Question 1. Logistic Regression

In this assignment you implement logistic regression and train and test it on real data.

Recall that the error function in logistic regression compares two probabilities using cross-entropy loss function. Formally for a single data example $(\mathbf{x}, y)$ and a prediction $\hat{y}$, the value of the loss would be

$$L(\hat{y}, y) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y}) \quad \text{where} \qquad \hat{y} = \sigma(\hat{z}) = \frac{1}{1 + \exp(-\hat{z})}$$

is the prediction for a single example and $\hat{z} = \mathbf{x}^\top \mathbf{w}$ the pre-prediction for a single example. The total training loss is the sum of losses of all example $(X_{i:}, y_i)$:

$$l = \sum_{i=1}^{t} L(\hat{y}_i, y_i), \quad \hat{y}_i = \sigma(X_{i:}\mathbf{w})$$

(i) (1.5%) Show that gradient of total loss function w.r.t. the weight vector $\mathbf{w}$ is

$$\nabla_{\mathbf{w}} l = X^\top (\hat{\mathbf{y}} - \mathbf{y})$$

Hint: First prove that

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)).$$

(ii) (0.5%) Implement a Matlab/Octave function to compute the sigmoid for a scalar input z.

```
function f = sigmoid(z)
 % Fill the function body to compute the sigmoid of a scalar z
end
```

Then plot your function and include the plot in your answers. In order to plot your function in Matlab/Octave use `fplot` as below.

```
fplot(@sigmoid, [-10000, 10000]);
```

(iii) (1.5%) Using the derivation in part (i), implement in Matlab/Octave

```
function [f, G] = cross_entropy(X, w, y)
  % Implement total loss value f (a scalar) and gradient G of
  % total loss function w.r.t. w (G and w are n by 1 vectors).
end
```

This function receives a $t \times n$ matrix of data, an $n \times 1$ vector of weights, and a $t \times 1$ vector of targets ($y_i \in \{0, 1\}$). It returns a scalar f, the total value of cross-entropy loss, and an $n \times 1$ vector, G, the gradient w.r.t. to $\mathbf{w}$.

Hint 1: You can call your sigmoid function inside your implementation.

Hint 2: $\lim_{x \to 0} x \log(x) = 0$.

(iv) (%0.5) Implement in Matlab/Octave

```
function [f, G] = l2_reg(w)
 % Fill in the funcion to compute the function value and gradient.
end
```

This function computes L_2 regularization on $\mathbf{w}$ and returns the value of the regularizer $f = \|\mathbf{w}\|_2^2$ and the gradient of it w.r.t. to $\mathbf{w}$.

(v) Use the following code snippet to combine your loss and regularizer to form the training objective function.

```
function [f, G] = obj(X, w, y,  b)
  [f1, G1] = cross_entropy(X, w, y); % apply loss function
  [f2, G2] =l2_reg(w); % apply regularizer
 % 'b' is the regularization param
  f = f1 + b * f2; % add value of loss and regularizer
  G = G1 + b * G2;  % add gradient of loss and regularizer wrt w
end
```

(vi) Check your implantation of gradient with a numerical package called gradest[1]. This numerical package can be downloaded and is also in the supplemental materials of this assignment. You may use the following code snippet.

```
function  [pass, maxErr] = grad_check(fun, sizeVec, rep, tol)
  if nargin < 4
    tol = 1E-6; %set the default value
   end

  maxErr = 0;
  for i = 1 : rep
    w0 = randn(sizeVec(1), sizeVec(2)); % the gradient would be computed in this random point
    [g1] = gradest(fun, w0); %receive the numerical estimation of gradient (slow)
    [f, g2] = fun(w0); %receive the gradient
    maxErr = max(maxErr, max(abs(g1(:) - g2(:)))); %compute the difference between computed & estimated gradients
  end
        %Check if the error is smaller than tol:
  if maxErr < tol
    pass = 1;
  else
    pass = 0;
  end
end
```

[1] https://www.mathworks.com/matlabcentral/fileexchange/13490-adaptive-robust-numerical-differentiation/
content/DERIVESTsuite/gradest.m

(vii) Use the following code snippet to train a logistic regression classifier. This code simply calls an iterative local decent solver of Matlab/Octave for unconstrained optimization (fminunc).

```
function [wOpt, Objval] = train_logistic_reg(X, y, b)
  obj_func = @(w) obj(X, w, y, b);
  [t, n] = size(X);
  w0 = zeros(n, 1);
  options = optimoptions('fminunc','Algorithm','quasi-newton','GradObj', 'on'); % in Matlab
  % Note: Comment the above line in Octave and uncomment the following line
  % options = struct('Algorithm','quasi-newton','GradObj', 'on')

  % Passes the handle for objective function and an initial point to a solver to perform minimization
  [wOpt,Objval,exitflag,output] = fminunc(obj_func, w0, options)
end
```

It is a good idea to train on small data of your own for debugging.

(viii) (1%) Write a function to compute probabilities and the classification results

```
function [yhat, phat] = classify(X, w)
 % X is a matrix of training/test data and w is the learned weights
 % Returns prediction yhat in {0, 1} and the probability of being a malignant case phat for each example.
 % yhat and phat must be vectors of the same length
end
```

Write a function to compute the accuracy of prediction:

```
function accuracy = compute_accuracy(y, yhat)
  % Accuracy is the fraction of correctly classified examples.
  % y and yhat must be vectors of the same length
end
```

(ix) (2%) Train with $\texttt{b} = 0.01$ on the real data provided in the supplemental material of this assignment. You should use `load('a2q1.mat')` to load `X_train, y_train, X_test, y_test` matrices. The classification task here is to diagnose breast cancer based on features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass (examples with label 0 and 1 are benign and malignant cases, respectively). The source of data is UCI machine learning repository. You can find more information about the data in `https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names`.

Report the training and test accuracies.

Question 2. Gradient Decent.

In this question, you will write a Matlab/Octave function to find a local minimum of a function by the gradient decent algorithm and study its shortcomings. For this exercise, it will be useful to know about passing a function to another function using a function handle in Matlab/Octave.

(a) (0.5%) Let $f = (3x_1 - 9)^2 + (x_2 - 4)^2$. Find the global minizer$\mathbf{x}^*$ and its corresponding global minimum value. Find the gradient of f at $\mathbf{x}$ and write a function

```
function [f, G] = simple_quadratic(x)
  % Compute the value of f and G based on input x.
end
```

where $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ is the input, and $f = f(\mathbf{x})$ and $G = \nabla f(\mathbf{x}) = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \end{bmatrix}$ are the outputs.

(b) (1%) Recall that in the gradient decent algorithm, we start from an initial point $\mathbf{x}_0$ and in each iteration, we update the current point to $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ where α is the step size. Write a function

```
function [xvals, fvals] = gradient_decent(func, x0, options)
  % xvals -> row i + 1 is the value of x at iteration i
  % fvals -> row i + 1 is the value of func(x) at iteration i
end
```

where

- `func` is a handle of some function f that returns the value and gradient at point $\mathbf{x}$. In other words, you can call `func` in `gradient_decent` as follows:

```
[f, G] = func(x)
```

  where $f = f(\mathbf{x})$ and $G = \nabla f(\mathbf{x})$. The function `simple_quadratic` is an example of such a function.

- `x0` is the starting point of gradient decent.
- `options` is a structure (`struct`) with two fields: 1) `NumIterations`: the number of iterations that we run the algorithm; 2) `StepSize`: the step size α. You can access these fields as `options.NumIterations` and `options.StepSize`, respectively.
- `xvals` is a matrix that contains the $\mathbf{x}$-values of all iterations. More formally, the $(i+1)$th row of `xvals` must contain the value of $\mathbf{x}$ after iteration i: $\mathbf{x}_i^\top$. For example, the first row must contain $\mathbf{x}_0^\top$.
- `fvals` is a column vector that contains the f-values of all iterations. More formally, the $(i+1)$th row of `fvals` must contain the value of $f(\mathbf{x}_i)$ where $\mathbf{x}_i$ is the value of $\mathbf{x}$ after iteration i. For example, the first row must contain $f(\mathbf{x}_0^\top)$.

(c) (0.5%) Use the following code snippet

```
function experiment1(num_iterations, step_size)
  x0 = [0; 0];
  options = struct('NumIterations', num_iterations, 'StepSize', step_size);
  [xvals, fvals] = gradient_decent(@simple_quadratic, x0, options);

  clf
  plot(fvals, 'b-')
  title('Change in the value of function over iterations.')
  xlabel('Iteration Num')
  ylabel('simple\_quadratic(x)')
  figure

  clf
  plot(xvals(:,1), xvals(:,2), ...
       '--gs', 'MarkerSize', 5, 'MarkerFaceColor','r')
  title('Trajectory of x over iterations. The first 10 points are numbered.')
  xlabel('x_1')
  ylabel('x_2')
  point_labels = num2str((0:10)','%d');
  offset = (max(xvals(1:11,1)) - min(xvals(1:11,1))) / 40;
  text(xvals(1:11,1) + offset, xvals(1:11,2), point_labels)
end
```

which calls `simple_quadratic` and `gradient_decent`. This function runs the gradient decent algorithm to find the minimum of `simple_quadratic` and plots two figures. The first figure shows the change in value of `simple_quadratic` over iterations. The second figure shows the tragectory of $\mathbf{x}$ values (the first 10 points are numbered). Run

```
experiment1(50, 0.1)
```

and report your observations (including the two figures).

Note: Try a smaller `step_size` (e.g. 0.05) to get some intuition.

(d) (1%) In this part, we will see the importance of step size. Run

```
experiment1(15, 0.12)
```

and report your observations (including the two figures). Is x_1 converging to the optimal value? Is x_2 converging to the optimal value? Justify the behavior of the algorithm.