = x[ntr:]
yts = y[ntr:]

# Loop over different model orders
models = [1,2]
rss = []
for mod in models:
    bethat = fit(xtr,ytr,mod)        # Fit the model on the training data
    yhat = predict(xts,betahat,mod)  # Predict on test data
    rssi = np.sum((yts - yhat)**2)   # Measure the test RSS
    rss.append(rssi)
```

i	u_{i1}	u_{i2}	y_i	$x_{i1} = u_{i1}$	$x_{i2} = u_{i1}u_{i2}$
1	0.5	6	0	0.5	3
2	2	2	1	2	4
3	3	0.5	0	3	1.5
4	4	0.75	1	4	3

Table 1: Nonlinear transformed points for the first four samples.

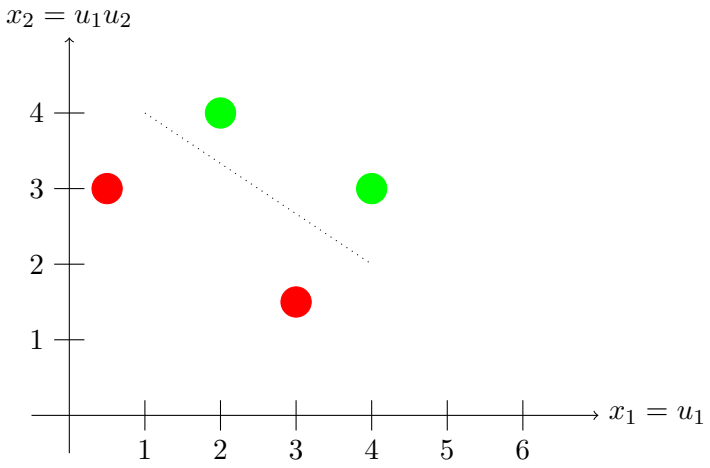

Figure 1: Scatter plot of the data points where the green circles are $y_i = 1$ and red circles are $y_i = 0$. The dotted line is the boundary of a potential linear classifier.

```
# Find the minimum test RSS
if rss[0] <= rss[1]:
    mod_opt = 1
else:
    mod_opt = 2
```

3. (a) Table 1 computes the transformed values (x_{i1}, x_{i2}) for the first four points. These are then plotted in the scatter plot in Fig. 1.

   (b) You can see that they are easily linearly separable. For example, we can use a line that goes from $(1, 4)$ to $(4, 2)$. The classifier would set $\hat{y} = 1$ when

$$x_2 > \frac{2-4}{4-1}(x_1 - 1) + 4 = -\frac{2}{3}x_1 + \frac{10}{3}.$$

   So, we can take the classifier

$$\hat{y} = \begin{cases} 1 & \text{if } 3x_2 + 2x_1 - 10 \geq 0, \\ 0 & \text{if } 3x_2 + 2x_1 - 10 < 0 \end{cases}$$

   Any other line that works, will also get full credit.

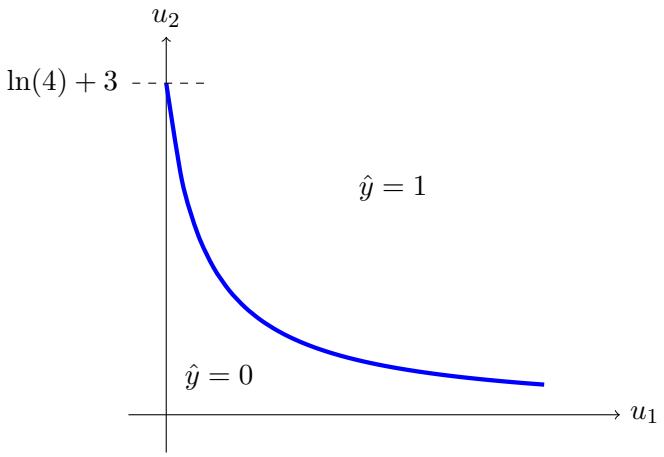

Figure 2: Classifier boundary and decision regions. The classifier selects $\hat{y}=1$ above the curve.

(c) We need,

$$P(y=1|\mathbf{u})>0.8 \iff \frac{1}{1+e^{-z}}>0.8 \iff z>-\ln\left[\frac{5}{4}-1\right]=\ln(4).$$

Hence,

$$\begin{aligned} P(y=1|\mathbf{u})>0.8 &\iff \beta_0+\beta_1 x_1+\beta_2 x_2 \geq \ln(4) \\ &\iff -3+u_1+2u_1u_2 \geq \ln(4) \iff u_2 \geq \frac{\ln(4)+3}{1+2u_2} \end{aligned}$$

The classification region is shown in Fig. 2.

(d) First let m be the maximum number of missed detections, which we can compute by

$$m=|\{i|\hat{y}_i=0, y_i=1\}|=P_{\text{MD,max}}\,|\{i|y_i=1\}|$$

You can compute this in python with

```
pmdmax = 0.1
m = np.floor(np.sum(y==1)*pmdmax)
```

If you forget to take the `floor` function to convert to an integer, you will not lose any marks.

Now let $v=\beta_1 x_1+\beta_2 x_2$ so that $z=v+\beta_0$. A missed detection occurs when,

$$z_i=v_i+\beta_0<0 \text{ and } y_i=1.$$

So, we should select $\beta_0<-v_i$ for at most m values of i. This is easiest done by sorting the values v_i and selecting β_0 to be the m-th largest value:

```
v = beta[1]*x[:,0] + beta[2]*x[:,1]
vsort = np.sort(-v)
beta[0] = vsort[m]
```

4. (a) First let

$$\mathbf{A} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ \vdots & \cdots & \cdots & \vdots \\ 1 & x_n & \cdots & x_n^d \end{bmatrix},$$

so that if we let $\mathbf{z} = \mathbf{A}\mathbf{w}$ then $\hat{\mathbf{y}} = \mathbf{z}$. Then, if we let

$$g(\mathbf{z}) = \sum_{i=1}^{n} g_i(z_i), \quad g_i(z_i) = [\ln(z_i) - \ln(y_i)]^2,$$

we have $J_{\log}(\mathbf{w}) = g(\mathbf{A}\mathbf{w})$.

(b) The gradient of $g(\mathbf{z})$ is

$$\nabla_{\mathbf{z}} g(\mathbf{z}) = \left[g_1'(z_1), \cdots, g_n'(z_n)\right]^{\mathsf{T}}, \quad g_i'(z_i) = 2(\ln(z_i) - \ln(y_i))\frac{1}{z_i}.$$

From the forward-backward rule,

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbf{A}^{\mathsf{T}} \nabla_{\mathbf{z}} g(\mathbf{z}), \quad \mathbf{z} = \mathbf{A}\mathbf{w}.$$

(c) We calculate the loss for each value of $\mathbf{w}$ in Table 2. For the regular LS loss, we see that

$$J_{\text{ls}}(\mathbf{w}) = \begin{cases} 100^2 + 1, & \mathbf{w} = \mathbf{w}^A \\ 49, & \mathbf{w} = \mathbf{w}^B. \end{cases}$$

So, for the regular LS loss, $\mathbf{w}^B$ gives a lower loss. For the log LS loss, we can write the loss as

$$J_{\log}(\mathbf{w}) = \frac{1}{\log_2^2(e)} \sum_{i=1}^{n} \left(\log_2(\hat{y}_i/y_i)\right)^2,$$

where we have used the fact that $\ln(x) = \log_2(x)/\log_2(e)$. So, we can pick the $\mathbf{w}$ that minimizes

$$\sum_{i=1}^{n} \left(\log_2(\hat{y}_i/y_i)\right)^2,$$

Using Table 2, we see that

$$\sum_{i=1}^{n} \left(\log_2(\hat{y}_i/y_i)\right)^2 = \begin{cases} 5, & \mathbf{w} = \mathbf{w}^A \\ 9, & \mathbf{w} = \mathbf{w}^B. \end{cases}$$

So, the log LS loss is minimized with $\mathbf{w} = \mathbf{w}^A$.

(d) One possible solution is:

```
def momentum_grad(feval,alpha,beta,winit,nit):
    """
    feval:  Function that returns f,fgrad representing
       loss and its gradient
    alpha, beta:  Momentum parameters
```

y_i	$\mathbf{w} = \mathbf{w}^A$			$\mathbf{w} = \mathbf{w}^B$		
	$\hat{y}_i$	$(y_i - \hat{y}_i)^2$	$(\log_2(\hat{y}_i/y_i))^2$	$\hat{y}_i$	$(y_i - \hat{y}_i)^2$	$(\log_2(\hat{y}_i/y_i))^2$
1	2	$(1)^2$	$(1)^2$	8	$(7)^2$	$(3)^2$
100	200	$(100)^2$	$(2)^2$	100	0	0
Total loss		$(100)^2 + 1$	5		49	9

Table 2: Loss calculations

```
    winit:  Initial condition
    nit:  number of iterations
    """

    # Initialize
    shape = winit.shape
    g = np.zeros(shape)
    w = winit

    # Main loop
    for it in range(nit):
        f, fgrad = feval(w)
        g = beta*g + fgrad
        w = w - alpha*g

    return w, f
```